DeepREG

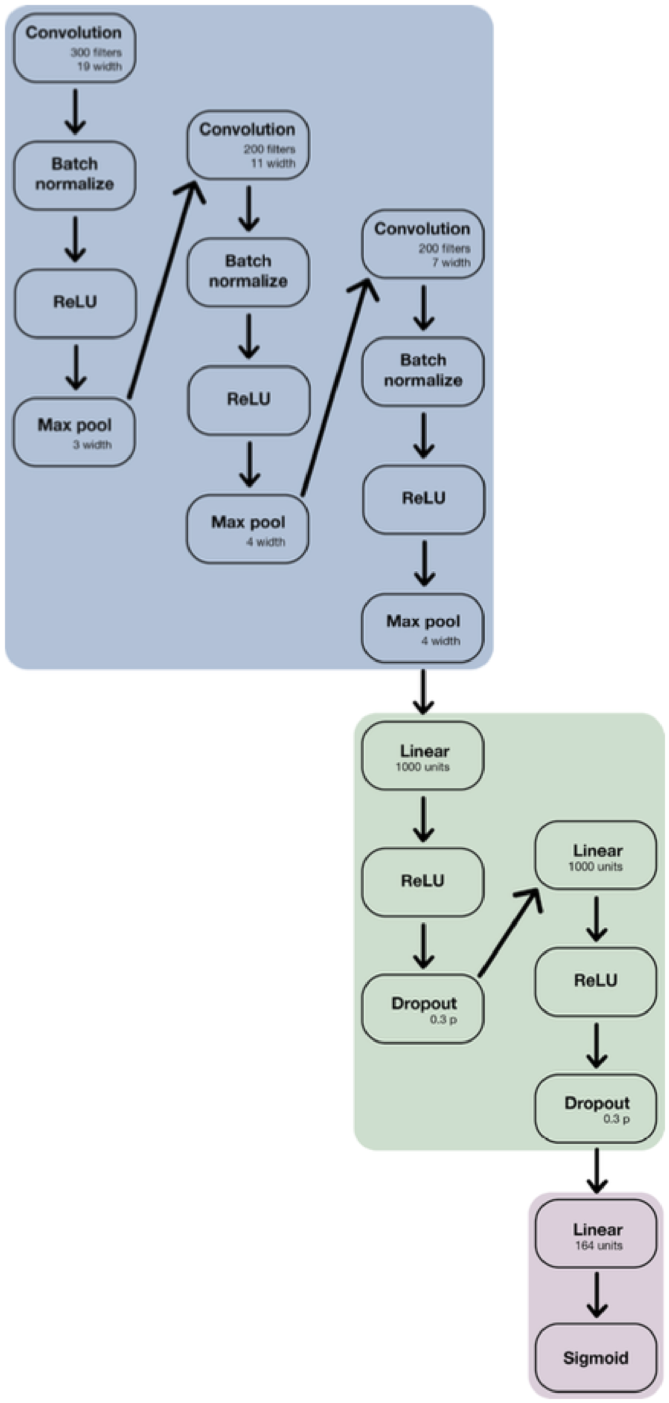
Boxiang Liu

July 27, 2017

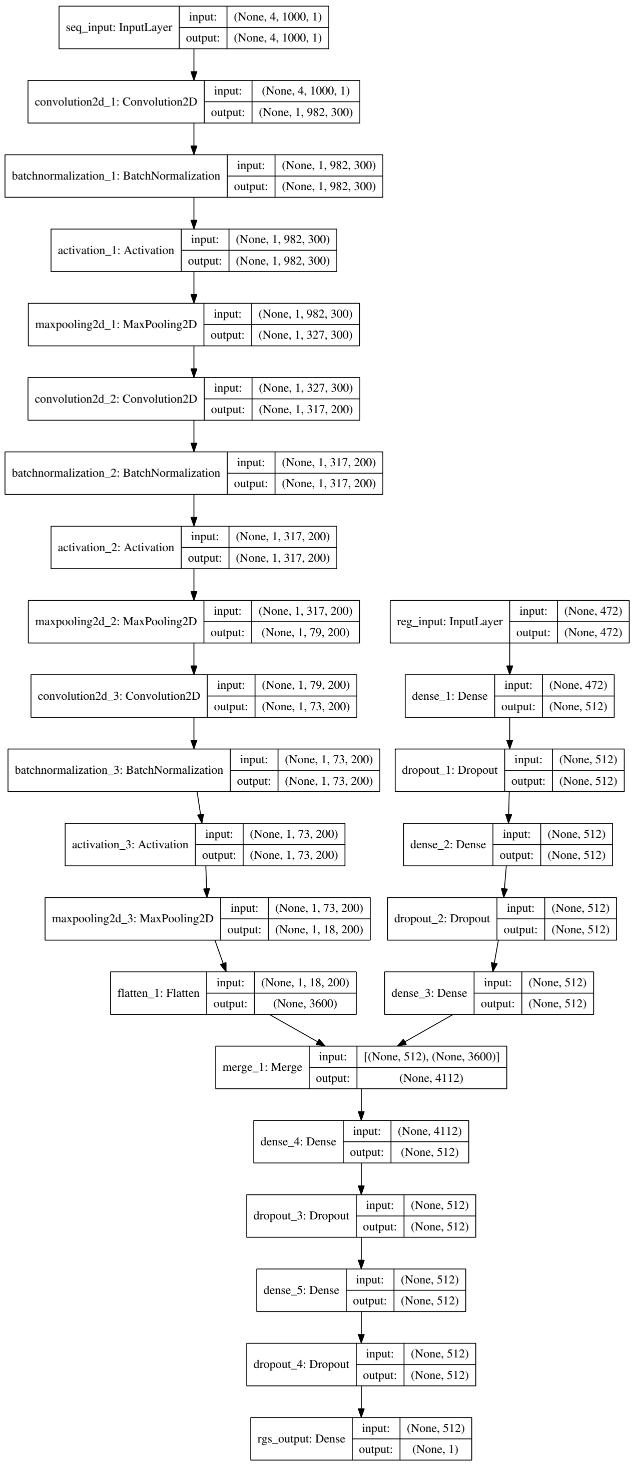
Contents

1 Basset

The Basset architecture represent the state-of-the-art for open chromatin predictions. The architecture is as follows:


Convolution
300 filters
19 width
Batch normalize
ReLU
Max pool
3 width
Convolution
200 filters
11 width
Batch normalize
ReLU
Max pool
4 width
Convolution
200 filters
7 width
Batch normalize
ReLU
Max pool
4 width
Linear
1000 units
ReLU
Dropout
0.3 p
Linear
1000 units
ReLU
Dropout
0.3 p
Linear
164 units
Sigmoid

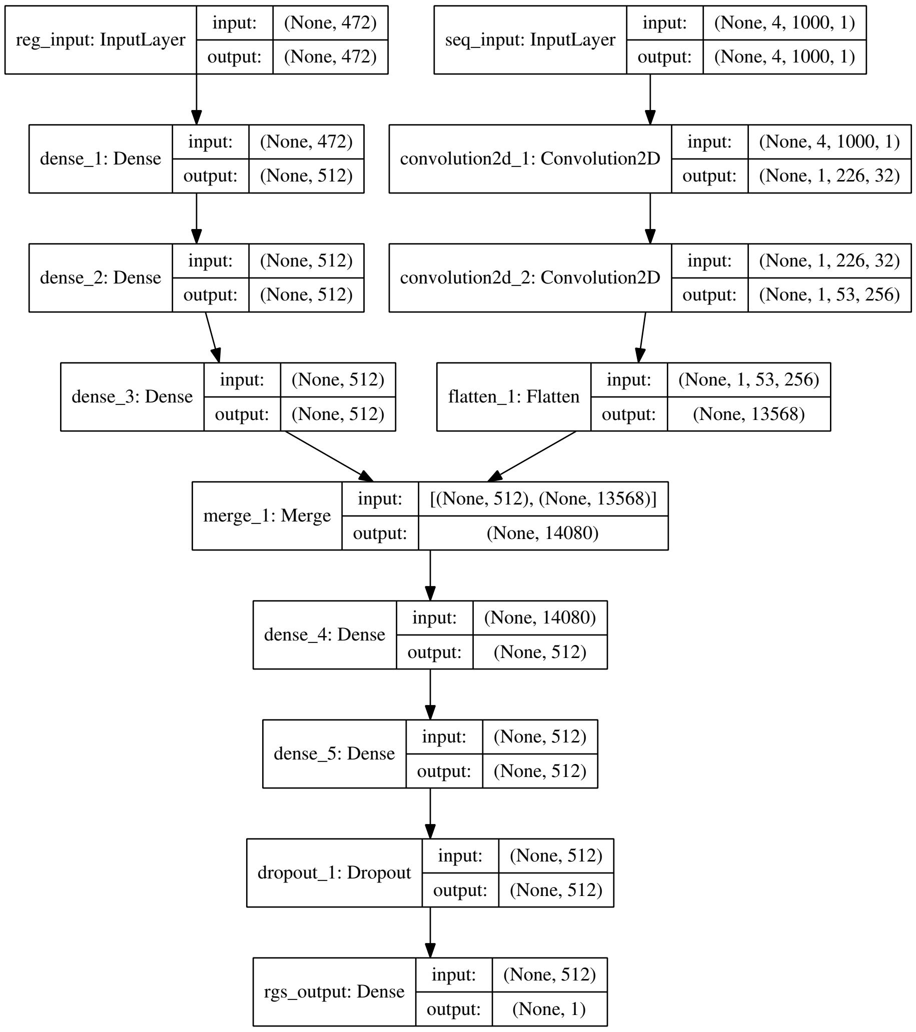
I used it for the CNN part of the network. The detailed network graph is below:


