



## **Deep Energy Networks for Data Completion**

**Olavo Gonçalves Bacelar**

Thesis to obtain the Master of Science Degree in

### **Mathematics and Applications**

Supervisors: Prof. Maria da Conceição Esperança Amado  
Prof. Bruno Emanuel da Graça Martins

### **Examination Committee**

Chairperson: Prof. António Manuel Pacheco Pires  
Supervisor: Prof. Bruno Emanuel da Graça Martins  
Members of the Committee: Prof. André Filipe Torres Martins  
Prof. Mário Alexandre Teles de Figueiredo

**July 2021**



# Acknowledgments

To Professors André Martins, Bruno Martins, and Conceição Amado, for the invaluable help and encouragement throughout this project, for the multiple careful revisions which resulted in a much improved final document, and for guiding me towards feasible incremental goals. I would also like to thank Vlad Niculae for a valuable suggestion that allowed our algorithm to be implemented in a much simpler fashion.

To my mother, my father, and siblings, for all the love and care, as well as the countless opportunities you have provided me all these years. To my friends, including Rui, Zé, Rodrigo, Filipe, Pedro, Momo, Kirtan, and Atri, for the unforgettable experiences we have shared, and for opening my eyes to whole new worlds through conversation.

Finally, to my dear Anita, for your unconditional support, and for filling my life with love and joy and the most beautiful smile there is. For having been such a wonderful fellow traveler in this writing journey, and for continuing to inspire me to the next adventure ahead.



# Abstract

The energy-based learning framework can inherently be leveraged to handle missing values. With Structured Prediction Energy Networks (SPENs), training is considerably simplified comparatively with other energy-based approaches, since these models can be trained end-to-end for a large class of energy functions. To generate missing data, we adapted SPENs to a framework that supports arbitrary masks. We show that the proposed framework performs significantly better than an autoencoder baseline in a font completion task, using a dataset that we prepared from online sources. In our framework, due to the functional form of the considered energy networks, we directly model dependencies between any of the components of the data, regardless of them being part of the input or the output. Moreover, unlike in previously reported experiments with SPENs, a separate initialization network was not necessary in our application to obtain good results, and only one step of gradient-based inference was needed. The developed code, which may be used for other data completion tasks, is available [online](#).

## Keywords

Deep Learning, Energy-Based Models, Structured Prediction Energy Networks, Image Completion



# Resumo

Os modelos baseados em energia podem ser usados de forma natural para lidar com dados omissores. Quando comparadas com outras abordagens baseadas em energia, as *Structured Prediction Energy Networks* (SPENs) simplificam consideravelmente o treino. Isto deve-se ao facto de elas poderem ser treinadas de maneira *end-to-end* para uma classe alargada de funções de energia. Nesta dissertação, adaptamos as SPENs de forma a gerar dados em falta, numa abordagem que também suporta máscaras arbitrárias. Mostramos que, numa tarefa de geração de letras em falta num conjunto de dados de fontes tipográficas, que preparamos a partir de recursos disponíveis online, este algoritmo tem um desempenho significativamente melhor do que um algoritmo com estrutura de codificador automático. Nesta abordagem, devido à forma funcional das redes de energia consideradas, modelamos directamente dependências entre qualquer componente dos dados, independentemente de estes fazerem parte dos valores de entrada ou de saída. Para além disso, ao contrário de experiências anteriores com SPENs, não foi necessário usar uma rede de inicialização separada, sendo apenas necessário um passo de inferência para obter bons resultados. O código desenvolvido está disponível [online](#) e pode ser aplicado a outras tarefas de preenchimento de dados.

## Palavras Chave

Deep Learning, Modelos Baseados em Energia, Structured Prediction Energy Networks, Preenchimento de Imagens



# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Thesis Objectives . . . . .	4
1.2	Contributions . . . . .	5
1.3	Structure of the Dissertation . . . . .	6
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Feed-Forward Models . . . . .	8
2.1.1	Neural Networks . . . . .	8
2.1.2	Convolutional Neural Networks . . . . .	10
2.1.3	Autoencoders . . . . .	10
2.1.4	Generative Models . . . . .	11
2.2	Energy-Based Models . . . . .	13
2.2.1	Overview . . . . .	13
2.2.2	Generative Energy-Based Models . . . . .	15
2.3	Structured Prediction Energy Networks . . . . .	16
2.3.1	Prediction Using SPENs . . . . .	17
2.3.2	End-to-End Learning for SPENs . . . . .	18
2.4	Related Work . . . . .	21
2.4.1	Feed-Forward Models . . . . .	21
2.4.2	Energy-Based Models . . . . .	22
2.5	Overview . . . . .	23
<b>3</b>	<b>Energy Networks for Data Completion</b>	<b>25</b>
3.1	Energy Network Architectures . . . . .	26
3.2	Inference and Learning . . . . .	28
3.3	Overview . . . . .	32
<b>4</b>	<b>Experiments with Font Completion</b>	<b>33</b>
4.1	The Fonts Dataset . . . . .	34
4.2	Data Completion in the Fonts Dataset . . . . .	36

4.3	Loss Function and Metrics . . . . .	38
4.4	Convolutional Blocks . . . . .	40
4.5	Baseline Models . . . . .	41
4.5.1	Average Baseline . . . . .	41
4.5.2	Convolutional Autoencoder Baseline . . . . .	42
4.6	Deep Energy Networks . . . . .	47
4.6.1	Architecture of the Energy Network . . . . .	47
4.6.2	Training . . . . .	50
4.6.3	Number of Inference Iterations . . . . .	51
4.6.4	Dropout . . . . .	52
4.6.5	Initialization Routines . . . . .	52
4.7	Discussion of the Experimental Results . . . . .	56
<b>5</b>	<b>Conclusions</b>	<b>59</b>
5.1	Conclusions . . . . .	60
5.2	Future Work . . . . .	60

# List of Figures

1.1	Predictive feed-forward model versus predictive energy-based model . . . . .	3
1.2	Data completion in the fonts dataset . . . . .	5
2.1	Computational graph of gradient-based prediction in SPENs . . . . .	17
2.2	Backpropagation through the unrolled computational graph of inference in SPENs . . . . .	19
3.1	Softplus function for different values of $\beta$ , their second derivatives, and ReLU function . . . . .	28
4.1	Examples of fonts from the training dataset. . . . .	35
4.2	Example of a font, highlighting the packing of the letters into a 3-dimensional tensor . . . . .	36
4.3	Two different input modes regarding the choice of visible letters . . . . .	38
4.4	Convolutional block . . . . .	40
4.5	Average and discretization of each letter in the training dataset . . . . .	42
4.6	Architecture of the convolutional autoencoder baseline . . . . .	43
4.7	Results of the convolutional autoencoder baseline . . . . .	46
4.8	SPEN algorithm adapted for performing data completion in the font dataset . . . . .	48
4.9	Plateau problem when training SPENs for data completion . . . . .	50
4.10	Intermediate steps and final estimates of gradient-based inference . . . . .	53
4.11	Quantitative results for the convolutional baseline and SPENs . . . . .	55
4.12	Qualitative results for the convolutional baseline and SPENs . . . . .	57



# List of Tables

4.1	Parameters of the architecture of the convolutional autoencoder baseline . . . . .	44
4.2	Parameters of the architecture of the energy function . . . . .	49
4.3	Results for different number of inference iterations . . . . .	51
4.4	Results for different dropout probabilities . . . . .	52
4.5	Results for different initialization routines . . . . .	54
4.6	Results for different methods . . . . .	56

# 1

## Introduction

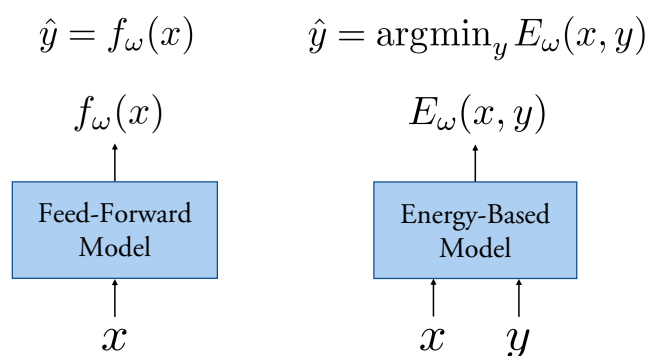
### Contents

---

1.1 Thesis Objectives . . . . .	4
1.2 Contributions . . . . .	5
1.3 Structure of the Dissertation . . . . .	6

---

Deep learning has been very successful at many different machine learning tasks, including language analysis and generation, computer vision, or control problems such as game playing. Many of the early experiments applied neural networks to problems involving outputs with a simple structure. However, we have also been increasingly seeing deep neural networks used as models for generating structured outputs, referring to outputs with multiple interdependent components. This includes research aimed at generating images [1, 2], video [3], or text [4]. In *data completion*, we wish to generate parts of the input which are missing or have been corrupted. For example, we may seek to reconstruct a part of an image that has been erased, estimate missing values in tabular data, or generate missing frames in a video. Interest in this type of problems has surged in recent years due to the influence of self-supervised learning, which leverages unlabeled data for use in pretext tasks similar to the ones involved in data completion. The objective of employing these pretext tasks is to learn representations with good generalization properties, in the sense that they may be useful for downstream tasks.



**Figure 1.1:** Left: predictive feed-forward model. Right: predictive energy-based model.

Deep neural networks can be used to model the dependencies not only between components of the input, but also between components of the output. Structured prediction handles the generation of structured, interconnected outputs [5, 6]. Belanger and McCallum [7] consider two different classes of models for structured prediction. *Feed-forward models* correspond to a class that includes many successful neural network families in use today. These models only require a forward pass for prediction, and we often train them in an end-to-end manner by directly optimizing the inference procedure by stochastic gradient descent. Conversely, with a predictive *energy-based model* (EBM), for a certain input  $x$  we try to output  $y$  corresponding to a minimum value of an energy function  $E(x, y)$ . One of the advantages of using these energy functions is that it may be possible to model the interactions between any of the components of the observed data with just one energy function and, as we will see, the treatment of missing values becomes very natural. The difference between predictive feed-forward models and predictive energy-based models is illustrated in Figure 1.1.

Within the predictive EBM framework, Belanger and McCallum introduced in 2016 the **Structured Prediction Energy Network** (SPEN) [7]. In this formulation we shape an energy function defined by a standard feed-forward neural network such that, at inference time, we may attain by gradient-based optimization in the output space a minimizer corresponding to the desired output. As in the case of other predictive EBMs, the energy function  $E_\omega(x, y)$  is a function of both inputs  $x$  and outputs  $y$ , the latter defined to be continuous, and we wish to learn its parameters  $\omega$ . Belanger and McCallum also distinguish SPENs from *factorized energy-based models*, where the energy function is factored as a sum of factors, i.e., terms that depend on a proper subset of the data components, which enables energy minimization and thus prediction to be performed. In contrast to factorized energy-based models, with SPENs we do not need to prescribe a hand-crafted factorization structure to the energy function for learning to be possible. Moreover, contrary to feed-forward models, with SPENs we can encode the interaction structure between output components with customly designed deep neural networks.

Leveraging SPENs, we can direct our attention to the identification of architectures that encode appropriate priors for the type of data in consideration. This is the case because, if we can implement the forward and back-propagation routines, the paradigm of SPENs will automatically provide us with learning and prediction procedures.

## 1.1 Thesis Objectives

In this dissertation, we seek to demonstrate that SPENs, which were originally designed with a focus on structured prediction problems that involve hard constraints between output elements, can be successfully ported to data completion tasks, particularly when dealing with the more loosely structured data of image problems. In this case the outputs are similar to the inputs, in that they are just the parts of the data that happen to be occluded. We believe EBMs are particularly well-suited for these types of problems, since the energy function can be constructed to not distinguish between visible and masked data. This way, we can in fact use a deep neural network to model dependencies between any components of the data, regardless of whether a component is hidden or not. We therefore modify the types of energy functions that were considered for SPENs by Belanger and McCallum [7] to take this into account. Note that SPENs are not well suited to model uncertainty in the data, but have advantages over other EBMs because of the relative simplicity of the inference and training procedure. Finally, we evaluate the suitability of SPENs for data completion with the help of a dataset built for that purpose from free online computer fonts (see Section 4.1). The data completion task, considering this dataset, consists in generating missing letters according to the style of the ones that are provided, and it is depicted in



**Figure 1.2:** Data completion in the fonts dataset. The letters that are visible to the algorithm are depicted in blue, and the generated letters are depicted in black.

Figure 1.2. Such an algorithm may help a font designer generate an entire font after having drawn only some of its letters.

Our approach is even more appealing given the recent trend of deep learning frameworks supporting higher order derivatives, including PyTorch [8], TensorFlow [9], and the recent JAX library [10]. These implementations of higher order derivatives can simplify considerably the approach taken by Belanger et al. [11], and also the one reported on this dissertation.

## 1.2 Contributions

In summary, the main contributions of this dissertation are as follows:

- We provide a framework for applying SPENs to a class of data completion problems that is often tackled with feed-forward models or generative EBMs. This framework supports arbitrary input masks, and enables the use of neural architectures similar to ones that are already very successfully deployed in discriminative tasks.
- We evaluated the proposed framework with a font completion task, using the aforementioned font dataset. We showed that the SPEN-based algorithm performs significantly better than an autoencoder baseline that we have also implemented.
- We showed that, by training the model only with the energy function (i.e., without adding an initialization network, as in the work of Belanger and McCallum [7]), we could obtain similar results

to those that use the autoencoder baseline for the initialization. This suggests that the energy function can encode by itself the relevant dependencies between components of the data. In fact, we tested different types of initialization routines, including alternatives that simply input random values, and concluded that the initialization did not greatly influence the final outcome.

- We showed that using just one step of gradient-based inference performed almost as well as using more steps, while significantly saving in test time computational requirements.
- We published our PyTorch implementation, including the code with which we created and preprocessed the font dataset. By the very nature of the SPEN paradigm, it is now relatively simple to adapt the code to other tasks and neural network architectures.

### **1.3 Structure of the Dissertation**

The remainder of the dissertation is organized as follows. Chapter 2 details background information on neural networks, energy-based models (including discriminative and generative ones), SPENs, and other related work. Chapter 3 describes how we adapted the SPEN framework for the task of data completion, also introducing some changes to the original procedure which have proven useful. Chapter 4 gives a detailed presentation on the setting in which we evaluated the proposed approach. Finally, Chapter 5 summarizes the main findings, and discusses possible avenues for future work.

# 2

## Background

### Contents

---

2.1	Feed-Forward Models . . . . .	8
2.2	Energy-Based Models . . . . .	13
2.3	Structured Prediction Energy Networks . . . . .	16
2.4	Related Work . . . . .	21
2.5	Overview . . . . .	23

---

This chapter describes the essential ingredients to understand the SPEN approach introduced by Belanger and McCallum [7], and some other aspects that are important for the model proposed in this dissertation. We start with an overview of neural networks in the feed-forward approach, and proceed with a general formulation of energy-based models, followed by a description of SPENs. The chapter ends with a summary of previous work involving deep neural networks for data completion.

## 2.1 Feed-Forward Models

With a feed-forward model we are interested in a direct mapping  $f_\omega(x)$  from the input  $x$  (like an image, a text segment, or a graph with node signals) to an approximation of the desired output  $y$  or, if we are dealing with discrete outputs, to soft predictions (in the form of modeled probabilities) of the corresponding output classes. Learning in such a model consists in searching for parameter values  $\omega$  such that the predicted values are close to the desired output, as measured by a particular loss function. When considering structured prediction problems, feed-forward models normally generate an output whose elements are conditionally independent given the input, thus not modeling direct interactions between those elements. However, a structure such as a Conditional Random Field (CRF) [6] can be added to the end of a feed-forward model [12] to estimate interactions between different elements of the output.

### 2.1.1 Neural Networks

A neural network is a general name for a parameterized function that, at least in one section of its structure, has multiple functions that compose serially, frequently involving linear operations with many parameters, intercalated with nonlinear activations that enable the learning of more complex functions. An example of a common nonlinear activation is the ReLU function [13] given by  $\text{ReLU}(x) = \max(0, x)$ . A deep neural network is simply a neural network with many composed layers. In general, more complex features are detected further away from the input and closer to the outputs.

We can train a neural network by randomly initializing its parameters, and change them so as to minimize a loss score between the predicted and the ground-truth values  $L(f_\omega(x), y)$ . The loss function  $L$  and the neural network should be differentiable so that we can compute the gradient  $\nabla_\omega L(f_\omega(x), y)$  using the backpropagation algorithm [14], which is an efficient way of obtaining gradients in a computational graph of functions that are composed together. The gradient is important to perform stochastic gradient descent [15, 16], a learning method in which we update the parameters  $\omega$  of the neural network in the opposite direction of an approximation to the expected value of  $\nabla_\omega L(f_\omega(x), y)$ ; that approximation is

estimated using only a minibatch of observations. For a minibatch of only one observation, the update rule of the parameters is given by:

$$\omega_{t+1} = \omega_t - \eta \nabla_{\omega_t} L(f_{\omega_t}(x), y),$$

in which  $\eta > 0$  is the learning rate, with which we can fine-tune the speed of change of those parameters. We could also train a neural network with another optimization algorithm, whether it is another variant of the first-order gradient-based optimization class or not.

