Uppsala University
Department of Information Technology
Division of Systems and Control
NW 2024-12
Last rev. January 15, 2025 by NW

Deep Learning

Instruction to the laboratory work

Lab 1: Linear regression and backpropagation

Language: Python/numpy

Preparation:
Read what is stated in the reading instructions below
Solve all preparatory exercises in Section 2

Reading instructions:

- UDL book: Chapter 7.1-7.4
- This lab-pm: Chapter 1-2.

Name		Assistant's comments
Program	Year of reg.	
Date		
Passed prep. ex.	Sign	
Passed lab.	Sign	

Contents

1 Introduction

After completing this laboratory assignment, you should:

- know how to implement and train a linear classification model from scratch
- understand and implement the equations in backpropagation
- have a clear path forward for solving Hand-in assignment 1

After this lab, you will put the pieces together in Hand-in Assignment 1, which has the goal

- implement a fully interconnected multilayer neural network from scratch
- optimize that model using backpropagation together with stochastic gradient descent

2 Preparation exercises

2.1 Linear regression

Consider a regression problem with multivariate input $\mathbf{x} = [x_1, \ldots, x_{D_i}]^\mathsf{T}$ and scalar output $y \in \mathbb{R}$. We want to find a model from the input to the output using a linear regression model,

$$\begin{aligned} y = f[\mathbf{x}, \boldsymbol{\phi}] &= \sum_{j=1}^{D_i} \omega_j x_j + \beta \\ &= \boldsymbol{\omega}^T \mathbf{x} + \beta \end{aligned} \tag{1a}$$

where the weight vector $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_{D_i}]^\mathsf{T}$ and the offset β are the parameters $\boldsymbol{\phi} = \{\boldsymbol{\omega}, \beta\}$ of the model.

Consider a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^{I}$. The cost L is computed by summing the following loss ℓ_i (mean squared error, MSE) over all training data points

$$L = \frac{1}{I} \sum_{i=1}^{I} \ell_i, \qquad \text{with the loss} \qquad \ell_i = (f_i - y_i)^2. \tag{1b}$$

where

$$f_i = f[\mathbf{x}_i, \boldsymbol{\phi}] = \sum_{j=1}^{D_i} \omega_j x_{ij} + \beta \tag{1c}$$

is the output of the model for input $\mathbf{x}_i$ and where x_{ij} is the j^{th} component of $\mathbf{x}_i$.

To train this model with gradient descent, we need access to the gradient of the loss with respect to the model parameters, i.e., the partial derivatives $\frac{\partial \ell_i}{\partial \omega_1}, \ldots, \frac{\partial \ell_i}{\partial \omega_{D_i}}$, and $\frac{\partial \ell_i}{\partial \beta}$. These you will derive in the following.

Question 2.1: *Given a data point* $\mathbf{x}_i, y_i$*, based on the model in* (1)*, derive expressions for*

$$\frac{\partial \ell_i}{\partial \beta} \quad \textit{and} \quad \frac{\partial \ell_i}{\partial \omega_j} \quad \textit{expressed in terms of} \quad \frac{\partial \ell_i}{\partial f_i}, \quad \frac{\partial f_i}{\partial \beta}, \quad \textit{and} \quad \frac{\partial f_i}{\partial \omega_j}. \tag{2}$$

Note: The "In terms of" here means that the answer should only include these three terms, no other variables.

Answer:

Question 2.2: *Based on the model in* (1)*, derive expressions for (the above used variables)*

$$\frac{\partial \ell_i}{\partial f_i}, \quad \frac{\partial f_i}{\partial \beta}, \quad \textit{and} \quad \frac{\partial f_i}{\partial \omega_j}. \tag{3}$$

Answer:

Question 2.3: *Write all the expressions in Question 2.1 and 2.2 in vectorized manner representing the derivative of the cost* (1b)*, i.e. using the following vectors and matrices*

$$\frac{\partial L}{\partial \mathbf{f}} = \frac{1}{I}\begin{bmatrix} \frac{\partial \ell_1}{\partial f_1} \\ \vdots \\ \frac{\partial \ell_I}{\partial f_I} \end{bmatrix}, \qquad \frac{\partial \mathbf{f}}{\partial \beta} = \begin{bmatrix} \frac{\partial f_1}{\partial \beta} & \cdots & \frac{\partial f_I}{\partial \beta} \end{bmatrix}, \qquad \frac{\partial \mathbf{f}}{\partial \boldsymbol{\omega}} = \begin{bmatrix} \frac{\partial f_1}{\partial \omega_1} & \cdots & \frac{\partial f_I}{\partial \omega_1} \\ \vdots & & \vdots \\ \frac{\partial f_1}{\partial \omega_{D_i}} & \cdots & \frac{\partial f_I}{\partial \omega_{D_i}} \end{bmatrix}$$

Answer:

2.2 Softmax and cross-entropy

For a classification problem with multiple classes $y \in \{1, \ldots, D_o\}$ we typically use a softmax function

$$\text{softmax}_k[\mathbf{f}] = \frac{\exp[f_k]}{\sum_{k'=1}^{D_o} \exp[f_{k'}]} \tag{4}$$

The likelihood that input $\mathbf{x}$ belongs to class y is then

$$Pr(y = k|\mathbf{x}) = \text{softmax}_k\big[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]\big] \tag{5}$$

where $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ is a regression model

$$\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}] = \boldsymbol{\Omega}\mathbf{x} + \boldsymbol{\beta}. \tag{6}$$

with the parameters $\boldsymbol{\phi} = \{\boldsymbol{\Omega}, \boldsymbol{\beta}\}$.

The recommended loss function ℓ_i for a multiclass classification problem is the cross-entropy loss, which, for numerical reasons when $z \ll 0$, is computed *together* with the softmax function[1]

$$\begin{aligned}
\ell_i &= -\sum_{k=1}^{D_o} \widetilde{y}_{ik} \log\left[\text{softmax}_k(\mathbf{f}_i)\right] \\
&= \log\left(\sum_{k=1}^{D_o} e^{f_{ik}}\right) - \sum_{k=1}^{D_o} \widetilde{y}_{ik} f_{ik}
\end{aligned} \tag{7}$$

where $\widetilde{y}_{ik}$ is the one-hot encoding of the true label y_i for data point i

$$\widetilde{y}_{ik} = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{if } y_i \neq k \end{cases} \qquad \text{for } k = 1, \ldots, D_o$$

and f_{ik} is the k^{th} output of the regression model $\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$.

Question 2.4: *Given a data point $\mathbf{x}_i, \tilde{y}_i$, based on the loss function in* (7)*, derive expressions for*

$$\frac{\partial \ell_i}{\partial f_{ik}}$$

.

Answer:

[1] Make sure you understand that (7) is the same expression as terms in eq. (5.24) in the course book

3 Laboratory exercises

This section contains instructions for the laboratory session. It consists of three notebooks. In the first notebook, you will implement and train a linear regression model using gradient descent only using `numpy`. In the following two notebooks, you will implement the equations required for implementing and training a neural network.

3.1 Linear classification and gradient descent

Task 3.1 Download the Jupyter notebook Auto_linear_regression.ipynb and open it. Alternatively, you can open the notebook on Google Colab. Work through the notebook. You can write your answers to the questions in the notebook below. ○

Answer:

3.2 Backpropagation in toy model

Now, we will start investigating the backpropagation algorithm. We will do so by examining the Toy model presented in Section 7.3 of the UDL book.

Task 3.2 Download the Jupyter notebook 7_1_Backpropagation_in_Toy_Model.ipynb and open it. Alternatively, you can open the notebook on Google Colab. Work through the notebook. ○

3.3 Backpropagation in neural network model

Now, we will proceed with the backpropagation algorithm for a fully connected neural network model.

Task 3.3 Download the Jupyter notebook backpropagation.ipynb and open it. Alternatively, you can open the notebook on Google Colab.