seq_input: InputLayer
input: (None, 4, 1000, 1)
output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D
input: (None, 4, 1000, 1)
output: (None, 1, 982, 300)
batchnormalization_1: BatchNormalization
input: (None, 1, 982, 300)
output: (None, 1, 982, 300)
activation_1: Activation
input: (None, 1, 982, 300)
output: (None, 1, 982, 300)
maxpooling2d_1: MaxPooling2D
input: (None, 1, 982, 300)
output: (None, 1, 327, 300)
convolution2d_2: Convolution2D
input: (None, 1, 327, 300)
output: (None, 1, 317, 200)
batchnormalization_2: BatchNormalization
input: (None, 1, 317, 200)
output: (None, 1, 317, 200)
activation_2: Activation
input: (None, 1, 317, 200)
output: (None, 1, 317, 200)
maxpooling2d_2: MaxPooling2D
input: (None, 1, 317, 200)
output: (None, 1, 79, 200)
reg_input: InputLayer
input: (None, 472)
output: (None, 472)
convolution2d_3: Convolution2D
input: (None, 1, 79, 200)
output: (None, 1, 73, 200)
dense_1: Dense
input: (None, 472)
output: (None, 512)
batchnormalization_3: BatchNormalization
input: (None, 1, 73, 200)
output: (None, 1, 73, 200)
dropout_1: Dropout
input: (None, 512)
output: (None, 512)
activation_3: Activation
input: (None, 1, 73, 200)
output: (None, 1, 73, 200)
dense_2: Dense
input: (None, 512)
output: (None, 512)
maxpooling2d_3: MaxPooling2D
input: (None, 1, 73, 200)
output: (None, 1, 18, 200)
dropout_2: Dropout
input: (None, 512)
output: (None, 512)
flatten_1: Flatten
input: (None, 1, 18, 200)
output: (None, 3600)
dense_3: Dense
input: (None, 512)
output: (None, 512)
merge_1: Merge
input: [(None, 512), (None, 3600)]
output: (None, 4112)
dense_4: Dense
input: (None, 4112)
output: (None, 512)
dropout_3: Dropout
input: (None, 512)
output: (None, 512)
dense_5: Dense
input: (None, 512)
output: (None, 512)
dropout_4: Dropout
input: (None, 512)
output: (None, 512)
rgs_output: Dense
input: (None, 512)
output: (None, 1)

However, the network did not train properly, likely due to too many layers.

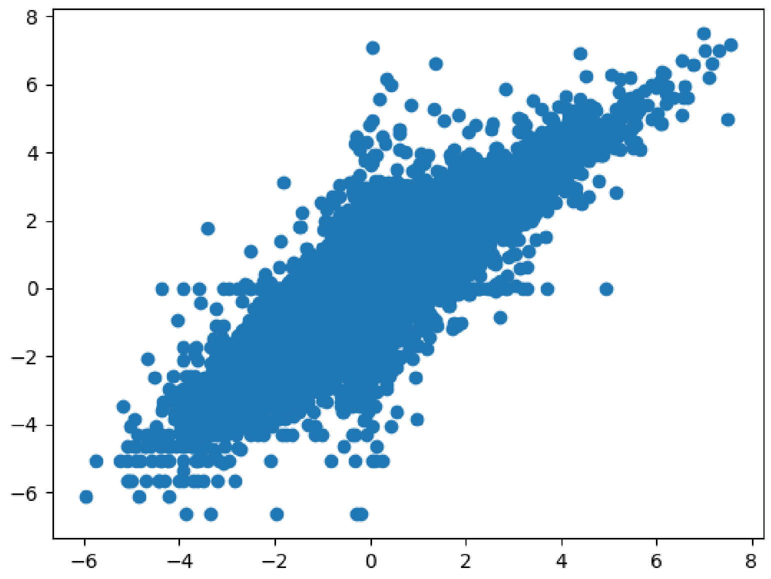
2 Reducing CNN layers

Given that 3 layers won't train, I removed one layer (in directory keras1).



reg_input: InputLayer | input: (None, 472) | output: (None, 472)
seq_input: InputLayer | input: (None, 4, 1000, 1) | output: (None, 4, 1000, 1)
dense_1: Dense | input: (None, 472) | output: (None, 512)
convolution2d_1: Convolution2D | input: (None, 4, 1000, 1) | output: (None, 1, 226, 32)
dense_2: Dense | input: (None, 512) | output: (None, 512)
convolution2d_2: Convolution2D | input: (None, 1, 226, 32) | output: (None, 1, 53, 256)
dense_3: Dense | input: (None, 512) | output: (None, 512)
flatten_1: Flatten | input: (None, 1, 53, 256) | output: (None, 13568)
merge_1: Merge | input: [(None, 512), (None, 13568)] | output: (None, 14080)
dense_4: Dense | input: (None, 14080) | output: (None, 512)
dense_5: Dense | input: (None, 512) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
rgs_output: Dense | input: (None, 512) | output: (None, 1)

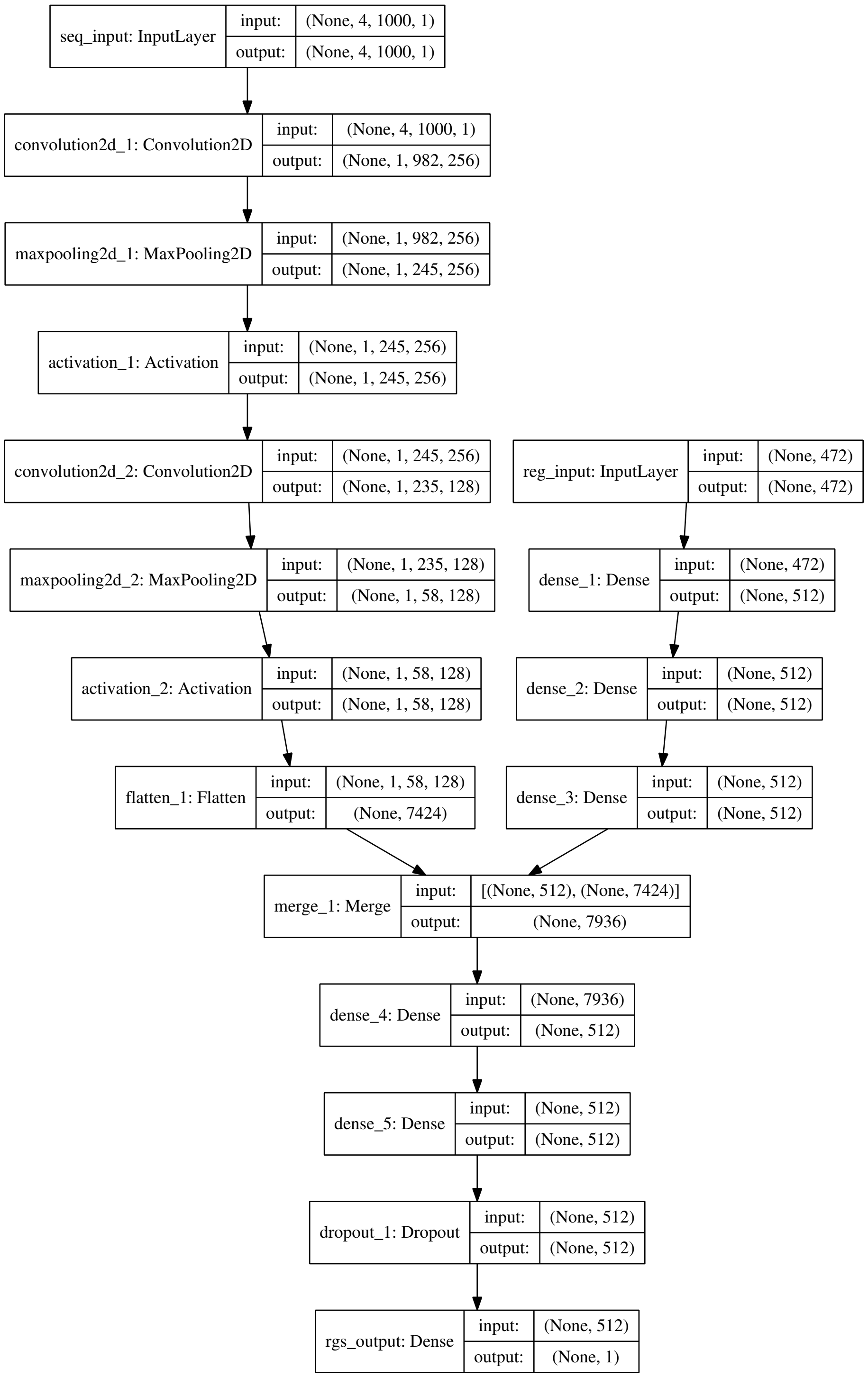
The result is quite promising.



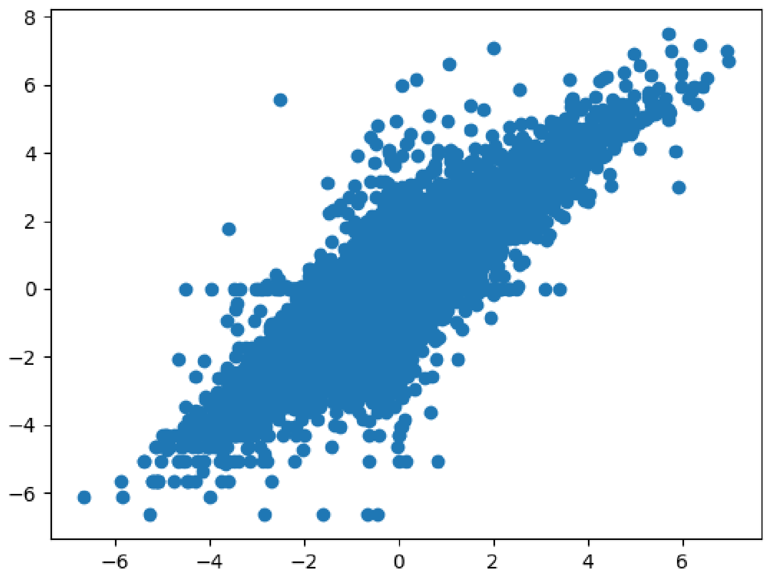
The first layer used filter width of 100, which is quite large.