Several potential pitfalls may occur when using neural networks. Care must be taken to not use a neural network with too much capacity that may overfit the training data, and to measure the gap between performance in the training data versus the validation data to look for signs that this is happening. To counteract overfitting one may for example reduce the number of parameters of the model or introduce a regularization technique, including dropout [17] and batch normalization [18]. Dropout works by randomly removing some non-output units of the forward pass of the neural network while training, and then using all available units at test time, thus promoting redundancy in the learned representations. Batch normalization learns the parameters of a shifting and rescaling transformation of the values of certain features across a minibatch of observations, with the result that the nonlinear activations receive more predictable inputs.

Consider also the issue of vanishing or exploding gradients, which happens when the learning signal is attenuated or amplified too much during the backward flow, especially if we are dealing with a deep neural network. This results in a deregulation of the learning process because the parameters do not change fast enough for learning to occur, or change too quickly so that what has been previously learned is forgotten. Much research work addressed this problem, including the use of the ReLU activation function that only saturates in one direction, and of residual networks, which allow gradient information to skip some of the layers in a neural network instead of having to pass through all of them.

One should also use a neural network architecture that encodes an inductive bias that is appropriate for the task at hand. For example, when handling image data, using a convolutional neural network (which we review in the next section) is often appropriate, because those models ensure the property of equivariance over translations of the input image.

## 2.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) [19] correspond to a class of neural network architectures that is particularly amenable to tasks that need to handle spatial information, by relying on a structure partly inspired by the visual cortex in the brain. Differently from other types of networks, these models use a convolutional layer with learnable filters having a relatively small size, resulting in multiple feature maps that may be passed to other convolutional layers or other functions to operate on. Intercalated with the convolutional layers, nonlinear functions (such as the aforementioned ReLU) are inserted, as well as pooling operations that reduce the dimension of the feature maps by using aggregating operations such as the maximum or the average of small square patches. It is common to apply a fully connected neural network to the last feature map, which will yield the desired output.

Notable CNN architectures include AlexNet [20], that famously achieved a large jump in performance at the 2012 ImageNet challenge. The VGGNet [21] is another popular model, which committed to use a particular order of the different layers, as well as a fixed specification for the kernel sizes (which were quite small) and corresponding stride values, inspiring multiple researchers to build on top of it. The ResNet [22] is also a common architecture that, by employing residual connections, improves the gradient flow and allows for higher depth to be used.

## 2.1.3 Autoencoders

An autoencoder is simply a model which passes the input through an encoder to build a compressed representation, and then passes what results to a decoder that attempts to reconstruct the original input using only the output of the encoder. It may seem pointless that the objective of an autoencoder is to try to generate its own input. However, this model allows the learning, in an unsupervised manner, of a lower-dimensional representation of the input that may be valuable for downstream tasks. Note also that the information bottleneck prevents the autoencoder from simply learning the identity mapping [19].

In more detail, the encoder part  $f$  of the autoencoder takes the input  $x$  into the lower-dimensionality space  $h$ , which we call the feature space or the latent space. We wish that the decoder  $g$  takes that compressed representation  $f(x)$  and produces an output that resembles as much as possible the input  $x$ . This is accomplished by computing a loss score between the input  $x$  and its reconstruction  $g(f(x))$ , backpropagating its gradient with respect to the parameters in the encoder and decoder parts of the autoencoder, and feeding it to an optimization algorithm to update those parameters. Because of its function, the loss function of an autoencoder is often called the reconstruction loss.

In addition to their use in unsupervised learning, autoencoders may also be used directly in a varied

range of settings, including image-denoising, anomaly detection and, using convolutional encoders and decoders, for the generation of structured outputs like segmentation masks [23,24] or images, which we review in the next subsection. For more information on other variants of autoencoders see [19].

#### 2.1.4 Generative Models

In this section we describe feed-forward generative models that are commonly used to produce images. Specifically, we cover generative adversarial networks and variational autoencoders, which are often trained in a completely unsupervised manner. It is often not clear how to evaluate the performance of these generative models and, sometimes, human evaluation is employed. Nonetheless, there exist more objective measures that take into account both the quality as well as the diversity of the generated images, sometimes using high-level features of discriminative CNNs [25].

In the generative adversarial network (GAN) [26] framework, a discriminator network assigns probabilities that a certain image is from the training dataset or was produced by a generator network, which in turn attempts to map a source of randomness  $z$  to realistic images that fool the discriminator. For that reason, each network receives a payoff that decreases when the payoff of the other increases, and vice versa. The idea is that, at the end of training, the generator network has learned to produce plausible images that the discriminator network can not perceive as being artificial, in the sense that it assigns them a probability approximately equal to 0.5.

GANs have evolved and are now capable of generating very realistic images. In spite of all the advances, there is still ongoing research regarding the often occurring problem of mode collapse. This issue manifests itself in the lack of diversity of the resulting images, as a result of a generator network that has specialized its capacity into generating only a few different samples, which it knows can fool the discriminator.

GANs use CNNs both in the discriminator and generator networks to deal with the image data. DCGANs [27] added batch normalization, which helped stabilize the learning process, as well as an entirely convolutional architecture. With those changes, the model was able to generate reasonable, albeit small, color images. The authors of the DCGAN method also demonstrated the existence of a semantic interpretation to some arithmetic operations in the latent space of  $z$ . The large-scale study that originated BigGANs [28] resulted in the identification of factors that are important in training GANs, which is often notoriously difficult. The authors also introduced other useful tricks and obtained good results in generating pictures corresponding to ImageNet classes.

With the technique of progressive growing of GANs [29], we start the training process with low resolution images and, as training stabilizes, progressively add layers to the generator and discriminator

networks to produce higher resolution images with finer details. The StyleGAN paper [30] introduced a non-linear mapping from the input latent vector to a different intermediate latent vector, which helps to disentangle factors of variation. Increasingly growing stages are also used and they are modulated by different style vectors that are based in this intermediate latent space. This is in opposition to usual GANs, which feed one source of noise, without previously transforming it, at the first layer of a feed-forward neural network. This, together with some other changes introduced in the StyleGAN paper and in the follow-up [1], made possible the generation of impressive high-resolution face pictures, which are often hard to distinguish from real ones.

The variational autoencoder (VAE) [31, 32] is another framework that is successful at generating images. It uses latent variables and generally employs complex neural networks, thus making the computation of the data likelihood intractable. However, approximate likelihood maximization is performed in a principled manner, helping alleviate the mode collapse problem with GANs. Differently from a regular autoencoder, in a VAE the encoder network learns the parameters of a distribution  $p(z|x)$ , relating a set of latent variables  $z$  with the observation  $x$ . On the other hand, the decoder attempts to model the distribution of the data given the latent variables,  $p(x|z)$ . During training, sampling of latent variables  $z$  is performed according to  $p(z|x)$ , and a final instance is produced according to  $p(x|z)$ . Similarly to regular autoencoders, a reconstruction loss between the generated sample and the observation is computed, and a regularization term over the latent variables is added to it. This last term encourages the distributions of  $p(z|x)$  to be similar to a prior distribution  $p(z)$ , with the result that it discourages the encoder from clustering each observation in its own place in the latent space (thus memorizing the data), in addition to promoting a semantically meaningful latent space. After the model is trained, we may simply sample from the prior distribution  $p(z)$  and generate a sample by leveraging the decoder network. The complete loss can be optimized with respect to the parameters of the encoder and decoder of the VAE. When a continuous stochastic sampling operation is used to generate examples from  $p(z|x)$  we need to use the reparameterization trick [31] so that the learning gradient can propagate through the operation.

The quality of the images generated by VAEs was not on par with GANs, as they were often quite blurry and not as high resolution. New models recently tried to overcome these problems. Instead of using continuous latent variables, the VQ-VAE [33] model learns a fixed size vocabulary of learnable embeddings. A PixelCNN [34, 35] autoregressive model is separately fit to learn a categorical prior over the latent variables. When we wish to generate novel images, those tokens are sampled from the learned distribution and passed to the VAE decoder. The subsequent VQ-VAE-2 model [2] added a hierarchical configuration with two latent spaces, one of which encodes general features of the image, and the other specializes in more local features like texture. This model could generate high quality

high-resolution images and, moreover, it did not have a problem with lack of diversity of the generated samples, for example covering more modes of the distribution of ImageNet pictures than BigGANs.

## 2.2 Energy-Based Models

In this dissertation, we consider energy-based models instead of feed-forward ones, and we therefore lose the property that only a forward pass is needed for training and inference. However, using only the structure of the neural network that defines an energy function, we can in principle model high-order interactions among elements of the output, as they are not conditionally independent given the input. In fact, we may even construct an energy function that models interactions among any elements of  $x$  and  $y$ . This is valuable in our setting of data completion, given that we wish to model the dependencies between any element of the data, and nothing distinguishes an input element from an output element besides one of them being visible and the other one not.

### 2.2.1 Overview

There have been at least two different but related ways in which the term *energy-based modeling* has previously been used. In this dissertation we cover both uses by defining an *Energy-Based Model* (EBM) as a model which contains an energy function that scores different configurations, expressing that a state with a lower energy has a more favorable configuration than one with a higher energy. Note that we do not impose any more constraints to this energy function and, in general, it may be a sophisticated multimodal function, such as a neural network.

Notice nonetheless that, if we read the previous definition in the widest sense, almost any machine learning algorithm may be considered to be an EBM. For example, a classification algorithm will assign, for any given input  $x$  and possible output value  $y$ , a probability  $p(y|x)$ . If we refer to the energy as  $E(x, y) = -\log p(y|x)$  we will be situated directly in this paradigm of energy-based modeling, since the prediction is  $\hat{y} = \arg \max_y p(y|x) = \arg \min_y E(x, y)$ . In this dissertation, to exclude cases like this, we will call a model an EBM only when the energy does not have normalization requirements. This definition of EBMs still covers many different types of models, some of which use probabilistic modeling and have been often used as generative models, and some others that do not and naturally fit in a discriminative setting. Nonetheless, recent work [36] showed that, by interpreting a traditional classifier as an EBM, one may leverage the same model to run discriminative and generative tasks and obtain good results in both, which normally does not happen.

In generative approaches that employ probabilistic modeling, the energy represents an unnormalized negative log density and is therefore a vehicle for modeling, in an unsupervised manner, the probability distribution of the data, which is done by way of the Gibbs distribution:

$$p_\omega(x) = \frac{\exp(-E_\omega(x))}{Z_\omega}, \quad (2.1)$$

in which  $p_\omega(x)$  is the probability of the data under the model with parameters  $\omega$ , and  $Z_\omega$  is the normalizing constant, often denoted in statistical physics as the partition function.

In contrast to these generative models, predictive EBM algorithms may also be considered, and they correspond directly to the definition of an EBM in [37]. In those models, the energy is a function of both the input  $x$  and output  $y$ , and therefore these models can naturally score joint configurations of both input and output. The inference procedure is the approximate search for the  $y$  that minimizes the energy  $E_\omega(x, y)$ , i.e., it is an approximation of:

$$\hat{y} = \arg \min_{y \in \mathcal{Y}} E_\omega(x, y). \quad (2.2)$$

Training such a model consists in configuring the energy function so that this search yields a good estimator  $\hat{y}$ . Note that, contrarily to feed-forward prediction, in energy-based prediction more than one forward pass is in general necessary to perform this search, as we shall see in Section 2.3.

A data completion task may be seen, quite generally, as a prediction task of the hidden elements  $y$  given the visible ones  $x$ . As such, the work in this dissertation was directly inspired by the energy-based prediction expressed in Equation 2.2. However, in our setting, there are no qualitative differences between the visible elements  $x$  and the hidden elements  $y$  that we wish to generate. Furthermore, we wish to complete missing values without knowing a priori which elements of the data are hidden or visible. Therefore, in our case, it is not sensible to have separate networks for encoding  $x$  and  $y$ , as was often done with the model we will use, namely the Structured Prediction Energy Network (see [7, 11] and the next section). Instead, the energy will be a function of a single object, which will also make our approach substantially similar to that of generative EBMs. For this reason, we review generative EBMs in the next subsection. Moreover, generative EBMs have previously been used in data completion tasks like image inpainting [38], which consists in the generation of plausible missing image patches and is the most relevant subclass of tasks to this dissertation. When we instead use SPENs, we do not enjoy the probabilistic modeling of uncertainty of that paradigm. On the other hand, we do not have to deal with the potential intractability or non-convergence of the normalization constant, or with sampling issues that are difficult to mitigate.

## 2.2.2 Generative Energy-Based Models

The EBMs that take into account the normalization constant of Equation 2.1 have often been applied to learn the data distribution in generative tasks. In the particular task of image generation, it is natural to consider an energy function that jointly scores all pixels in an image, and then sample from the distribution that this energy function induces to generate new images.

Despite enjoying a great expressive power, training this type of EBM often is not done in an end-to-end manner and can be notoriously challenging, since maximum likelihood training of  $p_\omega(x)$  in Equation 2.1 is intractable for sufficiently complicated energy functions. However, in recent years, there has been a flurry of research addressing different ways of training these models [39], showing that they can achieve good results in practice. MCMC sampling with Langevin dynamics [40] is often used for training these EBMs. Some work has focused in improving this process, managing to train an energy network by MCMC sampling directly from random noise [41, 42]. Aiming at lessening the computational burden, other methods use contrastive divergence training [43], often in a persistent form [36, 38], in which past samples are used to initiate the Markov chain. Some other studies avoid MCMC sampling during training altogether and use score matching instead [44, 45]. Most models that are already trained use MCMC with Langevin dynamics to generate samples. Note that training or sampling with MCMC may inherently entail mixing problems and difficulty in visiting different distribution modes.

A different recent method uses a diffusion process defined by a trainable stochastic differential equation, such that random noise can be gradually diffused to data [46]. This method also uses score matching and has achieved impressive results generating high resolution  $1024 \times 1024$  images. In different work [36, 38, 47, 48], EBMs have also shown improved calibration, out-of-distribution detection, and robustness to adversarial attacks without having explicitly been trained for them.

Some generative EBMs use implicit generation of the samples, in the sense that they are produced using only the energy function, without the aid of a different network that proposes those samples. Since in those models we do not employ separate discriminator and generator networks, as in GANs, redundant features do not need to be learned [38]. As we will see later in Section 4.6, we consider both the case where the modeling is done totally by the energy function, and the case where a separate network generates the initial values for gradient-based inference. This last case has conceptual similarities to the generation process in GANs [26], as well as to those EBMs that use a different trainable network to generate samples [48–50], or the initial iterates for the MCMC sampler [51].

Departing briefly from the world of image generation, energy-based models have also been recently used in conjunction with transformer architectures (such as BERT [52]) to define a joint probability distribution in natural language generation tasks [53]. Text generation is performed by sampling multiple

sequences from the self-attention based language model and resampling those sequences according to the probabilities attributed to them by an EBM. The whole system can be trained by noise contrastive estimation [54]. By scoring the generated sequences with a global energy function, the method from [53] enhanced coherency over long sequences.

There has also been substantial interest in exploring EBMs for reinforcement learning tasks. Energy-based models can naturally represent multimodal policy distributions to achieve distinct behaviors. This was explored in [55], in which different behaviours could be composed by first learning general-purpose policies that could later be refined for more specific tasks. A recent work [56] in the model-based reinforcement learning setting represents state dynamics without conditioning on the choice of action, resulting in multimodal transition probabilities that an energy-based model can describe. This model performed well in online learning settings, and effectively promoted exploration.

## 2.3 Structured Prediction Energy Networks

The model that we will use in this dissertation, namely the Structured Prediction Energy Network (SPEN), was introduced by Belanger and McCallum [7]. For the most thorough treatment, refer to the PhD dissertation of David Belanger [57]. A SPEN is a predictive EBM, in the sense that it employs a gradient-based search procedure in output space to approximate the predictor in Equation 2.1. The energy is used solely as a scoring function to indicate good configurations, but some modifications were made to these models by David Belanger [57] so that they could also model uncertainty, in the sense that, for a toy problem, a model could generate different outputs for the same input. SPENs are defined for continuous outputs  $y$ , and therefore a convex relaxation of discrete outputs may be needed in some tasks. The energy function  $E(x, y)$  is defined by a feed forward deep neural network. The authors intended to model, with that deep neural network, interactions between any components of the output in structured prediction problems. This is in opposition to factorized energy-based models like CRFs, that can not tractably model high-arity interactions with deep features, needing to have a simple structure in order for inference and training to be feasible.

The creators of SPENs found it valuable to further define  $E_\omega(x, y)$  to have a more specific form which, besides having modeling advantages, also helped lessen processing requirements. First, they introduced a *feature network*  $F(x)$  to encode a fixed representation of  $x$  that can be pre-trained and that can also be cached so that computing is saved in the multiple iterations of gradient-based inference. All of the terms of the energy function interact with  $x$  only through the feature network. Second, the energy

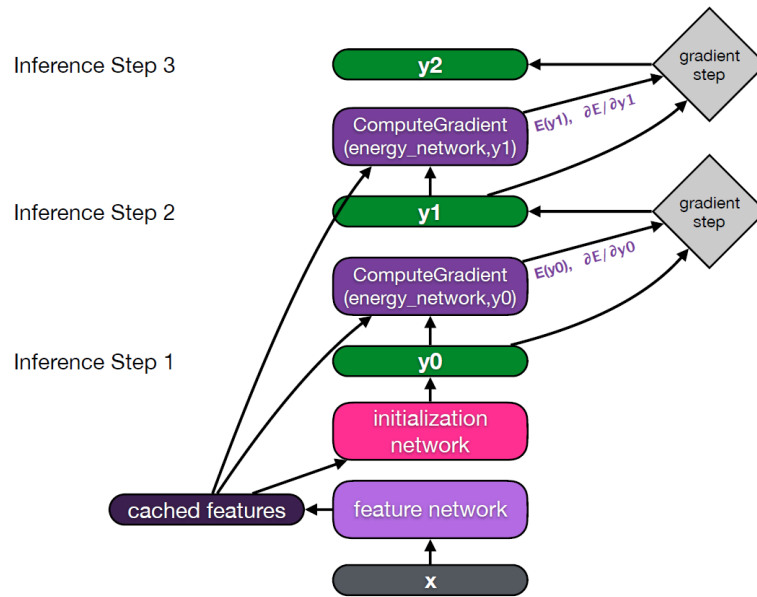
function is subdivided into a global term and a sum of local terms:

$$E(x, y) = E^g(F(x), y) + \sum_i E^l(F(x), y_i) . \quad (2.3)$$

Each local term scores independently only one component of  $y$ , and the global term can capture complicated global dependencies in those components.

