Exercises
Computational Intelligence Lab
SS 2019

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Prof. Dr. Thomas Hofmann
Web http://cil.inf.ethz.ch/

Series 8, April 11-12, 2019 (Neural Networks)

Problem 1 (Linear MLP):

In this exercise we consider a linear multilayer perceptron as the ones below (MLP1 and MLP2).
In a linear MLP each unit has a linear activation function, such as the identity function $g : \mathbb{R} \to \mathbb{R}$, $g(\mathbf{x}) = \mathbf{x}$.
For a vector $x \in \mathrm{R}^2$, $x = (x_1, x_2)^T$, and the identity as the activation function, the output of MLP1, o', is equal to :

$$o' = w_1 \times x_1 + w_2 \times x_2.$$

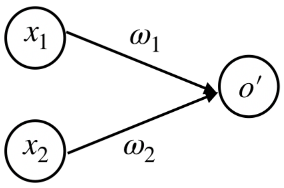


Figure 1: MLP1

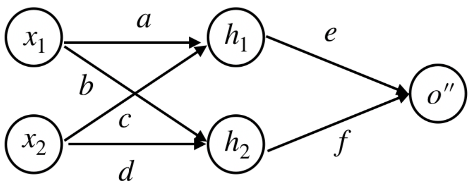


Figure 2: MLP2

1. Prove that for any $a, b, c, d, e, f \in \mathbb{R}$ in MLP2, there always exist $\omega_1, \omega_2 \in \mathbb{R}$ in MLP1 such that the two outputs, o' and o'', are the same.
2. Consider a vector $\mathbf{x} = (x_1, x_2)$ where $x_1, x_2 \in \{0, 1\}$, prove that a linear MLP cannot learn the XOR function of its input.
   Note: XOR$[x_1, x_2]$ is equal to 1 if $x_1 \neq x_2$ and 0 otherwise.

Problem 2 (Neural Networks & Backpropagation):

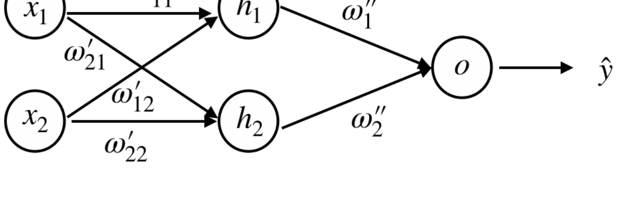
1. Consider the sigmoid function $s(x) = \frac{1}{1+e^{-x}}, x \in \mathbb{R}$ acting element-wise on a vector $\mathbf{x}$, which is a common activation function used in neural networks. Prove that

$$\nabla_x s(x) = s(x)(1 - s(x)).$$

2. Consider a binary classification problem for which we are given an input vector $\{\mathbf{x}_i\}_{i=1}^n$ and we want to predict an output variable $y_i = \{0, +1\}$ for each input $\mathbf{x}_i$. We use a neural network such as the one below, with parameters $\mathbf{w}'$, $\mathbf{w}''$, a sigmoid activation function for both the hidden layer and output layer and a cross-entropy loss function for each data point $\mathbf{x}_i$ defined as

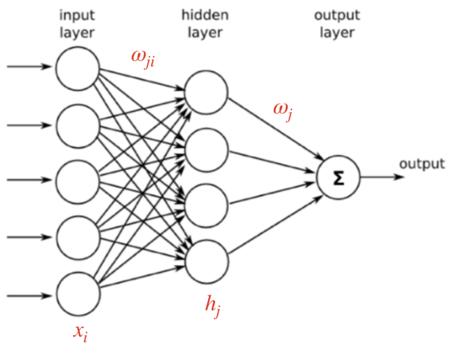
$$L(\mathbf{w}', \mathbf{w}'') = -y_i \log \hat{y}_i(\mathbf{x}_i; \mathbf{w}', \mathbf{w}'') - (1 - y_i) \log(1 - \hat{y}_i(\mathbf{x}_i; \mathbf{w}', \mathbf{w}'')), \tag{1}$$

where $\hat{y}_i(\mathbf{w}', \mathbf{w}'') \in [0, 1]$ is the output of the neural network.



Using the forward pass derive the loss function for an input $\mathbf{x} = (0.2, 0.8)$ and weights $w'_{11} = w'_{22} = 0.1$, $w'_{12} = 0.2$, $w'_{21} = 0.3$, $w''_1 = w''_2 = 0.2$ and for the target label $y = 0$.

3. Consider a single layer network as illustrated in the figure below. The input layer takes the vector $\mathbf{x} \in \mathbb{R}^d$, which is composed of d components $\{x_i\}_{i=1}^d$, while the hidden layer consists of m neurons $\{z_j\}_{j=1}^m$. The weights between the input and hidden notes are denoted by w_{ji}, while the ones between the output of the hidden layer and the final output are denoted as ω_j. The activation function of the hidden layer is a sigmoid function. There is one output neuron which consists of a weighted sum of the hidden layer output.



   (a) Write down the expression of the hidden layers h_i and the ouput, $\hat{y}$, as a function of the weights and the inputs.
   (b) Assuming the error function is the mean-squared error (MSE), compute the gradient of the error function E with respect to the weights w_j and the weigths w_{ji}, (consider only one single training example n).
   (c) Finally compute the gradient of the error function E with respect to the input x_i.