3 Small filter

Since most motifs are less than 20 bps, I used a filter witdh of 19 bps.

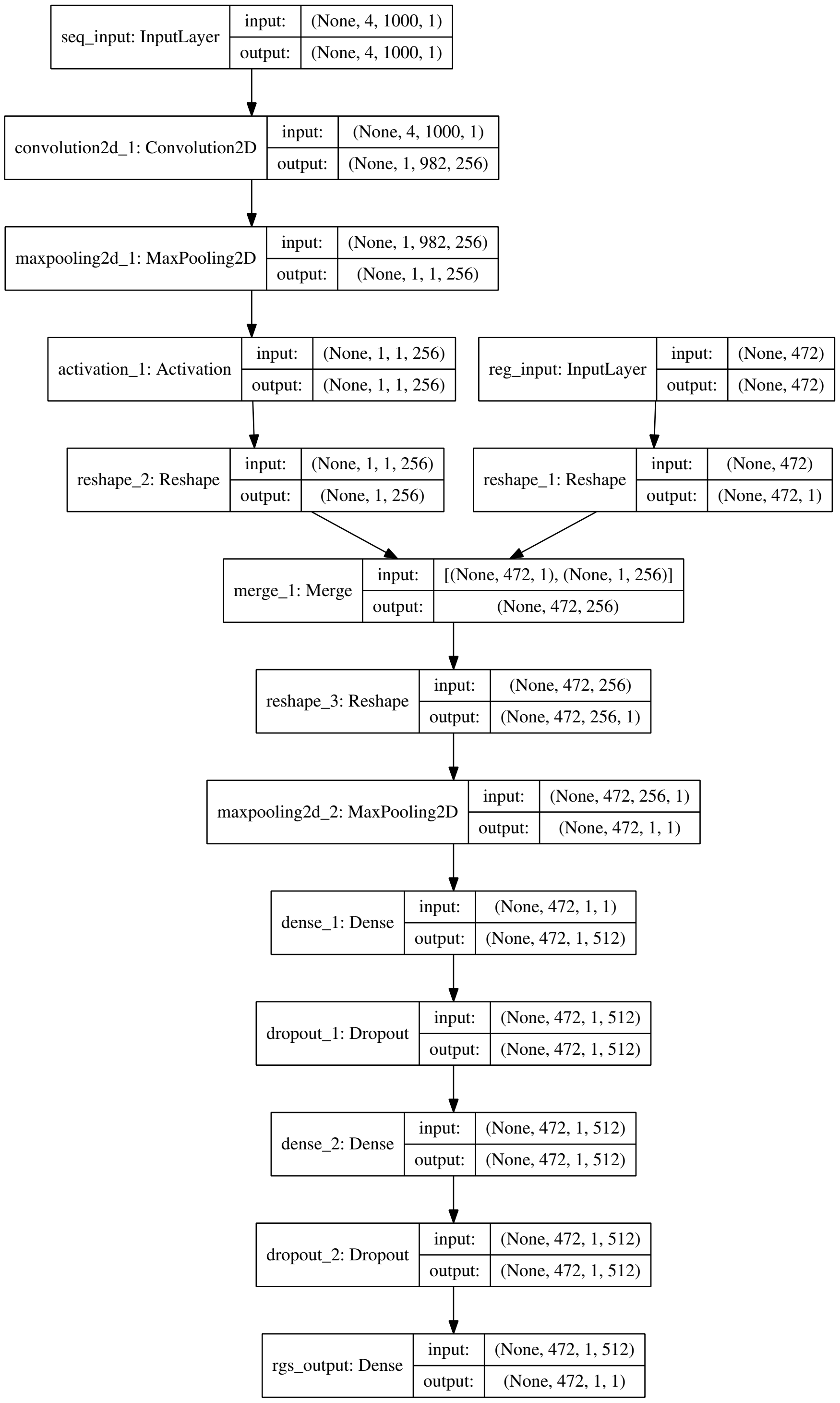

seq_input: InputLayer | input: (None, 4, 1000, 1) | output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D | input: (None, 4, 1000, 1) | output: (None, 1, 982, 256)
maxpooling2d_1: MaxPooling2D | input: (None, 1, 982, 256) | output: (None, 1, 245, 256)
activation_1: Activation | input: (None, 1, 245, 256) | output: (None, 1, 245, 256)
convolution2d_2: Convolution2D | input: (None, 1, 245, 256) | output: (None, 1, 235, 128)
maxpooling2d_2: MaxPooling2D | input: (None, 1, 235, 128) | output: (None, 1, 58, 128)
activation_2: Activation | input: (None, 1, 58, 128) | output: (None, 1, 58, 128)
flatten_1: Flatten | input: (None, 1, 58, 128) | output: (None, 7424)
reg_input: InputLayer | input: (None, 472) | output: (None, 472)
dense_1: Dense | input: (None, 472) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 512)
dense_3: Dense | input: (None, 512) | output: (None, 512)
merge_1: Merge | input: [(None, 512), (None, 7424)] | output: (None, 7936)
dense_4: Dense | input: (None, 7936) | output: (None, 512)
dense_5: Dense | input: (None, 512) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
rgs_output: Dense | input: (None, 512) | output: (None, 1)

The result is as good as using 100 width filters.

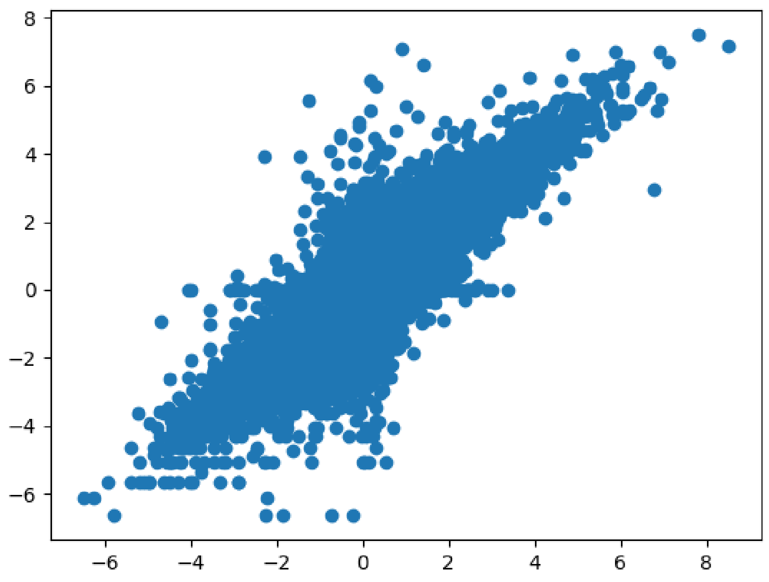


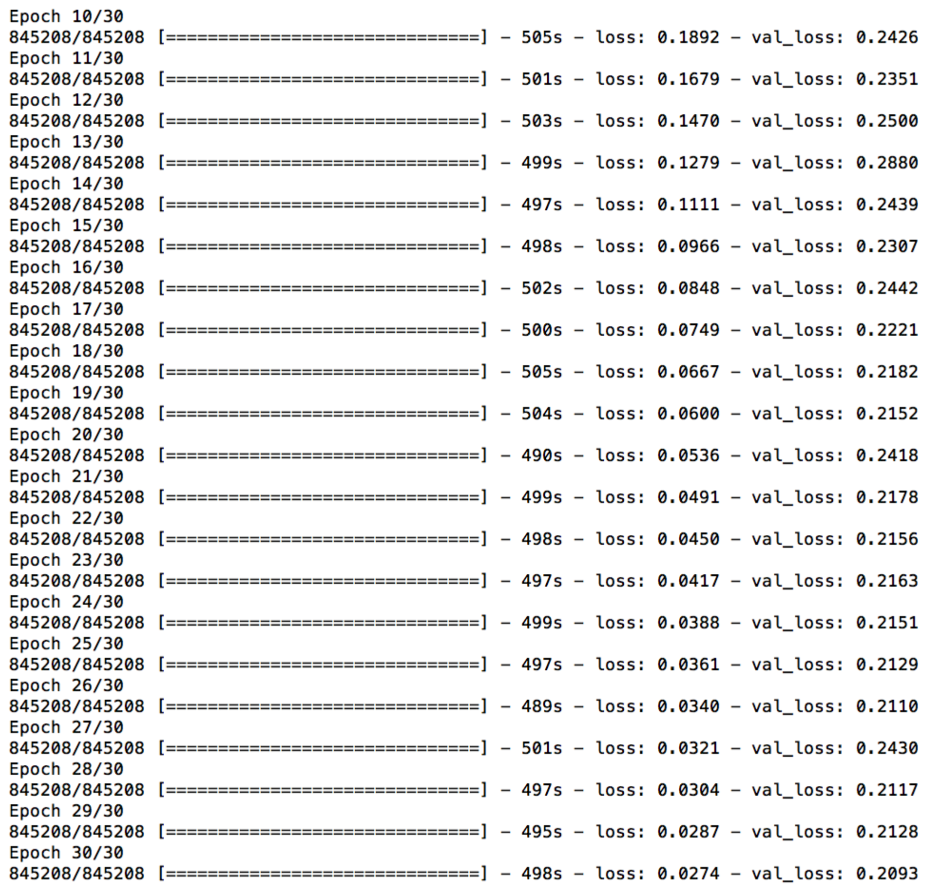
4 Single layer

When using more than one layer, either for the seq or the regulator network, the interpretability is lost. I therefore tried using one conv layer (as motif scanner) for the seq network, and no dense layer for the reg network.


