CS 475 Machine Learning: Homework 3
Deep Learning

Eric Rothman

(erothma6)

Instructions

We have provided this LaTeX document for turning in homework 3. We give you one or more boxes to answer each question. The question to answer for each box will be noted in the title of the box.

Other than your name, do not type anything outside the boxes. Leave the rest of the document unchanged.

For written answers, replace the **\TextRequired** (**Place Answer Here**) command with your answer. For the following example *Question 0.1* , you would place your answer where **\TextRequired** (**Place Answer Here**) is located,

Question 0.1

Place Answer Here

Do not change the height or title of the box. If your text goes beyond the box boundary, it will be cut off. We have given sufficient space for each answer, so please condense your answer if it overflows. The height of the box is an upper bound on the amount of text required to answer the question - many answers can be answered in a fraction of the space. Do not add text outside of the boxes. We will not read it.

For True/False or Multiple Choice questions, place your answers within the defined table. To mark the box(es) corresponding to your answers, replace **\Unchecked** (☐) commands with the **\Checked** (☒) command. Do not make any other changes to the table. For example, in *Question 0.2* ,

Question 0.2

☒ Logistic Regression

☐ Perceptron

For answers that require a single equation, we will provide a specific type of box, such as in the following example *Question 0.3* . Please type the equation where **\EquationRequired** (**Type Equation Here**) without adding any $ signs or **\equation** commands. Do not put any additional text in this field.

Question 0.3

w = **Type Equation Here**

For answers that require multiple equations, such as a derivation, place all equations within the specified box. You may include text short explanations if you wish (as shown in *Question 0.4*). You can put the equations in any format you like (e.g. within $ or $$, the **\equation** environment, the **\align** environment) as long as they stay within the box.

Question 0.4

$x + 2$ x is a real number

the following equation uses the variable y

$y + 3$

Do not change any formatting in this document, or we may be unable to grade your work. This includes but is not limited to the height of textboxes, font sizes, and the spacing of text and tables. Additionally, do not add text outside of the answer boxes. Entering your answers are the only changes allowed.
We strongly recommend you review your answers in the generated PDF to ensure they appear correct. We will grade what appears in the answer boxes in the submitted PDF, NOT the original latex file.

1) Dropout

Question 1.1 Advantages

The advantages of this is, much like the neural network, it decreases the dependency of the answer on a certain feature.
So in general it should decrease variance and overfitting.
Also much like in the case of neural networks, it would act as a kind of average over all the different svm's, sort of like boosting would.
It would give the same result as adding more data or adding noise to the data, which should make the trained SVM more rebust in general.
This would occur because each time an element is dropped out it makes an entire new data vector with that element 0 but every other element still what they used to be.
So in conclusion it should decrease variance.

Question 1.2 Disadvantages

There are many problems with this though.
Unlike general neural networks, but still could apply to specific networks, the SVM relies on the data vector to act like a guassian to calculate max min margin.
When an elements guassian position is changed, it could drastically impact the correctness of the SVM.
So it would take longer to train to guarentee that the SVM found is the optimal one.
In addition when a feature is zeroed, it no longer becomes sure that the data the SVM is training on that round is linearly seperable.
This is especailly true in data sets with a lot of features. zeroing an element in one of those data sets will most likely affect the linearity of the set more than actually helping remove variance.

2) Neural Networks

Question 2.1

What might be happening is that there are hidden features and connections in the data that can't come out within one layer of linear functions through nonlinear function.
If the data was organized such that the best features to learn on are hierarchical in nature, like image processing or connections between age/physical body ratio and location for example.
That could not be captured in a single layer network.

Question 2.2

This likely occurs in the neural network because the training data was overfit on by network.
A deeper network has more parts that are fit to the data, so every parameter is being accounted for even more since it impacts several different weights.
So a deeper network is more likely to overfit without precautions being taken since every variable is more relied on, so there is more variance in the trained data when a change occurs in the test data.
Finally a deeper network will diminish the gradient of the bottom layers when using back preprogation if its too long, also causing overfitting.

