Deep learning for genomics II

Matt Ploenzke

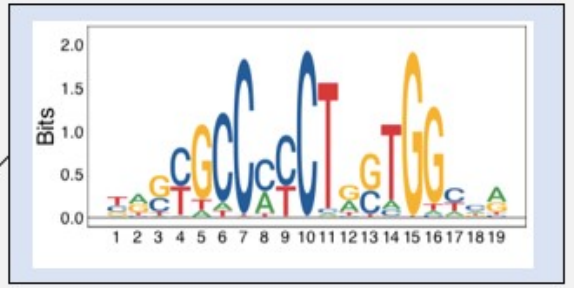
BST 261: Data Science II
Department of Biostatistics
Harvard University

May 1, 2019

Snapshot

- Problem formulation
  - *Where do we get the data?*
- Data setup
  - *What do the data look like?*
- Opening the black box
  - *What has the model learned?*
- Review
- Worked example with code

A) Transcription factors (TFs) bind to the genome according to sequence specificities characterized by consensus binding motifs.



CTCF

RNA Polymerase

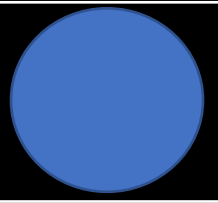
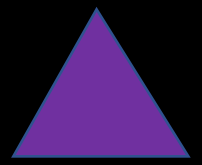
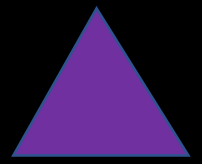
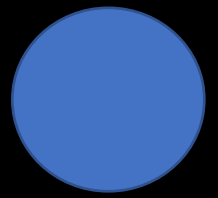
Next generation sequencing

In ChIP-seq, only sequences bound to the shape (DNA-binding protein/transcription factor) are sequenced

Want to learn how DNA sequences arising from different samples are different

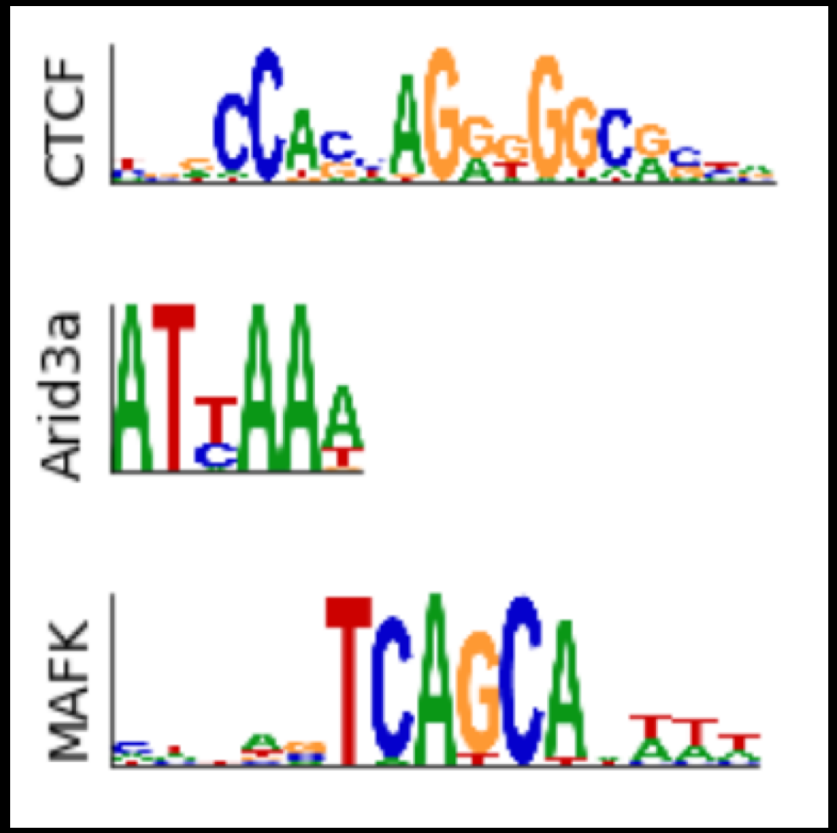
In Dnase-seq, all sequences bound to any shape are sequenced

Can our model achieve high accuracy and what does it learn?

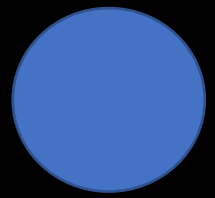
Sequence (X)	Label (Y)
ACTAATGAGAGTAGTTTTATACTGGTATCGCTGCTAAGAGCTTGATATAAACATCTCAACAGCCTTGATGGAATAAAGGGG GGGGCAAATACTCCGCGTTTAGGAGTATGTTGCCTCATAGCTTTGTAGGATAGAGCAGGGGCCGATAAAATTAACCATTTC AAGGTTTTGCCACTAACATAGAATGAACACCGTTAAGA	
TGCAATGACGATACTACAAAATTCTATAGACGTATGCGGAATTCCCCCCCCCAAGCACCTCCGCACGGTATAACATTAACATT TTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTATTTCGTATTACTGTGTGTCTATGTGACTCTCACCTTACAGAACCA GAAAGTGCAATTGGATATCTTCGACGAGTTAAA	
GCGAAAGACCCCCCCCTAATAGCCTCAGACTTCATTCACGCTTGTGGGTGGATTCCACCTATAGGACGTACTTACGGCCTAT TTAAGCAAAGCCAGAGAGAGGTTTTTTAATCGATGTTTTGGGTCACACATTCAAGCTATGTCGGCTTTAATGAGTTTATTCC TTGGAGACATCAGTCTATGTGGCTTACGAAGGAAA	
ATCAAAATAGAAGCCAGACACTTTGACCAAAAATTACTTGGTCATTGCTAAAATAGCCCTACATAGGAAAATAAAAGCAGA TTACTTTCAGATAGCAACAAGAACAGTGACTCCAGCATTCAAGTCAAACAAATTACGCAGTATGGGGGGGGGTATTAAGC GTTATGTGGGAACTGCGAGATCATCTCATTTGCCAGCAA	
⋮	⋮

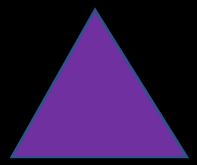
Sequence (X)	Label (Y)
ACTAATGAGAGTAGTTTTATACTGGTATCGCTGCTAAGAGCTTGATATAAACATCTCAACAGCCTTGATGGAATAAAGGGGGGGGCAAATACTCCGCGTTTAGGAGTATGTTGCCTCATAGCTTTGTAGGATAGAGCAGGGGCCGATAAAATTAACCATTTCAAGGTTTTGCCACTAACATAGAATGAACACCGTTAAGA	1
TGCAATGACGATACTACAAAATTCTATAGACGTATGCGGAATTCCCCCCCCCAAGCACCTCCGCACGGTATAACATTAACATTTTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTATTTCGTATTACTGTGTGTCTATGTGACTCTCACCTTACAGAACCAGAAAGTGCAATTGGATATCTTCGACGAGTTAAA	0
GCGAAAGACCCCCCCCCTAATAGCCTCAGACTTCATTCACGCTTGTGGGTGGATTCCACCTATAGGACGTACTTACGGCCTATTTAAGCAAAGCCAGAGAGAGGTTTTTTAATCGATGTTTTGGGTCACACATTCAAGCTATGTCGGCTTTAATGAGTTTATTCCTTGGAGACATCAGTCTATGTGGCTTACGAAGGAAA	0
ATCAAAATAGAAGCCAGACACTTTGACCAAAAATTACTTGGTCATTGCTAAAATAGCCCTACATAGGAAAATAAAAGCAGATTACTTTCAGATAGCAACAAGAACAGTGACTCCAGCATTCAAGTCAAACAAATTACGCAGTATGGGGGGGGGGTATTAAGCGTTATGTGGGAACTGCGAGATCATCTCATTTGCCAGCAA	1
⋮	⋮

