Exercise 1
ConvNets
Advanced Practical Course In Machine Learning

Alon Netser
31160253-6

November 16, 2019

1 Theoretical Questions

1.1 Parameterized ReLU

TODO

1.2 Sigmoid Derivative

TODO

2 Practical Exercise

2.1 Linear Regression

Figure 1 shows the mean loss (over all mini-batches in this epoch) as a function of the epoch number. The learning-rate that was used in 0.0001 (bigger learning-rates results in less "smoothed" graph).

2.2 Multi-Layer-Perceptron

Figure 2 shows the accuracy and loss as a function of the iteration number. The graphs are smoothed (using the TesnorBoard smoothing mechanism), because the values themselves (of each iteration) are quite noisy. One can see that the training process over-fit, as the training loss decreases and the testing loss increases.

2.3 ConvNet

Figure 3 shows the accuracy and loss as a function of the iteration number. The graphs are smoothed (using the TesnorBoard smoothing mechanism), because the values themselves (of

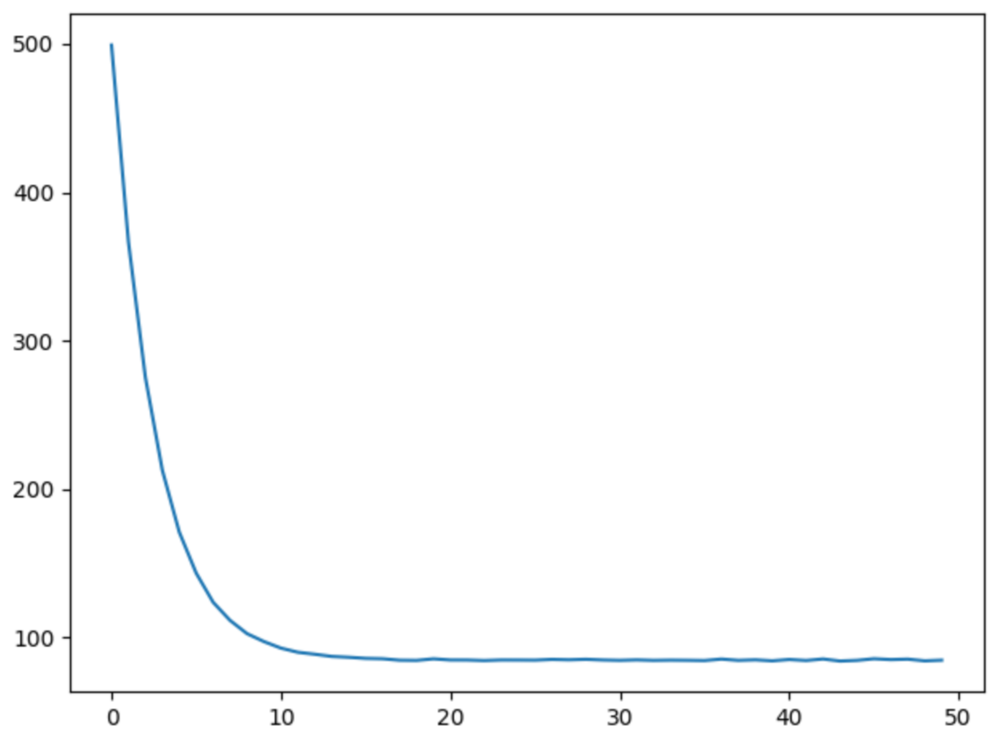


Figure 1: Linear regression loss

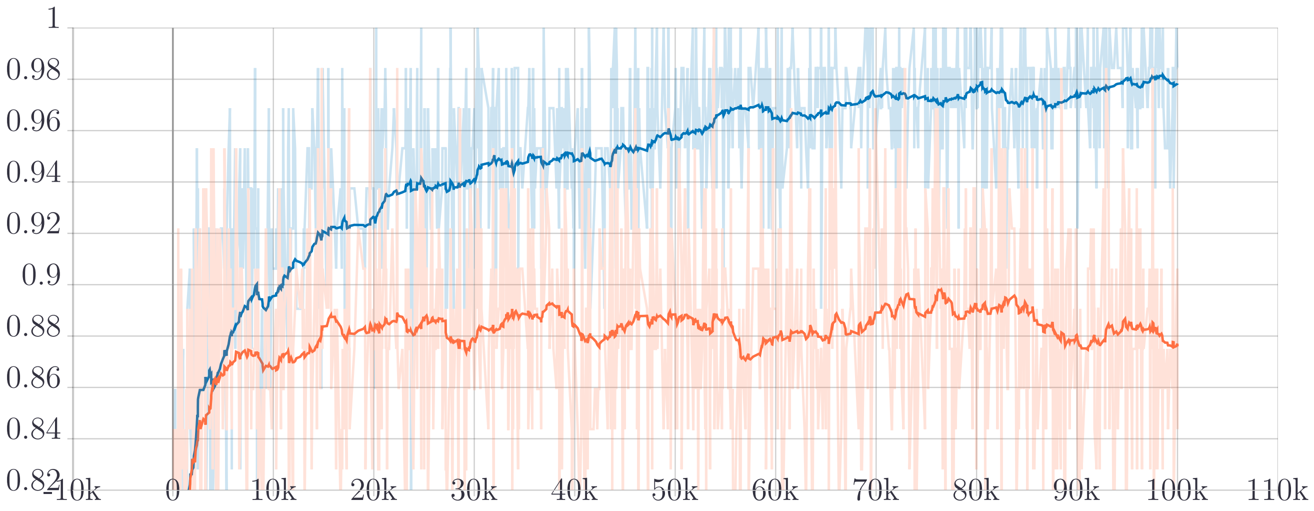
each iteration) are quite noisy. One can see that after about 10,000 iterations the training process begins to over-fit, as the training loss decreases and the testing loss increases.

2.4 Regularization - dropout

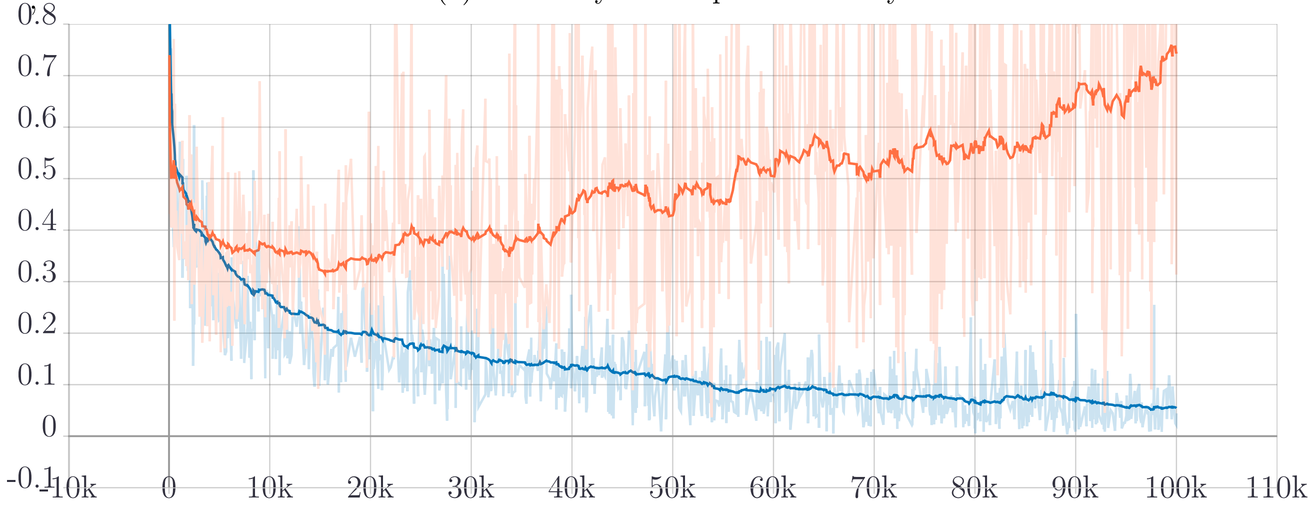
I tried to add regularization, in the form of dropout layers. In the Multi-Layer-Perceptron a dropout on the hidden layer was added, with a rate of 0.25. In the ConvNet two layers of dropout were added (after each conv-pool block), each with a rate of 0.25. Note that in both models different probabilities were taken, but 0.25 achieved the best performance (although the differences were quite small). The results show clearly that adding dropout helped avoid over-fitting the training-data, however the performance did not increases a lot.