Question 2.3

There are several different solutions that can be taken.
One of the most useful ones is dropout, where nodes are randomly dropped from the network during testing.
Other techniques would include adding more data, randomly augmenting data to create more data, and adding normalization to the training.

Question 2.4

Place Answer Here

3) Backpropagation

Question 3.1 a Yes/No

☒ Yes

☐ No

Question 3.1 b Yes/No

☒ Yes

☐ No

Question 3.1 c Loss (to 2 decimal places) or Justification

0.04

Question 3.2

$J(x_1, x_2) =$ $-log(\sigma(1.1 * \sigma(0.2x_1 + x_2) + 0.4 * \sigma(0.3x_1 + 0.1x_2)))$

Question 3.3 Backpropagation partial derivative updates

$\frac{dJ}{dy} = y * \frac{1}{y} + (1 - y*)\frac{1}{y-1}$
$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{e^b}{(e^b+1)^2}$
$\frac{dJ}{dz_1} = \frac{dJ}{db}\frac{db}{dz_1}, \frac{db}{dz_1} = \beta_1$
$\frac{dJ}{dz_2} = \frac{dJ}{db}\frac{db}{dz_2}, \frac{db}{dz_2} = \beta_2$
$\frac{dJ}{d\beta_1} = \frac{dJ}{db}\frac{db}{d\beta_1}, \frac{db}{d\beta_1} = z_1$
$\frac{dJ}{d\beta_2} = \frac{dJ}{db}\frac{db}{d\beta_2}, \frac{db}{d\beta_2} = z_2$
$\frac{dJ}{a_1} = \frac{dJ}{dz_1}\frac{dz_1}{a_1}, \frac{dz_1}{a_1} = \frac{e^{a_1}}{(e^{a_1}+1)^2}$
$\frac{dJ}{a_2} = \frac{dJ}{dz_2}\frac{dz_2}{a_2}, \frac{dz_2}{a_2} = \frac{e^{a_2}}{(e^{a_2}+1)^2}$
$\frac{dJ}{\alpha_{1,1}} = \frac{dJ}{a_1}\frac{a_1}{\alpha_{1,1}}, \frac{a_1}{\alpha_{1,1}} = x_1$
$\frac{dJ}{\alpha_{1,2}} = \frac{dJ}{a_1}\frac{a_1}{\alpha_{1,2}}, \frac{a_1}{\alpha_{1,2}} = x_2$
$\frac{dJ}{\alpha_{2,1}} = \frac{dJ}{a_2}\frac{a_2}{\alpha_{2,1}}, \frac{a_2}{\alpha_{2,1}} = x_1$
$\frac{dJ}{\alpha_{2,2}} = \frac{dJ}{a_2}\frac{a_2}{\alpha_{2,2}}, \frac{a_2}{\alpha_{2,2}} = x_2$
$\frac{dJ}{x_1} = \frac{dJ}{a_1}\frac{a_1}{x_1}, \frac{a_1}{x_1} = \alpha_{1,1} + \alpha_{2,1}$
$\frac{dJ}{x_2} = \frac{dJ}{a_2}\frac{a_2}{x_2}, \frac{a_2}{x_2} = \alpha_{1,2} + \alpha_{2,2}$
β is the weights on the linear part of the output layer and α are the weights on the linear parts of the hidden layer.

