High Performance Computing Laboratory



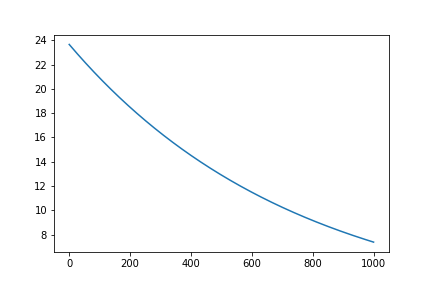
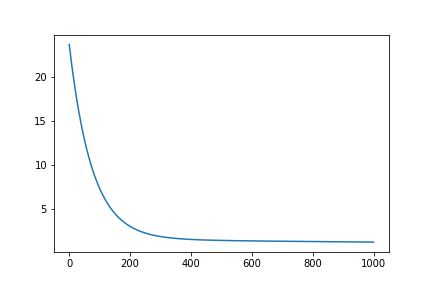
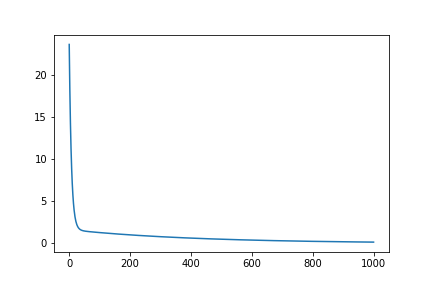
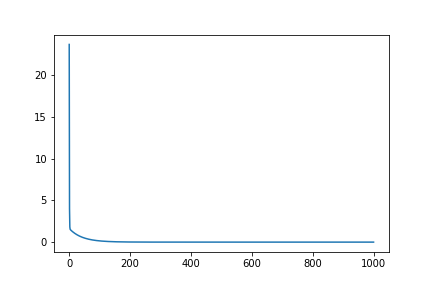
Master in Artificial Intelligence (MAI)

UPC Course 2021 - 2022

Ramon Mateo Navarro

## Exercise 1

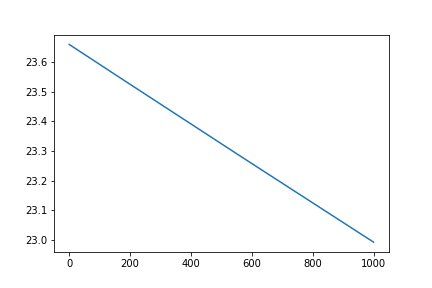
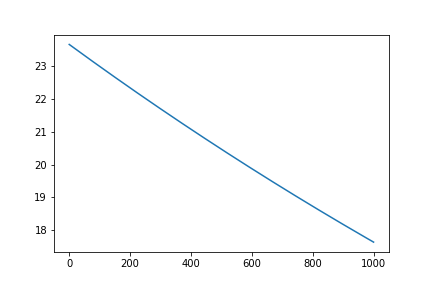
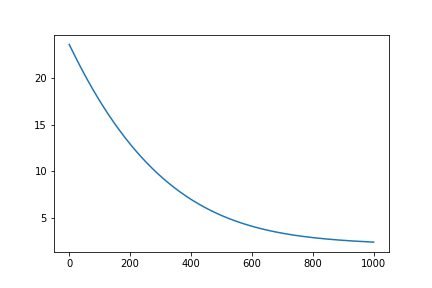
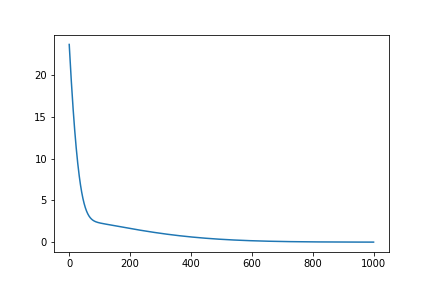
**Gradient descent optimizer**



Loss with Lr: 0.01, 0.001, 0.0001, 1e-5

It’s possible to see a different curve in how loss is reduced with different learning rates. The first difference is that 0.01 and 0.001 (top right and top left) have a better improvement with fewer steps, while 0.0001 requires more steps and 1e-5 doesn't converge.