Work through the notebook. When done, evaluate the code for different values of hidden units `K`, neurons per layer `D`, input dimension `D_i`, and output dimension `D_o`. You find these parameters at the beginning of the notebook. ○

3.4 Backpropagation for multiple data points

The code only runs the backpropagation algorithm for one data point, and you should now extend it to handle multiple data points. We now want our code to compute the derivative of the total *cost* with respect to our weights and biases

$$\frac{\partial L}{\partial \boldsymbol{\beta}_k} = \frac{1}{I}\sum_{i=1}^{I}\frac{\partial \ell_i}{\partial \boldsymbol{\beta}_k} \qquad\qquad \frac{\partial L}{\partial \boldsymbol{\Omega}_k} = \frac{1}{I}\sum_{i=1}^{I}\frac{\partial \ell_i}{\partial \boldsymbol{\Omega}_k}$$

You should not add any additional `for` or `while` loops over data points or hidden units, but rather use the equivalent matrix/vector operations. Therefore, you should decide which dimension indexes your data points and be consistent with that choice.

By choosing the second dimension (columns) representing the data index, the code requires only a minor modification. That means that the entries in the lists of activations, pre-activations `all_f` and `all_h`, corresponding derivatives `all_dl_df` and `all_dl_dh` all represent matrices, with one column for each data point.

Task 3.4 Make a copy of the notebook in Task 3.3 and extend it to handle multiple data points. Choose the second dimension (columns) to index the data points. Evaluate your code by changing the parameter `n_samples` $= 1$ to something greater than 1. Which line(s) in `forward_pass` and `backward_pass` do you need to change? Why do some lines not require any change at all? After doing the extension, make sure that the derivatives of weight matrices and biases still match up with the finite-difference approximation! ○

Hint: If you don't know where to start, just run your existing code with `n_samples` larger than 1 and debug from there based on your response.

In machine learning and Python, it is common to reserve the first dimension for indexing the data points (i.e., one row equals one sample).

Task (optional) 3.5 Modify your code in Task 3.4 such that the first dimension (rows) indexes the data points. This essentially requires transposing several of the questions. Evaluate your code in the same manner as above. ○

3.5 Backpropagation with softmax output and cross-entropy loss

The code only runs the backpropagation algorithm for the least square loss.

Task 3.6 Make a copy of the notebook in Task 3.4 (or Task 3.5).

- Implement a function `softmax` that computes the softmax for every network output. See Equation (4).
- Add or replace the `compute_cost` with the cross-entropy loss with softmax. See Equation (7).
- Add or replace the `d_cost_d_output` with the derivative of cross-entropy loss with softmax. See Preparatory Exercise2.4.

After doing the extension, make sure that the derivatives of weight matrices and biases still match up with the finite-difference approximation! ○

Note 1: Your implementation should work for multiple data points, i.e., you need to (also) sum over the data point dimension in `compute_cost`.

Note 2: The notation in the preparatory exercises is written with the first dimension representing data points. Make sure that your implementation is consistent with your choice of data point dimension.

Note 3: Preferably, normalize the cost (and its derivative!) with the number of data points, as we did for the least squared loss.

Note 4: To evaluate the code, we need to define a (random) output `y` in the same manner as the code does for evaluating the squared loss. Now, each column should be a one-hot encoded vector (if you use the column to index data points). This can (for example) be done with the following code line

```
y = np.eye(D_o)[np.random.choice(D_o, n_samples)].T
```

(or without the transpose if you did Task 3.5 and used rows as data point dimension).

Note 5: Output dimension `D_o` has to be larger than 1 for this cost function to make any sense (otherwise, we only consider one class!).

3.6 Moving forward for Hand-in assignment 1

Task 3.7 Read through the instructions for Hand-in Assignment 1 and identify the steps that need to be taken to complete that assignment based on the code you have created in this lab. List these items below, along with possible questions you might have for the teaching assistant related to this. Note, solution to Hand-in Assignment 1 shall be handed in individually! ◦

Answer: