
Deep Energy Networks for Data Completion

Olavo Bacelar

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](#).

1. Introduction

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 simple outputs, but we have also been increasingly seeing deep neural networks used as generative models for 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. 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, with the objective of learning representations that may be useful for downstream tasks.

Deep neural networks can be used to model the dependen-

cies 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.

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. 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. Furthermore, 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. Objectives

In this work, we seek to demonstrate that SPENs can be successfully ported to data completion tasks, particularly when dealing with the more loosely structured data of image problems. In this case we have that the outputs are not different from the inputs, in that they are just the parts of



Figure 1. 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.

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 evaluate the suitability of SPENs for data completion with the help of a dataset built for that purpose from free online fonts. The data completion task in this dataset consists in generating missing letters according to the style of the ones that are provided (see Figure 1). 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 such as PyTorch [8] or TensorFlow [9] to support higher order derivatives, which simplify considerably the approach taken by Belanger et al. [10].

1.2. Contributions

In summary, our contributions are as follows:

1. 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, and that supports arbitrary masks. Within this framework, we used similar neural architectures to ones that are already very successfully deployed in discriminative tasks.
2. 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.
3. 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.

4. 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.
5. We published our PyTorch implementation, including the code with which we created and the preprocessed 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.

2. Background

We start by describing the essential ingredients to understand the SPEN approach and some other aspects that are important for the proposed model.

2.1. Feed-Forward Models

With a feed-forward model we are interested in a direct mapping $f_{\omega}(x)$ from the input x 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) for the corresponding output classes. Unless a structure such as a Conditional Random Field (CRF) [6] is added to the end of a feed-forward model [11] to estimate interactions between different elements of the output, feed-forward models do not model direct interactions between those elements.

Feed-forward models can be integrated in the generative setting, namely with generative Adversarial Networks (GANs) [12] and Variational Autoencoders (VAEs) [13; 14]. In the GAN framework, a discriminator network attempts to distinguish whether its input images are from the dataset or are produced from a generator network. At the end of training the two networks simultaneously, the generator network should have learned to produce plausible images that the discriminator network cannot perceive as being artificial. GANs have evolved and are now capable of generating very realistic images [1], but it still remains the problem of lack of diversity in the resulting images. The VAE framework [13; 14] uses latent variables and, besides the reconstruction loss that is also used in usual autoencoders, a regularization term promotes continuous and semantically meaningful latent space. New models such as the VQ-VAE [15], which learns a fixed size vocabulary of learnable embeddings, recently improved on the quality of generated images.

2.2. Energy-Based Models

There have been at least two different but related ways in which the term *energy-based modeling* has previously been used. This includes models that use probabilistic modeling and have been often used as generative models, and other models that do not and naturally fit in a discriminative setting. 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. In this work we will call a model an EBM only when the energy does not have normalization requirements, which still covers both types of models.

Generative energy-based models employ probabilistic modeling through an energy function that represents an unnormalized negative log density. The probability distribution is represented by the Gibbs distribution:

$$p_\omega(x) = \frac{\exp(-E_\omega(x))}{Z_\omega}, \quad (1)$$

in which $p_\omega(x)$ is the probability of the data under the model with parameters ω , and Z_ω is the normalizing constant, which often cannot be computed. Despite enjoying a great expressive power, training is not done in an end-to-end manner and can be notoriously challenging, since maximum likelihood training of $p_\omega(x)$ is intractable for sufficiently complicated energy functions. In recent years there has been a flurry of research addressing different ways of training these models [16; 17; 18], showing that they can achieve good results in practice. For a recent review of those methods see [19]. Sampling is also often done with MCMC, which inherently entails mixing problems and difficulty in visiting different distribution modes.

In contrast to these generative models, **predictive energy-based models** correspond directly to the definition of an EBM in [20]. In those models, the energy is a function of both the input x and output y , naturally scoring 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:

$$\hat{y} = \arg \min_{y \in \mathcal{Y}} E_\omega(x, y). \quad (2)$$

Training such a model consists in configuring the energy function so that this search yields a good estimator \hat{y} .

