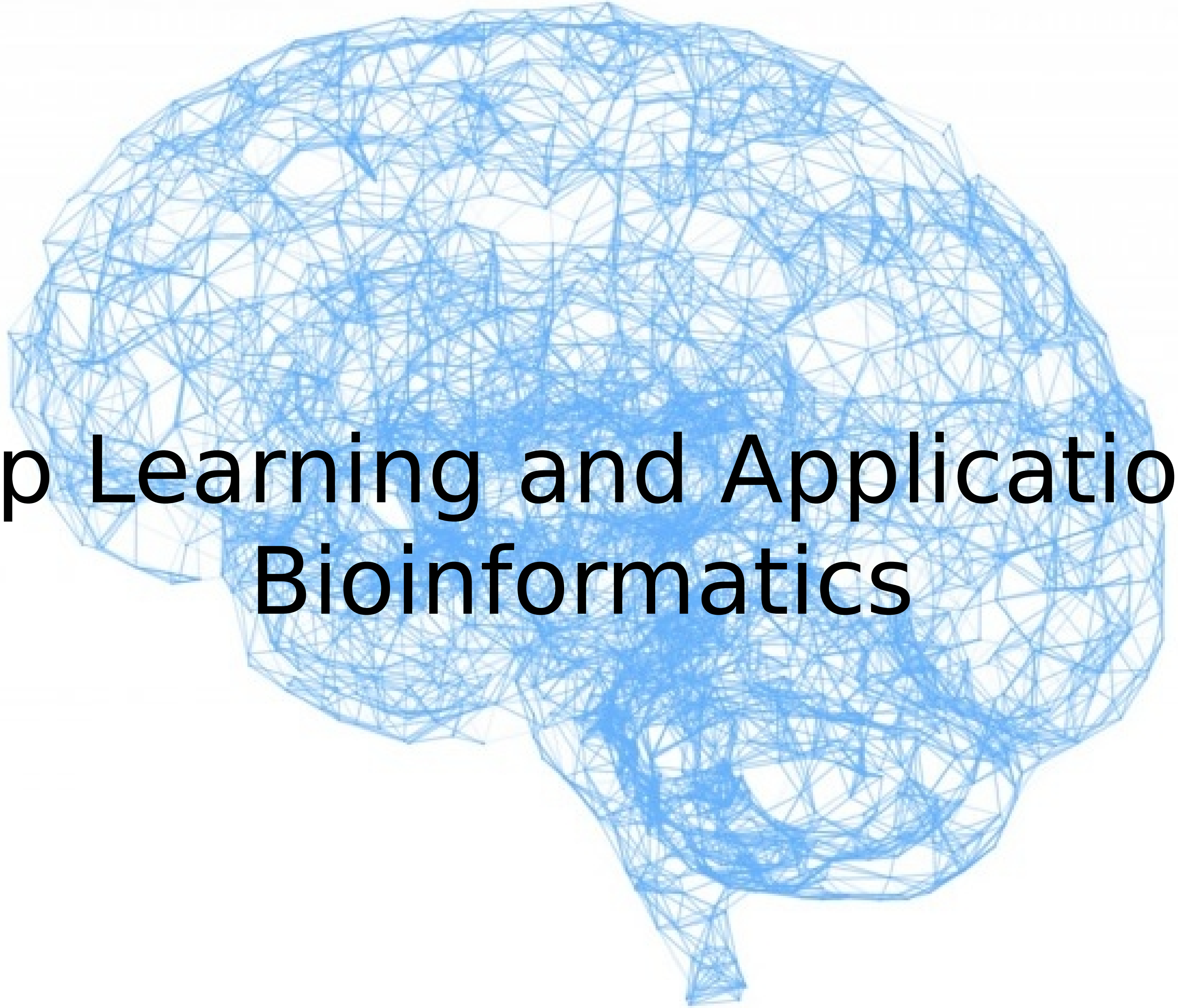
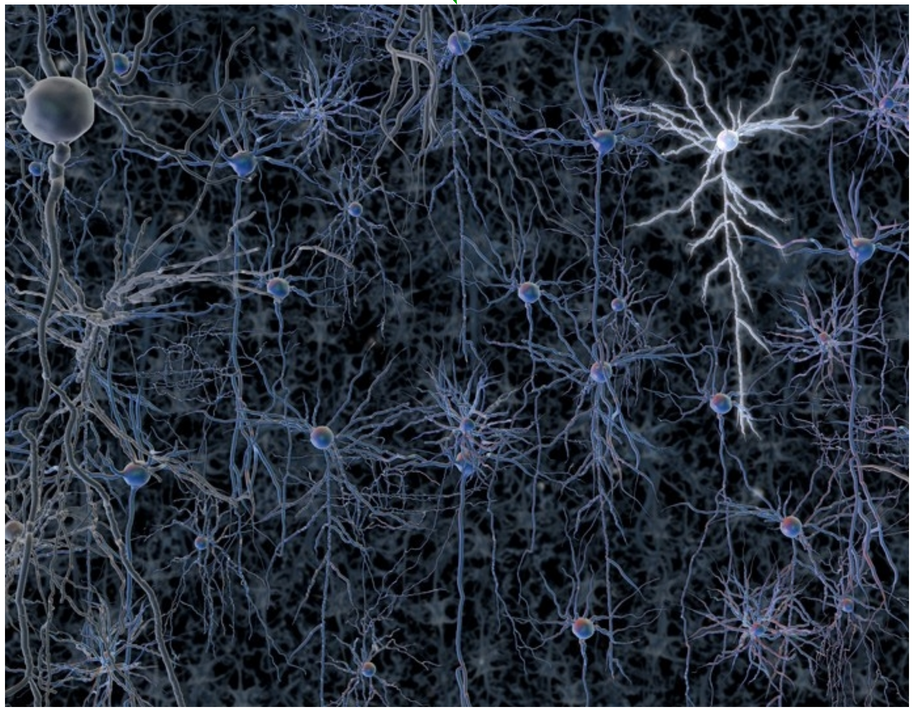


Deep Learning and Application in Bioinformatics

Neural Network

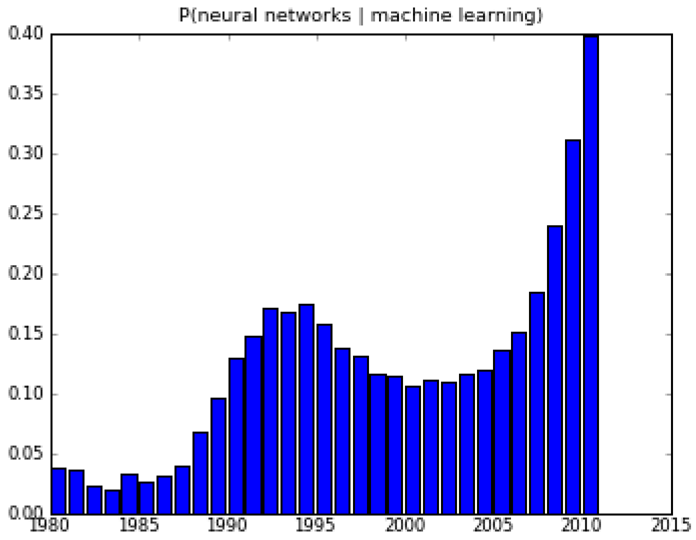
Human brain is the most sophisticated intelligence system so far.
Can we create algorithms to model the brain neural network?

Picture by me and Google AutoDraw



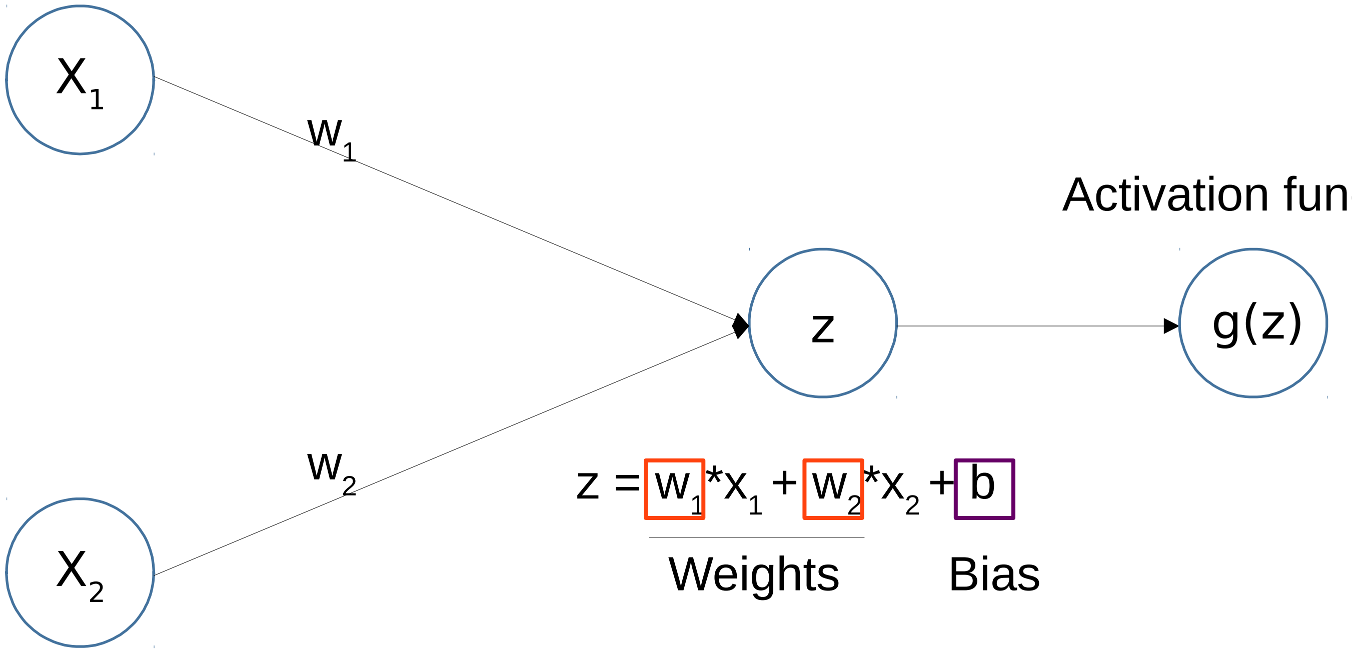
Neural Network

- Invented to mirror the function of the brain.
- Two resurgences:
  - 1980's: development of backpropagation
  - 2000's:
    - Improved design: CNN, RNN, GAN, ...
    - Techniques of training: unsupervised pre-training...
    - Increased computing power: GPU computation
    - Big Data
- Getting a fancy name: Deep Learning

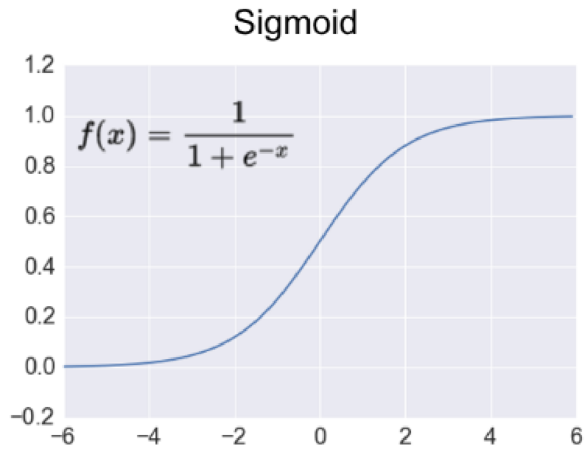


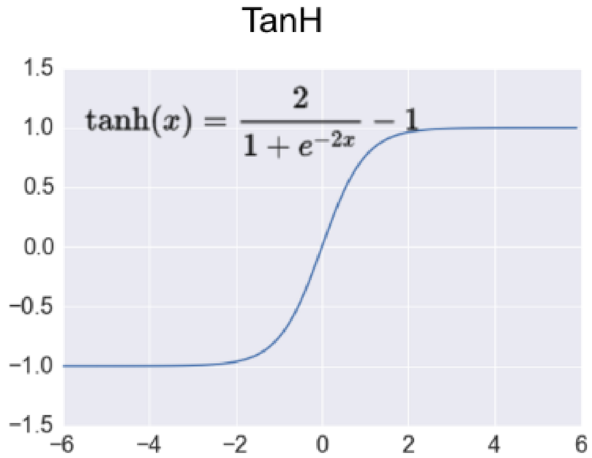
A series of techniques to construct neural networks and to facilitate their learning processes.

