CS273B: Deep learning for Genomics and Biomedicine

Lecture 3: Functional genomics, DenseNets, Backprop, Convnets & genomics applications

09/25/2017

Anshul Kundaje, James Zou

Outline

1

Functional Genomics

2

Dense Neural Networks

3

Convolutional architectures

- Multi-modal learning
- Multi-task learning

4

Training a DNN (SGD + Backprop)

Functional genomics

TGCCAAGCAGCAAAGTTTTGCTGCTGTTTATTTTTGTAGCTCTTACTATATTCT
ACTTTTACCATTGAAAATATTGAGGAAGTTATTTATATTTCTATTTTTTATATAT
TATATATTTTATGTATTTTAATATTACTATTACACATAATTATTTTTTATATATATGA
AGTACCAATGACTTCCTTTTCCAGAGCAATAATGAAATTTCACAGTATGAAA
ATGGAAGAAATCAATAAAATTATACGTGACCTGTGGCGAAGTACCTATCGTG
GACAAGGTGAGTACCATGGTGTATCACAAATGCTCTTTCCAAAGCCCTCTCC
GCAGCTCTTCCCCTTATGACCTCTCATCATGCCAGCATTACCTCCCTGGACCC
CTTTCTAAGCATGTCTTTGAGATTTTCTAAGAATTCTTATCTTGGCAACATCTT
GTAGCAAGAAAATGTAAAGTTTTCTGTTCCAGAGCCTAACAGGACTTACATA
TTTGACTGCAGTAGGCATTATATTTAGCTGATGACATAATAGGTTCTGTCATA
GTGTAGATAGGGATAAGCCAAAATGCAATAAGAAAAACCATCCAGAGGAA
ACTCTTTTTTTTTTCTTTTTCTTTTTTTTTTTTTCCAGATGGAGTCTCGCACTTC
TCTGTCACCCGGGCTGGAGCGCAGTGGTGCAATCTTGGCTCACTGCAACCT
CCACCTCCTGGGTTCAGGTGATTCTCCCACCTCAGCCTCCCGAGTAGTAGCT
GGAATTACAGGTGCGCGCTCCCACACCTGGCTAATTTTTTGTATTCTTAGTA
GAGATGGGGTTTCACCATGTTGGCCAGGCTGGTCTCAAACTCCTGCCCTCA
GGTGATCTGCCCACCTTGGCCTCCCAGTGTTGGGTTTACAGGCGTGAGCCA
CCGCGCCTGGCCTGGAGGAAACTCTTAACAGGGAAACTAAGAAAGAGTTG
AGGCTGAGGAACTGGGGCATCTGGGTTGCTTCTGGCCAGACCACCAGGCT
CTTGAATCCTCCCAGCCAGAGAAAGAGTTTCCACACCAGCCATTGTTTTCCT
CTGGTAATGTCAGCCTCATCTGTTGTTCCTAGGCTTACTTGATATGTTTGTAA
ATGACAAAAGGCTACAGAGCATAGGTTCCTCTAAAATATTCTTCTTCCTGTGT
CAGATATTGAATACATAGAAATACGGTCTGATGCCGATGAAAATGTATCAGCT
TCTGATAAAAGGCGGAATTATAACTACCGAGTGGTGATGCTGAAGGGAGAC
ACAGCCTTGGATATGCGAGGACGATGCAGTGCTGGACAAAAGGCAGGTAT
CTCAAAAGCCTGGGGAGCCAACTCACCCAAGTAACTGAAAGAGAGAAACA
AACATCAGTGCAGTGGAAGCACCCAAGGCTACACCTGAATGGTGGGAAGC
TCTTTGCTGCTATATAAAATGAATCAGGCTCAGCTACTATTATT ……

Functional genomics:
Decoding genome function

~ 3 billion nucleotides

Functional genomics: Decoding genome function

Function?

~ 3 billion nucleotides

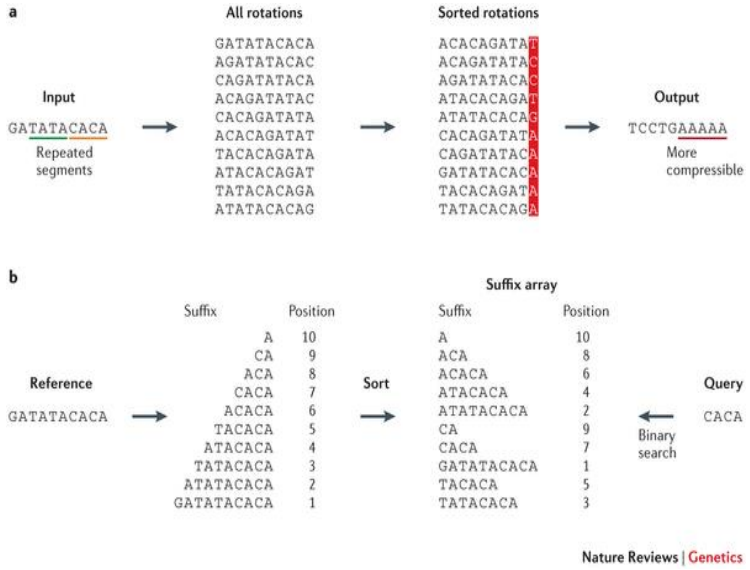
Mapping reads to reference genome

Naïve method

- Scan whole genome with every read
- Problem: Too slow

Indexing + Alignment approach

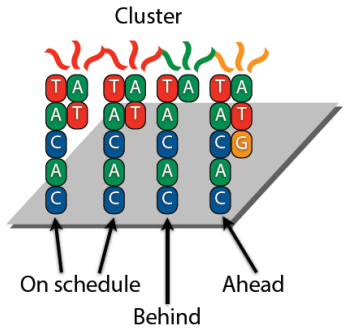
- Create a compressed reference 'genome index'
  - a map of where each short subsequence of length 'k' hits the genome
- Map reads using index via smart alignment algorithms and data structures (e.g suffix array)
- Allow for errors: insertions, deletions, mismatches in alignments



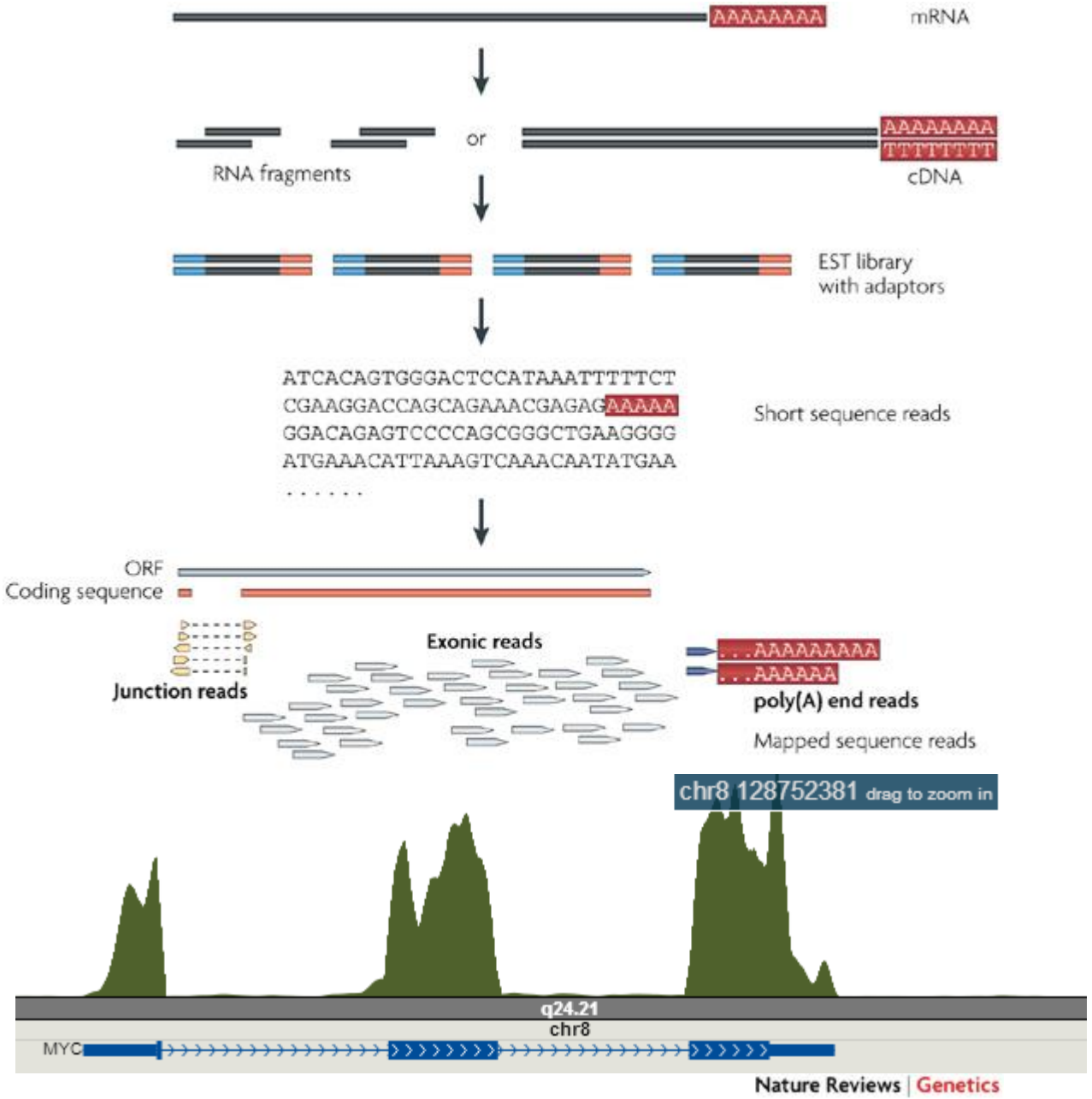
http://www.nature.com/nrg/journal/v14/n5/box/nrg3433_BX2.html

Run times for indexing alignment

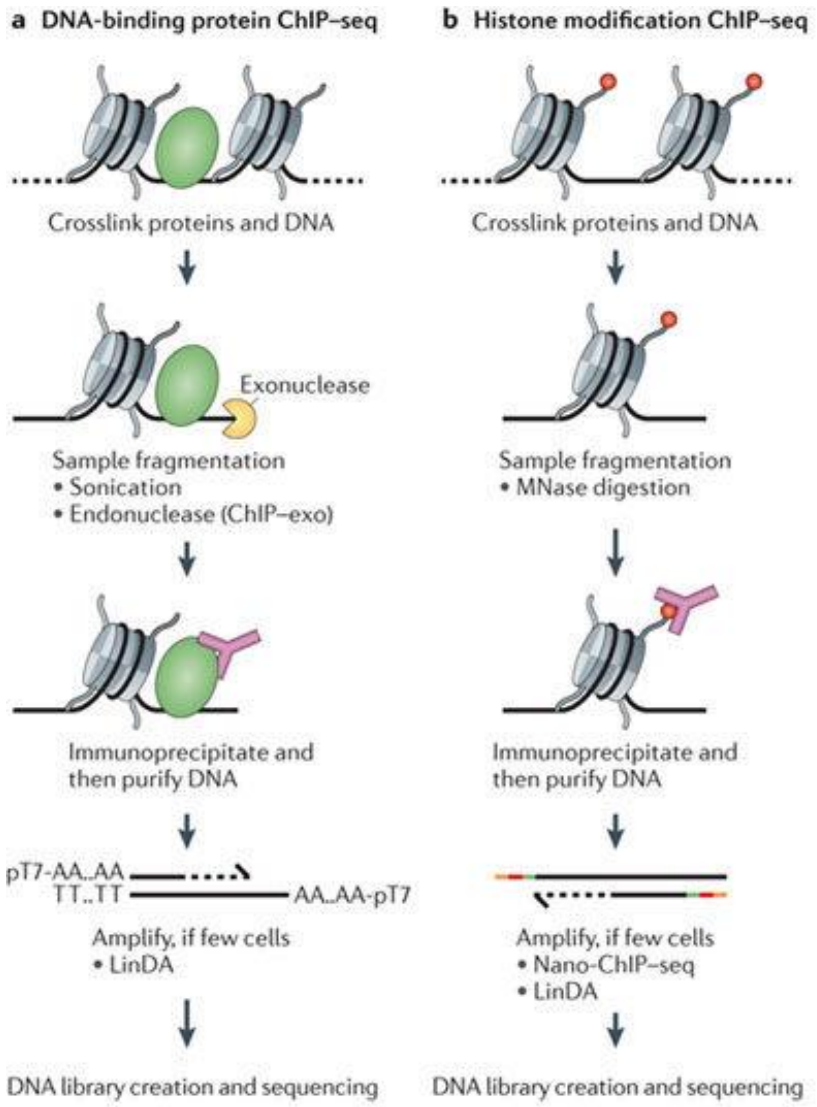
- Indexing human genome ~ 3 hours
- Alignment speed: 2 million 35 bp reads on 1 processor ~20 mins
- Alignment speed depends on error rate



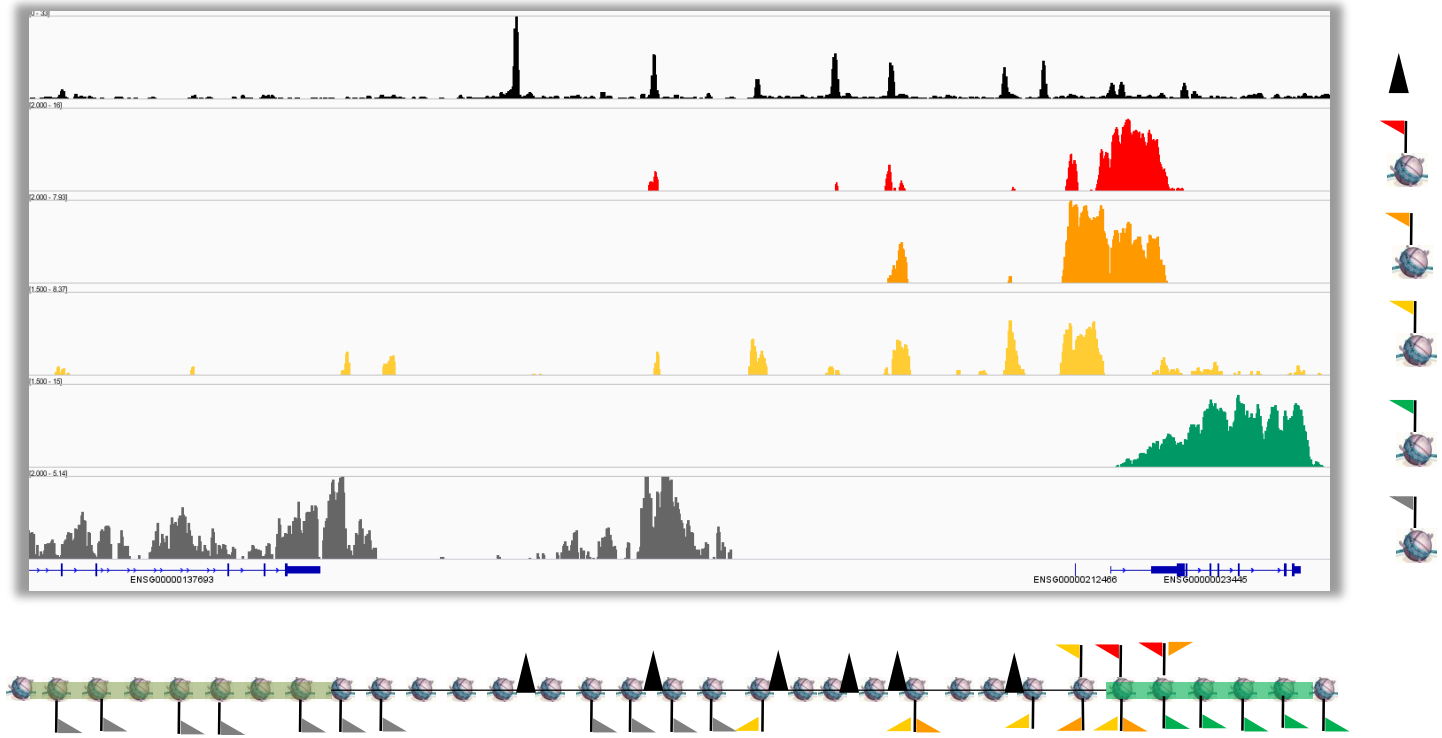
RNA sequencing (RNA-seq)



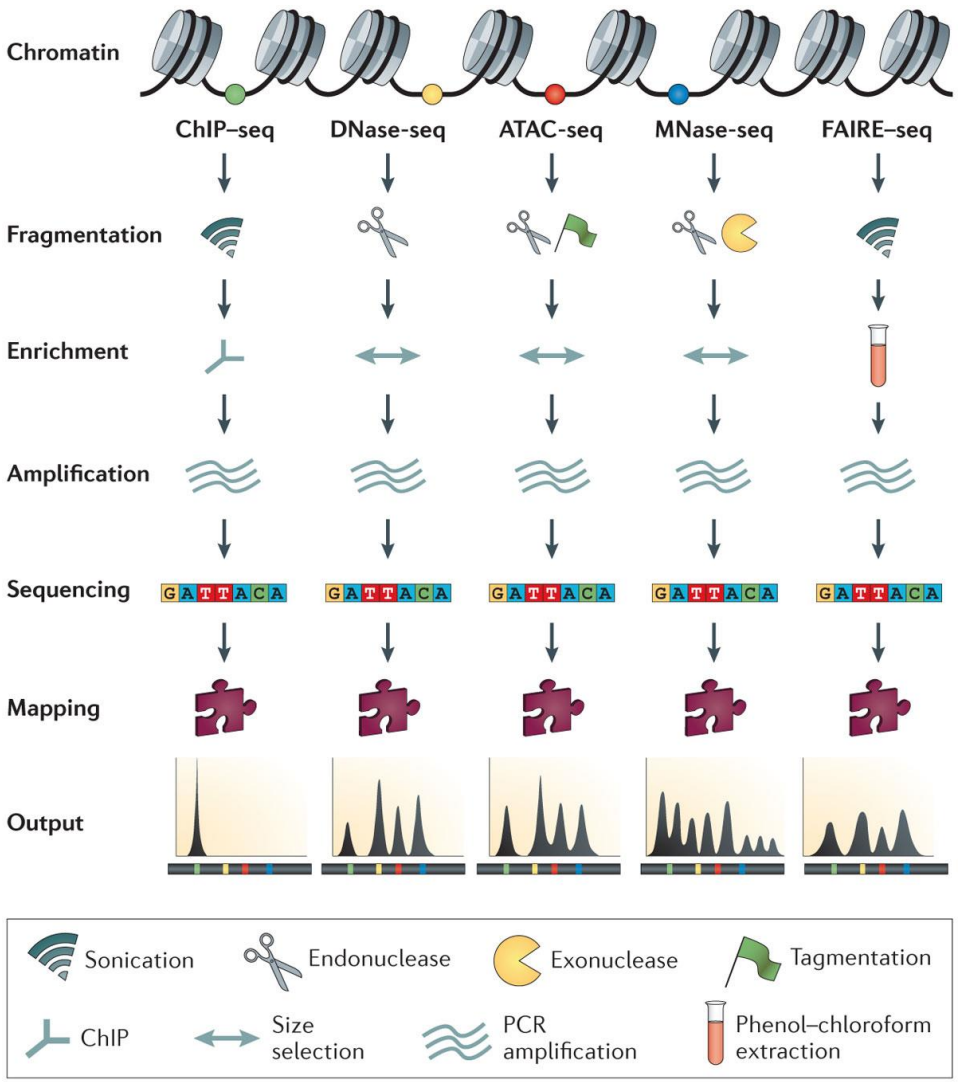
Chromatin immunoprecipitation (ChIP-seq)

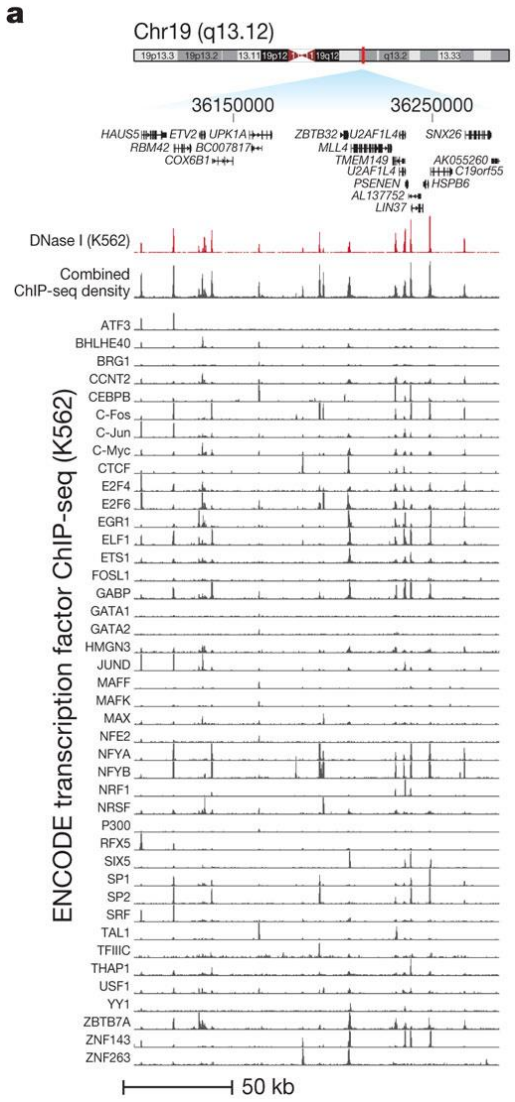


Protein-DNA binding maps
Maps of epigenomic modifications



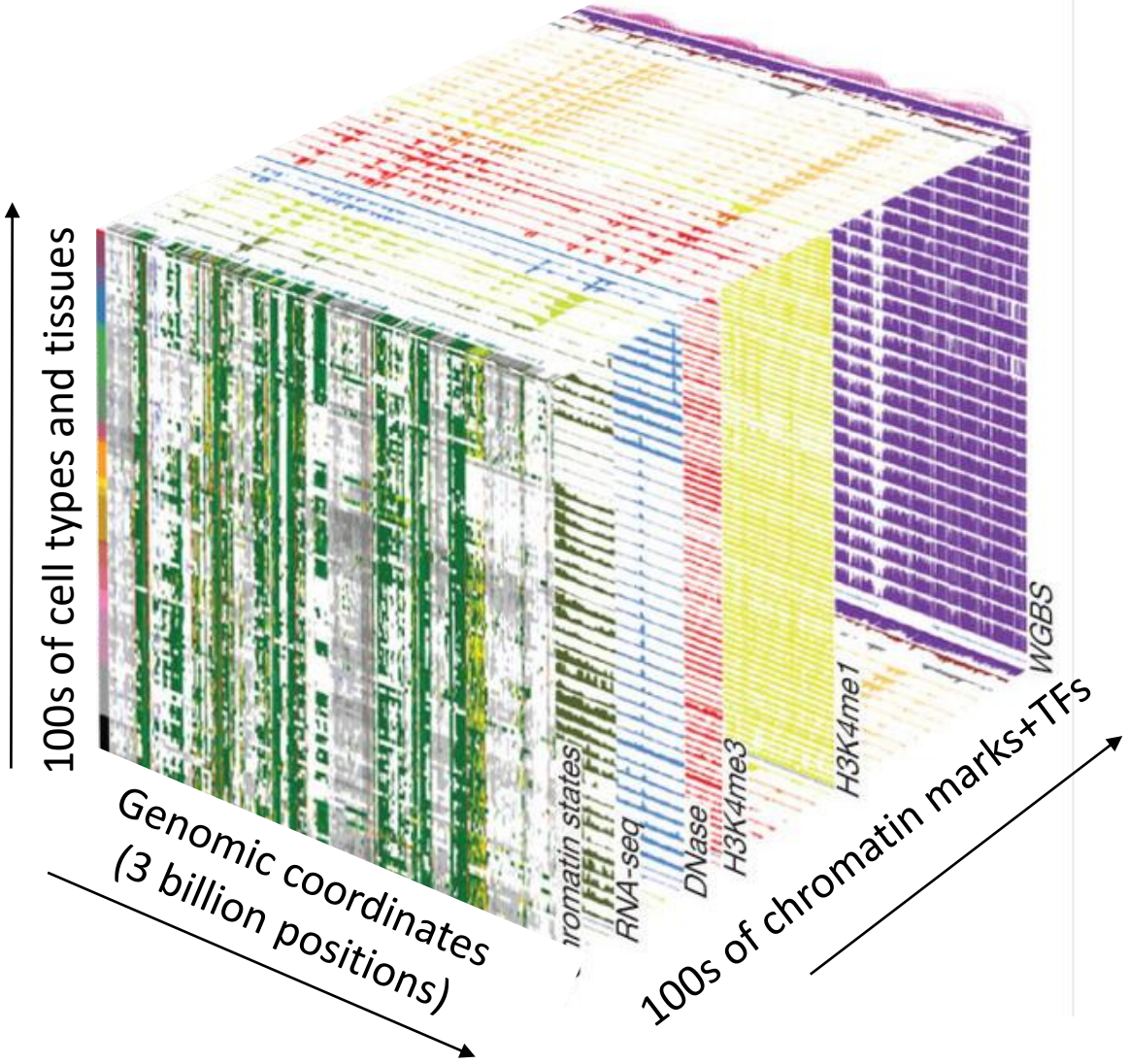
Chromatin accessibility (DNase-seq, ATAC-seq) and nucleosome sequencing (MNase-seq)

