

# Deep Structured Learning (IST, Fall 2020)

## Homework 1

**Instructor:** André Martins

**TA:** Marcos Treviso

**Deadline: Wednesday, October 28, 2020.**

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 email your solutions in **electronic format** (a single zip file) with the subject “Homework 1” to:

`deep-structured-learning-instructors@googlegroups.com`

**Hard copies will not be accepted.**

### Question 1

In this exercise, you will show that the binary and multinomial logistic losses are convex.

1. (5 points) The sigmoid activation function is  $\sigma(z) = 1/(1 + e^{-z})$ , where  $z \in \mathbb{R}$ . Show that its derivative can be expressed as  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ .

**Solution:**

*Proof.* By using the quotient rule:

$$\begin{aligned}\sigma'(z) &= \frac{-e^{-z} \cdot (-1)}{(1 + e^{-z})^2} \\ &= \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} \\ &= \sigma(z) \cdot \frac{e^{-z}}{1 + e^{-z}} \\ &= \sigma(z) \cdot \frac{1 + e^{-z} - 1}{1 + e^{-z}} \\ &= \sigma(z) \cdot \left(1 - \frac{1}{1 + e^{-z}}\right) \\ &= \sigma(z) \cdot (1 - \sigma(z)).\end{aligned}$$

□

2. (5 points) Consider a binary classification problem with  $y \in \{\pm 1\}$ . A binary logistic regression model defines  $P(y = +1 \mid x; \mathbf{w}, b) = \sigma(z)$  with  $z = \mathbf{w}^\top \phi(x) + b$ . The binary logistic loss is

$$L(z; y) = \begin{cases} -\log \sigma(z) & \text{if } y = +1 \\ -\log(1 - \sigma(z)) & \text{if } y = -1 \end{cases} = -\frac{1+y}{2} \log \sigma(z) - \frac{1-y}{2} \log(1 - \sigma(z)), \quad (1)$$

where  $y$  is the gold label. Assume  $y = +1$  and compute the first and second derivatives of  $L'(z; y = +1)$  and  $L''(z; y = +1)$  with respect to  $z$ . Show that the binary logistic loss is convex as a function of  $z$ . *Hint: Use the result from the previous question.*

**Solution:**

The first derivative of  $L$  w.r.t.  $z$  is:

$$\begin{aligned} L'(z; y) &= -\frac{1+y}{2} [\log \sigma(z)]' - \frac{1-y}{2} [\log(1 - \sigma(z))]' \\ &= -\frac{1+y}{2} \cdot \frac{1}{\sigma(z)} \cdot \sigma(z) \cdot (1 - \sigma(z)) - \frac{1-y}{2} \cdot \frac{-1}{1 - \sigma(z)} \cdot \sigma(z) \cdot (1 - \sigma(z)) \\ &= -\frac{1+y}{2} \cdot (1 - \sigma(z)) + \frac{1-y}{2} \cdot \sigma(z). \end{aligned}$$

And the second derivative is:

$$\begin{aligned} L''(z; y) &= \frac{1+y}{2} \cdot \sigma(z)' + \frac{1-y}{2} \cdot \sigma(z)' \\ &= \frac{1+y}{2} \cdot \sigma(z) \cdot (1 - \sigma(z)) + \frac{1-y}{2} \cdot \sigma(z) \cdot (1 - \sigma(z)) \\ &= \sigma(z) \cdot (1 - \sigma(z)) \cdot \left( \frac{1+y}{2} + \frac{1-y}{2} \right) \\ &= \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} \\ &= \frac{e^{-z}}{(1 + e^{-z})^2}. \end{aligned}$$

For the case  $L'(z; y = +1)$  we get only with the first term:

$$\begin{aligned} L'(z; y) \Big|_{y=+1} &= -\frac{1+1}{2} \cdot (1 - \sigma(z)) \\ &= \sigma(z) - 1 \\ &= \frac{-1}{1 + e^z} \end{aligned}$$

For the case  $L''(z; y = +1)$  we also get only with the first term:

$$\begin{aligned} L''(z; y) \Big|_{y=+1} &= \frac{e^{-z}}{(1 + e^{-z})^2} \\ &> 0 \quad \forall z \in \mathbb{R}. \end{aligned}$$

Since  $L''(z; y) > 0, \forall z \in \mathbb{R}$ ,  $L$  is a convex function.

