Artificial Neural Networks

Prof. Dr. Sen Cheng

Dec 16, 2019

Problem Set 11: Deep Neural Networks 2

Tutors: Eloy Parra Barrero (eloy.parrabarrero@rub.de), Nicolas Diekmann (nicolas.diekmann@rub.de)

1. **Learning rate in DNN.** Build a deep convolutional neural network similar to the one in problem set 10, problem 3, but insert a new dense layer with ReLU activation function and 64 units between the last max pooling layer and the output layer. As optimizer use `keras.optimizers.SGD`.

   (a) Train models with varying learning rates $\eta \in \{1, 0.1, 0.01, 0.001, 0.0001\}$ on the first $20,000$ samples of the fashion MNIST training data set. Train each model for a total of 10 epochs. After each epoch evaluate the accuracy and loss on the training set. **Note:** Make sure to rebuild your model and compile it with the new learning rate. Just calling *model.compile* is not enough, since it does not reinitialize the weights of your model.

   (b) Plot accuracy and loss for each model as a function of trained epochs. How do the different learning rates affect the speed of learning? Can the choice of learning rate prevent the model from learning?

2. **Overfitting in DNN.** Reuse the model architecture of the previous problem and increase the number of units of the last hidden layer to 128. You may use *keras.optimizers.Adam* as optimizer.

   (a) Train models on varying training set sizes $N \in \{30000, 15000, 2500, 500\}$ of the fashion MNIST data set. Train each model for 10 epochs. Evaluate accuracy and loss on both the training set and the test set.

   (b) Plot accuracy and loss for each model as a function of training set size. How does the size of the training set affect accuracy and loss on the test set?

3. **Data Augmentation.** The images in the fashion MNIST dataset are standardized (crop, scale and orientation) to make it easier to train neural networks. Of course, real images are not standardized. To simulate a more realistic dataset, we use the *ImageDataGenerator* (from *keras.preprocessing.image*), a preprocessing tool that generates augmented data by applying various transformations (rotation, zoom, deformations, etc.) to your original data set. Normally, it is used for data augmentation to increase the size of and variability in your training data set.

   (a) Initialize an *ImageDataGenerator* object and use its *flow* function to create an iterator from the first $10,000$ samples of the fashion MNIST training set. During initialization of the *ImageDataGenerator* object use the following parameters:

      i. rotation_range: 20
      ii. width_shift_range: 0.1
      iii. height_shift_range: 0.1
      iv. shear_range: 0.5
      v. zoom_range: (0.9, 1.1)
      vi. fill_mode: 'constant'

   Iterate over the iterator object to acquire pretend data until you have generated roughly $20,000$ samples. Split the data $50/50$ into training $Train_{pret}$ and test $Test_{pret}$ sets. Save the pretend data set for later use. **Note:** Iterator objects can be iterated over indefinitely and require manual stopping.

(b) Reuse the model architecture of problem 1 and use *keras.optimizers.Adam* as optimizer with a learning rate of $\eta = 0.01$. Train the network on the training data in the original fashion MNIST dataset $Train_{orig}$ for one epochs and save it. (**Note:** You can save your model by calling its *save* function.) By saving the model you save the architecture, the weights as well as the optimizer and its hyperparameters. Let's call this model M_{orig}.

(c) Evaluate M_{orig} on the original test set $Test_{orig}$ and the pretend test set $Test_{pret}$. We will refer to these accuracies as $M_{orig}(Test_{orig})$ and $M_{orig}(Test_{pret})$. Why does the accuracy differ between the two sets?

(d) Train the same model architecture as above on $Train_{pret}$ pretending that the pretend data was real data. Evaluate the accuracy of this model M_{pret} on $Test_{pret}$. Compare $M_{pret}(Test_{pret})$ to $M_{orig}(Test_{pret})$. What did you expect?

(e) Train the model on $Train_{pret}$ with data augmentation for one epoch. To do so, call *model.fit_generator* with the *ImageDataGenerator* object you created and *steps_per_epoch* set to 4000 (for more information check the Keras documentary). We'll call this model M_{aug}. Compare $M_{aug}(Test_{pret})$ to $M_{pret}(Test_{pret})$. What did you expect?

4. **Transfer learning.** To illustrate transfer learning, we reused the previously trained network M_{orig}.

(a) Load the trained network from problem 3. (**Note:** To load your previously saved model you have to import the *load_model* function from *keras.models*.) Also load the pretend data set you generated in problem 3.

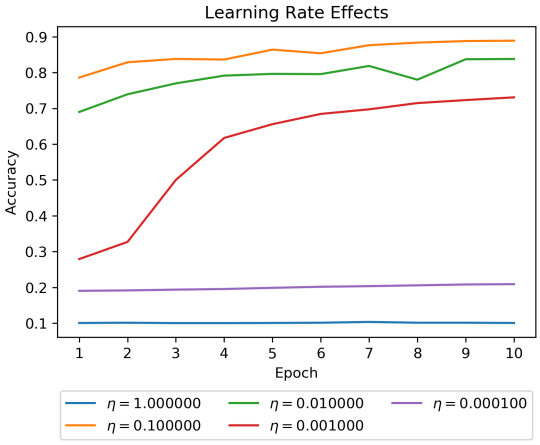
(b) Train the loaded model M_{orig} for one additional epoch on the first 500 samples of the pretend training set $Train_{pret}$. Also train a network of the same type *de novo* on the same 500 samples. Call this model M_{novo} Evaluate the test accuracy for both models. Repeat this for the full pretend training set $Train_{pret}$. How do accuracies differ in these two cases?