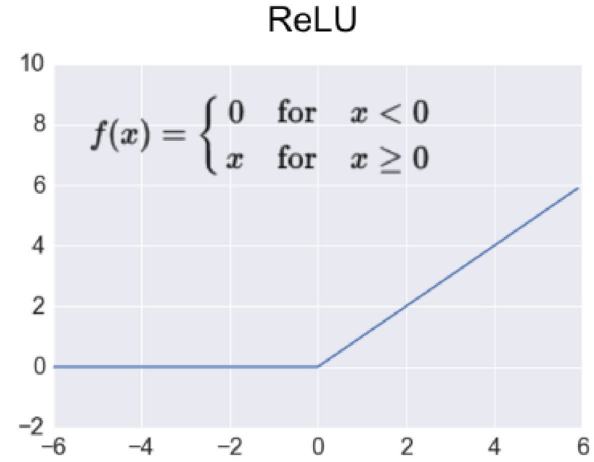
Neuron Model



g(z) is any form of an activation function

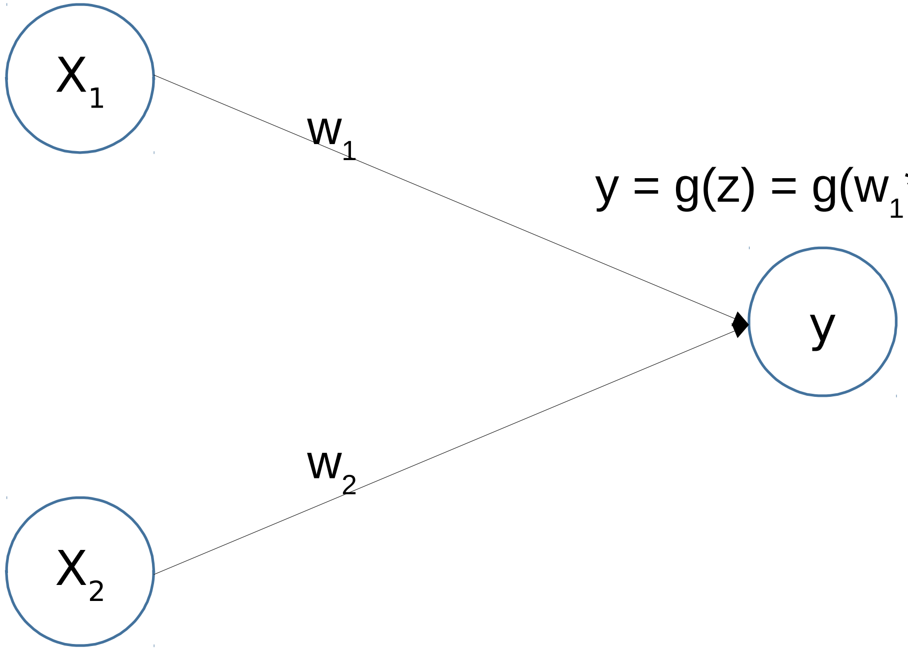


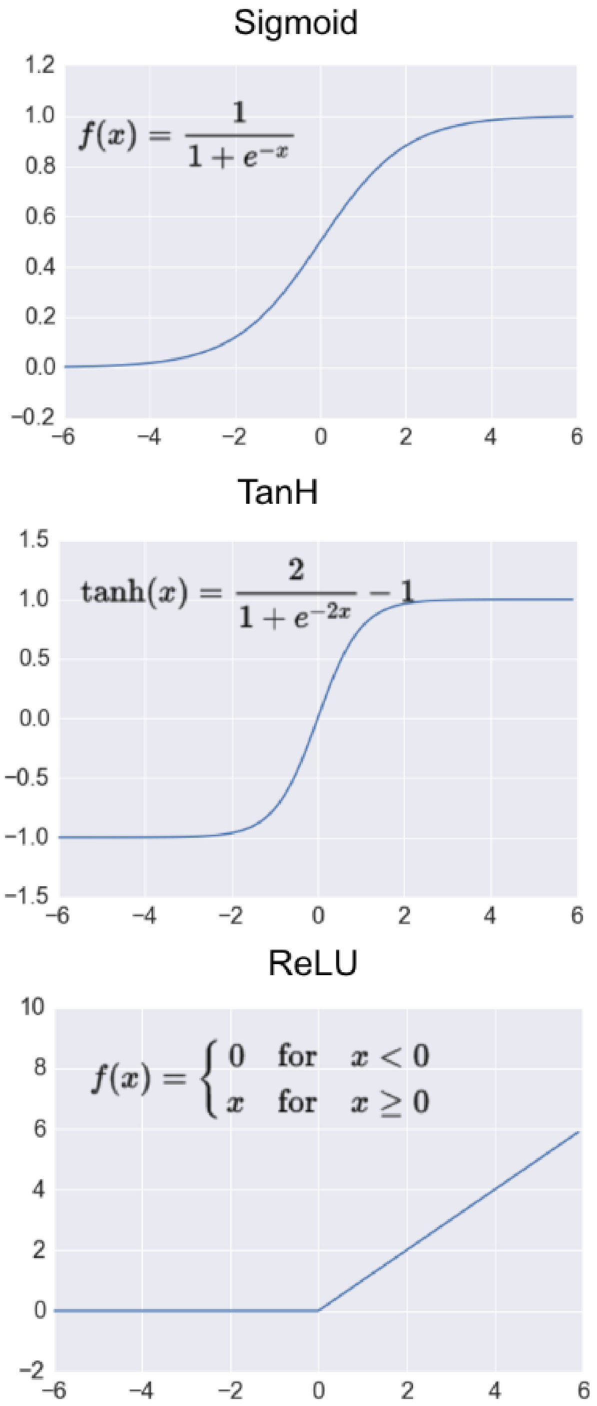




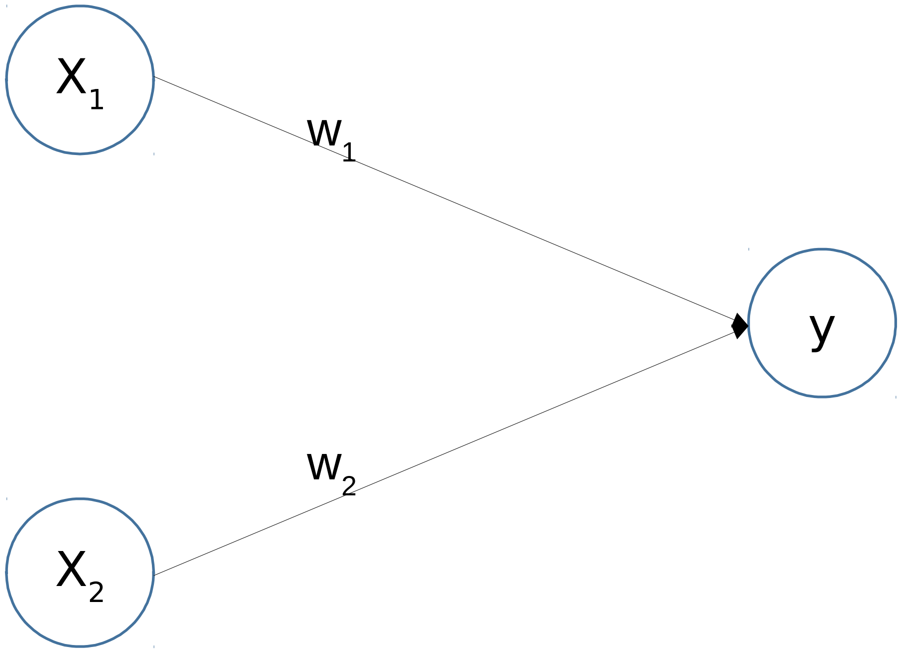
Neuron Model

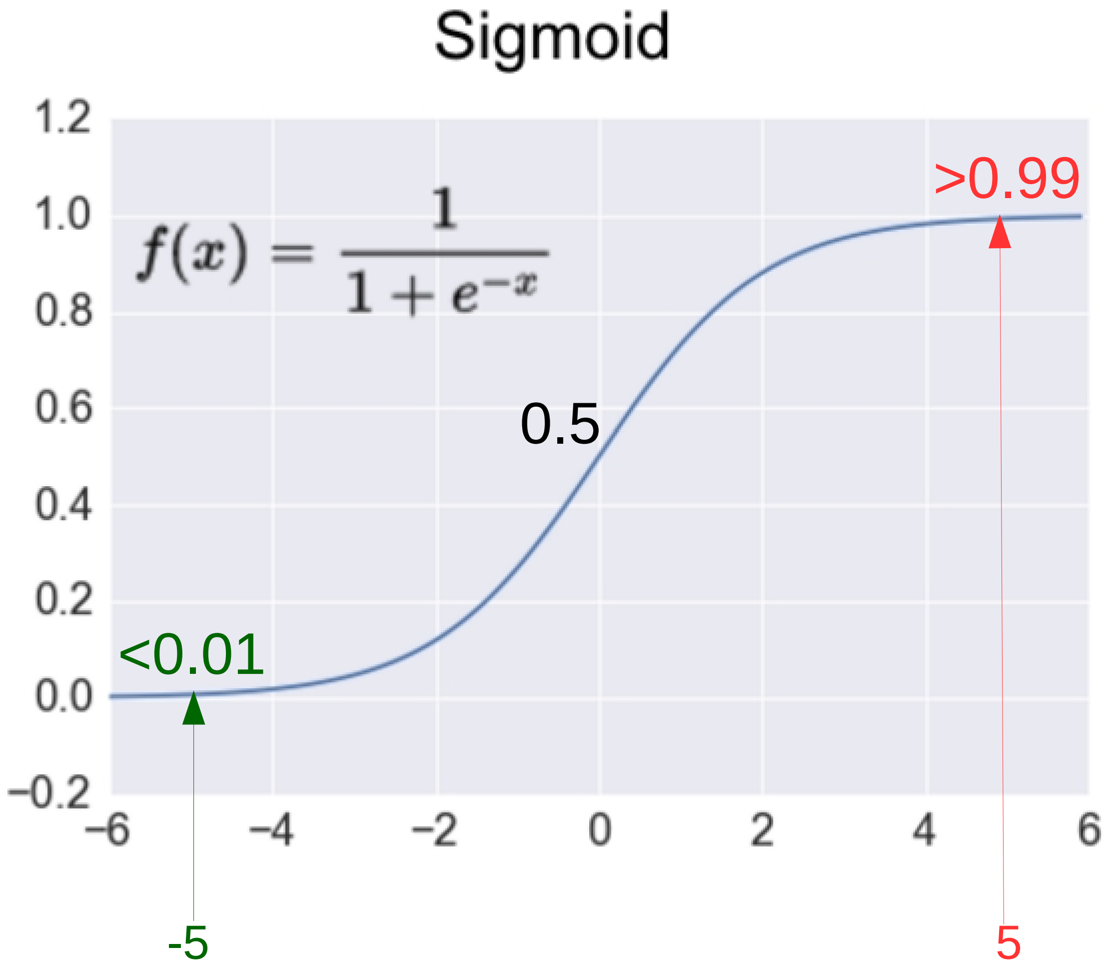
$$y = g(z) = g(w_1{*}x_1 + w_2{*}x_2 + b)$$