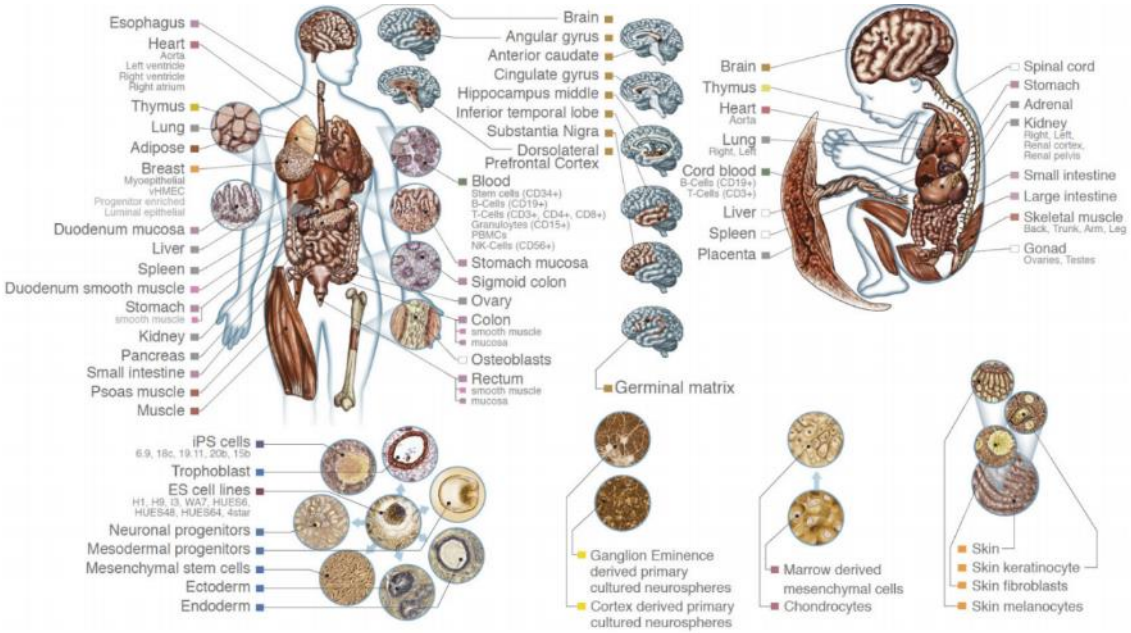




DNase-seq or ATAC-seq ~= sum(ChIP-seq for all TFs)

NIH funded collaborative consortia



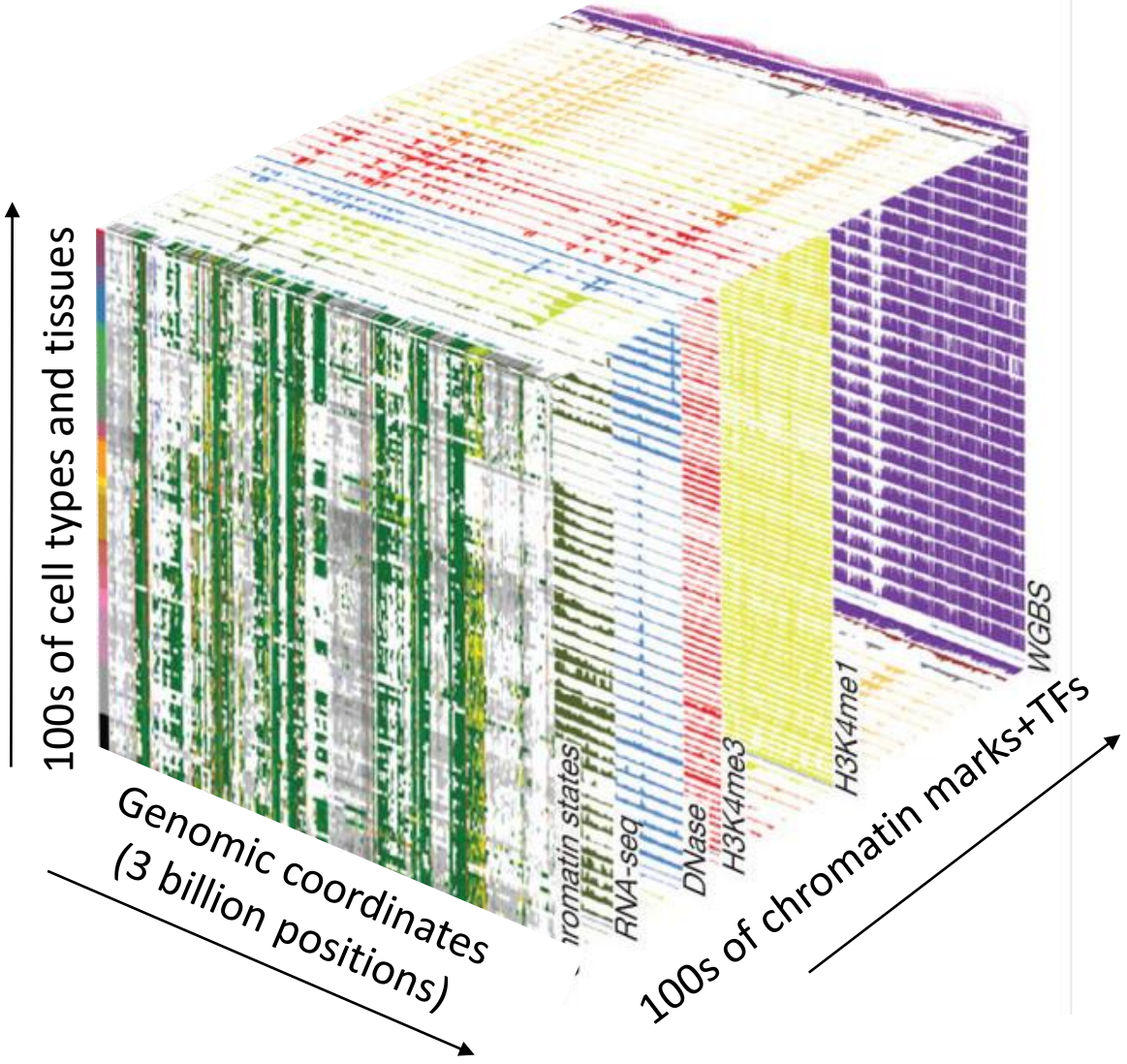


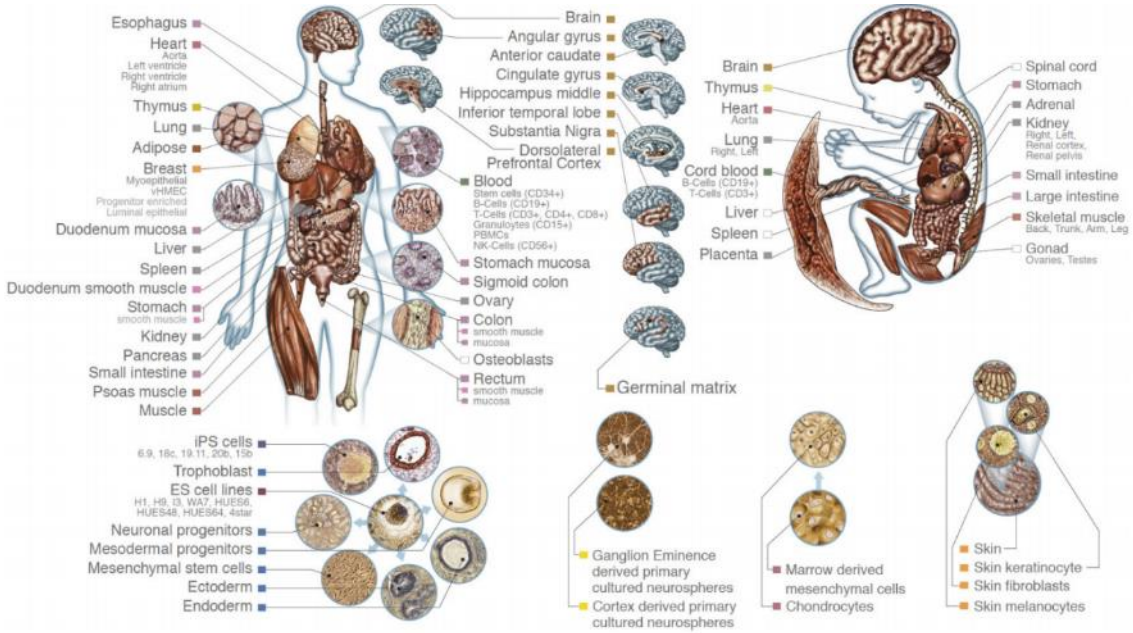
100s of Cell-Types/Tissues

NIH funded collaborative consortia

ENCODE

ROADMAP epigenomics PROJECT





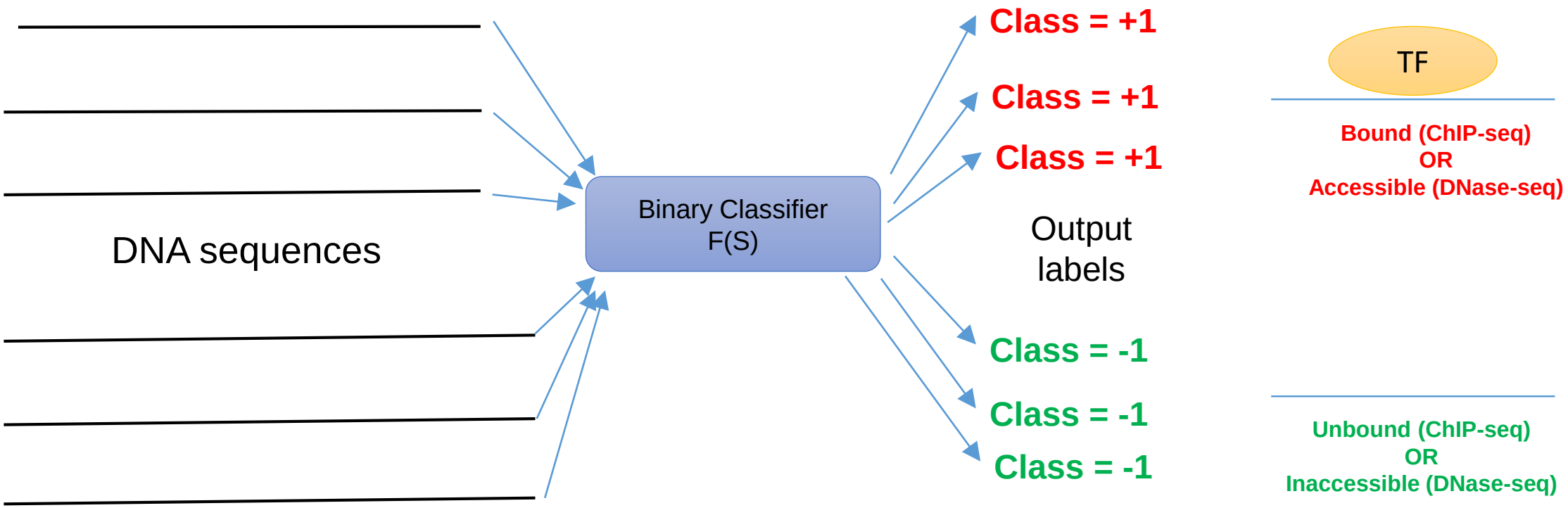
100s of Cell-Types/Tissues

Statistics, Machine Learning

- Denoise data
- Integrate and impute missing data
- Identify, interpret, understand properties of reg. elements & variants

Dense neural networks

Classifying regulatory DNA sequences



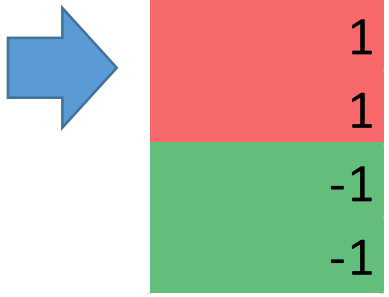
How to represent DNA sequence?

- Bag of words

K-mer features $(x_1, x_2, x_3 \ \ldots x_n)$

Input Examples $(S_1 \ \ldots S_m)$

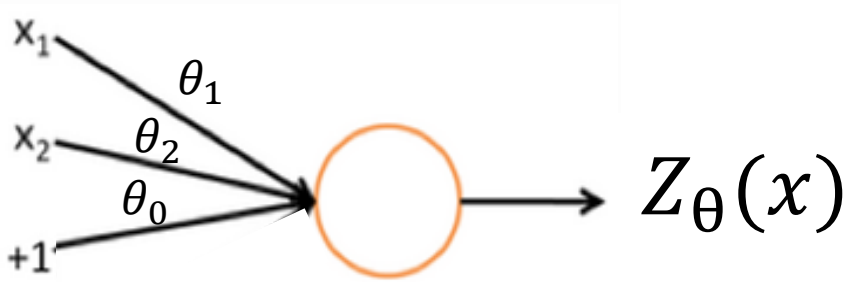
	AGA	GAT	ATC	TCG	CGA	GAG	AGT	GTG
Sequence1	1	4	5	2	0	0	0	0
Sequence2	4	1	1	0	0	2	8	8
Sequence3	6	3	3	2	2	4	10	10
Sequence4	5	5	6	2	0	2	8	8
Sequence5	10	4	4	2	2	6	18	18



Output labels $(Y_1 \ \ldots Y_m)$

- What are the advantages and disadvantages of bag of k-mers representation?

Logistic regression



$$Z_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

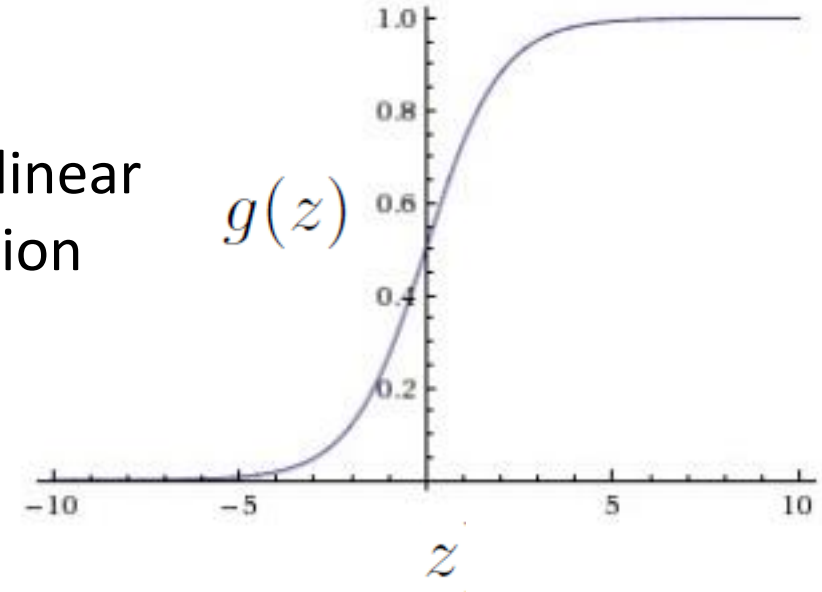
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

Logistic / Sigmoid

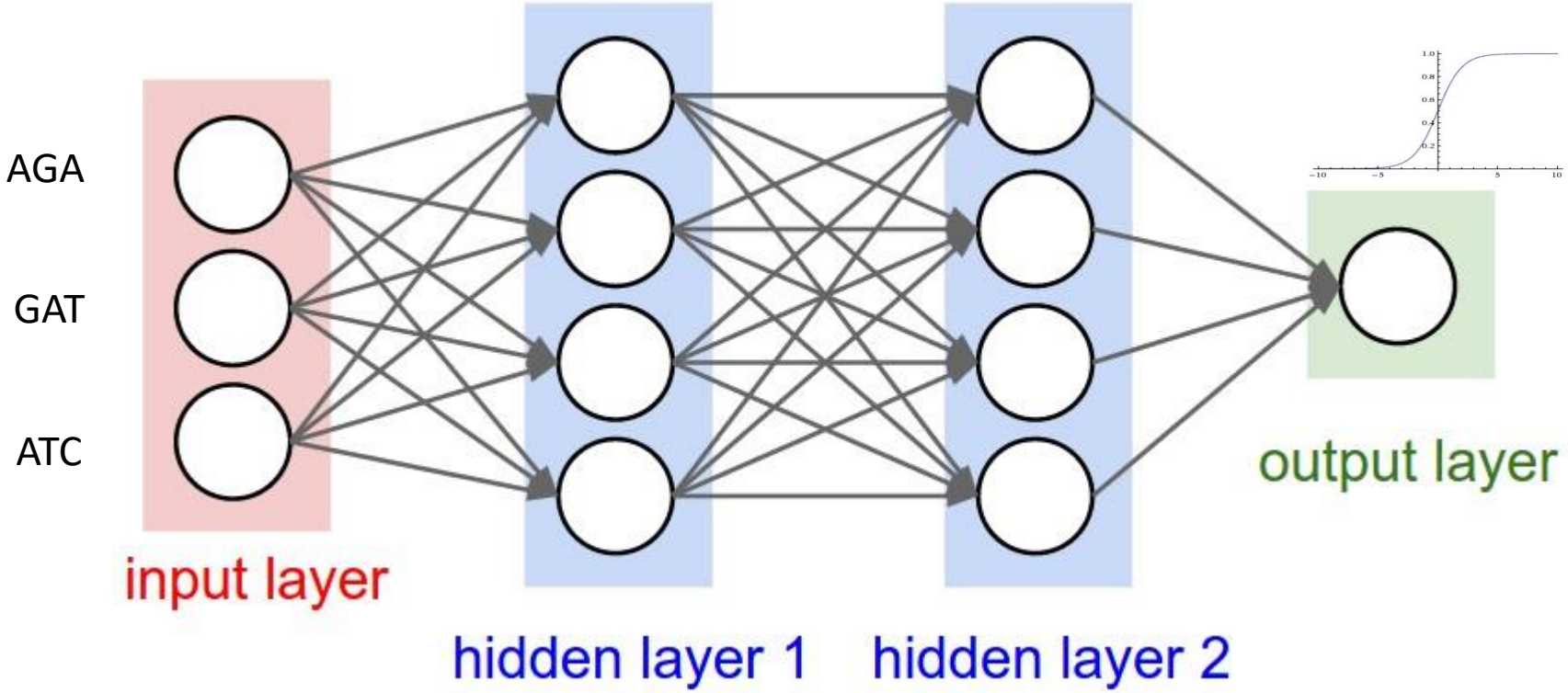
Useful for predicting probabilities

Non-linear function



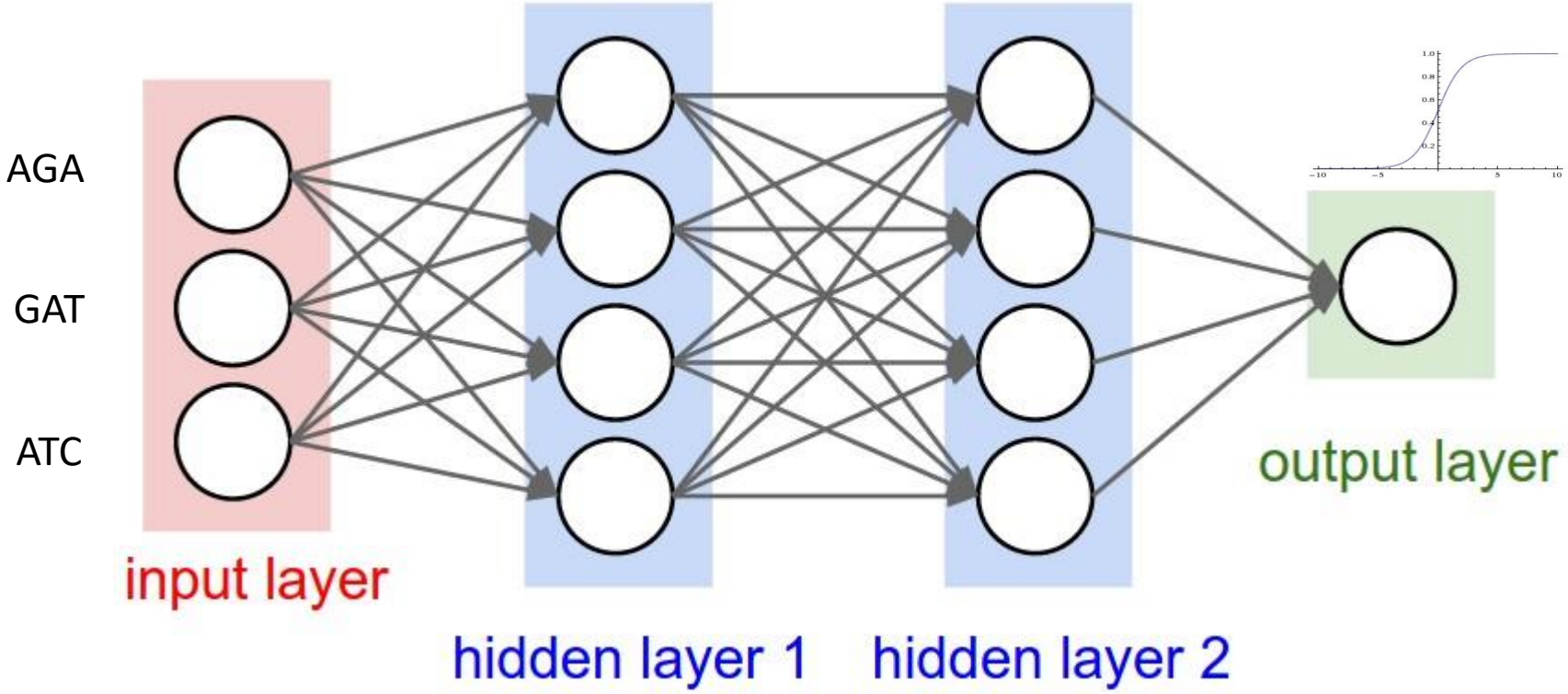
Training the model means learning the parameters to minimize some logistic loss

Dense (fully-connected) deep neural network



How many parameters does this DNN have?

