Project 1

Palani Johnson

November 11, 2021

Preface

In this project I trained five neural nets to recognize bees in image and audio datasets. Here is how it went.

Finding an Activation Function

One of the things that has confused me the most this semester were the plethora of different activation functions. We talked much about the sigmoid and ReLU functions but their specific qualities seemed to elude me. Before I began down the forking path of attempting to find suitable architectures for my nets, I wanted to figure out how these activation functions fared under similar conditions.

I began by choosing five diffrent activation functions: tanh, ReLU, sigmoid, softmax, and linear. There are of course many more, TFLearn defines 13, but alas I had limited computation power (and time). I specifically choose these functions because I have seen them used in publications or on the internet before and figured that if I, a lowly student, had seen them they would probably fit my simple applications. For my test I created a simple network of shape 4000x32x3 for training on the audio datasets. I wanted to see which function worked for the inner layers and which worked on the output layer. This is 25 nets in total, so I couldn't train for long. Some of the network parameters included a batch size of 25, 10 epochs of training, and a learning rate of 0.01. The final metric that I measured was the average final accuracy of our three validation datasets: BUZZ1, BUZZ2, and BUZZ3.

hidden → output ↓	tanh	relu	sigmoid	softmax	linear
tanh	31.16%	30.92%	34.04%	35.66%	30.91%
relu	36.71%	30.92%	36.33%	37.27%	36.57%
sigmoid	37.82%	36.71%	29.68%	28.93%	36.71%
softmax	30.92%	30.92%	39.01%	35.02%	36.71%
linear	30.84%	30.92%	31.61%	29.48%	35.76%

Table 1: Comparison of activation functions

Of course, this test is far from comprehensive and probably fails to generalize. From my limited anecdotal evidence I can say that much of the data displayed in this table holds true. Easily the worst performing pair of activations was a softmax hidden layer and a sigmoid output. I also found that ReLU and softmax was often a safer choice for the more complicated convolutions neural nets than some of the other combinations.

TFLearn Image ANN

On to actually finding an architecture. I started testing on various architectures. At the beginning this was mostly random.To be honest I didn't have any idea what would and wouldn't work so testing the application space was valuable for me to begin to figure out what to look out for. In the following table I show some of my initial tests. As a note on the architecture column the layers are separated by a "|" character, like the UNIX pipe, and a dropout is denoted as "da" where a is the dropout keep probability supplied to the function. Input and output is implied. All the hidden layers use ReLU and the output uses softmax activation functions. Like I said before, while my tests might have shown that some functions performed better than this combination, I found in practice that some of the other "good performing"

combinations were less stable on different architectures. ReLU and softmax was often the safest bet to use. All tests were done with batch size 20 training on all the data combined and shuffled.

architecture	learning rate	epochs	epoch 1 acc.	BEE1/2_1S/4 grey validation acc.
256|64|d0.9	0.05	20	0.82	0.90/0.84/0.69
10|10|10|10	0.05	5	0.65	0.48/0.24/0.55
10|10|10|10	0.1	5	0.62	0.50/0.31/0.58
1024|256	0.1	5	0.50	0.82/0.72/0.66
256|64|64|d0.9	0.05	20	0.79	0.91/0.85/0.70

Table 2: Comparison of ann architectures for BEE1, BEE2 1S, and BEE4 grey datasets

In the end I ended up sticking with the last architecture. I noticed that the accuracy was oscillating slightly during the final five epochs. To prevent this i tried lowering the learning rate to 0.01. Additionally, to prevent overfitting, I changed the dropout from 0.9 to 0.7. I also increased the batch size from 20 to 25 to speed up the training a little as I planned to push it out to 50 epochs. In the end my final accuracies for BEE1 grey, BEE2 1S grey, and BEE4 grey with this architecture was 93.03%, 79.04%, and 70.70% respectively. I persisted this model to `image_ann.tfl`.