Our work was directly inspired by the energy-based prediction paradigm that originated SPENs, since a data completion task may be seen as a prediction task of the hidden elements y given the visible ones x . However, there is also some similarity with the generative approach, since in our setting there are no qualitative differences between the visible elements x and the hidden elements y that we wish to generate, and we wish to complete missing values without knowing a priori which elements of the data are hidden or

visible. It is also the case that the energy will be a function of a single object, contrary to SPENs and similarly to generative EBMs. When we 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 training and sampling issues of generative EBMs.

2.3. Structured Prediction Energy Networks

The model that we will use, namely the Structured Prediction Energy Network (SPEN), was introduced by Belanger and McCallum [7]. A SPEN is a predictive EBM since it employs a gradient-based search procedure in output space to approximate the predictor in Equation 2. They 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 creators of SPENs found it valuable to add a learnable *feature network* $F(x)$ that encodes a fixed representation of x so that the energy function $E(F(x), y)$ interacts with x solely through the feature network.

2.3.1. PREDICTION

Prediction using SPENs derives from the approach in [21] and is defined to be a gradient-based minimization in y of the energy function $E_\omega(x, y)$, which is differentiable with respect to y . For each iteration t , this process is given by $y_t = y_{t-1} - \eta \nabla_y E_\omega(x, y_{t-1})$, where η is a learning rate. Belanger et al. [7] use a separate feed-forward network *Init* based on the features $F(x)$ to produce an initial estimate for y_0 , which is generally trained beforehand. Unrolling this entire process over T inference iterations results in:

$$y_T = \text{Init}(F(x)) - \sum_{t=1}^T \eta \nabla_y E_\omega(x, y_{t-1}). \quad (3)$$

2.3.2. END-TO-END LEARNING

The *end-to-end learning* method for SPENs that we apply has been previously introduced in [21], and adapted to SPENs by Belanger et al. [22]. It becomes much simpler to implement by employing automatic higher-order differentiation capabilities of modern neural network libraries like PyTorch [8], instead of the finite difference method in [10].

Suppose also that each step in Equation 3 is itself differentiable with respect to the parameters ω of the energy function. We can have this by taking twice differentiable energy functions. In this case, the whole algorithm is itself also differentiable with respect to ω , and it is possible to perform backpropagation through the entire unrolled computational graph, and therefore learn parameters that fit better the observations. For this reason, the authors of [10] did not use the ReLU nonlinear activation [23] in their energy networks, but the softplus function given by $\text{Softplus}_\beta(x) = \frac{1}{\beta} \log(1 + \exp(\beta x))$, which acts as a twice

differentiable approximation to the ReLU and tends to that function as $\beta \rightarrow \infty$.

With this method we have to deal with two different learning rates. The first one η is the *inner learning rate*, which controls the speed at which the gradient-based inference procedure proceed. The second is the *outer learning rate*, which is the one provided to the optimizer that actually regulates the change of the model parameters.

During training, the loss function is defined as a weighted sum of losses between the inference iterates y_t and the ground truth values y :

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^T w_t L(y_t, y). \quad (4)$$

The weighting scheme given by $w_t = \frac{1}{T-t+1}$ encourages good predictions to come after only a few iterations.

In general, we fix the number of iterations to be low so that we are learning an energy function in which the prediction process converges quickly, consuming also less computational resources. Even though other criteria are possible [24], we will consider that the number of iterations T of gradient-based inference is fixed.

3. Energy Networks for Data Completion

In this section, we discuss the modifications that we have made to the SPEN model for the data completion task.

3.1. Energy Network Architectures

We shall 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 work, 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. 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. 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.

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. Note that, in particular, the energy networks that we consider are indifferent to whether a feature is visible or hidden, therefore 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 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 $F(x)$.

Moreover, to simplify the notation, we express the energy of x and y in *their respective place* in the input as $E(x, y)$. In practice, we mostly compute the energy function with elements that are different than y , such as the iterative estimates that have as many components as y , and thus can be arranged in the corresponding place together with x .