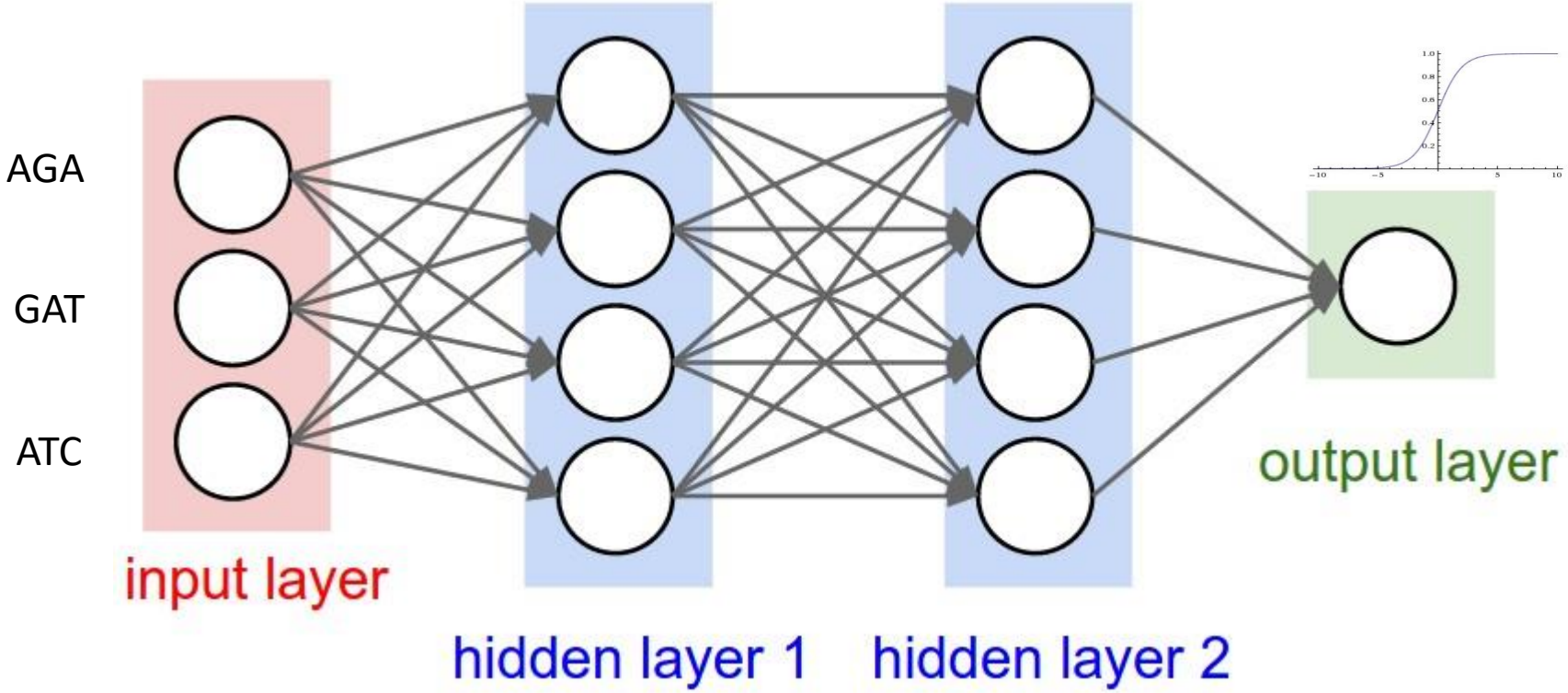
Dense (fully-connected) deep neural network



How many parameters does this DNN have?

3x4 + 4x4 + 4x1 = 12 + 16 + 4 = 32

Dense (fully-connected) deep neural network



How many parameters does this DNN have?

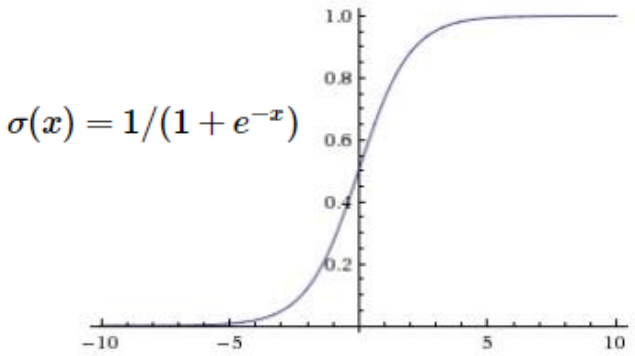
3x4 + 4x4 + 4x1 = 12 + 16 + 4 = 32

Architecture and hyperparameters of DNN

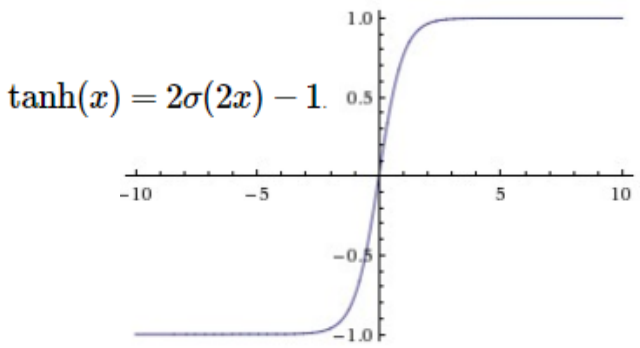
- How many layers?
- How many neurons per layer?
- What types of activations to use?

Search over architectures and identify optimal architecture by evaluating performance on validation set

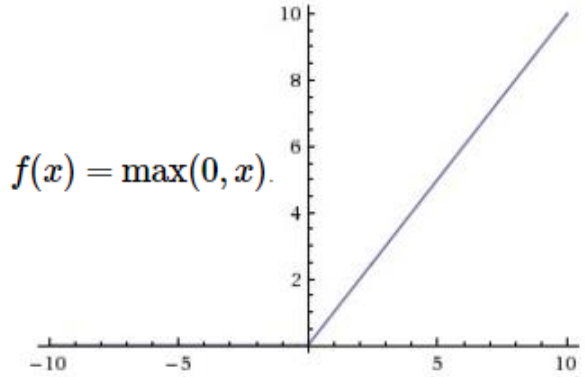
Types of activations



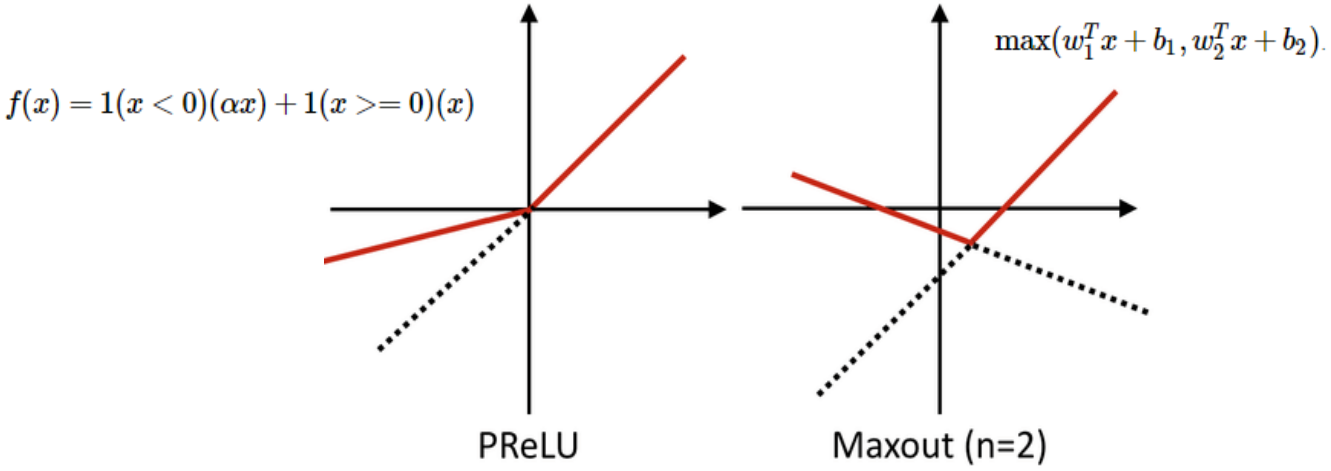
Sigmoid/Logistic



tanh



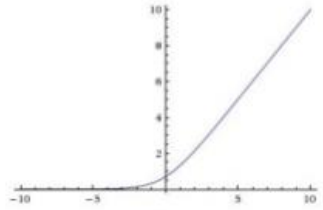
ReLU (rectified linear unit)



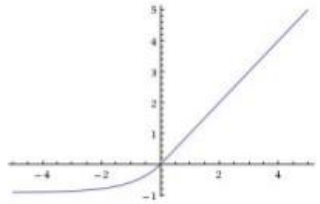
PReLU

Maxout (n=2)

Softplus and Exponential Linear Unit (ELU)



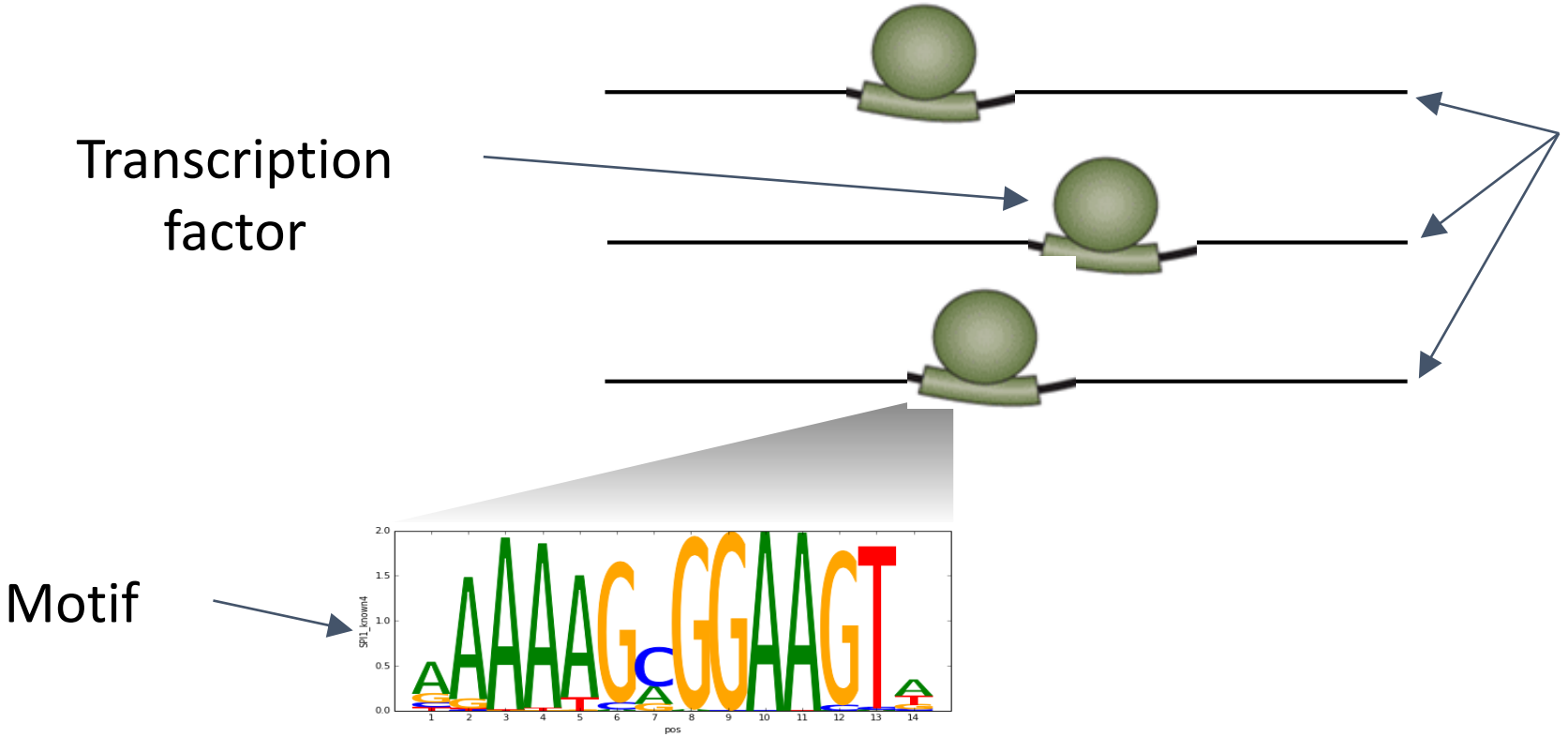
$softplus(x) = log(1 + e^x)$



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Convolutional architectures

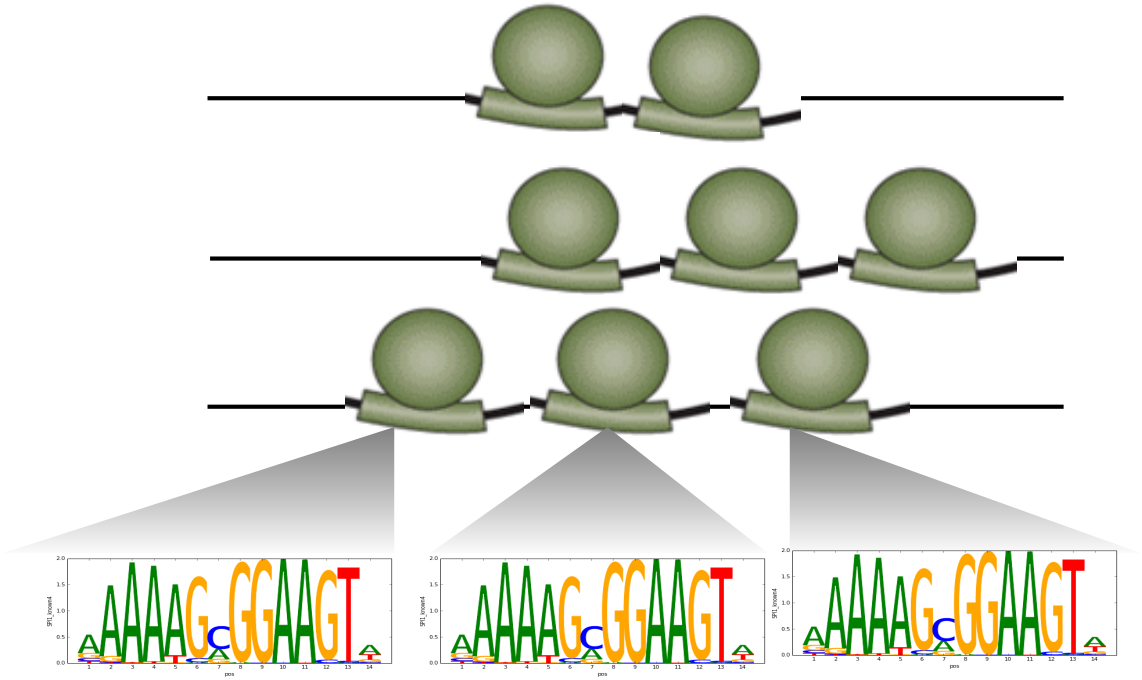
Key properties of regulatory sequence



TRANSCRIPTION FACTOR BINDING

Regulatory proteins called **transcription factors (TFs)** bind to high affinity sequence patterns (**motifs**) in regulatory DNA

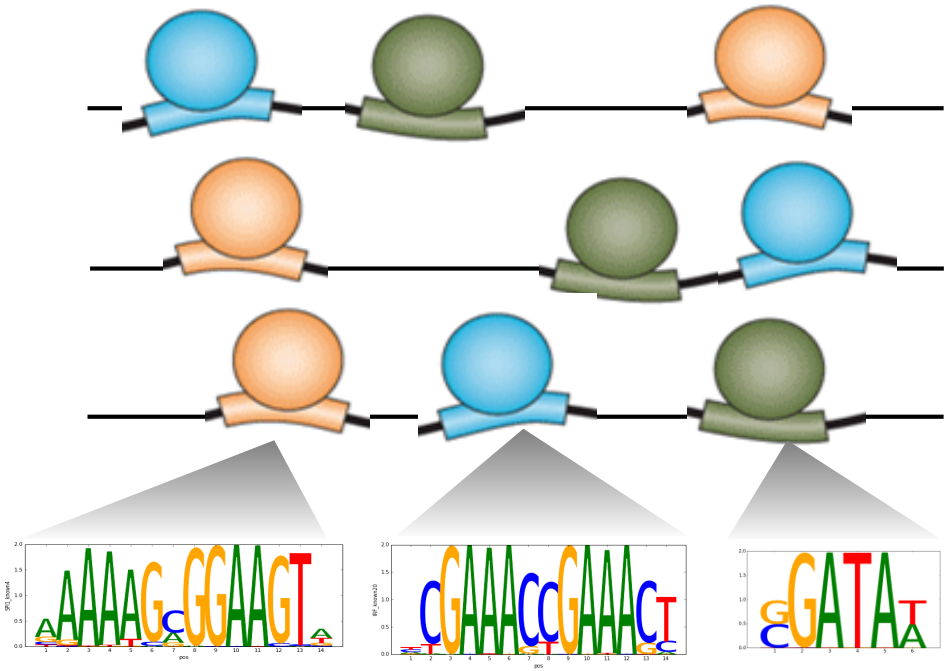
Key properties of regulatory sequence



HOMOTYPIC MOTIF DENSITY

Regulatory sequences often contain **more than one binding instance** of a TF resulting in **homotypic clusters of motifs of the same TF**

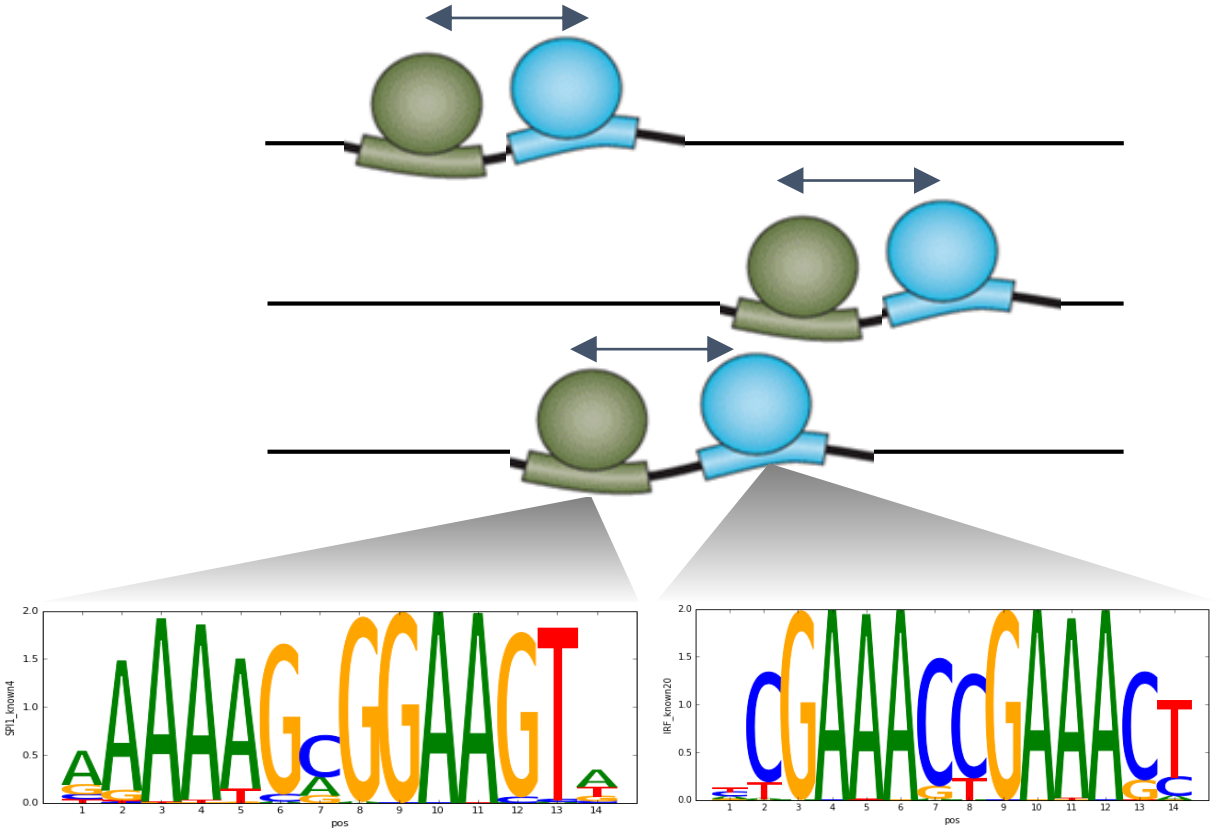
Key properties of regulatory sequence



HETEROTYPIC MOTIF COMBINATIONS

Regulatory sequences often bound by **combinations of TFs** resulting in **heterotypic clusters of motifs of different TFs**

Key properties of regulatory sequence



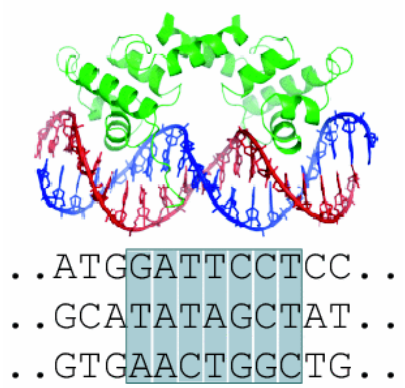
SPATIAL GRAMMARS OF HETEROTYPIC MOTIF COMBINATIONS

Regulatory sequences are often bound by **combinations of TFs** with specific **spatial and positional constraints** resulting in distinct **motif grammars**

Sequence motifs

GGATAA
CGATAA
CGATAT
GGATAT

Set of aligned sequences
Bound by TF



Sequence motifs

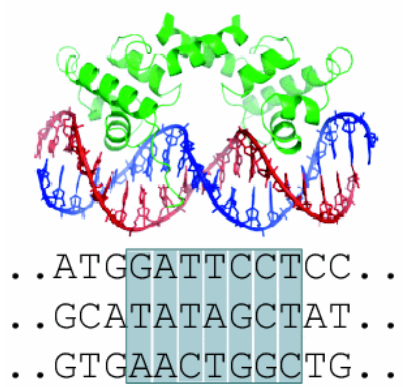
$$p_i(x_i = a_i)$$

GGATAA
CGATAA
CGATAT
GGATAT

Set of aligned sequences
Bound by TF

A	0	0	1	0	1	0.5
C	0.5	0	0	0	0	0
G	0.5	1	0	0	0	0
T	0	0	0	1	0	0.5

Position weight matrix
(PWM)



