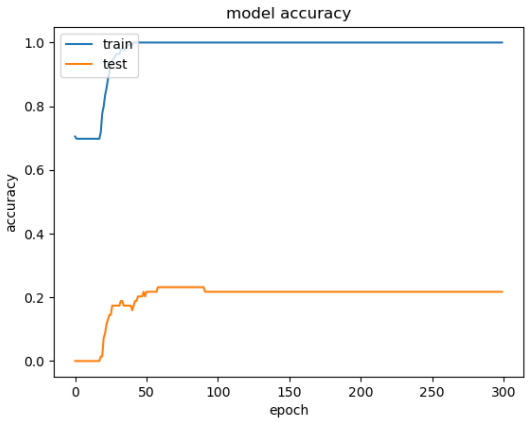
Deep Learning for Advanced Robot Perception Assignment 3: Parsing data and Regularization
Nathan Stallings
10/08/2021

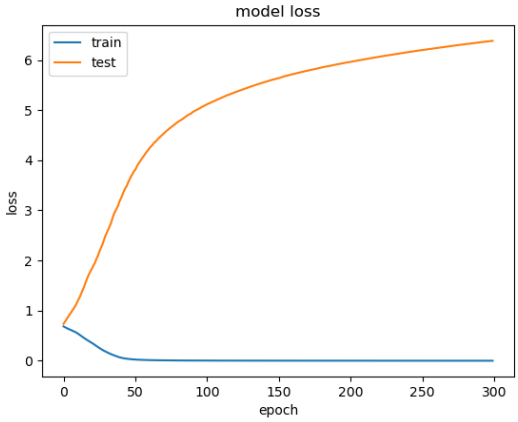
Baseline.py

The first implementation I tested was the baseline. This did not have any drop out and the un-modified code resulted in the following performance:

Baseline: 58.77% (10.22%)

The model's accuracy and loss plots looked like this:





To improve accuracy, the first thing I tried was modifying the batch size. I tried increasing the size to 32 to see how that would impact performance. Please find the results below:

Batch-size = 32
Baseline: 52.41% (9.30%)

Based on these results, it appears that a larger batch-size has a negative impact on the accuracy. To counteract this, I lowered the batch-size to 8 and got the following performance:

Batch-size = 8
Baseline: 55.38% (5.38%)

I then attempted to improve performance by changing the number of neurons and the number of layers. I tried many different versions but only documented a few significant ones. Here are my results:

```
model.add(Dense(60, input_dim=60, init='normal', activation='relu'))
model.add(Dense(30, init='normal', activation='relu'))
model.add(Dense(50, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Baseline: 52.91% (2.13%)

```
model.add(Dense(60, input_dim=60, init='normal', activation='relu'))
model.add(Dense(30, init='normal', activation='relu'))
model.add(Dense(10, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Baseline: 53.38% (1.23%)

I also attempted to modify the activation functions to see the impact that would have. Here are my results:

```
model.add(Dense(60, input_dim=60, init='normal', activation='relu'))
model.add(Dense(30, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='relu'))
```

Baseline: 14.76% (22.67%)

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu'))
model.add(Dense(30, init='normal', activation='selu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Baseline: 69.73% (10.03%)

It appears the selu activation function worked the best, so I implemented that with a few other changes. I went back to modifying the batch-size and the n_splits in the StratifiedKFold. After trying multiple approaches, I found these parameters produced the best results.

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu'))
model.add(Dense(30, init='normal', activation='selu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Baseline: 69.73% (10.03%)