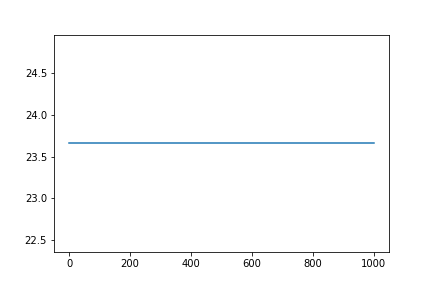
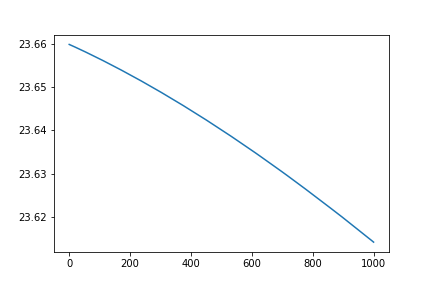
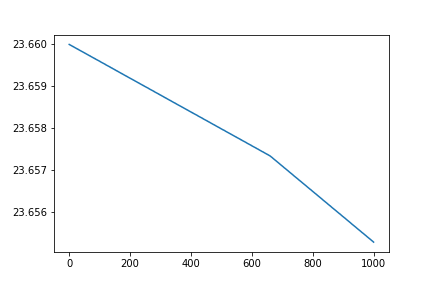
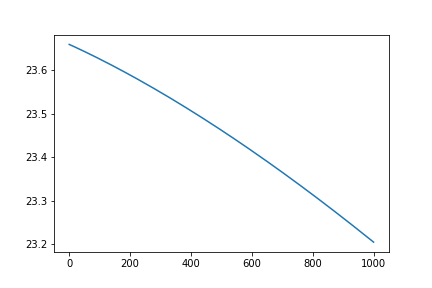
**Adam descent optimizer**



Loss with Lr: 0.01, 0.001, 0.0001, 1e-5

Using Adam Optimizer with Lr 0.0001 and 1e-5 does not converge meanwhile with Lr 0.01 yes but with 0.001 converges at final steps.

**Adadelta Optimizer**



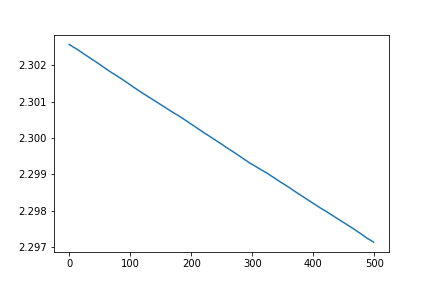
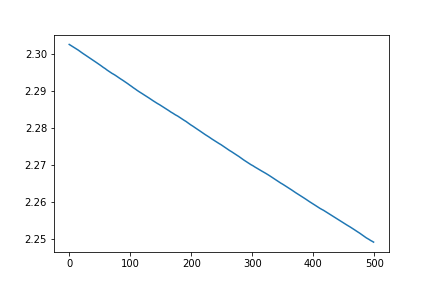
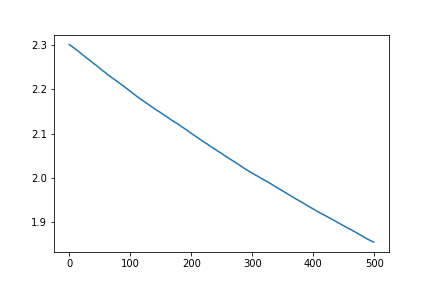
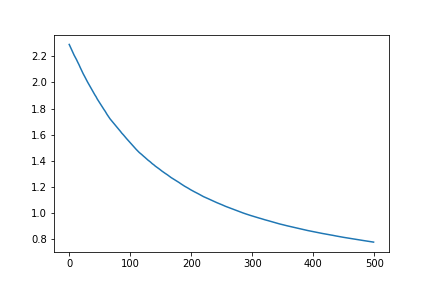
Loss with Lr: 0.01, 0.001, 0.0001, 1e-5

We can observe how never converge with different learning rates using Adadelta.