Neuron Model

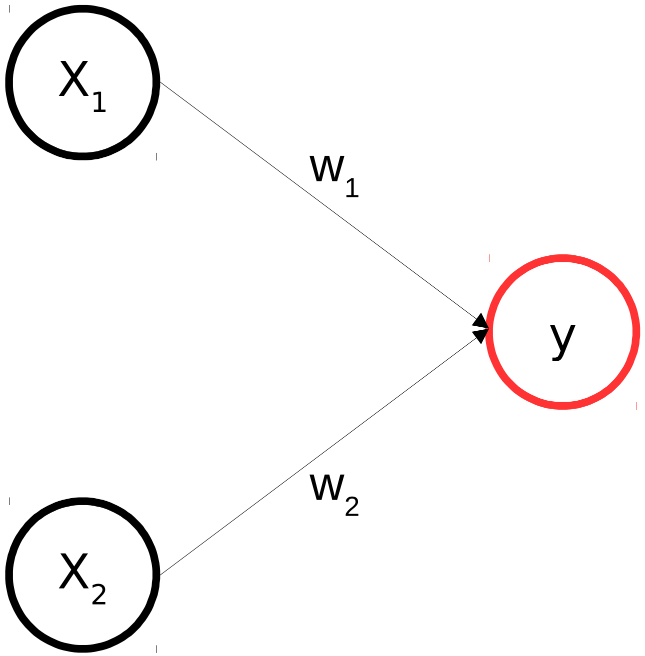




Neuron Model: And Logic

$x_1 = 0 \text{ or } 1,\ x_2 = 0 \text{ or } 1$

$$y \begin{cases} =1, & \text{if } x_1 = x_2 = 1 \\ =0, & \text{otherwise} \end{cases}$$



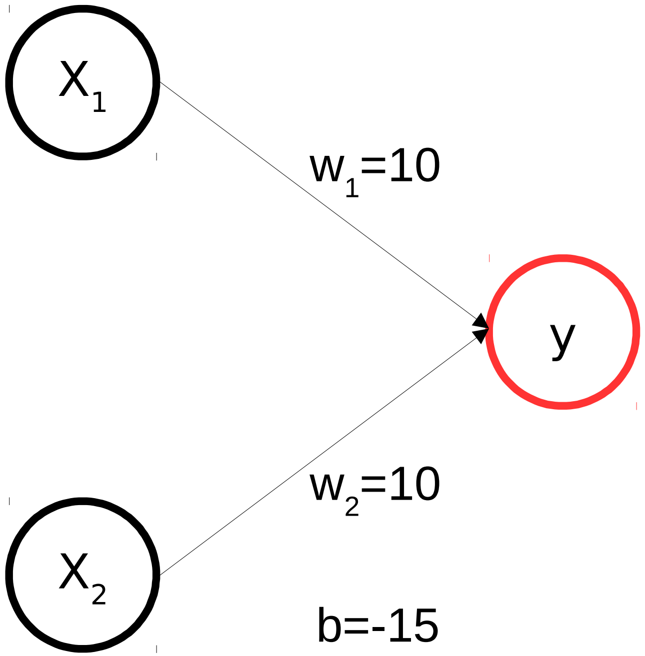
$y = \text{Sigmoid}(w_1{*}x_1 + w_2{*}x_2 + b)$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

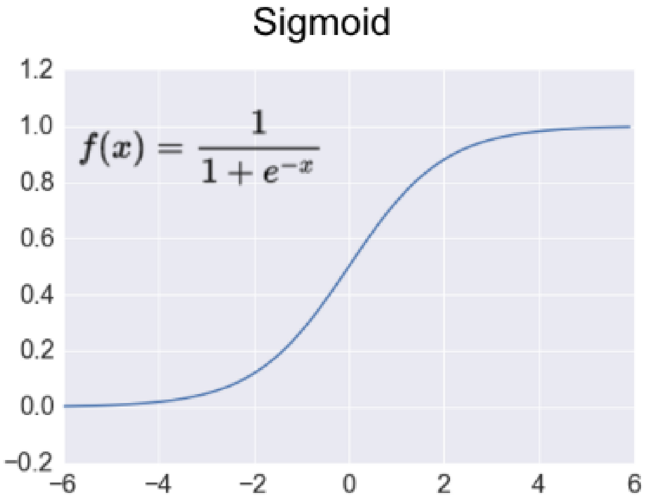
Neuron Model: And Logic

$x_1 = 0 \text{ or } 1,\ x_2 = 0 \text{ or } 1$

$$y = \begin{cases} =1, & \text{if } x_1 = x_2 = 1 \\ =0, & \text{otherwise} \end{cases}$$



$$y = \text{Sigmoid}(w_1 * x_1 + w_2 * x_2 + b)$$

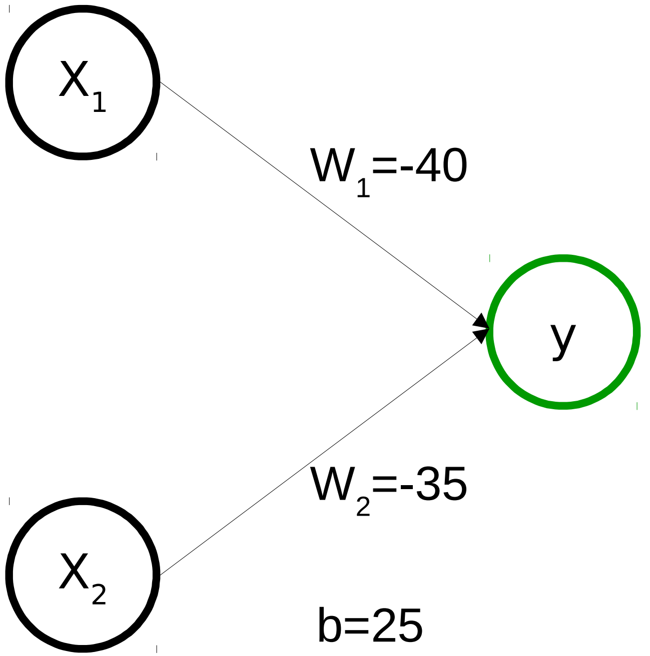


x_1	x_2	Wx+b	y
0	0	-15	0
0	1	-5	0
1	0	-5	0
1	1	5	1

Neuron Model: NOR Logic

$x_1 = 0$ or 1, $x_2 = 0$ or 1

$$y \begin{cases} =1, \text{ if } x_1 = x_2 = 0 \\ =0, \text{ otherwise} \end{cases}$$



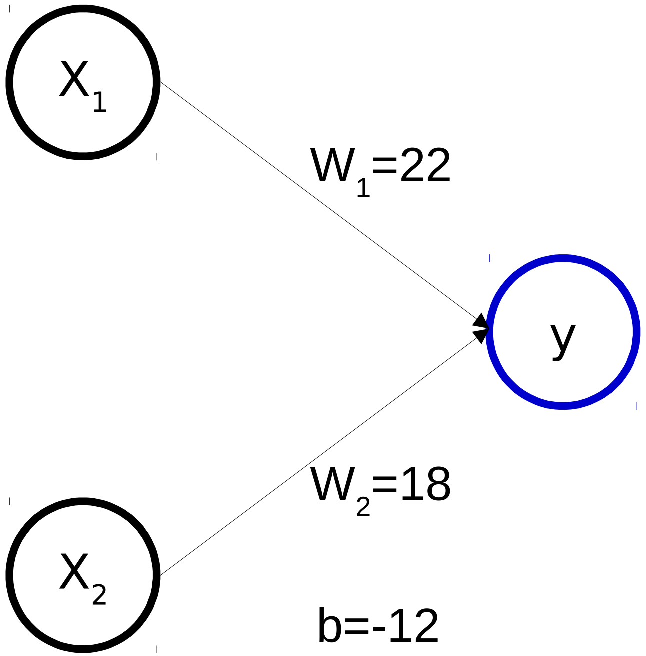
$$y = \mathrm{Sigmoid}(w_1 * x_1 + w_2 * x_2 + b)$$

x_1	x_2	Wx+b	y
0	0	25	1
0	1	-10	0
1	0	-15	0
1	1	-50	0

Neuron Model: OR Logic

x_1 = 0 or 1, x_2 = 0 or 1

$$y \begin{cases} =0, \text{ if } x_1 = x_2 = 0 \\ =1, \text{ otherwise} \end{cases}$$



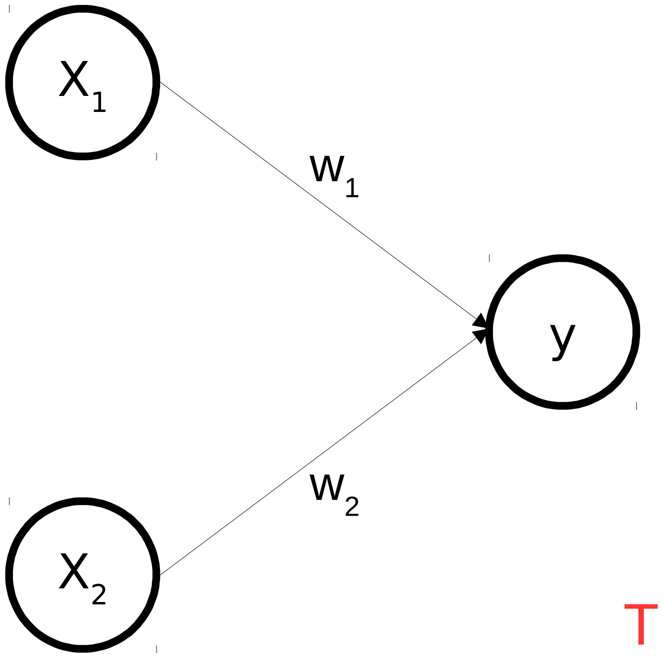
y = Sigmoid(w_1*x_1 + w_2*x_2 + b)

x_1	x_2	Wx+b	y
0	0	-12	0
0	1	6	1
1	0	10	1
1	1	28	1

Neuron Model: XNOR Logic

x_1 = 0 or 1, x_2 = 0 or 1

$$y = \begin{cases} =1, \text{ if } x_1 = x_2 = 0 \text{ or } x_1 = x_2 = 1 \\ =0, \text{ otherwise} \end{cases}$$