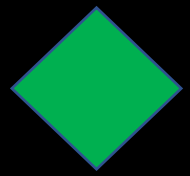
Embedded motif

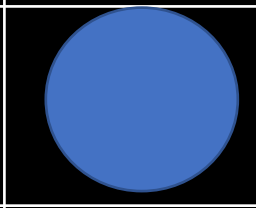
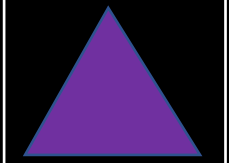
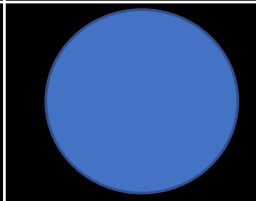
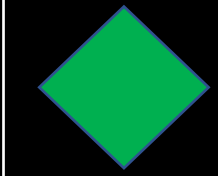


Shape/Sample

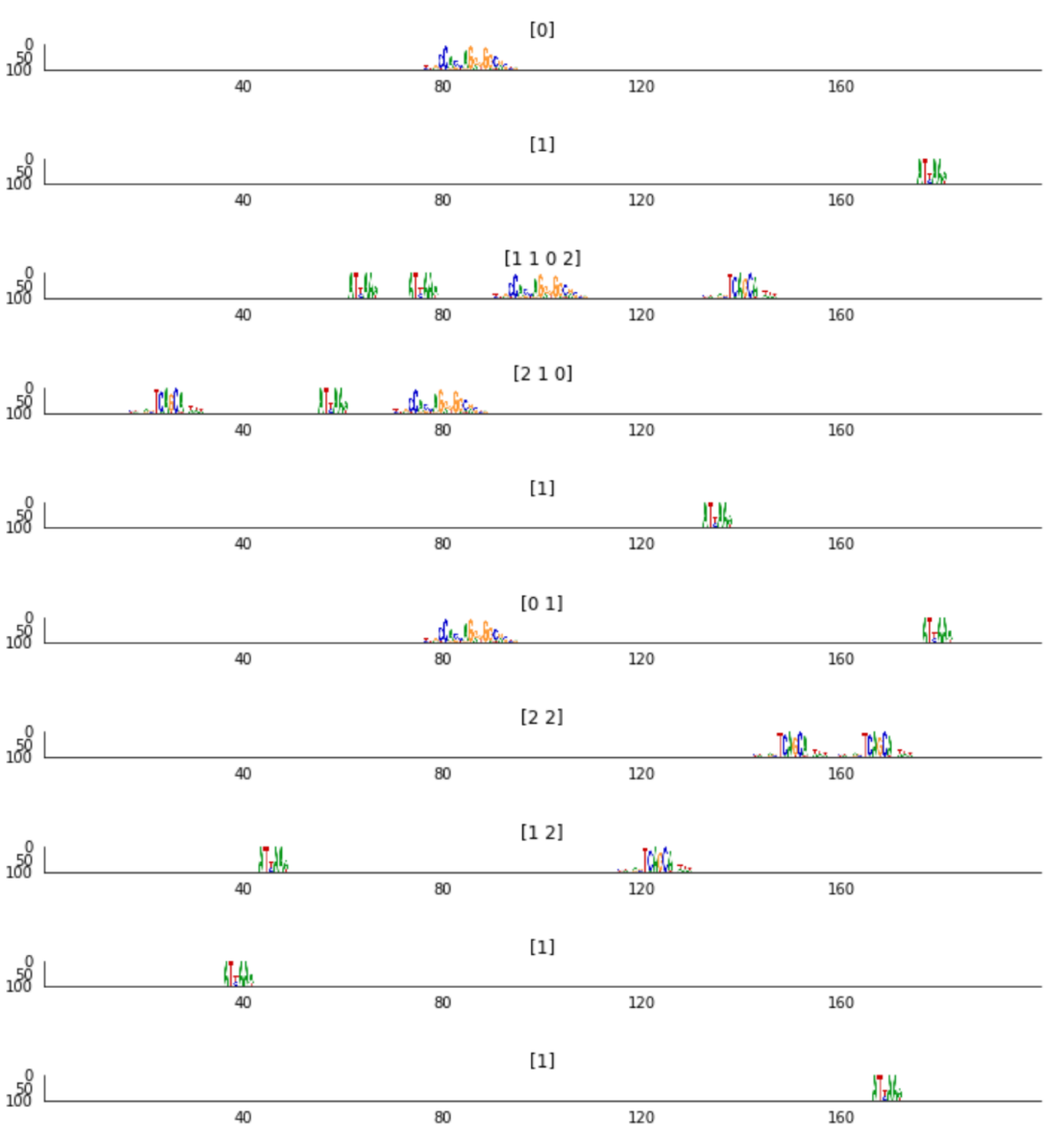






Sequence (X)	CTCF	Arid3a	MAFK
ACTAATGAGAGTAGTTTTATACTGGCCACCAGGTGGCGTATCGCTGCTAAGAGCTTGATATAAAC ATCTCAACAGCCTTGATGGAATAAACAAATACTCCGCGTTTAGGAGTATGTTGCCTCATAGCTTT GTAGGATAGCAGGGGCCGATAAAATTAACCATTTCAAGGTGCCACTAACATAGAATGAACACGA		0	0
TGCAATGACGATACTACAAAATTCTATAGACGTATGCGGAAGTTGGATCAAACAAGCACCTCCGC ACGGTATAACATTAACATTTTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTATTTCGTATTA CTGTGTGTCTATGTGACTCTCACCTTACAGAACCAGAAAGTGCAATTGGATATCTTCGACGA	0		0
GCGAAAGATAATAGCCTCAGACTTCATTCACGCTTGTGGGTGGATTCCACCTATAGGACGTACTT ACGGCCTATTTAAGCAAAGCCAGAGAGAGGTTTTCAGCAATGTTAATTGGGTCACACATTCAAG CTATGTCGGCTTTAATGAGTCCACCAGAGGGCTTATTCCTTGGAGACATCAGTCTATGTGGCTTA		0	
ATCAAAATAGAAGCCAGACACTTTGACCAAAAATTACTTGGTCATTGCTAAAATAGCCCTACATA GGAAAATAAAAGCAGATTACTTTCAGATAGCAACAAGAACAGTGACTCCAGCATTCAAGTCAA ACAAATTACGCAGTATGGGGGGGGGTATTAAGCGTTATGTGGGAACTGCGAGATCATCTCATTT GCCAGCAA	0	0	0
⋮	⋮	⋮	⋮

Sequence (X)	CTCF	Arid3a	MAFK
ACTAATGAGAGTAGTTTTATACTGGCCACCAGGTGGCGTATCGCTGCTAAGAGCTTGATATAAAC ATCTCAACAGCCTTGATGGAATAAACAAATACTCCGCGTTTAGGAGTATGTTGCCTCATAGCTTT GTAGGATAGCAGGGGCCGATAAAATTAACCATTTCAAGGTGCCACTAACATAGAATGAACACGA	1	0	0
TGCAATGACGATACTACAAAATTCTATAGACGTATGCGGAAGTTGGATCAAACAAGCACCTCCGC ACGGTATAACATTAACATTTTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTATTTCGTATTA CTGTGTGTCTATGTGACTCTCACCTTACAGAACCAGAAAGTGCAATTGGATATCTTCGACGA	0	1	0
GCGAAAGATAATAGCCTCAGACTTCATTCACGCTTGTGGGTGGATTCCACCTATAGGACGTACTT ACGGCCTATTTAAGCAAAGCCAGAGAGAGGTTTTCAGCAATGTTAATTGGGTCACACATTCAAG CTATGTCGGCTTTAATGAGTCCACCAGAGGGCTTATTCCTTGGAGACATCAGTCTATGTGGCTTA	1	0	1
ATCAAAATAGAAGCCAGACACTTTGACCAAAAATTACTTGGTCATTGCTAAAATAGCCCTACATA GGAAAATAAAAGCAGATTACTTTCAGATAGCAACAAGAACAGTGACTCCAGCATTCAAGTCAA ACAAATTACGCAGTATGGGGGGGGGTATTAAGCGTTATGTGGGAACTGCGAGATCATCTCATTT GCCAGCAA	0	0	0
⋮	⋮	⋮	⋮


[0]
[1]
[1 1 0 2]
[2 1 0]
[1]
[0 1]
[2 2]
[1 2]
[1]
[1]
0
50
100
40
80
120
160

In [2]:
```
# load dataset
data_path = 'data/lab_5_dataset.h5'
trainmat = h5py.File(data_path, 'r')

print("loading training data")
X_train = np.array(trainmat['X_train']).astype(np.float32)
y_train = np.array(trainmat['Y_train']).astype(np.float32)

print("loading cross-validation data")
X_valid = np.array(trainmat['X_valid']).astype(np.float32)
y_valid = np.array(trainmat['Y_valid']).astype(np.int32)

print("loading test data")
X_test = np.array(trainmat['X_test']).astype(np.float32)
y_test = np.array(trainmat['Y_test']).astype(np.int32)
```

