Project Documentation for Mathematics of Neural Networks

Johann Kipping

January 21, 2022

Abstract

The aim of this project is to test the learned things in the first half of the lecture series on Neural Networks on the example of the Fashion-MNIST Dataset

Things that could be done on Fashion-MNIST

- Hyperparameter search
- Activation functions
- Initialization
- Optimizers
- Regularization (Overfitting?) + Generality
- FFT/im2col/Winograd
- Attribution
- Feature Visualization
- Adversarial Attacks
- Going deep?

0 Getting TF to run with a GPU with CUDA CC of 3.0

As this is a documentation of work it will be mentioned that a significant amount of time needed for the setup of computations was put into getting tensorflow to compute on the available GPU (Geforce 660 Ti). The major obstacle was the fact that this particular GPU has a *CUDA compute capability* of 3.0. The least tensorflow supports as of now is 3.5. This lead to the necessity of **downgrading to tensorflow version 1.12**. This might impede the ability for coverage of advanced topics.

1 Idea collection and planning

This chapter aims to collect ideas to most of the topics that were discussed in the lecture and continues to formulate a plan on how to tackle them in this project work.

Loss functions used is categorical cross entropy. Why not Mean squared or something else

Hyperparameters:
Idea: Try several choices of hyperparameters to get a feel for the effects. Do this in a grid-like fashion.

Activation functions / network structure
Idea: Test small network structures (fast trainable) for effect of structural changes and different choices of activation functions using knowledge from the lecture and Exercise 4 from sheet 6 as starting point for network evaluation.

Initialization
Idea: Verify the validity of the three approaches discussed in the lecture. (Zero/Random/Known data)

Optimizers
Idea: Use Adam for all computations to reduce complexity of project as it incorporates other approaches.

Regularization (Overfitting?) + Generality
Idea: Test for overfitting and generalization with own pictures (for fun). Maybe try regularization if poor performance is observed.

FFT/im2col/Winograd
Idea: Let tensorflow decide on the usage of different approaches as insight was achieved in the exercises and modifications are not cheap.

Attribution/Feature Visualization
Idea: Visualize a net (with more layers) feature maps by using Saliency maps and grad-Cam. Try out guided backpropagation if possible! Relu has to be used! Gradient ascent (google feature visualization)!
Neuron/Channel/Layer(DeepDream)
Adversarial Attack
Idea: Use an adversarial attack to trick a net.

Everything plot not shown in this documentation will be available in the github project.

2 Getting to know Neural Networks

This section will describe the first set of investigative experiments. Mainly regarding stuff of the first couple of lectures its main aim is to allow to get an understanding and feeling for the behavior of Neural Networks.

2.1 Starting out

As a starting point the network structure from Exercise 4 of sheet 6 was chosen to allow further investigation. As described in section 1 Adam was be used as the optimizer throughout this section in order to reduce complexity.

2.2 Initialization experiments

The methods for initializations are to be tested in this section:

- zero
- known data
- random

As mentioned before the net structure used for this will be structure from Exercise 4 of sheet 6.
The random initialization will be employed before the known data approach as the results from the random initialization will be used as the known data.

2.2.1 Zero initialization

As expected little to no improvement can be seen from training.

2.2.2 Random initialization

The baseline performance seen in the exercise. Uniform he initializer was used.

2.2.3 Known data initialization

This method allow the last two dense layers for classification to be further trained. This improved the networks performance for the training data, not for the test data

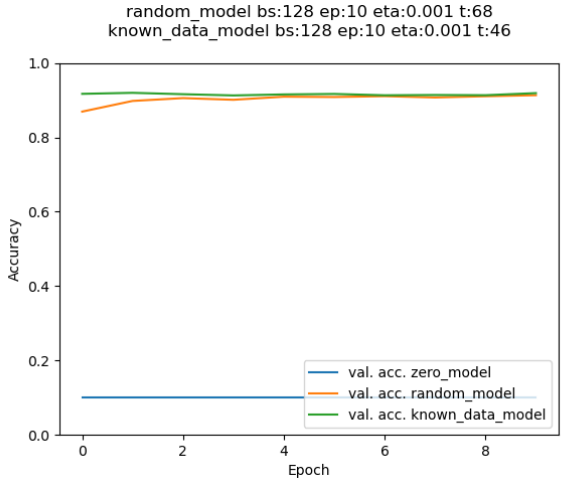

Figure 1

The resulting argument to be made is that form now on random initialization will be used due to lacking of known data. Zero initialization is not to be used.