3. (5 points) Let us now turn to the multi-class case, where  $y \in \{1, \dots, K\}$  with  $K \geq 2$ . Let  $\mathbf{z} \in \mathbb{R}^K$ . The softmax transformation is a function from  $\mathbb{R}^K$  to  $\mathbb{R}^K$  defined as

$$[\text{softmax}(\mathbf{z})]_j = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

for  $j = 1, \dots, K$  (the entries of  $\mathbf{z}$  are called logits or scores). Compute the Jacobian matrix of the softmax transformation on point  $\mathbf{z}$  – this is the  $K$ -by- $K$  matrix whose  $(j, k)$ -th entry is  $\frac{\partial[\text{softmax}(\mathbf{z})]_j}{\partial z_k}$ .

**Solution:**

Let  $\boldsymbol{\pi} := \text{softmax}(\mathbf{z})$  be the vector of probabilities which results from the softmax transformation, and let  $Z(\mathbf{z}) = \sum_{k=1}^K \exp(z_k)$  be the partition function. The Jacobian is given as follows:

$$\begin{aligned} \frac{\partial[\text{softmax}(\mathbf{z})]_j}{\partial z_i} &= \frac{\partial}{\partial z_i} \left[ \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)} \right] \\ &= \frac{Z(\mathbf{z}) \left( \frac{\partial \exp(z_j)}{\partial z_i} \right) - \exp(z_j) \left( \sum_k \frac{\partial \exp(z_k)}{\partial z_i} \right)}{Z(\mathbf{z})^2} \\ &= \frac{Z(\mathbf{z}) \exp(z_j) \delta_{ij} - \exp(z_j) \exp(z_i)}{Z(\mathbf{z})^2} \\ &= \frac{\exp(z_j) \delta_{ij}}{Z(\mathbf{z})} - \frac{\exp(z_j) \exp(z_i)}{Z(\mathbf{z})^2}, \end{aligned}$$

where  $\delta_{ij}$  is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

So, if  $i \neq j$  we have:

$$\begin{aligned} \frac{\partial[\text{softmax}(\mathbf{z})]_j}{\partial z_i} &= \frac{-\exp(z_j) \exp(z_i)}{Z(\mathbf{z}) \cdot Z(\mathbf{z})} \\ &= -\pi_j \pi_i. \end{aligned}$$

And if  $i = j$ :

$$\begin{aligned} \frac{\partial[\text{softmax}(\mathbf{z})]_j}{\partial z_i} &= \frac{Z(\mathbf{z}) \exp(z_j) - \exp(z_j) \exp(z_i)}{Z(\mathbf{z}) \cdot Z(\mathbf{z})} \\ &= \frac{\exp(z_j) (Z(\mathbf{z}) - \exp(z_i))}{Z(\mathbf{z}) \cdot Z(\mathbf{z})} \\ &= \frac{\exp(z_j)}{Z(\mathbf{z})} \cdot \frac{Z(\mathbf{z}) - \exp(z_i)}{Z(\mathbf{z})} \\ &= \pi_j (1 - \pi_i). \end{aligned}$$

So, in general, by joining both cases the Jacobian is given by:

$$\frac{\partial[\text{softmax}(\mathbf{z})]_j}{\partial z_i} = \pi_j (\delta_{ij} - \pi_i),$$

or in matrix notation:

$$J(\mathbf{z}) = \frac{\partial \text{softmax}(\mathbf{z})}{\partial \mathbf{z}} = \text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi} \boldsymbol{\pi}^\top.$$

4. (5 points) The multinomial logistic loss is defined as  $L(\mathbf{z}; y = j) = -\log[\text{softmax}(\mathbf{z})]_j$ . Compute the gradient and Hessian of this loss with respect to  $\mathbf{z}$  and show that this loss is convex with respect to  $\mathbf{z}$ .

