Deep Structured Learning (IST, Spring 2023)

Homework 1

Instructors: André Martins and Chryssa Zerva

Deadline: Friday, April 14, 2022.

Please turn in the answers to the questions below in a PDF file, together with the code you implemented to solve them (when applicable).
Please submit **a single zip file** in Fenix under your name.

Question 1

1. In this exercise, you will design a multilayer perceptron to compute a Boolean function of D variables, $f : \{0,1\}^D \to \{0,1\}$, defined as:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{D} x_i = K, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

   where K is an integer between 0 and D.

   (a) (5 points) Show that the function above cannot generally be computed with a single perceptron. *Hint: think of a simple counter-example.*

   (b) (15 points) Show that the function above can be computed with a multilayer perceptron with **a single hidden layer with two hidden units** and Heaviside step function activations $g\colon \mathbb{R} \to \{0,1\}$ with

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

   and where **all the weights and biases are integers** (positive, negative, or zero). Provide all the weights and biases of such network, and ensure that the resulting network is robust to infinitesimal perturbation of the inputs, i.e., the resulting function $h : \mathbb{R}^D \to \mathbb{R}$ should be such that $\lim_{t\to 0} h(\boldsymbol{x} + t\boldsymbol{v}) = h(\boldsymbol{x}) = f(\boldsymbol{x})$ for any $\boldsymbol{x} \in \{0,1\}^D$ and $\boldsymbol{v} \in \mathbb{R}^D$.

   (c) (10 points) Repeat the previous exercise if the hidden units use rectified linear unit activations instead. (You will get partial credit if you solve for $D = 2$ and $K = 1$ only.)

Question 2

Image classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple image classification problem. **Please do not use any machine learning library such as `scikit-learn` or similar for this exercise; just plain linear algebra (the `numpy` library is fine).**

You will use the Fashion MNIST dataset, which is available as a torchvision dataset. The file `hw1-dataload-fashionMNIST.py` includes code to load, split and save the dataset. Each example is a 28x28 grayscale image, associated with a label from 10 classes that represent a clothing garment: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot. There is a correspondence between the Fashion-MNIST and the original MNIST benchmark dataset.

The dataset statistics can be found in Table 1.

	Train	Dev	Test
Number of sentences	48000	12000	10000

Table 1: Fashion MNIST Dataset split statistics.

Skeleton code For this question, you are recommended but not required to use the skeleton script `hw1-q2-2023-skeleton-fashion.py`, which has been provided to you on the course webpage. The skeleton code also provides a function to load the data into 28x28 representations and includes a normalisation step. It also includes evaluation and plotting functions.

The script requires `Python3`, `numpy`, `tqdm`, `PIL` and `matplotlib`.

The main evaluation metric used is the accuracy:

$$\text{accuracy} = \frac{\sum_{i=0}^{N}(\hat{y}_i == y_i)}{\sum_{i=0}^{N} y_i}, \text{ where } N \text{ is the test set size.} \tag{3}$$

We will use the pixel values for the feature representation throughout this exercise.

1. In the first part of the exercise, we will focus on linear models.
   (a) (10 points) Implement the `update_weights` method of the `Perceptron` class in the file `hw1-q2-skeleton-fashion.py`. Then train 15 epochs of the perceptron on the training set and report its performance on the validation and test set. Plot the accuracy as a function of the epoch number (the functions to plot are already implemented).
   (b) (5 points) Repeat the same exercise using logistic regression instead (no regularisation necessary), using stochastic gradient descent as your training algorithm. Set a fixed learning rate $\eta = 0.001$. This can be solved by implementing the `update_weights` method in the `LogisticRegression` class.

2. Let us now focus on non-linear models: in this part, you will implement a multi-layer perceptron (a feed-forward neural network).
   (a) (15 points) Without using any neural network toolkit, implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm which is needed to train the model. Use a relu activation function for the hidden layers, and a multinomial logistic loss (also known as cross-entropy) in the output layer. Don't forget to include the bias terms in your hidden units. Train the model with stochastic gradient descent tuning the learning rate in the $\{0.001, 0.1\}$ in the validation set.
   (b) (5 points) Is the model able to learn feature representations? What about the previous models? Briefly justify.

Number of Epochs	15
Learning Rate	0.001
Hidden Sizes	[100]
Number of Layers	1
Dropout	0.1
Batch Size	1
Activation	ReLU
Optimizer	SGD

Table 2: Default hyperparameters for MLP.

Question 3

Image classification with an autodiff toolkit. In the previous question, you had to manually implement gradient backpropagation. In this question, you will implement the same system using a deep learning framework with automatic differentiation. A skeleton code for PyTorch is provided (`hw1-q3-skeleton_fashion.py`) but if you feel more comfortable with a different framework you are free to use it instead. The skeleton code also contains the evaluation functions and code to output a confusion matrix over the test set. You will additionally need the `torch` module and optionally (for the confusion matrix: `pandas`, `seaborn` and scikit-learn

1. (5 points) Implement a linear model with logistic regression, using stochastic gradient descent as your training algorithm (use a batch size of 1). Train your model for 15 epochs and tune the learning rate in your validation data, using values in: $[0.0001, 0.1]$.

