INF264 - Homework 4

Pierre Gillot **Natacha Galmiche**[*]

Week 37 - 2021

1 Gradient descent and backpropagation in a simple neural network

In this first section, we will perform by hand the learning process of a simple neural network, and see that doing so reduces the loss function of the network.

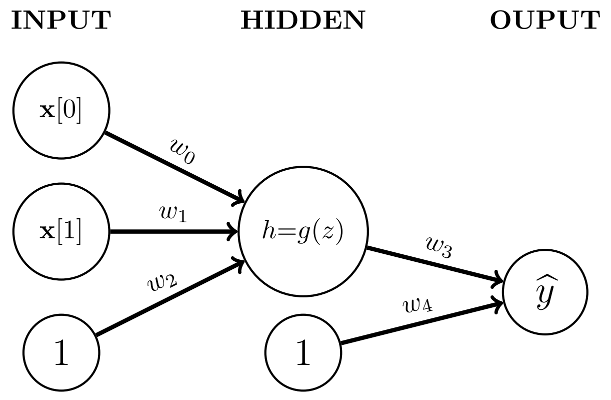
Usually learning a neural network on some data consists of two steps:

i A forward pass in which the data is processed in the neural network from the input layer to the output layer.

ii A backward pass in which we propagate the error from the output layer to the input layer in order to eventually recover an estimation of the loss function's gradient (backpropagation algorithm).

Finally, a gradient descent method is used on the network's parameters (weights and biases) in order to minimize the loss.

Consider the following neural network:



[*]Not a TA for this course this year.

where:

- $\mathbf{x} = (\mathbf{x}[0], \mathbf{x}[1]) \in \mathbb{R}^2$ is a feature vector from the data and represents the input layer.
- $\theta = (w_0, w_1, w_2, w_3, w_4)$ are the parameters of the network. Note that w_2, w_4 represent the biases, while w_0, w_1, w_3 represent regular weights.
- g is the ReLU activation function, defined as
$$t \mapsto \max(0, t) \quad \forall t \in \mathbb{R}.$$
- z is the input of the hidden layer:
$$z(\mathbf{x}) = w_0 \cdot \mathbf{x}[0] + w_1 \cdot \mathbf{x}[1] + w_2 \cdot 1.$$
- h is the output of the hidden layer:
$$h(\mathbf{x}) = g\big(z(\mathbf{x})\big).$$
- $\widehat{y}$ is the output of the network, the prediction of $\mathbf{x}$:
$$\widehat{y}(\mathbf{x}) = w_3 \cdot h(\mathbf{x}) + w_4 \cdot 1.$$

We define the loss function of this neural network as the mean-squared error, i.e. for a dataset $\mathcal{D}$ containing the feature vectors $\mathbf{x_1}, \ldots \mathbf{x_N}$ and their respective targets $y_1, \ldots, y_N$, the loss function $\mathcal{L}$ on this data is defined as:

$$\mathcal{L}(\mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} L(\mathbf{x_n}, y_n),$$

where

$$L(\mathbf{x_n}, y_n) = \big(\widehat{y}(\mathbf{x_n}) - y_n\big)^2.$$

The gradient of the neural network's loss function with respect to the network's parameters θ on the dataset $\mathcal{D}$ is then:

$$\nabla_\theta \mathcal{L}(\mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta L(\mathbf{x_n}, y_n),$$

where

$$\nabla_\theta L(\mathbf{x_n}, y_n) = \left(\frac{\partial L(\mathbf{x_n}, y_n)}{\partial w_0}, \frac{\partial L(\mathbf{x_n}, y_n)}{\partial w_1}, \frac{\partial L(\mathbf{x_n}, y_n)}{\partial w_2}, \frac{\partial L(\mathbf{x_n}, y_n)}{\partial w_3}, \frac{\partial L(\mathbf{x_n}, y_n)}{\partial w_4} \right).$$

Once the gradient of the neural network's loss function with respect to the parameters θ is computed, a gradient descent method is applied to update the parameters in an attempt to minimize the loss:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}(\mathcal{D}),$$

where $\eta > 0$ is the learning rate of the gradient descent.