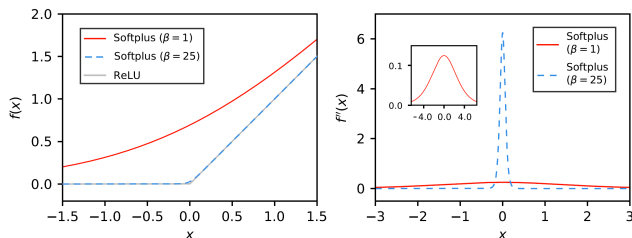


Figure 2. (Left) softplus functions for different values of β and ReLU. (Right) The second derivatives of the softplus functions. A higher value for β makes the softplus function more similar to the ReLU, but their second derivatives have a smaller range in which they are substantially different than 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.

Belanger et al. [10] used a softplus function with a large value for β , such as 25 as the non-linearity in their networks, motivated by a higher similarity with the ReLU function when this parameter has a higher value (see Figure 2). However, this choice hindered learning in our experiments. The differentiation of the inference process involves second derivatives of the energy function and, therefore, of the softplus function too. However, because the second derivative of the softplus is narrower as a result of a higher β (also depicted in Figure 2), upstream gradients do not properly flow backwards into the parameters when the values of the domain are outside that range. Therefore, we chose to work with softplus functions with a lower value of the parameter, in this case $\beta = 1$.

3.2. Inference and Learning

The learning algorithm is adapted from the one given in §2.3.2. The components of the data that we wish to complete often must belong to an interval I and not to \mathbb{R} but we can apply the method of Belanger et al. [10], in which gradient descent was actually realized in the unconstrained space of the logits of y . With the right choice of a mapping $f: \mathbb{R} \rightarrow I$, we can bring the iterate ℓ from the unbounded space \mathbb{R} (in which we can take gradient steps freely) to the relevant set I . A step in the unconstrained space is given by $\ell_t = \ell_{t-1} - \eta_t \nabla_{\ell} E_{\omega}(x, f(\ell_{t-1}))$, and the full inference procedure is given by:

$$\ell_T = \ell_0 - \sum_{t=1}^T \eta_t \nabla_{\ell} E_{\omega}(x, f(\ell_{t-1})). \quad (5)$$

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. The initial logits ℓ_0 , similarly to Equation 3, are generated by an initialization routine that may depend on the visible data, $\text{Init}(x)$, which we will describe shortly.

For a problem in which we wish to classify each component of y into two classes, as will be the case in Section 4 when we generate black and white images, we can model, for each component, a non-degenerate probability distribution identified by a single value $p \in (0, 1)$. We can optimize in an unconstrained space by instead applying the logit transformation $\ell = \text{logit}(p)$ and, to convert back from logit space to the probability space of p , we take f to be $\sigma(x) = \text{Sigmoid}(x) = \frac{1}{1 + \exp(-x)}$.

Similarly to Equation 4, we compute a loss function with the intermediate and final values $f(\ell_t)$. To propagate its loss along the unrolled computational graph of the inference process in Equation 5, we take advantage of a feature of PyTorch that allows the creation of the computational graph of the backwards pass that produces $\nabla_{\ell} E_{\omega}(x, f(\ell_{t-1}))$. We can therefore differentiate the whole inference process with respect to the parameters ω of the energy function.

Initialization of the inference iterates

Different initialization procedures $\text{Init}(x)$ can be used for generating the initial inference iterates ℓ_0 but, as described earlier, we often need to pass them to a function f before integrating them together with the observed data and computing the energy value. We need to place some care into the values that serve as input to f , otherwise the upstream gradients will not flow backwards through the function properly. For example, in the case case that f is the sigmoid function its derivatives are low for values that deviate too much from zero (see right side of Figure 2 in red). 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 like $[-1, 1]$. In §4 we choose to use the Compress function given by:

$$\text{Compress}(x)_i = \frac{x_i}{\max_i \text{abs}(x_i)}. \quad (6)$$

The initialization routines that we have considered are:

Initialization with an estimate: In this case, we have a pre-trained initialization function that depends on the visible data and suggests initial values for the iterates ℓ_0 . This is closer to what Belanger et al. [10] suggested.

Initialization to zeros. All iterates ℓ_0 are initialized to zero.

Random initialization. We initialize ℓ_0 by sampling random numbers from a distribution concentrated near zero. At test time, we may choose to initialize ℓ_0 with zero instead.

With those last two initialization procedures, we now expect the energy function to be sufficiently rich to do the entire job of modeling dependencies without any external help.