seq_input: InputLayer
input: (None, 4, 1000, 1)
output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D
input: (None, 4, 1000, 1)
output: (None, 1, 982, 256)
maxpooling2d_1: MaxPooling2D
input: (None, 1, 982, 256)
output: (None, 1, 1, 256)
activation_1: Activation
input: (None, 1, 1, 256)
output: (None, 1, 1, 256)
reg_input: InputLayer
input: (None, 472)
output: (None, 472)
reshape_2: Reshape
input: (None, 1, 1, 256)
output: (None, 1, 256)
reshape_1: Reshape
input: (None, 472)
output: (None, 472, 1)
merge_1: Merge
input: [(None, 472, 1), (None, 1, 256)]
output: (None, 472, 256)
reshape_3: Reshape
input: (None, 472, 256)
output: (None, 472, 256, 1)
maxpooling2d_2: MaxPooling2D
input: (None, 472, 256, 1)
output: (None, 472, 1, 1)
dense_1: Dense
input: (None, 472, 1, 1)
output: (None, 472, 1, 512)
dropout_1: Dropout
input: (None, 472, 1, 512)
output: (None, 472, 1, 512)
dense_2: Dense
input: (None, 472, 1, 512)
output: (None, 472, 1, 512)
dropout_2: Dropout
input: (None, 472, 1, 512)
output: (None, 472, 1, 512)
rgs_output: Dense
input: (None, 472, 1, 512)
output: (None, 472, 1, 1)

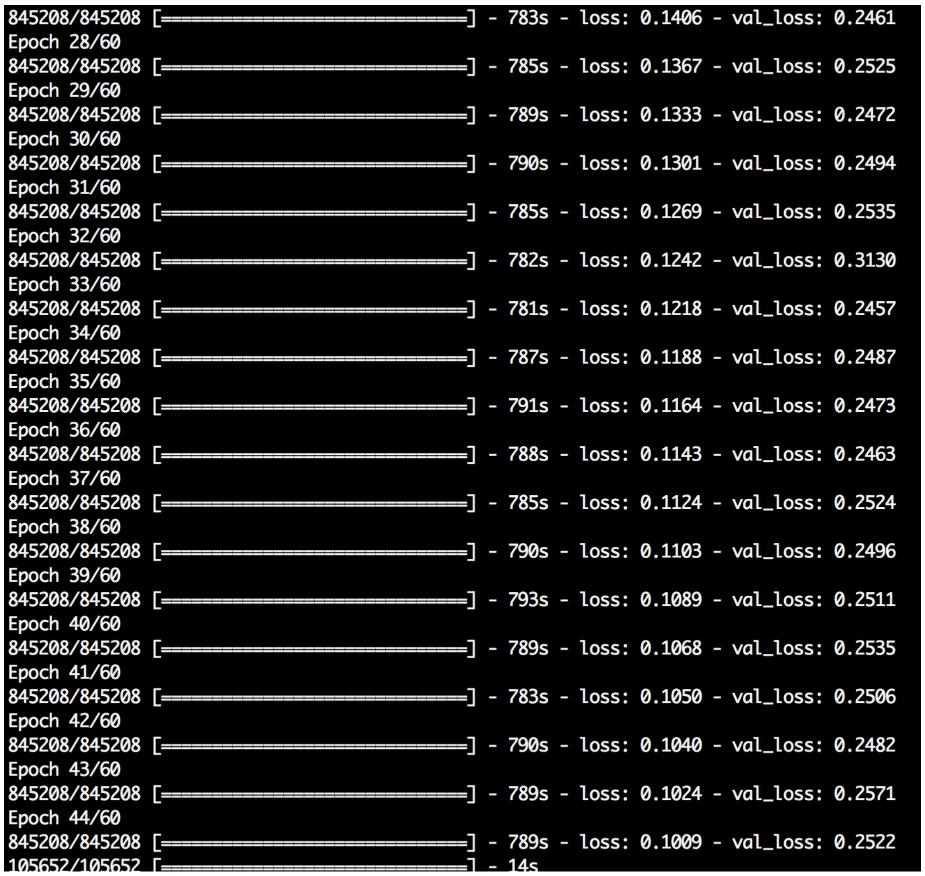
The model worked really well. The training loss dropped to almost zero after 30 epochs, and does not show any sign of plateau (compared to)small filter). However the test loss does not decrease as much indicating overfitting.





```
Epoch 10/30
845208/845208 [==============================] - 505s - loss: 0.1892 - val_loss: 0.2426
Epoch 11/30
845208/845208 [==============================] - 501s - loss: 0.1679 - val_loss: 0.2351
Epoch 12/30
845208/845208 [==============================] - 503s - loss: 0.1470 - val_loss: 0.2500
Epoch 13/30
845208/845208 [==============================] - 499s - loss: 0.1279 - val_loss: 0.2880
Epoch 14/30
845208/845208 [==============================] - 497s - loss: 0.1111 - val_loss: 0.2439
Epoch 15/30
845208/845208 [==============================] - 498s - loss: 0.0966 - val_loss: 0.2307
Epoch 16/30
845208/845208 [==============================] - 502s - loss: 0.0848 - val_loss: 0.2442
Epoch 17/30
845208/845208 [==============================] - 500s - loss: 0.0749 - val_loss: 0.2221
Epoch 18/30
845208/845208 [==============================] - 505s - loss: 0.0667 - val_loss: 0.2182
Epoch 19/30
845208/845208 [==============================] - 504s - loss: 0.0600 - val_loss: 0.2152
Epoch 20/30
845208/845208 [==============================] - 490s - loss: 0.0536 - val_loss: 0.2418
Epoch 21/30
845208/845208 [==============================] - 499s - loss: 0.0491 - val_loss: 0.2178
Epoch 22/30
845208/845208 [==============================] - 498s - loss: 0.0450 - val_loss: 0.2156
Epoch 23/30
845208/845208 [==============================] - 497s - loss: 0.0417 - val_loss: 0.2163
Epoch 24/30
845208/845208 [==============================] - 499s - loss: 0.0388 - val_loss: 0.2151
Epoch 25/30
845208/845208 [==============================] - 497s - loss: 0.0361 - val_loss: 0.2129
Epoch 26/30
845208/845208 [==============================] - 489s - loss: 0.0340 - val_loss: 0.2110
Epoch 27/30
845208/845208 [==============================] - 501s - loss: 0.0321 - val_loss: 0.2430
Epoch 28/30
845208/845208 [==============================] - 497s - loss: 0.0304 - val_loss: 0.2117
Epoch 29/30
845208/845208 [==============================] - 495s - loss: 0.0287 - val_loss: 0.2128
Epoch 30/30
845208/845208 [==============================] - 498s - loss: 0.0274 - val_loss: 0.2093
```

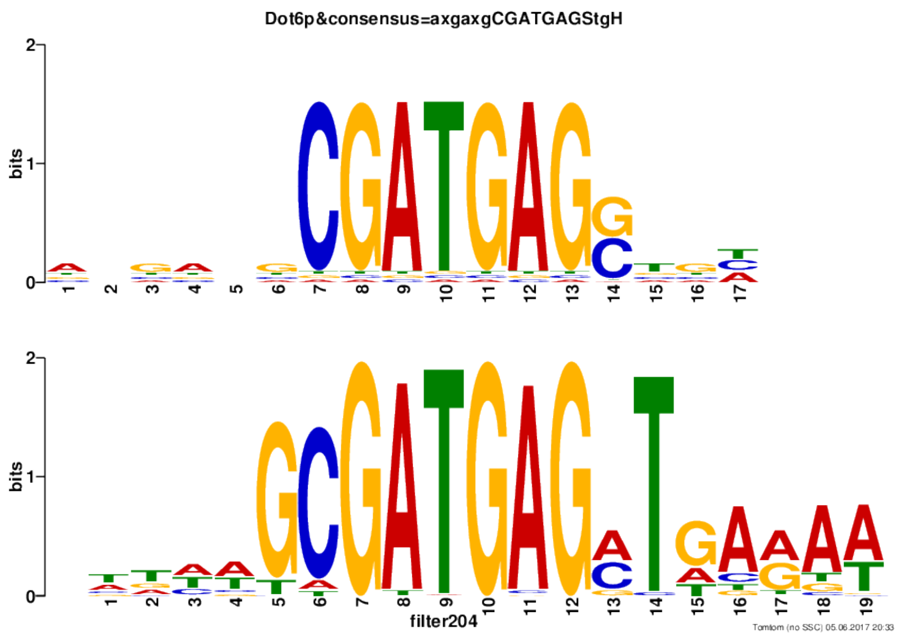
Therefore I created a new model with l1 (=1e-7) and l2 (=1e-7) on all weights regularization. Although this model prevented overfitting on the training set, the test set performance actually worsened.