Even though SPENs are a class of EBMs, we can sidestep the difficulty of training those models by doing so in an end-to-end manner, similarly to a feed-forward model. In fact, as we will see in Subsection 2.3.2, the gradient-based procedure for generating predictions at test time is itself directly trained by gradient descent.

### 2.3.1 Prediction Using SPENs



**Figure 2.1:** Computational graph of gradient-based prediction in SPENs. Reprinted from [57].

Prediction using SPENs is defined to be a gradient-based minimization in  $y$  of the energy function  $E_\omega(x, y)$ , which is differentiable in  $y$ . For each iteration  $t$ , this process is given by

$$y_t = y_{t-1} - \eta_t \nabla_y E_\omega(x, y_{t-1}), \quad (2.4)$$

where  $\eta_t$  is a possibly variable learning rate. Gradient descent with momentum can also be used. Belanger et al. [7] use a separate feed-forward network  $\text{Init}$ , based on the features  $F(x)$ , to produce an

initial estimate for  $y_0$ . Note that we generally train  $\text{Init}(F(x))$  beforehand to obtain a good initial iteration. Unrolling this entire process over  $T$  inference iterations results in:

$$y_T = \text{Init}(F(x)) - \sum_{t=1}^T \eta_t \nabla_y E_w(x, y_{t-1}).$$

The unrolling procedure is also depicted in Figure 2.1.

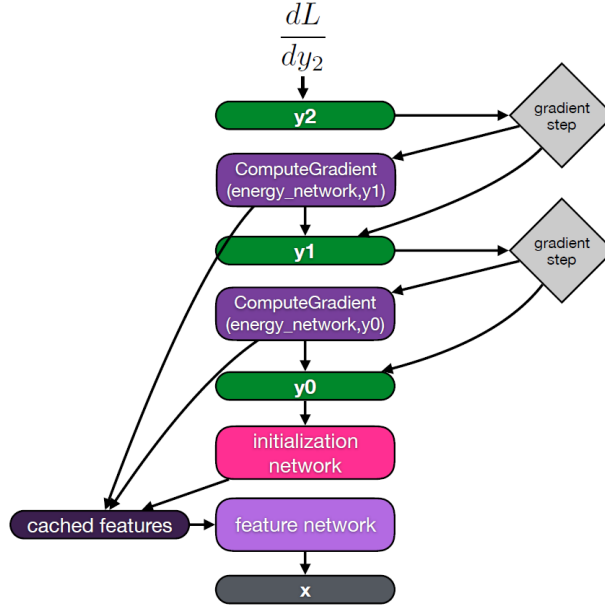
Prediction in SPENs derives from the approach in [58]. These models are also related to Input Convex Neural Networks (ICNNs) [59], in the sense that they also use an energy minimization process for inference. With ICNNs we further impose that the energy function is convex in the output space  $y$ , so that global minimization with respect to  $y$  is feasible. Still, this constraint seemed to have a negative impact in some tasks [11, 59].

Prediction with SPENs should in general be slower than with feed-forward approaches, since more steps are needed, including the backpropagation computation of  $\nabla_y E_w(x, y_{t-1})$ . We will nonetheless see that, in the application of Chapter 4, we can reduce the number of gradient-based inference steps to just one, thus making the running time more similar to a feed-forward approach, and still maintaining a good quality in the generated samples.

Because the inference procedure in SPENs is itself an iterative procedure based on taking gradient steps in search of a minimum of the loss, to do gradient descent on such a model would require backpropagation through that whole process. In the next section, after considering other possibilities, we will indeed present a method that employs this approach.

### 2.3.2 End-to-End Learning for SPENs

To train SPENs, we may use the implicit function theorem to differentiate the loss with respect to the parameters of the energy function. Still, this is not the method that was preferred [7], as prior experience showed that it is computationally expensive [58]. We could also consider the method used by Belanger and McCallum in the paper in which SPENs were first proposed [7], which used a SSVM loss for training [5, 60]. In this approach, computing just one gradient requires exact energy minimization with respect to  $y$ , and it is difficult to predict how much effect an error in this computation has on the overall learning task. Belanger et al. found that this method often works worse than the one they proposed in a follow-up paper [11], namely an *end-to-end learning* method for SPENs which, in addition, considerably simplifies the implementation of SPENs. End-to-end learning of a gradient-based inference procedure had been previously introduced in [58], but Belanger et al. introduced modifications to the aforementioned work for faster prediction.



**Figure 2.2:** Backpropagation through the unrolled computational graph of the gradient-based inference in SPENs, which forms the basis of the end-to-end learning algorithm. Reprinted from [57].

To understand end-to-end learning of SPENs, consider first that we are dealing with a feed-forward model  $f_\omega$ . In this case, the loss for one observation is  $\mathcal{L}_\omega = L(y, f_\omega(x))$ . If  $f_\omega$  is differentiable, we can just propagate the gradient through the loss and the prediction back into the parameters of the model, and use an optimization algorithm to search for parameters that minimize this loss. We would like to train SPENs in such an end-to-end manner, which has proved itself to be very successful and it is often not difficult to implement. Consider that instead of a feed-forward model,  $f_\omega$  corresponds to an indirect map, i.e., the minimizer with respect to  $y$  of an energy network  $E_\omega(x, y)$ . The loss function would then be  $\mathcal{L}_\omega = L(y, \arg \min_y E_\omega(x, y))$  and, if we had an  $\text{Algorithm}_\omega(x)$  for approximately searching this minimizer, the loss would be  $\mathcal{L}_\omega = L(y, \text{Algorithm}_\omega(x))$ . This process is in general not differentiable and therefore we can not use gradient-based learning. However, suppose that  $\text{Algorithm}_\omega(x)$  is the usual gradient descent algorithm in  $y$ , i.e., the inference process for SPENs that we have already introduced in the last subsection, unrolled over  $T$  iterations:

$$\text{Algorithm}_\omega(x) = y_0 - \sum_{t=1}^T \eta_t \nabla_y E_\omega(x, y_{t-1}) .$$

Suppose also that all the steps that are used by the algorithm are themselves differentiable. In this case, the whole algorithm is itself also differentiable, and it is possible to perform backpropagation through the entire unrolled computational graph (depicted in Figure 2.2).

With end-to-end learning for SPENs, we also need the components of the energy network to be twice differentiable. That is the case since we need each step of the inference process (Equation 2.4) to be differentiable with respect to  $\omega$ , and we already include in each step the gradient of the energy function with respect to  $y$ . For this reason, the authors of [11] did not use the ReLU nonlinear activation in their energy networks. Instead, they used the Softplus function, given by:

$$\text{Softplus}_\beta(x) = \frac{1}{\beta} \log(1 + \exp(\beta x)) . \quad (2.5)$$

This function acts as a twice differentiable approximation to the ReLU and, in fact, tends to that function as  $\beta \rightarrow \infty$ . Recently, this end-to-end training method became much simpler to implement, since we can employ the automatic higher-order differentiation capabilities of modern neural network libraries like PyTorch [8], instead of the finite difference method that Belanger et al. implemented [11].

During training, the loss function is defined as a sum of losses at each step of the iteration between the inference iterates  $y_t$  and the ground truth value  $y$ :

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^T w_t L(y_t, y) , \quad (2.6)$$

using a weighting scheme such as  $w_t = \frac{1}{T-t+1}$  that gives more weight to the last iterates, encouraging the energy function to be shaped so as to give good predictions early.

In some cases, we need to change the optimization space. For instance, in the example given in [57], one may assign one of  $D$  possible values to  $K$  labels, i.e., we wish to learn a mapping from the input  $x$  to  $\{0, \dots, D-1\}^K$ . One could optimize SPENs in the constrained space of the probability simplex on  $D$  elements  $C_{K,D} = \{y \mid y \in [0, 1]^{K \times D}, \forall i \sum_j y_{ij} = 1\}$ . Belanger et al. considered multiple ways to optimize within it, but it was simpler, and most of the time produced better results, to instead optimize in the unconstrained space of the logits  $\ell$  of  $y$ , such that  $y = \text{Softmax}(\ell)$ . In this space, the gradient-based inference step is:

$$\ell_t = \ell_{t-1} - \eta_t \nabla_\ell E_\omega(x, \text{Softmax}(\ell_{t-1})) , \quad (2.7)$$

where the Softmax function has the form:

$$\text{Softmax}(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^K \exp(y_j)} , \quad (2.8)$$

and it is applied separately to each of the  $D$  components of  $y$  (each with dimension  $K$ ). We are now working in the space  $(-\infty, \infty)^{K \times D}$  and we are free to train with many different common optimizers, such

as Adam [61]. We naturally need to come back to the  $y$  output space and apply a discretizing function to obtain the final class prediction.

It is worth emphasizing that when we use SPENs we have to concern ourselves with at least two different learning rates. The first one,  $\eta$ , is the *inner learning rate*, which controls the speed at which the gradient-based inference procedure proceeds. In this dissertation, we will impose that all of them have the same value, but we might also have, as in the example above, different  $\eta_t$  for each inference step. The second is the *outer learning rate*, which is the one we would normally call learning rate and that actually changes the model. We provide it to the optimizer, such as the aforementioned Adam, and it regulates the speed of change of the parameters of the energy function.

Because inference in SPENs is done by optimization in output space on a generally non-convex energy surface, global energy minimization can not be guaranteed (unlike with some CRF models with a simple structure), and no guarantees about the number of iterations until convergence can be made. Normally, we fix the number of iterations to be low so that we are learning an energy function in which the prediction process converges quickly and which, at the same time, consumes less computational resources. Even though other criteria are possible [57], in this dissertation we will consider that the number of iterations  $T$  of gradient-based inference is fixed, and that  $T$  takes the same value at both training and inference time.

## 2.4 Related Work

There has been a considerable amount of research dedicated to the most relevant type of data completion problem to this dissertation, which is the inpainting problem. In this task, missing pixels in an image are often given as a mask over the visible pixels. It is a difficult task that requires some level of semantic understanding when the area of the image affected is large, in which case it is not enough to fill the missing parts by directly adapting parts of the surrounding image. The most impressive work in this area has been achieved with feed-forward approaches but, as we have previously explained, it is natural to apply EBMs to data completion problems and some work has previously been done in this area. We review both cases in the subsequent subsections.

### 2.4.1 Feed-Forward Models

Many image inpainting methods employ adversarial training since GANs have proved to be very successful at unconditional image generation. The Context Encoder [62] employs a convolutional auto-encoder architecture that accepts the masked input and tries to output the image with the holes filled in.

A standard pixel-wise reconstruction loss is then computed, together with an adversarial loss that is the result of a separate discriminator network attempting to indicate whether a sample is real or not. A later paper [63] increased the quality of the completion of the Context Encoder by using the intermediate features of a pre-trained VGG network [21] to add an additional loss term that penalized differences in texture between the completed hole versus the visible part of the image. Other work [64] employed two discriminator networks, a global one that is aware of the whole generated image, and a local discriminator centered around the completed region to emphasize it. It also used dilated convolutions to increase the receptive field in the generator.

The DeepFill model [65], which includes two generator autoencoders and two discriminator networks, was able to enhance the results of the previous studies by introducing a contextual attention mechanism to model spatially distant dependencies in the generated image. Like other models, the first generator network performs coarse reconstruction of general features of the missing patch. The second generator is responsible for the refinement of it, and it is the one that employs the contextual attention mechanism. This entirely differentiable procedure computes correlations between small background patches with the foreground patch that we wish to reconstruct, and uses this information in the decoder part of that generator. Another model [66] introduced partial convolutions, which distinguish between visible and hidden input pixels, heuristically updating the mask to be applied in the next layer, and enabling the model to effectively deal with irregular shaped masks. The model DeepFill v2 [67] introduced gated convolutions, which are similar mechanisms to partial convolutions. However, they are differentiable and function as learnable mechanisms for selecting features at different layers. These gated convolutions also allow for user guidance of the inpainting process to be included. Many of the innovations that we have previously mentioned were also incorporated, making this version of DeepFill one of the best performing methods overall.

### **2.4.2 Energy-Based Models**

As we have previously explained, it is natural to use EBMs in data completion tasks. There have been experiments with times series data imputation [68], and image inpainting tasks have also been a frequent testing ground for EBMs. These previous studies have for instance used Input Convex Neural Networks (ICNNs) [59], which have a similar formulation to SPENs but are convex in the space of missing data, which may limit its expressive power. Some of the probabilistic models we have previously mentioned were also used for image completion tasks. Du and Mordatch [38] adapt the Langevin dynamics to only modify the hidden pixels and maintain the visible ones fixed, and Song and Ermon [45] modify their annealed Langevin dynamics in a similar way. Song et al. [46] modify the SDE they use to generate

unconditional images to a conditional SDE of the hidden pixels based on the visible ones. Note that those EBMs naturally allow for a diversity of possible completions to be generated.

## **2.5 Overview**

In this chapter we reviewed the background information necessary to understand the algorithm that we propose. This includes the fundamentals of feed-forward and energy-based approaches and a review of the SPEN framework, which forms the foundation for the data completion algorithm that we propose in the next section. Finally, we described other data completion algorithms that are applied to a task that is similar to our own, namely, the task of image inpainting.



# 3

## Energy Networks for Data Completion

### Contents

---

3.1 Energy Network Architectures . . . . .	26
3.2 Inference and Learning . . . . .	28
3.3 Overview . . . . .	32

---

Having described in the previous chapter the necessary background information on Structured Prediction Energy Networks (SPENs), the question arises of how we can adapt the algorithm (originally applied to problems such as multi-label classification or image-denoising) to the data completion setting.

Most models that are not energy-based, when used for data completion, have an architecture specific for that task, as we have seen in Section 2.4. Similarly to other EBMs, with our model we can use different architectures that may be already familiar to researchers for discriminative tasks. Dependencies between any of the components of the data may be represented, whether those components happen to be visible or not. This is possible without having to worry about the learning algorithm, as the SPEN paradigm directly provides a learning procedure.

In this chapter, we discuss the modifications that we have made to the original SPEN model. This includes the specification of the class of energy networks to employ, the use of different initialization procedures, and some other alterations that were made to facilitate learning.

### 3.1 Energy Network Architectures

In our data completion setting we consider the separation of one sample into the multiple components that constitute it. Some of those components are observed, and they constitute  $x$ . Others are absent and we wish to generate good estimates for them. They constitute the hidden components  $y$ .

Many tasks can be seen as being part of this class of data completion problems. For example, consider the very general problem of learning a mapping from input to output. We may simply consider that the output is part of the original data but it is missing, and we wish to complete it. Therefore, in this dissertation, we further constrain the class of data completion problems that we consider.

We assume that each component of one sample, regardless of being part of  $x$  or part of  $y$ , can be represented by a vector with the same number of dimensions, i.e., it belongs to  $\mathbb{R}^k$ , for fixed  $k$ . These are the vectors that serve as input. Note that this requirement does not exclude textual representations such as word embeddings. A task that is included in this paradigm is the generation of missing values in tabular data consisting of features of the same type. Another one is image inpainting or image completion, in which we wish to generate plausible values for missing pixels. In this task each component is given by a vector with 1 or 3 dimensions, corresponding to black and white or color images in the RGB space.

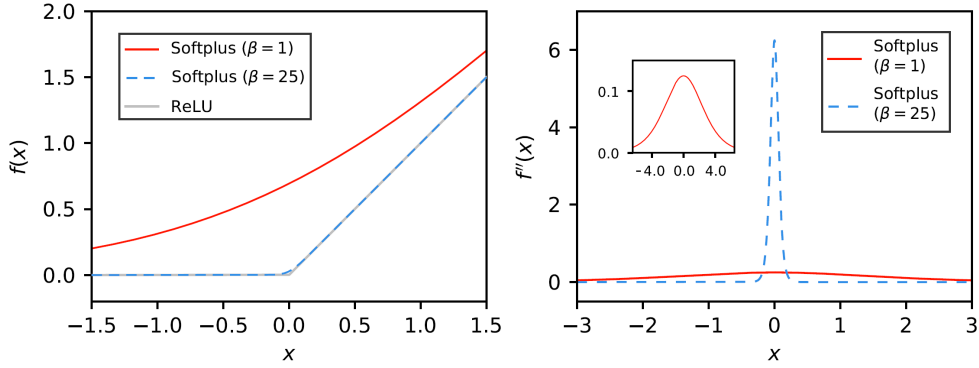
Despite the focus of our dissertation not being this, there is no a priori reason why one cannot relax the constraints that all elements can be represented by a vector with the same size. We could, for example, complete tables with missing data that is either quantitative or qualitative – whose components

are in general represented by vectors of different sizes. To accomplish this, we just need to adapt the energy function so that it is able to interface in different ways with the features, depending on their type. This framework also includes text, audio, or video completion but in this dissertation we do not consider an explicit modeling of the sequence structure that may be needed to be able to execute those tasks adequately. Moreover, we do not consider tasks in which the size of one observation varies.