4. Experiments with Font Completion

In this section, we explain the implementation of the proposed algorithm, performing data completion in a dataset composed of fonts, with their letters formatted as images.

4.1. Fonts Dataset

We have downloaded a dataset containing open access ttf font files from the [DaFont](#) website, and built a visual dataset through which experiments can quickly be iterated. It is composed of 1076 bitmap fonts, each comprising the 26 letters of the English alphabet in their uppercase form.

We can save computation by reducing each letter to a 32×32 picture which, because of the simple nature of the bitmap letters, does not degrade their quality significantly. A single font is represented by a $(26, 32, 32)$ -shaped tensor. This structure is visible in multiple places in Figure 5.

4.2. Data Completion in the Fonts Dataset

The data completion task is defined as the generation of missing letters in a font, given some of the other 26 letters. The quality of the generated images may be evaluated both visually and quantitatively (see next subsection).

We can describe the task in the same way as in §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 , that the algorithm will try to estimate with \hat{y} . With this definition, we avoid having to choose between different types of masks in image inpainting, or even producing them manually. To hide a certain letter in the $(26, 32, 32)$ font tensor so it can be presented to the algorithm, we fill the corresponding $(1, 32, 32)$ slice with zeros.

With SPENs, we always relax discrete problems so that, instead of generating directly the letters in the space $\{0, 1\}^{N \times 32 \times 32}$, we generate matrices in the space $[0, 1]^{N \times 32 \times 32}$, where N is the number of letters that are hidden in a font. To generate the final discrete estimates, we apply a discretizing function to each pixel $\hat{p}_i \in [0, 1]$, given simply by:

$$\text{Discretize}_\tau(x) = \begin{cases} 1, & x \geq \tau \\ 0, & x < \tau \end{cases}, \quad x \in [0, 1]. \quad (7)$$

Unless indicated differently, we will take $\tau = 0.5$.

To train the algorithms in a situation with a fixed number of visible letters, as well as in a situation in which every possible configuration can be present, we created two different input modes. These modes provide a mask that indicates which letters are shown to the algorithm, thus requiring that the remaining ones are for the algorithm to generate. In *fixed mode*, a fixed number of letters from each font are chosen randomly to be input to the algorithm, and the others are masked. In *mixed mode*, a number `random_n_visible_letters` between 1 and 25 is drawn randomly for each font; then, `random_n_visible_letters` of the letters from the same font are chosen randomly to be input to the algorithm. With these input modes one may test the dependence of the performance of the algorithms on the information they are provided with.

To present a larger variety of situations during training, we make a different choice on which letters of a certain font are visible in each new epoch, independently of the chosen input mode.

4.3. Metric and Loss Function

A score for evaluating the generated letters is the Intersection over Union (IoU), which we deemed to correlate well with perceived visual similarity. It is commonly used to evaluate the similarity of bounding boxes in object detection problems. For a single letter, we define the IoU score as $\text{IoU} = \frac{|B \cap \hat{B}|}{|B \cup \hat{B}|}$, where we denote by B the set of pixels that are black in the ground truth letter, and by \hat{B} the set of pixels that are black in the corresponding generated letter. The loss function that we use to train the various models is the average of the pixel-wise Binary Cross-Entropy (BCE).

4.4. Convolutional Blocks

We will use convolutional layers [25] as the main constituent of the autoencoder baseline in §4.5.2, and the SPEN-based EBM in §4.6, thereby enforcing an appropriate inductive bias for a task in which the spatial structure of the data is essential. We call a component that aggregates different operations that are commonly used together in these networks a *convolutional block*. These were inspired by the constituents of the VGG16 [26] architecture. We define a

convolutional block as a sequence of multiple intercalated groups, each with a convolutional layer, the softplus as the twice-differentiable non-linear operation, and an optional batch normalization [27] layer in-between. Each convolutional operation employs filters that span the whole depth of the input, and an odd kernel side size s with padding $p = \lfloor \frac{s}{2} \rfloor$. Between convolutional blocks there will often be a downsampling or upsampling operation.

4.5. Baseline Models

In the following, we describe two baseline models that we have used for comparison with the SPEN-based algorithm.