*Hint: Use again the result from the previous question.*

**Solution:**

Let  $\mathbf{e}_j = [0, \dots, 0, \underbrace{1}_j, 0, \dots, 0]$  be the one-hot vector that indicates the  $j$ -th label. We have

$$\begin{aligned}\nabla_{\mathbf{z}}(-\log[\text{softmax}(\mathbf{z})]_j) &= -\nabla_{\mathbf{z}} \log[\text{softmax}(\mathbf{z})]_j \\ &= -\frac{1}{[\text{softmax}(\mathbf{z})]_j} \nabla_{\mathbf{z}} [\text{softmax}(\mathbf{z})]_j \\ &= -\frac{1}{\pi_j} \frac{\partial \text{softmax}(\mathbf{z})}{\partial \mathbf{z}} \mathbf{e}_j \\ &= -\frac{1}{\pi_j} (\text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi} \boldsymbol{\pi}^\top) \mathbf{e}_j \\ &= -\frac{1}{\pi_j} (\pi_j \mathbf{e}_j - \pi_j \boldsymbol{\pi}) \\ &= -(\mathbf{e}_j - \boldsymbol{\pi}),\end{aligned}$$

That is:

$$\nabla_{\mathbf{z}}(-\log[\text{softmax}(\mathbf{z})]_j) = \text{softmax}(\mathbf{z}) - \mathbf{e}_j.$$

The Hessian  $H(\mathbf{z})$  is given by:

$$\begin{aligned}H(\mathbf{z}) &= \frac{\partial \text{softmax}(\mathbf{z})}{\partial \mathbf{z}} - \underbrace{\frac{\partial \mathbf{e}_j}{\partial \mathbf{z}}}_{=0} \\ &= \text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi} \boldsymbol{\pi}^\top.\end{aligned}$$

We will show that the Hessian  $H(\mathbf{z})$  is symmetric and positive semidefinite for any  $\mathbf{z}$ , which implies that the loss  $L$  is convex. From the expression above, it is clear that  $H(\mathbf{z})$  is symmetric. To show that it is positive semidefinite we need to prove that  $\mathbf{v}^\top H(\mathbf{z}) \mathbf{v} \geq 0$  for

any  $\mathbf{v} \in \mathbb{R}^K$ . Using the expression above for the Hessian, we obtain

$$\begin{aligned}
\mathbf{v}^\top H(\mathbf{z}) \mathbf{v} &= \mathbf{v}^\top (\text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi} \boldsymbol{\pi}^\top) \mathbf{v} \\
&= \mathbf{v}^\top \text{Diag}(\boldsymbol{\pi}) \mathbf{v} - (\mathbf{v}^\top \boldsymbol{\pi})^2 \\
&= \sum_j \pi_j v_j^2 - \underbrace{\left( \sum_j \pi_j v_j \right)^2}_{:= \bar{v}^2} \\
&= \sum_j \pi_j v_j^2 - \underbrace{\sum_j \pi_j}_{=1} \bar{v}^2 \\
&= \sum_j \pi_j (v_j^2 - \bar{v}^2) \\
&= \sum_j \pi_j (v_j^2 + \bar{v}^2 - 2v_j \bar{v}) - 2 \sum_j \pi_j (\bar{v}^2 - v_j \bar{v}) \\
&= \sum_j \pi_j (v_j - \bar{v})^2 - 2\bar{v} \sum_j \pi_j (\bar{v} - v_j) \\
&= \sum_j \pi_j (v_j - \bar{v})^2 - 2\bar{v} \left( \bar{v} - \sum_j \pi_j v_j \right) \\
&= \sum_j \pi_j (v_j - \bar{v})^2 - \underbrace{2\bar{v}(\bar{v} - \bar{v})}_{=0} \\
&= \sum_j \pi_j (v_j - \bar{v})^2 \\
&\geq 0,
\end{aligned}$$

since  $\pi_j \geq 0$  for every  $j$ .