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu'))
model.add(Dense(30, init='normal', activation='selu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
Batch_size = 8
```

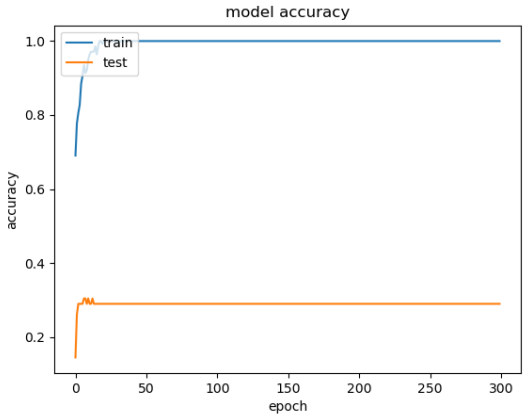
Baseline: 74.54% (8.09%)

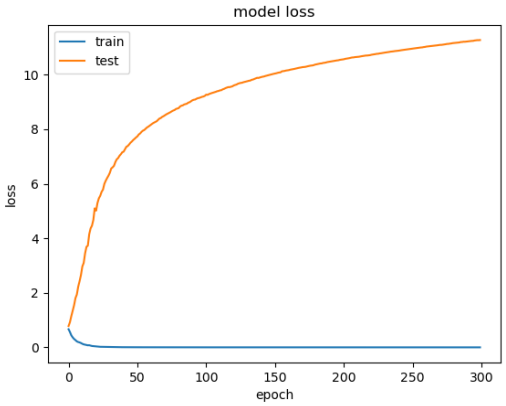
An accuracy of 77.04% (8.93%) was the best performance I achieved with the baseline model. This significantly outperforms the original accuracy of 58.77% (10.22%).

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu'))
model.add(Dense(30, init='normal', activation='selu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
Batch_size = 8
kfold = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
```

Baseline: 77.04% (8.93%)

The model's accuracy and loss plots looked like this:



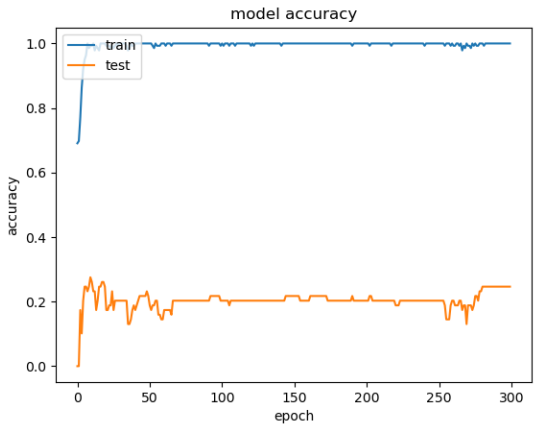


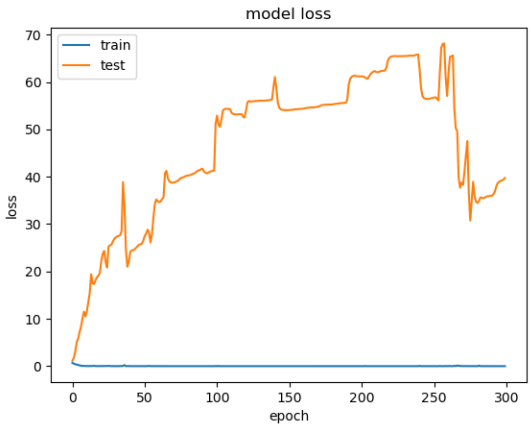
Dropout_hidden.py

The second implementation I tested was the dropout on the hidden layer. The un-modified code resulted in the following performance:

Hidden: 53.38% (1.23%)

The model's accuracy and loss plots looked like this:





Taking what I learned from the last test, my first change included modifying the activation function to selu and changing the batch_size to 8. This produces the following results.