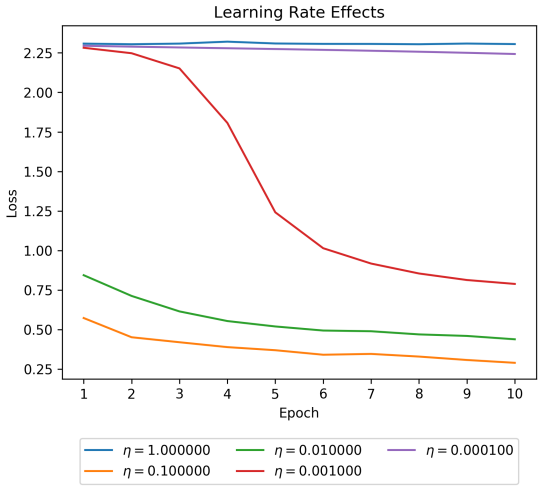
Solutions

1. **Learning rate in DNN.**

   (a) Please refer to *solution_learning_rate.py* for the code.

   (b) The learning rate mostly affects the speed of learning as should be expected. Smaller learning rates slow down learning and larger learning rates speed up learning. However, with too large a learning rate, such as $\eta = 1$, the model is not able to learn at all. Good performance is achieved with $\eta = 0.1$ and $\eta = 0.01$.

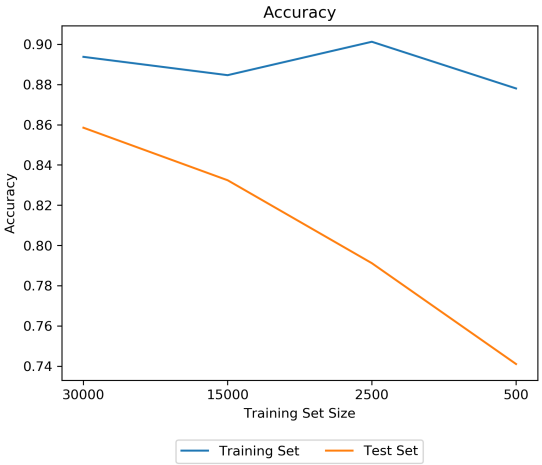


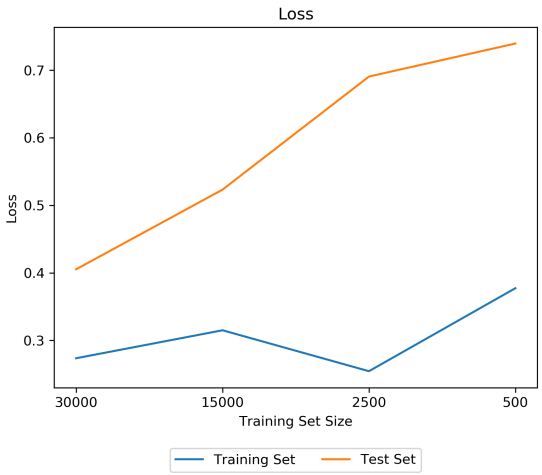


2. **Overfitting in DNN.**

   (a) Please refer to *solution_overfitting.py* for the code.

   (b) With decreasing training set size, the accuracy decreases and the loss increases on the test set. When decreasing the training set size from $N = 30000$ to $N = 500$, the test accuracy falls by almost 10% while the training accuracy stays roughly the same.





3. **Data Augmentation.**

   (a) Please refer to *solution_ data_augmentation.py* for the code.

   (b) Please refer to *solution_data_augmentation.py* for the code.

   (c) The trained network performs significantly worse on the augmented test set (*accuracy* ≈ 0.61) compared to the original test set (*accuracy* ≈ 0.85). This is because examples of the original data set are cropped, scaled and orientated the same way. The network overfits this specific data distribution and does not generalize well to different types of images.

   (d) A model trained on the pretend data set performs better on the pretend test set than the original network does (*accuracy* ≈ 0.72), since the network can learn to cope with the transformations introduced by the image generator.

(e) Training the network using the *fit_generator* method is functionally equivalent to training on the pretend data set obtained from the image generator. However, since now the network is trained using more samples ($steps\_per\_epoch \cdot batch\_size \approx 4000 \cdot 32 = 128000$), the performance is significantly better ($accuracy \approx 0.77$).

4. **Transfer learning.**

   (a) Please refer to *solution_transfer_learning.py* for the code.

   (b) After training on the first 500 samples, the pretrained network outperforms the *de novo* network ($accuracy \approx 0.67$ vs. 0.36). This is because the pretrained network can transfer what it learned to the new data set. The difference in test accuracy diminishes when training the networks on the complete augmented training set ($accuracy \approx 0.74$ and 0.7 for the pretrained and *de novo* networks respectively). **Note:** Due to the stochastic nature of the training process, it is possible that sometimes you get a different result.