4.5.1. AVERAGE BASELINE

A naive first approach is to estimate a certain letter by simply averaging all the images corresponding to the same letter in the training dataset, and then apply the discretizing function in Equation 7, such that:

$$F_l^{avg} = \text{Discretize}_\tau \left(\frac{1}{M} \sum_{f=1}^M F_l^{(f)} \right), \quad (8)$$

in which $l = 1, \dots, 26$, and $F_l^{(f)} \in \{0, 1\}^{32 \times 32}$ is the l -th letter of the f -th font in the train dataset, and M is the number of elements of the train dataset. In the discretizing process we use the threshold $\tau \in (0, 1)$ which maximizes the IoU score between the resulting images and the fonts in the validation set. In our case the best threshold was $\tau = 0.27$, to which it corresponded an average IoU score in the test set of 36.3%.

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 use an autoencoder with convolutional neural networks, in which the task is not solely

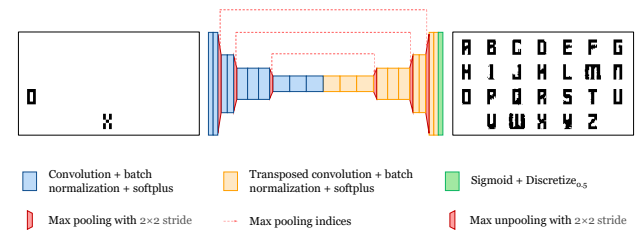


Figure 3. Architecture of the convolutional autoencoder baseline. Both the encoder part in the left and the decoder part in the right consist of 10 convolutional layers each, organized in 4 convolutional blocks.

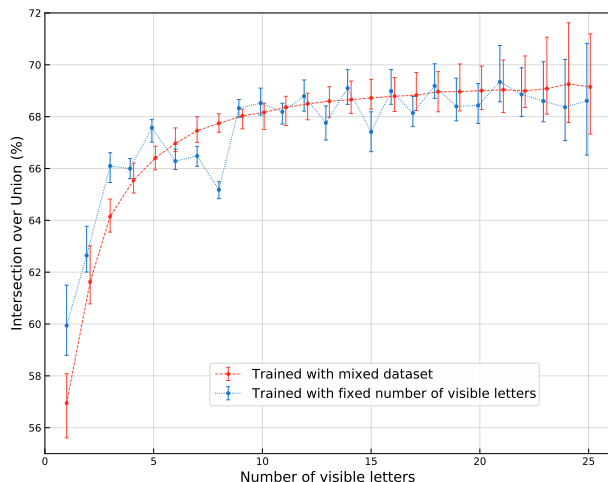


Figure 4. The IoU score of the baseline convolutional autoencoder model evaluated in fixed input mode on the test dataset, for different numbers of visible letters. Each dot shows the median of 100 evaluations with different masks but the same number of visible letters. The error bars exhibit the full range of IoU scores obtained with those 100 evaluations. 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.

to reconstruct the input images, but also to generate the ones that are not visible in the masked input, similarly to the generator network in Context Encoders [28]. To that end, we have adapted the SegNet architecture [29], which has a convolutional encoder-decoder structure, employing however the convolutional blocks previously described. We have modified SegNet to receive as input the masked font, and to output a reconstruction of the entire font, including the hidden letters in the input. The loss function is the reconstruction loss of an autoencoder model, therefore it takes into account the whole output.

We kept a feature of SegNet in which the indices of the max pooling downsampling operations in the encoder part are cached for later use by an upsampling operation. We have also modified the SegNet architecture for performance sake. We did not employ the usual convolutional layers in the decoder part of the network, but transposed convolutional ones. Moreover, we did not use the max pooling operation right before an unpooling operation in the middle of the network. In the convolutional blocks we used the softplus 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. In each convolutional block, we have used batch normalization which did make a significant difference in the results.

We searched over the free parameters of the architecture for the best results in the validation set, employing also early stopping. The best scoring architecture of this convolutional baseline in the validation set is depicted in a simplified manner in Figure 3. In Figure 4 we depict the results corresponding to two different procedures for training and testing the baseline. There, we may observe that training one single model in mixed input mode proved to be well-performing and stable when compared to models trained specifically with the same number of visible letters.