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu', W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(30, init='normal', activation='selu', W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
batch_size=8
```

Hidden: 62.85% (22.44%)

I then further modified the batch size to see if I could improve the performance. I attempted different values until I found a sweet spot at 12 which provided the best accuracy with the changes already discussed.

Batch-size = 16
Hidden: 72.54% (5.52%)

Batch-size = 14
Hidden: 72.99% (8.21%)

Batch-size = 12
Hidden: 73.90% (8.46%)

I then attempted to improve performance by modifying the dropout probability. I found that higher probabilities decreased performance. An example can be found below:

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.8))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.8))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Hidden: 49.13% (13.92%)

After reverting the dropout probability of the first two layers back 0.2, I attempting to improve accuracy by changing the number of neurons and by adding layers. I used different dropout probabilities for the additional layers. I found the following results:

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.8))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.8))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Hidden: 49.13% (13.92%)

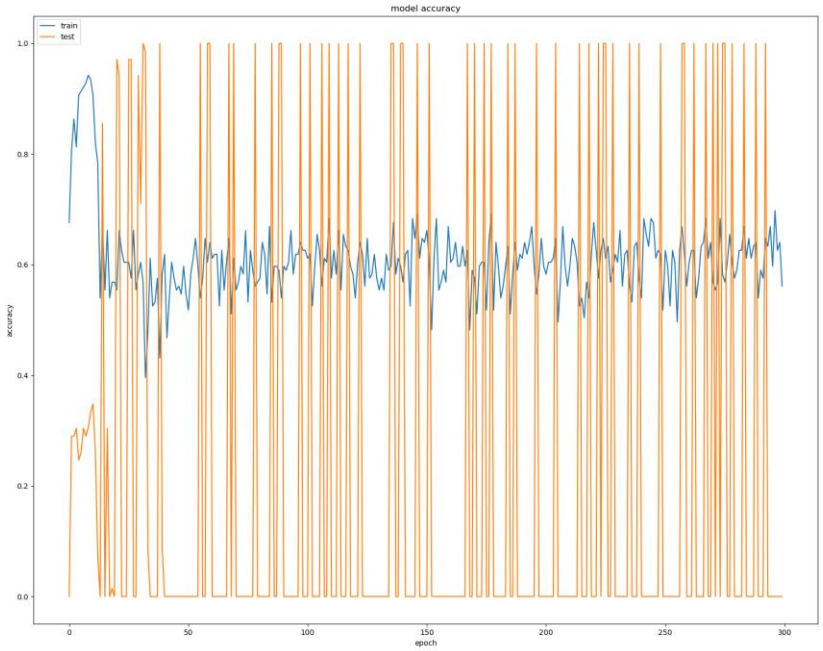
```
model.add(Dense(60, input_dim=60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.4))
kfold = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
```

Hidden: 77.54% (12.40%)

```
model = Sequential()
model.add(Dense(60, input_dim=60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(45, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(45, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Hidden: 75.12% (7.56%)

NOTE: If found that these models had high means, but the plots reveal a different story. The accuracy plot shows how unstable models with more than 3 layers are for this problem set. For this reason, I kept my final version of the model to 3 layers, which slightly decreased the mean but delivered a better and more consistent accuracy plot.



I also attempted to change the activation method again to see how that would impact performance:

```
model.add(Dense(60, input_dim=60, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(30, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(45, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(45, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Hidden: 53.38% (0.94%)

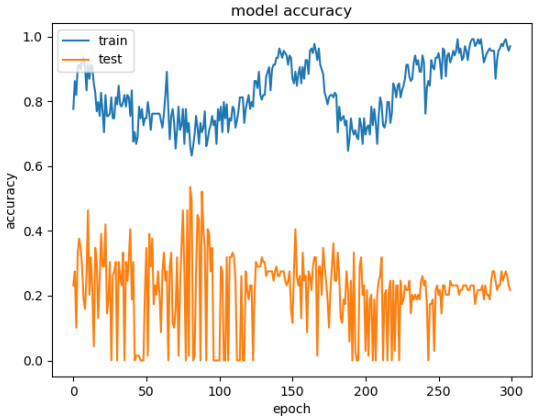
(Rest of page left blank on purpose, results on page 8)

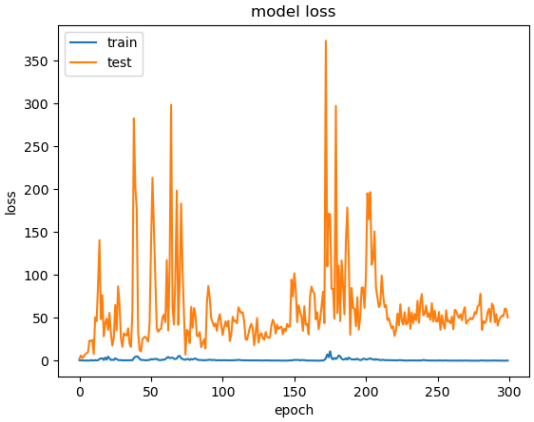
My best result was achieved by combining all of my findings above. I found that a dropout probability of 0.2 worked best with 3 layers (see Note on page 6), one which had 60 neurons, then 10, then 1. The activation function is selu and there is a batch size of 12, with n_splits of 15. I found that changing the number of epoch had minimal impact on the accuracy.