Notebook Questions

Question 1.5.2a

Approximate accuracy for random classifier, rounded to whole integer = 20

Question 1.5.2b

Approximate accuracy for majority-vote classifier, rounded to whole integer = 20

Question 1.5.7a

Number of Weights = 68100

Question 1.5.7b

Number of Biases = 105

Question 1.5.20a

Best Validation accuracy, rounded to whole integer = 76

Question 1.5.20b

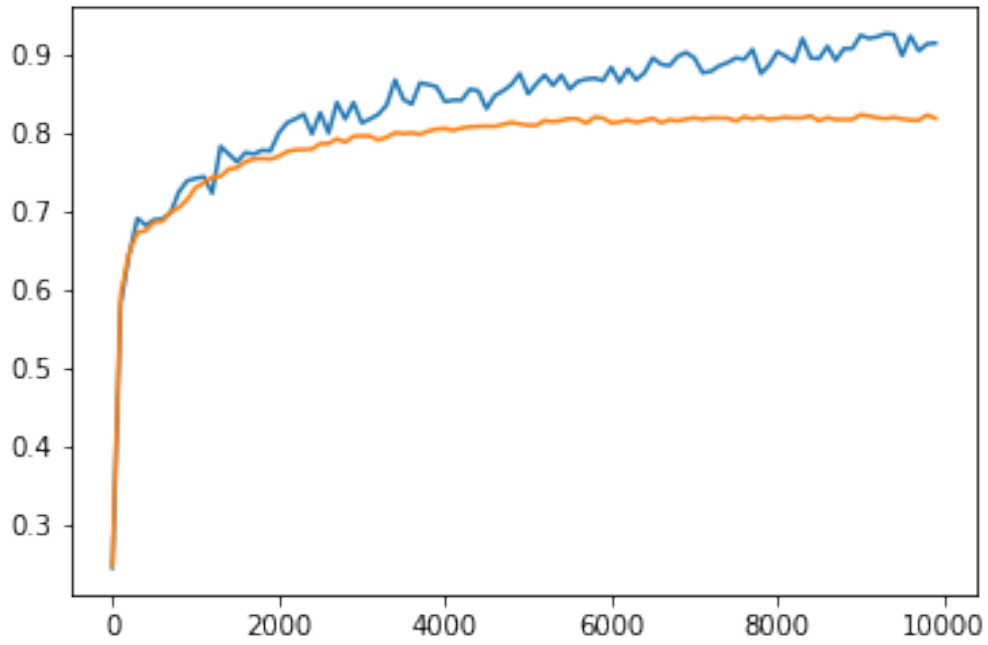
Time taken, in seconds, rounded to whole integer = 44

Question 1.5.20c: Did training take 10 times longer than in the case of pure SGD?

☐ Yes

☒ No

Question 1.5.22



Question 1.6.2. Compare comvolutional filters to fully connected NN

Its different. In full neural networks, each pixel/region would be an independent input and node in the network. In the covolution, the filters and the regions mapped through the filters are what is saved. So the filters are like the linear function for the fully connected NN.
Instead of having to calculate the weight of each function, the convolution network can calculate the weight for each part of the a filter. This allows for more possiblities to be stored in an easier way. It also means that the fact that parts of images give information and group up with other parts can be preserved rather than in straing linear where they get cut off depending on the number of nodes.

