Going Deep: Using Neural Nets to Make Fantasy Football Predictions

Rohin Bhargava, *University of Chicago*

June 11, 2017

Abstract

In this paper, I use commonly aggregated statistics with different Neural Net Architectures to figure out if there were consistencies between previous season performance and future performance in the game of Fantasy Football. What I found was that deeper Neural Nets performed better on the data and that recurrent and convolutional architectures made impacts, but a combined deep architecture made the most.

1 Introduction and Remarks

Fantasy Football is a popular game primarily played in the United States where players are awarded points for picking a roster every week consisting of a Quarterback, two or three Running Backs, two or Three Wide Receivers, a Tight End and a Defense and a Kicker. Based on the statistical performance of each individual Football player that week, Fantasy players are awarded points and these points are aggregated (Fantasy points) in pairwise competitions. Whoever has more points in each paired competition is declared that week's winner. While each Football pick is important, Running Backs, Wide Recievers, Quarterbacks and Tight Ends generally make the difference between a "good" team and a "bad" team. For this reason, I chose to analyze data for each of these positions.

In figuring out how to structure this problem, I initially thought to track progress of individual players. When I learned that the average player stays within the NFL for 3.3 years, I determined that assessing individual performance would not be a good way to make these predictions. Instead, I resolved to make a regression model which would predict future Fantasy point totals, based on statistics in the prior year and the label of next years point total. For the input vector for this training, there were 23 statistics including Position, Team, Games Appeared in, Games Started, 5 passing statistics, 4 rushing statistics, 5 receiving statistics, conventionally and alternatively scored Fantasy points, position ranking, team ranking, and overall ranking. In this way, I could create a model independent of who the player was, focusing solely on how the stats contribute to performance. Some of the features I wish to incorporate in the future are discussed in the conclusion.

To determine goodness of fit, I used standard squared loss in backpropogation and I looked to see if the predicted value of points was within a 30 point delta of the actual points scored in the following year. Considering there were, on average, around 28000 points realistically distributed between around 100-150 players, I figured this was a good delta (around 10%). I used the Adaptive Momentum Optimizer, as it proved to be the best optimizer of the ones tested. I also empirically determined that a learning rate of 0.0001 was the best for this exercise.

I tested 5 different algorithms: a vanilla Neural Net with 1 layer with 69 nodes, a deep Neural Net with 10 69 node layers, a recurrent LSTM Neural Net, splitting up the vector into 12 discrete groups with an always 0 dummy variable appended, a convolutional Neural Net with 4 2x2 filters on the same 24 vector as above, and an optimized Neural Net that combined the above Neural Nets. I will discuss their performance in subsequent sections.

2 Relevant Works (and Acknowledgements)

Since this is a fairly bare project, there were very few papers I looked at on this specific topic, but there were certainly tutorials and discussions I looked at to figure out just what type of Neural Nets to use. They follow below, along with

a brief description of how I used them:
https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/recurrent_network.py: A tutorial on LSTM Recurrent Networks.
https://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html: Practical understanding of Recurrent Nets.
https://pdfs.semanticscholar.org/2e04/f58fa272a6eea79d23113f6a4743f8f53ac0.pdf: Sigmoid-Transfer Simple Neural Net Fantasy Football Prediction
http://static.pfref.com/: Fantasy Football reference with explanation of statistics.

3 Algorithms and Architectures

To figure out if there is anything relevant in the data collected, I tried five different algorithms. I will summarize their effectiveness in the following tables, and then discuss and delineate the specifics of their architectures.

The following table contains accuracy rates against epochs where the entire training set was sampled from in batches of 25 examples:

Epochs	Vanilla	Deep	Recurrent	Convolution	Combined
1	0	0.246445	0	0	0.210506
2	0.00236967	0.419431	0.014218	0	0.201915
3	0.007109	0.419431	0.0331754	0	0.208136
4	0.007109	0.454976	0.0402844	0	0.293049
5	0.00473934	0.175355	0.0402844	0	0.28594
6	0.0118483	0.424171	0.0402844	0.00236967	0.502468
7	0.00473934	0.43128	0.0521327	0	0.411829
8	0.00473934	0.445498	0.0331754	0.00947867	0.30233
9	0.0165877	0.537915	0.0545024	0.0947867	0.35387
10	0.014218	0.518957	0.056872	0.0829384	0.187401
11	0.035545	0.35545	0.056872	0.0473934	0.163013
12	0.0189573	0.516588	0.063981	0.0379147	0.562303
13	0.0189573	0.336493	0.0853081	0.042654	0.479463
14	0.00473934	0.552133	0.0829384	0.049763	0.415284
15	0.00473934	0.324645	0.0947867	0.0473934	0.493286
16	0.007109	0.518957	0.0805687	0.0473934	0.393562
17	0.00236967	0.50237	0.0853081	0.049763	0.125494
18	0.0189573	0.227488	0.0876777	0.0450237	0.550652
19	0.00473934	0.184834	0.0876777	0.0592417	0.339949
20	0	0.528436	0.0947867	0.0616114	0.457445
21	0.0189573	0.196682	0.0900474	0.0734597	0.581655

Epochs	Vanilla	Deep	Recurrent	Convolution	Combined
22	0.0165877	0.516588	0.106635	0.0758294	0.538803
23	0.021327	0.274882	0.116114	0.0805687	0.569115
24	0.00236967	0.327014	0.116114	0.0900474	0.539396
25	0.014218	0.201422	0.127962	0.0853081	0.33363
26	0	0.334123	0.135071	0.0853081	0.560723
27	0.014218	0.277251	0.14218	0.0900474	0.515008
28	0.014218	0.533175	0.132701	0.0971564	0.590936
29	0.0118483	0.381517	0.158768	0.104265	0.411039
30	0.00236967	0.317536	0.149289	0.116114	0.497433
31	0.0165877	0.483412	0.170616	0.127962	0.205371
32	0.014218	0.218009	0.175355	0.146919	0.218108
33	0.0189573	0.523697	0.194313	0.154028	0.347848
34	0.007109	0.542654	0.177725	0.156398	0.603673
35	0.007109	0.279621	0.191943	0.163507	0.590146
36	0.014218	0.2891	0.189573	0.170616	0.486967
37	0.0118483	0.490521	0.172986	0.175355	0.522709
38	0.0165877	0.485782	0.196682	0.180095	0.553614
39	0.0260663	0.305687	0.189573	0.184834	0.583037
40	0.021327	0.388626	0.203791	0.206161	0.518069
41	0.0236967	0.445498	0.2109	0.199052	0.550158
42	0.0331754	0.393365	0.199052	0.218009	0.165284
43	0.0189573	0.189573	0.21327	0.229858	0.319412
44	0.0236967	0.507109	0.208531	0.229858	0.29374
45	0.021327	0.49763	0.220379	0.236967	0.361769
46	0.0165877	0.526066	0.220379	0.236967	0.2656
47	0.0189573	0.191943	0.227488	0.241706	0.47344
48	0.021327	0.542654	0.220379	0.241706	0.286137
49	0.021327	0.478673	0.222749	0.241706	0.289593
50	0.0189573	0.514218	0.227488	0.251185	0.356931
51	0.0308057	0.528436	0.234597	0.241706	0.599427
52	0.0236967	0.464455	0.239336	0.253554	0.462283
53	0.021327	0.402844	0.246445	0.244076	0.590936
54	0.028436	0.545024	0.239336	0.239336	0.536039
55	0.0331754	0.400474	0.234597	0.246445	0.582543
56	0.0260663	0.334123	0.244076	0.246445	0.337579
57	0.0260663	0.263033	0.253554	0.248815	0.589356
58	0.035545	0.488152	0.258294	0.244076	0.562697
59	0.0165877	0.175355	0.251185	0.239336	0.423183
60	0.049763	0.42654	0.244076	0.236967	0.322077
61	0.0545024	0.154028	0.241706	0.244076	0.236078
62	0.035545	0.0995261	0.220379	0.244076	0.528239
63	0.0402844	0.388626	0.253554	0.246445	0.60782
64	0.042654	0.189573	0.248815	0.248815	0.59222
65	0.0379147	0.369668	0.265403	0.251185	0.57188
66	0.042654	0.490521	0.272512	0.255924	0.254542
67	0.0331754	0.495261	0.248815	0.255924	0.563882
68	0.0402844	0.533175	0.28673	0.255924	0.486276
69	0.049763	0.518957	0.251185	0.255924	0.284064