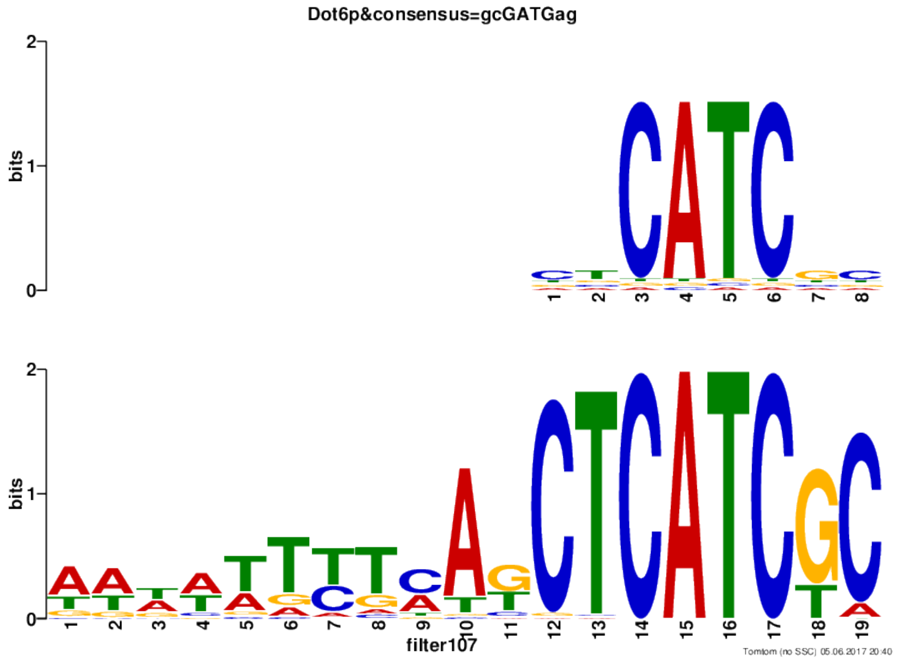


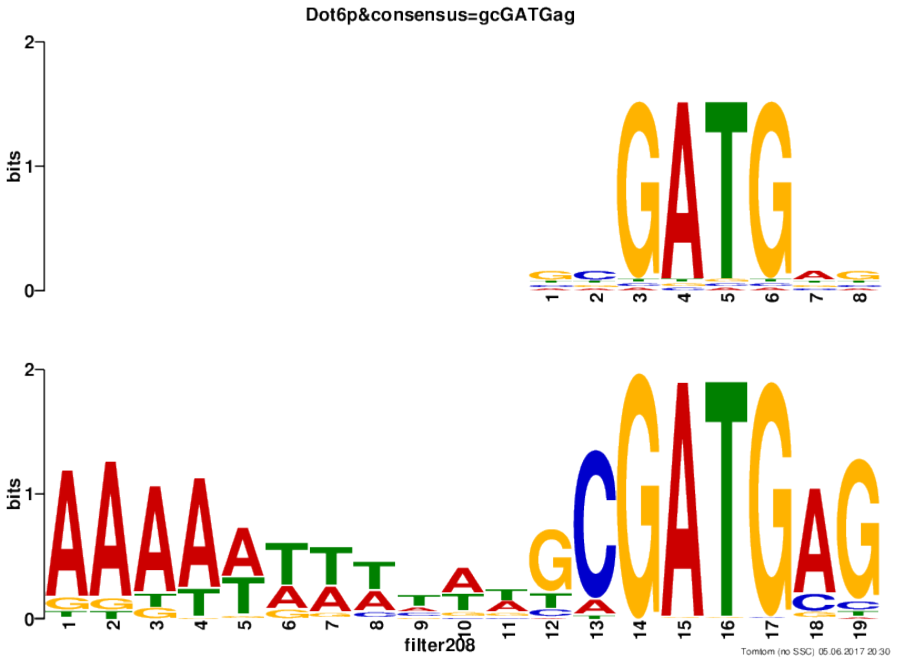
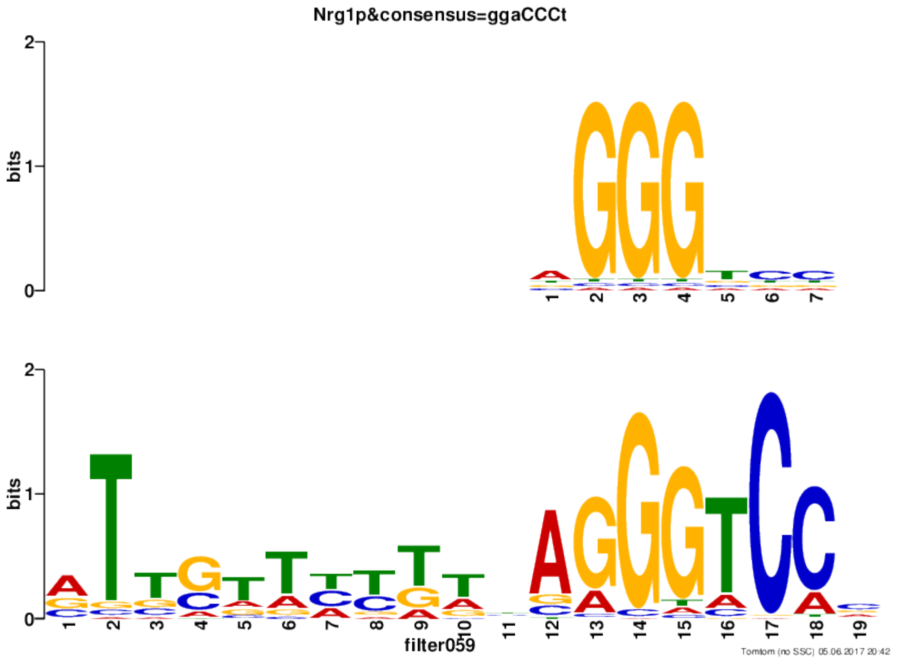
```
845208/845208 [==============================] - 783s - loss: 0.1406 - val_loss: 0.2461
Epoch 28/60
845208/845208 [==============================] - 785s - loss: 0.1367 - val_loss: 0.2525
Epoch 29/60
845208/845208 [==============================] - 789s - loss: 0.1333 - val_loss: 0.2472
Epoch 30/60
845208/845208 [==============================] - 790s - loss: 0.1301 - val_loss: 0.2494
Epoch 31/60
845208/845208 [==============================] - 785s - loss: 0.1269 - val_loss: 0.2535
Epoch 32/60
845208/845208 [==============================] - 782s - loss: 0.1242 - val_loss: 0.3130
Epoch 33/60
845208/845208 [==============================] - 781s - loss: 0.1218 - val_loss: 0.2457
Epoch 34/60
845208/845208 [==============================] - 787s - loss: 0.1188 - val_loss: 0.2487
Epoch 35/60
845208/845208 [==============================] - 791s - loss: 0.1164 - val_loss: 0.2473
Epoch 36/60
845208/845208 [==============================] - 788s - loss: 0.1143 - val_loss: 0.2463
Epoch 37/60
845208/845208 [==============================] - 785s - loss: 0.1124 - val_loss: 0.2524
Epoch 38/60
845208/845208 [==============================] - 790s - loss: 0.1103 - val_loss: 0.2496
Epoch 39/60
845208/845208 [==============================] - 793s - loss: 0.1089 - val_loss: 0.2511
Epoch 40/60
845208/845208 [==============================] - 789s - loss: 0.1068 - val_loss: 0.2535
Epoch 41/60
845208/845208 [==============================] - 783s - loss: 0.1050 - val_loss: 0.2506
Epoch 42/60
845208/845208 [==============================] - 790s - loss: 0.1040 - val_loss: 0.2482
Epoch 43/60
845208/845208 [==============================] - 789s - loss: 0.1024 - val_loss: 0.2571
Epoch 44/60
845208/845208 [==============================] - 789s - loss: 0.1009 - val_loss: 0.2522
105652/105652 [==============================] - 14s
```

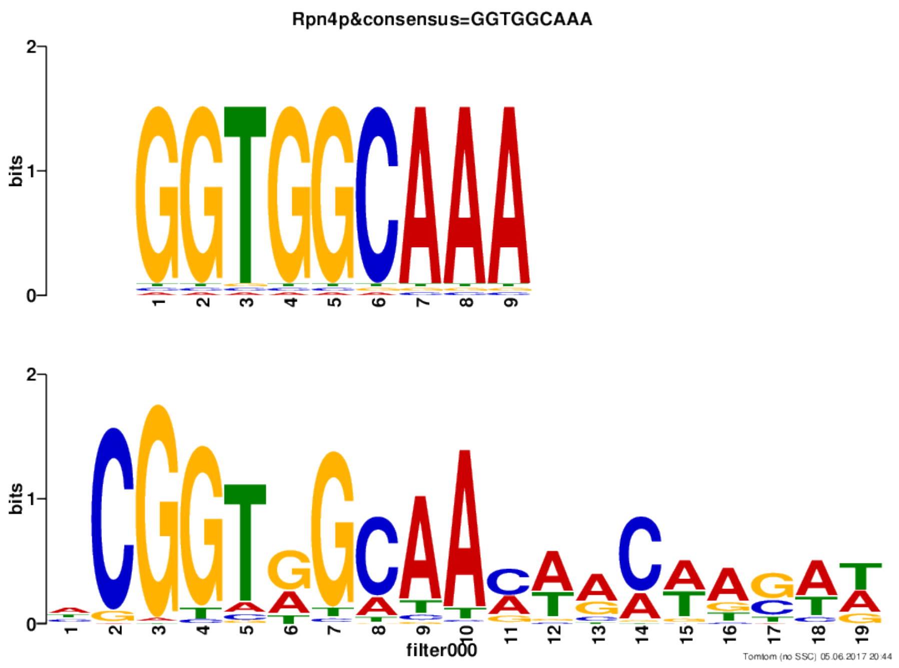
4.1 Motif discovery

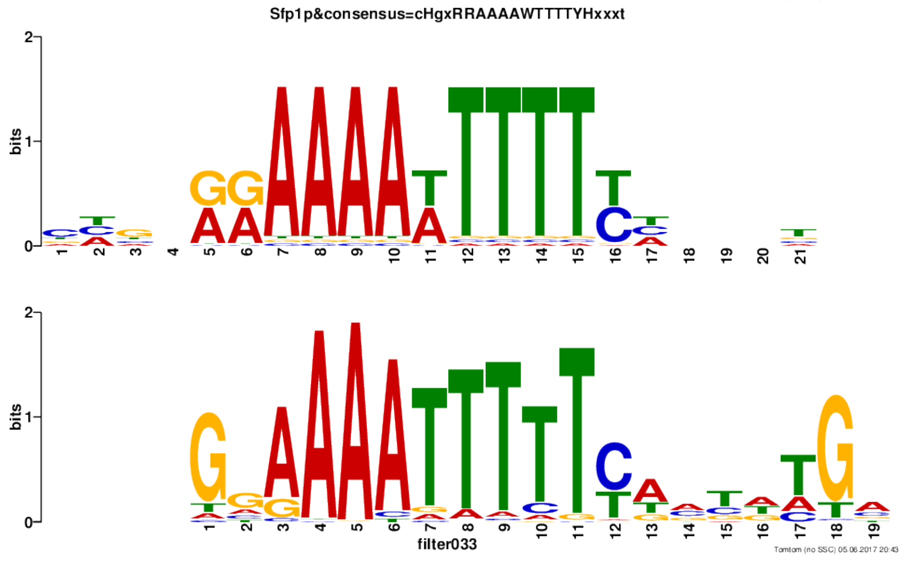
Is the network find known motifs? I took top 100 sequences with largest activation for each filter and use TomTom to match them to known motifs.