2.4.1 Multi-Layer-Perceptron

Figure 4 shows the accuracy and loss as a function of the iteration number. The graphs are smoothed (using the TesnorBoard smoothing mechanism), because the values themselves (of each iteration) are quite noisy. Clearly, the version with the 0.25 dropout layers helped avoid over-fitting, despite the fact that the train/test performance are still not the same. Note that the dropout helped decreasing the loss (although the accuracy remains about the same), comparing to the regular version without dropout.

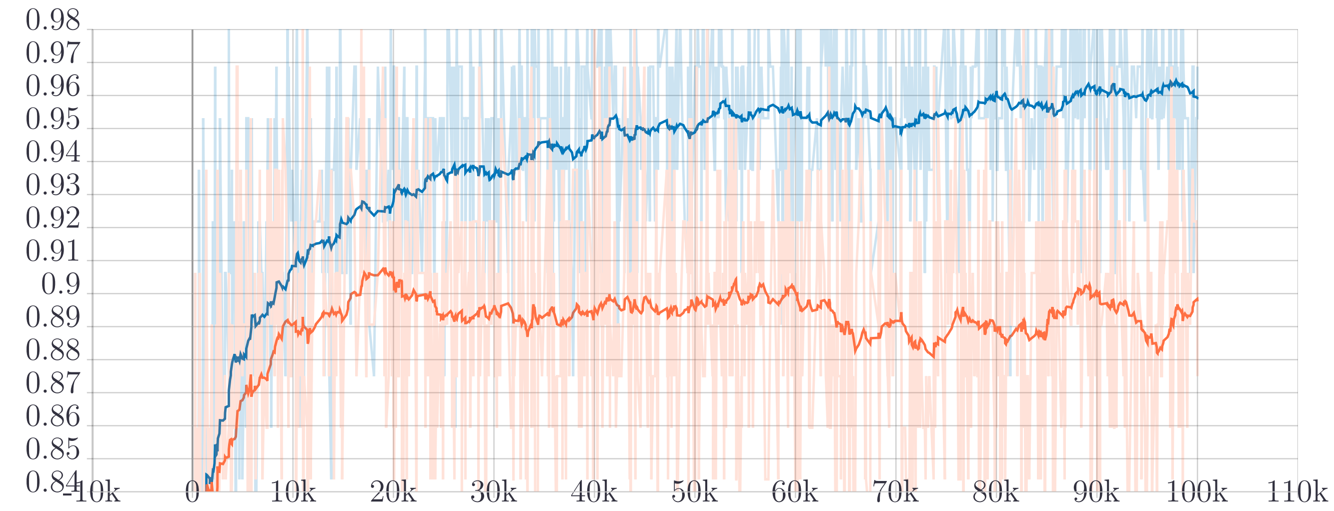


(a) Multi-Layer-Perceptron accuracy

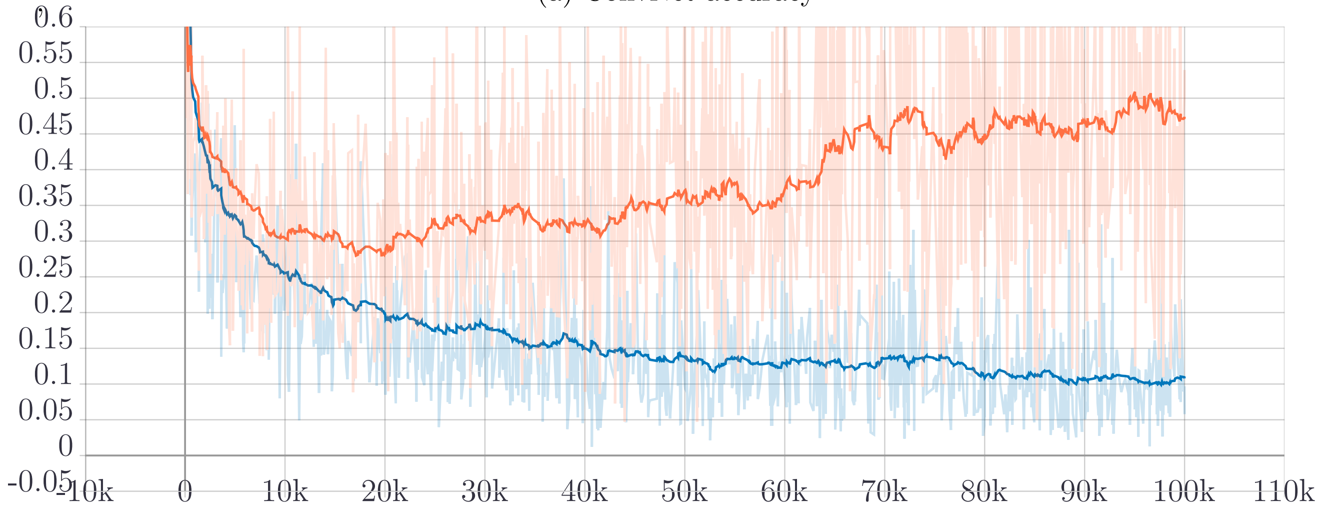


(b) Multi-Layer-Perceptron loss

Figure 2: The accuracy and loss of the Multi-Layer-Perceptron, as a function of the iteration number. The blue graph refers to the training performance, whereas the orange one refers to the testing performance.

