EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATH. - NATURWISS. FAKULTÄT
FACHBEREICH INFORMATIK
KOGNITIVE SYSTEME · PROF. A. ZELL

Deep Neural Networks

Assignment 4

Assignment due by: 20.11.2019, Discussions on: 04.12.2019

Note: All further programming exercises in this course have to be done in python 3.5 or later. To ensure that we can run your code, you are not allowed to use any external python libraries except for *numpy, scipy, matplotlib* (and later *tensorflow* when we get to the corresponding exercises). Upload your code together with a PDF of the written exercises in a ZIP-file. Make sure you include your names in all documents.

Question 1 Classification with Linear Regression using Gradient Descent (11 Points)

In this task you are asked to implement a Linear Regression Model that is trained using Gradient Descent in order to classify images from the MNIST dataset. MNIST consists of greyscale images of handwritten digits from 0 to 9. Each image is of dimension $28 \times 28 (= 784 \text{ pixels})$. A common operation is to flatten a matrix to a vector which means in this case that every image is identified with a vector of length 784 that is fed into the model. In this model, the output is not a scalar but a 10-dimensional vector:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{1}$$

where $\hat{\boldsymbol{y}}, \boldsymbol{b} \in \mathbb{R}^{10}$, $\boldsymbol{x} \in \mathbb{R}^{784}$ and $\boldsymbol{W} \in \mathbb{R}^{10\times 784}$. As the error measure (loss function) we take the squared L_2-norm;

$$\mathcal{L}(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2 = (\boldsymbol{y} - \hat{\boldsymbol{y}})^T(\boldsymbol{y} - \hat{\boldsymbol{y}}) = (\boldsymbol{y} - (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}))^T(\boldsymbol{y} - (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})),$$

where $\boldsymbol{y} \in \mathbb{R}^{10}$ is the one-hot encoded ground truth label of the corresponding $\boldsymbol{x}$, which means that all elements of $\boldsymbol{y}$ are 0 except of the element whose index is that of the correct class of $\boldsymbol{x}$, which has the value 1 (e.g. if $\boldsymbol{x}$ is an image representing the number 6, then only $y_6 = 1$ and $y_j = 0$ for all $j \in \{1, \ldots, 5, 7, \ldots, 10\}$. In the inference step, we simply choose the index of the maximal element of $\hat{\boldsymbol{y}}$ to be the predicted digit.
In the lecture you saw that you can find the optimal solution of such a regression problem easily by solving the normal equations. However, since solving the normal equations often is computationally infeasible, we can also obtain the solution by means of Gradient Descent.

(a) Show that

$$\nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x}) = 2(\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{x}^T \;\; (= 2(\hat{\boldsymbol{y}} - \boldsymbol{y}) \otimes \boldsymbol{x}) \text{ and } \nabla_{\boldsymbol{b}}\mathcal{L}(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x}) = 2(\hat{\boldsymbol{y}} - \boldsymbol{y}).$$

where $\otimes$ is the outer product. (2 points)

(b) In the above problem formulation the model only accepts one image at a time. However, to obtain a more stable (stable in the sense that the influence of outliers that don't represent the true distribution is reduced) gradient at each update step, we can modify the model to accept multiple images as input. In this case, the model can be rephrased as $\hat{\boldsymbol{Y}} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{B}$, where $\boldsymbol{Y}, \hat{\boldsymbol{Y}}, \boldsymbol{B} \in \mathbb{R}^{10\times n}$ and $\boldsymbol{X} \in \mathbb{R}^{784\times n}$. The columns of $\boldsymbol{Y}, \hat{\boldsymbol{Y}}$ and $\boldsymbol{X}$ consist of a ground truth label, model prediction and a corresponding image, respectively, while in every column of $\boldsymbol{B}$ there is a copy

of $\boldsymbol{b}$. The corresponding loss function changes to $\mathcal{L}(\boldsymbol{W}, \boldsymbol{B}, \boldsymbol{Y}, \boldsymbol{X}) = \sum_{j=1}^{n} \mathcal{L}(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{Y}_j, \boldsymbol{X}_j)$, where $\boldsymbol{Y}_j$ and $\boldsymbol{X}_j$ are the $j-$th columns of $\boldsymbol{Y}$ and $\boldsymbol{X}$, respectively.
How do the derivatives of (a) change in this case for n inputs? (1 point)

(c) Complete the file *linear_regression.py*. Note that the dimensions are transposed. (4 points)

(d) Let the network train. Provide a plot of the training error and the evaluation error per epoch for learning rate=0.01, epochs=10 and batch size=100. Give also the corresponding test errors and test accuracies after the final epoch. Experiment with the training parameters (learning rate, epochs, batch size). Write down the best parameter combination with the corresponding test error and accuracy that you found. (2 points)

(e) When experimenting in (d) you may run into overflow issues (e.g. learning rate=0.1, epochs=10, batch size=100). This can happen since we allow the output of our model to be arbitrarily large. One solution can be to squish the output of $\hat{\boldsymbol{y}}$ into the interval $[0, 1]$ by using the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. How do the derivatives in (a) change when instead of (1), we have the model $\hat{\boldsymbol{y}} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ where we take the sigmoid function element-wise? (2 points)

Question 2 Softmax (9 points)

In classification problems where we have more than two classes, we often use the softmax function defined by

$$\boldsymbol{S}(\boldsymbol{a}) : \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \cdots \\ S_N \end{bmatrix} \text{ with } S_i = \frac{e^{a_i}}{\sum_j e^{a_j}}.$$

The properties of softmax (all output values in the range $[0, 1]$ and sum up to 1) make it suitable for a probabilistic interpretation that is very useful in machine learning. In particular, in multiclass classification tasks, we often want to assign probabilities that our input belongs to one of a set of output classes. To train a model with this, we can minimize the cross-entropy $H(y, \boldsymbol{S}(\boldsymbol{a}))$ between the true distribution (which is just probability 1 on the correct class, 0 on all others) and the predicted probabilities $\boldsymbol{S}(\boldsymbol{a})$ (where $\boldsymbol{a}$ is the un-normalized output of our classifier, the preactivation). Thus we get the following loss function (for a single point of data):

$$\mathcal{L}(\boldsymbol{a}, y) = H(y, \boldsymbol{S}(\boldsymbol{a})) = -\sum_c \delta_{yc} \log(\boldsymbol{S}(\boldsymbol{a})_c) = -\log(\boldsymbol{S}(\boldsymbol{a})_y),$$

where δ_{yc} is the Kronecker delta which is 1 if $y = c$ and 0 otherwise.

(a) Compute $\nabla_{\boldsymbol{a}}\mathcal{L}$, i.e. the gradient of the output of $\mathcal{L}$ wrt. the preactivations $a_1, \ldots, a_N$. (2 points)

(b) Implement the softmax classification function and the softmax cross-entropy loss $\mathcal{L}(\boldsymbol{a}, y)$. (For this part, (c) and (d), use the provided python template file *softmax.py*). (2 points)

(c) Implement the analytical gradient of the softmax cross-entropy loss from (a). (2 points)

(d) Test your functions from (b) and (c) by comparing the analytical gradient to a numerical gradient at three randomly chosen points and write down the error. (3 points)

Note: Make sure you apply the numerical stabilization techniques for the Softmax function discussed so far.