Assignment 2: Neural Networks

MA-INF 2313: Deep Learning for Visual Recognition

Due Date: Sunday 29.11.2020

1 Theoretical Exercises (15 points)

1. *(6 pts)* Neural networks can be interpreted geometrically as a partitioning of the input feature space $\boldsymbol{x}$. Now consider the trapezoid in Figure 1, which represents a decision boundary, where the region inside belongs to class A and the region outside belongs to class B. How should one design a two-layered network[1] to represent this trapezoid? Fully specify your network, including the form of input, hidden and output activation functions as well as associated parameters.

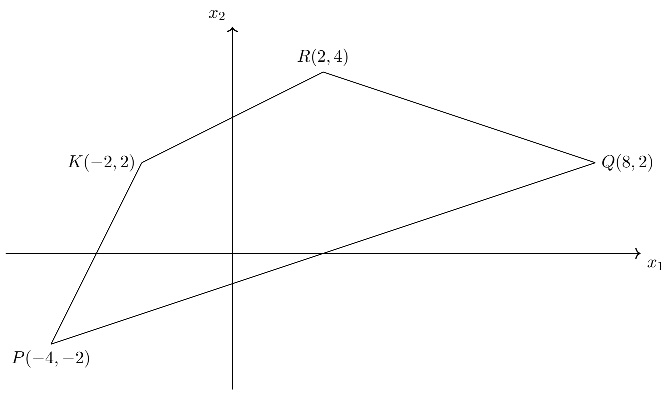

Figure 1: The region in the trapezoid belongs to class A and the other region to class B.

2. *(9 pts)* Consider the Multi-Layer Perceptron (MLP) with a single hidden-layer given in Figure 2 with two input units x_1, x_2, two hidden units h_1, h_2, two output units o_1, o_2 and additionally, the hidden and output bias units b_1, b_2.

   In this task you are going to work with a single training set: given inputs 0.1 and 0.4, we want the neural network to output 0.1 and 0.9. You will use the Sigmoid function as your activation function for each hidden layer unit h_1, h_2 and the Softmax function for the output units o_1, o_2.

   Run a forward pass to see if the neural network with the current weights predicts the outputs correctly. Show your work step by step for the forward pass. Calculate the total error in the network using mean squared error.

   Run backpropagation in this network and update each weight in the network. Show your work step by step for the backward pass. Later on, run a forward pass to see if the neural

[1]as per Bishop's notation, which results in one input layer, one hidden layer and one output layer.

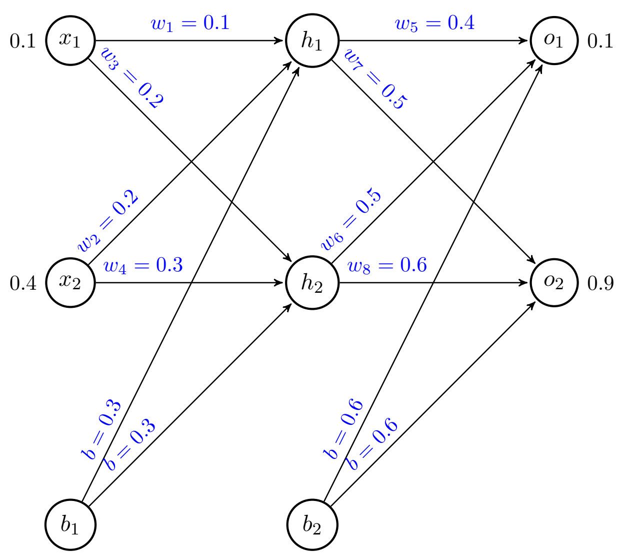

Figure 2: Multi-Layer Perceptron with a single hidden-layer

network with the updated weights predicts outputs better than in the first forward pass. (Set your learning rate to 0.5)

2 Programming Exercises *(20 points)*

1. *(14 pts)* Equipped with our newfound knowledge about MLPs we can try to improve the linear classifier we created on sheet 1. Instead of giving a binary answer, we instead create a net that has C output neurons, where C is the number of classes. The prediction of our model is the class of the neuron with highest activation.

   - *(2 pts)* Improve upon your linear classifier from sheet 1 by adding the possibility to create a variable number of (hidden) linear layers. The constructor should accept a tuple of numbers indicating the number of neurons per layer, e.g.

     ```
     model = FashionMNISTClassifier(num_neurons=(50,20))
     ```

     should generate an MLP with 28^2 input neurons, two hidden layers with 50 and 20 neurons respectively and an output layer with $C = 10$ neurons. You also have to adjust the activation function of the output layer and the loss metric for the multi-class setting.
   - *(6 pts)* Here are some ideas of how to best train the net with all classes. Implement and compare the following approaches:
     (a) Train the classes separately, i.e. filter the dataset and train for five epochs on all images depicting a 0, then on all images depicting a 1, and so on.
     (b) Sort the images by class and train them in order for 25 epochs. Make sure that the `DataLoader` is not shuffling the data.
     (c) Shuffle the images every epoch and train them for 25 epochs.

Report the accuracy for each approach on the whole test set after every epoch by showing them in a common plot. To make the plot align better, simply take every other epoch for approach (a). Make sure you create a new model and a new optimizer for each approach. Like on sheet 1, use the basic `SGD` optimizer with a learning rate of 0.01. Choose a batch size of 64 and the number of neurons as in the example above. Which approach works best and why?

- *(6 pts)* Evidently, the shuffling approach worked best for training. We could be inclined to call it a day now, but our MLP is still missing a crucial ingredient: Non-linear activation functions between hidden layers. Again, as we are not sure what activation function to choose, we will compare their performance.

  (a) For hyperparameter optimization, it might be best to use cross-validation as we did on sheet 1. Refactor you cross-validation code to work with the `DataLoader` we use here. Also parameterize the model initializer such that it accepts an activation function to be added between hidden layers.

  (b) Run a 5-fold cross-validation for the following activation functions:
    - identity function (no special activation)
    - Tanh
    - Sigmoid
    - ReLu
    - Leaky ReLu

  All of these functions are available as a `torch.nn.Module` in the `torch.nn` package. To speed up convergence, you should be able to run 10 epochs per split with a learning rate of 0.1.

  (c) We are not alone in our quest to find a good activation function. Researchers at Google Brain propose the *Swish* function in this article. Implement this function as a `torch.nn.Module` and compare its performance to the other functions above.

  (d) Create a plot showing the average accuracy per epoch for each activation function. Now check how the trained models perform on the actual test set. Which function performed best?

  (e) How does the accuracy of your MLP compare to the baseline nearest-neighbors classifier from sheet 1?

2. *(6 pts)* Using the lessons learned from the previous exercise, create an MLP to classify the CIFAR-10 dataset. Explain your choice of hyperparameters (number of layers/neurons, activation functions, learning rate, etc.) and report the accuracy on the test set. To compare your results to the current "world record", you can check `https://benchmarks.ai/cifar-10`. In a future exercise we will see that there are more sophisticated approaches that can greatly improve the performance e.g. for CIFAR-10.