By choosing this subclass of data completion problems, in which features can be represented by a vector with the same size, we can explicitly consider only energy functions with an uniform architecture across those features. This means that, in particular, the energy networks that we consider are indifferent to whether a feature is visible or hidden. In other words, they can naturally be seen as a function of a single input  $z$ , which contains both the visible elements  $x$  and the invisible ones  $y$ . This choice for the type of energy functions is not the only possible one, but it is natural, since it suits the fact that we are dealing with data that was originally aggregated, and we symbolically subdivide it in  $x$  and  $y$  to express that only part of the original data is available to the algorithm. This choice also suits the fact that we want to use neural networks to directly model dependencies between any of the components of the data, regardless of them being part of the observed input  $x$  or the hidden output  $y$ . This is different from the original experiments with SPENs in which the output elements  $y$  (which in this case always correspond to the same variables) are scored independently of the values of  $x$  given the cached values of a feature network  $F(x)$ , as can be seen in Equation 2.3. This was chosen so that computing resources could be saved, but the use of the feature network results in a loss of expressive power, in the sense that direct relationships between members of  $x$  and  $y$  are not modeled explicitly. For our work we considered a different architecture, without a feature network and not distinguishing between visible and hidden elements of the data.

Moreover, to simplify the notation, we express the energy of  $x$  and  $y$  in their respective place in the input as  $E(x, y)$ . For example, in a problem with images,  $(x, y)$  might refer to a *single* image  $z$  with size  $m \times n$ , in which we distinguish pixels that are visible, constituting  $x$ , and pixels that are invisible, constituting  $y$ . In practice, we mostly compute the energy function with elements that are different than  $y$ , such as the iterative guesses that we will talk about in the next section. These iterative guesses have as many components as  $y$ , and thus can be arranged in the corresponding place together with  $x$ .

Neural networks often use nonlinear functions as part of their architecture, and a common choice is the ReLU [13]. Belanger et al. [11] used a Softplus function with a large value for  $\beta$ , such as 25, as a twice differentiable approximation for the ReLU to be used in SPENs. Their choice was motivated by a higher similarity with the ReLU function when this parameter has a higher value (see Figure 3.1). However, in our case, this choice hindered learning. In order to differentiate the inference process, one



**Figure 3.1:** (Left) Softplus functions for different values of  $\beta$  and ReLU. (Right) The second derivatives of the Softplus functions. A higher value for  $\beta$  makes the Softplus function more similar to the ReLU, but their second derivatives have a smaller range in which they are substantially different than zero. Note that, since the derivative of the Softplus function with  $\beta = 1$  is the Sigmoid, the red line in the right-hand plot also denotes the first derivative of the Sigmoid function.

must differentiate  $\nabla_{\ell} E_{\omega}(x, f(\ell_0))$  with respect to the parameters  $\omega$  of the energy function. On the other hand, this differentiation involves knowing the second derivative of the Softplus function and, as also depicted in Figure 3.1, the second derivative is narrower as a result of a higher  $\beta$ , implying that a smaller range of values in the domain produces second derivatives that are considerably different than zero. This causes upstream gradients to not properly flow backwards into the parameters when the values of the domain are outside that range. We chose to work with Softplus functions with  $\beta = 1$  as the nonlinear activations, but in other applications different values for  $\beta$  may be appropriate.

## 3.2 Inference and Learning

Performing the inference process with SPENs implies unrolling a computational graph of a gradient-based search operation on a twice-differentiable energy network. Because that whole procedure is also differentiable, we can use a regular stochastic optimization algorithm to train the energy network such that we obtain the output we want in the end of it.

The basis of the algorithm is the inference step defined in Equation 2.4. The full gradient-based inference, in which  $T$  of those steps are iterated, is given by:

$$y_T = y_0 - \sum_{t=1}^T \eta_t \nabla_y E_{\omega}(x, y_{t-1}). \quad (3.1)$$

The components of the data that we wish to complete often cannot take any value, but instead belong to an interval  $I$  or, in case they are  $K$ -dimensional vectors, to the Cartesian product of  $K$  such sets, i.e.,

$I^K$ . In this case we can apply the method of Belanger et al. [11], in which gradient descent was actually realized in the unconstrained space of the logits of  $y$  (see Equation 2.7). More generally, with the right choice of a mapping  $f: \mathbb{R}^K \rightarrow I^K$  (such as the softmax function in the aforementioned equation), we can bring the iterate  $\ell$  from the unbounded space  $\mathbb{R}^K$  (in which we can take gradient steps freely) to the relevant set  $I^K$ . A step in the unconstrained space is given by:

$$\ell_t = \ell_{t-1} - \eta_t \nabla_{\ell} E_{\omega}(x, f(\ell_{t-1})), \quad (3.2)$$

in which the transformation  $f$  is applied independently to each  $K$ -dimensional component of the logits, and the full inference procedure is given by:

$$\ell_T = \ell_0 - \sum_{t=1}^T \eta_t \nabla_{\ell} E_{\omega}(x, f(\ell_{t-1})). \quad (3.3)$$

The initial logits  $\ell_0$ , as well as the initial  $y_0$  in Equation 3.1, are generated by an initialization routine that depends on the visible data,  $\text{Init}(x)$ , which we will describe shortly.

Another possibility corresponds to cases in which each component of the data that we wish to generate is discrete with  $N$  ordered values (e.g., a component of a color channel with values between 0 and 255). Similarly to what is done in other cases, we can normalize all the data to an interval  $I$  such as  $[0, 1]$  and, for each component of  $y$ , we can relax the problem from a search for the minimum energy among  $N$  values, to a search within that interval. To obtain discrete values, for example, when computing scores, we can simply apply a discretizing function. The discretizing function should be increasing and it should map each of the  $N$  subintervals of a handpicked partition of the interval  $I$ , to the  $N$  discrete values. Being discontinuous and integer-valued, this function should not be part of the computational graph that we must be able to differentiate.

If we have a problem in which we wish to classify each component of  $y$  into two classes, as will be the case later in Chapter 4 when we generate black and white images, we can model, for each component, a non-degenerate probability distribution that is fully identifiable by only one number  $p \in (0, 1)$ , which is the probability for one of the classes to occur. We can work with the logit instead by applying the transformation  $\ell = \text{logit}(p)$ , and thus obtain an equivalent problem in an unconstrained space. Furthermore, to convert back from logit space to the probability space of  $p$ , we just need to apply the inverse of the logit, which is the Sigmoid function (also denoted by  $\sigma$ ), given by:

$$\sigma(x) = \text{Sigmoid}(x) = \frac{1}{1 + \exp(-x)}. \quad (3.4)$$

The complete inference algorithm is then given by:

$$\ell_T = \ell_0 - \sum_{t=1}^T \eta_t \nabla_{\ell} E_{\omega}(x, \sigma(\ell_{t-1})). \quad (3.5)$$

We might also have chosen a different strictly increasing mapping  $f$  from  $\mathbb{R}$  to the interval  $(0, 1)$ , instead of  $\sigma$ . A possible discretizing function for this problem, which is the one that will be used in our practical applications, is described later in Equation 4.1.

Coming back to the functioning of the algorithm, consider Equation 3.2 again. After computing  $f(\ell_t)$ , those values will be assigned to their place in a tensor together with the observed data  $x$  before the forward pass of the energy function. As we will see in Chapter 4, a fixed and quite low value for the number of iterations  $T$  is used, in line with previous recommendations [11].

Similarly to Equation 2.6, the intermediate and final values  $f(\ell_t)$  are compared with the ground truth value  $y$  to compute a loss function, which is expressed by:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^T w_t L(f(\ell_t), y), \quad (3.6)$$

using the same weighting scheme  $w_t = \frac{1}{T-t+1}$ . We now wish to backpropagate the gradient of the loss along the whole unrolled computational graph of the inference process in Equation 3.3. Our implementation is assisted by a feature of PyTorch that allows the creation of the computational graph of the backwards pass that produces  $\nabla_{\ell} E_{\omega}(x, f(\ell_{t-1}))$ . Note that this gradient is part of the inference process expressed in Equation 3.3. The computational graph is itself differentiable if all the operations that define the energy function are twice differentiable. In this case, we can differentiate the whole inference process and thus obtain the gradients with respect to the parameters  $\omega$  of the energy function. By feeding these gradients to a first order unconstrained optimization algorithm, we can then train the energy function so that our algorithm performs as desired. During test time we avoid the overhead that comes with the creation of the computational graph of the gradient of the energy function by turning off the flag `create_graph` of the automatic differentiation routine.<sup>1</sup>

Furthermore, we remark that the last layer of the energy network (which will often be a linear layer) should not have a bias term. This is because the derivative of this energy function with respect to its inputs  $\nabla_{\ell} E_{\omega}(x, \sigma(\ell_{t-1}))$  does not depend on this term and, in the process of differentiating the loss,

---

<sup>1</sup>A separate PyTorch package for higher-order differentiation, which is called `higher` [69], may simplify further the approach taken here. Furthermore, as it supports differentiation with respect to internal parameters of many different PyTorch optimizers, it may also enable an extension of our approach that uses different optimizers with learnable parameters in the inner loop, instead of the standard gradient descent.

PyTorch does not output a zero for the derivative of that gradient term with respect to the parameters  $\omega$ , but instead a `None` value.

## Initialization of the Inference Iterates

Different initialization procedures `Init(x)` can be used for generating the initial inference iterates  $\ell_0$ . Note however that, as described earlier, we often need to pass them to a function like the Sigmoid before integrating them together with the observed data. Only afterwards can the energy function be applied. This effectively constrains the permissible range of values that the initialization routine can output. To see why, take as an example the case with two classes described earlier, in which the Sigmoid was used as the mapping from  $\mathbb{R}$  to the interval  $(0, 1)$ . Note that the derivative of that function is low for values that deviate too much from zero. This is visible in the right side of Figure 3.1 in red, as the the first derivative of the Sigmoid function is the second derivative of the Softplus with  $\beta = 1$ . If we do not place some care into giving the Sigmoid inputs that are close to zero, the upstream gradients will not flow backwards through the function properly. Therefore, in those cases, either we design the initialization procedure to output values that already have a low modulus, or we should pass them through a function that maps  $(-\infty, +\infty)$  to an interval around and close to zero, such as  $[-1, 1]$ . As will be described in Chapter 4, we choose to use a function that compresses linearly the values around 0, dividing by the maximum of the absolute value of the initial logits  $\ell_0$  with the `Compress` function given by:

$$\text{Compress}(x)_i = \frac{x_i}{\max_i \text{abs}(x_i)}. \quad (3.7)$$

The different initialization routines that we have considered are:

- **Initialization with an estimate:** This is closer to what is suggested by Belanger et al. [11], which used a feature network to initialize  $y_0$ . In this case, we have a pre-trained initialization function that depends on the visible data and suggests initial values for the iterates  $\ell_0$ .
- **Initialization to zeros:** We simply initialize all the iterates  $\ell_0$  to zero.
- **Random initialization:** Another possibility is to initialize  $\ell_0$  by sampling random numbers from a distribution concentrated near zero. This way, we require that a larger variety of gradient descent paths is capable of generating good solutions. At test time, we may choose to initialize  $\ell_0$  with zero instead. As we will see in Chapter 4, this seemed to work better.

It is important to emphasize that, with those last two initialization procedures, the job of the energy function is not anymore to enhance an initial suggestion. Instead, we now expect the energy function to

be sufficiently rich to do the entire job of modeling dependencies without any external help.

When the iterates are passed through the Sigmoid function in both the first and the last possibilities for initialization, respectively (i.e., initialization with an estimate or at random), one may need to use the `Compress` function to ensure that learning can proceed.

### **3.3 Overview**

This chapter described how we have modified the SPEN algorithm so that it can handle a particular class of data completion problems. We detailed which class of energy networks we are going to consider, how the process of learning proceeds, and how we can implement it in PyTorch. We introduced some changes that facilitate learning by promoting a better gradient flow. We also discussed different initialization methods for the inference process. Moreover, we have seen how the learning algorithm can be adapted to produce different types of data. In particular, we discussed how it applies to an example in which there are only two classes from which we wish to generate values, which will be valuable help for the practical application detailed in the next chapter.

# 4

## Experiments with Font Completion

### Contents

---

4.1 The Fonts Dataset . . . . .	34
4.2 Data Completion in the Fonts Dataset . . . . .	36
4.3 Loss Function and Metrics . . . . .	38
4.4 Convolutional Blocks . . . . .	40
4.5 Baseline Models . . . . .	41
4.6 Deep Energy Networks . . . . .	47
4.7 Discussion of the Experimental Results . . . . .	56

---

In the previous chapter we proposed a procedure to bring SPENs into the realm of data completion. We can now specialize that algorithm for handling a dataset composed of fonts, with the letters that constitute each font formatted as images. This is a subproblem of the data completion class of the last chapter, with which we can employ energy functions with an uniform architecture across features.

We begin this chapter with a description of the font dataset, and with a definition of the data completion task, which consists of generating images for letters that are not visible in a font. We proceed with a discussion about the choice of the loss function for training the energy networks, as well as the score for evaluating the quality of the generated missing data. We then present different baseline models that we have built for comparison. Subsequently, we discuss the architecture of the specific energy network that we have employed in this problem, and provide practical considerations for applying our method. Finally, we compare the results that we obtained with different configurations of the algorithm, and also the results that we have obtained with SPENs versus those from the baseline models.

## 4.1 The Fonts Dataset

In order to build a model to test the capabilities of the algorithm proposed in the previous chapter, we use a visual dataset through which experiments can quickly be iterated. We downloaded a dataset containing open access `ttf` font files from the [DaFont](#) website [70]. The dataset is composed of 1076 bitmap fonts, each comprising the 26 letters of the English alphabet in their uppercase form. As usual, we divided the data randomly into three datasets: the training dataset (with 716 fonts), the test dataset, and an additional validation dataset for hyperparameter tuning (each with 180 fonts). A sample of the fonts from the training dataset is depicted in Figure 4.1.

With this choice of dataset, we avoid introducing additional complexity that comes with performing general image inpainting, which is a standard data completion task in image domains.

- As we will see in the next section, it is simple to define the data completion task in this font dataset as the generation of missing letters of a given font, knowing only the provided letters. Therefore, we avoid having to choose between different types of masks in image inpainting, or having to produce these masks manually.
- If we wish to perform image inpainting with complex pictures, the choice of a customized loss function can itself be a difficult process. By choosing the images to be small black and white fonts, we are able to avoid this issue with a simple loss function and score for training and evaluation.

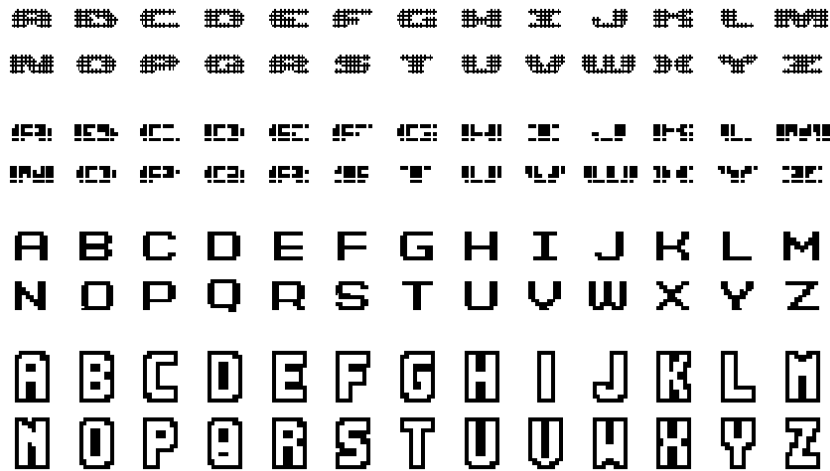


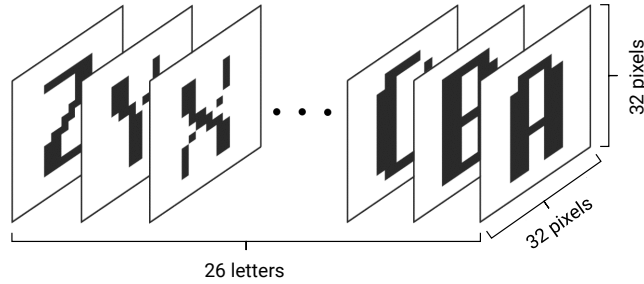
Figure 4.1: Examples of fonts from the training dataset.

- Because the inputs are simple bitmap fonts, we can format them into very small black and white pictures without a need for an anti-aliasing treatment. This has two benefits:
  - We can reduce the memory that is needed to store the small pictures, as well as shorten the time it takes to perform multiple experiments.
  - Because we do not use gray values, we can encode each black and white pixel solely with the values of 0 and 1, in a single channel. As a result, we can simply treat the generation of the missing images as a binary classification problem for each pixel, using what we have introduced in Section 3.2.

Note that a data completion task in this dataset, defined in this way, is part of the class of data completion tasks specified in Section 3.1, since the components of the data are of the same data type, and each component can be represented by a vector with the same number of dimensions.