```
model.add(Dense(60, input_dim=60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(10, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(1, init='normal', activation='sigmoid'))
estimators.append(('mlp', KerasClassifier(build_fn=create_model,
nb_epoch=300, batch_size=12, verbose=0)))
kfold = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
```

Hidden: 75.12% (9.50%)

An accuracy of 75.12% (9.50%) was the best performance I achieved with this model (taking the plots into consideration). This significantly outperforms the original accuracy of 53.38% (1.23%). The model's accuracy and loss plots looked like this:



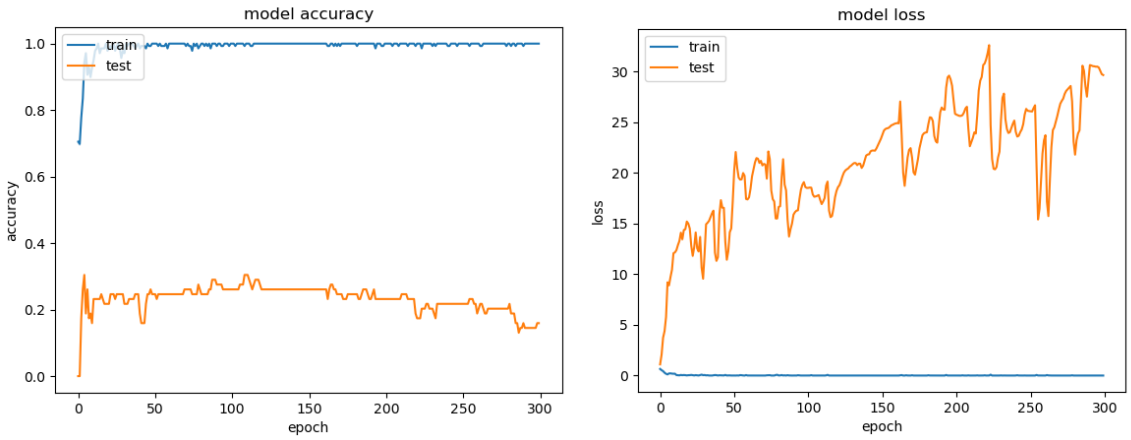


Dropout_visible.py

The third implementation I tested was the dropout on the visible layer. The un-modified code resulted in the following performance:

Visible: 55.38% (6.63%)

The model's accuracy and loss plots looked like this:



The first thing I tried was modifying the dropout probability. Like before, I found a higher dropout probability resulted in worse performance than a smaller drop out probability.

Dropout Probability = 0.6
Visible: 53.38% (1.23%)

Dropout Probability = 0.1
Visible: 55.38% (6.63%)

The next thing I attempted to change was the batch size. I found that this had a drastic impact on performance and that a batch size of 8 produced the best results. I also found that changing the epoch size had minimal impact.

Batch-size = 8
Visible: 70.11% (5.57%)

I found that changing the n_splits size for the StratifiedKFold improved performance slightly.

n_splits = 15
Visible: 77.92% (9.14%)

Like before, I assumed that selu would be the best activation function. I was surprised to see that they decreased accuracy for this version of dropout.

```
model.add(Dense(60, init='normal', activation='selu',
W_constraint=maxnorm(3)))
model.add(Dense(30, init='normal', activation='selu',
W_constraint=maxnorm(3)))
```

Visible: 67.30% (16.89%)

I attempted to increase performance by changing the number of neurons and by adding layers. I found that this only decreased performance, and an example can be seen in the following results:

```
model.add(Dropout(0.1, input_shape=(60,)))
model.add(Dense(60, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dense(30, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dense(20, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dense(1, init='normal', activation='sigmoid'))
```

Visible: 52.43% (2.54%)

(Rest of page left blank on purpose, results on page 11)

I found the best performance by combining all the techniques into one model. I had a dropout probability of 0.1, relu activation functions, a batch_size of 8, and n_splits as 15.

```
model.add(Dropout(0.1, input_shape=(60,)))
model.add(Dense(60, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dense(30, init='normal', activation='relu',
W_constraint=maxnorm(3)))
model.add(Dense(1, init='normal', activation='sigmoid'))
estimators.append(('mlp', KerasClassifier(build_fn=create_model,
nb_epoch=300, batch_size=8, verbose=0)))
kfold = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
```

Visible: 77.92% (9.14%)

An accuracy of 77.92% (9.14%) was the best performance I achieved with this model. This significantly outperforms the original accuracy of 55.38% (6.63%).