TDT4171 2021 Assignment 5

TDT4171

Feed-forward neural network

- **Delivery deadline: April 28, 2021 by 23:59 sharp**.
- Required reading for this assignment: Chapter 18[1] (parts of the syllabus) and the slideset for Lecture 8.
- Deliver your source code (a single .py file) on *Blackboard*. Please **do not** put it into an archive (e.g., .zip).
- Students **cannot** work in groups. Each student can only submit the assignment individually.
- This assignment totals 10 points. Point allocation is described in Table 1.
- Copying ("koking") from other students is not accepted, and if detected, will lead to immediate failure of the course. The consequence will apply to both the source and the one cribbing.
- For help and questions related to the assignment, ask the student assistants during the guidance hours or use Piazza. The guidance hours and link to Piazza can be found under "Assignments" on Blackboard. For other inquires, an email can be sent to tdt4171@idi.ntnu.no.

1 Feed-forward Neural Network

In this assignment, you will implement a feed-forward neural network (NN) with one hidden layer that supports binary classification (shown in Figure 1b). The input layer should be able to take n (arbitrary number) features. The hidden layer should support sigmoid activation functions (see Figure 2a) and 25 hidden units. Unlike the decision tree from Assignment 4, a NN is a parametric model meaning that it learns from data by optimizing some parameters $\vec{\boldsymbol{w}}$ guided by a loss function

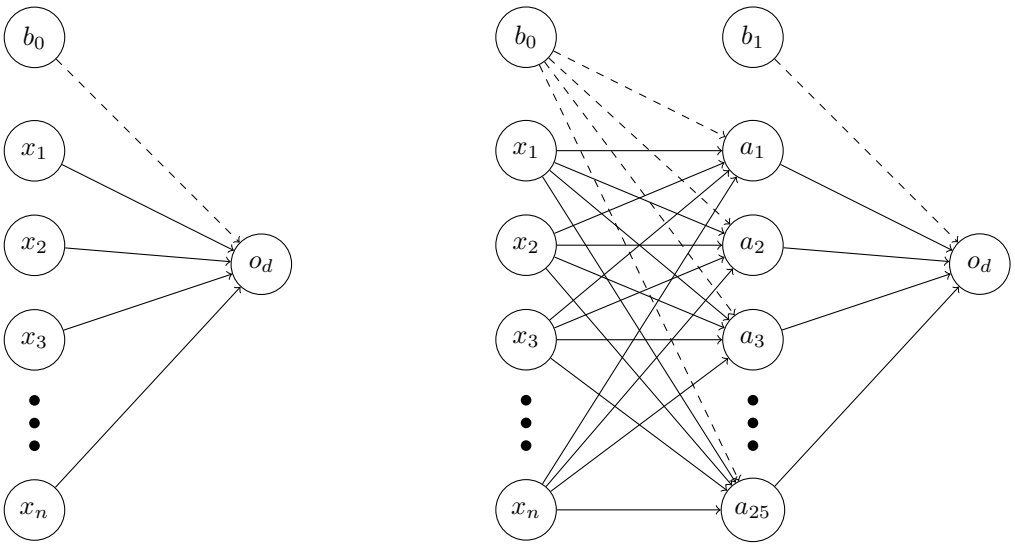
$$\mathbb{E}[\vec{\boldsymbol{w}}] = \frac{1}{2}\sum_{d\in\mathcal{D}}(t_d - o_d)^2 \tag{1}$$

where $\mathcal{D}$ is set of training examples, each of the form $d = \langle \boldsymbol{x}_d, t_d \rangle$. We will refer to this process as training. To train a feed-forward NN with one hidden layer, you will need to partially[1] implement the backpropagation algorithm outlined in Figure 18.24[1]. Be aware that Figure 18.24 has an error. Line 7 and 8: "**for each** weight $w_{i,j}$ in *network* **do**; $w_{i,j} \leftarrow$ a small random number" should come before line 6: "**repeat**". Although the goal is to implement a feed-forward NN with one hidden layer, you will still get partial points if only a perceptron is implemented (see Figure 1a). We will allocate points for this assignment according to how many of the functionalities listed in Table 1 are implemented. Each of the tasks has its own unit test that can be used to assess whether the functionality is implemented correctly or not.

Functionality	**Points**	**Unit test**
Part 1: Perceptron (Figure 1a)	5	`test_perceptron()`
Part 2: One hidden layer with 25 hidden units and sigmoid activation function (Figure 1b)	5	`test_one_hidden()`

Table 1: Point allocation. Functionality describes what needs to be implemented to get the points. With each functionality, we have provided a unit test that you can use to assess if the functionality is correctly implemented.