The critical aspect when learning a neural network is thus to compute efficiently the loss gradient $\nabla_\theta L(\mathbf{x}, y)$ for each individual sample $(\mathbf{x}, y)$ in the training set $\mathcal{D}$, and is the purpose of the backpropagation algorithm, which works as follows:

For a given sample $(\mathbf{x}, y)$, define the error in the output layer and the error in the hidden layer as

$$\delta_{\widehat{y}}(\mathbf{x}, y) = \frac{\partial L(\mathbf{x}, y)}{\partial \widehat{y}}, \quad \delta_h(\mathbf{x}, y) = \frac{\partial L(\mathbf{x}, y)}{\partial z} \quad \text{respectively.}$$

We can show that the error in the hidden layer propagates from the error in the output layer. Indeed, L is a function of $\widehat{y}$, $\widehat{y}$ is a function of h and h is a function of z, so the chain rule yields:

$$\begin{aligned}
\delta_h(\mathbf{x}, y) &= \frac{\partial L(\mathbf{x}, y)}{\partial z} \\
&= \frac{\partial L(\mathbf{x}, y)}{\partial \widehat{y}} \cdot \frac{\partial \widehat{y}(\mathbf{x})}{\partial h} \cdot \frac{\partial h(\mathbf{x})}{\partial z} \\
&= \delta_{\widehat{y}}(\mathbf{x}, y) \cdot \frac{\partial \big(w_3 \cdot h(\mathbf{x}) + w_4 \cdot 1\big)}{\partial h} \cdot \frac{\partial g\big(z(\mathbf{x})\big)}{\partial z} \\
&= \delta_{\widehat{y}}(\mathbf{x}, y) \cdot w_3 \cdot g'\big(z(\mathbf{x})\big),
\end{aligned}$$

where g' is the left derivative of the ReLU activation g, defined as

$$t \mapsto \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t \leq 0. \end{cases}$$

Applying the chain rule also gives the partial derivatives of L with respect to the network's parameters:

$$\frac{\partial L(\mathbf{x}, y)}{\partial w_0} = \delta_h(\mathbf{x}, y) \cdot \mathbf{x}[0] \qquad (1)$$

$$\frac{\partial L(\mathbf{x}, y)}{\partial w_1} = \delta_h(\mathbf{x}, y) \cdot \mathbf{x}[1] \qquad (2)$$

$$\frac{\partial L(\mathbf{x}, y)}{\partial w_2} = \delta_h(\mathbf{x}, y) \qquad (3)$$

$$\frac{\partial L(\mathbf{x}, y)}{\partial w_3} = \delta_{\widehat{y}}(\mathbf{x}, y) \cdot h(\mathbf{x}) \qquad (4)$$

$$\frac{\partial L(\mathbf{x}, y)}{\partial w_4} = \delta_{\widehat{y}}(\mathbf{x}, y) \qquad (5)$$

In all that follows, we consider the dataset $\mathcal{D}$ containing two samples:

i $\mathbf{x_1} = (2, 1) \quad y_1 = 1.3$

ii $\mathbf{x_2} = (-3, 2) \quad y_2 = 1.9$

We also assume that the parameters of the network are initialized as

$$\theta = (w_0, w_1, w_2, w_3, w_4) = (1, 1, 0, 1, 0).$$

Once you have read carefully the explanations above, answer the following questions:

1. Perform a forward pass on the dataset $\mathcal{D}$: compute the predictions $\widehat{y}(\mathbf{x_1})$ and $\widehat{y}(\mathbf{x_2})$.