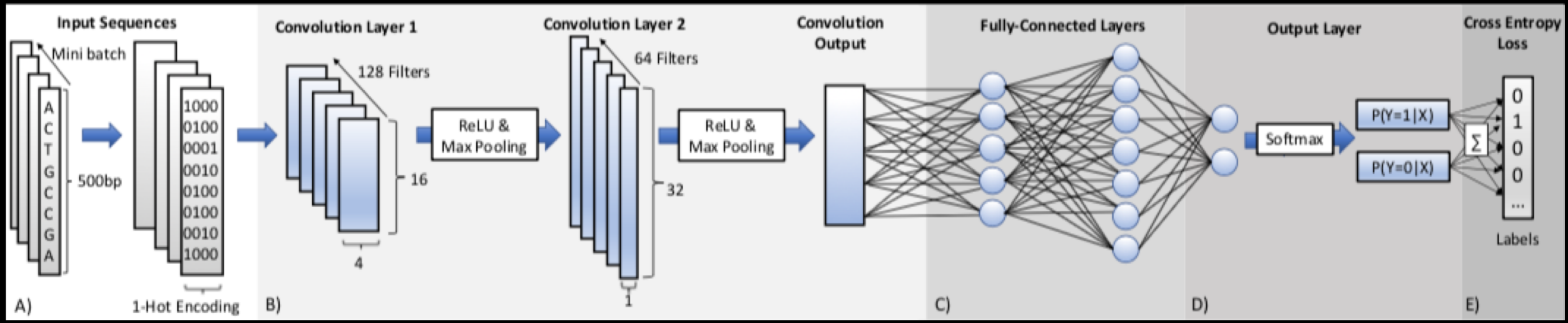
```
loading training data
loading cross-validation data
loading test data
```

We need to permute the data arrays for keras to accept them.

In [5]:
```
X_train = np.expand_dims(X_train, axis=3).transpose([0,2,3,1])
X_valid = np.expand_dims(X_valid, axis=3).transpose([0,2,3,1])
X_test = np.expand_dims(X_test, axis=3).transpose([0,2,3,1])

train = {'inputs': X_train, 'targets': y_train}
valid = {'inputs': X_valid, 'targets': y_valid}
test = {'inputs': X_test, 'targets': y_test}
```

In [16]:
```
# get data shapes
input_shape = list(train['inputs'].shape)
print(input_shape)
print(train['inputs'][0])
```

```
[17500, 200, 1, 4]
[[[0. 0. 0. 1.]]

 [[0. 0. 1. 0.]]

 [[1. 0. 0. 0.]]

 [[0. 0. 0. 1.]]

 [[0. 0. 0. 1.]]

 [[0. 0. 0. 1.]]

 [[1. 0. 0. 0.]]

 [[1. 0. 0. 0.]]

 [[0. 0. 1. 0.]]

 [[1. 0. 0. 0.]]
```



*Perhaps the most common architecture used in genomics (2 conv layers, 2 FC layers).

In [8]:
```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32,(19,1), activation='relu',input_shape = input_shape,use_bias=True))
model.add(layers.MaxPooling2D(1,4))
model.add(layers.Conv2D(16,(8,1), activation='relu', use_bias=True))
model.add(layers.MaxPooling2D(1,8))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(output_shape, activation='sigmoid'))
model.summary()
```

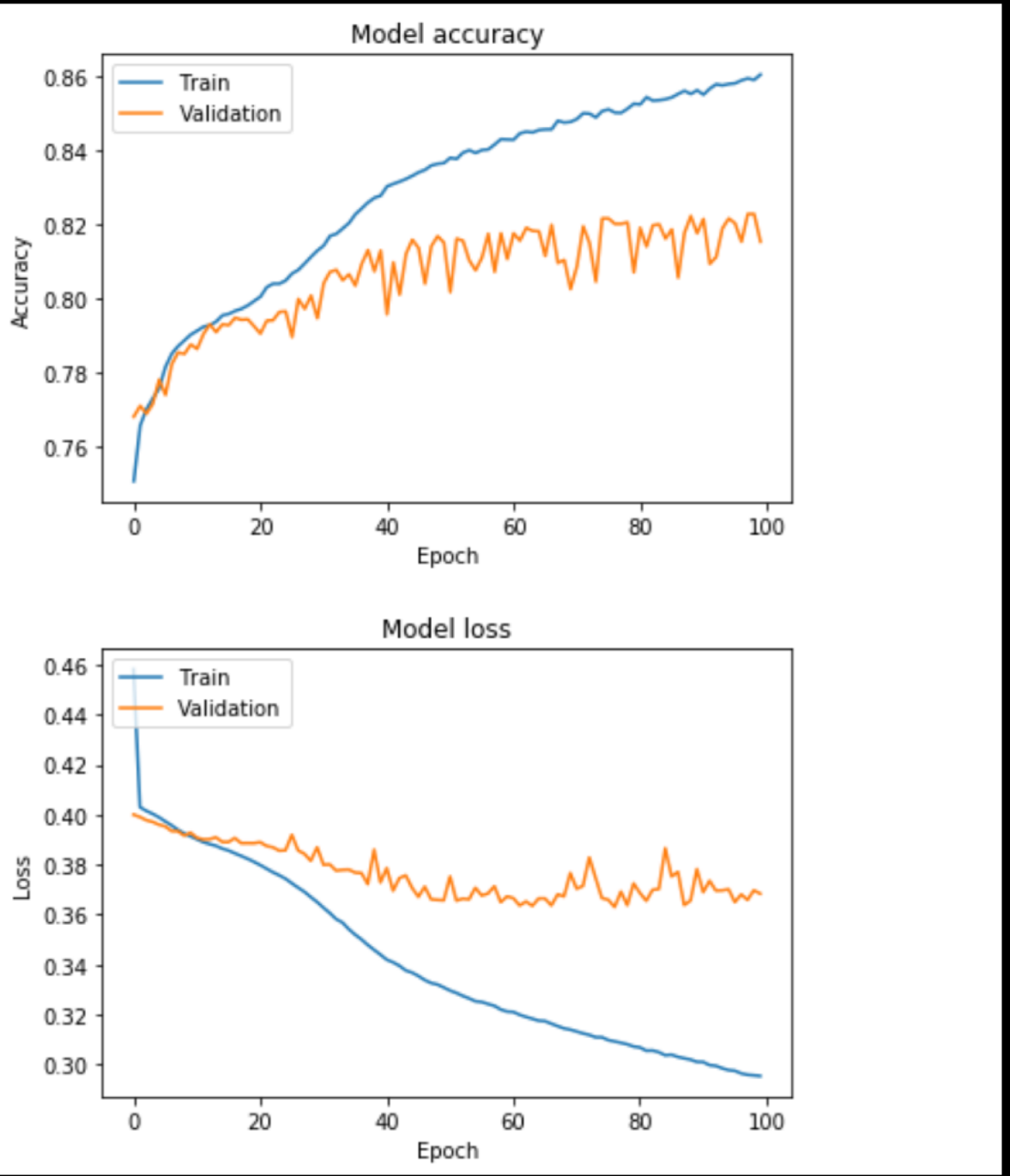
```
WARNING:tensorflow:From /usr/local/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263: col
ocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 182, 1, 32)        2464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 46, 1, 32)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 39, 1, 16)         4112
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 1, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 80)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               10368
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 387
=================================================================
Total params: 17,331
Trainable params: 17,331
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
from keras import optimizers
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Train your model for a suitable number of epochs.

In [9]:
```
history = model.fit(train['inputs'], train['targets'], epochs=100, batch_size=64,
          validation_data = (valid['inputs'],valid['targets']),
          shuffle=True)
```