Some minor resizing was made for each font so that all letters fit in a  $32 \times 32$  picture. We have chosen this exact size because it was the smallest which is a multiple of 2 (so that we could downsample the pictures with max pooling layers of a CNN), that still keeps a reasonable image quality.

We use the PyTorch neural network library [8], and format an entire training example containing a number of fonts equal to `batch_size` as a  $(\text{batch\_size}, 26, 32, 32)$  4-dimensional tensor. The PyTorch default of using 32 bytes floating-point numbers for use in the GPU was kept. However, we often take the primary unit to be the font, which is represented by a  $(26, 32, 32)$ -shaped tensor. This 3D arrangement of a font is depicted in Figure 4.2. Symbolically, we refer to the  $l$ -th letter of the  $f$ -th font of the dataset as  $F_l^{(f)} \in \{0, 1\}^{32 \times 32}$ .



**Figure 4.2:** Example of a font, highlighting the packing of the letters into a 3-dimensional tensor

## 4.2 Data Completion in the Fonts Dataset

In this section we fully describe the learning task that we have introduced before. We have already explained in Subsection 4.6 why the energy network formulation is natural in this problem.

The data completion task can be defined as the generation of missing letters in a font, given that only some of the 26 letters that compose the font are presented to the algorithm. By establishing the task like this, there is a well-defined application scenario corresponding to a font designer who wishes to automatically generate an entire font, after having drawn only a few of its letters. The algorithm must be able to recreate different levels of structure in the image data, i.e., the structure imposed by the alphabetic ordering of the letters, and the structure which is inherent to the style of the font, which should permeate all missing letters. We can evaluate both requirements visually and, moreover, we can employ the quantitative measures described in the next section.

We can describe the task in the same way as in Section 3.1. To evaluate the algorithm, we have a test set composed of fonts  $z = (x, y)$ , of which we only know some of the letters  $x$ . We will call them *visible letters*, in opposition to the *hidden letters*  $y$ . The algorithm does not have access to those letters and will try to generate them. Therefore, the objective to generate a prediction  $\hat{y}$  for the images of the hidden letters  $y$ , knowing only the images of the visible letters  $x$ .

In order to produce similar letters as the inputs  $x$ , the images we generate are also black and white, i.e., they are composed only of 0 and 1 pixels. However, with SPENs, we always relax discrete problems so that, instead of generating directly the letters in the space  $\{0, 1\}^{\text{n\_hidden\_letters} \times 32 \times 32}$ , we generate matrices in the space  $[0, 1]^{\text{n\_hidden\_letters} \times 32 \times 32}$ , where `n_hidden_letters` is the number of letters that are hidden in this font. As we have detailed in Section 3.2, we can use an intermediate soft guess  $\hat{p}_i \in [0, 1]$  (a gray pixel) that, because of the domain that it belongs to, we may consider as a probability.

In the end, we apply a discretizing function to obtain a 0 or 1, which is simply:

$$\text{Discretize}_\tau(x) = \begin{cases} 1, & x \geq \tau \\ 0, & x < \tau \end{cases}, \quad x \in [0, 1], \quad (4.1)$$

which is a shifted Heaviside function limited to the  $[0, 1]$  domain, applied pixel-wise. Unless indicated differently, we will take  $\tau = 0.5$ .

When we say that we hide certain letters and show others, we are in fact filling the matrix corresponding to the hidden letter with zeros, i.e., we mask it. Therefore, we still present the algorithm with a whole font represented as a  $(26, 32, 32)$  tensor, with each letter in its alphabetic order along the first dimension if it is visible and, otherwise, we fill the corresponding  $(1, 32, 32)$  slice with zeros. We also pass as input to the learning algorithm, for computing the loss function and scores, a mask tensor with boolean values indicating which letters are visible and which ones are not.

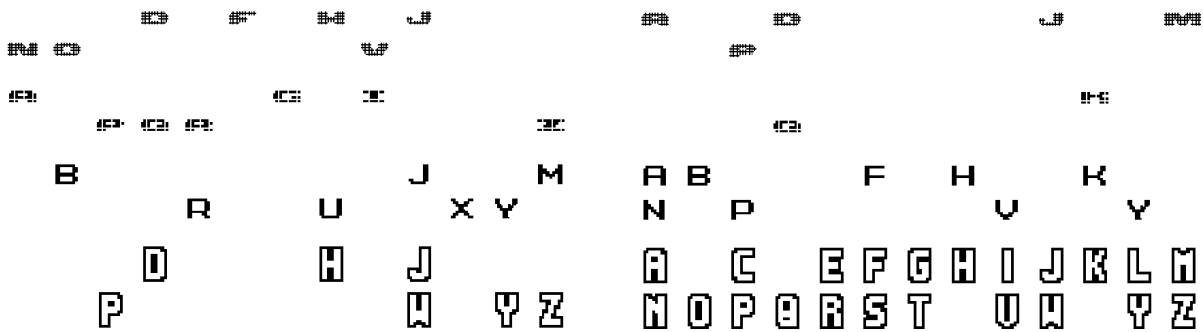
### Input Modes

Two different modes were created to choose which letters are shown to the algorithm, thus requiring that the remaining ones are for the algorithm to generate:

- **Fixed mode:** From each font, a fixed number of letters, with value `n_visible_letters` between 1 and 25, are chosen randomly to be input to the algorithm, and the others are masked. We chose this range because at least one letter has to be presented to the algorithm for it to be able to learn anything, and we want it to have to generate at least one letter.
- **Mixed mode:** For each font a number `random_n_visible_letters` is drawn randomly from the integers between 1 and 25. Then, `random_n_visible_letters` of the letters are chosen randomly to be input to the algorithm, and the others are masked. This way every possible configuration of visible letters can happen with a reasonable probability.

With these two different modes specified, we can then train the algorithms with a fixed number of visible letters, or in a situation in which every possible configuration can be present. This is useful to test the dependence of the performance of different algorithms on the information they are provided with.

As it is commonly done, the order by which different fonts are presented to the algorithm during training is random. We have also configured the training procedure so that, in every new training epoch,



**Figure 4.3:** The two different input modes regarding the choice of which letters from a given font are visible to the algorithm, which then has to generate the other ones. *(Left)* Fixed mode: We always present 7 letters, chosen randomly, as input for the algorithm (here `n_visible_letters = 7`). *(Right)* Mixed mode: The number of letters presented to the algorithm is itself random, ranging from 1 to 25.

we make a different choice on which letters of each font are visible.<sup>1</sup> This is the case for both input modes. This way, instead of deciding for each font in advance which letters are shown or fixed, we present a larger variety of situations during training. This can be seen as a type of dataset augmentation, and it is comparable to the common procedure of drawing a random mask in image inpainting tasks. However, for the validation set, we make that choice only when we set it up, so that each font has always the same letters visible. For the test set we do the same, except that we repeat the testing process 100 times with different choices of the letters that are visible. This is made for the collection of statistics regarding the final results. For a better comparison, the choice regarding which letters are visible over the 100 trials is reproducible.

In order to train our models we used the GPUs of the Google Colab platform. This platform has obvious advantages for training the energy networks since significant computing speed and memory are required.

### 4.3 Loss Function and Metrics

A standard loss function that can be used to train font completion models is the Binary Cross-Entropy (BCE). This pixel-wise loss independently penalizes guesses for the pixels which are far away from the truth, and it corresponds to the average negative log likelihood of a Bernoulli model for each pixel. Recall that the generation of a black and white image can be seen as multiple binary classification problems and that it is enough to model, for its  $i$ -th pixel  $\hat{y}^{(i)}$ , the probability  $\hat{p}^{(i)}$  of being black. The loss between

<sup>1</sup>To make sure that every time we run the algorithm with the same configuration the same outcome is reached, we have used seeds for the pseudo-random number generators of those choices. However, if the code is run in the CPU instead of the GPU this will not happen, and neither is this guaranteed to happen when we use different PyTorch versions.

one ground truth pixel  $y^{(i)}$  and a generated pixel with modelled probability  $\hat{p}^{(i)}$  is given by:

$$\text{BCE}(y^{(i)}, \hat{p}^{(i)}) = - \left[ y^{(i)} \log \hat{p}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]. \quad (4.2)$$

The BCE loss between two different possibilities for the same set of letters is simply the average of the BCE loss between the pixels of the individual letters.<sup>2</sup>

An objective score for evaluating the generated letters, that we deemed to correlate well with perceived visual similarity, is the Intersection over Union (IoU) score. It is applied to the final generated letters  $\hat{y}$ , i.e., after they have passed through the discretizing function. This score is often used to evaluate the similarity of bounding boxes in object detection problems, but we can adapt it to compute the similarity of the letters. For a single letter, we define the IoU score as:

$$\text{IoU} = \frac{|\text{B} \cap \hat{\text{B}}|}{|\text{B} \cup \hat{\text{B}}|}, \quad (4.3)$$

where we denote by  $\text{B}$  the set of pixels that are black in the ground truth letter, and by  $\hat{\text{B}}$  the set of pixels that are black in the corresponding generated letter. In other words, this score favors letters that resemble each other by computing the percentage of pixels that are black in both the ground truth letter and the generated one, versus the total number of pixels which are black in at least one of the letters. The final IoU score for a certain algorithm is simply the average of all the IoU scores for the letters that count for the computation. Both in the computation of the BCE loss and the IoU scores, the letters that are considered will depend on the chosen algorithm. This is discussed further ahead (see Subsections 4.5.1, 4.5.2, and Section 4.6).

We use the IoU metric to evaluate which version of the parameters of an algorithm to keep during training, and to compare algorithms during the hyperparameter optimization phase. Both of these choices are made based on the performance of the IoU metric in the validation dataset.

The Hamming Loss, which measures the percentage of pixels with the same color in both images, is not a good choice for this problem since, for example, a good score might be obtained for small letters that do not coincide at all.

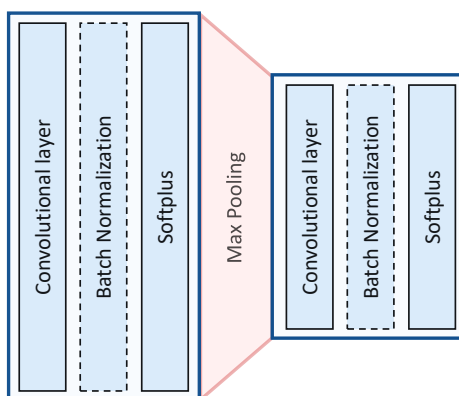
---

<sup>2</sup>As we will see later, because both our models output the logits of the guesses, the PyTorch function that we use is in fact a slight modification of this BCE loss, which accepts as inputs those logits. It has a different implementation than the general BCE loss because it takes advantage of the log-sum-exp trick for numerical stability.

## 4.4 Convolutional Blocks

We will use convolutional layers as the main constituent of the autoencoder baseline in Subsection 4.5.2, and the SPEN-based EBM in Section 4.6, thereby enforcing an appropriate inductive bias for a task in which the spatial structure of the data is essential. We will base the architectures that we use in the VGG16 [71] neural networks that were introduced in Section 2.1.2, which were also the basis for previous encoder-decoder architectures such as SegNet [24].

We call a component that aggregates different operations that are commonly used together in these networks a *convolutional block*. We define a convolutional block as a collection of intercalated convolutional and non-linear operations, with an optional batch normalization layer in-between. Each convolutional operation employs filters that span the whole depth of the input and the same odd kernel side size  $s$  with padding  $p = \lfloor \frac{s}{2} \rfloor$ , chosen such that the spatial dimension of the resulting feature maps is the same as the inputs. The softplus nonlinearity function is used instead of the ReLU that is used in the VGG16 architecture because it is twice differentiable, which will be important with the energy networks.



**Figure 4.4:** Example of two encoder convolutional blocks, separated by a max pooling operation. The batch normalization layer is optional.

The free parameters of one of these blocks are the number of convolutional layers per block, the kernel side size, and the number of output channels of a certain block, which we call `out_channels`. Depending on whether we are dealing with an *encoder* or *decoder* convolutional block, the value of `out_channels` will have a different effect. With encoder blocks, `out_channels` defines the number of output feature maps of the first convolutional layer, and the subsequent ones have that same number of input and output channels. Conversely, with a decoder block (containing *transposed convolutional* layers), the number of output feature maps of each layer stays the same as the number of input channels for that block, until we reach the final layer, which yields `out_channels` feature maps. The decoder blocks

will be relevant in the following section, having to do with autoencoders. The energy networks in Section 4.6 will only use the encoder blocks.

As we will see in the next two sections, between convolutional blocks there will often be a down-sampling or upsampling operation being performed, so that consecutive convolutional blocks deal with feature maps of different shapes.

## 4.5 Baseline Models

We implement two baseline models for comparison with the EBM that we detail in the next section. Specifically, we describe a very simple model in Subsection 4.5.1 that also serves as an initialization procedure for the later EBM, and an an autoencoder-based model in Subsection 4.5.2 that makes use of the convolutional blocks that we have introduced in the last section.

### 4.5.1 Average Baseline

A truly naive first approach to solve the font completion problem would be to guess a certain missing letter by simply averaging all the images corresponding to the same letter in the training dataset, and then applying the discretizing function in Equation 4.1, with  $\tau = 0.5$ .

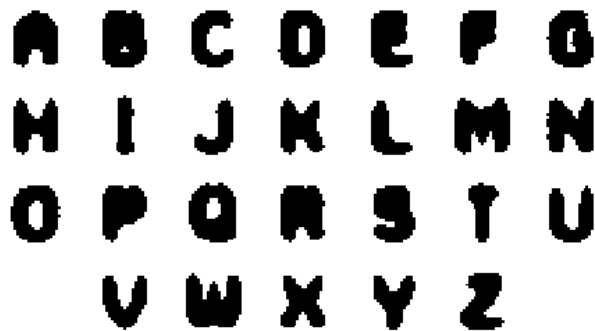
In a slightly more complicated way, we can search for the best performing threshold  $\tau \in (0, 1)$  in the validation set, i.e., the one which maximizes the IoU score between the resulting images and the fonts in that set. The complete expression of this averaging and discretizing process is given by:

$$F_l^{avg} = \text{Discretize}_\tau \left( \frac{1}{M} \sum_{f=1}^M F_l^{(f)} \right), \quad l = 1, \dots, 26, \quad (4.4)$$

where  $F_l^{(f)} \in \{0, 1\}^{32 \times 32}$  is the  $l$ -th letter of the the  $f$ -th font in the train dataset, and  $M$  is the number of elements of the train dataset. We found that the best threshold was  $\tau = 0.27$ , which corresponded an average IoU score in the validation set of 37.6%. Evaluating with that threshold in the test set, we obtained an average IoU score of 36.3%.<sup>3</sup>

Having only 36.3% of the black pixels co-occurring between the true and generated images is not a good result, and this can be confirmed by observing Figure 4.5, representing the average letters. In this

<sup>3</sup>To evaluate the estimates that we have generated by averaging, we just need to go through each font in the set and compute the IoU between that whole font and the letters in Figure 4.5. Later on, a more complicated process of testing will itself be dependent on which letters are visible and which ones are not, but here the guess that was made is completely indifferent to which letters we present the algorithm with.



**Figure 4.5:** Average of each letter in the training dataset, with threshold of the discretizing function  $\tau = 0.27$  chosen so that it maximizes the IoU score in the validation set. Using these images we obtained a score in the test dataset of IoU = 36.3%.

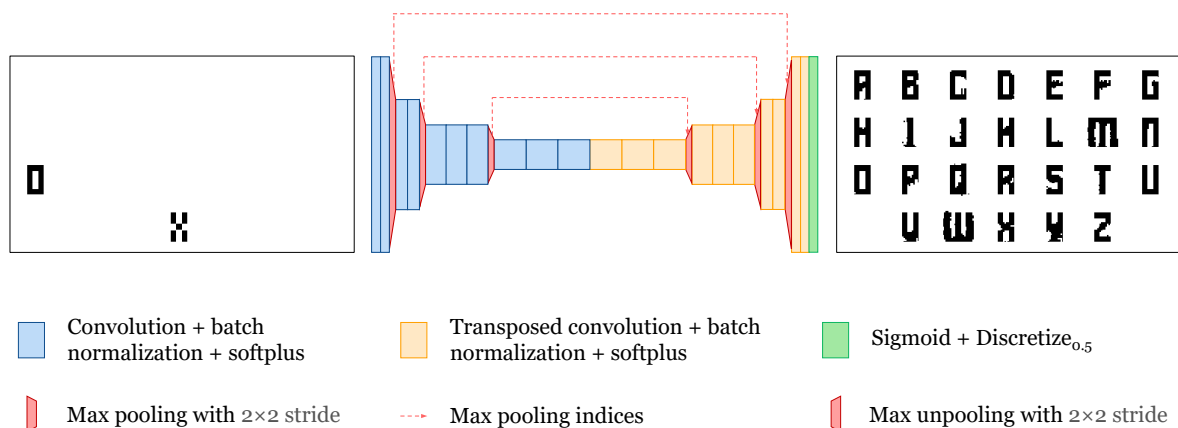
simple method we do not take into account at all the letters from the same font that are not hidden, which usually have rich information that can help us generate the ones that are missing.

#### 4.5.2 Convolutional Autoencoder Baseline

We can build much more sophisticated models that take into account information about letters of the same font that are visible to the algorithm and generate images for the hidden letters accordingly, while also imposing an appropriate inductive bias for the type of data that we are dealing with.

One way to do this is to build an autoencoder with convolutional neural networks, in which the task is not solely to reconstruct the input images, but also to generate the ones that are not visible in the original input. We might have chosen various different types of architectures here, but ultimately we decided that the most important requirement was that we could build a model that was structurally similar to the energy networks, so that the comparison between them was more meaningful. To that end, we have decided to base both our autoencoder baseline and the SPEN-based algorithm in the classic VGG16 architecture, although employing the convolutional blocks described in the previous section. To obtain a baseline capable of generating the missing letters, we may adapt the SegNet architecture [24], which has a convolutional encoder-decoder structure.

In the original design, SegNet performed pixel-wise semantic segmentation, in which each pixel is assigned to a corresponding class. The model receives as input a whole image (with 3 color channels), passes it through an encoder with multiple convolutional, batch normalization and ReLU layers, intercalating with downsampling procedures that use max pool operations. After the encoder part the features are reduced to a much lower-dimensional space. In the decoder part, additional convolutional, batch



**Figure 4.6:** Architecture of the convolutional autoencoder baseline. It is similar to SegNet [24], but uses the convolutional blocks defined in Section 4.4, besides other modifications. Note that the indices of the maxima obtained during the max pooling downsampling layer are saved to be used by the max unpooling layer in the upsampling process of the decoder part.

normalization, and ReLU layers are applied to the encoder representation, together with upsampling operations (which we will describe shortly), resulting in images with the same size of the original. The decoder does not output three channels in the end of the process, instead it outputs a number of output channels equal to the number of classes in the segmentation task and, by applying a softmax function across these channels, we model the probability that a certain pixel belongs to a certain class.

It is important to describe a feature that is particular to the SegNet architecture that we adopted, which relates to how upsampling is done. In this architecture, the indices of the max pooling downsampling operations in the encoder part are cached for later use by an upsampling operation. This procedure is located symmetrically with respect to the middle of the autoencoder and has, in terms of the spatial size of input and output feature maps, the opposite effect to the downsampling procedure. The max pooling indices are the indices that indicate which element, between the four that are affected by one max pool kernel, corresponds to the maximum value. In this upsampling operation, those indices define which elements of the output are going to be equal to the input, and which ones are set to zero. In effect, by using those indices, we remember some of the structural information of the original feature maps [72], which helps the reconstruction work. These connections only slightly complicate the architecture, but we noticed that they do help performance.

Notice that the objective of our task is not to perform semantic segmentation, neither does it involve reconstructing an entire font after passing it through an informational bottleneck. Differently from a standard autoencoder or from SegNet, the objective is still to reconstruct the whole 26 letters of the font, but having now as input the masked version of the font, similarly to Context Encoders [62]. To that end, we use an autoencoder architecture with 26 output channels, i.e., the same number as the input

**Table 4.1:** Parameters of the architecture of the convolutional autoencoder baseline. Each block is composed of multiple convolutional layers, the kernel side size is kept constant across an entire block, and between two successive blocks a downsampling or upsampling operation is done, except between the last encoder block and the first decoder one. For more information on the constitution of the convolutional blocks see Section 4.4. Note that, in terms of the parameters listed in this table, the encoder part is symmetrical with respect to the decoder one.

Convolutional blocks (with batch normalization and softplus activations)		
Nr. layers	Kernel side size	Input×output channels
2	5	26×40
2	5	40×60
3	3	60×80
3	3	80×100
Transposed convolutional blocks (with batch normalization and softplus activations)		
Nr. layers	Kernel side size	Input×output channels
3	3	100×80
3	3	80×60
2	5	60×40
2	5	40×26

channels and corresponding to the number of letters, and a Sigmoid output layer defining the grayness of the pixel, which can be discretized according to Equation 4.1 to generate the final predicted images. During training, we also have the complete, unmasked, version of the font. Therefore, we can use an autoencoder with an architecture similar to a SegNet, applied to the masked input, and in the end compute the BCE loss between the complete font and the generated one, having then all the ingredients to perform backpropagation as usual.

There were some other changes that we have made to the SegNet architecture, besides changing the input/output channels. We did not employ the usual convolutional layers in the decoder part of the network, but instead transposed convolutional ones, simply because those performed better in our dataset. Moreover, the original SegNet architecture used, between the encoder and decoder parts of the network, a max pooling operation right before an unpooling operation. We did not understand the reasoning behind this, as it seems we are losing information without an evident purpose. By removing those operations the network did perform better, so we have kept that change.

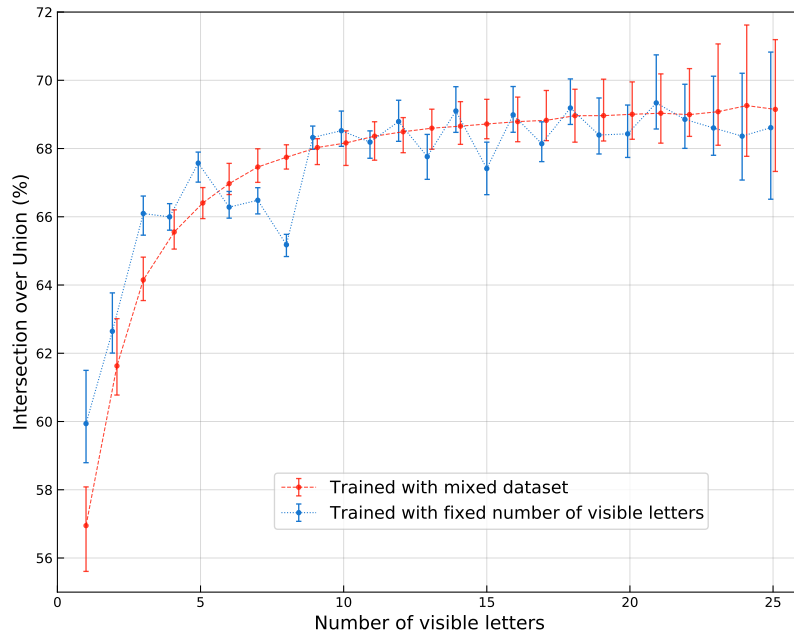
The convolutional blocks described in the previous section were used for this baseline. We used batch normalization in each convolutional block, which did make a significant difference in the results.

Moreover, we used the softplus function with  $\beta = 1$  as the non-linearity instead of the more commonly used ReLU because, after testing both, the results were almost indistinguishable, and this way we have a more direct comparison with the energy networks which also use a softplus function with the same value for  $\beta$ .

We searched for the architecture that scored as much as possible in the validation set, varying the number of feature maps, number of layers per convolutional block, kernel sizes, number of feature maps, and the learning rate for the Adam optimizer. During the training we employed early stopping, and made sure that there was a difference of at least 8% between the maximum of all the observed IoU's in the training set and the maximum of all the observed IoU's in the validation set. This indicated that we were starting to overfit significantly to the training set, and the validation IoU score was decreasing at the same time that the training one was increasing, the wide margin suggesting that the model probably would not start performing better in the validation test anymore. In this process we adopted a safer and wider gap than we selected during training with the energy networks (only 5%), since generally the baseline trained significantly faster and we could afford to wait.

The best scoring architecture of this convolutional baseline, in the validation set, is depicted in Figure 4.6 and it is described in detail in Table 4.1. Both the encoder part (the left half in the figure) and the decoder part (the right half) consist of 10 convolutional layers each, organized in 4 convolutional blocks. In our energy-based model, we take the decoder of this autoencoder baseline as the basis for the first part of the energy function.

In Figure 4.7 we depict the results corresponding to two different procedures for training and testing the baseline, with the architecture given in Table 4.1. In the first procedure, the model was trained in *mixed* input mode (in which all possible configurations of visible and hidden letters may happen), but was then evaluated over the test dataset in the mode in which the number of visible letters is fixed. In the second procedure, the baseline was trained in fixed input mode, specifically using the same number of visible letters as the test dataset in which it would later be tested. As can be observed in Figure 4.7, training one single model in mixed input mode was quite advantageous, since this model proved to be well-performing across test datasets with different numbers of visible letters in fixed input mode. The values were also quite stable across different numbers of visible letters. In the cases with few known letters, training specifically for those situations yielded better results. In contrast, when each model was both trained and tested in datasets with the same number of visible letters, the results were more unstable across `n_visible_letters`. Some variations could be naturally expected but the drop for when `n_visible_letters` is in the range 6-8 remains unexplained.



**Figure 4.7:** The IoU score of the baseline convolutional autoencoder model evaluated in fixed input mode on the test dataset, for different numbers of visible letters. Each dot shows the median of 100 evaluations with different masks and the same number of visible letters. The error bars exhibit the full range of IoU scores obtained with those 100 evaluations. They are shown slightly apart in those cases where both bars would overlap. The red data points correspond to only one model, trained in mixed input mode (in which all possible configurations of visible and hidden letters may happen), but evaluated on a test set with a fixed number of visible letters. The blue data points encompass 25 different models, each one trained with a fixed number of visible letters, corresponding to the number of visible letters of the test set in which each model was evaluated.

We note that until now we have been computing the loss between the visible and hidden letters  $(x, y)$ , and a reconstruction of  $(x, y)$  that is based solely on the visible letters  $x$ . Therefore, it seemed plausible to think that, by modifying the BCE loss so that it enhances the importance given to the generation of those letters  $y$  that are hidden in the input, we could train our model to perform better at generating those letters, at the expense perhaps of being moderately worse at reconstructing the letters  $x$  that are in fact visible to the algorithm. For the same reasons, we also tested a variant in which the loss was computed only between the generated letters and their ground truth, and no weight was given to the reconstruction of the visible letters at all. Neither of these versions seemed to have a noteworthy effect however and, to keep everything more similar to the subsequent energy networks, we decided not to use those losses in the final baseline.

## 4.6 Deep Energy Networks

In the font dataset, a font style is related to the dependencies that exist between components of different letters, and we may not in fact know a priori which letters are going to be visible at test time. To shape an energy function that does not distinguish in its structure between visible and invisible letters, and to try to model directly (with deep neural networks) dependencies among any of the components of both seems, a priori, to be a good approach to solve this problem.

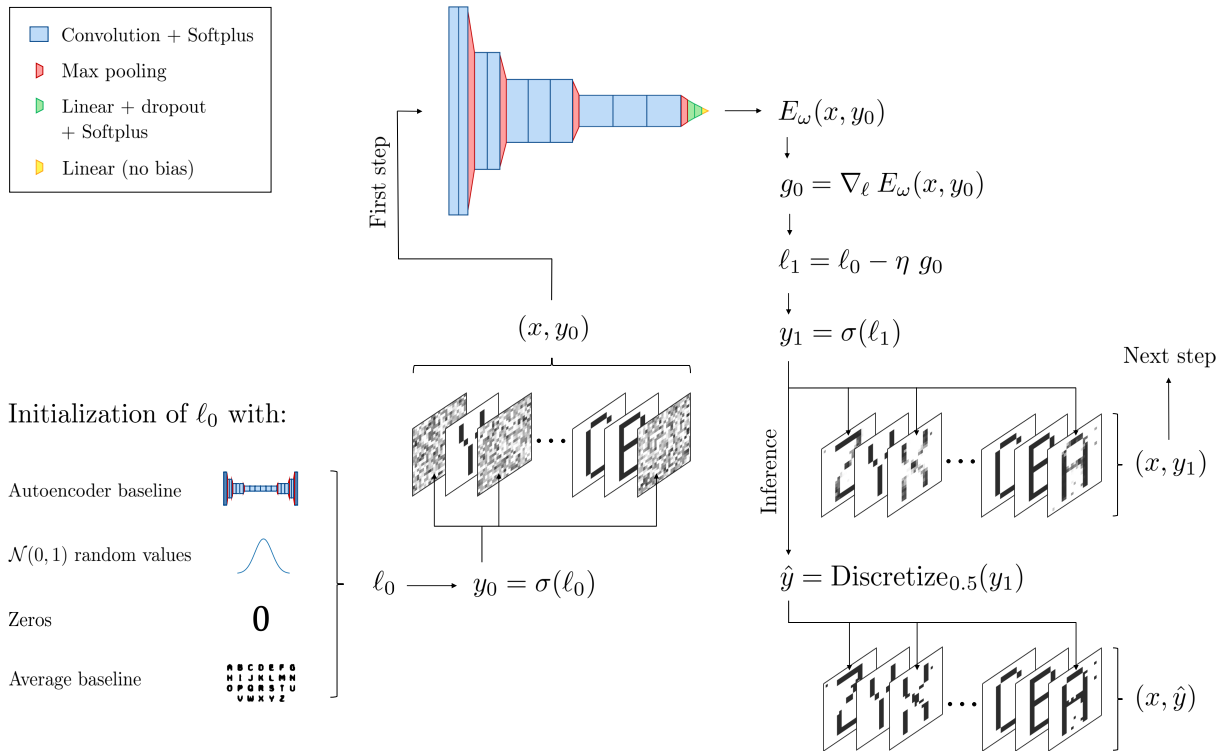
The algorithm that we propose, which is based in SPENs, is fundamentally different from the autoencoder approach since here we are not trying to reconstruct the whole input, but trying to predict missing values based on observed ones. Therefore, we only take into account in the BCE loss the generated estimates to the logits of the missing letters. We recall however that, as was described in Chapter 3, the loss function takes into account all the intermediate logits of  $y$  that are obtained as a result of the  $T$  steps of gradient based inference, and weighs their contribution according to Equation 3.6.

As we have explained in Section 3.2, in our problem,  $(x, y)$  defines a font in which some of the letters  $x$  are visible and some others  $y$  are hidden. Moreover, at each step of the inference algorithm, a forward pass of the energy network must be computed for  $(x, y_t)$ . In the implementation this corresponds to assigning the intermediate soft estimates for the missing letters  $y_t$  to the appropriate places within the font tensor. In Figure 4.8, we depict the entire process for data completion in the font dataset. This includes the assignment process we have just described, as well as the architecture of the energy network that was used (described in Subsection 4.6.1), together with the available initialization routines (described in Subsection 4.6.5). Furthermore, the figure includes the first step of gradient based inference, the preparation of the data for the following step if applicable and, in alternative, the discretizing step that concludes the inference process. As we have explained in Section 2.3.2, if we have a twice differentiable energy function and all other operations are also differentiable, the whole process that we see in Figure 4.8 is itself differentiable with respect to the parameters of the energy function  $\omega$ . Moreover, as we have explained in Section 3.2, we can implement the training entirely in PyTorch with native tools.

In the following, we describe the implementation and results of SPENs for data completion (introduced in Chapter 3) for the problem of completing missing letters in the fonts dataset. We start by describing the energy network architecture that we used for this purpose.

### 4.6.1 Architecture of the Energy Network

As we have explained in Subsection 2.3.2, we may choose for the energy network any twice differentiable energy function with an uniform architecture across different letters. Note that, in particular, such



**Figure 4.8:** SPEN algorithm adapted for performing data completion in the font dataset: architecture of the energy network, initialization routines, first gradient-based inference step, and generation of the missing letters. As an example, we start the initial logits  $\ell_0$  by random initialization and, after a sigmoid transformation, we set these random soft estimates  $y_0$  in their appropriate places next to the visible letters  $x$ . We repeat that procedure each step of the algorithm, before the computation of the energy function. Gradient based optimization of the inference process is performed in logit-space. The whole process resulting in a soft prediction  $y_T$  is differentiable with respect to the parameters of the energy function, since that is also the case for every step along the way. Therefore, the gradient of a loss function may be computed and propagated backwards into the parameters of the energy function.

architecture does not distinguish whether a letter is visible or not. As in the last section, we are going to make use of convolutional neural networks, but only in the first part of the energy function. After that convolutional part, the feature tensor corresponding to a single observation is flattened, and passed through a fully connected part which allow all of the features that were learned by the convolutional part to be combined to produce the final score, corresponding to the energy  $E$ .

To be more specific, we use the encoder of the previous autoencoder baseline as the convolutional part (again with filters that extend depthwise), with the difference that batch normalization was not added. We decided not to include batch normalization, since initial experiments showed that the model had problem in learning the reconstruction when considering this feature. However, batch normalization may be something to analyze in more detail in the future. The 4 convolutional blocks with a total of 10

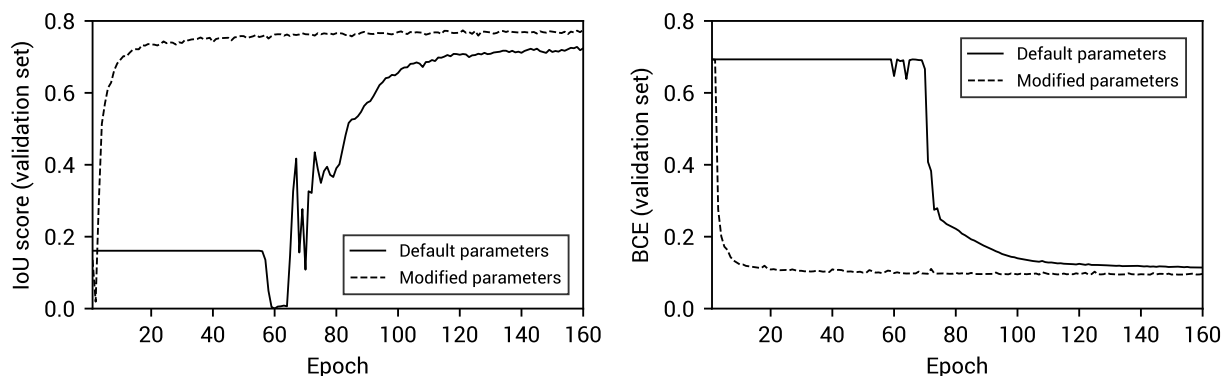
**Table 4.2:** Parameters of the architecture of the energy function. The convolutional part is similar to the encoder part of the autoencoder baseline, but with added capacity that still resulted in better generalization ability.