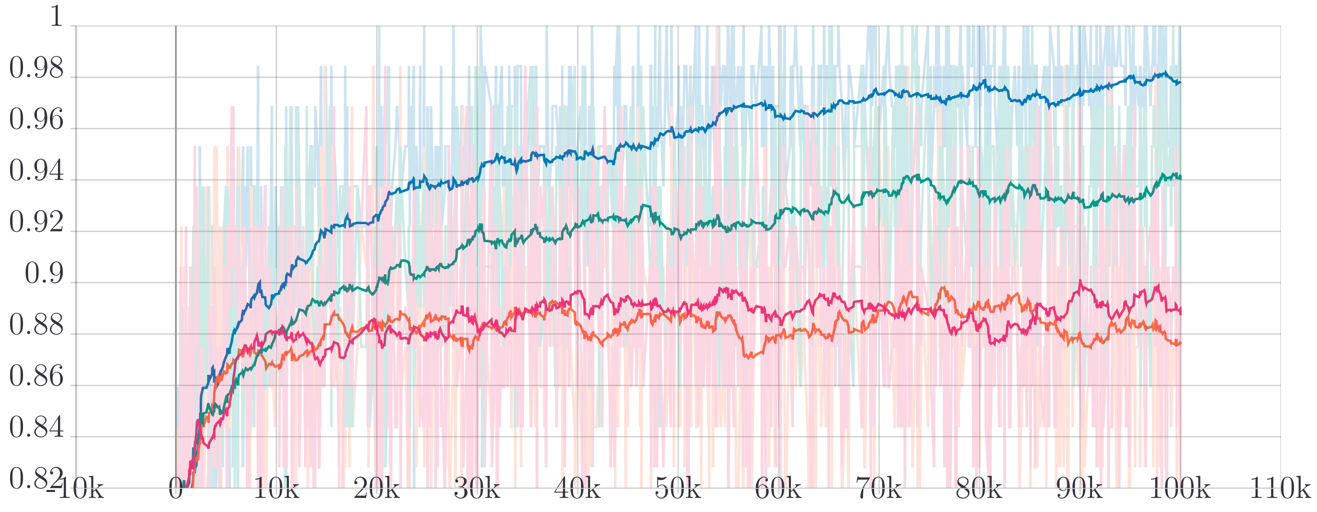


(a) ConvNet accuracy

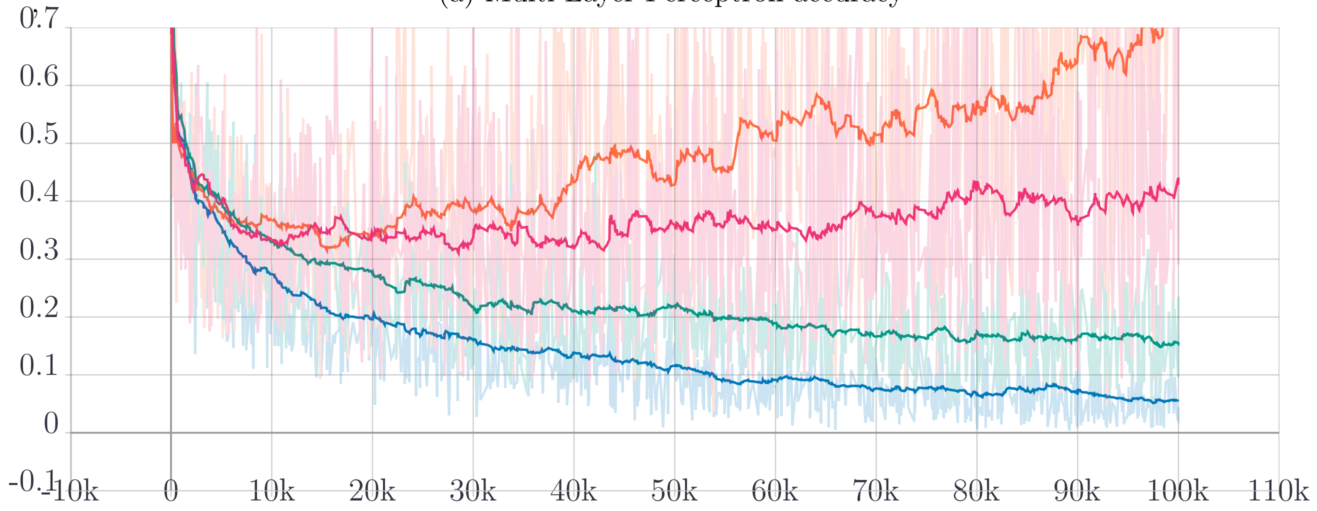


(b) ConvNet loss

Figure 3: The accuracy and loss of the ConvNet, as a function of the iteration number. The blue graph refers to the training performance, whereas the orange one refers to the testing performance.

