4-class problem

Optimizer – Adam

* Learning rates – 0.1, 0.01, 0.001, 0.0001
* Loss – Categorical cross entropy

Batch size – 128, 256, 512, 1024

Epochs – 10, 100, 1000, 2000

Metric – Accuracy

* Early stopping
  + Monitor – val\_loss
  + Min\_delta=0
  + Patience=2
  + Mode=’auto’

Layers

* Regularization – Dropout and L1 or L2
  + Lambda = 0.1, 0.01, 0.001
* Hidden layers
  + 1 or 2
  + 2 to double number of input neurons
  + Softmax, sigmoid, tanh, ReLU
* Output layer – Softmax

Opimizing Single Hidden Layer

* Attempt 1
  + Using 1% of data (about 10000 samples)
  + 35 neurons, 768 samples per batch, 500 epochs, L2 reg, 0.001 for lambda: mean 0.297826, std 0.00347
    - Mean 0.320647, std 0.001050 for whole data set
  + Trying L1 and L2 – Experiment 1 and 2
    - Best acc is L2 where lambda is 0.05 and gives mean acc of 0.308112, std of 0.005134
    - Best acc for L1 was where lambda is 0.01 and gives mean acc of 0.300510, std of 0.001950
  + Trying neurons – Experiment 3
    - Best acc is 45 neurons where acc is 0.309901 and std is 0.006778
  + Got acc of 0.317090 with std of 0.00208, indicating overfitted sample, so sample is too small to generalize, so need to use bigger sample
* Attempt 2
  + Found sample with better parameters since start of last attempt, 32 neurons, 768 samples per batch, 400 epochs, L2 reg, 0.001 for lambda: mean of 0.320992 and std of 0.001550 over entire test set
  + Will attempt to zero in on best number of epochs, then increase the size of the sample to tune on
    - 200 epochs gives best accuracy of 0.321233 with std of 0.001491
  + Will try using 10% of training data for tuning
    - Gives acc of 0.319259 and std of 0.00372
  + Trying neurons
    - Best acc is 75 neurons, with acc of 0.32178 and std of 0.00167
  + Changing lambda
    - Tried standard range, found for both L1 and L2 regularization accuracy was increasing as lambda was getting smaller, suggesting lambda is too larger (experiment 6 and 7)
    - To start with I tried setting lambda to 0, in case dropout alone is enough to avoid overfitting, and found accuracy increased to 0.32211 with std of 0.00399
    - Seems that dropout alone is enough with the current probability set to 0.5, will try adjusting this to see the effect, will not use regularization to avoid underfitting
  + Changing rate of dropout
    - Found 0.3 to be best, with accuracy of 0.32297, and std of 0.00326
  + Trying L1 and L2 reg without dropout
    - They seemed to not work with dropout, but it seemed a sensible idea to see what accuracies they gave when there was no dropout
    - Found best L1 to give acc of 0.32232 with std of 0.00291, found best L2 to give acc of 0.32128 and std of 0.00271
    - Decided to go with dropout
  + Changing lambda for L1 with dropout (experiment 11)
    - Tried retuning lambda for L1 to see if we could achieve a greater accuracy with that and dropout too
    - Found a value of 10^-9 for lambda gave the highest accuracy, achieving acc of 0.32361 with std of 0.00229
  + Retuning neurons
    - Found 60 neurons gave best accuracy of 0.323265 with std of 0.00211
  + Returned to using 100% of training data to see if the tuned hyperparameters generalized well
    - Achieved an accuracy of 0.325936 and std of 0.00174, so tuning yielded a 0.5% increase in accuracy
  + Retuning batch size and epochs
    - Batch size of 756 (experiment 13)
      * Best acc at 200 epochs with acc of 0.327148 and std of 0.002250
    - Batch size of 512 (experiment 14)
      * Best acc at 200 epochs with acc 0.326904 and std of 0.002194
    - Batch size of 256 (experiment 15)
      * Best acc at 200 epochs with acc 0.326133 and std of 0.00192718
    - Found that batch size does not seem to have a large impact on accuracy, so will keep with 756
  + Tuning learning rate on sample – Experiment 16
    - Found default learning rate of 10^-3 is best
  + Tuning learning rate and epochs
    - Suspect the default learning rate performed best as we optimised the number of epochs for it
    - Experimented with number of epochs on 10% of training data, found that optimal number of epochs was reached quicker than for the whole training data, but behaved similarly to tuning on the whole training data (experiment 17)
    - Tried tuning epochs for learning rate of 10^-4 on sample (experiment 18)
      * Found no increase in accuracy
    - Tried tuning epochs for learning rate of 10^-5 on sample (experiment 19)
      * Found no increase in accuracy
    - Seems increasing the learning rate just leads to requiring more epochs for the same accuracy, but has the benefit of reducing standard deviation
    - Will keep with learning rate of 10^-3
  + Fine tuning number of epochs
    - Tried 150 to 300 in steps of 10 (experiment 20)
      * No significant increase in accuracy
    - Tried same range but with a learning rate of 10^-4, as previously I found a smaller learning rate gave more stable accuracy (experiment 21)
      * Significant increase in accuracy for each epoch, suggesting smaller learning rate is optimal when training over entire training set
    - Tried 10^-5 (experiment 21 and 22)
      * Used tensorboard, explored with large range of epochs alongside 10^-4, 10^-4 performed best
    - Tried 10^-4 and 2\*10^-4 with seed 1 (experiment 23, 10^-4 log moved to experiment 24)
      * + 10^-4 performed best over 3000 epochs
    - Tried 10^-4 and 9\*10^-5 with seed 1 over 6000 epochs (experiment 24)
      * 10^-4 plateued at 0.3285, 9\*10^-5 loss fell faster at first, but then got stuck heading towards a bad minimum?
    - Repeated 9\*10^-5 and tried 7\*10^-5 with seed 2 (experiment 25)
      * Plateued
    - Tried 1.1\*10^-4 and 1.2\*10^-4 with seed 1 (experiment 26)
      * Lower loss than with smaller learning rates