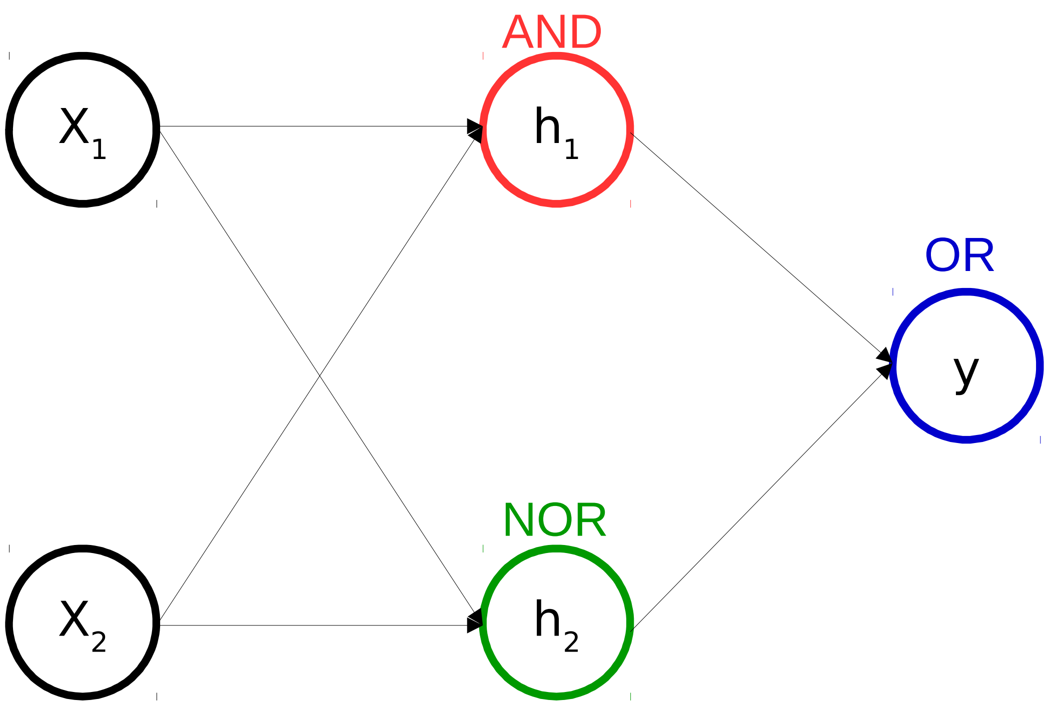
y = Sigmoid(w_1*x_1 + w_2*x_2 + b)

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

This is impossible with one single neuron!

Neural Network: XNOR Logic

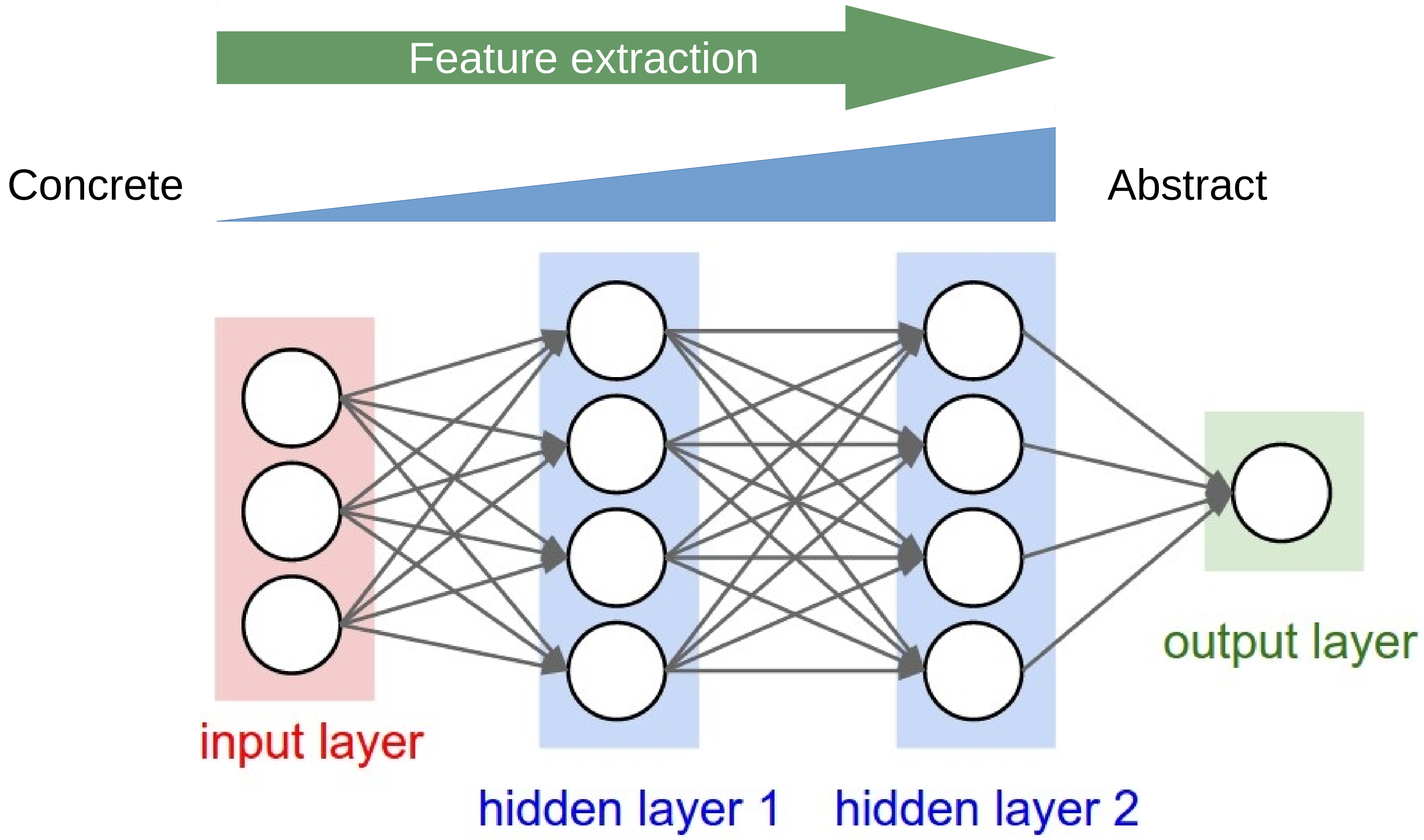
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1



x_1	x_2	h_1	h_2	y
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Neural networks could approximate complex functions by adding hidden layers.
Universal approximation theorem: a NN could approximate any function with one layer and finite parameters.

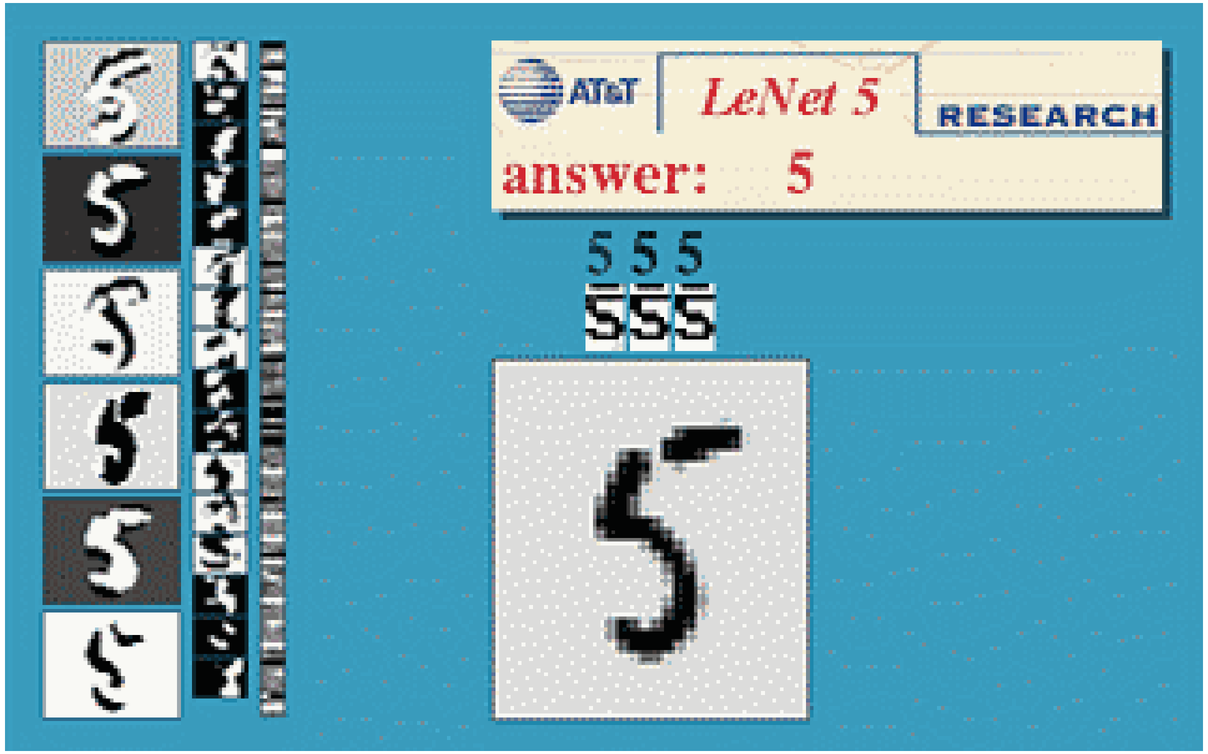
Neural Network: Hidden Layers



Example of a feedforward neural network

Neural Network: Hidden Layers

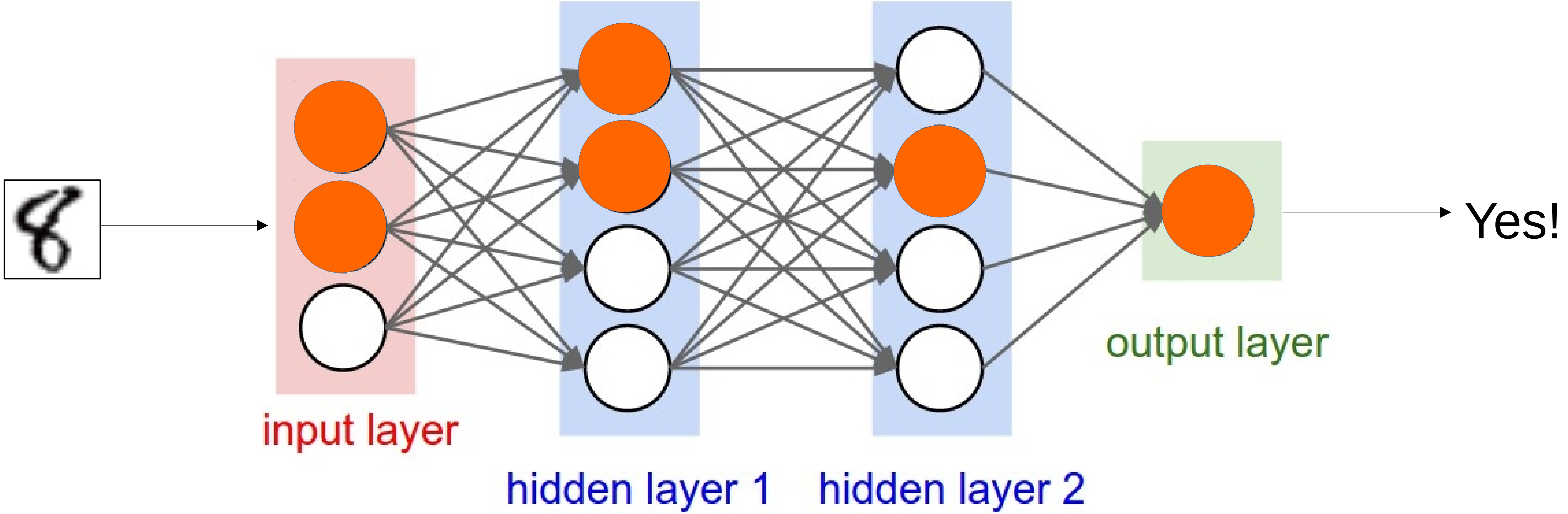
Hidden layers are usually hard to explain.



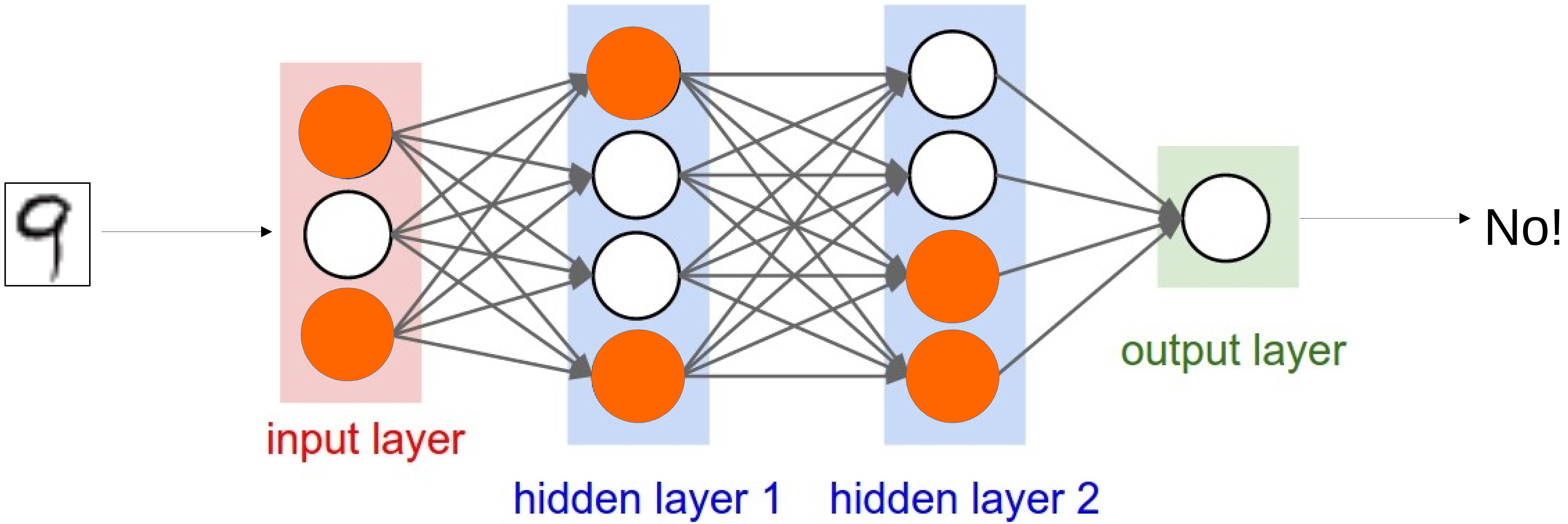
Yann Lecun, Facebook AI research, father of the convolutional neural network (CNN)

Neural Network: Hidden Layers

Example: “Is this an 8?”

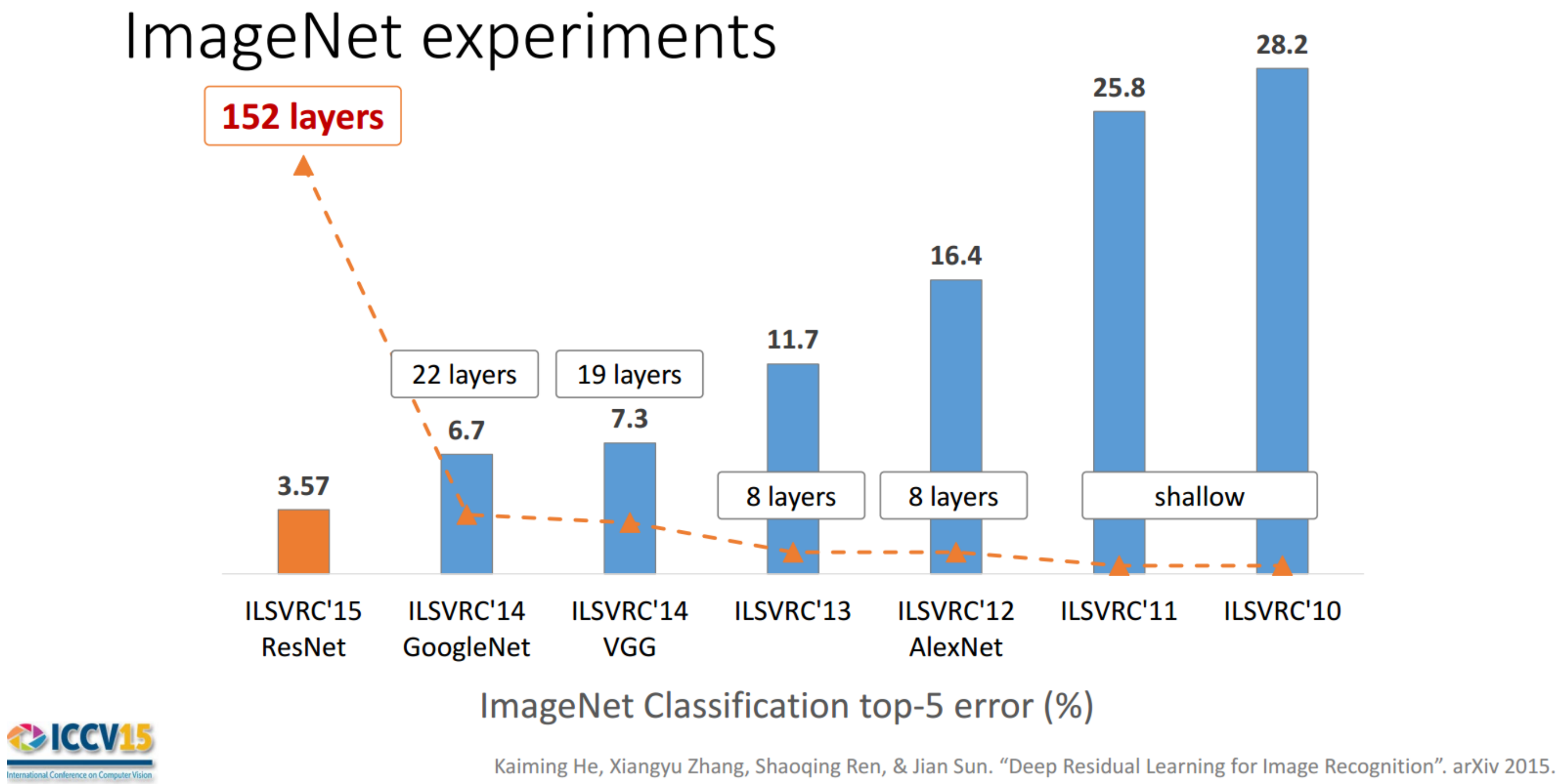


Neural Network: Hidden Layers

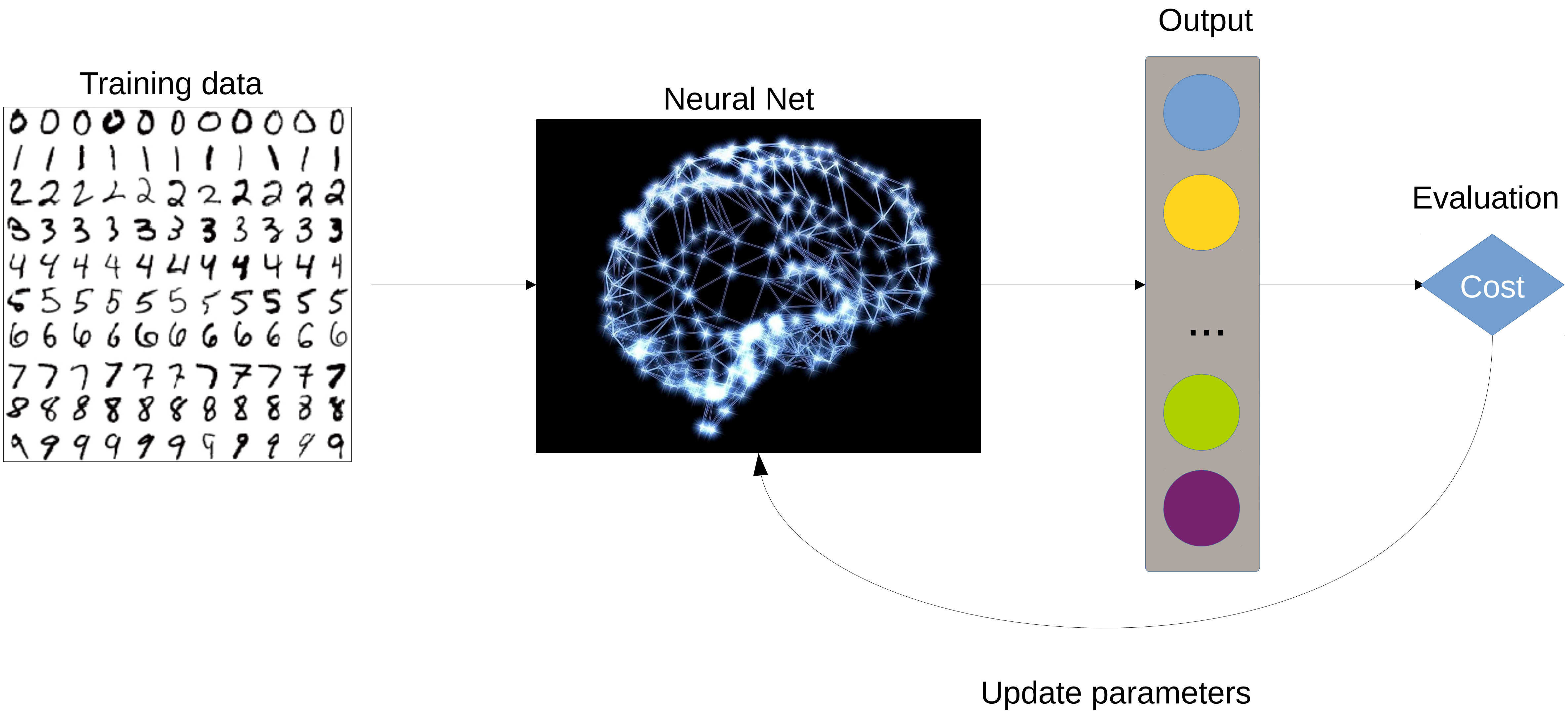


Neural Network: Hidden Layers

The deeper, the better? How deep is “deep”?



Training NN: How Does A NN Learn?



Training NN: Cost Function

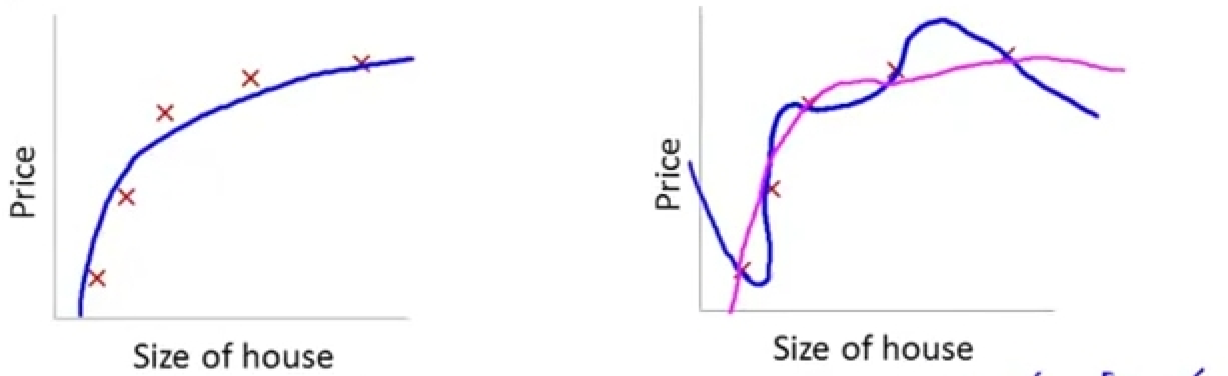
Cost of classification models

- Binary
  - One sample: $-[y^{(i)} * log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * log(1 - h_\theta(x^{(i)}))]$
  - Many samples: $-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} * log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * log(1 - h_\theta(x^{(i)})]$
  - Regularization term: $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

Training NN: Cost Function

Why regularization?