Epochs	Vanilla	Deep	Recurrent	Convolution	Combined
70	0.035545	0.521327	0.2891	0.255924	0.460407
71	0.056872	0.507109	0.253554	0.265403	0.266884
72	0.0402844	0.244076	0.229858	0.267773	0.416074
73	0.042654	0.488152	0.263033	0.274882	0.338369
74	0.0450237	0.488152	0.281991	0.272512	0.301244
75	0.035545	0.511848	0.265403	0.274882	0.247927
76	0.0545024	0.528436	0.300948	0.277251	0.493681
77	0.0402844	0.469194	0.296209	0.28673	0.336789
78	0.0450237	0.419431	0.248815	0.291469	0.273499
79	0.042654	0.395735	0.303318	0.28436	0.258096
80	0.0402844	0.490521	0.251185	0.2891	0.211197
81	0.042654	0.331754	0.2891	0.2891	0.273795
82	0.0473934	0.450237	0.277251	0.300948	0.178021
83	0.0521327	0.556872	0.265403	0.298578	0.284459
84	0.0521327	0.412322	0.293839	0.298578	0.489731
85	0.0331754	0.440758	0.270142	0.300948	0.546801
86	0.0545024	0.523697	0.251185	0.305687	0.517871
87	0.0521327	0.471564	0.277251	0.310427	0.344293
88	0.0900474	0.473934	0.267773	0.312796	0.530411
89	0.0545024	0.234597	0.296209	0.303318	0.537224
90	0.0592417	0.308057	0.248815	0.310427	0.27419
91	0.0734597	0.511848	0.263033	0.305687	0.401461
92	0.0545024	0.537915	0.274882	0.308057	0.328397
93	0.0473934	0.481043	0.279621	0.298578	0.308847
94	0.0734597	0.511848	0.324645	0.312796	0.10545
95	0.0592417	0.414692	0.2891	0.308057	0.343207
96	0.0545024	0.241706	0.312796	0.310427	0.539001
97	0.0781991	0.485782	0.334123	0.315166	0.334716
98	0.0876777	0.469194	0.255924	0.319905	0.410545
99	0.049763	0.504739	0.305687	0.317536	0.329878
100	0.0663507	0.485782	0.310427	0.319905	0.532385
Max Epoch	87	82	96	97	62
Max Accuracy	0.0900474	0.556872	0.334123	0.319905	0.60782

As we can see, the different Neural Nets performed differently, each improving on the vanilla Neural Net and the combined Neural Net behaving erratically, but also improving upon any single-technique Neural Net. Equations for the various architectures are as follows:

$$f_{vanilla} = w_{out} \tanh(w_0 x + b_0) + b_{out}$$
$$f_{deep} = w_{out} \tanh(h_{10}) + b_{out} \text{ where } h_{i+1} = \tanh(w_i h_i + b_i) \forall i \in 0, 1, .., 10$$
$$f_{recurrent} = RNN(LSTM, x), \text{ as this is implemented in TensorFlow.}$$
$$S(i,j)_{convolution} = \sum_{m=0}^{4} \sum_{n=0}^{6} x_i(i-m, j-n) K(m,n), \text{ where we pool with 2x2,}$$
and use stride 1x1 for 4 random filters, using TensorFlow methods.

Expanding on this, the vanilla architecture was a simple 1 layer Neural Net with 69 nodes. Nothing special here, and it wasn't really effective, achieving a 9% success rate.

The Deep architecture included the same basic Neural Net architecture, with the addition of 10 69-node layers in between the input and output layer. This was incredibly effective, achieving around a 55% success rate.

The recurrent Neural Net was implemented by TensorFlow, but was a LSTM NN using a bias of 1.0. It is static, and splits the 24 input vector into 12 2-vectors at each timestep. This proved to be fairly effective as there was a monotonic loss reduction, eventually arriving at a 35% success rate.