Therefore,  $H(\mathbf{z})$  is a positive semidefinite matrix, and therefore  $L(\mathbf{z}; y) = -\log[\text{softmax}(\mathbf{z})]_j$  is a convex function.

5. (5 points) Show that in a linear model where  $\mathbf{z} = \mathbf{W}\boldsymbol{\phi}(x) + \mathbf{b}$  the multinomial logistic loss is also convex with respect to the model parameters  $(\mathbf{W}, \mathbf{b})$ , and therefore a local minimum is also a global minimum. Is this also true in general when  $\mathbf{z}$  is not a linear function of the model parameters?

**Solution:** The loss as a function of  $(\mathbf{W}, \mathbf{b})$  is the composition of the function  $L(\mathbf{z}; y)$ , which we proved to be convex in the previous exercise, with the mapping  $\mathbf{z} = \mathbf{W}\boldsymbol{\phi}(x) + \mathbf{b}$ , which is linear in  $(\mathbf{W}, \mathbf{b})$ . Since the composition of a convex function with a linear mapping (and more generally an affine mapping) is also convex, we have that the loss is also convex when seen as a function  $(\mathbf{W}, \mathbf{b})$ . Since it is convex, any local minimum is a global minimum. This is not true in general if  $\mathbf{z}$  is not a linear function of  $(\mathbf{W}, \mathbf{b})$ , since the composition of a convex function with an arbitrary mapping is not necessarily convex.

## Question 2

**Optical character recognition 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).

Download the OCR dataset from <http://ai.stanford.edu/~btaskar/ocr>. This dataset contains binary image representations of 52,152 alphabetical characters `a-z` (the characters are grouped together to form English words, but this structure will be ignored in this exercise). The task is to take each image representation as input (with 16x8 pixels) and to predict as output the correct character in `a-z` (i.e., a multi-class classification problem with 26 classes). The dataset is organized into 10 folds: folds 0–7 are for training (41,679 examples), 8 is for validation (5,331 examples), and 9 is for testing (5,142 examples). The evaluation metric is the fraction of characters correctly classified.

**Skeleton code** For this question, you are recommended but not required to use the skeleton script `hw1-q2.py`, which has been provided to you on the course webpage. In order to use it, make sure it is located in the same directory as the `letter.data` file from the OCR dataset. The script requires `Python3`, `numpy`, and `matplotlib`.

1. In the first part of the exercise, we will use as a feature representation the binary pixel values.

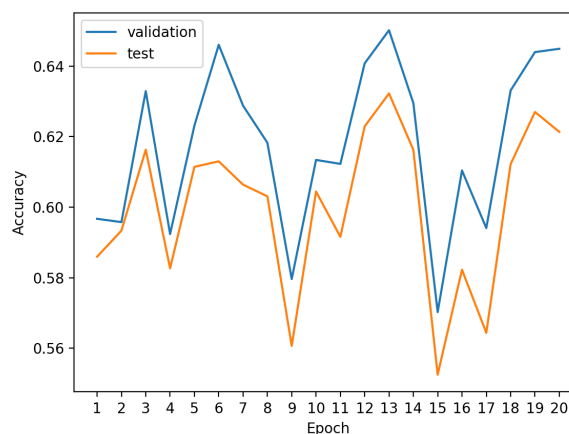
- (a) (5 points) Do you think this is a good choice of feature representation? Justify.

**Solution:** Not for use with a linear classifier, since it is highly translation variant and does not exploit the fact that pixels are highly correlated.