Intuition



$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$

Suppose we penalize and make θ_3, θ_4 really small.

$$\rightarrow \quad \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\, \theta_3^2 + 1000\, \theta_4^2$$

Slides from Andrew Ng

Training NN: Cost Function

Cost of classification models

- Binary

$$J(\theta) = \boxed{-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} * log(h_\theta(x^{(i)})) + (1 - y^{(i)}) * log(1 - h_\theta(x^{(i)}))]} + \boxed{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

Loss of incorrect predictions
Making your model more accurate

Loss of model complexity
Prevent overfitting

Training NN: Cost Function

Cost of classification models

- Multi-class classification

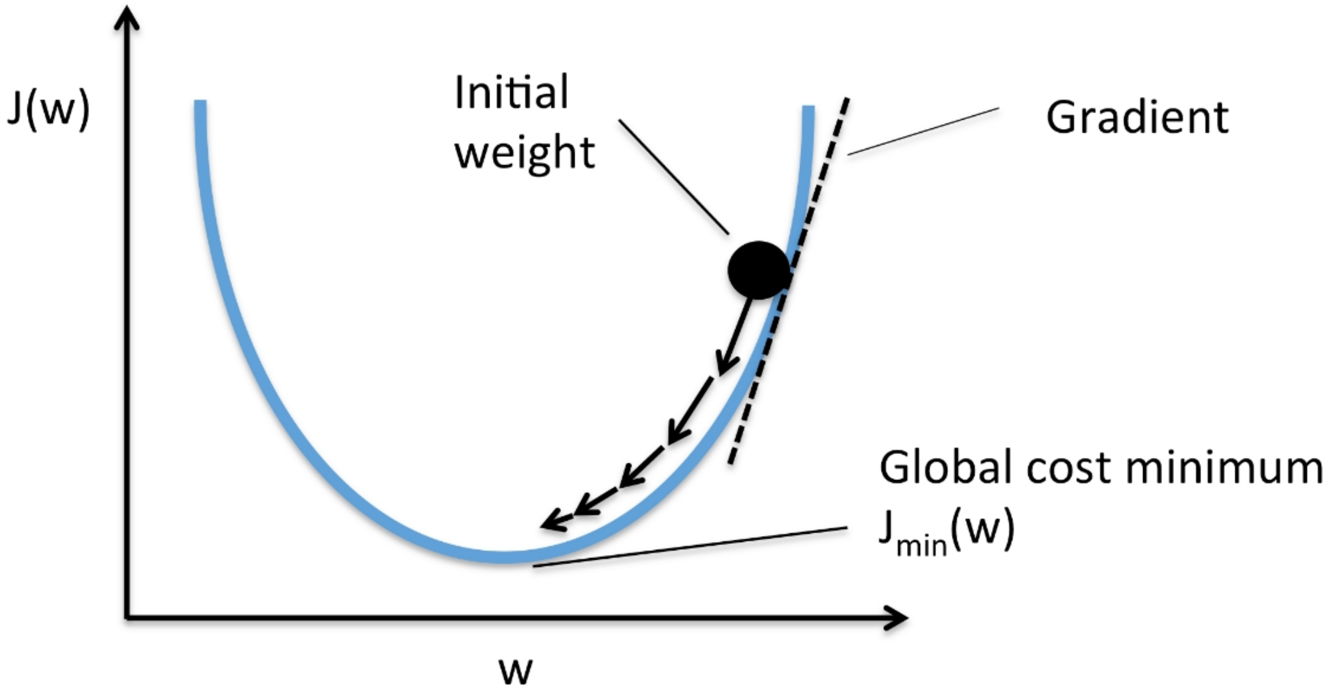
$$J(\theta) = \boxed{-\frac{1}{m}\sum_{k=1}^{K}\sum_{i=1}^{m}[y_k^{(i)} * log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) * log(1-h_\theta(x^{(i)}))_k]} + \boxed{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

Loss of incorrect predictions
Making your model more accurate

Loss of model complexity
Prevent overfitting

Training NN: Gradient Descent

Idea: minimize cost function
J(w) decreases fastest when w moves the direction of negative gradient



$$w^{r+1} = w^r - \alpha \frac{\partial}{\partial w} J(w)$$

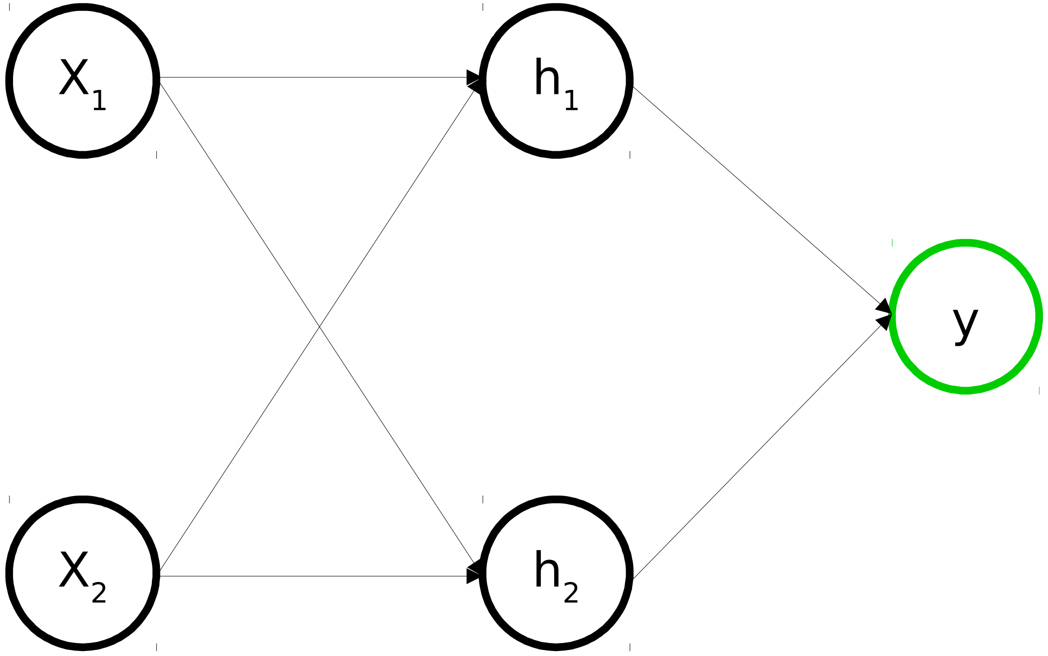
Updated w

Old w

Learning rate

Training NN: Backpropagation

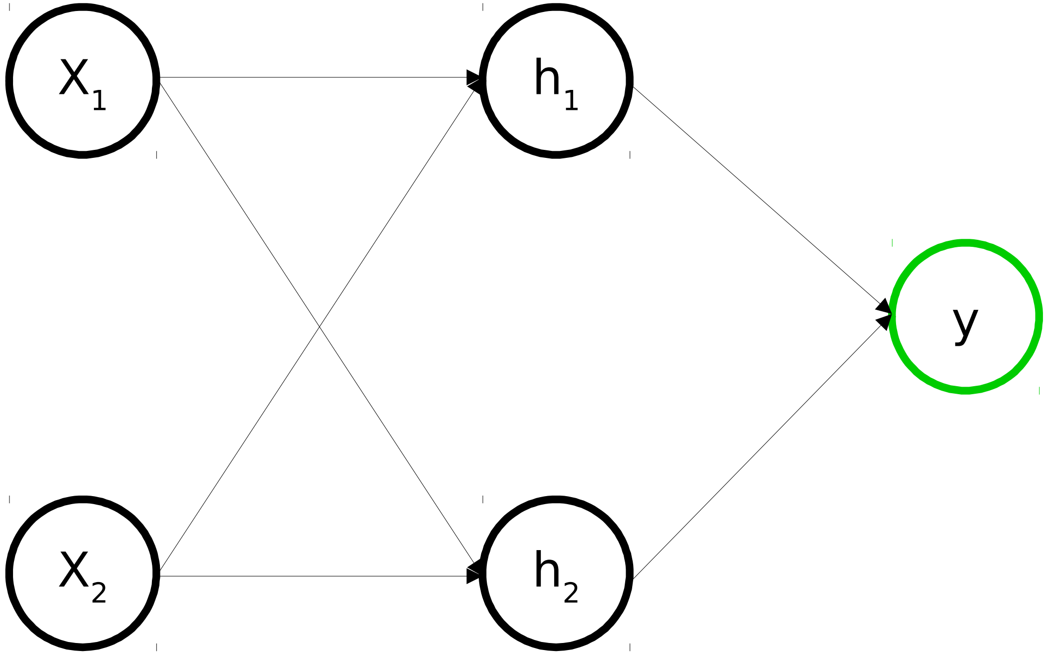
With multiple hidden layers, it's hard to get an analytic form of a neural net, let alone its gradient. Backpropagation is an approach to estimating gradient numerically.



Step 1:
Forward propagation

Training NN: Backpropagation

With multiple hidden layers, it's hard to get an analytic form of a neural net, let alone its gradient. Backpropagation is an approach to estimating gradient numerically.

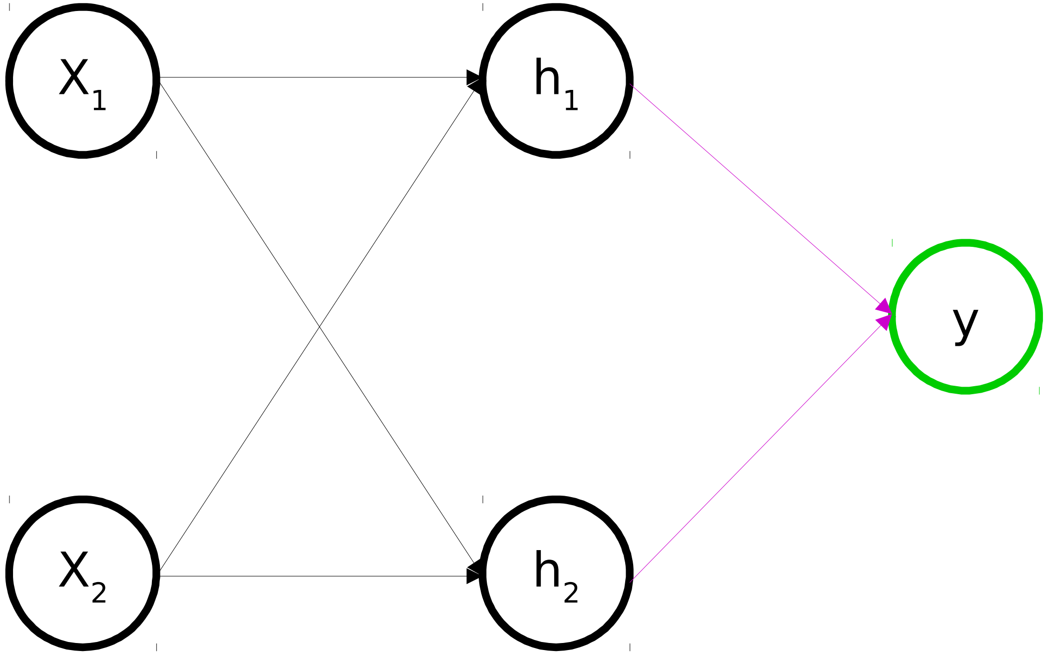


Step 2: Calculate error of y

$$\Delta y = y_{truth} - y_{predition}$$

Training NN: Backpropagation

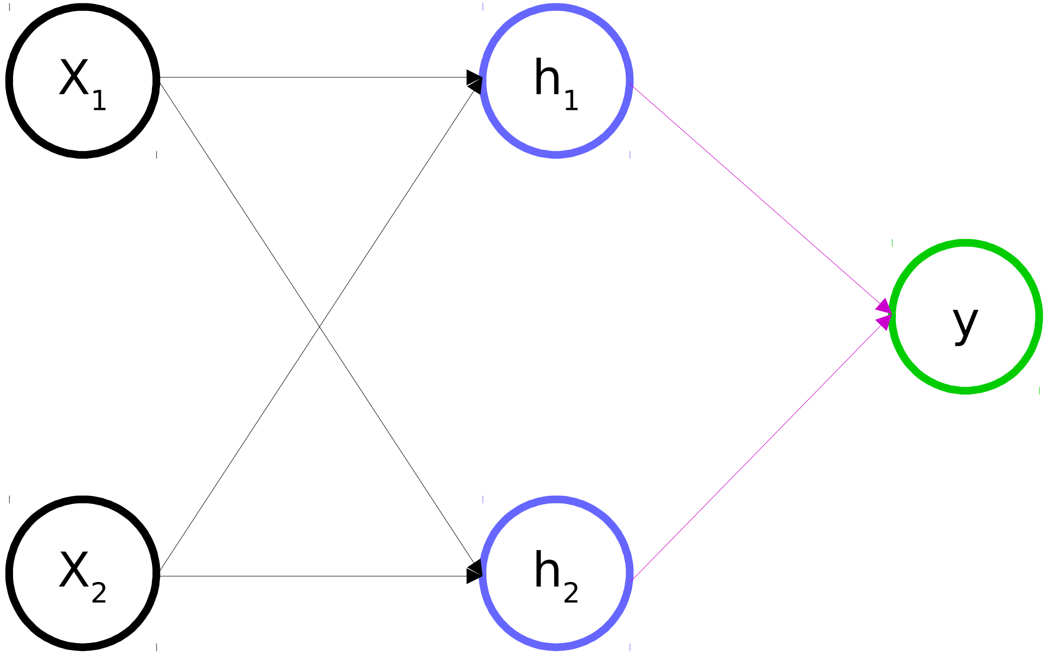
With multiple hidden layers, it's hard to get an analytic form of a neural net, let alone its gradient. Backpropagation is an approach to estimating gradient numerically.



Step 3:
Calculate gradients of edges connected to y

Training NN: Backpropagation

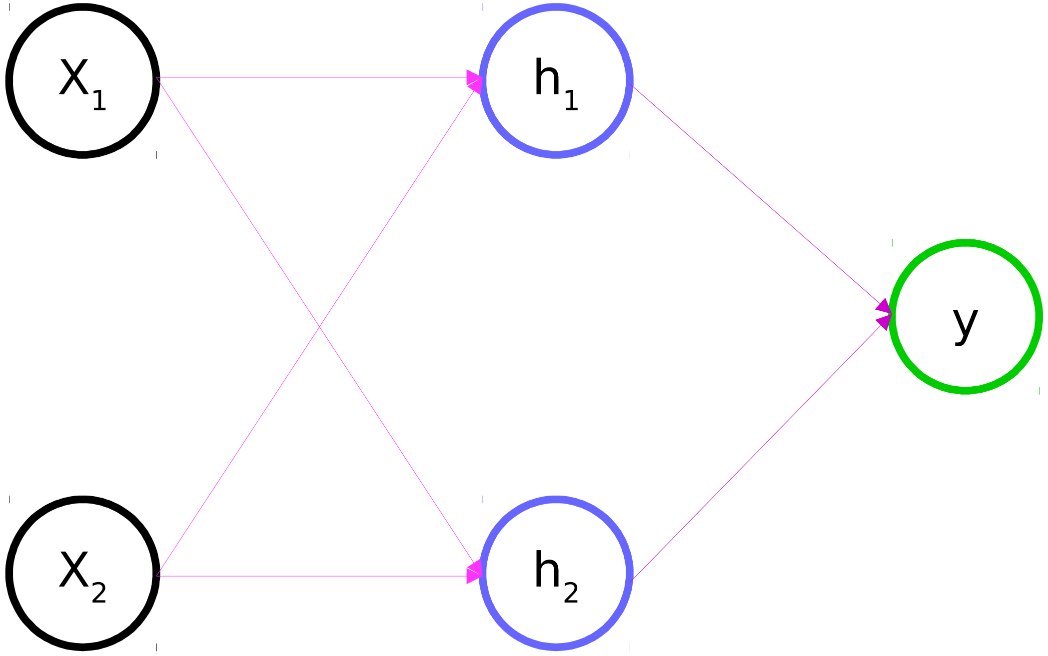
With multiple hidden layers, it's hard to get an analytic form of a neural net, let alone its gradient. Backpropagation is an approach to estimating gradient numerically.



Step 4:
Calculate errors of hidden units

Training NN: Backpropagation

With multiple hidden layers, it's hard to get an analytic form of a neural net, let alone its gradient. Backpropagation is an approach to estimating gradient numerically.



Step 5:
Calculate gradients of edges connected to the hidden layer

Training NN: A Bag of Tricks (Geoffrey Hinton)

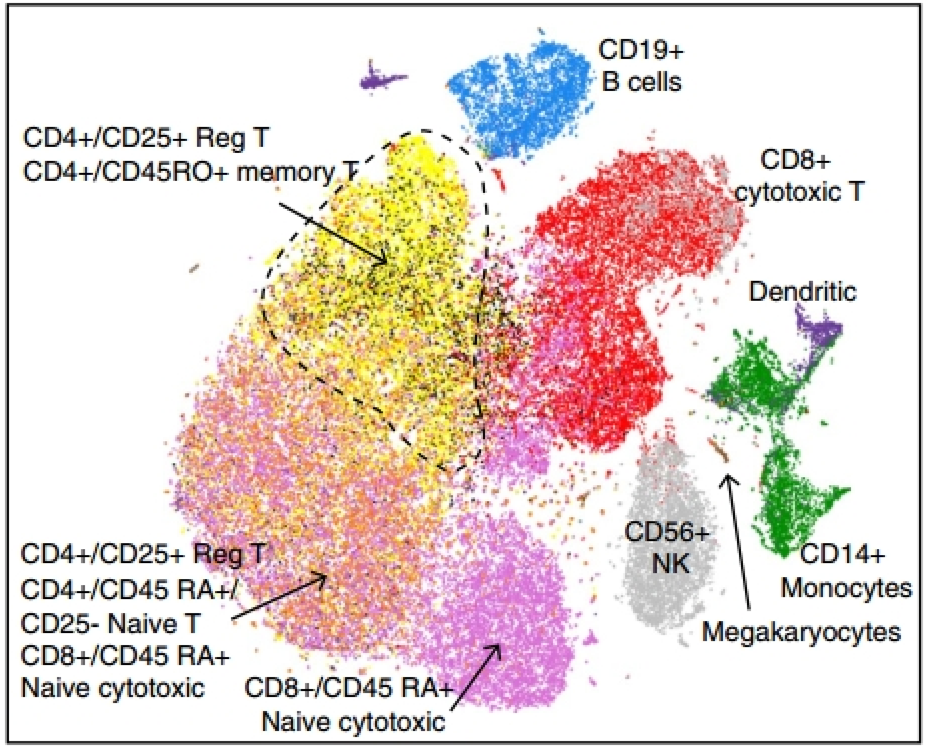
- Unsupervised pre-training: better initial parameters
- Momentum method: more efficient updates
- Batch normalization: prevent gradient vanishing/explosion
- Stochastic gradient descent: dealing with large dataset
- Dropout: prevent overfitting
- Early termination: prevent overfitting
- ……

Python libraries for implementation

TensorFlow

Example: celltype predictor

Sinlge-cell RNA-seq data from 10xGenomics
PBMC sample from healthy donors



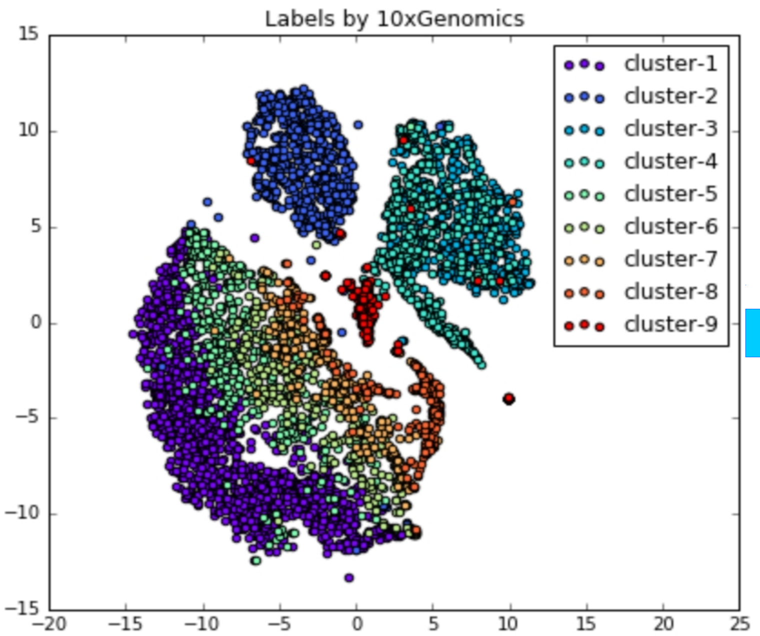

```python
tf.reset_default_graph()
norm = True
n_feature = train_x.shape[1]
xs = tf.placeholder(tf.float32, [None, n_feature])
ys = tf.placeholder(tf.float32, [None, 9])
kp_1 = tf.placeholder(tf.float32) # keep_prob
kp_2 = tf.placeholder(tf.float32) # keep_prob

# Fully-connected layer 1
W_fc1 = weight_variables([n_feature, 200])
b_fc1 = bias_variables([200])
h_fc1 = tf.nn.relu(tf.matmul(xs, W_fc1)+b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, kp_1)
if norm:
    fc_mean, fc_var = tf.nn.moments(
        h_fc1_drop,
        axes=[0],
    )
    scale = tf.Variable(tf.ones([n_feature]))
    shift = tf.Variable(tf.zeros([n_feature]))
    epsilon = 0.001
    h_fc1_drop = (h_fc1_drop - fc_mean)/tf.sqrt(fc_var+epsilon)

# Fully-connected layer 2
W_fc2 = weight_variables([200, 100])
b_fc2 = bias_variables([100])
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2)+b_fc2)
h_fc2_drop = tf.nn.dropout(h_fc2, kp_2)
if norm:
    fc_mean, fc_var = tf.nn.moments(
        h_fc2_drop,
        axes=[0],
    )
    scale = tf.Variable(tf.ones([n_feature]))
    shift = tf.Variable(tf.zeros([n_feature]))
    epsilon = 0.001
    h_fc2_drop = (h_fc2_drop - fc_mean)/tf.sqrt(fc_var+epsilon)

# Fully-connected layer 3
W_fc3 = weight_variables([100, 9])
b_fc3 = bias_variables([9])
h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3)+b_fc3)
prediction = tf.nn.softmax(h_fc3)
```