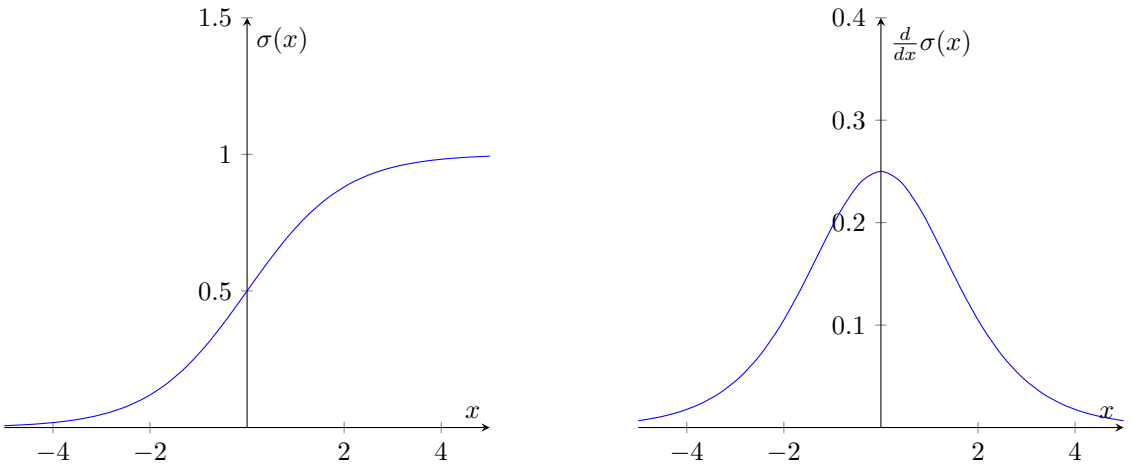
[1] We say partially because Figure 18.24 generalizes the backpropagation algorithm to arbitrary NN sizes.



(a) A perceptron

(b) A feed-forward NN with one hidden layer and 25 hidden units

Figure 1: The circles represent neurons, and a collection of neurons on the same level is named a layer. b_0, b_1 are the bias neurons.



(a) Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

(b) The derivative of the sigmoid function $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$

Figure 2: The activation and its derivative you are going to use for this assignment.

1.1 Implementation

We will provide the skeleton code of a NN for you to implement a feed-forward NN. Please use Python 3.8 or newer (`https://www.python.org/downloads/`). The skeleton code contains unit tests to help you assess whether the tasks are implemented correctly or not. Please run them to see if your implementation works before delivering your code. However, note that the unit tests are not a guarantee for correctness because there can still be edge cases the unit tests do not cover.

We will now provide an overview of the skeleton code; more details can be found in the code. The constructor of the NN class takes two arguments:

input_dim. An integer specifying the input dimension of the NN. This is the same as the number of features in the dataset.

hidden_layer. A boolean specifying whether or not to initialize the NN with a hidden layer. We have added this option, so it should be possible to get points for implementing only a perceptron.

You will need to make changes to it. For example, assigning the arguments to variables and so forth.

Your task is then to implement the methods `train()` and `predict(x: numpy.ndarray)`.

train. This method should implement the backpropagation algorithm outlined in Figure 18.24[1] that is used to train NNs. To run the backpropagation algorithm, some hyperparameters, such as learning rate, are needed. We have given all the necessary hyperparameters in the constructor of the NN class.

predict. This method should take an example ($\boldsymbol{x}_d$, a NumPy vector) and classify it by outputting a prediction. The classification involves passing the example through the NN. This is known as the forward pass and involves sending the example $\boldsymbol{x}_d$ through the network from input to the output layer.

1. At the hidden layer, multiply the input by the weights of the hidden layer and then add in the bias. Finally, apply the sigmoid activation function to the sums to produce the outputs. This step applies only if you implement a NN with a hidden layer (see Figure 1b).
2. At the output layer, multiply the output of the input/hidden layer with the weights of the output layer and then add in the bias. Apply the sigmoid activation function to the sum to produce the output. The output should be a scalar value between 0 and 1.

You are free and encouraged to implement additional methods and classes to support your implementation as long as the methods `train()` and `predict()` are implemented as intended and the unit tests pass. To implement the NN, it can help to implement classes for layer and neuron.

Note

Using implementation of NNs from machine learning and deep learning libraries such as Scikit-learn[2] and Keras[3] are not allowed (fully or partially). Furthermore, it is not allowed to copy any code from the internet (using information from Piazza is allowed). However, supporting libraries, such as NumPy[4] is necessary and allowed.

Read the entire assignment, the comments in the provided skeleton code, Chapter 18[1] (parts of the syllabus) and the slideset for Lecture 8 before writing any code. The book provides a high-level overview of the backpropagation algorithm. On the other hand, the lecture slides explicitly

derives the equations you need to implement the NN. Hence, it is essential to read all of the teaching materials.

The code must be runnable without any modifications after delivery. Moreover, the code must be readable and contain comments explaining it. This is especially important if the code does not work. Finally, do not archive your source file when delivering on *Blackboard*. Failing to follow the instructions may result in points being deducted.

1.2 Datasets

We will now provide an overview of the dataset you will use to train and test the NN. You will use the breast cancer Wisconsin dataset[5] (`https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`) for this assignment. The dataset can be found on Blackboard under "Assignments" along with this assignment text. This is a binary classification dataset (see Table 2 for its content). We have preprocessed the dataset and provided the method to load the dataset `load_data(file_path: str)` in the skeleton code. By default, the method assumes that the dataset is in the current working directory. However, you can change the path by providing the method with a different `file_path` argument, which is a string. The data will be loaded into four different variables:

self.x_train A matrix (`numpy.ndarray`) where each row represents one example and each column a feature. You will need to use this to implement the `train()` method.

self.y_train A vector (`numpy.ndarray`) of labels for the examples in `self.x_train`. The vector's length is the same as the number of rows in `self.x_train`. You will need to use this to implement the `train()` method.

self.x_test Same format as `self.x_train` used in the unit tests. You do not need to touch this.

self.y_test Same format as `self.y_train` used in the unit tests. You do not need to touch this.

More details are given in the skeleton code on how to use the dataset.

Attribute	Value
Classes	2
Examples per class	212 (Malignant), 357 (Benign)
Examples total	569
Dimensionality (number of features)	30

Table 2: Properties of the breast cancer Wisconsin dataset.

References

[1] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Always learning. Pearson, 2016.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.