4.6. Deep Energy Networks

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. In this energy-based approach we only take into account in the BCE loss the generated estimates to the logits of the missing letters, since we are not trying to reconstruct the whole input. In Figure 5 it is depicted the process for data completion in this font dataset, including the assignment procedure in which intermediate soft estimates for the missing letters y_t are placed in the appropriate places within the font tensor.

4.6.1. ARCHITECTURE OF THE ENERGY NETWORK

The architecture of the energy network consists of a convolutional part followed by a fully connected part. We use the encoder of the previous autoencoder baseline, consisting of 4 convolutional blocks with a total of 10 convolutional layers, as the convolutional part, with the difference that batch normalization was not added. 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, the energy E . Those consist of two linear layers that use dropout and softplus activation functions with $\beta = 1$, and a final linear layer without dropout or a softplus activation.

Contrarily to what happened with the autoencoder baselines, we could expand substantially the capacity of this convolutional part (increasing the number of feature maps of the convolutional part) and still improve the performance.

4.6.2. TRAINING

We observed that the choice of the learning rates was quite dependent on the number of inference iterations T . Moreover, the training process was more sensitive to adjusting properly the value of the inner learning rate.

Similarly to the training of the convolutional autoencoder baseline (see §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

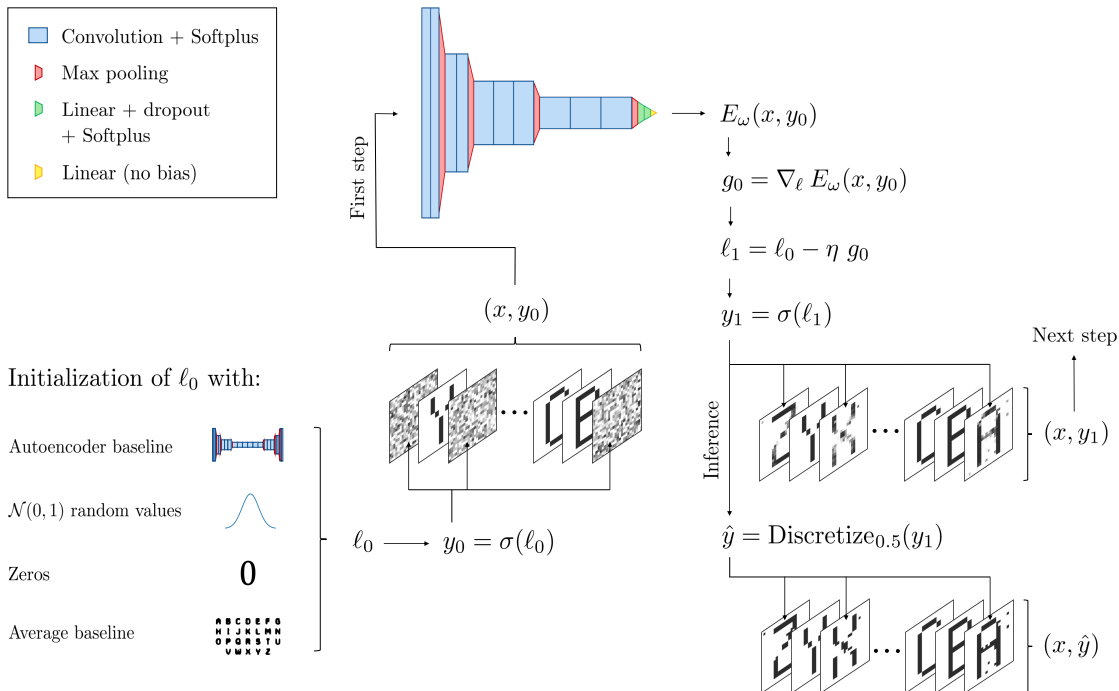


Figure 5. SPEN algorithm adapted for performing data completion in the font dataset: architecture of the energy network, initialization routines, first gradient-based inference step, and generation of the missing letters. As an example, we start the initial logits ℓ_0 by random initialization and, after 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.

of all the observed IoU’s in the validation set. Even with this choice, we observed significantly longer running times than with the convolutional baseline, particularly when the number of visible letters was very low and the number of iterations in the inference routine was one. However, as we will see later, those were not the configurations for which the best results were attained.

4.6.3. EXPERIMENTS

Number of inference iterations. Belanger et al. [10] 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 result. However, the training of the energy networks when $T = 1$ and when `n_visible_letters` was also small took considerably longer than when $T = 3$.

Initialization routines. We tested the different initialization procedures for filling in the place of the masked letters (see §3.2). Each of the four initialization routines 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.

The first two are initializations with an estimate. In the first one the autoencoder baseline generates the initial logits ℓ_0 . In the second case we initialize ℓ_0 with a version of the average baseline, which may be a good starting point for doing inference. However, to solve the issue that all initial values predict the same class (because all pixel values predicted by the average baseline are equal or greater than zero), we decided to simply subtract 0.5 to each one. The third one, consisting of initialization to zeros, is exactly equal to what has been described in §3.2. Finally, with random initialization, the logits ℓ_0 are initialized by sampling random pseudorandom values from the $\mathcal{N}(0, 1)$ distribution at training time.

We observed that varying the initialization procedure of the energy-based algorithms did not have a very significant impact in the results. However, training time could be quite longer if random initialization was applied. All initialization methods produced significantly better results than the autoencoder baseline.

Dropout [30] was not used in the initial work with SPENs, for the reasons given in [24]. Even though we cannot guarantee that the same random dropout pattern is chosen for each call of the energy function, we still decided to employ it in the previous experiments, with probabilities $p_1 = 0.3$

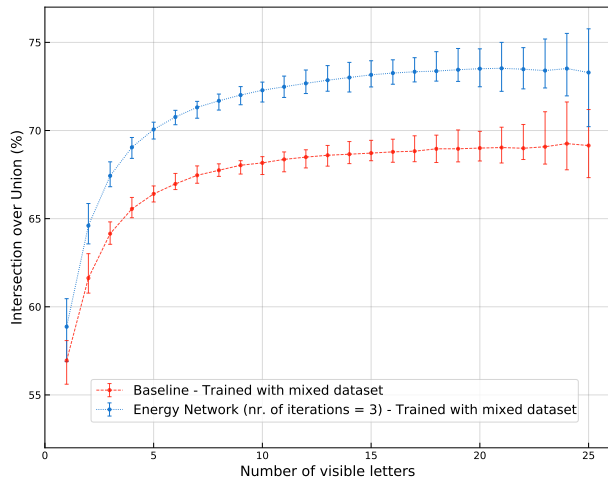


Figure 6. The IoU score of the energy network algorithm and the baseline convolutional autoencoder model, trained in mixed input mode, but evaluated in fixed input mode on the test dataset, for different numbers of visible letters. The dots and bars have the same meaning as in Figure 4.

and $p_2 = 0.5$. However, when making a systematic study, different dropout probabilities did not result in a substantial difference in performance.

4.7. Discussion of the Experimental Results

We have observed that the results that we have obtained are not very dependent on the choices of the hyperparameters. By contrast, they are quite sensitive to the choice of learning rates, in particular the inner rate of the inference routine.

As we can see in Figure 6, the energy network trained in mixed input mode generalized better than the autoencoder baseline when tested in the fixed input mode case for the entire range of `n_visible_letters`. 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. The difference in quality of the resulting images and the plausibility of the generated letters between our algorithm and the baseline is also significant. Furthermore, the energy network and the baseline performed better than the average baseline by a large margin, as would be expected.

5. Related Work

The most relevant type of data completion problem to our application in the fonts dataset is the inpainting problem, which is a difficult task that requires some level of semantic understanding when the affect area is large.

Feed-forward models. Most image inpainting methods employ adversarial training which has already proved to be very successful at unconditional image generation. The Context Encoder [28] employs a convolutional autoencoder

architecture that accepts the masked input and tries to output a reconstructed image. The DeepFill model [31] was able to enhance results by introducing a contextual attention mechanism to model spatially distant dependencies in the generated image. Another model [32] introduced partial convolutions which enabled the model to effectively deal with irregular shaped masks.

Predictive Energy-Based Models. Input Convex Neural Networks [33] have a similar formulation as SPENs but are convex in the space of missing data, which facilitates the inference process but limits its expressive power.

Generative Energy-Based Models naturally allow for multiple completions to be generated. Du and Mordatch [17] adapt the sampling Langevin dynamics to only modify the hidden pixels and maintain the visible ones fixed, and Song and Ermon [18] modify their annealed Langevin dynamics in a similar way.

6. Conclusions and Future Work

We have 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 that is capable of filling in an arbitrary subset of missing data.

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 §3.2, including changing the value for the β 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.

More work can be done to assess the performance of generation of higher dimensionality data. In fact, our method can be directly translated into the domain of image inpainting by just having as input the 3 color channels instead of the 26 different letters. Our method also presents a natural way of dealing with randomly shaped masks in that setting.

References

- [1] T. Karras *et al.*, “Analyzing and improving the image quality of StyleGAN,” in *CVPR*, 2020.
- [2] A. Razavi *et al.*, “Generating diverse high-fidelity images with VQ-VAE-2,” *NIPS*, 2019.
- [3] S. Tulyakov *et al.*, “MoCoGAN: Decomposing motion and content for video generation,” *arXiv e-prints*, 2017.
- [4] A. Radford *et al.*, “Language models are unsupervised multitask learners,” 2019.
- [5] B. Taskar *et al.*, “Max-Margin Markov Networks,” in *NIPS*, 2003.
- [6] J. D. Lafferty *et al.*, “Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data,” in *ICML*, 2001.
- [7] D. Belanger and A. McCallum, “Structured Prediction Energy Networks,” in *ICML*, 2016.
- [8] A. Paszke *et al.*, “PyTorch: an imperative style, high-performance deep learning library,” in *NeurIPS*, 2019.
- [9] M. Abadi *et al.*, “TensorFlow: a system for large-scale machine learning,” *arXiv e-prints*, 2016.
- [10] D. Belanger *et al.*, “End-to-end learning for structured prediction energy networks,” in *ICML*, 2017.
- [11] S. Zheng *et al.*, “Conditional random fields as recurrent neural networks,” in *ICCV*, 2015.
- [12] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” in *NIPS*, 2014.
- [13] D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” *arXiv e-prints*, 2014.
- [14] D. J. Rezende *et al.*, “Stochastic backpropagation and approximate inference in deep generative models,” in *ICML*, 2014.
- [15] A. van den Oord *et al.*, “Neural discrete representation learning,” in *NeurIPS*, 2017.
- [16] E. Nijkamp *et al.*, “On the anatomy of MCMC-based maximum likelihood learning of energy-based models,” *arXiv e-prints*, 2019.
- [17] Y. Du and I. Mordatch, “Implicit generation and generalization in energy-based models,” *arXiv e-prints*, 2020.
- [18] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” in *NeurIPS*, 2020.
- [19] Y. Song and D. P. Kingma, “How to train your energy-based models,” *arXiv e-prints*, 2021.
- [20] Y. LeCun *et al.*, “A tutorial on energy-based learning,” 2006.
- [21] J. Domke, “Generic methods for optimization-based modeling,” in *ICML*, 2012.
- [22] M. Welling and Y. W. Teh, “Bayesian learning via stochastic gradient Langevin dynamics,” in *ICML*, 2011.
- [23] X. Glorot *et al.*, “Deep sparse rectifier neural networks,” in *ICML*, 2011.
- [24] D. Belanger, *Deep energy-based models for structured prediction*. PhD thesis, 2017.
- [25] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*. MIT Press, 2016.
- [26] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv e-prints*, 2015.
- [27] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating deep network training by reducing internal covariate shift,” in *ICML*, 2015.
- [28] D. Pathak *et al.*, “Context Encoders: feature learning by inpainting,” *arXiv e-prints*, 2016.
- [29] V. Badrinarayanan *et al.*, “SegNet: a deep convolutional encoder-decoder architecture for image segmentation,” *IEEE PAMI*, 2016.
- [30] N. Srivastava *et al.*, “Dropout: a simple way to prevent neural networks from overfitting,” *JMLR*, 2014.
- [31] J. Yu *et al.*, “Generative image inpainting with contextual attention,” in *CVPR*, 2018.
- [32] G. Liu *et al.*, “Image inpainting for irregular holes using partial convolutions,” *arXiv e-prints*, 2018.
- [33] B. Amos *et al.*, “Input Convex Neural Networks,” in *IMCL*, 2017.