Convolutional blocks (with softplus activations)		
Nr. layers	Kernel side size	Input×output channels
2	5	26×120
2	5	120×140
3	3	150×180
3	3	180×220
Fully connected part (with dropout and softplus activations)		
Dropout probability	Input×output channels	
0.3	880×200	
0.5	200×80	
–	80×1	

convolutional layers are structured in such a way as to increase the number of feature maps channels at the same time that the spatial size reduces, as was the case with the autoencoder baseline. At the end of the convolutional part, the feature maps are now reduced to have the spatial size (i.e., width and height) equal to 2. The data is then flattened and fed into the fully connected part of the energy network. This part consists of two linear layers that use dropout and the softplus activation functions with  $\beta = 1$ , and a final linear layer without dropout or a softplus activation (so that the energy may take any value, including a negative one). Furthermore, as we have explained in Section 3.2, this last linear layer does not contain a bias term.

From the moment that we had a stable implementation, and in spite of the problem that we describe in the next subsection, identifying an architecture that worked well was not problematic. Contrarily to what happened with the autoencoder baselines, we could expand quite substantially the capacity of the convolutional part (by increasing the number of feature maps of the convolutional part) and still improve the performance. The whole energy network architecture that we arrived at, which was used in most experiments (except when we change the dropout probabilities), is described in Table 4.2.

Concerning the two learning rates that are used in SPENs, the only issue with choosing the right ones was the initial plateau in the scores (described in the previous subsection), which could take quite long to disappear during training and which presented some difficulties to the discovery of combinations of learning rates that would work well. We observed that adjusting properly the inner learning rate was



**Figure 4.9:** Comparison of the IoU score and BCE in the validation set of a model trained with the default PyTorch parameters, and another in which some of the parameters of the model were multiplied by a factor of two. The problem in which both scores do not improve for almost 80 epochs does not occur in the latter case.

more important, and that the training process was less sensitive with respect to the learning rate of the Adam optimizer. Moreover, the choice of the learning rates was quite dependent on the number of iterations in the inference loop.

## 4.6.2 Training

Similarly to the training of the convolutional autoencoder baseline (see Subsection 4.5.2), we used early stopping by enforcing a minimum gap, this time of 5%, between the maximum of all the observed IoU scores in the training set and the maximum of all the observed IoU's in the validation set. We observed significantly longer running times than with the convolutional baseline, since attaining such a margin was sometimes rather difficult. This happened specially when the number of visible letters was very low and the number of iterations in the inference routine was one. Under those circumstances, the training routine of the energy network could take very long, as much as 8 hours. As we will see later, those were not the configurations for which the better results were attained and, if we chose the number of iterations of the inference loop well, we could train quite faster.

A curious issue emerged in the beginning of training during most our experiments with font completion. As Figure 4.9 shows, both the training and validation BCE and IoU scores hardly moved in any noticeable way around a fixed value, sometimes during more than 100 epochs. A sudden drop in the IoU scores announced that learning would effectively start occurring soon, as after a few more epochs the IoU would start raising and the BCE dropping, as would be expected during normal learning. This problem complicated the hyperparameter optimization stage, because often we needed to wait for a few hundred of epochs before we were confident that we could reject a combination of hyperparameters that

did not enable learning to happen.

We decided to investigate this question and, suspecting an initialization issue (we were using the default PyTorch initial values for the convolutional kernels), we decided to inspect how the parameters of the neural network changed during training. We noticed that the weights and biases of the first two convolutional blocks did change quite significantly, their distribution spreading out approximately by a factor of two as the training progressed. Since it seemed that most of the work in these initial phases was done to change the distribution of those parameters so that learning could occur, we decided to simply change the distribution of the parameters of this convolutional block ourselves, multiplying them by a factor of two. With this change we did succeed in eliminating the plateau – the IoU score now started lower than before, almost at zero, but immediately started improving without a stagnation phase.

There were nonetheless still some problems in the beginning of training for different hyperparameter settings, even after making these changes. The IoU would stay near zero sometimes for a while, and only after some epochs would it start climbing up. As this procedure was not very reliable and ultimately did not change much the final results, we did not apply it to any of the studies that we present next.

### 4.6.3 Number of Inference Iterations

Belanger et al. [11] reported that using a small value for the number of gradient-based inference iterations  $T$  (such as  $T = 3$ ) worked well with their experiments. We also decided to test the case where only one gradient step is taken ( $T = 1$ ) and, surprisingly, we only obtained slightly worse results (see Table 4.3). However, the training of the energy networks when  $T = 1$  and when `n_visible_letters` was also small took considerably longer than when  $T = 3$ .

**Table 4.3:** IoU scores (in percentage, average  $\pm 2 \times$  standard deviation) for different number of inference iterations and different input modes – with mixed mode, in which every configuration of visible and hidden letters is possible, and fixed mode, with different numbers of visible letters  $N$ .

Nr. of inference iterations $T$	Mixed	Fixed		
		$N = 1$	$N = 11$	$N = 21$
$T = 1$	$69.5 \pm 1.7$	$60.7 \pm 1.2$	$72.2 \pm 0.4$	$74.6 \pm 1.0$
$T = 3$	$70.2 \pm 1.6$	$59.6 \pm 1.7$	$73.1 \pm 0.4$	$74.7 \pm 0.9$

## 4.6.4 Dropout

Belanger referred in his PhD dissertation [57] that dropout could not be used in the main part of a SPEN, although he still uses it in the feature network. He presents the argument that we should not use it since a new set of zeros is sampled randomly each time we call the energy function, which would cause the energy function to be different in each inference step.

However, as we have seen in Subsection 4.6.3, we do in fact test the case where the number of inference iterations  $T$  is simply one (and therefore we only compute the energy function once), in which case this problem is not present. Therefore we did use dropout, and we decided to also try to test it with other values of  $T$ , even though we cannot guarantee that the same random pattern is chosen for each call of the energy function.

**Table 4.4:** IoU scores (in percentage, average  $\pm 2 \times$  standard deviation) for different dropout probabilities  $p_1$  and  $p_2$  and different input modes – with mixed mode, in which every configuration of visible and hidden letters is possible, and fixed mode, with different numbers of visible letters  $N$ . Here we used the number of inference iterations  $T = 3$ .

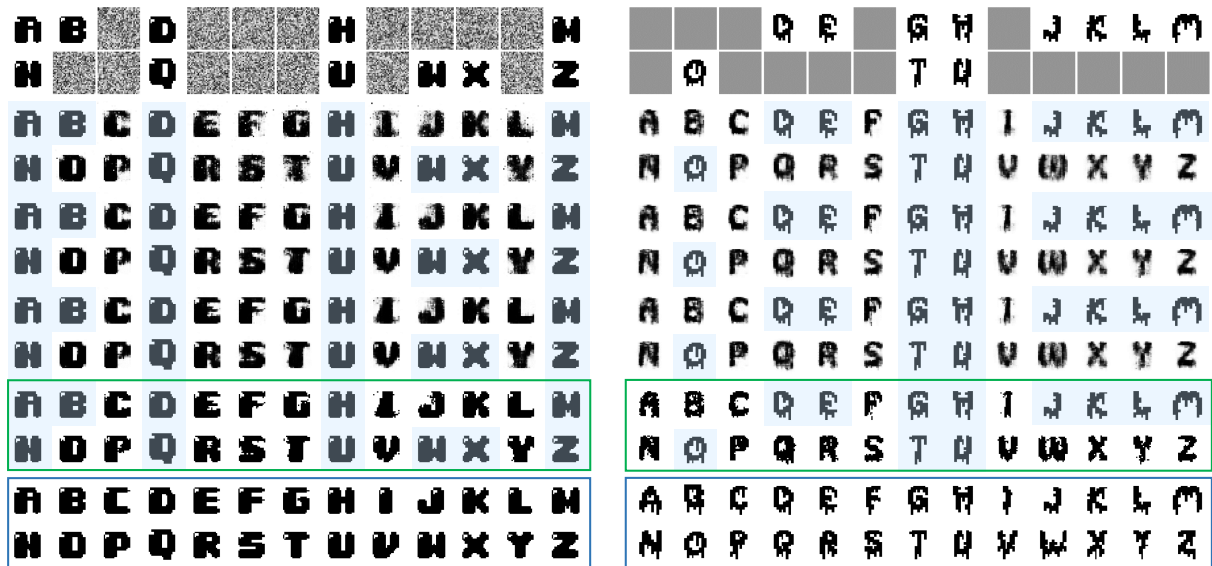
Dropout probabilities	Mixed	Fixed		
		$N = 1$	$N = 11$	$N = 21$
$p_1 = 0.3, p_2 = 0.5$	$70.0 \pm 1.7$	$59.6 \pm 1.7$	$73.1 \pm 0.4$	$74.7 \pm 0.9$
$p_1 = 0.2, p_2 = 0.2$	$70.3 \pm 1.6$	$59.0 \pm 1.3$	$72.3 \pm 0.4$	$75.0 \pm 1.0$
No dropout	$69.8 \pm 1.6$	$60.7 \pm 1.3$	$72.7 \pm 0.4$	$75.1 \pm 0.9$

Initially, we chose the dropout probabilities  $p_1 = 0.3$  and  $p_2 = 0.5$ , following the recommendation of using smaller values of the probability of dropping one of the activations for layers closer to the inputs [17]. We have subsequently searched for other combinations of these values, including not using dropout at all. Even though the different combinations that we have tested did not have an important impact in the results (Table 4.4), we did use the initial choice of dropout in most other experiments.

## 4.6.5 Initialization Routines

We also analyzed the impact of the different initialization procedures that we have previously described in Section 3.2. For this analysis, we set the number of iterations of the inference process to  $T = 3$  (which was a choice with which we had generally good results, and which allowed the training time to be shorter), and used learning rates adapted to this choice.

Recall that, in both fixed and mixed modes, we receive a font with some of the letters masked out, i.e., filled with zeros. Each initialization routine below provides different values to fill in the places of the



**Figure 4.10:** SPEN model trained with random initialization: initialization of the inference process, three intermediate gradient-based inference steps, final guess for the missing letters (highlighted in green), and ground truth font letters (highlighter in blue). In the left-hand side, the initial logits are initialized with random  $\mathcal{N}(0, 1)$  distributed values, and in the right-hand side they are initialized with zeros (which correspond to a gray 0.5 value in the constrained space). The ground truth letters are highlighted in light blue (when they are not signaled in another way). Note the increasingly better soft intermediate guesses in each inference step.

masked letters, substituting those zeros. Each initialization routine provides logits that are transformed with the sigmoid function and positioned in the correct place together with the visible letters  $x$ , substituting the masked letters with different values. This version of the input, with the changed values of the hidden letters, is what we provide to the energy function. Note also that we do not enforce that those values are either 0 or 1, even though the original images themselves are like this – the values will instead belong to the interval  $[0, 1]$ .

The four initialization procedures of the logits of the hidden letters  $\ell_0$  used in the fonts completion problem are reviewed next:

- **Initialization with the autoencoder baseline:** This is the most similar initialization to the one that Belanger et al. used [11]. In it, we simply have the autoencoder generate the initial logits  $\ell_0$ .
- **Initialization with average baseline:** We initialize  $\ell_0$  with a modified version of the average baseline. The average of all the fonts has useful information on the structure of the letters of a font (see Figure 4.5), which may be a good starting point for doing inference with our models. One must pay attention however that it does not work well to have all the initial values predict the same class. That is the case since all the values of the average baseline are equal or greater than zero (they belong to  $[0, 1]$ ) and, in fact, we did not obtain good results this way. To solve this issue,

**Table 4.5:** IoU scores (in percentage, average  $\pm 2 \times$  standard deviation) for different initialization routines and different input modes – with mixed mode, in which every configuration of visible and hidden letters is possible, and fixed mode, with different numbers of visible letters  $N$ . Here we used the number of inference iterations  $T = 1$ .

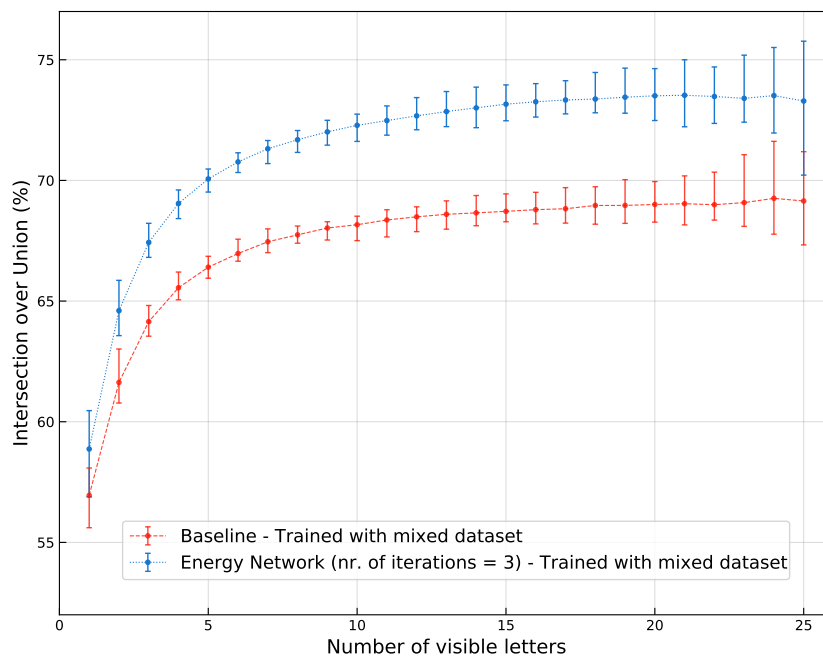
Initialization method	Mixed	Fixed		
		$N = 1$	$N = 11$	$N = 21$
Autoencoder Baseline	$70.4 \pm 1.6$	$60.6 \pm 1.2$	$72.9 \pm 0.4$	$74.7 \pm 1.0$
Average Baseline	$69.7 \pm 1.6$	$60.2 \pm 1.0$	$72.6 \pm 0.4$	$74.6 \pm 1.0$
Initialization to Zeros	$69.5 \pm 1.7$	$60.7 \pm 1.2$	$72.2 \pm 0.4$	$74.6 \pm 1.0$
Random Initialization	$69.1 \pm 2.0$	$57.2 \pm 1.2$	$72.3 \pm 0.4$	$74.3 \pm 0.9$
Random Initialization (no dropout)	$68.4 \pm 2.0$	$59.5 \pm 1.1$	$72.3 \pm 0.4$	$74.2 \pm 0.9$

we decided to simply subtract 0.5 to each component of the data to have those values distributed between  $[-0.5, 0.5]$ .

- **Initialization to zeros:** All logits  $\ell_0$  are initialized to 0, which corresponds to changing all the masked input values to 0.5 (after passing  $\ell_0$  through the Sigmoid function).
- **Random initialization** The logits  $\ell_0$  are initialized by sampling pseudorandom values from the  $\mathcal{N}(0, 1)$  distribution at training time. A version of this initialization without dropout was also tested.

As we have described in Subsection 3.2, when using random initialization at test time, we found that it was beneficial to start the logits at zero, instead of randomly. In Figure 4.10, starting from random and zeros, we show the soft estimates in the intermediate inference steps and the final estimate for the missing letters.

Compared with the autoencoder baseline, the SPEN algorithm did attain significantly better results. As can be seen in Table 4.5, varying the initialization procedure of the energy-based algorithms did not have a very significant impact in the results. Using the autoencoder baseline was not necessary to obtain good performance. However, training time could be quite longer if random initialization was applied.



**Figure 4.11:** The IoU score of the energy network algorithm and the baseline convolutional autoencoder model evaluated in fixed input mode on the test dataset, for different numbers of visible letters. Each dot shows the median of 100 evaluations with different masks and the same number of visible letters. The error bars exhibit the full range of IoU scores obtained with those 100 evaluations. Each of the models was trained in mixed input mode (in which all possible configurations of visible and hidden letters may happen), but evaluated with fixed input mode on the complete range of `n_visible_letters`.

## 4.7 Discussion of the Experimental Results

As demonstrated by Table 4.6, the best performing method was our SPEN-based algorithm. That was the case for any initialization procedure that we have chosen. We have observed that the results that we have obtained with the energy networks are not very dependent on the choices of the hyperparameters of the previous section, and substantially better than those. By contrast, results are quite sensitive to the choice of learning rates, in particular the inner rate of the inference routine.