Question 1.6.6a

Best validation accuracy, rounded to whole integer = 76

Question 1.6.6b

Time taken, in seconds, rounded to whole integer = 25

Question 1.6.9a

Best validation accuracy, rounded to whole integer = 88

Question 1.6.9b

Batch size = 800

Question 1.6.9c

Learning rate = 0.01

Question 1.6.9d

Number of optimization steps taken = 3500

Question 1.6.9e

Time taken, in seconds, rounded to whole integer = 630

Question 1.7.2.8

The starting place was my answer from 1.6.9 which I changed to have a larger validation accuracy.
I eventually settled on Adam with a learning rate of 1e-3 at the start and an SGD of a learning rate of 1e-4 for when the network's val accuracy converges past a certain point. This is to ensure that the training settles on the local minimum. I tried every adjustment I could, including everything from adding dropout, batch normalization, and both. I increased the number of convolution layers and linear layers while also increasing the number of sieves/hidden nodes to be able to capture more data. I also increased and decreased the kernel size, how many layers and which layers were pooled, whether avg or max pool should be used, the weights for the adam optimization, and I preprocessed the data by normalizing it and adding some images to the training set by vertically flipping random images in the original set.
Training was sometimes too slow when the bin size was large, like 1000. To remedy this I decreased the bin size so that less examples were trained on at once. I also tried to pool as soon as possible so that the training would run faster. The most important change I made was to either add dropout to the model or to increase the initial kernel size to 5 or to add pooling earlier on in the training with a kernel of size 3.
I have 2 different models. I will talk about the one that did the best which is the 2nd one, and the one my optimal answer should have come from but for some reason I can't replicate. This model receives an input of a 26x26 pixel image. It then runs it through a convolution layer of kernel size 5, decreasing the new collection of features to be 22x22 with 8 sieves. It then runs that through another convolution layer and avg pools the outcome to be an image of 6x6 and 64 sieves. Then it runs through another layer creating the pool to be 4x4 and a final pool making it 3x3 with the first layer making the total number of sieves to be 128 and the second making it 512. Then it pools the remaining pixels together using a max pool so that the colection is a 1x1 group. Then it runs it through 3 linear functions with the first layer making it go from 1024 nodes to 400 and the second going from 400 hidden nodes to 200. Finally the last layer converts the 200 hidden nodes to 5 which predict the class the image is of.

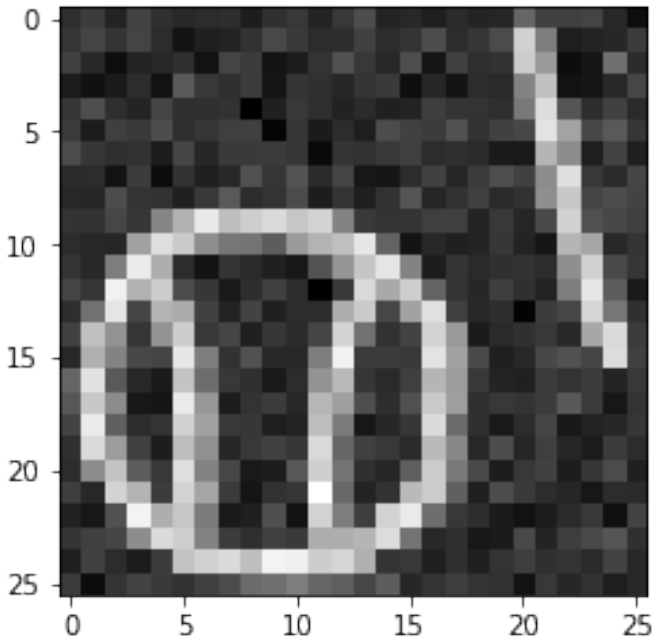
Question 1.8.4a: Are there any qualitative differences between these sets of images

- ☒ Yes
- ☐ No

Question 1.8.4b: Are the misclassified examples more difficult for you to classify

- ☒ Yes
- ☐ No

Question 1.8.6



Question 1.8.7: Does the classifier still classify all 10 images correctly

- ☒ Yes
- ☐ No

Question 1.8.10: Does the classifier still classify all 10 images correctly

- ☒ Yes
- ☐ No

Question 1.8.11

It is not necessarily a failure, for example if the image trying to be processed is text. Then when flipped horizontally, the outcome should be indecipherable gibberish or even unreadable.
There are things, like letters and numbers, which become meaning less when flipped horizontally.
For these classes though, each one has meaning when flipped horizontally, so it would be considered a failure.
To train to accomidate for the flipped images we could randomly flip images in the training set or during training time have a drop out effect sort of where instead of nodes disappearing, there is a random chance for each image in the minibatch that it will be flipped.
So during training have a line like: with p=0.1 an image will be flipped horizontally.
Exactly how often you should flip will change and is something the user should decide.
Or you could just add the flipped images to the data set as a whole at the beginning before devying up training and val data.
You could also add more nodes or use convolution layers to try to catch general patterns, not necessarily strict pixel ordering, but that might not necessarily work since you can't exactly decide what features to train on.
Out of the two things I suggested, I think convolution layers are the better choice since they are build to reuse patterns efficiently.