Sequence motifs

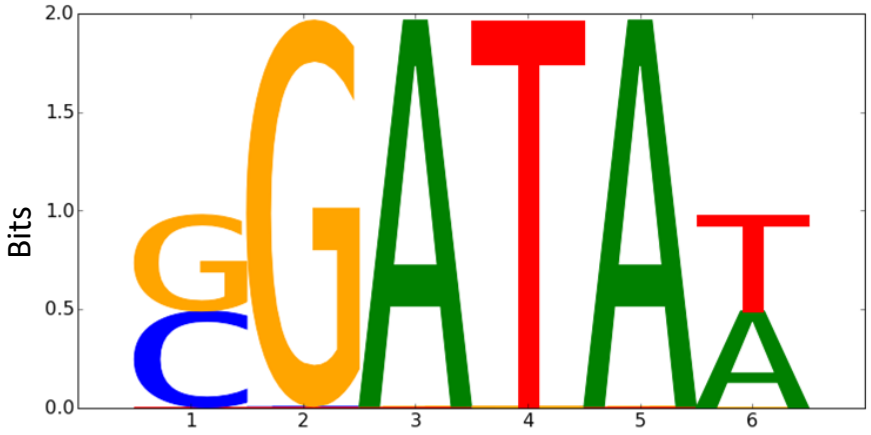
GGATAA
CGATAA
CGATAT
GGATAT

Set of aligned sequences
Bound by TF

$$p_i(x_i = a_i)$$

A	0	0	1	0	1	0.5
C	0.5	0	0	0	0	0
G	0.5	1	0	0	0	0
T	0	0	0	1	0	0.5

Position weight matrix
(PWM)



PWM
logo

https://en.wikipedia.org/wiki/Sequence_logo

The information content (y-axis) of position i is given by:[2]

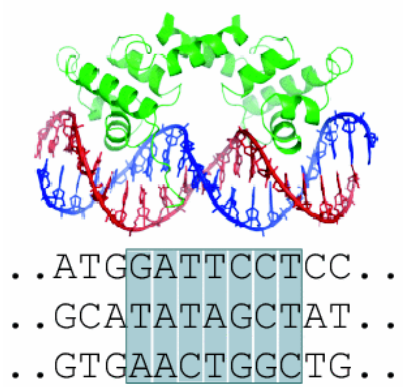
$$R_i = \log_2(4) - (H_i + e_n)$$

where H_i is the uncertainty (sometimes called the Shannon entropy) of position i

$$H_i = -\sum f_{a,i} \times \log_2 f_{a,i}$$

. The height of letter a in column i is given by

$$\text{height} = f_{a,i} \times R_i$$



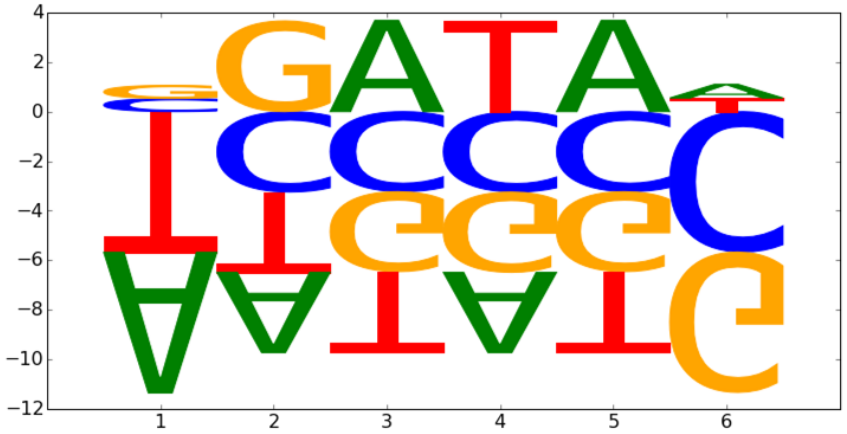
Sequence motifs

Accounting for genomic background nucleotide distribution

Position-specific scoring matrix (PSSM)

$$\log_2\left(\frac{p_i(x_i = a_i)}{p_{bg}(x_i = a_i)}\right)$$

A	-5.7	-3.2	3.7	-3.2	3.7	0.6
C	0.5	-3.2	-3.2	-3.2	-3.2	-5.7
G	0.5	3.7	-3.2	-3.2	-3.2	-5.7
T	-5.7	-3.2	-3.2	3.7	-3.2	0.5



PSSM logo

Scoring a sequence with a motif PSSM

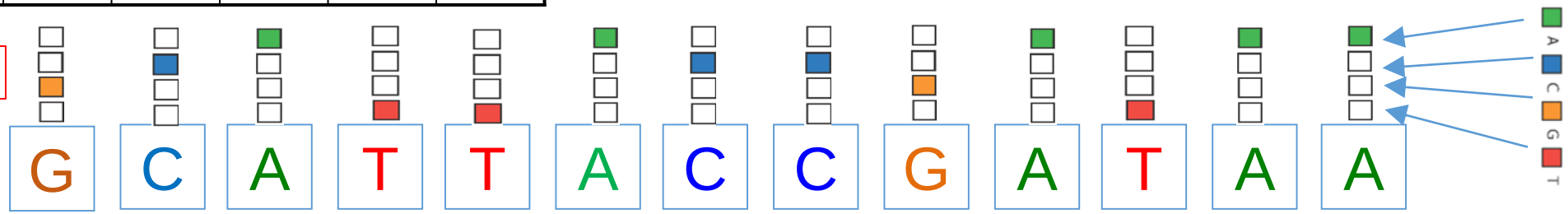
PSSM parameters

Scoring weights **W**

A	-5.7	-3.2	3.7	-3.2	3.7	0.6
C	0.5	-3.2	-3.2	-3.2	-3.2	-5.7
G	0.5	3.7	-3.2	-3.2	-3.2	-5.7
T	-5.7	-3.2	-3.2	3.7	-3.2	0.5

One-hot encoding **(X)**

Input sequence



Convolution:
Scoring a sequence with a PSSM

Motif match Scores
sum(W * x)

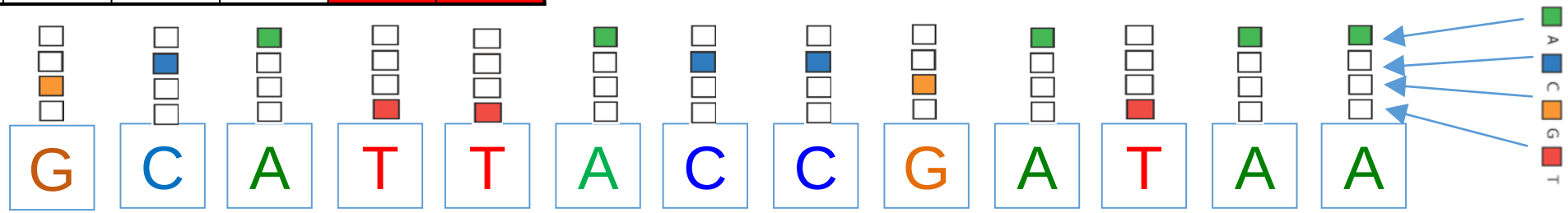
	-5.4											

Scoring weights W

A	**-5.7**	**-3.2**	**3.7**	**-3.2**	**3.7**	**0.6**
C	**0.5**	**-3.2**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
G	**0.5**	**3.7**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
T	**-5.7**	**-3.2**	**-3.2**	**3.7**	**-3.2**	**0.5**

One-hot encoding (X)

Input sequence



Convolution

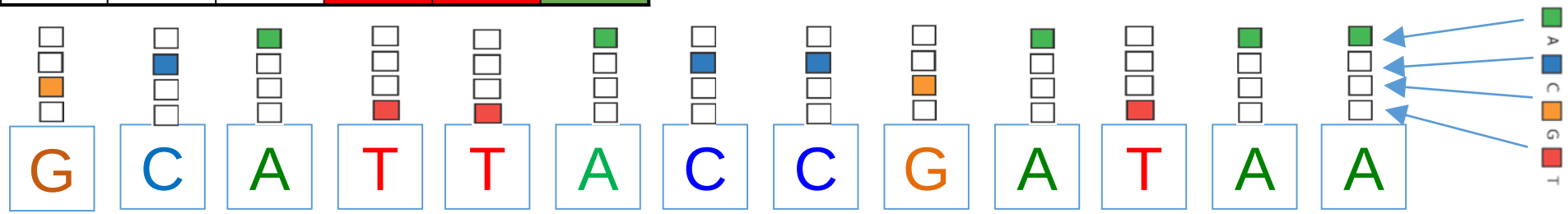
Motif match Scores
sum(W * x)

	-5.4	2.0										

Scoring weights W

A	**-5.7**	**-3.2**	**3.7**	**-3.2**	**3.7**	**0.6**
C	**0.5**	**-3.2**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
G	**0.5**	**3.7**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
T	**-5.7**	**-3.2**	**-3.2**	**3.7**	**-3.2**	**0.5**

One-hot encoding (X)



Input sequence

G C A T T A C C G A T A A

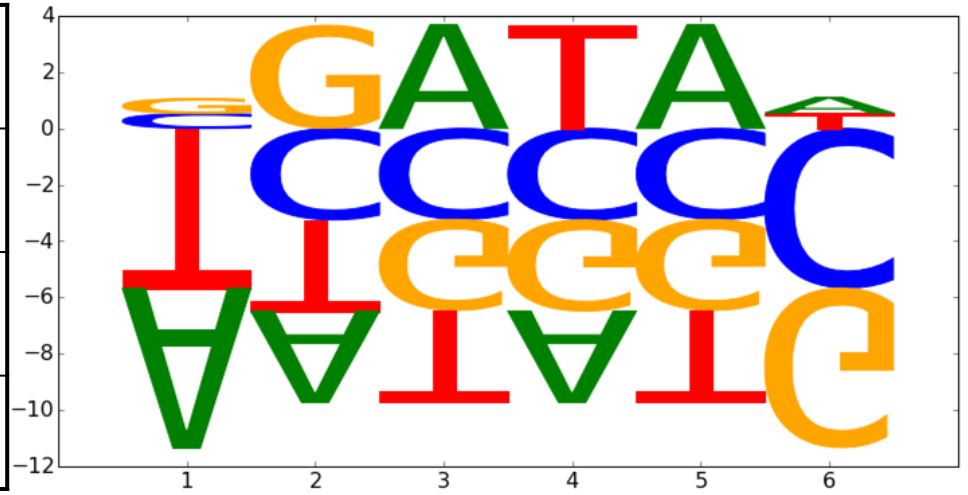
Convolution

Motif match Scores
sum(W * x)

-2.2	-5.4	2.0	-4.3	-24	-17	-18	-11	-12	16	-5.5	-8.5	-5.2

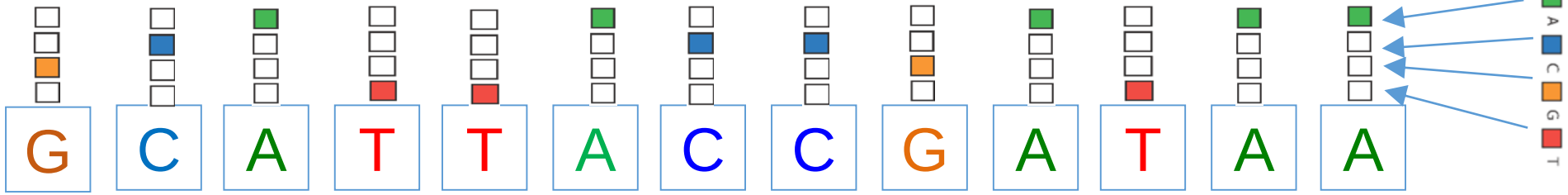
Scoring weights W

A	**-5.7**	**-3.2**	**3.7**	**-3.2**	**3.7**	**0.6**
C	**0.5**	**-3.2**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
G	**0.5**	**3.7**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
T	**-5.7**	**-3.2**	**-3.2**	**3.7**	**-3.2**	**0.5**



One-hot encoding (X)

Input sequence



Thresholding scores

Thresholded Motif Scores
max(0, W*x)

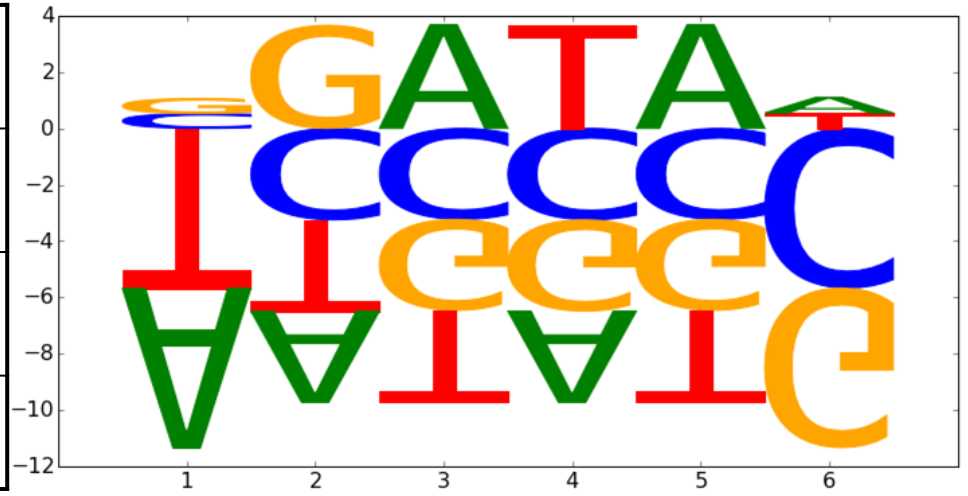
0	0	2.0	0	0	0	0	0	0	16	0	0	0

Motif match Scores
W*x

-2.2	-5.4	2.0	-4.3	-24	-17	-18	-11	-12	16	-5.5	-8.5	-5.2

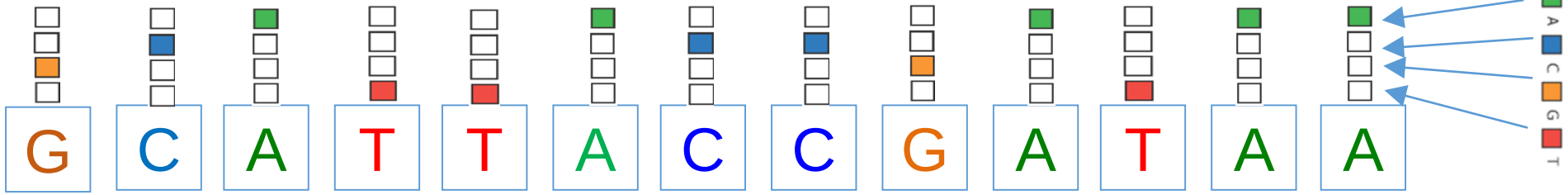
Scoring weights
W

A	**-5.7**	**-3.2**	**3.7**	**-3.2**	**3.7**	**0.6**
C	**0.5**	**-3.2**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
G	**0.5**	**3.7**	**-3.2**	**-3.2**	**-3.2**	**-5.7**
T	**-5.7**	**-3.2**	**-3.2**	**3.7**	**-3.2**	**0.5**

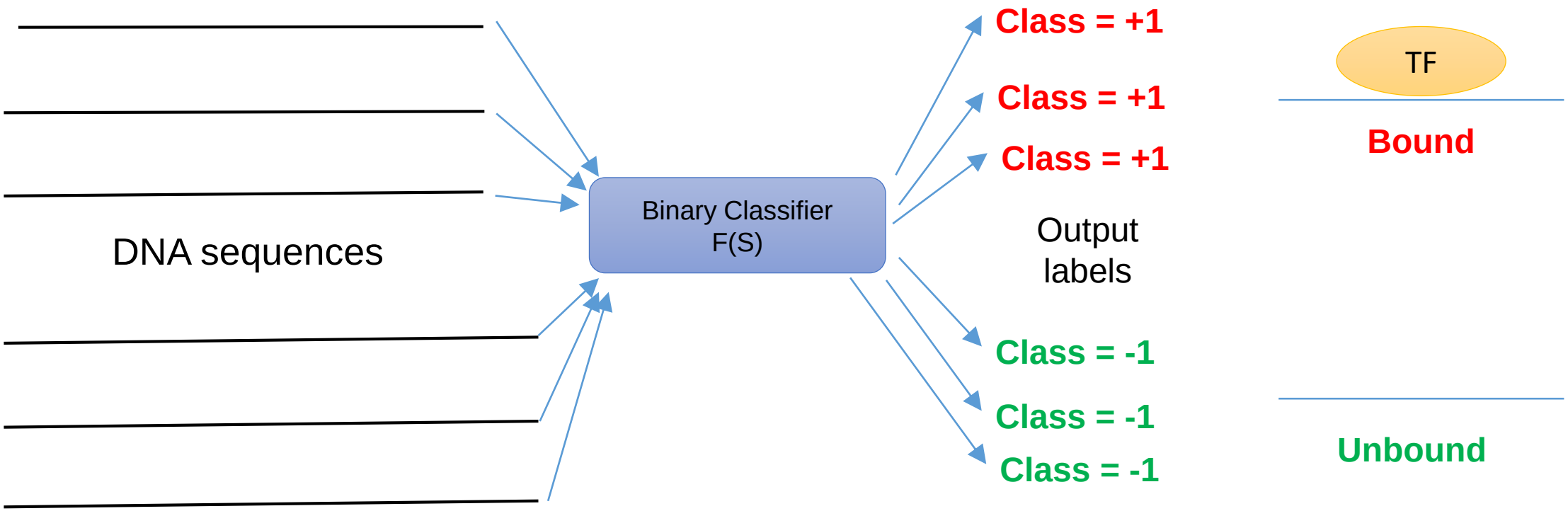


One-hot encoding (X)

Input sequence

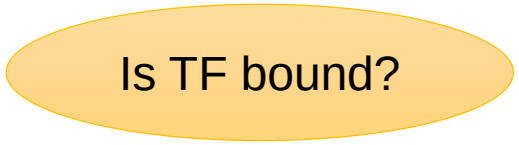


Binary classification of regulatory DNA sequences



An artificial neuron on DNA sequence is a motif pattern detector

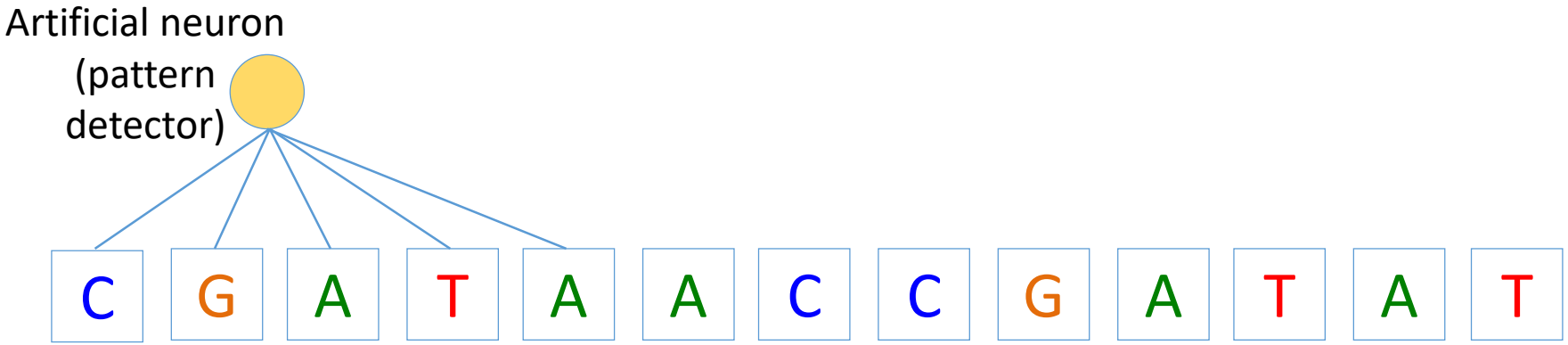
Binary Output:
Yes (1) vs No (0)



	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

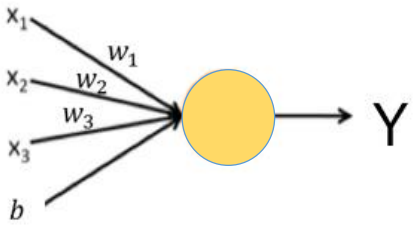
One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector



One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector



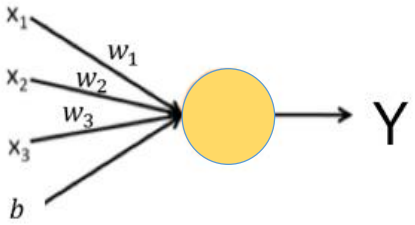
$$Y = F(x_1, x_2, x_3)$$

Artificial neuron (pattern detector)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector



$$Y = F(x_1, x_2, x_3)$$

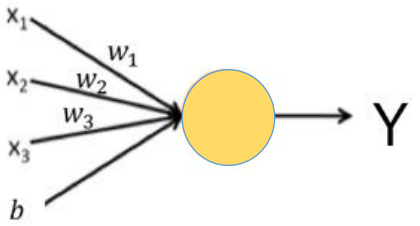
Artificial neuron (pattern detector)

Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1 . x_1 + w_2 . x_2 + w_3 . x_3 + b$$

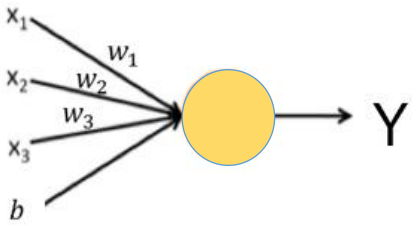
Artificial neuron (pattern detector)

Input $\mathbf{X_{1..20}}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2

Artificial neuron (pattern detector)

Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

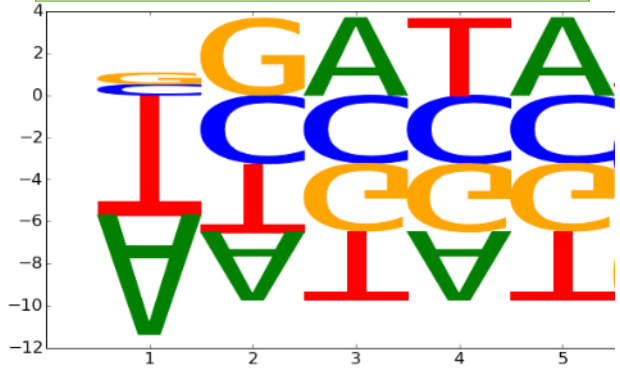
Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

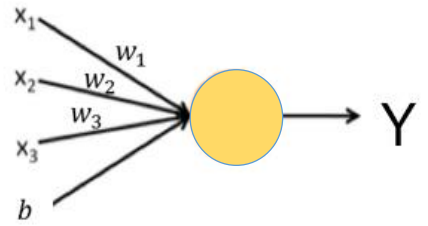
One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector

Neuron = Motif (position weight matrix)!