   Report the results for the best configuration and plot:

   - Training and validation loss
   - Validation accuracy

   as a function of the epoch number. Report the final accuracy and F1-score in the test set.

   In the skeleton code, you will need to implement the method `train_batch()` and the class `LogisticRegression`'s `__init__()` and `forward()` methods.

2. (10 points) Implement a feed-forward neural network with 2 hidden layers, using dropout regularization and relu activation. Use the hyperparameters and training/model design choices shown in Table 2 as a starting point.

   Tune the values presented in the table as default. Tune each of these hyperparameters while leaving the remaining at their default value:

   - The learning rate: $\{0.001, 0.01\}$.
   - The hidden sizes, within the range of: $[200, 20]$.
   - The number of layers: $\{1, 2\}$.
   - The dropout probability: $\{0.1, 0.2, 0.4\}$.

   Report your best configuration, make similar plots as in the previous question, and report the final test accuracy.

   In the skeleton code, you will need to implement the class `FeedforwardNetwork`'s `__init__()` and `forward()` methods.

3. In this question we will focus on CNN architectures.

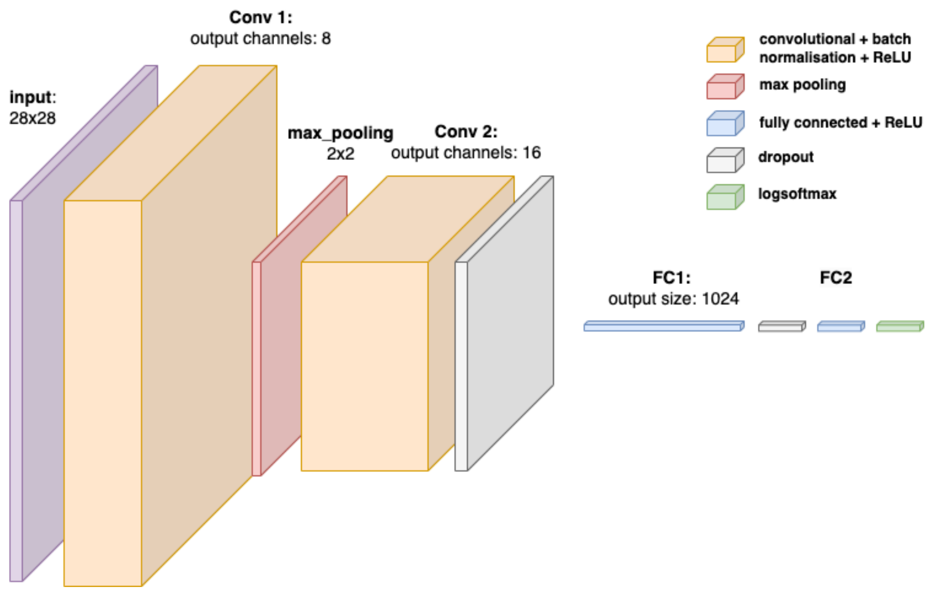

Figure 1: Initial architecture for the required CNN

Number of Epochs	15
Learning Rate	0.001
# convolutions	2
Kernel size	2 (for all convolutions)
Padding	1
Stride	1
MaxPooling	2x2
Dropout	0.1
Batch Size	1
Activation	ReLU
1st Linear Layer (FC1) Output dim	1024
Optimizer	SGD

Table 3: Default hyperparameters for MLP.

(a) (15 points) Implement a simple convolutional neural network (CNN) to address the same task. Use 2 convolutional layers with one layer of max-pooling between them and batch normalisation after each convolution. They should be followed by a dropout layer and two fully connected layers with dropout between them. Use the rectified linear unit activation function for all layers except for the output one, for which you should use the LogSoftmax activation. The hyper-parameters of the network that should be used for the starting point model are provided in Table 3 and Figure 1 illustrates the corresponding architecture.

Hint: use the functions `nn.Conv2d` `nn.BatchNorm2d` *and* `nn.MaxPool2d`.

Tune the values presented in the table as default. Tune each of these hyperparameters while leaving the remaining at their default value:

- The learning rate: $\{0.001, 0.01\}$.
- The kernel size: $\{2, 3\}$.
- Batch Size: $\{1, 4\}$.
- The convolution filters (output channels):

  - convolution layer 1: $\{8, 16, 32\}$.
  - convolution layer 2: $\{16, 32, 64\}$.
- The dropout probability: $\{0.1, 0.2, 0.5\}$.

(b) (5 points) How does the feature representation of CNN differ to the model you had to implement for the previous question? What are the advantages of using a CNN over a FFNN?