TFLearn Image Convolutional Net

As before I began by testing various, random architectures. Besides bumping the batch size to 50, all of the parameters were the same as before with the addition of two new layers:

- "*a*x*b*" represents a convolutional layer with a filters of size b
- "p*a*" represents a max pool layer with size a

architecture	learning rate	epochs	epoch 1 acc.	BEE1/2_1S/4 validation acc.
64x4|p2|64|d0.9	0.05	5	0.89	0.93/0.86/0.66
16x16|p2|4x2|p2|64|d0.9	0.05	5	0.86	0.96/0.91/0.73
16x16|p2|8x4|p2|64|d0.9	0.05	10	0.91	0.98/0.90/0.69
continued from last*	0.01	10	0.96	0.98/0.92/0.72
16x16|p4|4x4|p4|64|d0.7	0.05	10	0.84	0.97/0.90/0.73

Table 3: Comparison of cn architectures for BEE1, BEE2 1S, and BEE4 datasets
*Continued training from previous architecture, changed dropout to 0.7

Like last time, the last network in this table is the one I stuck with. I wanted to try out the third network more, but It looked like it wasn't doing well with BEE4 and I wanted a better *overall* network than one that worked well on only one dataset. I decided to continue to train this final network in batches of 10 epochs because I was worried about over fitting. After a total of 40 epochs, and lowering the learning rate to 0.01 for the final 20, my final validation accuracy BEE1, BEE2 1S, and BEE4 with this architecture was 97.28%, 92.17%, and 75.00% respectively. I persisted this model to `image_cn.tfl`.

TFLearn Audio ANN

Jumping into the audio ANNs I knew from friends that these networks trained much faster but were also much more difficult to train. Because of this my plan was to lower the batch size to 20 and I wanted to train for 100 epochs for my final net. Before I found this net, however, I wanted to do some more tests:

architecture	learning rate	epochs	epoch 1 acc.	BUZZ1/2/3 validation acc.
32	0.01	10	0.38	0.53/0.50/0.52
32|32	0.01	10	0.37	0.27/0.22/0.68
32|32	0.10	10	0.40	0.45/0.53/0.48
64	0.10	10	0.54	0.53/0.47/0.52
64|d0.7	0.10	10	0.36	0.42/0.39/0.53
64|64|d0.7	0.10	50	0.33	0.31/0.35/0.59
32|32|d0.7	0.025	50	0.41	0.51/0.52/0.70

Table 4: Comparison of ann architectures for BUZZ1, BUZZ2, and BUZZ3 datasets

After lowering the learning rate to 0.01, I tried training the final net for another batch of 50 epochs for a grand total of 100 epochs of training. Unfortunately this overfit the validation data and lost me about 20% accuracy on all datasets. I ended up not persisting this net and keeping with the original 50 epoch net with final accuracies of 51.48%, 52.33%, 70.95%. This net was saved as `aud_ann.tfl`.

TFLearn Audio Convolutional Net

Before we jump right into bulldozing an architecture I wanted to test out a thought I had. Until now most of convolutional layers have had relatively small filter sizes. This worked well for the images because these filters are square. This dataset, being audio, is one dimensional. I wanted to see if relatively large filter sizes could help here. Here is what I found

architecture	learning rate	epochs	epoch 1 acc.	BUZZ1/2/3 validation acc.
32x5|p2|32	0.1	10	0.53	0.53/0.46/0.50
32x15|p2|32	0.1	10	0.45	0.60/0.54/0.52
32x25|p2|32	0.1	10	0.30	0.67/0.67/0.74
32x35|p2|32	0.1	10	0.42	0.62/0.65/0.66

Table 5: Comparison of cn architectures for BUZZ1, BUZZ2, and BUZZ3 datasets

At this point it appeared that my assumption was more or less on track. The epochs were starting to take really long with these large filters so I jumped back to a 25 filter size, set the learning rate to 0.05, and trained for 20 more epochs. After this long my accuracy on the training data was approaching 99% so I knew I couldn't squeeze much more out of this network. My final accuracies were 70.52%, 69.03%, 78.05%. I saved the network to `aud_cn.tfl`.

Keras Image Convolutional Net

Since I had already bulldozed a convolutional net architecture with TFLearn, I figured I wouldn't have to do any more bulldozing so I loaded up a Keras version of that TFLearn network and began training for 20 epochs. On the topic of copying the TFLearn code over to Keras, I found Keras much easier to work with. In the future I think I will stick with Keras because they also seemed to have a larger verity of layers, activation functions, and data augmentation and reshaping capabilities. Back to the neural network at hand, this training session ended with a final validation accuracy of 81.98%. Considering this was a smaller dataset than before I was pretty happy with this. I ended up retraining the neural net out to 50 epochs. This time the accuracy rose to 83.39%.