# Artificial Neural Networks

Prof. Dr. Sen Cheng

Dec 9, 2019

## Problem Set 10: Deep Neural Networks

**Tutors:** Filippos Panagiotou (filippos.panagiotou@ini.rub.de), Thomas Walther (thomas.walther@rub.de)

1. **Fully connected (dense) networks.** Build and train a fully connected network with 4 layers. Note that *keras* does not have an explicit representation of the input layer. The input is passed directly to the first hidden layer. Make use of the Keras documentation (keras.io).

   (a) Import the 'fashion_mnist' dataset from the 'keras' library and load it. The data set contains 60 000 images with a dimension of 28x28 pixels. Visualize a few images to get a feel for the data. Vectorize the images for the training and test set.

   (b) Note that the images are in greyscale. Scale of the pixel values appropriately to the range [0.0;1.0]. Why is this necessary? (Hint: Think about the backward pass and what it does to the weights.)

   (c) Build a Sequential network. Add two `Dense` layers, with 10 units each and ReLU activation functions. Add another `Dense` layer with 10 units and a softmax activation function. The sizes of the hidden layers are flexible, but the size of the output layer must remain at 10 units. Why?

   (d) Calculate by hand how many trainable parameters are in your network (include the bias terms). Note it down for later.

   (e) Compile (`.compile()`) the network using Stochastic Gradient Descent (SGD) as an optimization method, the `sparse_categorical_crossentropy` loss function, and `accuracy` as metrics.

   (f) Train the network for 3 epochs. Use the test set for validation. Note down the accuracy metric on the training and test sets. Compile new networks and train them a few times to get a feel for the training process.

   (g) What learning rate did you use? Change the learning rate to 0.05 and train the network for a few times.

   (h) Increase the number of neurons in the hidden layers to 64. Calculate the number of trainable parameters and compare this and the network's accuracies to the values of the smaller one.

   (i) Reduce the training data to 20.000 items. Train a new network and look at its accuracy. Why does the test set accuracy diverge from the training set accuracy? Does it matter for us? Overfitting

   (j) Using the trained network, find all elements of the test set that the model predicts to be ankle boots (label= 9). Visualize the retrieved items. Use a display loop to do that. Stop the loop whenever an item has been classified incorrectly. Visually inspect the incorrectly classified items. Give a reason for those items being mis-classified as ankle boots.

2. **Convolutions.**

   (a) Given two 'image' matrices:

$$\mathbf{I}_A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{I}_B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and two filter (kernel) matrices:

$$\mathbf{K}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \mathbf{K}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

manually compute the 2d convolutions $\mathbf{K}_i * \mathbf{I}_i$ for $i = x, y$ and $j = A, B$ on paper. Assume that the image matrices are padded with the appropriate number of additional rows and colums, which are filled by repeating their current border (you can compare with 'nearest' method in 'scipy.ndimage.convolve').

(b) Inspect the results of the convolutions: What kind of features do the filter kernels extract?

3. **Convolutional Neural Networks (CNN).** Build and train a 6-layer CNN using the *keras* library.

   (a) Proceed as in problem 1, except for the following two points:

      i. Change the dimensionality of your data appropriately so that they can serve as input for a CNN, i.e. $28 \times 28 \times 1$. padding='valid'

      ii. Instead of the two hidden dense layers, use two consecutive pairs of 2-d convolution (`Conv2D`) and max-pooling (`MaxPooling2D`) layers. Use `Conv2D` with 16 and 8 filters of size of 3x3 respectively, and ReLU activation function, and `MaxPooling2D` layers with a 2x2 receptive field.

   (b) Compile and train the network as in problems 1f–1h. Compare the final accuracies. Does the discrepancy between training and test error remain? What are the reasons for this? No. #parameters is smaller => less risk of overfitting

   (c) Calculate the number of trainable parameters. Compare with the number of parameters for the fully connected network above. 9*16+16 + 9*8*16+8 + (5*5*8 *10+10) = 3,330

4. **BONUS.** Modify the network from problem 3 according to the following instructions and observe how the training time, the training accuracy and the test accuracy change.

   (a) Remove the two `MaxPooling2D` layers and add a stride of 2 to the `Conv2D` layers. Train this network and compare epoch speed. Why the speedup? #convolution operations is less

   (b) Use the ADAM optimizer in place of the SGD. Why the increase in accuracy?