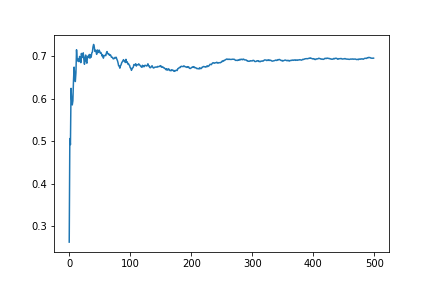
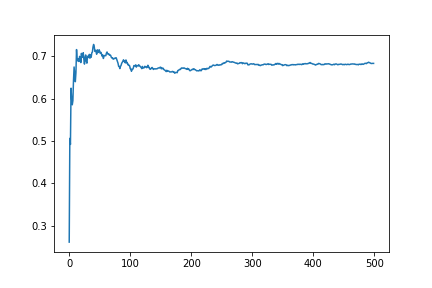
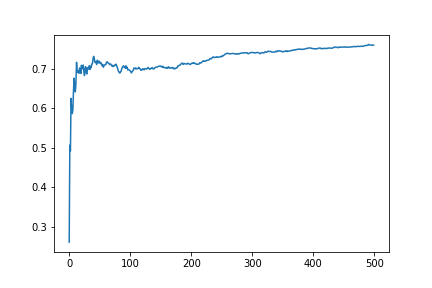
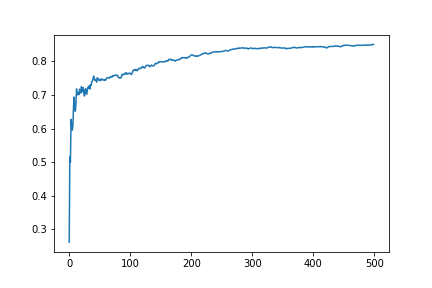
So in this exercise after testing different optimizers we can conclude that in this case Gradient descent converges better.

## Exercise 2

**Gradient Descent**

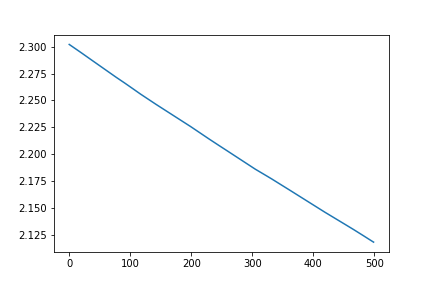
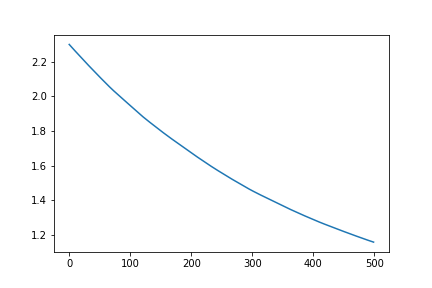
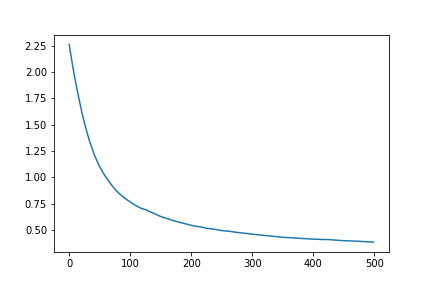
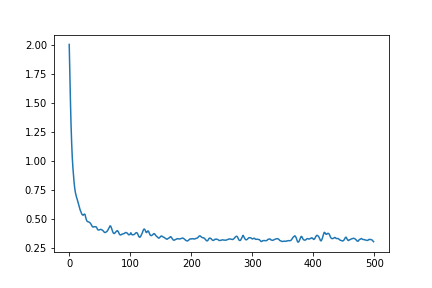