2. Compute the loss of the network on the dataset $\mathcal{D}$, that is $\mathcal{L}(\mathcal{D})$.

3. Prove the backpropagation equalities (1), (2), (3), (4), (5) using the chain rule.
   Hint 1: For the first three equalities, justify that you can use the chain rule
   $$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_i}, \quad i \in \{0, 1, 2\}.$$
   Hint 2: For the two last equalities, justify that you can use the chain rule
   $$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \widehat{y}} \cdot \frac{\partial \widehat{y}}{\partial w_i}, \quad i \in \{3, 4\}.$$
   Hint 3: If you struggle, ask for our help and do not get stuck for hours on this !

4. Compute the gradient of the neural network's loss function, that is $\nabla_\theta \mathcal{L}(\mathcal{D})$.
   Hint 4: Compute first the gradients $\nabla_\theta L(\mathbf{x_1}, y_1)$ and $\nabla_\theta L(\mathbf{x_2}, y_2)$ using the backpropagation rules, then compute their mean.
   Hint 5: During backpropagation, you should first compute the error in the output layer, then propagate this error to obtain the error in the hidden layer, and finally use the equalities (1), (2), (3), (4), (5) to get the gradients.

5. Perform the gradient descent update on the parameters $\theta = (w_0, w_1, w_2, w_3, w_4)$, with a learning rate $\eta = 0.01$.

6. Recompute the loss of the network on the dataset $\mathcal{D}$ and check that it decreased after the network's parameters were updated.

7. Intuitively, what it the effect of the learning rate hyper-parameter η on the update of the parameters θ ?

2 Neural networks in Keras

In this second section, we will get familiar with Keras, a deep learning high-level library with a strong focus on enabling fast experimentation. More information

about Keras can be found following this link: `https://keras.io`. Keras is now part of TensorFlow (a deep learning low-level library achieving state-of-the-art performance) and can be installed in python very easily using the command **'conda install tensorflow'**.
We will use Keras in order to manipulate simple neural networks. More specifically, we will build a MLP (Multi-Layer Perceptron) that consists of a few dense layers, and we will assess its performance on the Wine dataset. Finally, we will perform model selection of our MLP on a set of hyper-parameters.

Note that you can use the provided template code.

1. Load the Wine dataset using the 'sklearn.datasets.load_wine' function. Alternatively, you can find the Wine dataset following this link: `https://archive.ics.uci.edu/ml/datasets/wine`.
2. Preprocess the data:
   (a) Center and normalize the features (you can use the function 'sklearn.preprocessing.scale').
   (b) Transform the target labels into one-hot vectors (you can use the function 'tensorflow.keras.utils.to_categorical').
   (c) Split the data into a train, a validation and a test sets (indicate the ratios of your split).
3. Build a simple MLP in Keras:
   (a) Create a 'tensorflow.keras.Sequential' model, consisting of 2 'tensorflow.keras.layers.Dense' layers. The first layer (the hidden layer) must have 10 units and a ReLU activation. The second layer (the output layer) must have one unit for each class in the dataset and a softmax activation.
   (b) Set the loss function to 'categorical_crossentropy', the optimizer to 'SGD' (stochastic gradient descent) and the metric to 'accuracy' (use the 'compile' method of your Keras model).
4. Display your MLP architecture using the 'summary' method of your Keras model.
5. Train and validate your MLP model on the Wine dataset:
   (a) Use the 'fit' method to train your model, you can train for 20 epochs with a batch size of 16. **Note:** You can store the output of the 'fit' method in a variable to get access to the training and validation loss and accuracy values throughout training.
   (b) Print the train/validation accuracy and loss curves.
6. Evaluate your model on the test set. Comment on the achieved performance of your model.
7. Model Selection:

(a) Perform model selection of your MLP model on a set of hyper-parameters containing at least the depth of the MLP (implement a flag that adds an additional hidden layer to your model when set to true) and the activation type of the hidden layers in the MLP ('sigmoid' or 'relu').

(b) Try to add more hyper-parameters (in this case, specify them in your report).

(c) Evaluate the performance of the best model on the test set.