A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

Artificial neuron (pattern detector)

Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

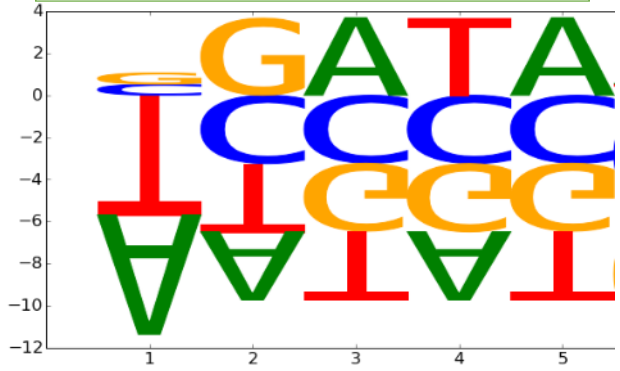
Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

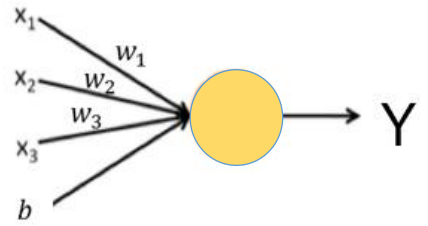
An artificial neuron on DNA sequence is a motif pattern detector

Neuron = Motif (position weight matrix)!



Z =W*X = 0.5+3.7+3.7+3.7 = 11.6

A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

Artificial neuron (pattern detector)

Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

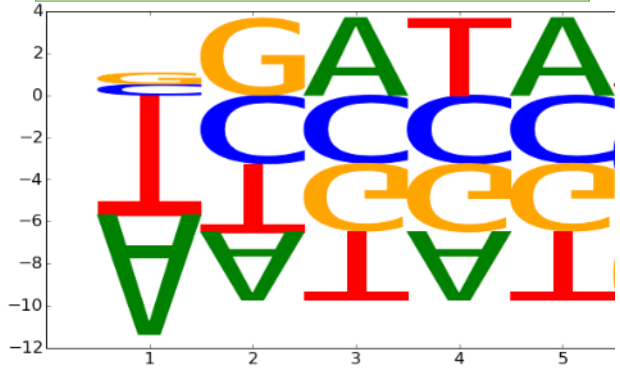
Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

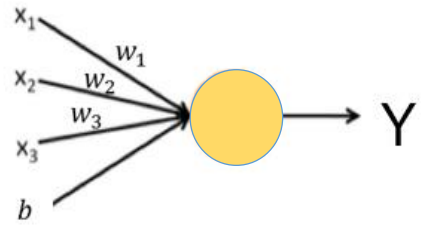
An artificial neuron on DNA sequence is a motif pattern detector

Neuron = Motif (position weight matrix)!



Z =W*X = 0.5+3.7+3.7+3.7 = 11.6

A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

$$Y = h(Z)$$

Non-linear function

Artificial neuron (pattern detector)

Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

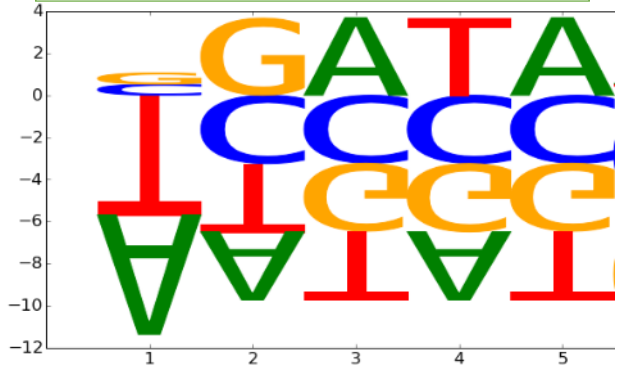
Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

An artificial neuron on DNA sequence is a motif pattern detector

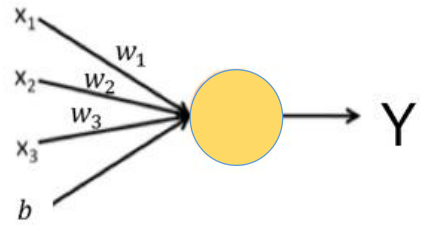
Neuron = Motif (position weight matrix)!



Y = 0.9996

Z =W*X = 0.5+3.7+3.7+3.7 = 11.6

A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

$$Y = h(Z)$$

Non-linear function

Artificial neuron (pattern detector)

Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

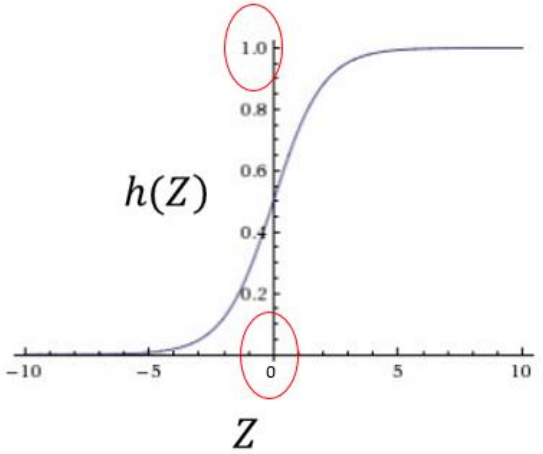
Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

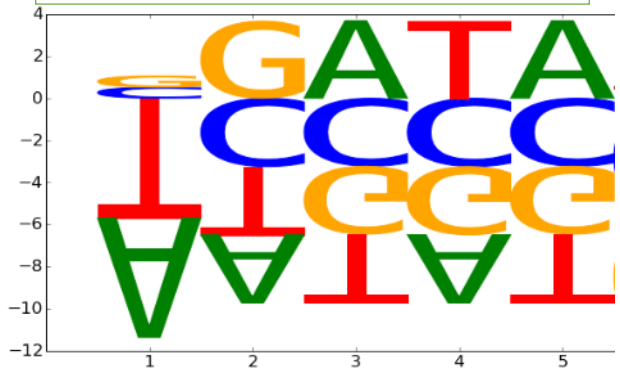
Logistic / Sigmoid

Useful for predicting probabilities



An artificial neuron on DNA sequence is a motif pattern detector

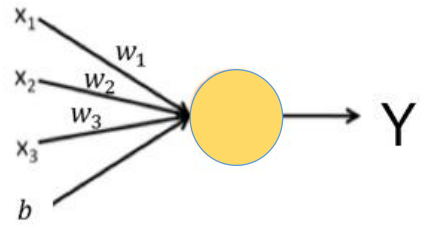
Neuron = Motif (position weight matrix)!



Y = 11.6

Z =W*X = 0.5+3.7+3.7+3.7 = 11.6

A	-5.7	-3.2	3.7	-3.2	3.7
C	0.5	-3.2	-3.2	-3.2	-3.2
G	0.5	3.7	-3.2	-3.2	-3.2
T	-5.7	-3.2	-3.2	3.7	-3.2



$$Y = F(x_1, x_2, x_3)$$

weight parameters

$$Z = w_1.x_1 + w_2.x_2 + w_3.x_3 + b$$

$$Y = h(Z)$$

Non-linear function

Artificial neuron (pattern detector)

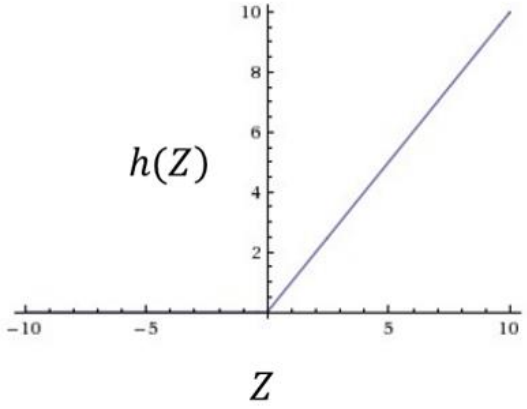
Neuron weights $W_{1..20}$ (20 values = 5 positions x 4 nucleotides)

Input $X_{1..20}$ (20 binary values = 5 positions x 4 nucleotides)

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

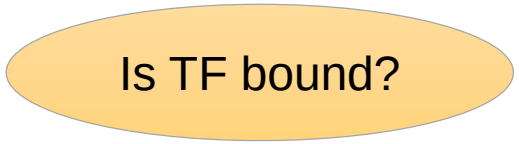
One-hot encoded input: DNA sequence represented as ones and zeros

ReLu (Rectified Linear Unit)
Useful for thresholding



Deep convolutional neural network: Scanning pattern detectors

Binary Output:
Yes (1) vs No (0)



	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

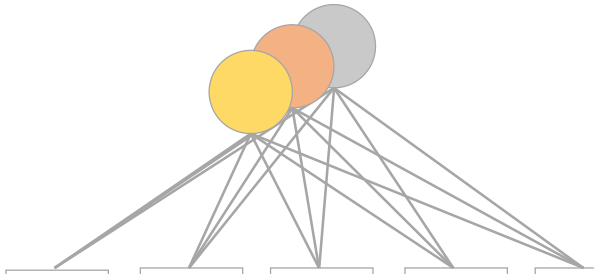
One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

Binary Output:
Yes (1) vs No (0)

Is TF bound?

3 artificial neurons
(motif detectors)



	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

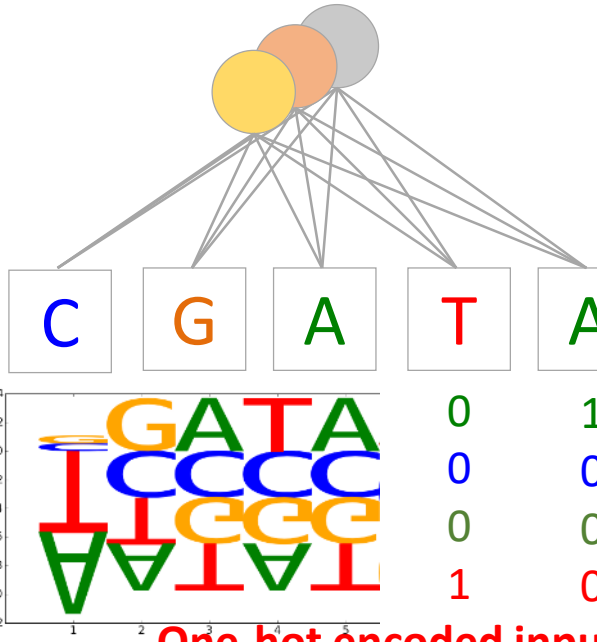
One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

Binary Output:
Yes (1) vs No (0)

Is TF bound?

3 artificial neurons
(motif detectors)



C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	1	1	0	0	0	1	0	1	0
C	0	0	0	1	1	0	0	0	0	0
G	0	0	0	0	0	1	0	0	0	0
T	1	0	0	0	0	0	0	1	0	1

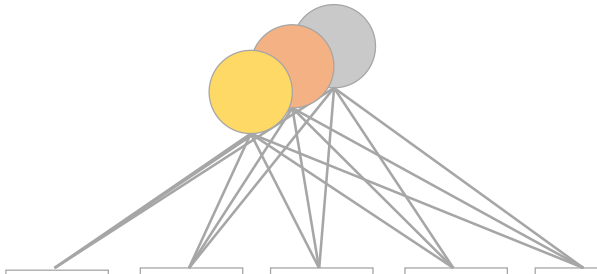
One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

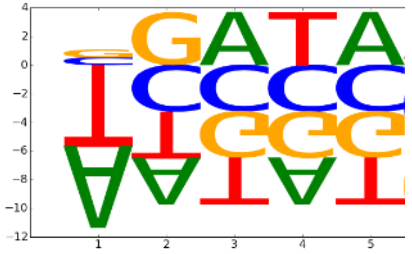
Binary Output:
Yes (1) vs No (0)

Is TF bound?

3 artificial neurons
(motif detectors)



	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1



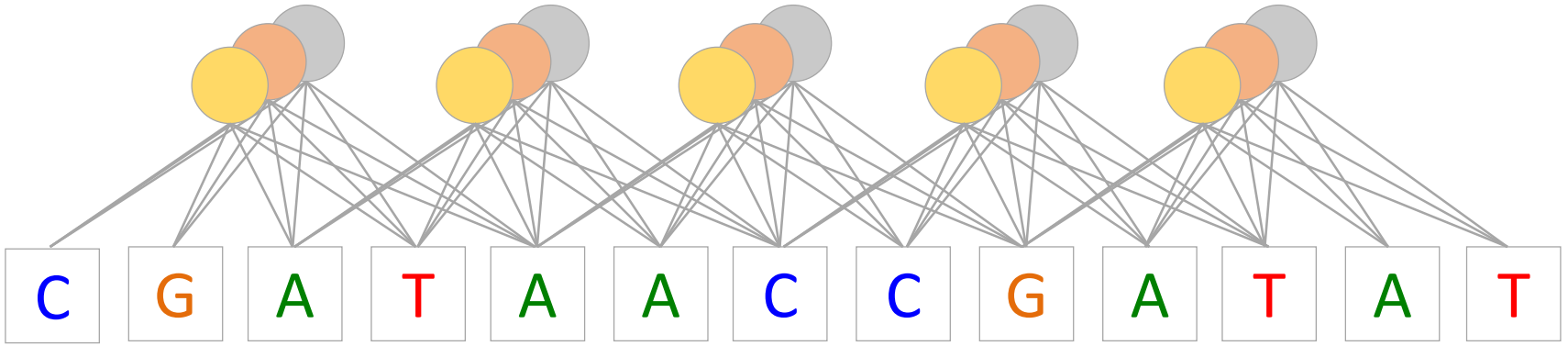
One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

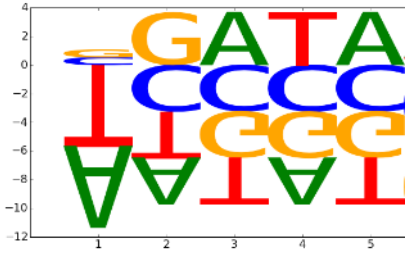
Binary Output:
Yes (1) vs No (0)

Is TF bound?

3 artificial neurons
(motif detectors)



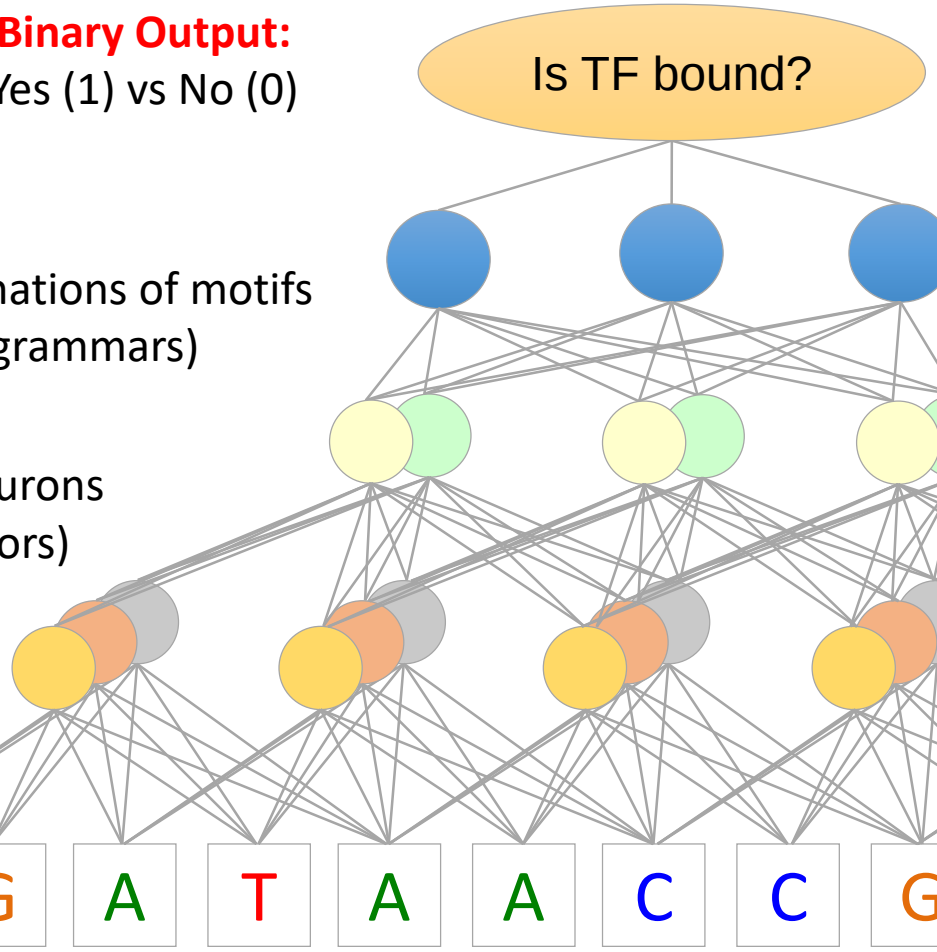
	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1



One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

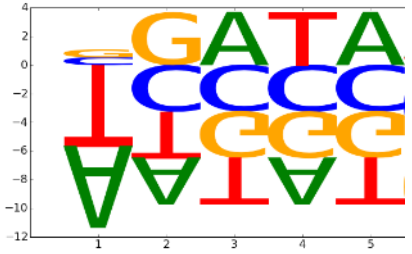
Binary Output:
Yes (1) vs No (0)



Combinations of motifs
(motif grammars)

3 artificial neurons
(motif detectors)

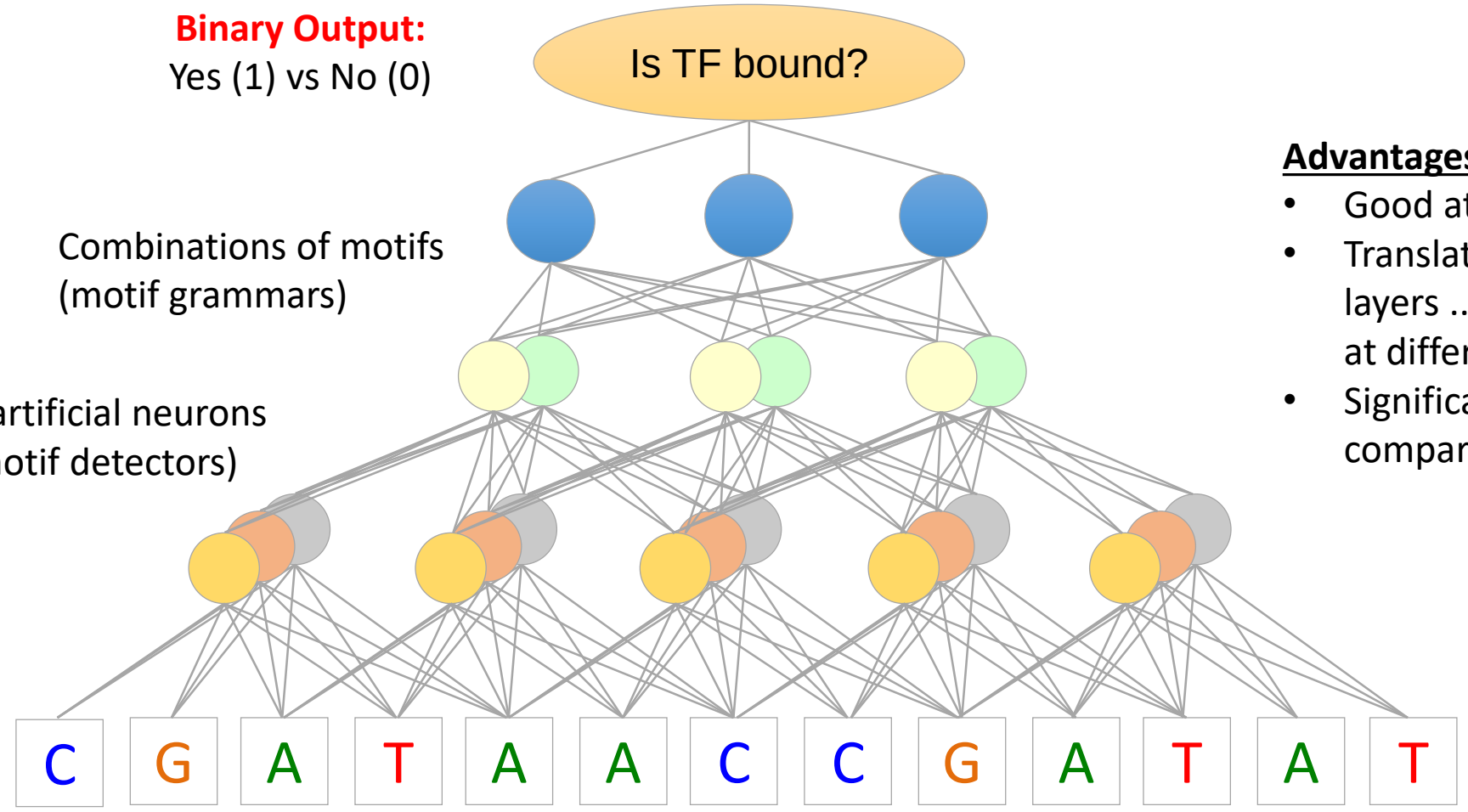
	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1



One-hot encoded input: DNA sequence represented as ones and zeros

Deep convolutional neural network: Scanning pattern detectors

Binary Output:
Yes (1) vs No (0)

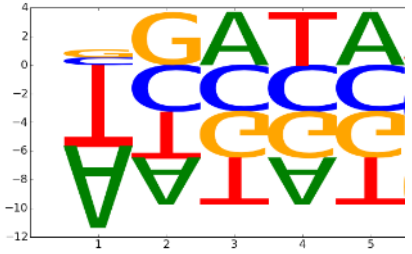


A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

One-hot encoded input: DNA sequence represented as ones and zeros

Advantages of convnets

- Good at capturing local patterns
- Translational invariance (will pooling layers .. Next slide). Patterns can occur at different positions in the input.
- Significant reduction in parameters compared to denseNets