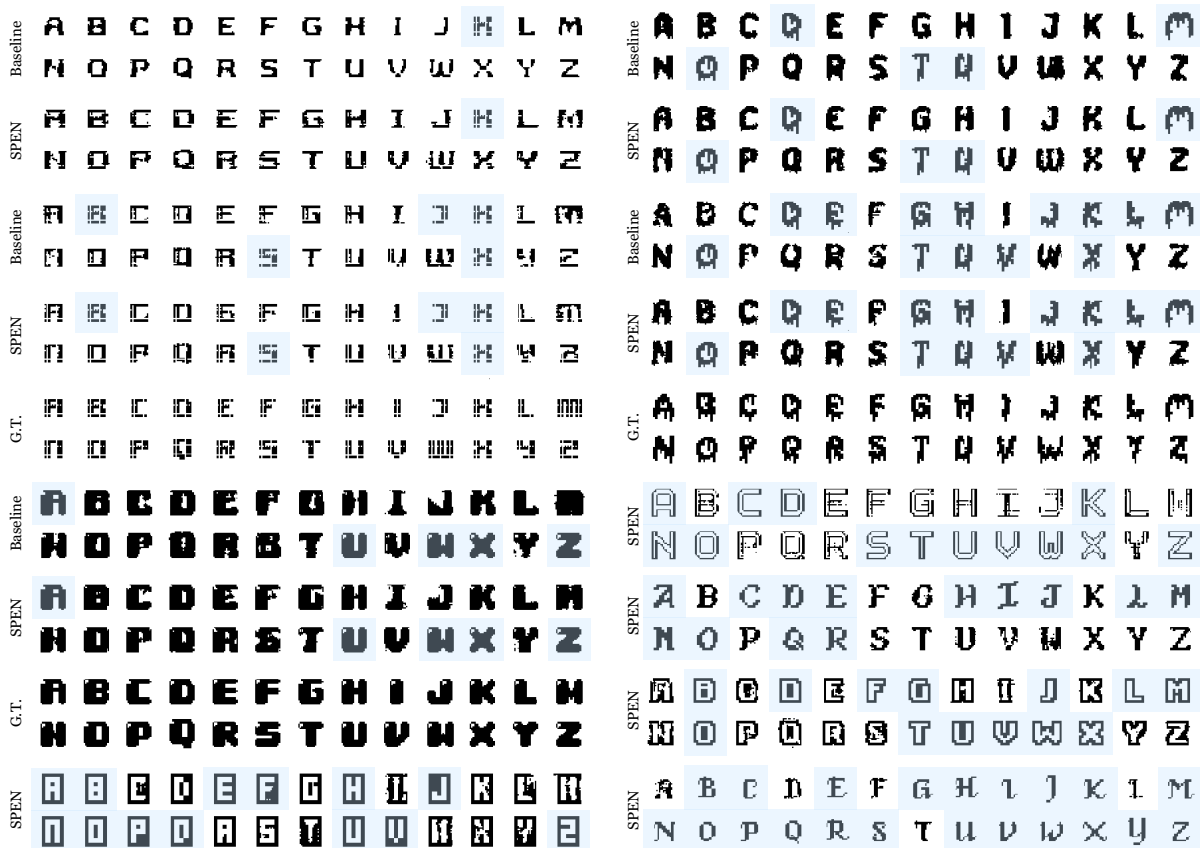
**Table 4.6:** Comparison of the IoU scores (in percentage, average  $\pm 2 \times$  standard deviation) for the different tested algorithms and different input modes (when applicable) – with mixed mode, in which every configuration of visible and hidden letters is possible, and fixed mode, with different numbers of visible letters  $N$ . Only one score is given for the average baseline, whose performance is not dependent on the input mode.

Method	Mixed	Fixed		
		$N = 1$	$N = 11$	$N = 21$
Energy Network (Init. to Zeros)	$69.5 \pm 1.7$	$60.7 \pm 1.2$	$72.2 \pm 0.4$	$74.6 \pm 1.0$
Autoencoder Baseline	$66.9 \pm 1.6$	$60.0 \pm 1.1$	$68.6 \pm 0.4$	$69.4 \pm 0.8$
Average Baseline		36.3		

The fact that using a number of inference iterations  $T = 1$  works well is significant, since with this choice we save memory and computing resources, which might make it the only choice for some applications. It must be noted however that the time of convergence during training is shorter for higher numbers of iterations. It is also the case that, in general, the energy networks take quite longer to train than the baseline autoencoder.

As we can see in Figure 4.11, the energy network trained in mixed input mode generalized better when tested in the fixed input mode case for the entire range of `n_visible_letters` than the autoencoder baseline. We also observed separately that, in most cases, it seems to be worthwhile to train our models in mixed input mode instead of training specifically for a certain number of visible letters, unless one wishes to have slightly higher performance in the case where few letters are known. Note also that both the energy network and the baseline performed better than the average baseline by a large margin, as would be expected.

The difference in quality of the resulting images and the plausibility of the generated style for the missing letters is also significant. As can be seen in Figure 4.12, when the number of visible letters is low neither the autoencoder nor the SPEN can transfer well the style of the visible letters to the hidden letters. Sometimes, however, even with a low number of visible letters, SPENs start yielding more plausible results, even though the quality is still not very good. With higher numbers of visible letters,



**Figure 4.12:** Qualitative results from SPENs (marked with “SPEN”), from the autoencoder baseline (marked with “Baseline”), and ground truth values (marked with “G.T.”) for comparison. The letters that were visible are marked light blue and juxtaposed next to the generated ones, even in the case of the autoencoder baseline. Some of the styles can be generated with only a small number of visible letters, while others seem to be more difficult to generate. In most cases that are not trivial, the results from SPENs seem more reasonable than the ones provided by the autoencoder. Some details in more complex fonts could be generated when the number of visible letters is higher. Note that what one might consider a more complex style was often more difficult for the SPEN to generate, but that was not always the case.

SPENs often could adequately transfer more subtle styles to unknown letters.

There are plenty of examples where the results were more similar, and we include some of them, but there was a clear trend in which, if one of the generated fonts was clearly better than the other, most often that was the result of a SPEN and not of the baseline. Still, SPENs had difficulty transferring some styles. This dataset is not very extensive and, by using richer and larger datasets, we could investigate further the capabilities of SPENs for image generation, as we will discuss in Section 5.2.



# 5

## Conclusions

### Contents

---

5.1	Conclusions	60
5.2	Future Work	60

---

## 5.1 Conclusions

In this dissertation, we reviewed energy-based models and Structured Prediction Energy Networks, and described how they can be applied to a class of data completion problems. The proposed modification to the SPEN framework allows for architectures that are already used in discriminative tasks to be integrated in a data completion generative model, and it is capable of filling in an arbitrary subset of missing data. Among other changes, this entailed modifying the form of the energy networks that were considered, in order to directly model dependencies between any of the components of the data, regardless of them being part of the input or the output.

The experiments with our font completion task showed that, in almost any way we chose to modify the algorithm, the energy networks performed quite better, both quantitatively as well as qualitatively, than a comparable feed-forward autoencoder baseline. That was still the case when we tested the performance of the algorithm without the help of an initial estimate, which suggests that the energy network alone could encode well the dependencies between components of the data.

We also verified that using only one iteration in gradient-based inference proved to work almost as well as using more. This suggests that, by using SPENs, computational resources and memory usage should not be too different from comparable feed-forward approaches. In addition, the various adaptations described in Section 3.2, including changing the value for the  $\beta$  parameter that was used in previous studies, and the introduction of the `Compress` function, helped to train SPENs and may be suitable for further experimentation with these models.

## 5.2 Future Work

More work can be done to assess the performance of generation for higher dimensionality data. Generating large color images depicting different scenes (instead of black and white fonts) with random masks is a more difficult problem that may also require more attention to the choice of the loss function. However, the method that we adopted here can be directly translated into the realm of general image inpainting. Instead of having 26 input channels, we would have only 3 color channels. An energy function could then be computed for the whole image, and estimates for the missing pixels can be iteratively generated, as we have done in this dissertation for the missing letters. This presents a natural way to deal with randomly shaped masks.

Furthermore, we could leverage the representations that we obtain by these energy functions and verify their generalization properties by, for example, attaching a multi-layer perceptron to the end of the convolutional part and test the accuracy in classification tasks.

In the future, it may also be interesting to modify the energy function architecture to be able to accept variable sized inputs and, by including recurrence as an adaptation for tasks with natural temporal dependencies, it may be possible to use this framework in tasks such as natural language generation.



# Bibliography

- [1] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and improving the image quality of StyleGAN,” 2020. [Online]. Available: <https://arxiv.org/abs/1912.04958>
- [2] A. Razavi, A. van den Oord, and O. Vinyals, “Generating diverse high-fidelity images with VQ-VAE-2,” *CoRR*, vol. abs/1906.00446, 2019. [Online]. Available: <http://arxiv.org/abs/1906.00446>
- [3] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz, “MoCoGAN: Decomposing motion and content for video generation,” 2017. [Online]. Available: <http://arxiv.org/abs/1707.04993>
- [4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019.
- [5] B. Taskar, C. Guestrin, and D. Koller, “Max-Margin Markov Networks,” in *Proceedings of the 16th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2003, pp. 25–32. [Online]. Available: <https://proceedings.neurips.cc/paper/2003/hash/878d5691c824ee2aaf770f7d36c151d6-Abstract.html>
- [6] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data,” in *Proceedings of the 18th International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.
- [7] D. Belanger and A. McCallum, “Structured Prediction Energy Networks,” in *Proceedings of the 33rd International Conference on Machine Learning*, M.-F. Balcan and K. Q. Weinberger, Eds., vol. 48, 2016, pp. 983–992. [Online]. Available: <http://proceedings.mlr.press/v48/belanger16.html>
- [8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style,

- high-performance deep learning library,” in *Advances in Neural Information Processing Systems* 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>
- [10] R. Frostig, M. Johnson, and C. Leary, “Compiling machine learning programs via high-level tracing,” 2018. [Online]. Available: <http://www.sysml.cc/doc/2018/146.pdf>
- [11] D. Belanger, B. Yang, and A. McCallum, “End-to-end learning for Structured Prediction Energy Networks,” in *Proceedings of the 34th International Conference on Machine Learning, {ICML} 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 429–439. [Online]. Available: <http://proceedings.mlr.press/v70/belanger17a.html>
- [12] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, “Conditional Random Fields as Recurrent Neural Networks,” in *Proceeding of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1529–1537.
- [13] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. PMLR, 2011. [Online]. Available: <http://proceedings.mlr.press/v15/glorot11a.html>
- [14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: <http://www.nature.com/articles/323533a0>
- [15] H. Robbins and S. Monro, “A stochastic approximation method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. [Online]. Available: <http://www.jstor.org/stable/2236626>
- [16] L. Bottou, “Stochastic gradient learning in neural networks,” in *Proceedings of Neuro-Nîmes*, 1991. [Online]. Available: <https://leon.bottou.org/publications/pdf/nimes-1991.pdf>

- [17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>
- [18] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: <http://proceedings.mlr.press/v37/ioffe15.html>
- [19] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org/>
- [20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>
- [21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [23] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: <http://arxiv.org/abs/1505.04597>
- [24] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," 2016. [Online]. Available: <http://arxiv.org/abs/1511.00561>
- [25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," 2018. [Online]. Available: <http://arxiv.org/abs/1706.08500>

- [26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," 2014. [Online]. Available: <https://arxiv.org/abs/1406.2661>
- [27] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016. [Online]. Available: <http://arxiv.org/abs/1511.06434>
- [28] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," 2019. [Online]. Available: <http://arxiv.org/abs/1809.11096>
- [29] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," 2018. [Online]. Available: <http://arxiv.org/abs/1710.10196>
- [30] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019. [Online]. Available: <http://arxiv.org/abs/1812.04948>
- [31] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2014. [Online]. Available: <http://arxiv.org/abs/1312.6114>
- [32] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," 2014. [Online]. Available: <http://arxiv.org/abs/1401.4082>
- [33] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf>
- [34] A. V. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1747–1756. [Online]. Available: <http://proceedings.mlr.press/v48/oord16.html>
- [35] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves, "Conditional image generation with PixelCNN decoders," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon,

- and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: <https://proceedings.neurips.cc/paper/2016/file/b1301141feffabac455e1f90a7de2054-Paper.pdf>
- [36] W. Grathwohl, K. Wang, J. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky, “Your classifier is secretly an energy based model and you should treat it like one,” *CoRR*, vol. abs/1912.03263, 2019. [Online]. Available: <http://arxiv.org/abs/1912.03263>
- [37] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, “A tutorial on energy-based learning.” [Online]. Available: <http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf>
- [38] Y. Du and I. Mordatch, “Implicit generation and generalization in energy-based models,” 2020. [Online]. Available: <http://arxiv.org/abs/1903.08689>
- [39] Y. Song and D. P. Kingma, “How to train your energy-based models,” 2021. [Online]. Available: <https://arxiv.org/abs/2101.03288>
- [40] M. Welling and Y. W. Teh, “Bayesian learning via stochastic gradient Langevin dynamics,” in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 681–688.
- [41] E. Nijkamp, M. Hill, T. Han, S.-C. Zhu, and Y. N. Wu, “On the anatomy of MCMC-based maximum likelihood learning of energy-based models,” 2019. [Online]. Available: <https://arxiv.org/abs/1903.12370>
- [42] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu, “Learning non-convergent non-persistent short-run MCMC toward energy-based model,” 2019. [Online]. Available: <http://arxiv.org/abs/1904.09770>
- [43] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002. [Online]. Available: <https://doi.org/10.1162/089976602760128018>
- [44] A. Hyvärinen, “Estimation of non-normalized statistical models by score matching,” *Journal of Machine Learning Research*, vol. 6, no. 24, pp. 695–709, 2005. [Online]. Available: <http://jmlr.org/papers/v6/hyvarinen05a.html>
- [45] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” 2020. [Online]. Available: <https://arxiv.org/abs/1907.05600>
- [46] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” 2020. [Online]. Available: <https://arxiv.org/abs/2011.13456>

- [47] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," *CoRR*, vol. abs/1605.07717, 2016. [Online]. Available: <http://arxiv.org/abs/1605.07717>
- [48] R. Kumar, S. Ozair, A. Goyal, A. Courville, and Y. Bengio, "Maximum entropy generators for energy-based models," 2019. [Online]. Available: <http://arxiv.org/abs/1901.08508>
- [49] J. J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *CoRR*, vol. abs/1609.03126, 2016. [Online]. Available: <http://arxiv.org/abs/1609.03126>
- [50] T. Kim and Y. Bengio, "Deep directed generative models with energy-based probability estimation," 2016. [Online]. Available: <http://arxiv.org/abs/1606.03439>
- [51] J. Xie, Y. Lu, R. Gao, S.-C. Zhu, and Y. N. Wu, "Cooperative training of descriptor and generator networks," 2018. [Online]. Available: <http://arxiv.org/abs/1609.09408>
- [52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [53] Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato, "Residual energy-based models for text generation," 2020. [Online]. Available: <https://arxiv.org/abs/2004.11714>
- [54] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterton, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, 13–15 May 2010, pp. 297–304. [Online]. Available: <http://proceedings.mlr.press/v9/gutmann10a.html>
- [55] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," 2017. [Online]. Available: <https://arxiv.org/abs/1702.08165>
- [56] Y. Du, T. Lin, and I. Mordatch, "Model based planning with energy based models," 2019. [Online]. Available: <https://arxiv.org/abs/1909.06878>
- [57] D. Belanger, "Deep energy-based models for structured prediction," Ph.D. dissertation, University of Massachusetts Amherst, 2017. [Online]. Available: [https://scholarworks.umass.edu/dissertations\\_2/1030](https://scholarworks.umass.edu/dissertations_2/1030)

- [58] J. Domke, “Generic methods for optimization-based modeling,” in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, N. D. Lawrence and M. Girolami, Eds., vol. 22. La Palma, Canary Islands: PMLR, 21–23 Apr 2012, pp. 318–326. [Online]. Available: <http://proceedings.mlr.press/v22/domke12.html>
- [59] B. Amos, L. Xu, and J. Z. Kolter, “Input convex neural networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 146–155. [Online]. Available: <http://proceedings.mlr.press/v70/amos17b.html>
- [60] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, “Support vector machine learning for interdependent and structured output spaces,” in *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2004, p. 104. [Online]. Available: <https://doi.org/10.1145/1015330.1015341>
- [61] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. [Online]. Available: <http://arxiv.org/abs/s1412.6980>
- [62] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context Encoders: Feature learning by inpainting,” 2016. [Online]. Available: <http://arxiv.org/abs/1604.07379>
- [63] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, “High-resolution image inpainting using multi-scale neural patch synthesis,” 2017. [Online]. Available: <http://arxiv.org/abs/1611.09969>
- [64] S. Iizuka, E. Simo-Serra, and H. Ishikawa, “Globally and locally consistent image completion,” *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)*, vol. 36, no. 4, pp. 107:1–107:14, 2017.
- [65] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, “Generative image inpainting with contextual attention,” 2018. [Online]. Available: <https://arxiv.org/abs/1801.07892>
- [66] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, “Image inpainting for irregular holes using partial convolutions,” 2018. [Online]. Available: <http://arxiv.org/abs/1804.07723>
- [67] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang, “Free-form image inpainting with gated convolution,” 2019. [Online]. Available: <http://arxiv.org/abs/1806.03589>
- [68] P. Brakel, D. Stroob, t, and B. Schrauwen, “Training energy-based models for time-series imputation,” *Journal of Machine Learning Research*, vol. 14, no. 48, pp. 2771–2797, 2013. [Online]. Available: <http://jmlr.org/papers/v14/brakel13a.html>

- [69] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, "Generalized inner loop meta-learning," *CoRR*, vol. abs/1910.01727, 2019. [Online]. Available: <http://arxiv.org/abs/1910.01727>
- [70] "Bitmap, pixel, screen fonts, small fonts, userbar — dafont.com." [Online]. Available: <https://www.dafont.com/bitmap.php>
- [71] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [72] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013. [Online]. Available: <http://arxiv.org/abs/1311.2901>