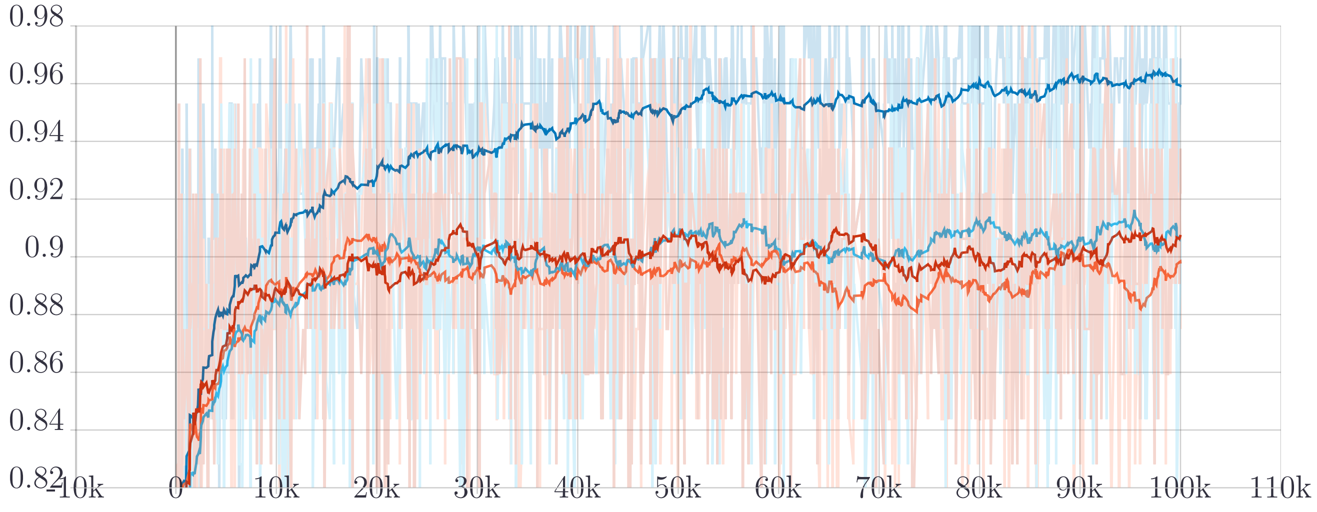


(a) Multi-Layer-Perceptron accuracy

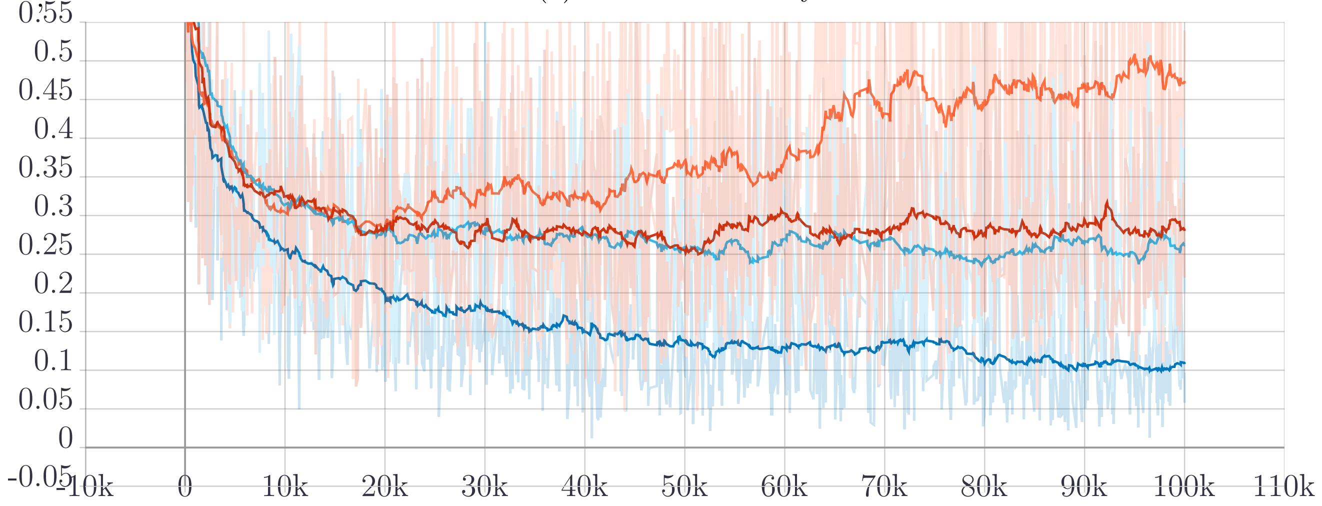


(b) Multi-Layer-Perceptron loss

Figure 4: The accuracy and loss of the Multi-Layer-Perceptron, as a function of the iteration number. Comparison between a version without dropout to a version with a dropout layer with rate 0.25, after each hidden layer. As before, the blue/orange graphs refer to the training/testing performance of the no-dropout version. The green/pink graphs refer to the training/testing performance of the dropout version.



(a) ConvNet accuracy



(b) ConvNet loss

Figure 5: The accuracy and loss of the ConvNet, as a function of the iteration number. Comparison between a version without dropout to a version with two layers