- (b) (10 points) Implement the `update_weights` method of the `Perceptron` class in `hw1-q2.py`. Then train 20 epochs of the perceptron on the training set and report its performance on the validation and test set. Plot the accuracies as a function of the epoch number. You can do this with the command

```
python hw1-q2.py perceptron
```

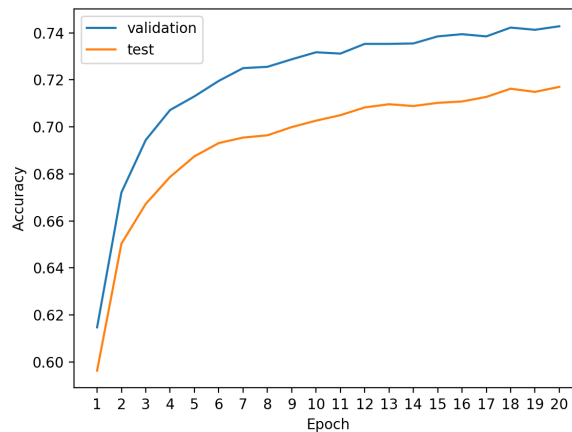
**Solution:** Accuracies were 0.6449 on the validation set and 0.6214 on the test set.



- (c) (5 points) Repeat the same exercise using logistic regression instead (without regularization), 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. You can do this with the command

```
python hw1-q2.py logistic_regression
```

**Solution:** Accuracies were 0.7428 on the validation set and 0.7170 on the test set.



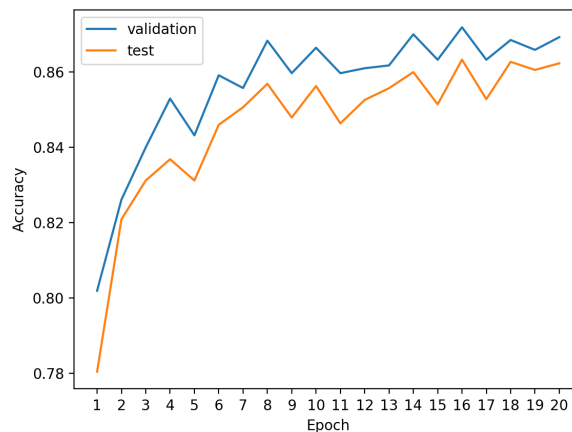
2. Let us now do some feature engineering.

- (a) (10 points) Can you think of a better feature representation? Come up with one and train the perceptron there. You can implement your feature representation in the function `custom_features` and then run

```
python hw1-q2.py perceptron -custom_features
```

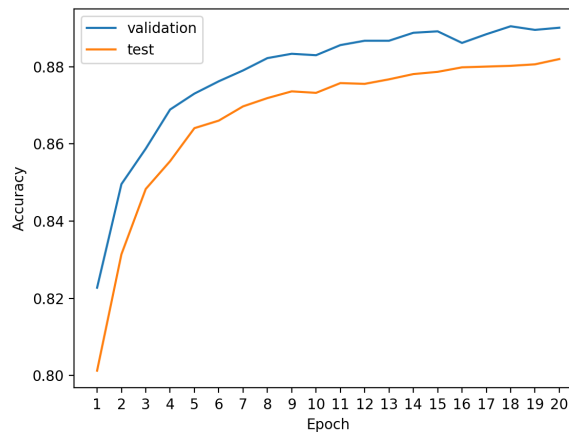
Suggestion: instead of individual pixel binary values  $\phi_i(\mathbf{x}) = x_i$  (where  $i$  indexes a pixel position), use as features all pixel pairwise combinations,  $\phi_{ij}(\mathbf{x}) = x_i x_j$ .

**Solution:** With this feature representation we got 0.8693 on the validation set and 0.8623 on the test set.



- (b) (5 points (bonus)) Repeat the previous exercise using logistic regression instead (with  $\eta = 0.001$  and without regularization).

**Solution:** With this feature representation we got 0.8901 on the validation set and 0.8820 on the test set.



3. In the previous part, you might have noticed that feature engineering can be tedious. Now, you will implement a multi-layer perceptron (a feed-forward neural network) again using as input the original feature representation (i.e. simple independent pixel values).