Cross Entropy with Lr: 0.01, 0.001, 0.0001, 1-e5

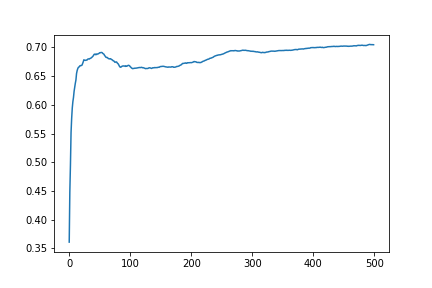
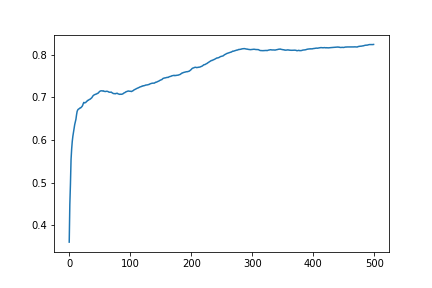
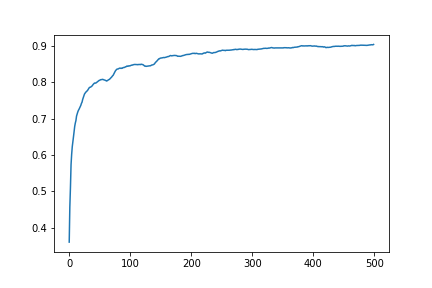
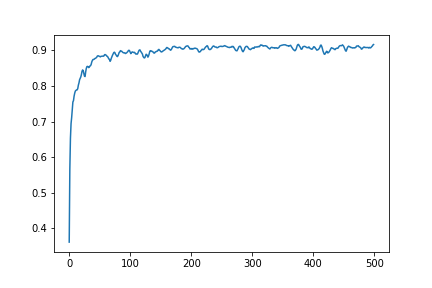


Accuracy with Lr: 0.01, 0.001, 0.0001, 1-e5

**Adam**



Cross Entropy with Lr: 0.01, 0.001, 0.0001, 1-e5

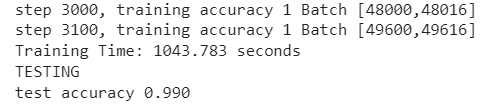


Accuracy with Lr: 0.01, 0.001, 0.0001, 1-e5

Comparing both optimizers can observe how using Adam it’s possible to achieve more accuracy. If we observe the Cross entropy loss it’s possible to see the reason. In Adam the loss converges with Lr 0.01 and Lr 0.001. Meanwhile in both cases that not converge the accuracy don’t achieve more than 80%

## Exercise 3

Accuracy achieved: **99%**



The first change was to add an epoch. The entire training is done 5 times and at the end of each epoch a shuffle of the entire dataset is done.

c = list(zip(data\_train, real\_output))

random.shuffle(c)

data\_train , real\_output = zip(\*c)

The second was to reduce the dropout.

* In the first version the keep\_prob was 0.5 so I increment this until 0.75

train\_step.run(feed\_dict={x: batch\_xs, y\_: batch\_ys, keep\_prob: 0.75})

The third was to reduce the batchsize. I reduced the batch size from 50 to 16. So this change requires that change the number of iterations in the training changing 1000 to 3125

With this change I achieved 99% accuracy and the time was 1043 seconds.

## 

## 

## 

## 

## 

## Exercise 4:

**GPU:** 1

**Time**: 35.214

**Test Accuracy:** 0.991

**GPU**: 2:

**Time:** 19.000 Expected time

**Test Accuracy:** ??

**GPU:** 4

**Time:** 4.276

**Test Accuracy:** 0.965

It’s possible to see how when we increment the number of GPUs the time needed reduces significantly. Approximately in factor 1.6 . But if we compare when a model is trained only in one GPU vs when its trained in four GPUs it is noted that the accuracy is reduced.