```
17500/17500 [==============================] - 5s 309us/step - loss: 0.2985 - acc: 0.8574 - val_loss: 0.3697 - val_ac
c: 0.8188
Epoch 95/100
17500/17500 [==============================] - 5s 309us/step - loss: 0.2978 - acc: 0.8577 - val_loss: 0.3702 - val_ac
c: 0.8215
Epoch 96/100
17500/17500 [==============================] - 5s 293us/step - loss: 0.2975 - acc: 0.8579 - val_loss: 0.3650 - val_ac
c: 0.8201
Epoch 97/100
17500/17500 [==============================] - 5s 294us/step - loss: 0.2964 - acc: 0.8587 - val_loss: 0.3681 - val_ac
c: 0.8152
Epoch 98/100
17500/17500 [==============================] - 5s 285us/step - loss: 0.2959 - acc: 0.8593 - val_loss: 0.3659 - val_ac
c: 0.8227
Epoch 99/100
17500/17500 [==============================] - 5s 287us/step - loss: 0.2957 - acc: 0.8588 - val_loss: 0.3697 - val_ac
c: 0.8227
Epoch 100/100
17500/17500 [==============================] - 5s 299us/step - loss: 0.2954 - acc: 0.8603 - val_loss: 0.3684 - val_ac
c: 0.8152
```


Model accuracy
Train
Validation
Accuracy
0.86
0.84
0.82
0.80
0.78
0.76
0
20
40
60
80
100
Epoch
Model loss
Train
Validation
Loss
0.46
0.44
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0
20
40
60
80
100
Epoch

Can our model achieve high accuracy and what does it learn?

Yes

Snapshot

- Problem formulation
  - *Where do we get the data?*
- Data setup
  - *What do the data look like?*
- Opening the black box
  - *What has the model learned?*

→ 1. Feed-forward feature maps (activation-based)

Input Sequences

Testing Set

A C T G C C G A

200bp

Convolutional Layer

1000
0100
0001
0010
0100
0100
0010
1000

1-Hot Encoding

12

4

.1
0
.2
.4
.9
.3
.2
0

Vector of activations

Rank Activations

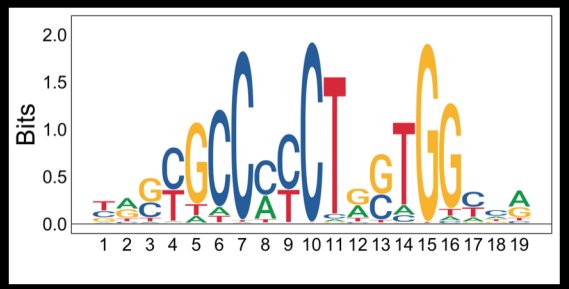
Testing Set

.9 .8 .9 1

Select top 10%

Extract Sequences

12



Stack Sequences

Compute Position Frequency Matrix

A
C
G
T

12

Compute Position Probability Matrix

A
C
G
T

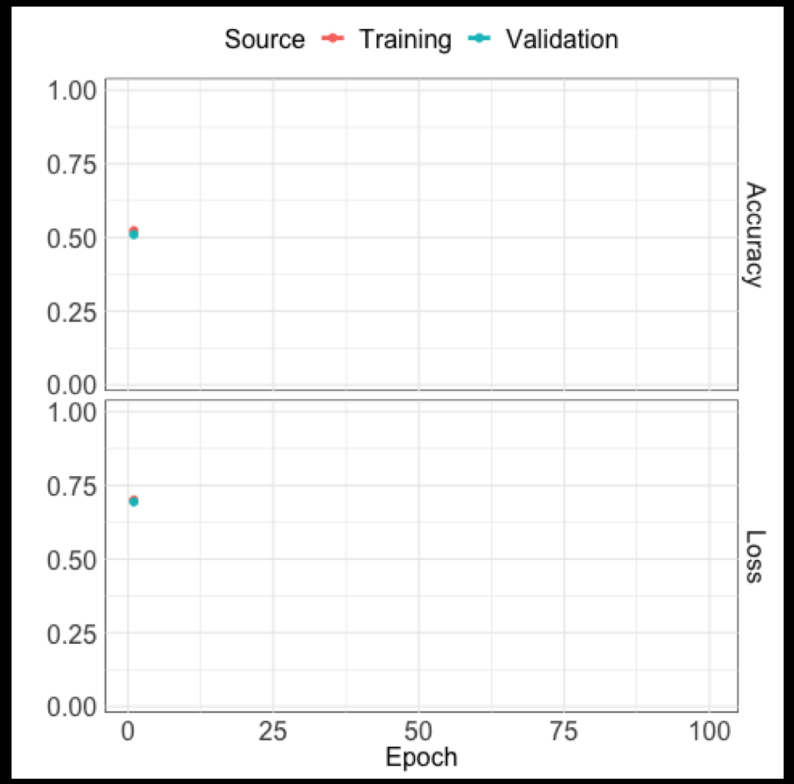
12

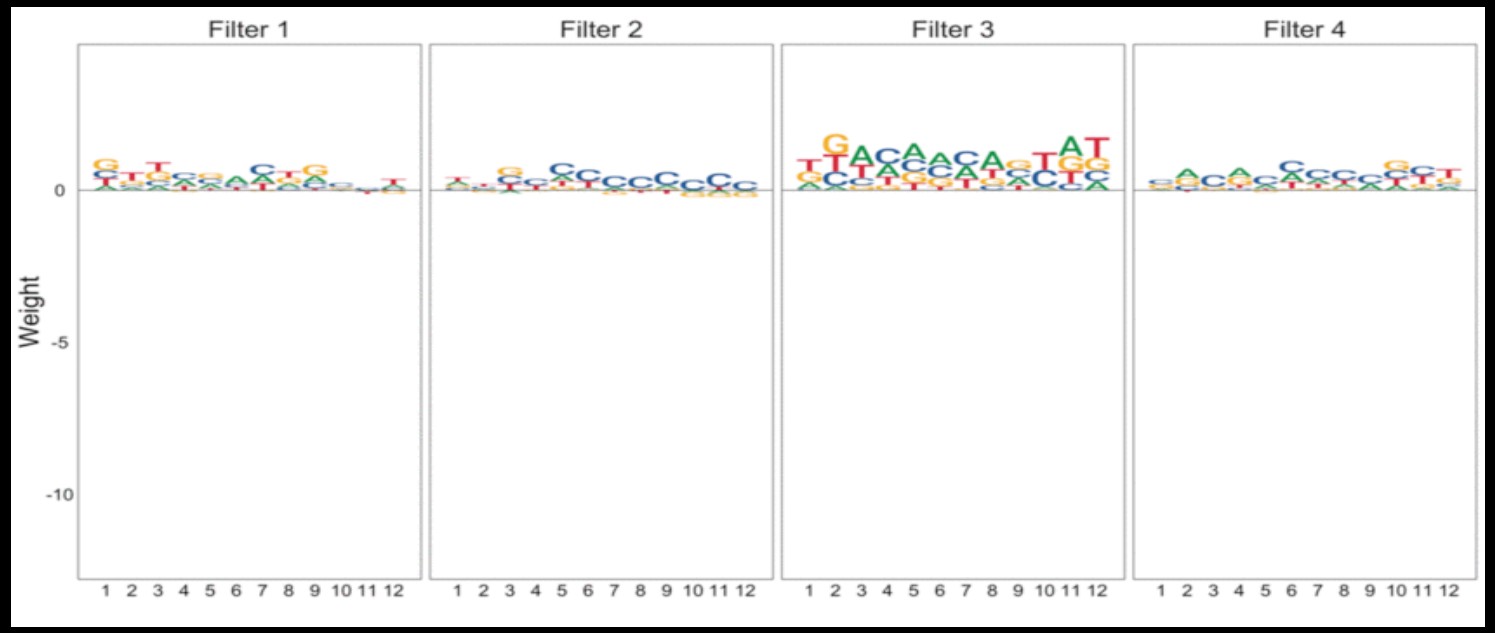
Compute Position Weight or Information Content

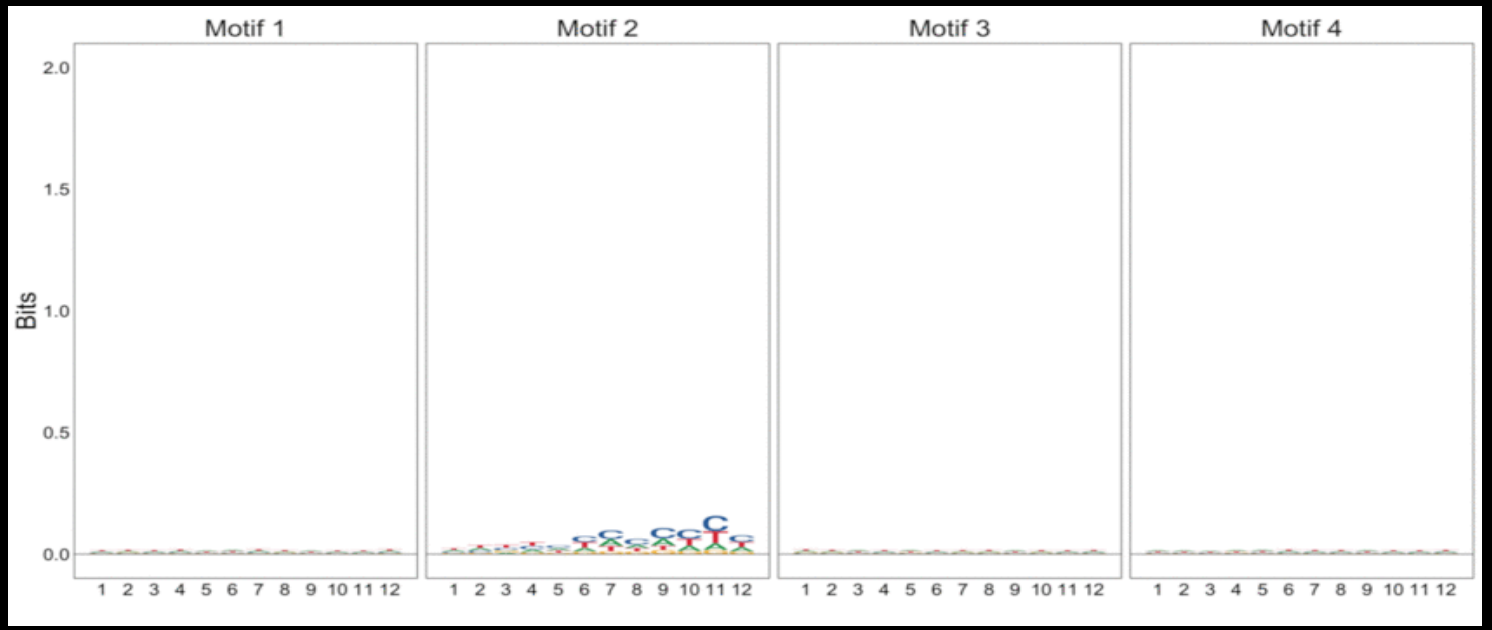
A
C
G
T

12

Schneider & Stevens, 1990


Source
Training
Validation
Accuracy
Loss
Epoch
1.00
0.75
0.50
0.25
0.00
0
25
50
75
100


Filter 1
Filter 2
Filter 3
Filter 4
Weight
0
-5
-10
1 2 3 4 5 6 7 8 9 10 11 12

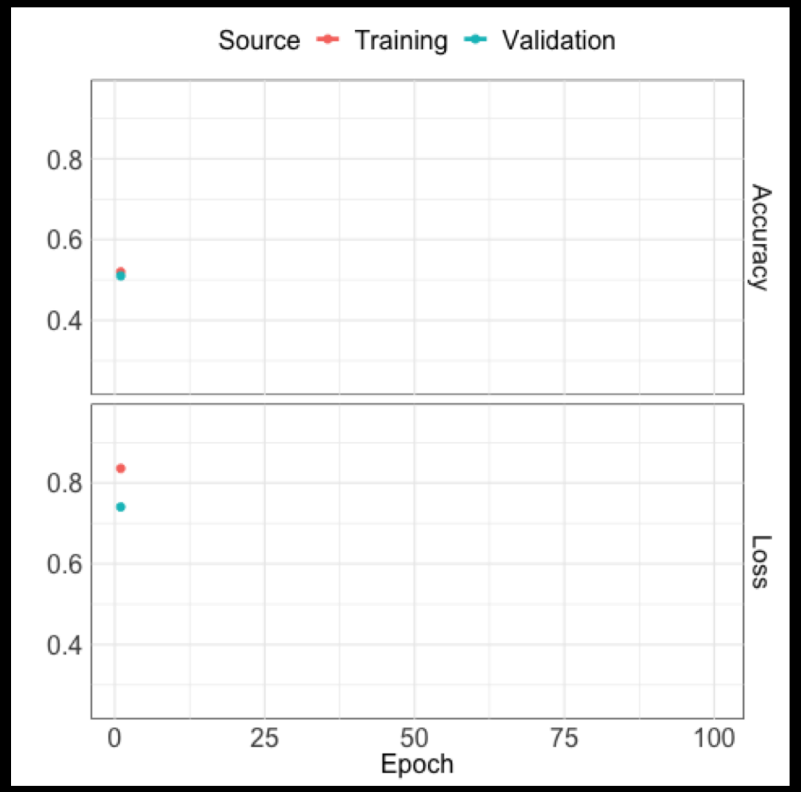

Motif 1
Motif 2
Motif 3
Motif 4
Bits
2.0
1.5
1.0
0.5
0.0
1 2 3 4 5 6 7 8 9 10 11 12

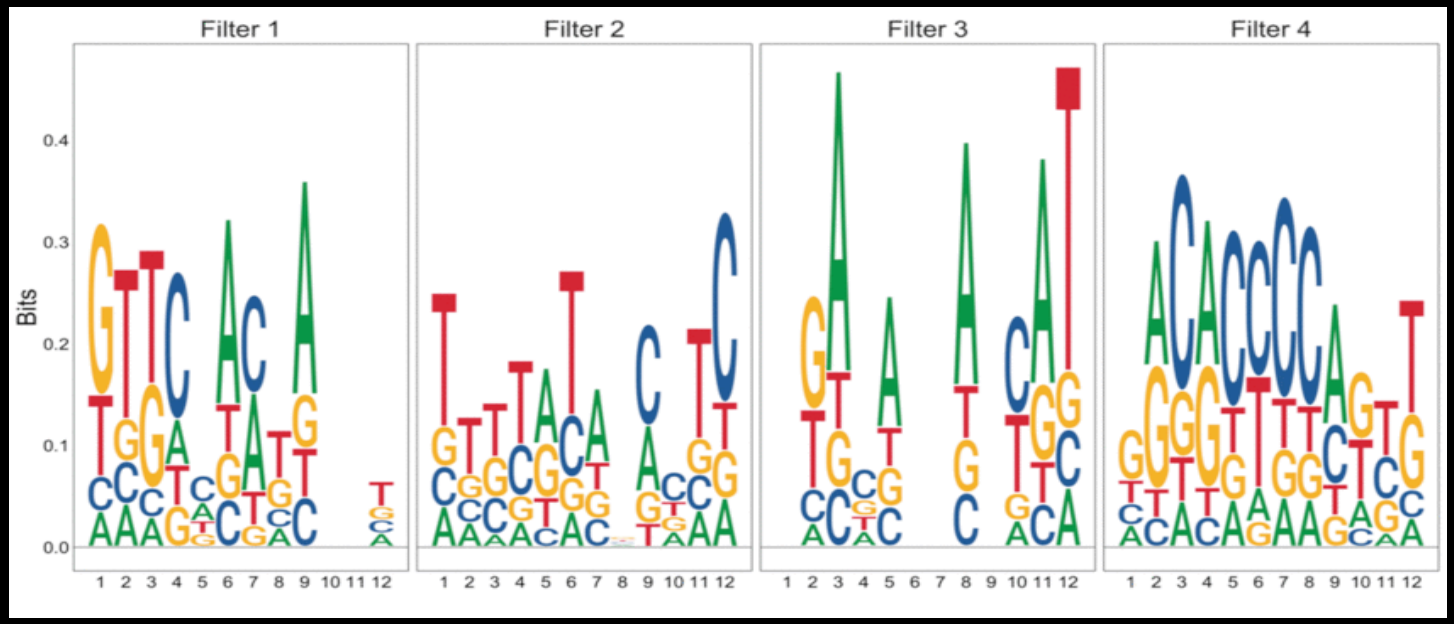
Snapshot

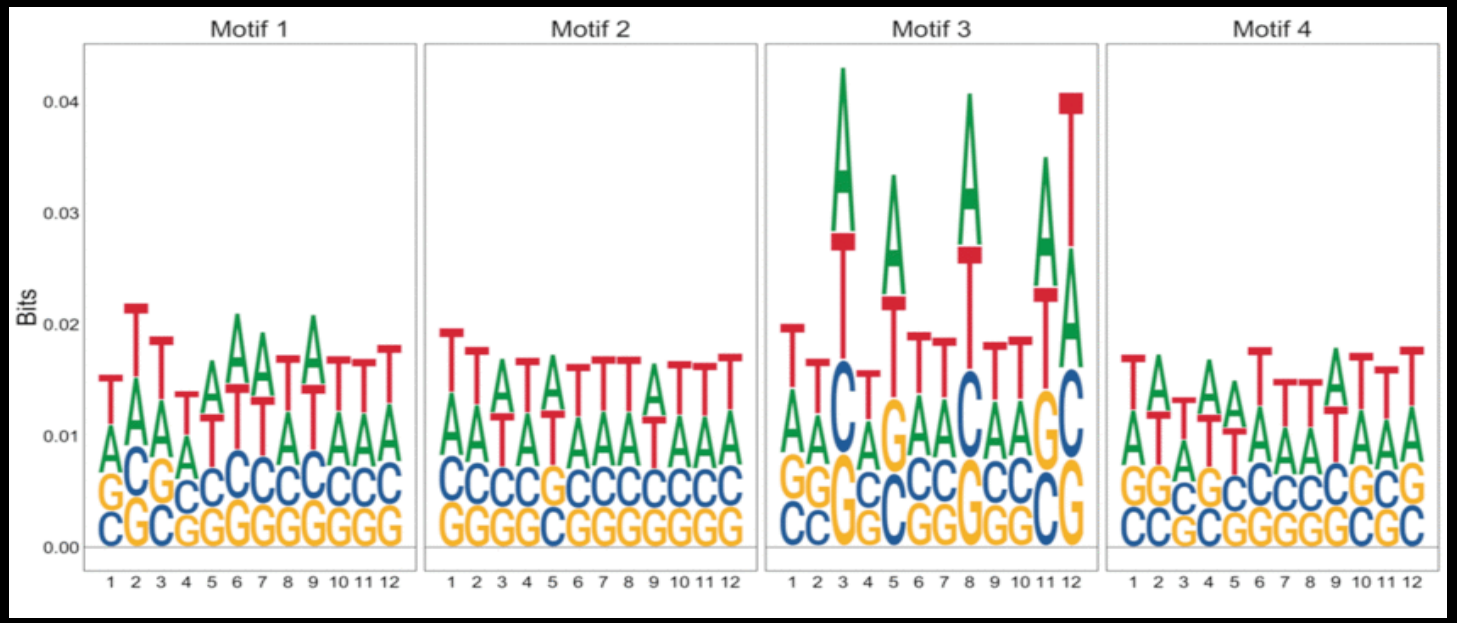
- Problem formulation