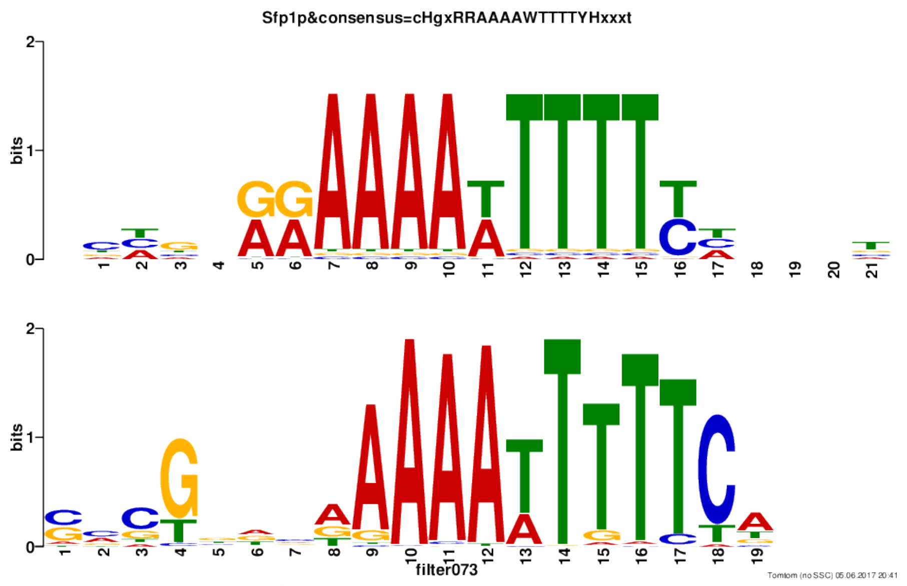

Dot6p&consensus=axgaxgCGATGAGStgH
bits
filter204
Tomtom (no SSC) 05.06.2017 20:33

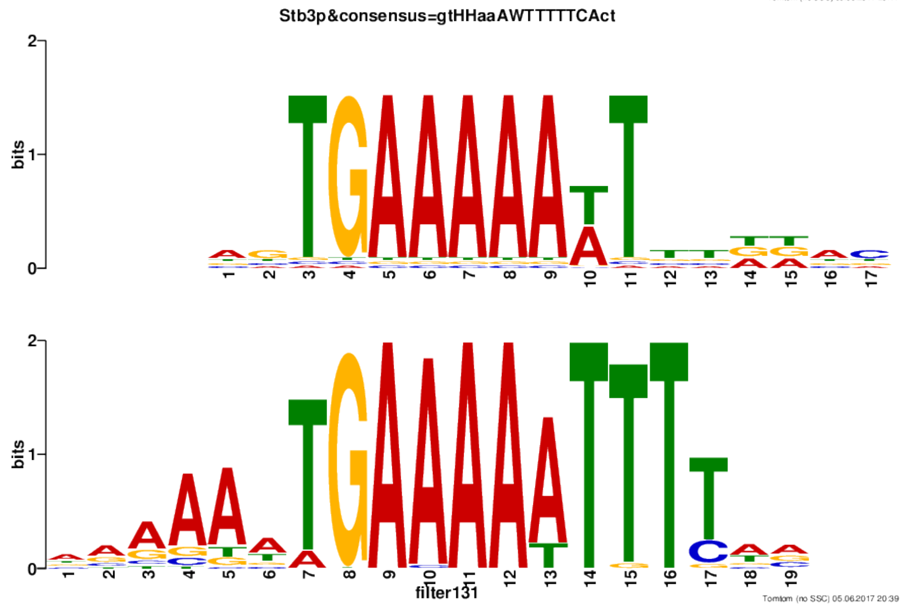

Dot6p&consensus=gcGATGag
bits
filter107
Tomtom (no SSC) 05.06.2017 20:40


Dot6p&consensus=gcGATGag
bits
filter208
Tomtom (no SSC) 05.06.2017 20:30

Nrg1p&consensus=ggaCCCt
bits
filter059
Tomtom (no SSC) 05.06.2017 20:42


Rpn4p&consensus=GGTGGCAAA
bits
filter000
Tomtom (no SSC) 05.06.2017 20:44


Sfp1p&consensus=cHgxRRAAAAWTTTTYHxxxt
bits
filter033
Tomtom (no SSC) 05.06.2017 20:43



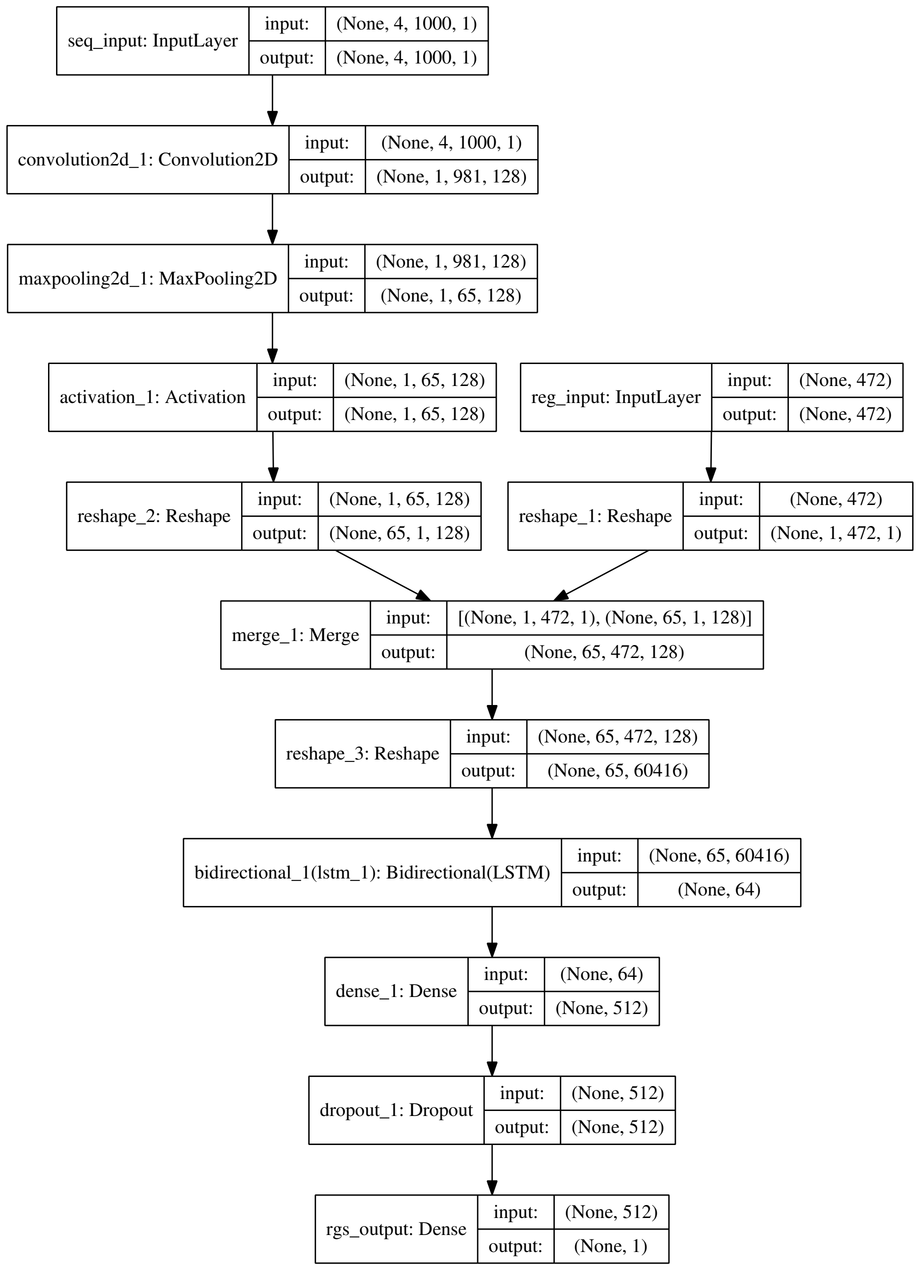


4.2 Maxpooling after single layer

The best validation loss is 0.4233 which is worse than just tensor product network.