Pooling operations allow positional invariance

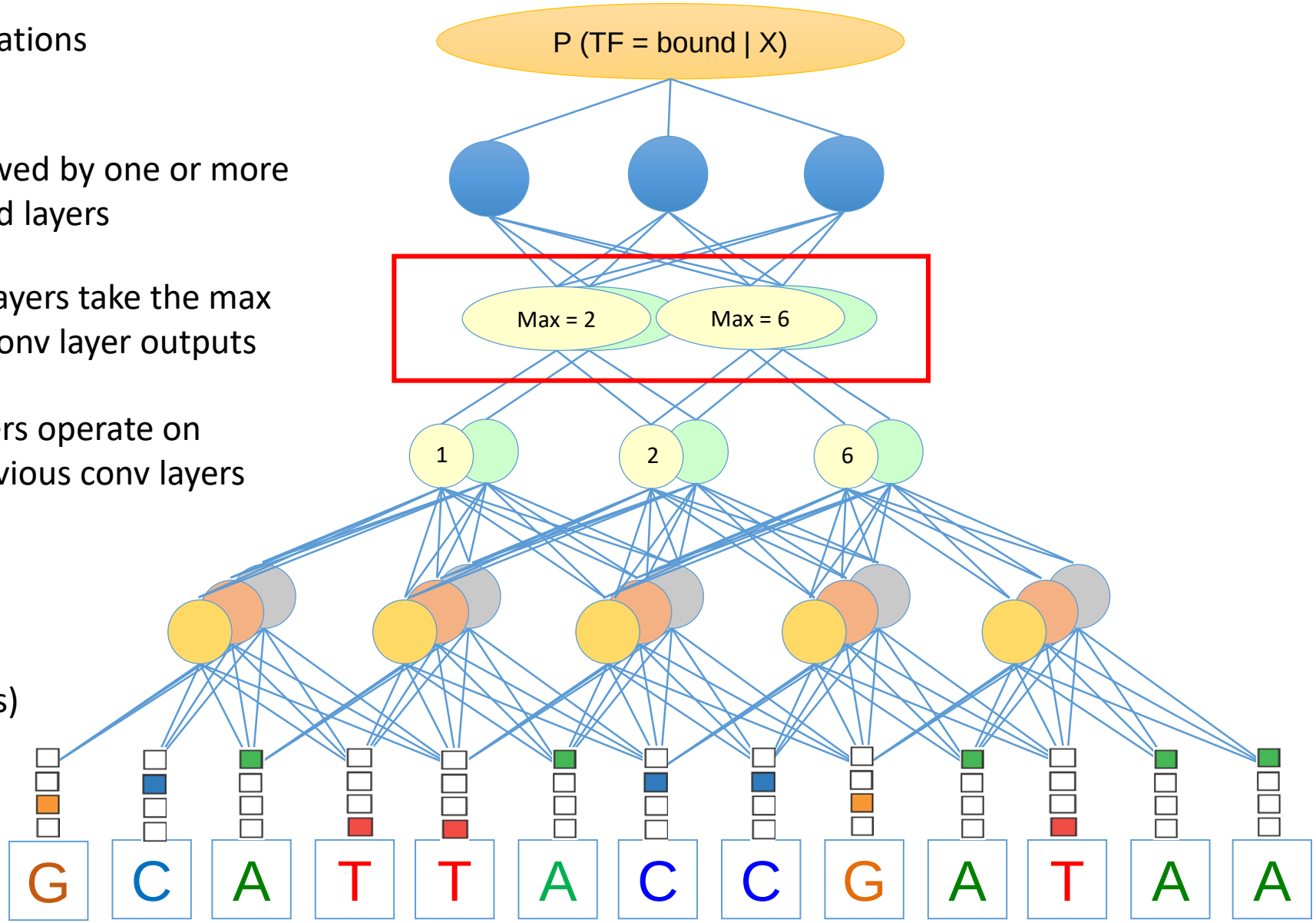
Sigmoid activations

Typically followed by one or more fully connected layers

Maxpooling layers take the max over sets of conv layer outputs

Later conv layers operate on outputs of previous conv layers

Convolutional layer (same color = shared weights)



Maxpooling layer
pool width = 2
stride = 1

Conv Layer 2
Kernel width = 3
stride = 1
num filters / num channels = 2
total neurons = 6

Conv Layer 1
Kernel width = 4
stride = 2*
num filters / num channels = 3
Total neurons = 15

***for genomics, a stride of 1 for conv layers is recommended**

Multi-task CNN

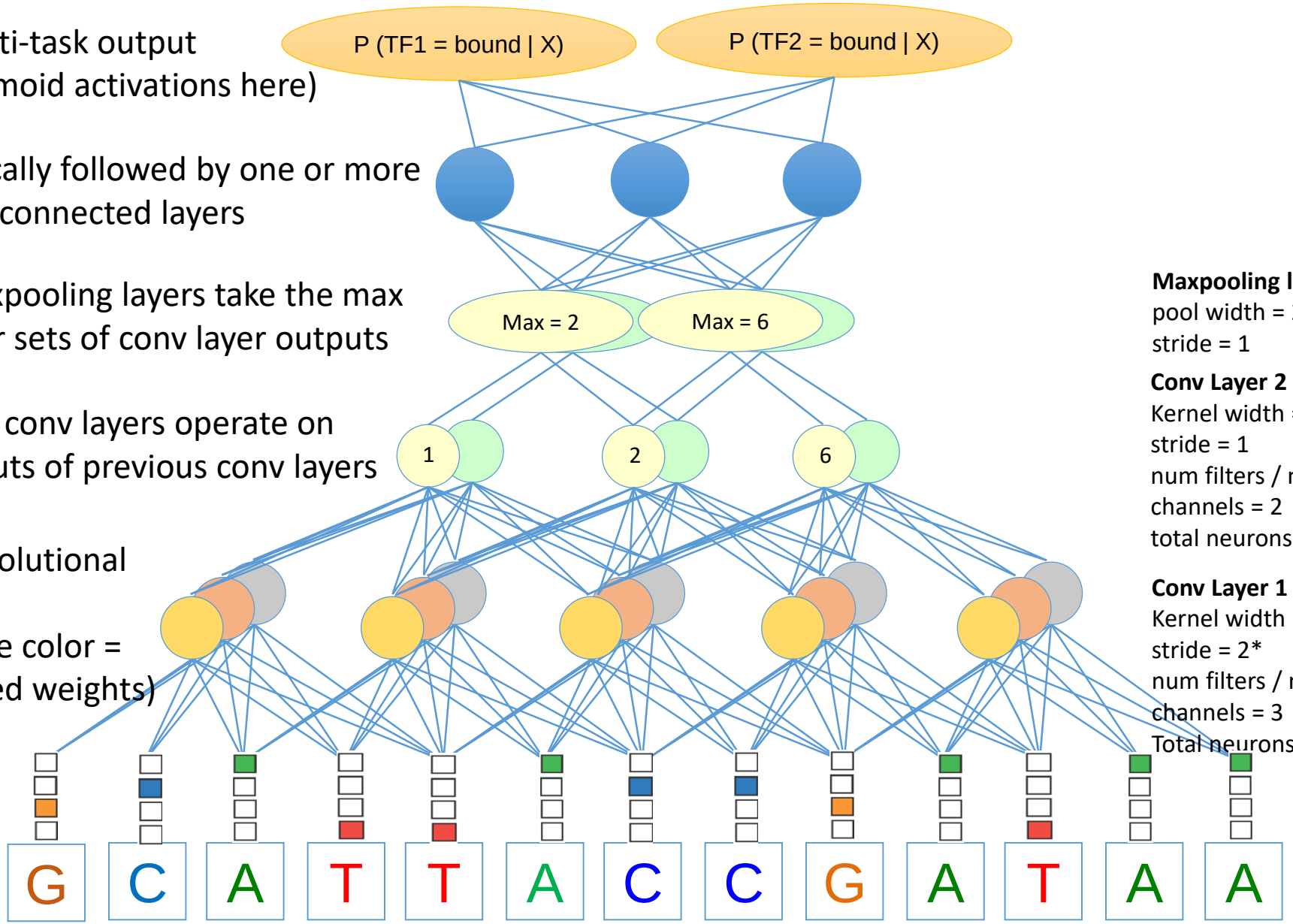
Multi-task output
(sigmoid activations here)

Typically followed by one or more fully connected layers

Maxpooling layers take the max over sets of conv layer outputs

Later conv layers operate on outputs of previous conv layers

Convolutional layer
(same color = shared weights)

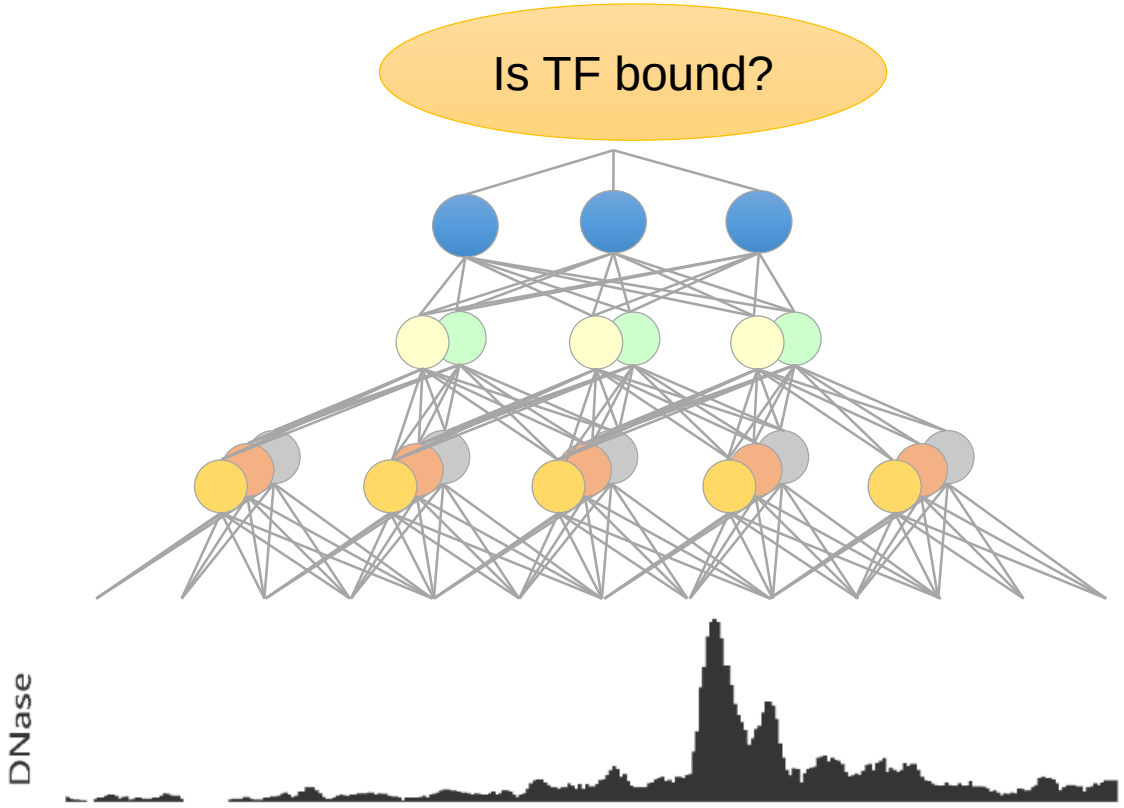


Maxpooling layer
pool width = 2
stride = 1

Conv Layer 2
Kernel width = 3
stride = 1
num filters / num channels = 2
total neurons = 6

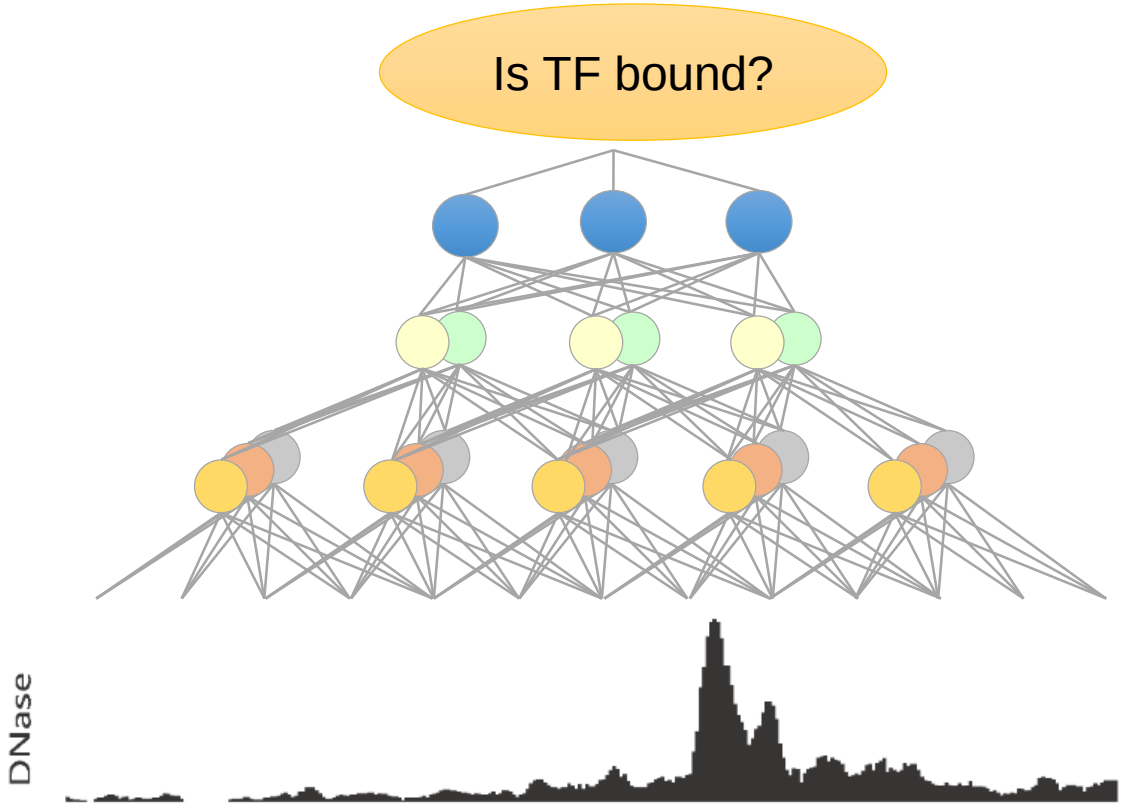
Conv Layer 1
Kernel width = 4
stride = 2*
num filters / num channels = 3
Total neurons = 15

Learning predictive patterns from continuous chromatin accessibility (DNase-seq/ATAC-seq) profiles

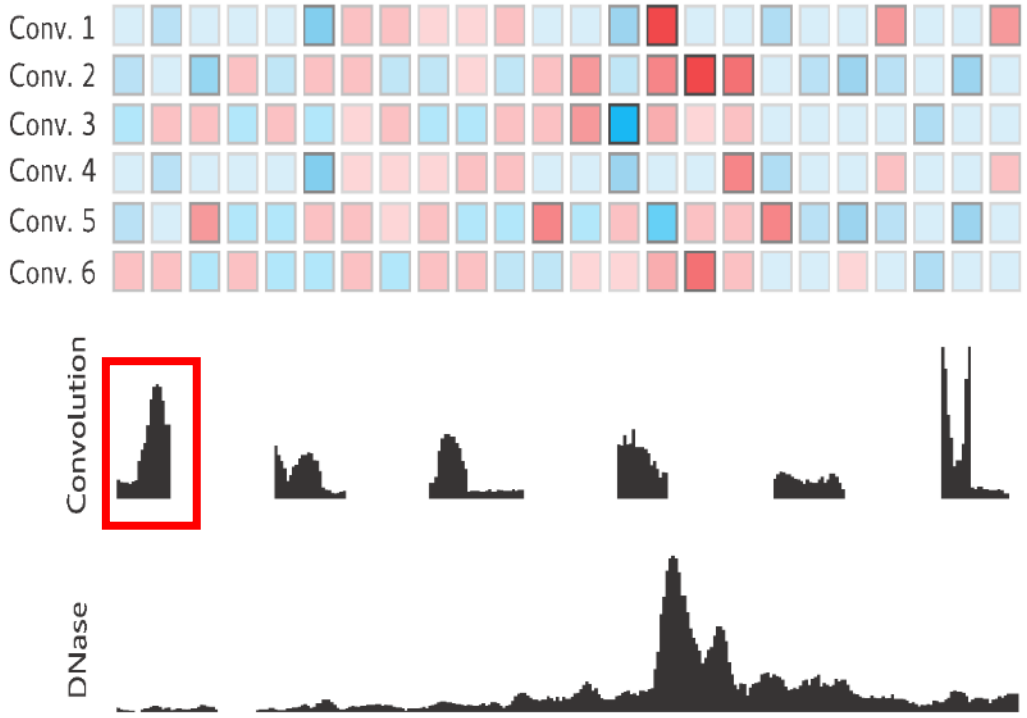


1.5 Kb DNase/ATAC-seq profile

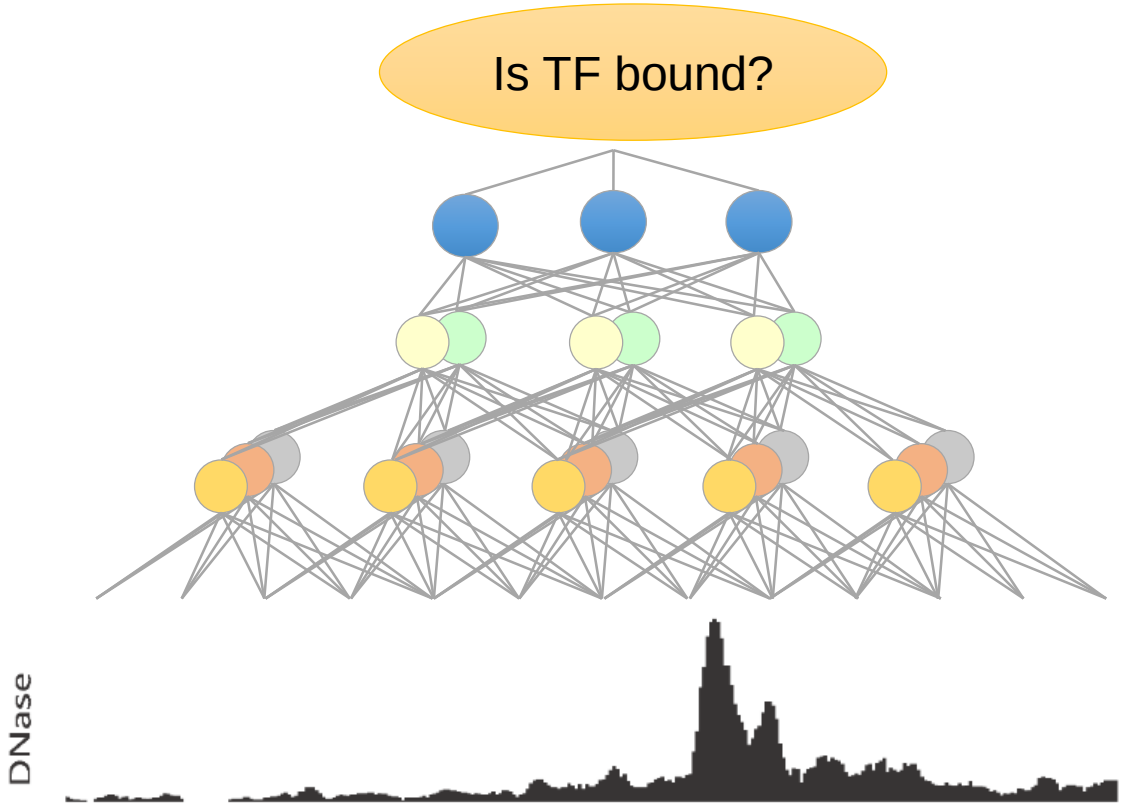
Learning predictive patterns from continuous chromatin accessibility (DNase-seq/ATAC-seq) profiles



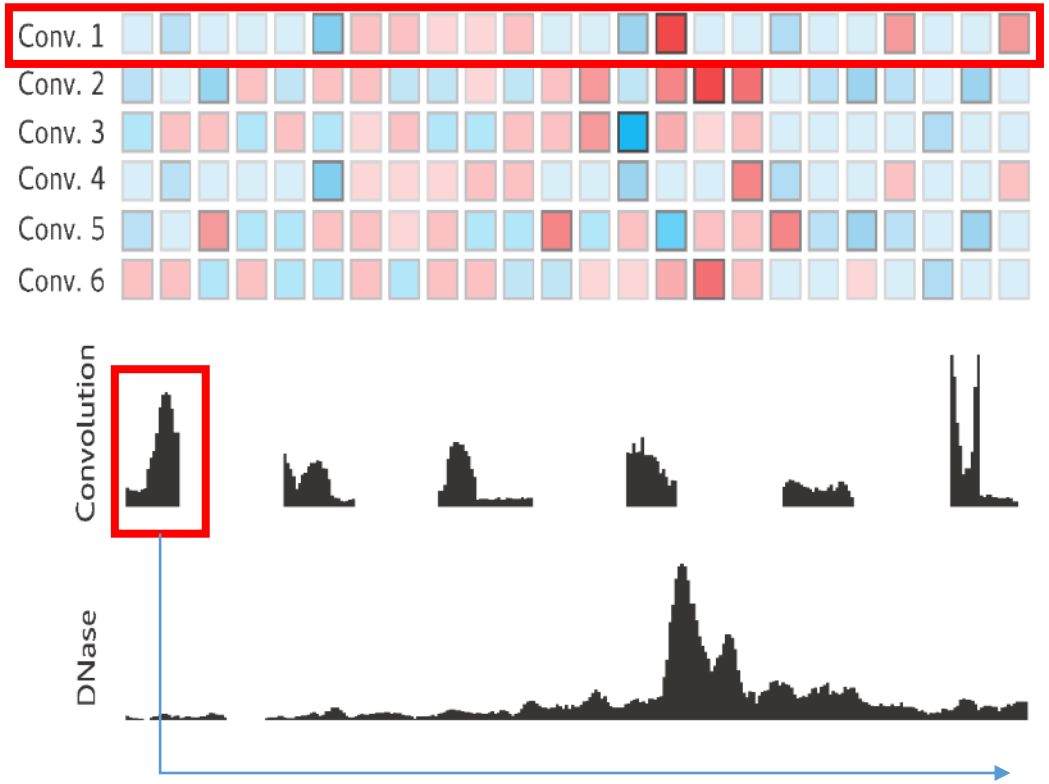
1.5 Kb DNase/ATAC-seq profile



Learning predictive patterns from continuous chromatin accessibility (DNase-seq/ATAC-seq) profiles



1.5 Kb DNase/ATAC-seq profile



Scan DNase profile using filter

Multi-modal, multi-task CNN: Integrating raw seq + continuous DNase profiles to predict binding

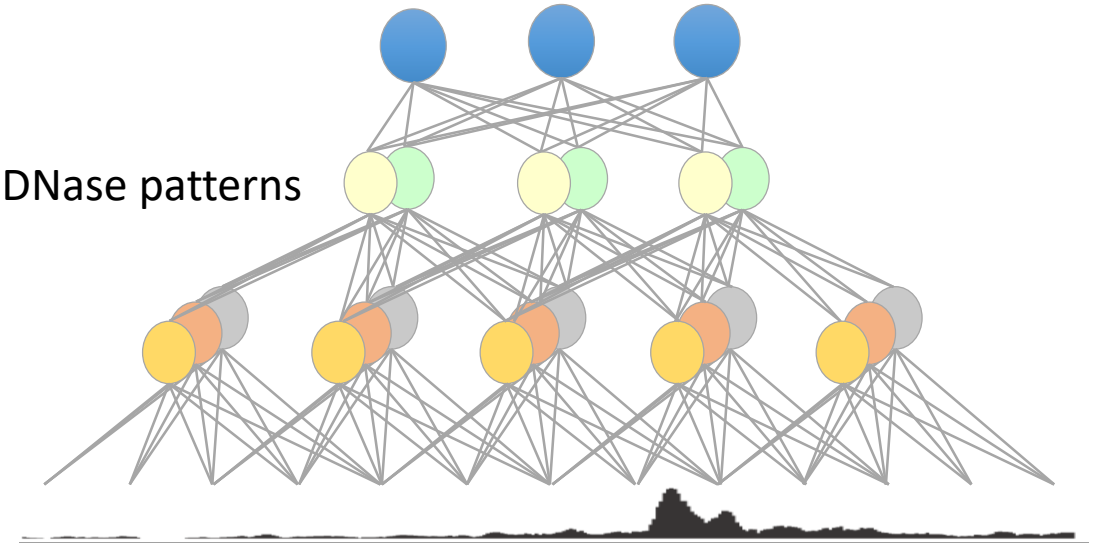
1.5 Kb DNase-seq profile

C G A T A A C C G A T A T

1.5 Kb raw DNA sequence

Multi-modal, multi-task CNN: Integrating raw seq + continuous DNase profiles to predict binding