  - *Where do we get the data?*
- Data setup
  - *What do the data look like?*
- Opening the black box
  - *What has the model learned?*

→

1. Feed-forward feature maps (activation-based)
2. Model architecture (design-based)


Source
Training
Validation
Accuracy
Loss
0.8
0.6
0.4
0.8
0.6
0.4
0
25
50
75
100
Epoch


Filter 1
Filter 2
Filter 3
Filter 4
Bits
0.4
0.3
0.2
0.1
0.0
1 2 3 4 5 6 7 8 9 10 11 12


Motif 1
Motif 2
Motif 3
Motif 4
Bits
0.04
0.03
0.02
0.01
0.00
1 2 3 4 5 6 7 8 9 10 11 12

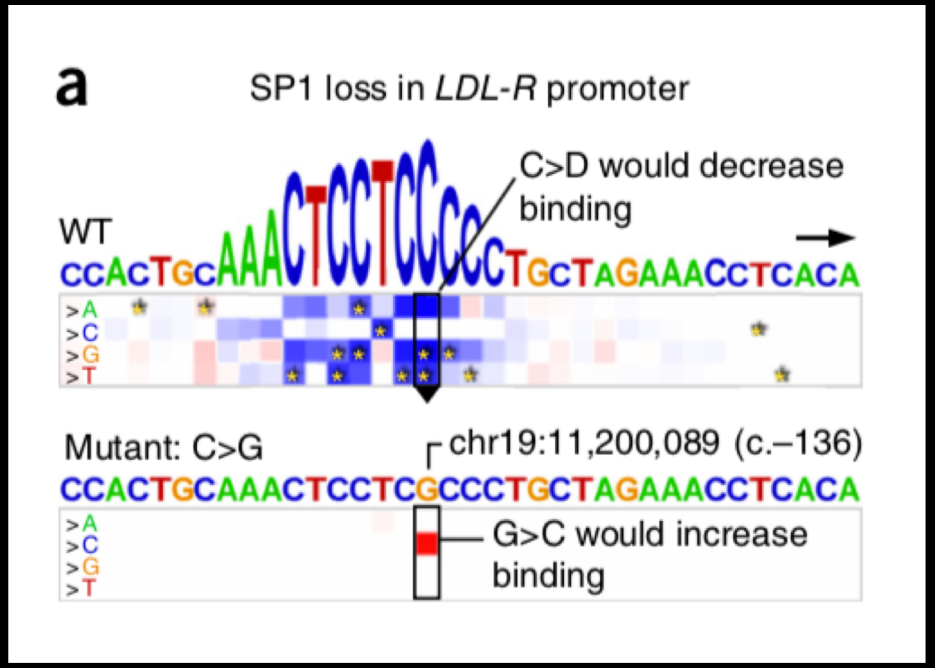
Snapshot

- Problem formulation
  - *Where do we get the data?*
- Data setup
  - *What do the data look like?*
- Opening the black box
  - *What has the model learned?*

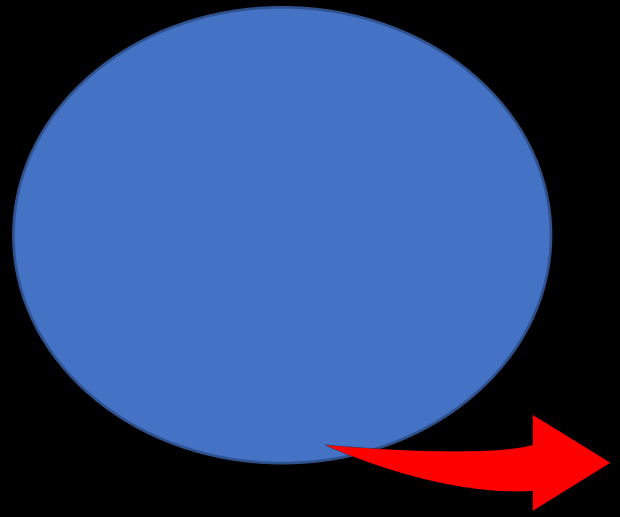
→

1. Feed-forward feature maps (activation-based)
2. Model architecture (design-based)
3. In-silico mutagenesis (perturbation-based)

1. For a given observation of interest, compute prediction from model or node of interest
   - Observation could be a promotor associated with cancer, for example
2. Modify a given value of the input
   - Swap A to C
3. Obtain a new prediction
4. Calculate and visualize the difference



Alipanahi, Babak, et al. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning." *Nature biotechnology* 33.8 (2015): 831.



TATAGACGTATGCGGAATTCCCGCCCCCAAGCACCTCCGCACGGTATAACATTAACATTTTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTA

Biological interpretation: Lack of blue circle results on unregulated gene expression of oncogene X

I.e. individuals with variant Y exhibit higher risk of cancer associated with oncogene X

Snapshot

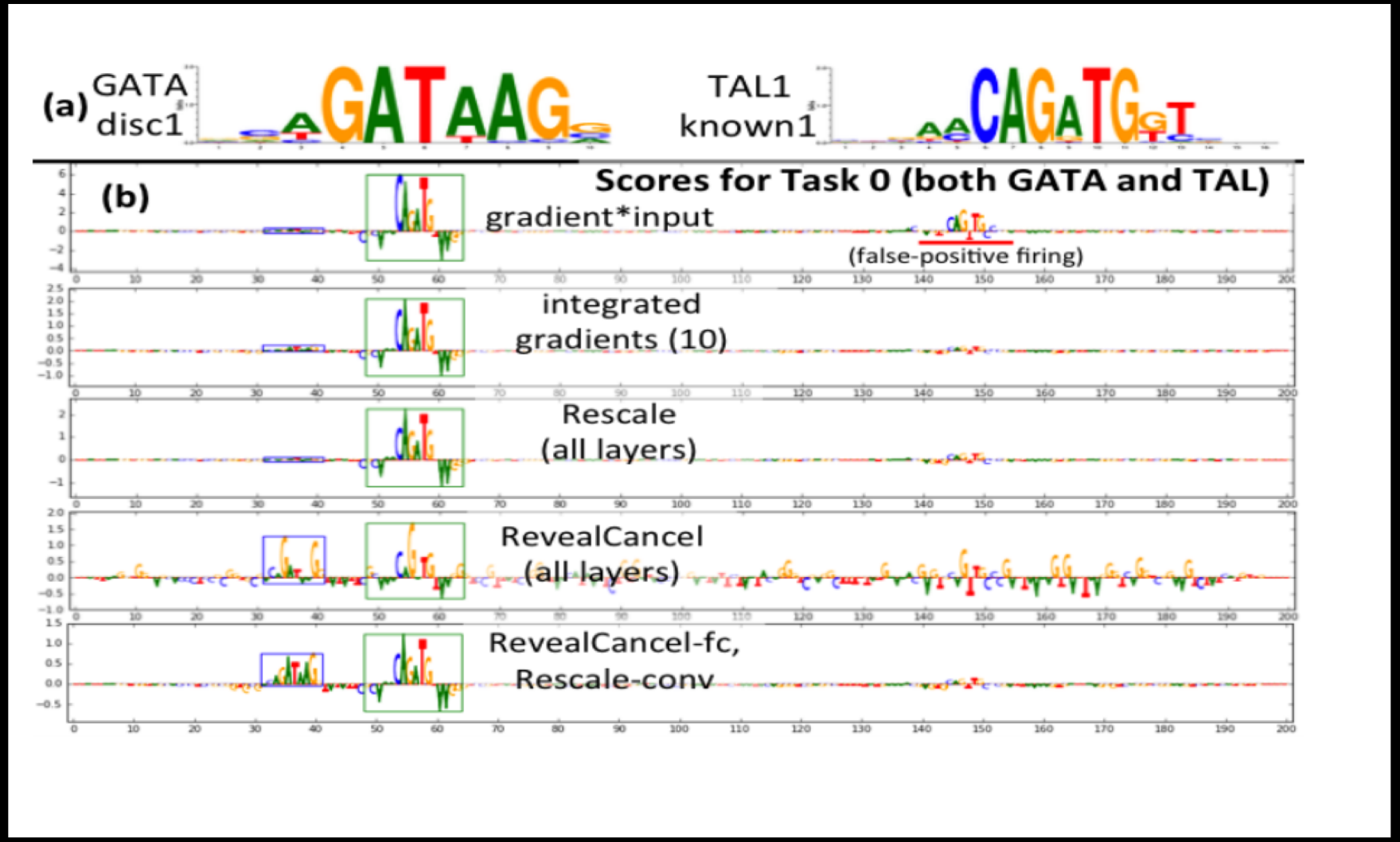
- Problem formulation
  - *Where do we get the data?*
- Data setup
  - *What do the data look like?*
- Opening the black box
  - *What has the model learned?*

→

1. Feed-forward feature maps (activation-based)
2. Model architecture (design-based)
3. In-silico mutagenesis (perturbation-based)
4. Backpropagation-based

1. Many algorithms exist
   - Saliency maps/gradient times input (Simonyan et al 2013)
   - Guided backpropagation (Springenberg et al 2014)
   - Layerwise relevance propagation (Bach et al 2015)
   - Integrated gradients (Sundararajan et al 2016)
   - Grad-CAM, Guided CAM (Selvaraju et al 2016)
   - DeepLIFT (Shrikumar et al 2017)
2. Connections exist between the methods
3. Pros and cons to different approaches but DeepLIFT recommended for genomics applications due to the discrete nature of nucleotide data

- DeepLIFT (Shrikumar et al 2017)