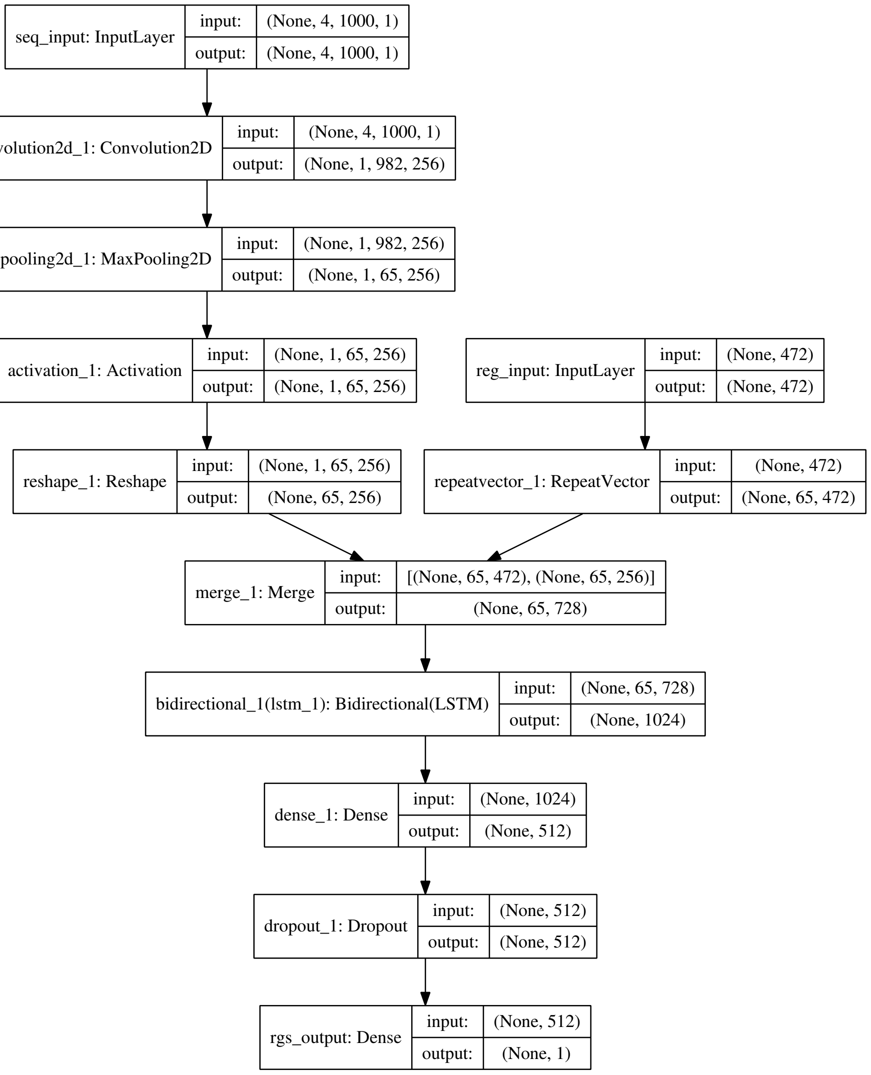
5 LSTM

Using outer product network ignores the spatial information along the genome. An LSTM with temporal dimension set to the genomic coordinate should be able to incorporate additional information in theory. The network is as follows:



seq_input: InputLayer | input: (None, 4, 1000, 1) | output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D | input: (None, 4, 1000, 1) | output: (None, 1, 981, 128)
maxpooling2d_1: MaxPooling2D | input: (None, 1, 981, 128) | output: (None, 1, 65, 128)
activation_1: Activation | input: (None, 1, 65, 128) | output: (None, 1, 65, 128)
reg_input: InputLayer | input: (None, 472) | output: (None, 472)
reshape_2: Reshape | input: (None, 1, 65, 128) | output: (None, 65, 1, 128)
reshape_1: Reshape | input: (None, 472) | output: (None, 1, 472, 1)
merge_1: Merge | input: [(None, 1, 472, 1), (None, 65, 1, 128)] | output: (None, 65, 472, 128)
reshape_3: Reshape | input: (None, 65, 472, 128) | output: (None, 65, 60416)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (None, 65, 60416) | output: (None, 64)
dense_1: Dense | input: (None, 64) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
rgs_output: Dense | input: (None, 512) | output: (None, 1)

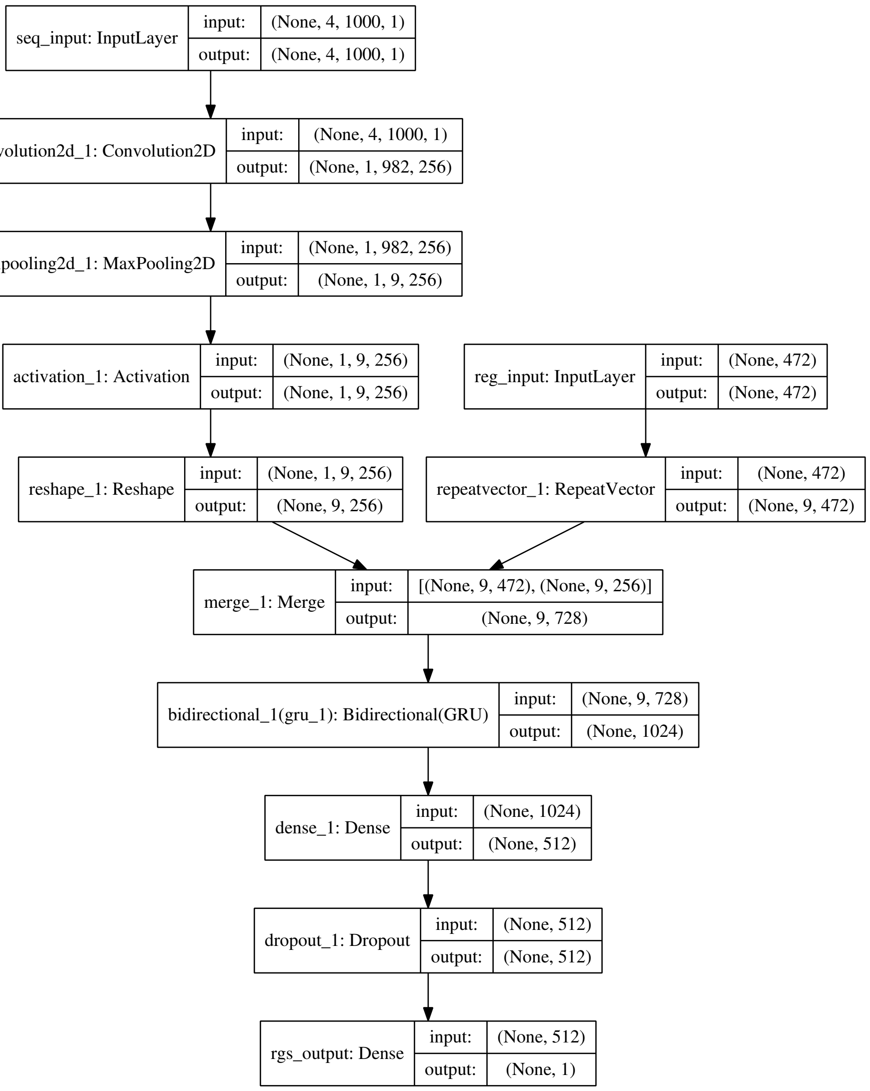
This initial model uses an outer product as the input dimension and the genomic coordinate as the time dimension. The size of input dimension is 256*472=120832, too large to fit into memory of GeForce GTX 970. A simpler variant replaces the outer product dimension with concatenation, effective reducing the memory requirement by two orders of magnitude.



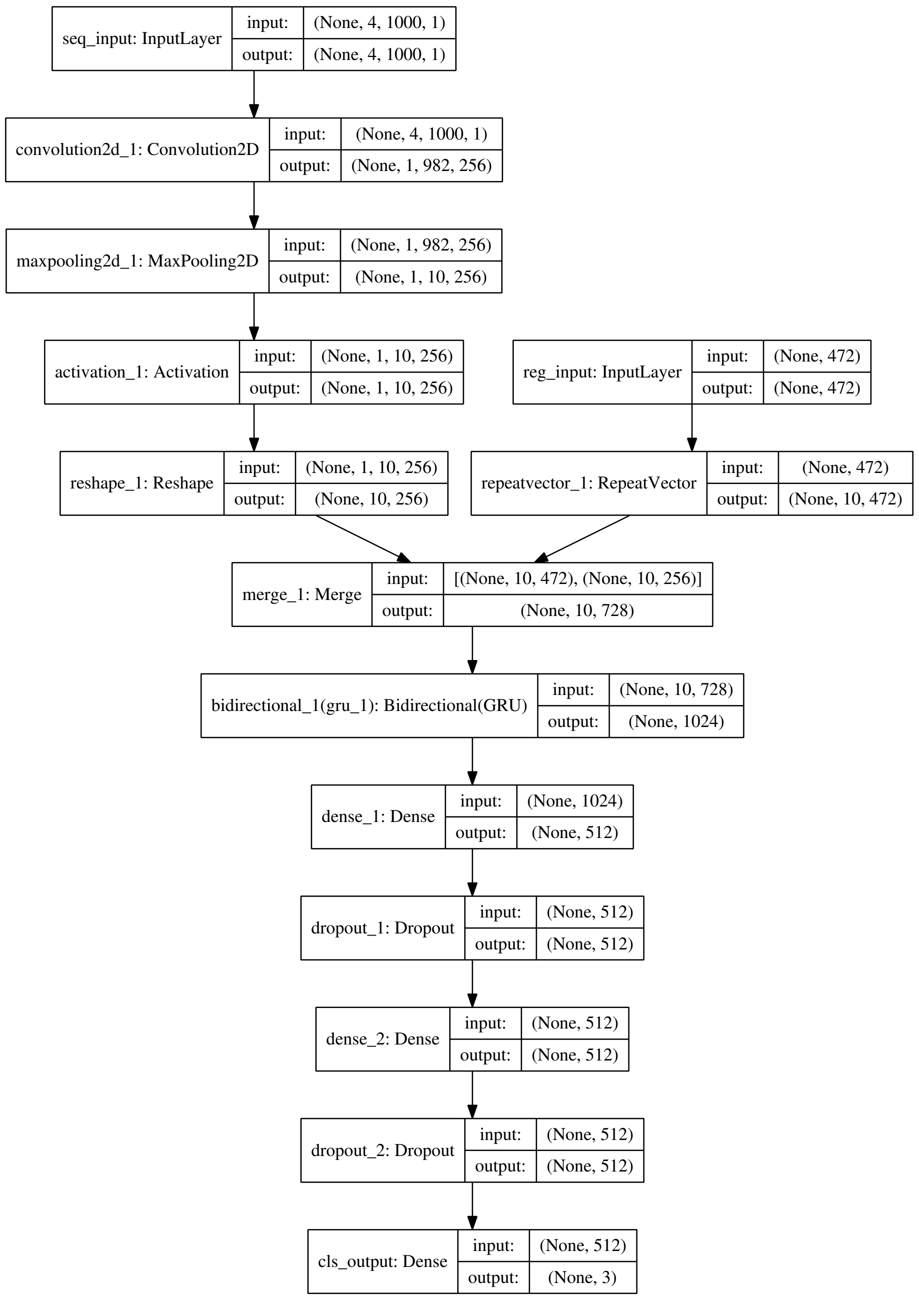
The concatenation model was trained with both SGD and ADAM. ADAM worked better than SGD. The best validation loss for LSTM is 0.3799, worse than 0.21 for tensor product network.