Learn DNase patterns



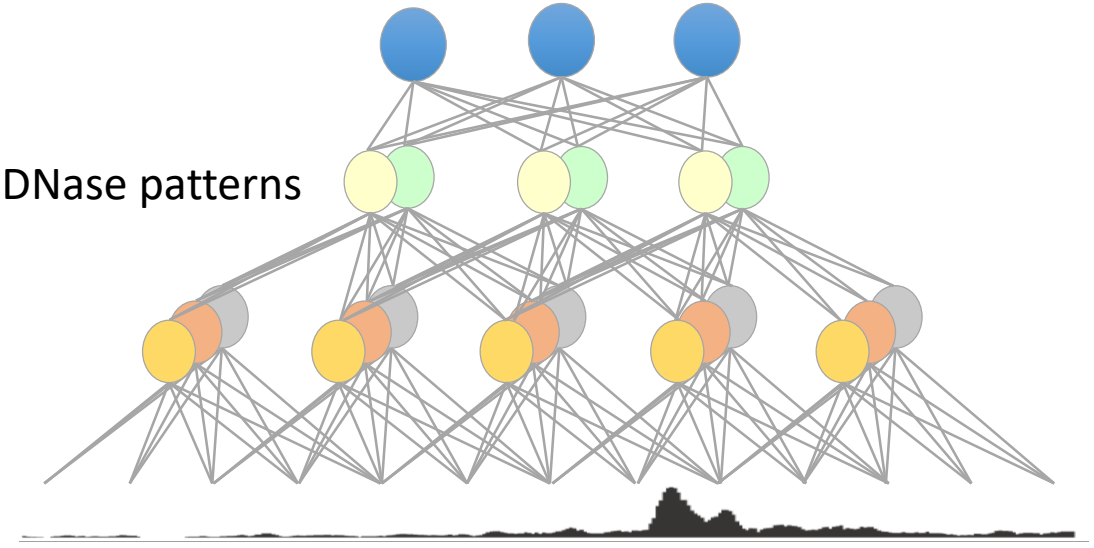
1.5 Kb DNase-seq profile

C G A T A A C C G A T A T

1.5 Kb raw DNA sequence

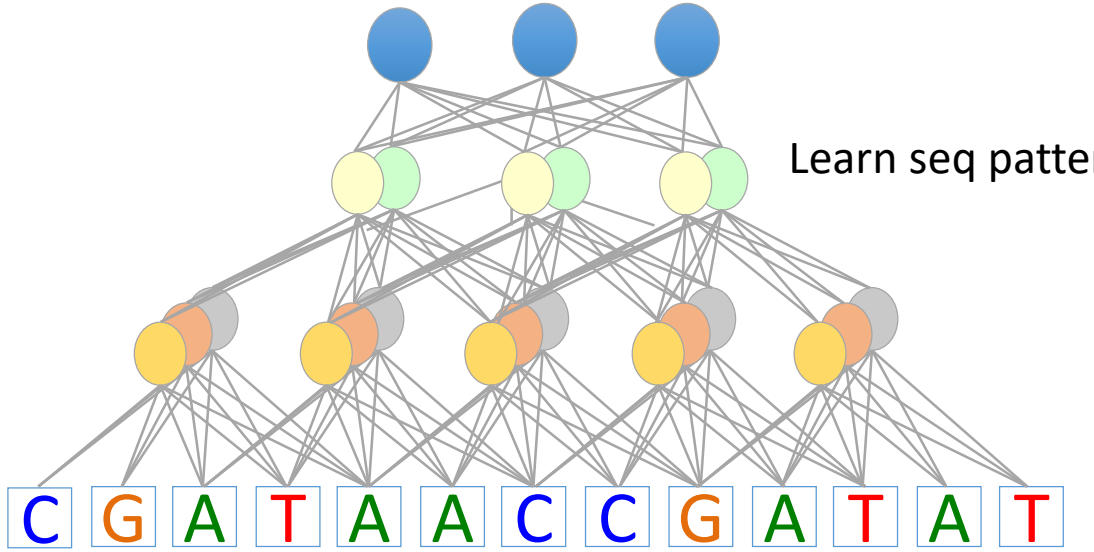
Multi-modal, multi-task CNN: Integrating raw seq + continuous DNase profiles to predict binding

Learn DNase patterns



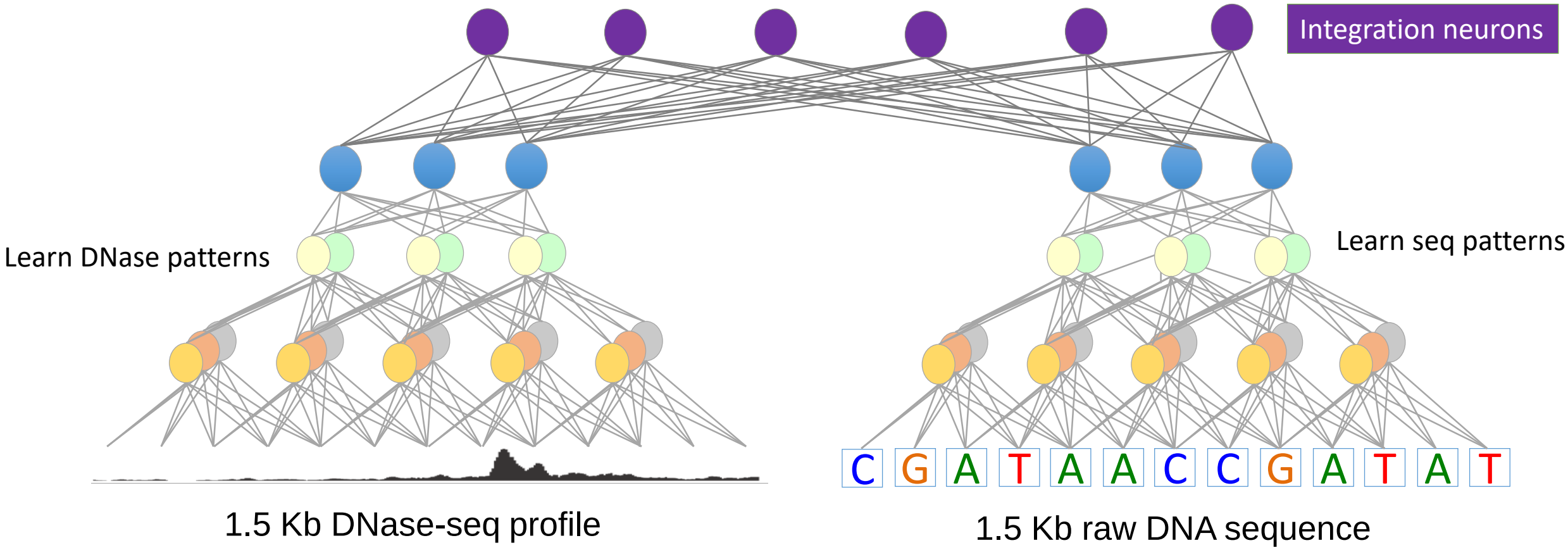
1.5 Kb DNase-seq profile

Learn seq patterns

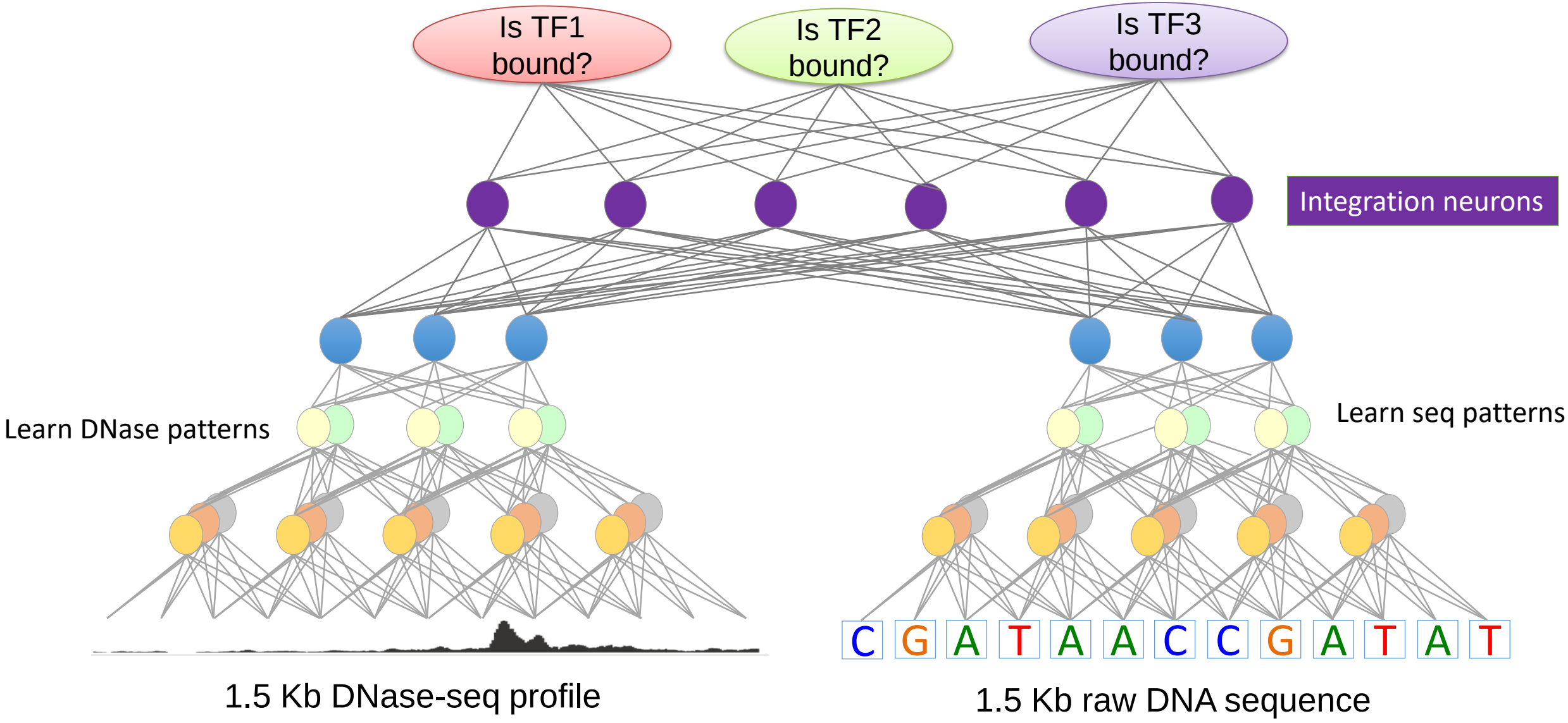


1.5 Kb raw DNA sequence

Multi-modal, multi-task CNN: Integrating raw seq + continuous DNase profiles to predict binding



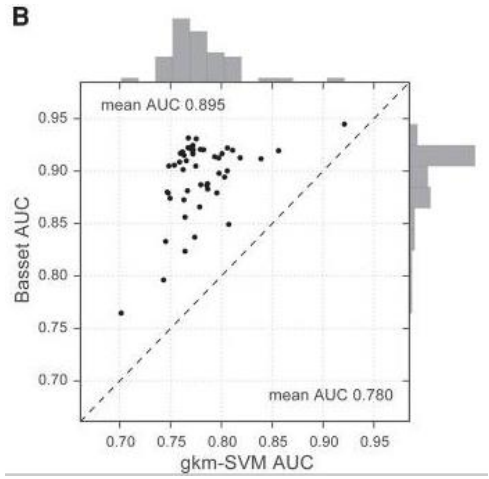
Multi-modal, multi-task CNN: Integrating raw seq + continuous DNase profiles to predict binding



Genome Res. 2016 Jul;26(7):990-9. doi: 10.1101/gr.200535.115. Epub 2016 May 3.

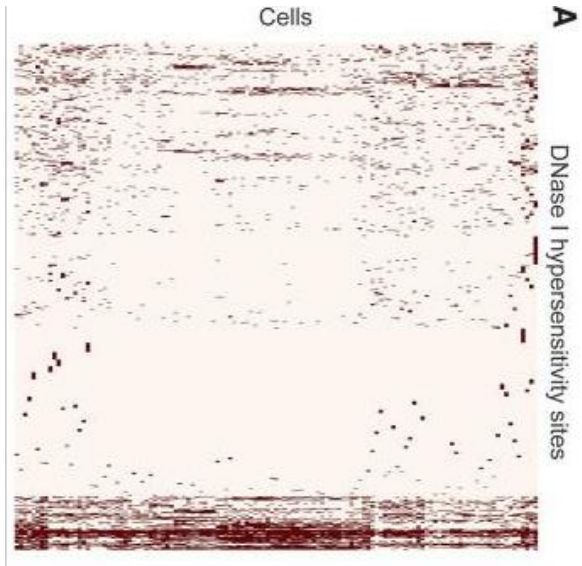
Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks.

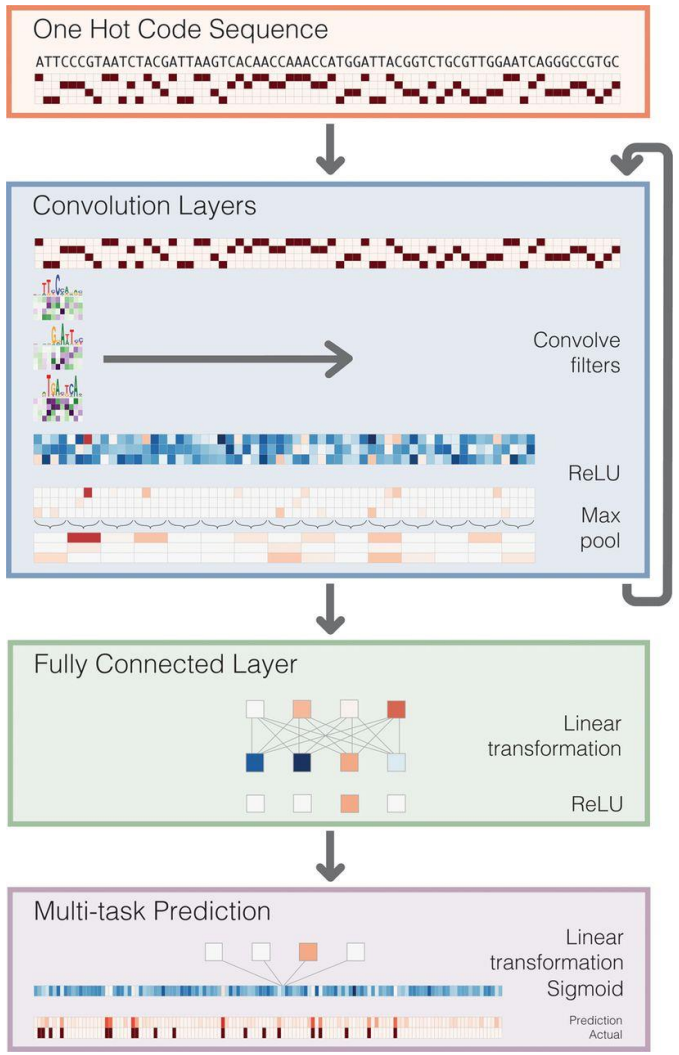
Kelley DR[1], Snoek J[2], Rinn JL[1].

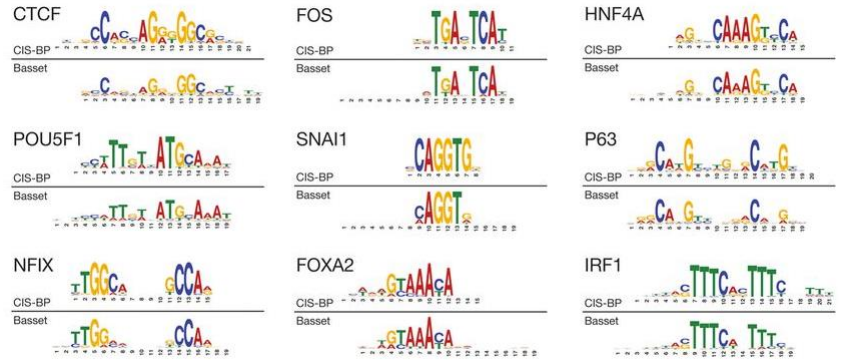


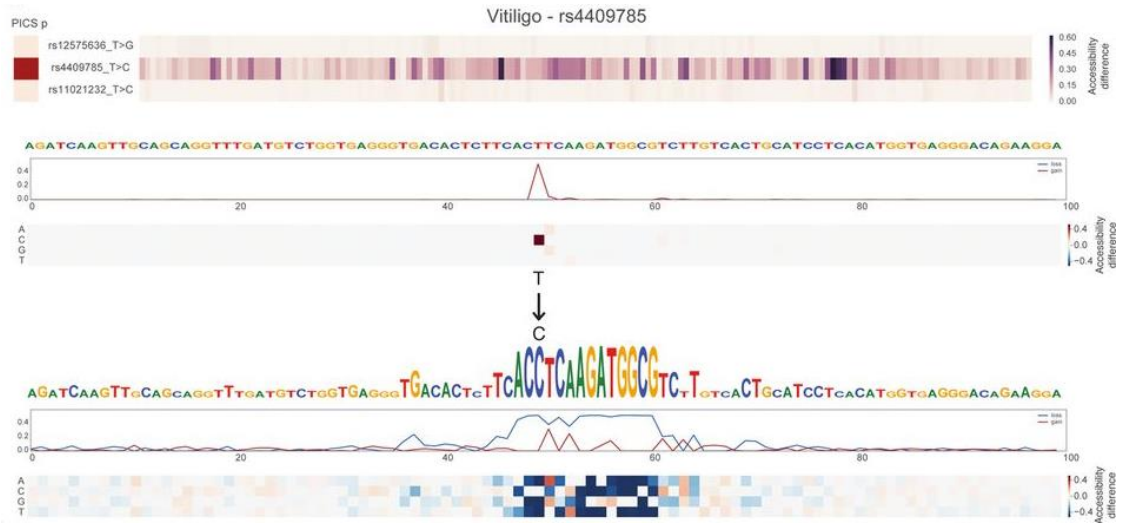
164 Tasks (cell types)

2 million examples (DHS)





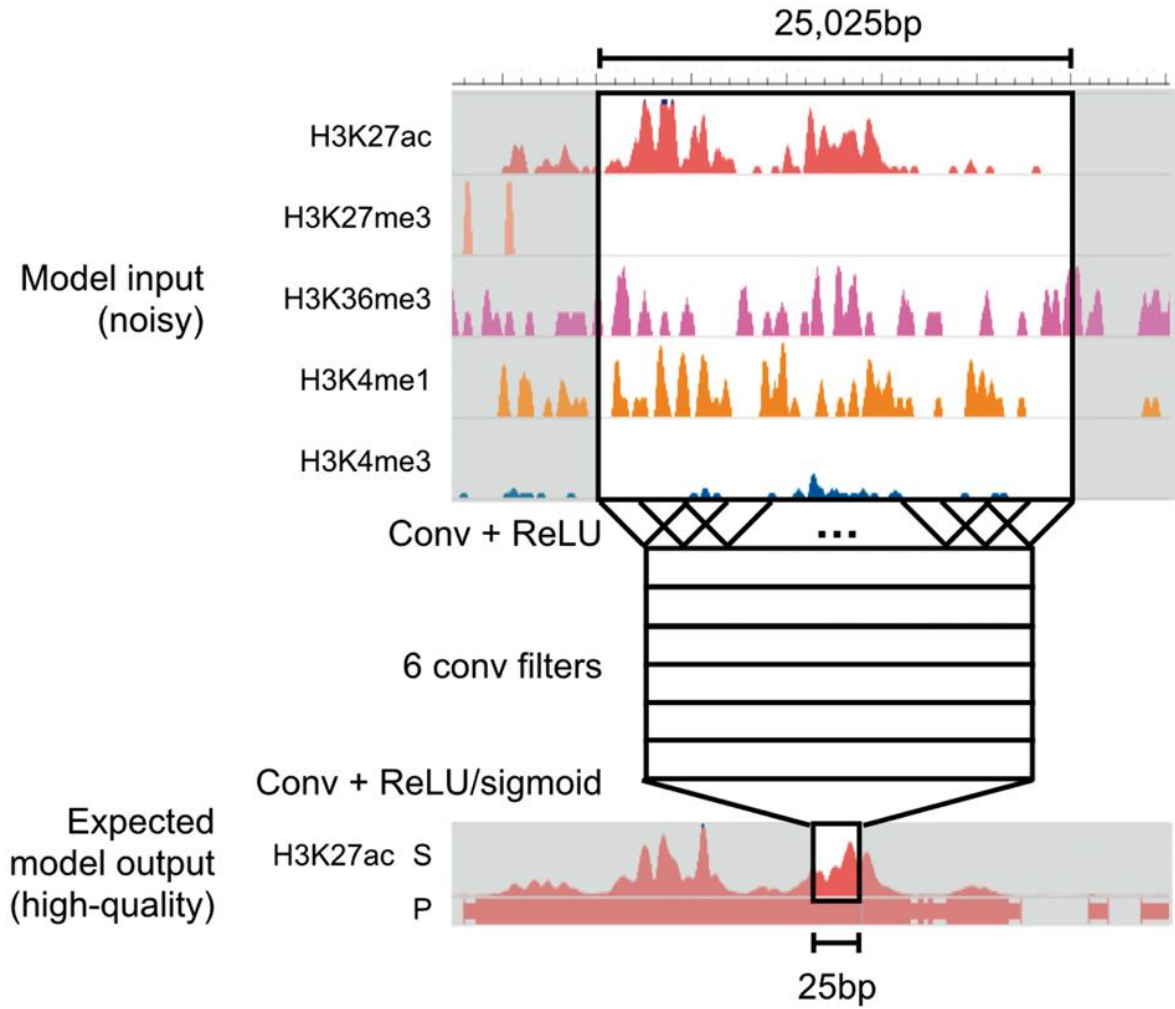




Bioinformatics. 2017 Jul 15;33(14):i225-i233. doi: 10.1093/bioinformatics/btx243.