of 0.25 dropout, after each pooling layer. As before, the blue/orange graphs refer to the training/testing performance of the no-dropout version. The light-blue/red graphs refer to the training/testing performance of the dropout version.

2.4.2 ConvNet

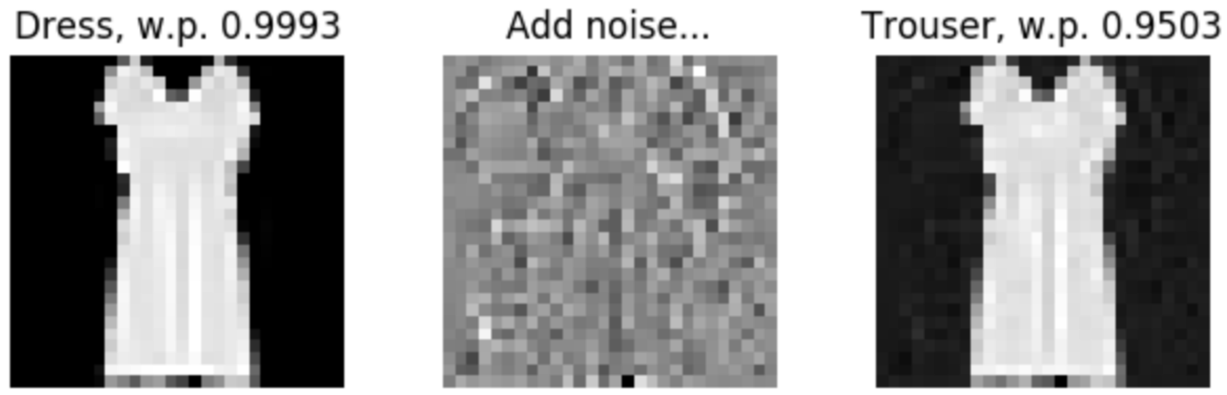
Figure 5 shows the accuracy and loss as a function of the iteration number. The graphs are smoothed (using the TesnorBoard smoothing mechanism), because the values themselves (of each iteration) are quite noisy. Clearly, the version with the 0.25 dropout layers avoids over-fitting, and the train/test performance are quite the same. Note that the dropout helped decreasing the loss (and also slightly increasing the accuracy), comparing to the regular version without dropout.