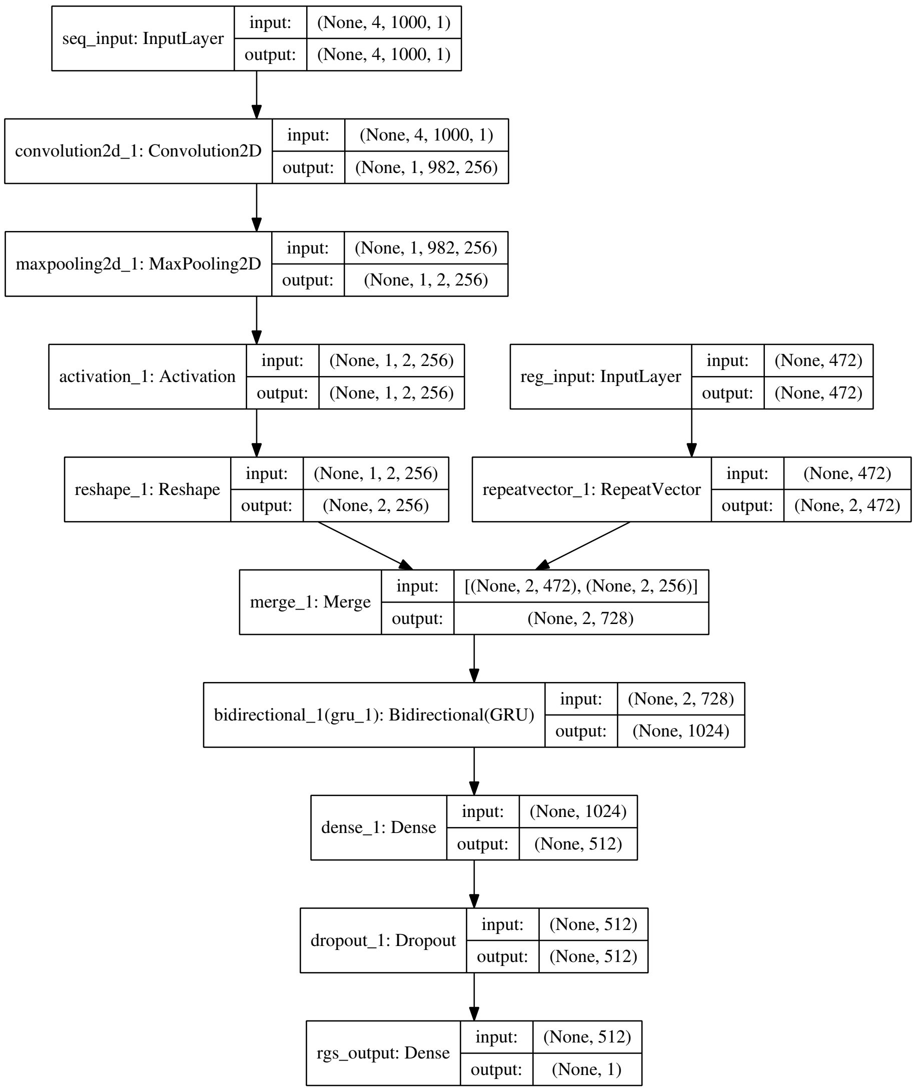
6 GRU

Since the GRU uses two gates, one gate less than LSTM, it is believed to be more computational efficient. I replaced the LSTM with GRU, and used maxpool = 15, 100, 491. In theory, when we increase the subsample ratio to 982, the GRU should become equivalent to the concatenation network.



seq_input: InputLayer | input: (None, 4, 1000, 1) | output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D | input: (None, 4, 1000, 1) | output: (None, 1, 982, 256)
maxpooling2d_1: MaxPooling2D | input: (None, 1, 982, 256) | output: (None, 1, 9, 256)
activation_1: Activation | input: (None, 1, 9, 256) | output: (None, 1, 9, 256)
reg_input: InputLayer | input: (None, 472) | output: (None, 472)
reshape_1: Reshape | input: (None, 1, 9, 256) | output: (None, 9, 256)
repeatvector_1: RepeatVector | input: (None, 472) | output: (None, 9, 472)
merge_1: Merge | input: [(None, 9, 472), (None, 9, 256)] | output: (None, 9, 728)
bidirectional_1(gru_1): Bidirectional(GRU) | input: (None, 9, 728) | output: (None, 1024)
dense_1: Dense | input: (None, 1024) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
rgs_output: Dense | input: (None, 512) | output: (None, 1)


seq_input: InputLayer
input: (None, 4, 1000, 1)
output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D
input: (None, 4, 1000, 1)
output: (None, 1, 982, 256)
maxpooling2d_1: MaxPooling2D
input: (None, 1, 982, 256)
output: (None, 1, 10, 256)
activation_1: Activation
input: (None, 1, 10, 256)
output: (None, 1, 10, 256)
reg_input: InputLayer
input: (None, 472)
output: (None, 472)
reshape_1: Reshape
input: (None, 1, 10, 256)
output: (None, 10, 256)
repeatvector_1: RepeatVector
input: (None, 472)
output: (None, 10, 472)
merge_1: Merge
input: [(None, 10, 472), (None, 10, 256)]
output: (None, 10, 728)
bidirectional_1(gru_1): Bidirectional(GRU)
input: (None, 10, 728)
output: (None, 1024)
dense_1: Dense
input: (None, 1024)
output: (None, 512)
dropout_1: Dropout
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 512)
dropout_2: Dropout
input: (None, 512)
output: (None, 512)
cls_output: Dense
input: (None, 512)
output: (None, 3)



seq_input: InputLayer | input: (None, 4, 1000, 1) | output: (None, 4, 1000, 1)
convolution2d_1: Convolution2D | input: (None, 4, 1000, 1) | output: (None, 1, 982, 256)
maxpooling2d_1: MaxPooling2D | input: (None, 1, 982, 256) | output: (None, 1, 2, 256)
activation_1: Activation | input: (None, 1, 2, 256) | output: (None, 1, 2, 256)
reg_input: InputLayer | input: (None, 472) | output: (None, 472)
reshape_1: Reshape | input: (None, 1, 2, 256) | output: (None, 2, 256)
repeatvector_1: RepeatVector | input: (None, 472) | output: (None, 2, 472)
merge_1: Merge | input: [(None, 2, 472), (None, 2, 256)] | output: (None, 2, 728)
bidirectional_1(gru_1): Bidirectional(GRU) | input: (None, 2, 728) | output: (None, 1024)
dense_1: Dense | input: (None, 1024) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
rgs_output: Dense | input: (None, 512) | output: (None, 1)

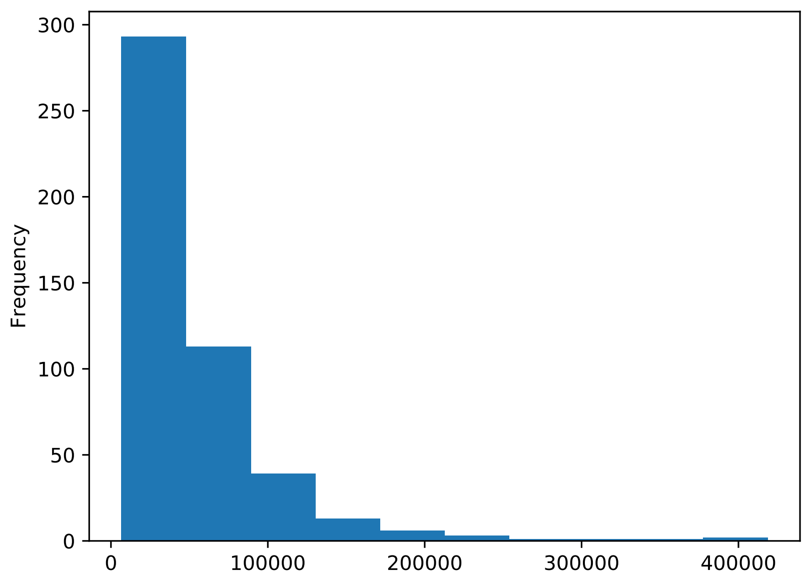
Performance-wise, the best validation loss for GRU is 0.3556, worse than 0.21 for concatenation network.

7 Gene-Gene relationship

8 deepLIFT

I rerun the **concatenation** network with *valid* padding (see *modeling/concatnation*) and calculated deeplift scores with respect to the regulator layer. Each of the 472 regulators receives 1056511 scores, one for each [gene,conditions] pair (there are

173 conditions × 6107 genes). I used the sums of absolute values as the overall importance score for each regulator.



The distribution is highly skewed to the right, indicating that several important (potentially master regulators) dominates the model weights.

The top 10 regulators are

1. GAC1 / YOR178C
2. RAS1 / YOR101W
3. USV1 / YPL230W
4. MSN2 / YMR037C
5. PDR3 / YBL005W
6. YVH1 / YIR026C
7. PRR2 / YDL214C
8. SLT2 / YHR030C
9. BAS1 / YKR099W
10. SIP2 / YGL208W

Within these, SIP2 / YGL208W, SLT2/YHR030C, USV1 / YPL230W, GAC1 / YOR178C, MSN2 / YMR037C are also noted in Kundaje (2006) as top parents.

Since MSN2 is a known master regulator with over a hundred known targets, we tested whether MSN2 have higher deepLIFT scores for known targets. We downloaded a total of 381 genes from (http://www.yeastgenome.org/locus/S000004640/overview). Indeed, the known target has higher deepLIFT scores other genes (50.2 vs 49.9, p¡0.0016).