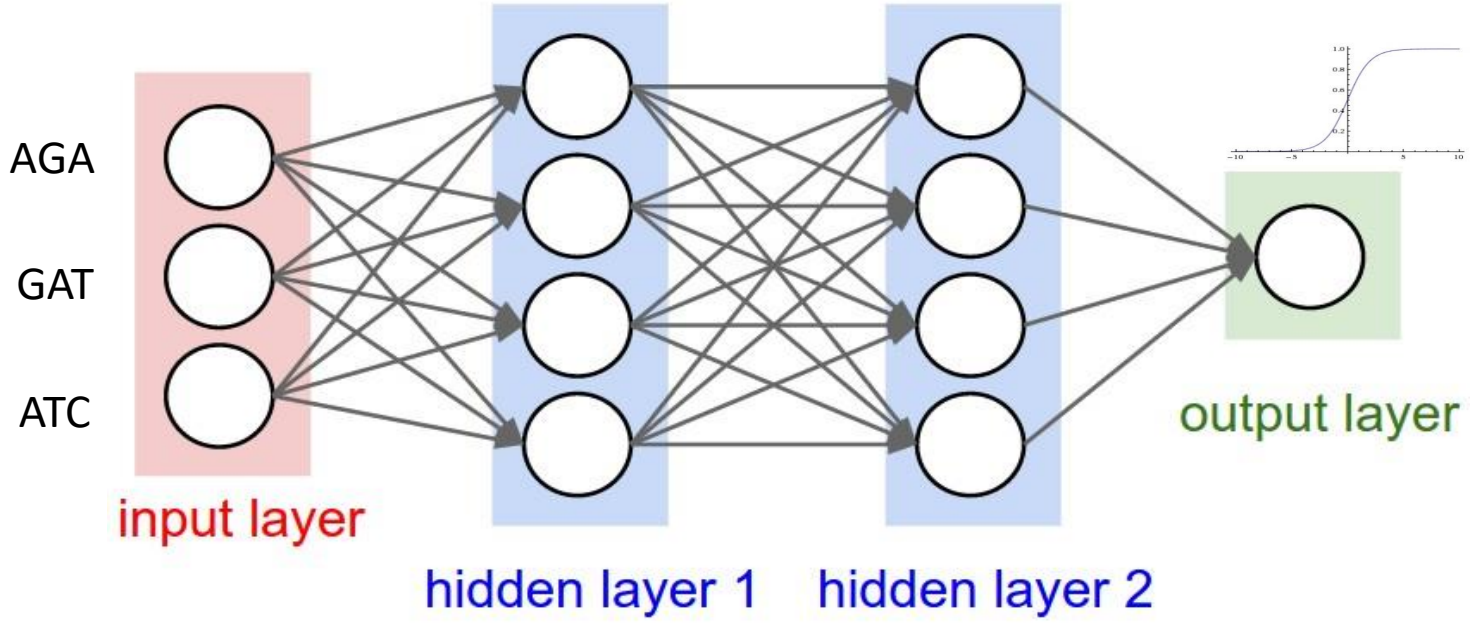
Denoising genome-wide histone ChIP-seq with convolutional neural networks.

Koh PW[1,2], Pierson E[1], Kundaje A[1,2].



How to train: SGD with backpropagation to compute gradients

Stochastic gradient descent



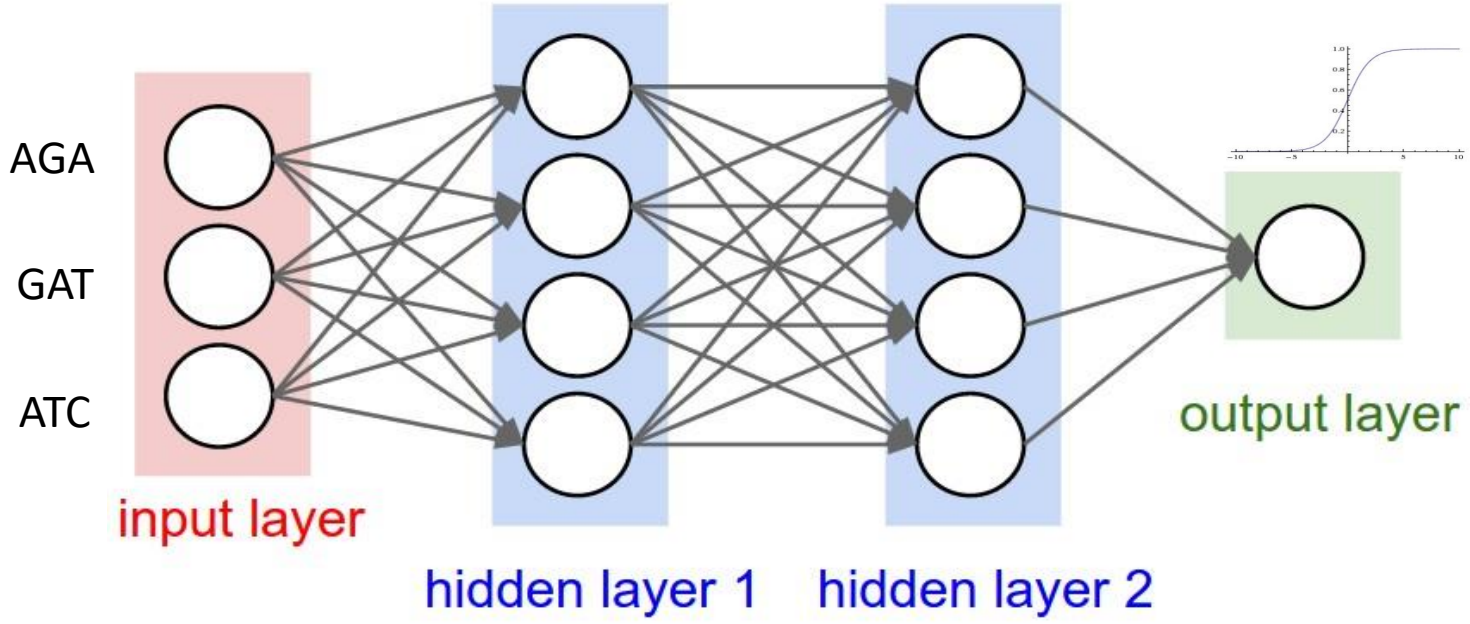
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

One pass of updates over all training examples = 1 Epoch

Run through multiple Epochs until ‘convergence’

Early stopping: Stop when validation set error stops decreasing

Stochastic gradient descent



$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

One pass of updates over all training examples = 1 Epoch

Run through multiple Epochs until ‘convergence’

Early stopping: Stop when validation set error stops decreasing

Challenge: How to compute gradients of loss wrt. parameters in a deep network

Ideas behind backpropagation: Chain rule of derivatives and reverse-mode differentiation

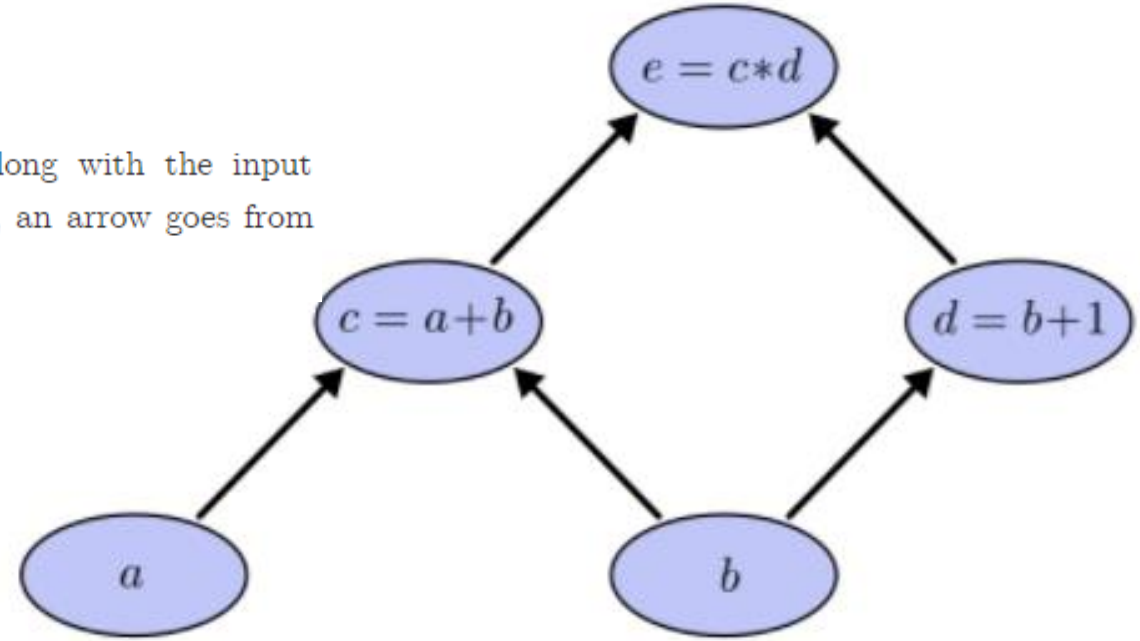
Computational graphs are a nice way to think about mathematical expressions. For example, consider the expression $e = (a + b) * (b + 1)$. There are three operations: two additions and one multiplication. To help us talk about this, let's introduce two intermediary variables, c and d so that every function's output has a variable. We now have:

$$c = a + b$$

$$d = b + 1$$

$$e = c * d$$

To create a computational graph, we make each of these operations, along with the input variables, into nodes. When one node's value is the input to another node, an arrow goes from one to another.



http://colah.github.io/posts/2015-08-Backprop/

Derivatives on Computational Graphs

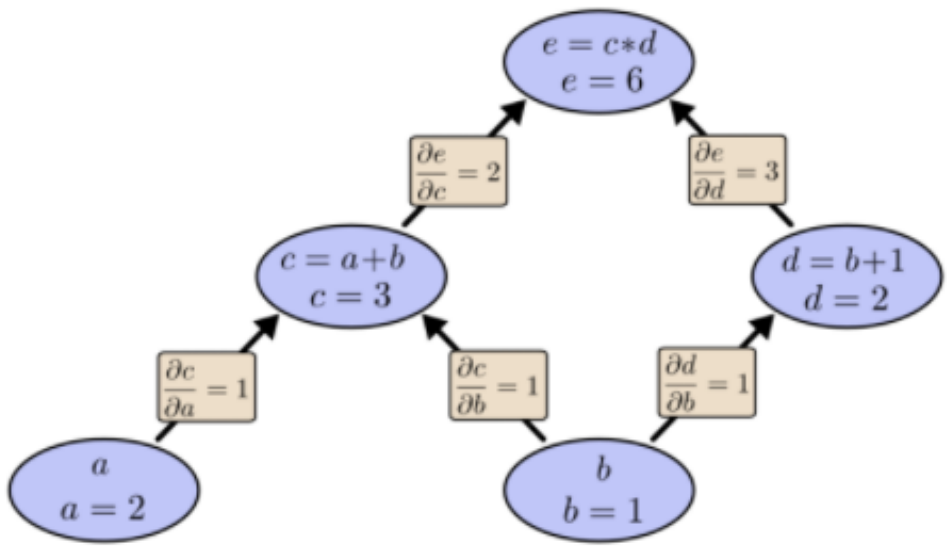
If one wants to understand derivatives in a computational graph, the key is to understand derivatives on the edges. If a directly affects c, then we want to know how it affects c. If a changes a little bit, how does c change? We call this the partial derivative of c with respect to a.

To evaluate the partial derivatives in this graph, we need the sum rule and the product rule:

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$

$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$

Below, the graph has the derivative on each edge labeled.



http://colah.github.io/posts/2015-08-Backprop/

Derivatives on Computational Graphs

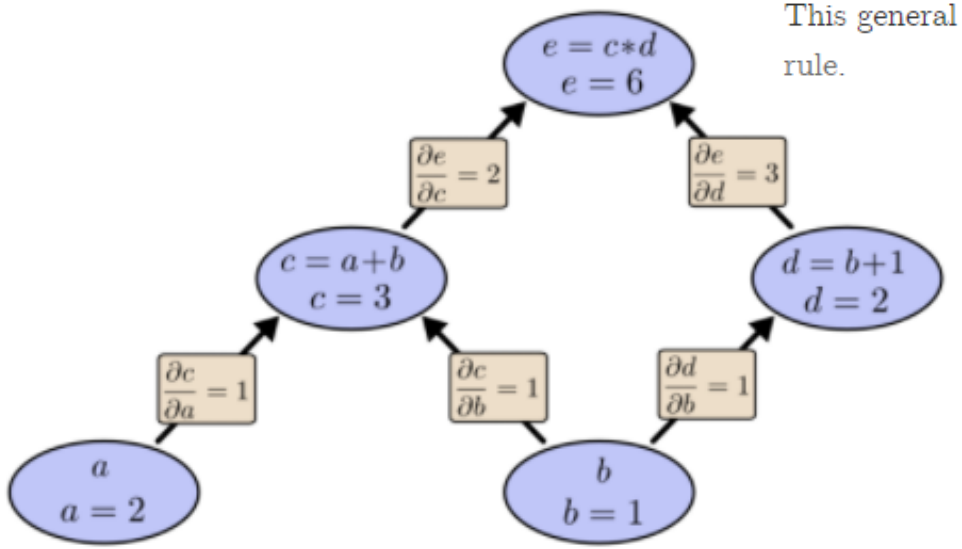
If one wants to understand derivatives in a computational graph, the key is to understand derivatives on the edges. If a directly affects c, then we want to know how it affects c. If a changes a little bit, how does c change? We call this the partial derivative of c with respect to a.

To evaluate the partial derivatives in this graph, we need the sum rule and the product rule:

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$

$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$

Below, the graph has the derivative on each edge labeled.



The general rule is to sum over all possible paths from one node to the other, multiplying the derivatives on each edge of the path together. For example, to get the derivative of e with respect to b we get:

$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3$$

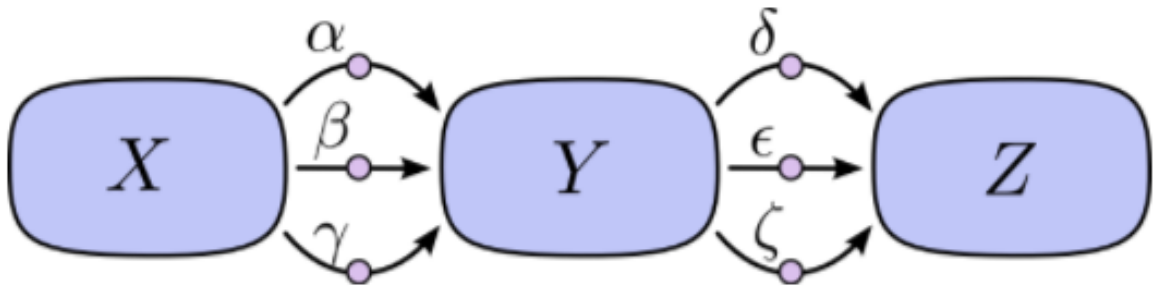
This accounts for how b affects e through c and also how it affects it through d.

This general "sum over paths" rule is just a different way of thinking about the multivariate chain rule.

http://colah.github.io/posts/2015-08-Backprop/

Factoring Paths

The problem with just "summing over the paths" is that it's very easy to get a combinatorial explosion in the number of possible paths.



In the above diagram, there are three paths from X to Y, and a further three paths from Y to Z. If we want to get the derivative $\frac{\partial Z}{\partial X}$ by summing over all paths, we need to sum over $3 * 3 = 9$ paths:

$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

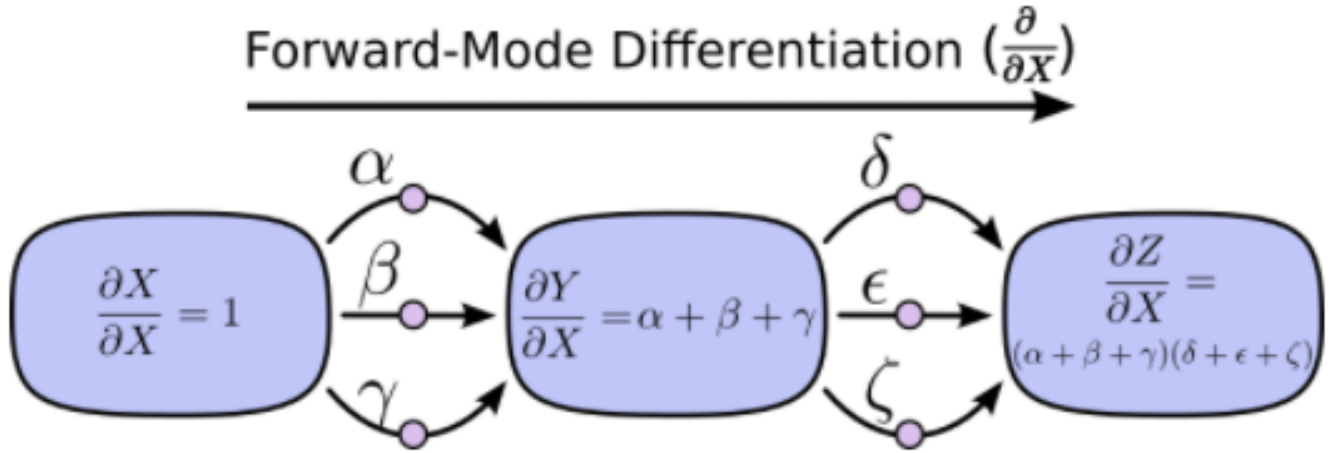
The above only has nine paths, but it would be easy to have the number of paths to grow exponentially as the graph becomes more complicated.

Instead of just naively summing over the paths, it would be much better to factor them:

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$$

This is where "forward-mode differentiation" and "reverse-mode differentiation" come in. They're algorithms for efficiently computing the sum by factoring the paths. Instead of summing over all of the paths explicitly, they compute the same sum more efficiently by merging paths back together at every node. In fact, both algorithms touch each edge exactly once!

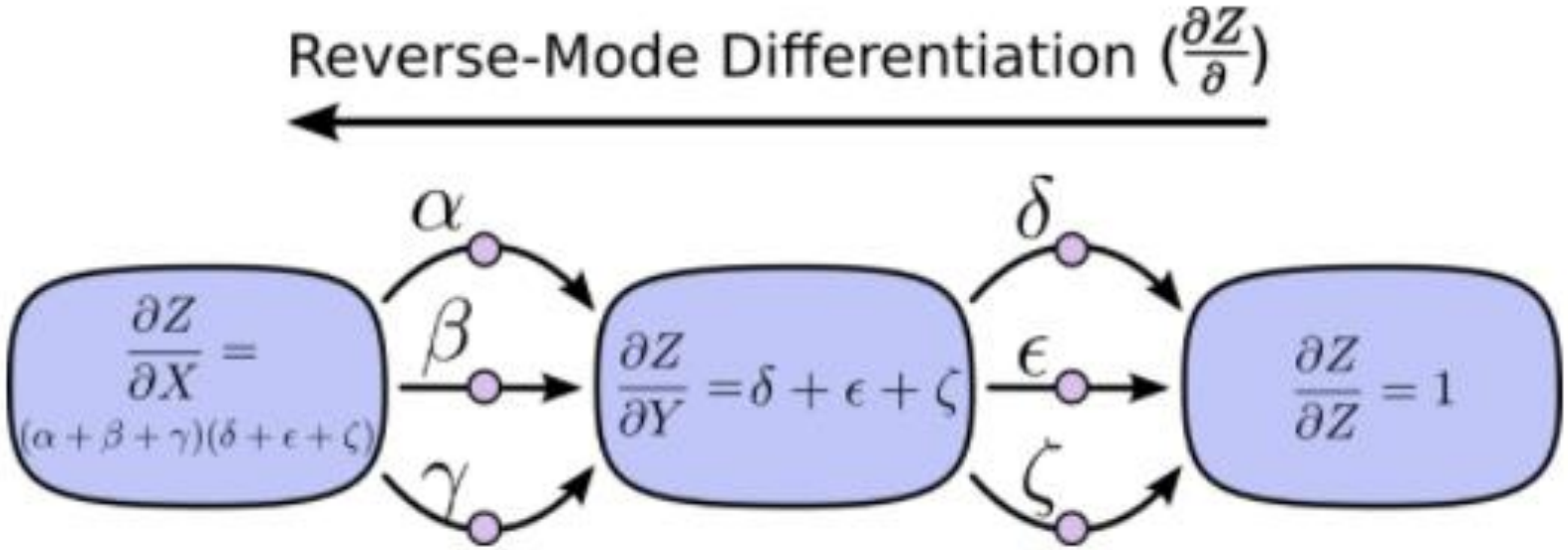
Forward-mode differentiation starts at an input to the graph and moves towards the end. At every node, it sums all the paths feeding in. Each of those paths represents one way in which the input affects that node. By adding them up, we get the total way in which the node is affected by the input, it's derivative.



Though you probably didn't think of it in terms of graphs, forward-mode differentiation is very similar to what you implicitly learned to do if you took an introduction to calculus class.

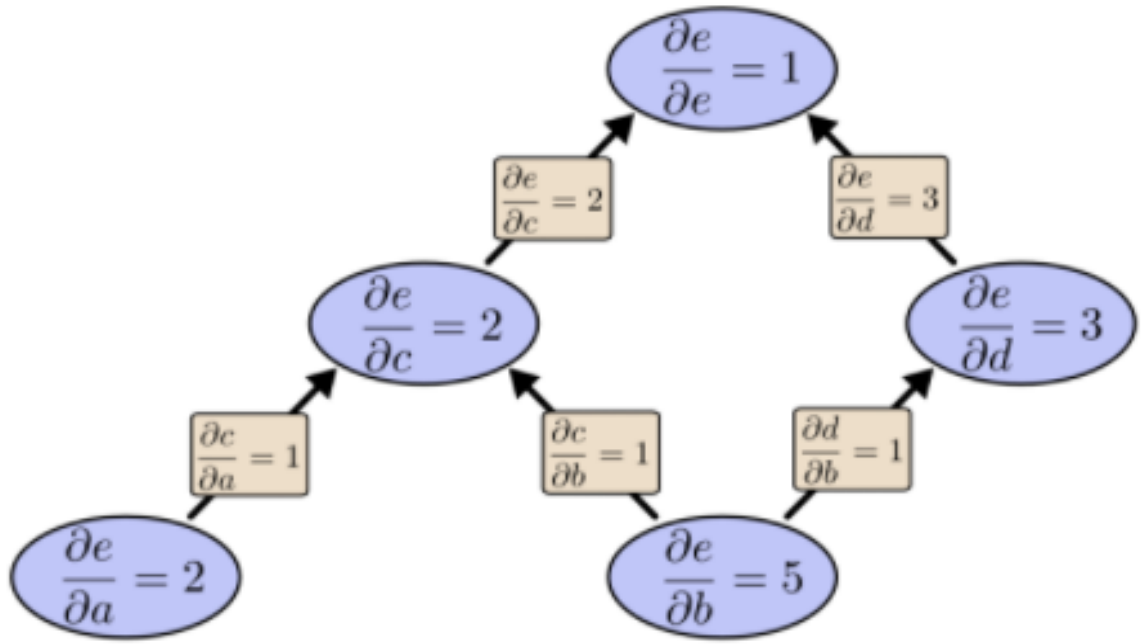
http://colah.github.io/posts/2015-08-Backprop/

Reverse-mode differentiation, on the other hand, starts at an output of the graph and moves towards the beginning. At each node, it merges all paths which originated at that node.



Forward-mode differentiation tracks how one input affects every node. Reverse-mode differentiation tracks how every node affects one output. That is, forward-mode differentiation applies the operator $\frac{\partial}{\partial X}$ to every node, while reverse mode differentiation applies the operator $\frac{\partial Z}{\partial}$ to every node.[1]

What if we do reverse-mode differentiation from e down? This gives us the derivative of e with respect to every node:



When I say that reverse-mode differentiation gives us the derivative of e with respect to every node, I really do mean *every node*. We get both $\frac{\partial e}{\partial a}$ and $\frac{\partial e}{\partial b}$, the derivatives of e with respect to both inputs. Forward-mode differentiation gave us the derivative of our output with respect to a single input, but reverse-mode differentiation gives us all of them.

http://colah.github.io/posts/2015-08-Backprop/

When training neural networks, we think of the cost (a value describing how bad a neural network performs) as a function of the parameters (numbers describing how the network behaves). We want to calculate the derivatives of the cost with respect to all the parameters, for use in gradient descent. Now, there's often millions, or even tens of millions of parameters in a neural network. So, reverse-mode differentiation, called backpropagation in the context of neural networks, gives us a massive speed up!

http://colah.github.io/posts/2015-08-Backprop/