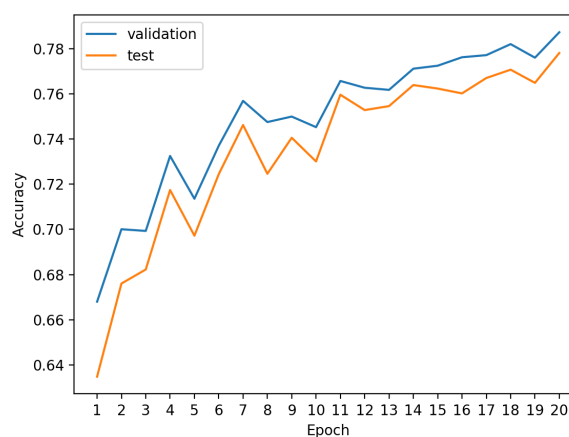
- (a) (5 points) Explain why multi-layer perceptrons can learn internal representations and avoid manual feature engineering.

**Solution:** The hidden layers in a multi-layer perceptron are able to play the part of a “learned” feature function  $\Psi(x)$ , trained jointly with the final classification layer. The non-linearities ensure that the learned features can be expressive.

- (b) (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 200 hidden units, a `relu` activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer. Don’t forget to include bias terms in your hidden units. Train the model with stochastic gradient descent tuning the learning rate in  $\{0.001, 0.01, 0.1\}$  in the validation set.

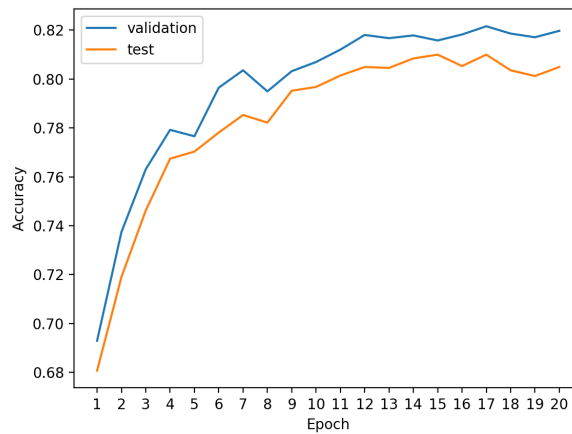
**Solution:**

For  $lr = 0.001$  the accuracies were 0.7873 on the validation set and 0.7781 on the test set.

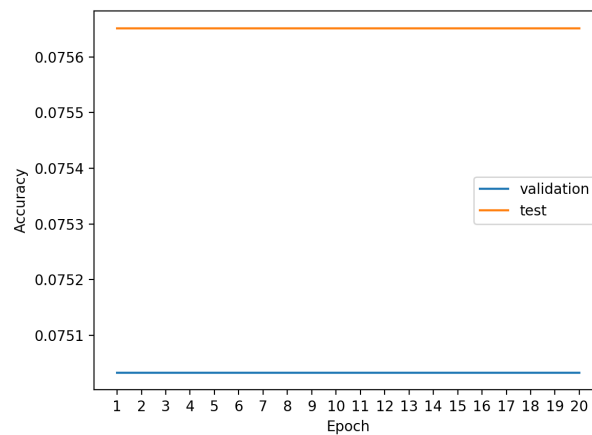


For  $lr = 0.01$  the accuracies were 0.8197 on the validation set and 0.8049 on the test set.





The model diverges. For  $lr = 0.1$  the accuracies were 0.0750 on the validation set and 0.0757 on the test set.