    1. Calculate activations for each layer for input observation and reference
    2. Compute difference between activations and input (X)
    3. Use rules for assigning importance scores to preceding neurons
        1. Ensures summation to delta property
            1. Contribution scores sum to the total difference from reference
    4. Requires a single backward pass to calculate importance scores
    5. Parallels to the chain rule with finite differences



Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

In [13]:
```
import deeplift
from deeplift.conversion import kerasapi_conversion as kc
```

Load the model saved in Question 1. Use the `kc.convert_model_from_saved_files` function to convert the model into the necessary format for deeplift and use default options provided in lab (`nonlinear_mxts_mode=deeplift.layers.NonlinearMxtsMode.DeepLIFT_GenomicsDefault`).

In [14]:
```
deeplift_model =\
    kc.convert_model_from_saved_files(
        'my_model.h5',
        nonlinear_mxts_mode=deeplift.layers.NonlinearMxtsMode.DeepLIFT_GenomicsDefault)
```

In [13]:
```
import deeplift
from deeplift.conversion import kerasapi_conversion as kc
```

Load the model saved in Question 1. Use the `kc.convert_model_from_saved_files` function to convert the model into the necessary format for deeplift and use default options provided in lab (`nonlinear_mxts_mode=deeplift.layers.NonlinearMxtsMode.DeepLIFT_GenomicsDefault`).

In [14]:
```
deeplift_model =\
    kc.convert_model_from_saved_files(
        'my_model.h5',
        nonlinear_mxts_mode=deeplift.layers.NonlinearMxtsMode.DeepLIFT_GenomicsDefault)
```

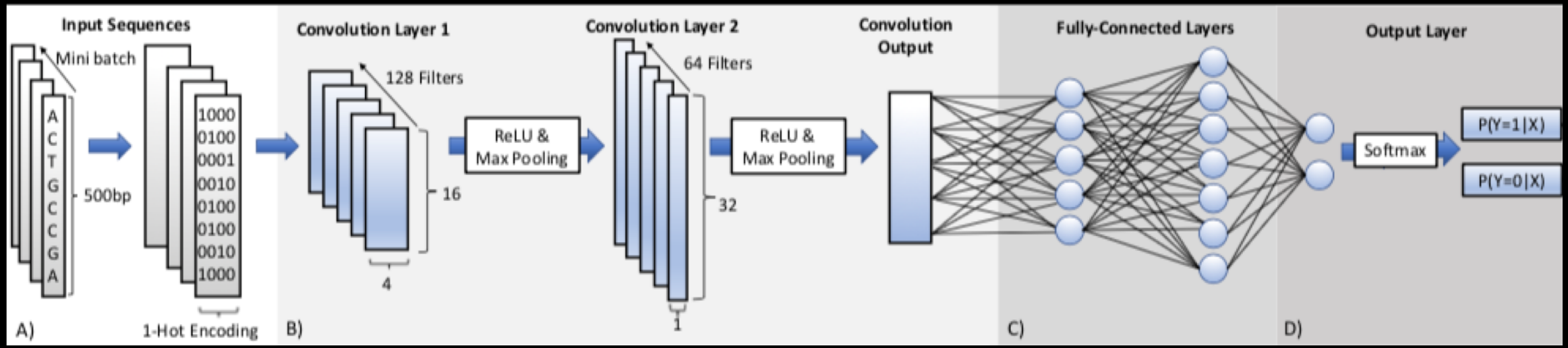
```
nonlinear_mxts_mode is set to: DeepLIFT_GenomicsDefault
For layer 0 the preceding linear layer is preact_0 of type Conv2D;
In accordance with nonlinear_mxts_mode=DeepLIFT_GenomicsDefault we are setting the NonlinearMxtsMode to Rescale
Heads-up: current implementation assumes maxpool layer is followed by a linear transformation (conv/dense layer)
For layer 2 the preceding linear layer is preact_2 of type Conv2D;
In accordance with nonlinear_mxts_mode=DeepLIFT_GenomicsDefault we are setting the NonlinearMxtsMode to Rescale
Heads-up: current implementation assumes maxpool layer is followed by a linear transformation (conv/dense layer)
For layer 5 the preceding linear layer is preact_5 of type Dense;
In accordance with nonlinear_mxts_modeDeepLIFT_GenomicsDefault we are setting the NonlinearMxtsMode to RevealCancel