Problem 3 (Convolutional Neural Networks):

In this exercise we consider a convolutional neural network (CNN) with one convolutional layer, followed by a non-linearity ReLU, and then followed by a 3x3 max-pooling with stride 1, on top of which we apply a softmax. Assume that this small CNN takes as input images with two channels, and that the convolutional layer has two filters, each filter containing an odd number of pixels, so that we can index it with pixels (i, j) for $-k \le i, j \le k$. Further assume that convolutions/max-pooling are performed with *zero-padding*, i.e. by completing I with zeros in such a way that the convolution of I by a filter has the same size as I.

1. Write down the mathematical expression of the convolutional layer.

2. Write down the mathematical expression of the ReLU non-linearity and the max-pooling. What do you get by composing it with the previous question? This is the signal you get just before applying the softmax.

   In the following questions, consider a 2-channels image $I = (I_c)_{1 \le c \le 2}$, where each I_c is of size $4 \times 4 = 16$ pixels. Assume that each of the two filters K^l, for $l = 1$ or $l = 2$, is a 2-channel image of size $3 \times 3 = 9$ pixels per channel. We choose the image to have pixel values given by $(I_c)_{i,j} = ij + c$, and the filters to be defined as

$$K_c^l = \begin{pmatrix} 0 & l & 0 \\ l & -c & l \\ 0 & l & 0 \end{pmatrix},$$

   where $c \in \{1, 2\}$ is the channel and $l \in \{1, 2\}$.

3. Compute the two 4x4 1-channel convolved images $I \star K^l$ for $l \in \{1, 2\}$. (Computing all coefficients by hand is tedious but do at least the first 4 ones, in the top left corner).

4. Write down the result of applying a non-linearity ReLU to each of the convolved images.

5. Apply the 3x3 max-pooling to each of the two images obtained in the previous question (compute at least the first coefficient in the top left corner).

Problem 4 (Getting Started with TensorFlow):

For the implementation exercises (and your project), we suggest that you use one of the many machine learning frameworks. These frameworks allow building complex models by means of simple building blocks (e.g. fully-connected layers, convolutional layers, activation functions), which form a *computational graph*. Most importantly, they only require you to specify the forward function, as the framework will take care of computing the gradients – a feature called *automatic differentiation* – and optimizing the model parameters.

The two leading frameworks are TensorFlow and PyTorch. They are almost equivalent in terms of features, so their use is a matter of preference. For this exercise, we provide resources for TensorFlow.

Set up TensorFlow on your system, by following

https://www.tensorflow.org/install

The workflow for TensorFlow is as follows:

1. Specify your model using variables (parameters) and placeholders (data).
2. Specify your loss function (usually some form of distance between the model output and the desired output).
3. Specify a built-in optimization algorithm to find parameters that minimize the loss (e.g. gradient descent).
4. Split the data into a training set and a test set, and start a TensorFlow `Session` to minimize the loss using the training set. During training, you can monitor the value of the loss function (and the accuracy, if you are working on a classification problem) by printing the results as training evolves, or by using *TensorBoard*.
5. Evaluate the model on the test set.

Get the provided code template `exercise08.ipynb` and follow the instructions.

You are given two toy datasets: `blobs` and `circles`. Your goal is to build a classifier that can assign an unseen data point to the correct class.

1. Randomly split the data into a training set (80%) and a test set (remaining 20%).
2. Let's start with a simple linear model. Write a logistic classifier to classify `blobs`. In this case you should optimize two variables $\mathbf{W}$ (weights) and $\mathbf{b}$ (biases) using gradient descent. Plot the decision boundary using the provided code template. Does it fit the data properly? Why?
   Hint: the probability of an item having $y = 1$ is $\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where σ is the sigmoid function.
   Hint2: a suitable loss function in TensorFlow is the binary cross entropy
   ($tf.nn.sigmoid\_cross\_entropy\_with\_logits$), which takes care of computing both the sigmoid and the loss (cross entropy) in a single step for numerical stability.
3. Do the same for the other dataset, `circles`, and discuss the results.
4. Now we construct a simple neural network by adding one hidden layer to our model. Experiment with various numbers of nodes and activation functions, and observe how the error/decision boundary evolves.
5. Replace full gradient descent with *stochastic* gradient descent, meaning we do not want to use the whole dataset in each iteration, but just a random subset of it. TensorFlow does this by replacing the data variables with *placeholders*, which are then successively fed with a subset of the data by means of the *feed dictionary* parameter. You do not actually need to change the optimizer – you should only work on the way you preprocess the data fed to the optimizer. Experiment with different learning rates and minibatch sizes. Does the algorithm converge faster? What are the advantages of each approach?

Problem 3 (Basic Classification):

Go through the TensorFlow tutorials and try “Basic Classification”.

https://www.tensorflow.org/tutorials.

Recently, Keras (which was formerly a third-party framework) has been integrated into TensorFlow, and it provides an easy-to-use interface for most purposes. However, if you want to implement non-standard architectures, you might still need to use the low-level TensorFlow API. For reference, here is an old tutorial on image classification (CIFAR-10 dataset):

https://www.tensorflow.org/tutorials/images/deep_cnn.