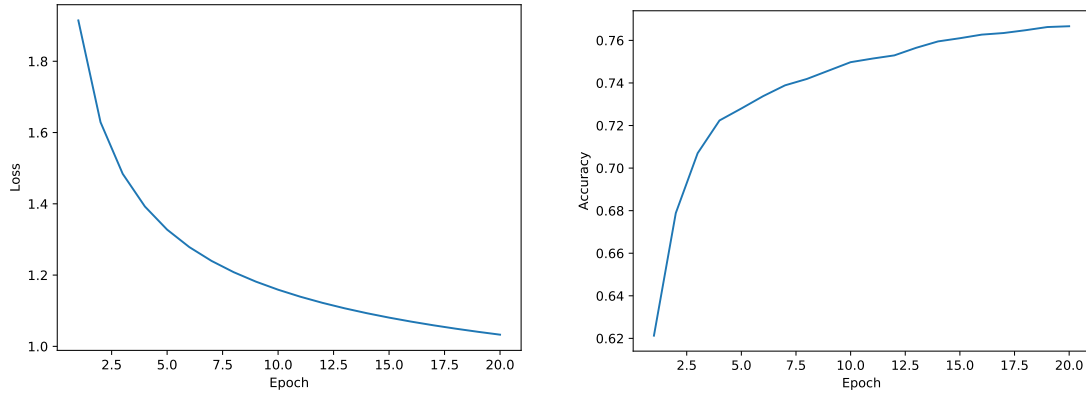
### Question 3

**OCR with an autograd toolkit.** In the previous question, you had to write gradient back-propagation by hand. This time, you will implement the same system using a deep learning framework with automatic differentiation. A skeleton code for PyTorch is provided (`hw1-q3.py`) but if you feel more comfortable with a different framework you are free to use it instead.

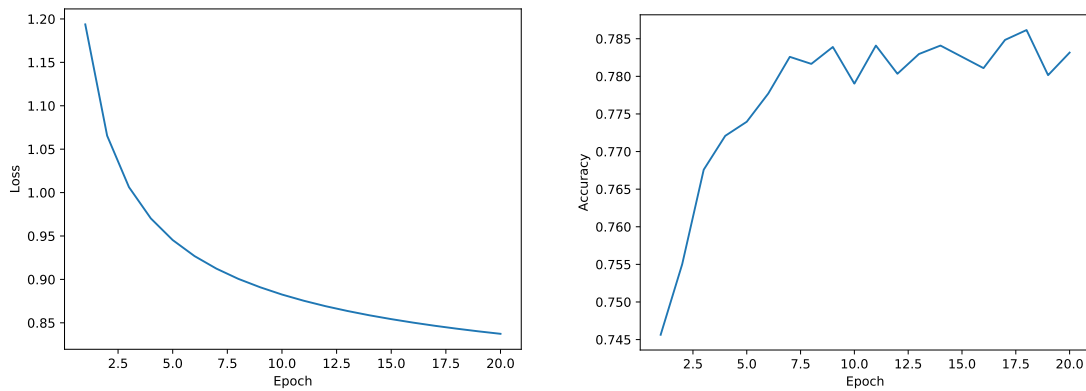
- (10 points) Implement a linear model with logistic regression, using stochastic gradient descent as your training algorithm (use a batch size of 1). Train your model for 20 epochs and tune the learning rate in your validation data, using the following values:  $\{0.001, 0.01, 0.1\}$ . For the best configuration, report it and plot two things: the training loss and the validation accuracy, both as a function of the epoch number. Report the final accuracy 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.

