

Assignment 2

Markus Gall

Carmina Coronel

How does SGD work? What are your findings on the example data (part1)?

Stochastic Gradient Descent (SGD) is used to minimize loss on training data by processing the training data in mini batches of size $S=1$. In practice, using $S>1$ is common and still referred to as SGD. In deep learning, common S values are 64, 128, 256. The algorithm for SGD is the Minibatch Gradient Descent, where a subset of the training data, sampled randomly, is used to calculate the model error and update the model coefficients.

What are your findings on the example data (part1)?

For beale and madsen, it was easy to find the global minima. Using nesterov momentum provided faster results than not using it. However, for Schaffern4 has many local minima, thus sometimes, the optimization fails.

Which network architecture did you choose for part 2 and why?

For part 2, we use the following model architecture:

CONV1(activation=relu) → POOL → CONV1(activation=relu) → POOL → FCL(activation=relu) → FCL(activation=relu) → SOFTMAX

*FCL - fully connected layer

Learning rate = 0.01

Momentum = 0.9

Results (epoch=100):

train loss: 0.000 +- 0.000

val acc: 0.726

Best Accuracy of 0.7300342969132778 at epoch 74

We used CNN because for image classifications this type of network works well. It works well because it extracts features from the images using spatial relationship of the pixels. We initially chose this architecture as we felt a 2 level convolutional layer would give good results.

We then added another convolutional layer in an attempt to improve the results.

CONV1(activation=relu) → POOL → CONV1(activation=relu) → POOL → CONV1(activation=relu) → POOL → FCL(activation=relu) → FCL(activation=relu) → SOFTMAX

Results (epoch=100):

train loss: 0.000 +- 0.000

val acc: 0.736

Best Accuracy of 0.7491425771680549 at epoch 39

However results did not substantially improve.

Did you have problems reaching a low training error?

Training loss is almost 0 but our validation accuracy is around 75% suggesting possible overfitting and model can still be improved.

What are the goals of data augmentation, regularization, and early stopping?

The goals of data augmentation, regularization, and early stopping is to avoid underfitting and overfitting.

How exactly did you use these techniques (hyperparameters, combinations) and what were your results (train and val performance)?

We performed data augmentation (flipping, cropping, and blurring), with or without dropout methods, different weight decays, different location of dropout layers, with and without batch normalization.

We improved our validation accuracy by using data augmentation (flipping, cropping of images) from 73% to 83%.

List all experiments and results, even if they did not work well, and discuss them.

(results shown are at epoch 100)

- Data Augmentation

train loss: 0.297 +- 0.051

val acc: 0.826

Best Accuracy of 0.8280254777070064 at epoch 92

Better validation accuracy by using flipping and cropping of images. If more data augmentation techniques is employed, results can still be improved.

We tried adding Gaussian filters but it did not improve the validation accuracy.

- Weight Decay to 0.0001

train loss: 0.309 +- 0.053

val acc: 0.825

Best Accuracy of 0.8358647721705047 at epoch 93

Minimal change to results when weight decay is lowered. We can get away with 0.001

- Dropout Layers

- Before the 2nd convolutional layer

train loss: 0.402 +- 0.054

val acc: 0.805

Best Accuracy of 0.8054875061244487 at epoch 100

- As first layer

Acc: 0.78

- After the last convolutional layer

Data missing

Dropout layers did not work for us. We tried putting it in different places in the model but no improvement, worst, it reduces our model results.

- Batchnormalization

Batch normalization did not improve the accuracy.

If you utilized transfer learning, explain what you did and your results.

We utilized transfer learning by using densenet121. We trained the whole training set with no weights freeze. This is different from what was usually done, which is to freeze weights of the convolutional parts of the pretrained model while the fully connected layer is trained anew.

Using the pretrained model, we achieved a 95% validation and test accuracy. This model is now the model saved in `best_model`.