Exercises
Computational Intelligence Lab
SS 2017

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

Series 8, May 4-5, 2017 (Neural Networks)

Problem 1 (Neural Networks):

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

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

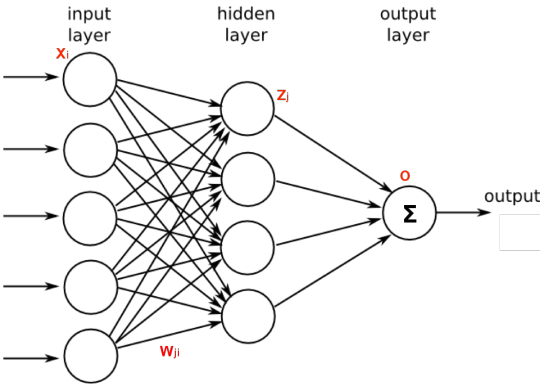
2. Consider a binary classification problem for which we are given a set of input vectors $\{\mathbf{x}_i\}_{i=1}^n$ and we want to predict an output variable $y_i = \{0, +1\}$ for each input $\mathbf{x}_i$. We use a neural network with parameters $\mathbf{w}$ and a cross-entropy loss function defined as

$$H(\mathbf{w}) = \sum_{i=1}^{n} -y_i \log \hat{y}_i(\mathbf{x}_i; \mathbf{w}) - (1 - y_i) \log(1 - \hat{y}_i(\mathbf{x}_i; \mathbf{w})), \tag{1}$$

   where $\hat{y}_i(\mathbf{w}) = \Pr(\mathbf{x}_i; \mathbf{w})$ is the output of the neural network.

   Derive the gradient of $H(\mathbf{w})$ with respect to the parameters $\mathbf{w}$.

3. Consider a single layer network as illustrated in the figure below. The input layer consists of d inputs $\{x_i\}_{i=1}^d$ 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}. There is one output neuron which consists of a logistic activation function.



   Assuming the error function is the mean-squared error (MSE), compute the gradient of the error function E with respect to the weights w_{ji} (consider only one single training example n).

Problem 2 (Getting Started with TensorFlow):

Set up TensorFlow on your system, by following

https://www.tensorflow.org/get_started/.

Get the two provided code templates `ex1.py` and `ex2.py` from

github.com/dalab/lecture_cil_public/tree/master/exercises/ex8.

We implement a multi-valued linear regression model, where we are supposed to find the weights $\mathbf{W} \in \mathbb{R}^{d \times k}$ and $\mathbf{b} \in \mathbb{R}^k$ in

$$\frac{1}{n} \sum_{i=1}^{n} (\mathbf{X}_{i:} \mathbf{W} + \mathbf{b} - \mathbf{Y}_{i:})^2$$

for a set of n data examples given as the rows of the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, and the target observations given as the corresponding rows of $\mathbf{Y} \in \mathbb{R}^{n \times k}$.

We would like to find the optimal $\mathbf{W}$ and $\mathbf{b}$ by means of gradient descent. Tensorflow (as well as Torch, Theano, etc.) makes this very easy because of the built-in *autodifferentiation*. This feature allows differentiating arbitrary computation graphs. For simple (stochastic) gradient descent, TensorFlow even hides this differentiation by means of an *optimizer*. The optimizer will use the autodifferentiation to obtain the gradient of the loss with respect to $\mathbf{W}$ and $\mathbf{b}$ and then make a descent step each time its computation is triggered.

The workflow when using systems like TensorFlow is thus as follows:

1. Specify your model using variables (parameters) and placeholders (data).
2. Specify your loss (usually some form of distance between the model output and the desired output).
3. Use a built-in optimization algorithm to find parameters that minimize the loss.

For our example of multivariate linear regression, do the following tasks:

1. Fill in the model and loss function in `ex1.py`, and visualize the progress in the loss function during each step of gradient descent for the weight variables $\mathbf{W}$ and $\mathbf{b}$, by inspecting the tensorboard in the browser. We want to do full gradient descent, so just use the (numpy) dataset variables for $\mathbf{X}$ and $\mathbf{Y}$.
2. Fill in the model and loss function in `ex2.py`, and visualize the progress in the loss function during each step of stochastic gradient descent for the weight variables $\mathbf{W}$ and $\mathbf{b}$, by inspecting the tensorboard in the browser. Note that now we want to *stochastic* gradient descent, meaning we don't want to use the whole dataset in each iteration, but just a 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.
3. What difference do you see in the plots between full and stochastic gradient descent? What are advantages and disadvantages of each? Play with the step sizes, minibatch sizes and dataset noise settings. Can you find an instance where one method works, but the other one does not?

Problem 3 (MNIST For ML Beginners):

Go through the MNIST digit recognition tutorial where you will learn how about the MNIST dataset and how to create a softmax regression model in Tensorflow:

https://www.tensorflow.org/get_started/mnist/beginners.