Heads-up: I assume sigmoid is the output layer, not an intermediate one; if it's an intermediate layer then please bu
g me and I will implement the grad func
For layer 6 the preceding linear layer is preact_6 of type Dense;
In accordance with nonlinear_mxts_modeDeepLIFT_GenomicsDefault we are setting the NonlinearMxtsMode to RevealCancel
```

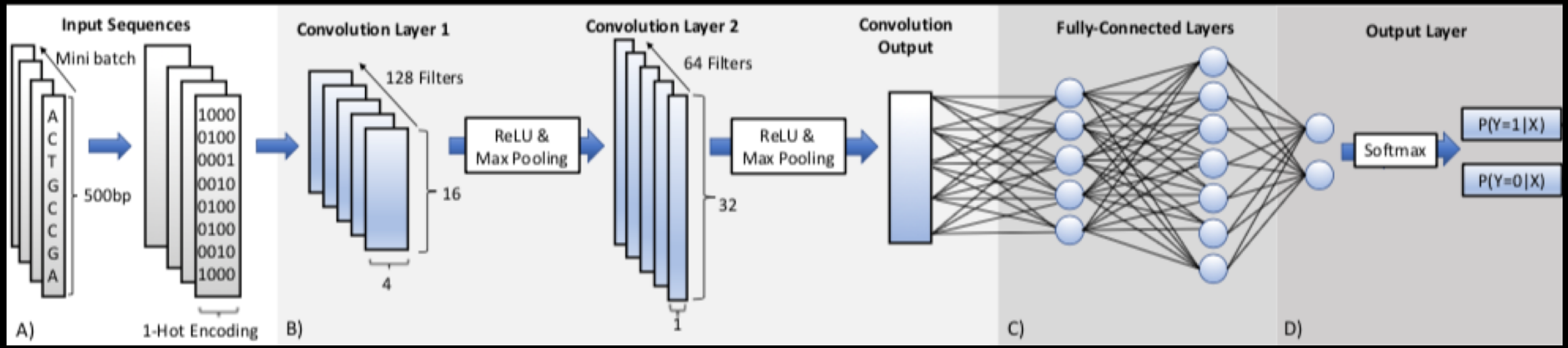
Print the layers of your model using the `.get_layers()` method.

In [15]:
```
deeplift_model.get_layers()
```

Out[15]:
```
[<deeplift.layers.core.Input at 0x148563278>,
 <deeplift.layers.convolutional.Conv2D at 0x133f43fd0>,
 <deeplift.layers.activations.ReLU at 0x133f43eb8>,
 <deeplift.layers.pooling.MaxPool2D at 0x133f4c278>,
 <deeplift.layers.convolutional.Conv2D at 0x133f4c240>,
 <deeplift.layers.activations.ReLU at 0x133f4c128>,
 <deeplift.layers.pooling.MaxPool2D at 0x133f4c390>,
 <deeplift.layers.core.Flatten at 0x133f4c3c8>,
 <deeplift.layers.core.Dense at 0x133f4c470>,
 <deeplift.layers.activations.ReLU at 0x133f4c438>,
 <deeplift.layers.core.Dense at 0x133f4c4e0>,
 <deeplift.layers.activations.Sigmoid at 0x14898d6d8>]
```

- DeepLIFT (Shrikumar et al 2017)
    1. Calculate activations for each layer for input observation and reference
    2. Compute difference between activations and input (X)
    3. Use rules for assigning importance scores to preceding neurons
        1. Ensures summation to delta property
            1. Contribution scores sum to the total difference from reference
    4. Requires a single backward pass to calculate importance scores
    5. Parallels to the chain rule with finite differences


Input Sequences
Mini batch
A
C
T
G
C
C
G
A
500bp
1000
0100
0001
0010
0100
0100
0010
1000
1-Hot Encoding
A)
Convolution Layer 1
128 Filters
16
4
B)
ReLU &
Max Pooling
Convolution Layer 2
64 Filters
32
1
ReLU &
Max Pooling
Convolution
Output
Fully-Connected Layers
C)
Output Layer
Softmax
P(Y=1|X)
P(Y=0|X)
D)



In [ ]:
```
find_scores_layer_idx = 0
output_scores_layer_idx = -2
```

In [ ]:
```
find_scores_layer_idx = 0
output_scores_layer_idx = -2
```

In [ ]:
```
find_scores_layer_idx = 0
output_scores_layer_idx = -2
```

Now use the `get_target_contribs_func` method to calculate the contribution function for your input and output layers. This uses the model architecture of your CNN to define a function which may be used to calculate the importance scores for given observations and tasks (output channels).

In [ ]:
```
deeplift_contribs_func = deeplift_model.get_target_contribs_func(
                             find_scores_layer_idx=find_scores_layer_idx,