#### Solution:

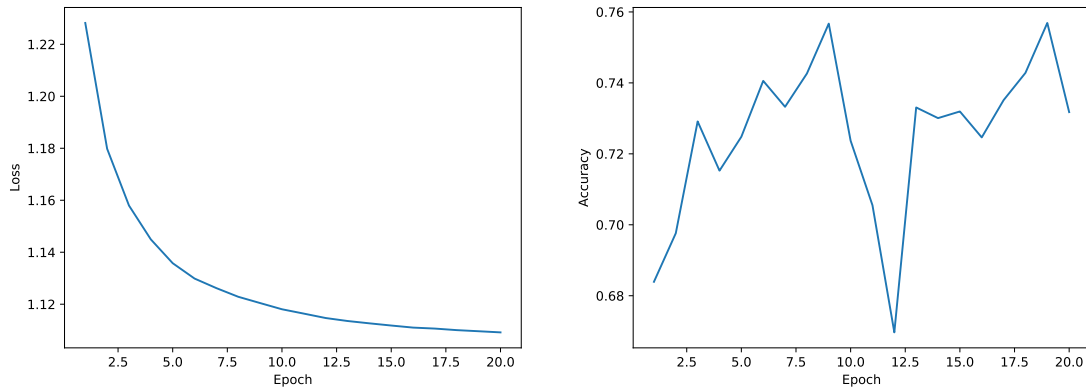
For  $lr = 0.001$  the accuracies were 0.7666 on the validation set and 0.7419 on the test set.



For  $lr = 0.01$  the accuracies were 0.7832 on the validation set and 0.7563 on the test set.



For  $lr = 0.1$  the accuracies were 0.7318 on the validation set and 0.7122 on the test set.



2. (15 points) Implement a feed-forward neural network with a single layer, using dropout regularization. Make sure to include all the hyperparameters and training/model design choices shown in Table 1. Use the values presented in the table as default. Tune each of these hyperparameters while leaving the remaining at their default value:

- The learning rate:  $\{0.001, 0.01, 0.1\}$ .
- The hidden size:  $\{100, 200\}$ .
- The dropout probability:  $\{0.3, 0.5\}$ .
- The activation function: `relu` and `tanh`.

- The optimizer: SGD and Adam.

Number of Epochs	20
Learning Rate	0.01
Hidden Size	200
Dropout	0.3
Batch Size	1
Activation	ReLU
Optimizer	SGD

Tabela 1: Default hyperparameters.

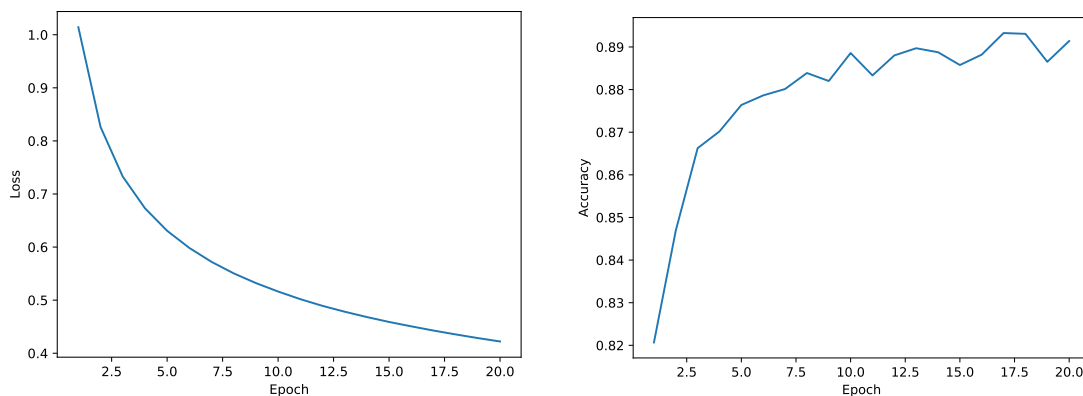
Report your best configuration, make similar plots as in the previous question, and report the final test accuracy.

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

**Solution:** Best configuration:

- The learning rate: 0.01
- The hidden size: 200
- The dropout probability: 0.3
- The activation function: ReLU
- The optimizer: SGD

Accuracies were 0.8914 on the validation set and 0.8821 on the test set.



3. (5 points (bonus)) Using the same hyperparameters as in Table 1, increase the model to 2 and 3 layers. Report your best configuration, make similar plots as in the previous question, and report the final test accuracy.

**Solution:** Best configuration: 2 hidden layers of size 200.

Accuracies were 0.8912 on the validation set and 0.8755 on the test set.