The convolutional Neural Net was interesting, as it did not seem to improve, and then suddenly had monotonic jumps in terms of success. It eventually converged to around a 30% success rate, which is pretty decent. I used 4 1x1 filters on a 4x6 "image" of the reshaped 24 vector. Then I used 2x2 pooling to condense the image and combine the results, and I used this to make the prediction.

The combined Neural Net, while chaotic, proved to be a slight to moderate improvement over the Deep Neural Net, as there were more peaks, but also lower valleys. I set it up by first convolving the "image", sending that through a deep net, and then using the recurrent Neural Net. Interestingly enough, invariance was not applicable as when I used convolution last, I achieved very poor results. In next iterations, I will try a vote by committee Neural Net too. This Neural Net was the best, achieving a success rate of around 60%.

4 Experiment and Error Analysis

Using just a simple squared loss function for backpropogation and using a threshold for prediction error, I was able to see some interesting results for how accuracy evolved. Commenting on the experimental design, I used batches of 25 inputs from pairwise years from 2012 to the current year (as before then is different, as the game has changed substantially), and then tested it on the most recent set of data. As described in the introduction, I used either a 23 or 24 vector to train against. Below is a plot of the error that was experienced as a result:

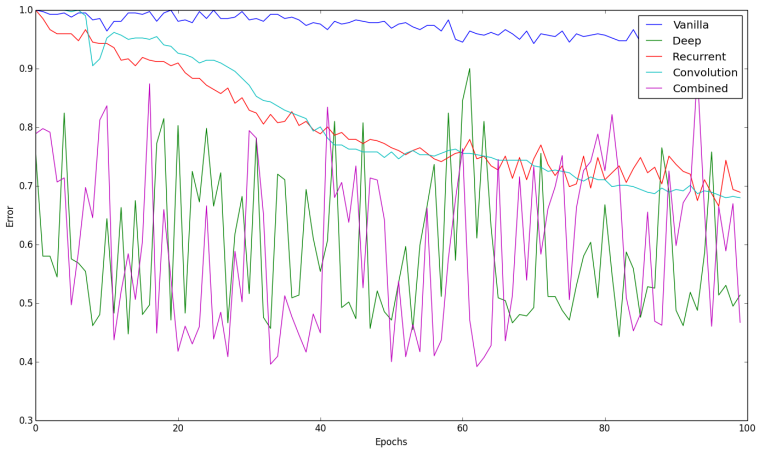

Figure 1: Loss on the various Neural Nets.

As we can see, there were crazy spikes with the two deep Nets. With the convolution and recurrent Nets, there is a monotonic and good error descent. With the vanilla net, there is a slight error descent. This is consistent with our ideas of how Neural Nets work, and is indicative that this data is not easily visualized.

5 Conclusion and Future Work

Based on the time I had to complete this project, I was not able to address every single one of the things I would like to try, but I am pleased with the results. Since I have been talking to a few professors about this idea, I plan to continue to research to see if I can improve upon the 60% success I have seen with this project. When the 2017 roster is finalized, I will even use this project to choose my Fantasy team, to see if this is actually viable. As I continue to research, I will try to develop classes and visualizations for different point ranges, so that I can assess loss in a more powerful way. Additionally, I would like to try new architectures, and understand which combinations provide both invariant and powerful results (at least 90%). Fantasy Football has always been a passion of mine, and it has been fun to see how I could improve by 6-10 times of a vanilla Neural Net by implementing simple Neural Net technique in such a real world example. Hopefully I can continue to learn more about Neural Nets (for example, if changing the ordering of my data vectors based on position has a bearing on the RNN), and turn this into a full fledged research project.

Summarizing the results from what I have tried, the rank of effectiveness of Neural Nets is as follows: 1. Combined, 2. Deep, 3. Recurrent, 4. Con-

volution, 5. Vanilla. While this may not seem revolutionary, most literature I've read claims 30-40% success rates, so I think this is an improvement. In terms of predicting these scores, there are many factors that are not captured that also play into these results, like injury proneness or diet, or even just general mood/temperament. This is why I'm fairly pleased with the results I've achieved, however, I can always improve! Therefore, there is much to be explored in the way of new architectures, and I look forward to seeing if I can find powerful new data and if there are even better ways to analyze such data.