                             target_layer_idx=output_scores_layer_idx)
```

- DeepLIFT (Shrikumar et al 2017)
    1. Calculate activations for each layer for input observation and reference
    2. Compute difference between activations and input (X)
    3. Use rules for assigning importance scores to preceding neurons
        1. Ensures summation to delta property
            1. Contribution scores sum to the total difference from reference
    4. Requires a single backward pass to calculate importance scores
    5. Parallels to the chain rule with finite differences

Define the output you would like to visualize importance scores for. Also, define a reference sequence based on genome background if desired.

In [20]:
```
output_class = 0
reference = np.array([.3,.2,.2,.3])[None,None,:]
```

Sequence (X)	CTCF	Arid3a	MAFK
ACTAATGAGAGTAGTTTTATACTGGCCACCAGGTGGCGTATCGCTGCTAAGAGCTTGATATAAAC ATCTCAACAGCCTTGATGGAATAAACAAATACTCCGCGTTTAGGAGTATGTTGCCTCATAGCTTT GTAGGATAGCAGGGGCCGATAAAATTAACCATTTCAAGGTGCCACTAACATAGAATGAACACGA	1	0	0
TGCAATGACGATACTACAAAATTCTATAGACGTATGCGGAAGTTGGATCAAACAAGCACCTCCGC ACGGTATAACATTAACATTTTTATCCCATCTCTTGTATACAGAGAACGTCTTAACTTATTTCGTATTA CTGTGTGTCTATGTGACTCTCACCTTACAGAACCAGAAAGTGCAATTGGATATCTTCGACGA	0	1	0
GCGAAAGATAATAGCCTCAGACTTCATTCACGCTTGTGGGTGGATTCCACCTATAGGACGTACTT ACGGCCTATTTAAGCAAAGCCAGAGAGAGGTTTTCAGCAATGTTAATTGGGTCACACATTCAAG CTATGTCGGCTTTAATGAGTCCACCAGAGGGCTTATTCCTTGGAGACATCAGTCTATGTGGCTTA	1	0	1
ATCAAAATAGAAGCCAGACACTTTGACCAAAAATTACTTGGTCATTGCTAAAATAGCCCTACATA GGAAAATAAAAGCAGATTACTTTCAGATAGCAACAAGAACAGTGACTCCAGCATTCAAGTCAA ACAAATTACGCAGTATGGGGGGGGGTATTAAGCGTTATGTGGGAACTGCGAGATCATCTCATTT GCCAGCAA	0	0	0
⋮	⋮	⋮	⋮

Calculate importance scores (predict on observations)

In [21]:
```
scores = np.array(deeplift_contribs_func(task_idx=output_class,
                                         input_data_list=[test['inputs']],
                                         batch_size=100,
                                         progress_update=1000,
                                        input_references_list=[reference]))
```

```
Done 0
Done 1000
Done 2000
Done 3000
Done 4000
```

Visualize the scores

In [21]:
```
#visualize scores
%matplotlib inline
from deeplift.visualization import viz_sequence
idxs = np.argwhere(test['targets'][:,output_class]==1)
idx = idxs[1]
```

In [22]:
```
scores_for_idx = scores[idx,:,0,:]
original_onehot = test['inputs'][idx,:,0,:]
scores_for_idx = original_onehot*scores_for_idx
viz_sequence.plot_weights(scores_for_idx, subticks_frequency=10)
```