2.3 Hyperparameters for baseline network

The hyperparameters for the fixed network structure of the baseline network with Adam as a fixed optimizer are batch size, epochs, learning rate, activation functions and initializers. The latter two have been or will be examined at another point in this documentation. In the following part a small grid-like search is employed.

	64	128	256
.01	0.91	0.87	0.1
.001	0.91	0.91	0.9
.0001	0.89	0.89	0.88

Epochs: 5

	64	128	256
.01	0.1	0.87	0.1
.001	0.92	0.91	0.91
.0001	0.91	0.9	0.9

Epochs: 20

Different training times
Curve appearance/saturation

2.4 Network structure

The goal with changing network structure is to achieve better approximation of the wanted function. In this section three different approaches are chosen to try to increase performance.

1. Increasing the number of neurons/filters motivated by the universal approximation theorem.
2. Going deeper: allow for more steps of abstraction by increasing the numer of layers.
3. Combining previous approaches and testing performance for different activation functions.

The starting point for the hyperparameters will be those taken from section 2.3.

2.4.1 UAT: Going wider

In trying to reach a higher test accuracy the approach to use more neurons, motivated by the universal approximation theorem was taken.
The structure use in the previous sections was modified to use 4 times the amount of filters for the convolutional layers.

Results did not show a visible improvement in testing accuracy neither for increasing the numer of filter in the convolutional layers nor the dense classification layers. This could indicate overfitting which will be investigated in section 3. A clear bottleneck for computation here was the rather tiny 2 GiB of memory on the GPU.

To investigate the effects of increasing the number of neurons anyway a network structure with a flattening and one dense hidden layer is chosen and tested for the effect of having more neurons. The results show minimal improvement even for 32 times more neurons.

2.4.2 Going deeper

Another trick up the network designers sleeve is making the network deeper. As soon as another network structure was tried out and no improvement was observed a harder dataset had to be consulted as it seemed as though the baseline net was near optimal for Fashion-MNIST in the sense of efficient implementation with the tools at hand in the last couple of sections.

The next step was then to try the CIFAR-10 dataset with the baseline net and compare the performance to a deeper version of the net. This resulted in a better performance at last but the high difference in the training accuracy and validation accuracy is still observable. This will be tackled by the regularization attempts in the following section.

2.4.3 Activation functions

At this point it should be investigated why the baseline net contains the activations it does. To this end the following was modifications were made and analyzed for the effects in training and validation accuracy:

Several activation functions were tested for the baseline net to **replace ReLU**:

Elu: The exponential ReLU activation performed marginally worse than ReLU in all tests with the baseline net.

Sigmoid: The sigmoid function showed very low training and validation accuracy although with a monotone increase until the last epoch. The difference of training and validation accuracy was smaller than the other activation functions.

Tanh: The tangens hyperbolicus outperformed all other functions and will investigated further in the next section.

3 Regularization

3.1 Possible methods

- Depending on the Eigen/Singular Values vanishing/exploding gradients. As ReLU was used this is not regarded as possible.

- Bias very negative -¿ Dying neurons -¿ use leaky ReLU.

- Gradient clipping used for keeping out of saturation where gradient is almost zero -¿ usage of ReLU makes this probable to be unsuccessful

- Tikhonov (L^2) -¿ insert penalizing term into minimization to prevent weights getting to large. (Frobenius norm has to be used)

- Batch normalization -¿ normalize input to layers

- Dropout (Dropconnect) -¿ drop output of neurons with probability for each minibatch

As the previous chapter revealed we.. (Use ReLU and cost to focus on dropout)

3.2 Dying neurons

Using leaky ReLU

3.3 Batch normalization

To make the input more readable for the net we normalize each element of the input to every layer to have mean $\mu = 0$ and variance $\sigma = 1$. To restrict less we learn parameters (linear transformation) $\beta and \gamma$ during backpropagation. Usage of ReLU

3.4 Dropout

Localized features, more specialized filters

4 Attribution

Check for generality and attribute in the same step by using an image from the test set and an image that was taken by me

Using gradient tape -¿ saliency map
Global average pooling -¿ class activation map
Grad-CAM! am ehesten