2.5 Adversarial sample

I found an adversarial using the following technique. I loaded a model (the trained ConvNet with dropout from the previous section). I took some random image from the training-set, and sampled a random label (i.e. target-label) that is different from its correct label. Then, I ran the network with that image as a input.

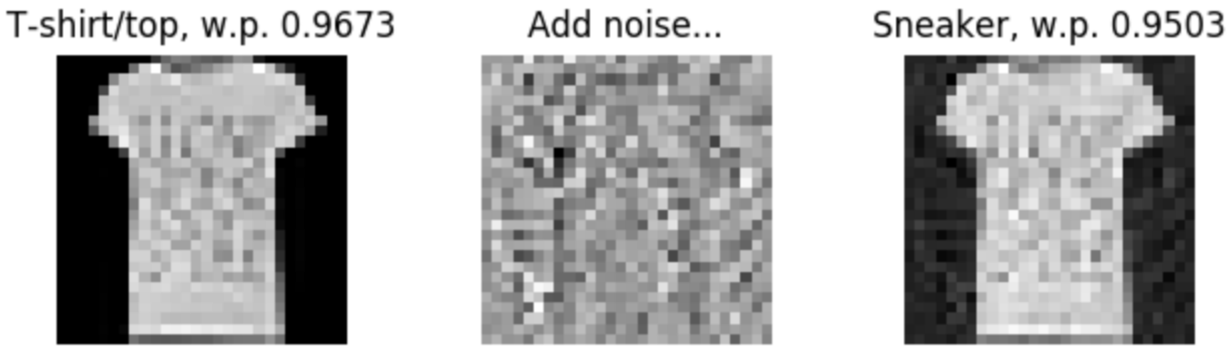
The new loss function was an addition of two terms - the first was that the prediction should be the same as the target-label, and the second is that the image should not be far from the original image (using ℓ_1-norm). Then, I calculated the gradient of this loss function, with respect to the input image, and updated the image according to this gradient (with learning-rate of 0.05).

In Figure 6 some nice adversarial images are shown, together with the original images and the added "noise" that resulted in the adversarial images.

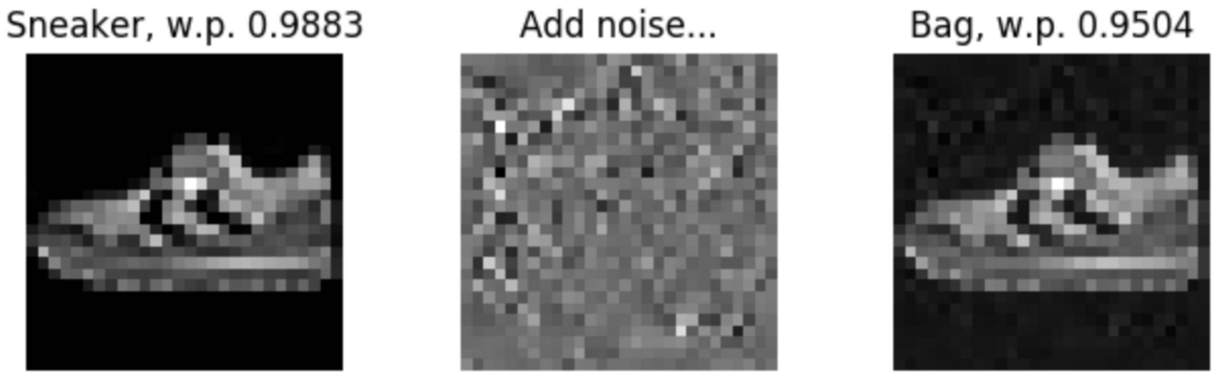


(a) A dress becomes a trouser

,



(b) A shirt becomes a sneaker



(c) A sneaker becomes a bag

Figure 6: These are all adversarial samples created using the method discussed.