Optimizing Two Hidden Layers

* Reran training for best two layer neural network, found loss for validation set was all over the place, suggesting regularization and dropout was not well optimised (experiment 27)
* Decided to simplify the problem by applying the same regularization across both layers, and also to use a relu. Also as training deeper network decided to use leaky relu in both layers
* Investigating dropout without regularization (experiment 28)
  + Started with batch size of 756 for 200 epochs, as this will be fairly quick and we can optimise these values later
  + Decided to use sample of 50% of data, as this gave reasonably smooth loss and accuracy curves whilst halving training time
  + Experiment 28.1 – Found 0.1 best, so continued to next experiment to explore 0.02 to 0.18 in steps of 0.02
  + Experiment 28.2, found 0.08 to be best, mean acc of 33.108 std of 0.0265
* Investigating regularization without dropout
  + Again using 50% of data
  + L2 Regularization (experiment 29)
    - Experiment 29.1 – Found best val to be between 0.001 and 0.00001, with 0.0001 appearing optimal
    - Experiment 29.2 – Found best val to be between 0.0001 and 0.00001
    - Experiment 29.3 – Found best val to be around 0.0001
  + L1 Regularization (experiment 30)
    - Experiment 30.1 – Found best val between 0.0001 and 0.00001, with loss increasing for 0.00001 after a small number of epochs
    - Experiment 30.2 – Found best val to be 0.00003, with mean acc of 32.962 and std of 0.0959
  + L1 best of the two, with more stable loss
* Investigated dropout rate of 0.08 with L1 regularization, changing lambda (experiment 31)
  + Tried 10^-i for i 1 to 10, acc still rising beyond 10 (experiment 31.1)
    - Mean 33.121 std 0.0713
  + Tried 10^-i for i 11 to 13, found best acc between 10^-10 and 10^-11 (experiment 31.2)
  + 6\*10^(-11) gave best acc of 33.053 with std of 0.121, no better than with dropout alone, so will use dropout alone (experiment 31.3)
* Tune learning rate on 1000 epochs with training set and validate with validation set (experiment 32)
  + Found 1e-6 best compromise
* Tune number of neurons in each layer, 100 epochs, 4 fold cross validation, 50% of the data (experiment 33)