TensorFlow

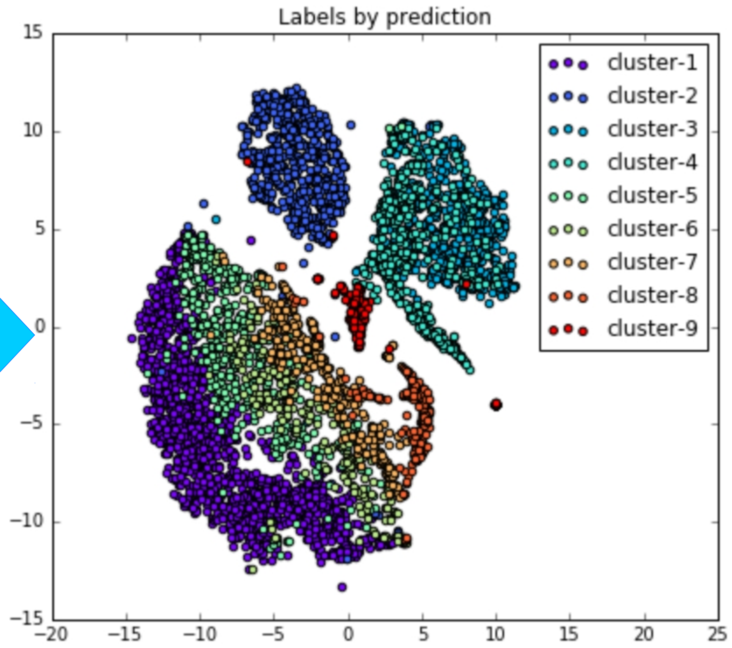
Example: celltype predictor

Feedforward neural net
with two hidden layers

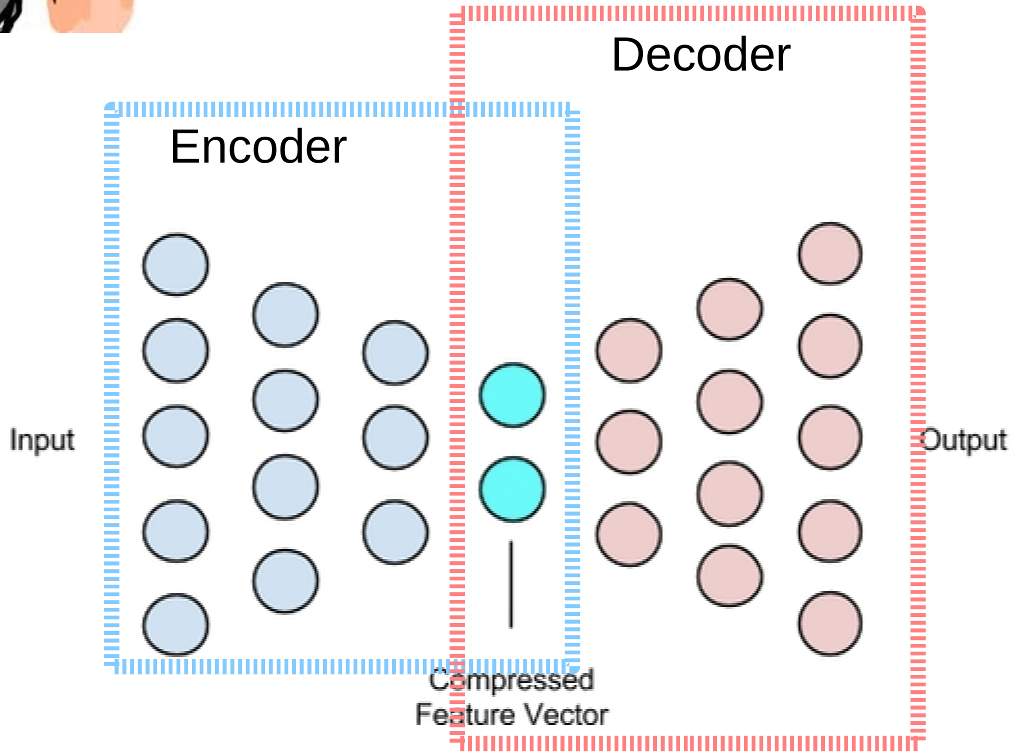
Sinlge-cell RNA-seq data from 10xGenomics
PBMC sample from healthy donors



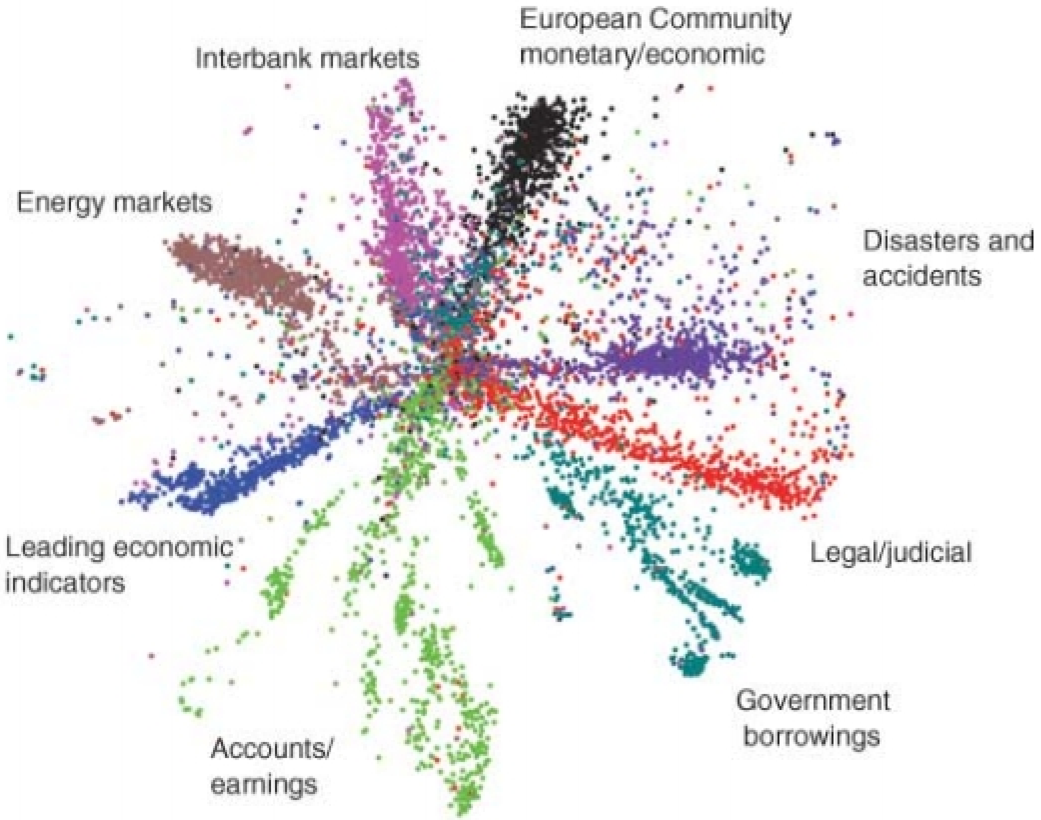

```
tf.reset_default_graph()
norm = True
n_feature = train_x.shape[1]
xs = tf.placeholder(tf.float32, [None, n_feature])
ys = tf.placeholder(tf.float32, [None, 9])
kp_1 = tf.placeholder(tf.float32) # keep_prob
kp_2 = tf.placeholder(tf.float32) # keep_prob

# Fully-connected layer 1
W_fc1 = weight_variables([n_feature, 200])
b_fc1 = bias_variables([200])
h_fc1 = tf.nn.relu(tf.matmul(xs, W_fc1)+b_fc1)
h_fc1_drop = tf.nn.dropout(h_fc1, kp_1)
if norm:
    fc_mean, fc_var = tf.nn.moments(
        h_fc1_drop,
        axes=[0],
    )
    scale = tf.Variable(tf.ones([n_feature]))
    shift = tf.Variable(tf.zeros([n_feature]))
    epsilon = 0.001
    h_fc1_drop = (h_fc1_drop - fc_mean)/tf.sqrt(fc_var+epsilon)

# Fully-connected layer 2
W_fc2 = weight_variables([200, 100])
b_fc2 = bias_variables([100])
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2)+b_fc2)
h_fc2_drop = tf.nn.dropout(h_fc2, kp_2)
if norm:
    fc_mean, fc_var = tf.nn.moments(
        h_fc2_drop,
        axes=[0],
    )
    scale = tf.Variable(tf.ones([n_feature]))
    shift = tf.Variable(tf.zeros([n_feature]))
    epsilon = 0.001
    h_fc2_drop = (h_fc2_drop - fc_mean)/tf.sqrt(fc_var+epsilon)

# Fully-connected layer 3
W_fc3 = weight_variables([100, 9])
b_fc3 = bias_variables([9])
h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3)+b_fc3)
prediction = tf.nn.softmax(h_fc3)
```



Types of NN: Autoencoder

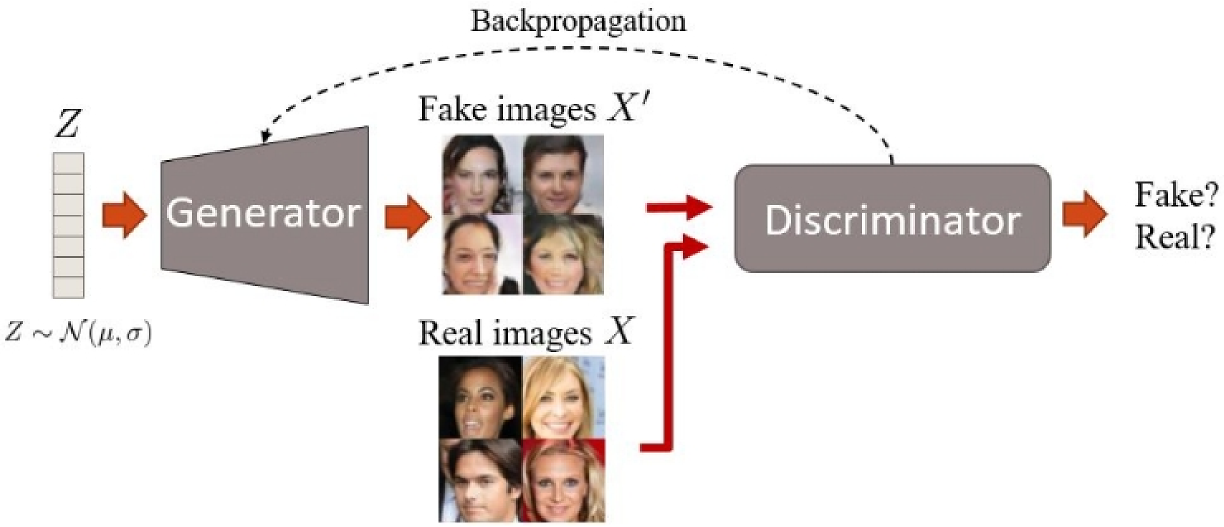


Dimension reduction by autoencoder



Hinton & Salakhutdinov, *Science*, 2006

Types of NN: Generative Adversarial Networks (GAN)

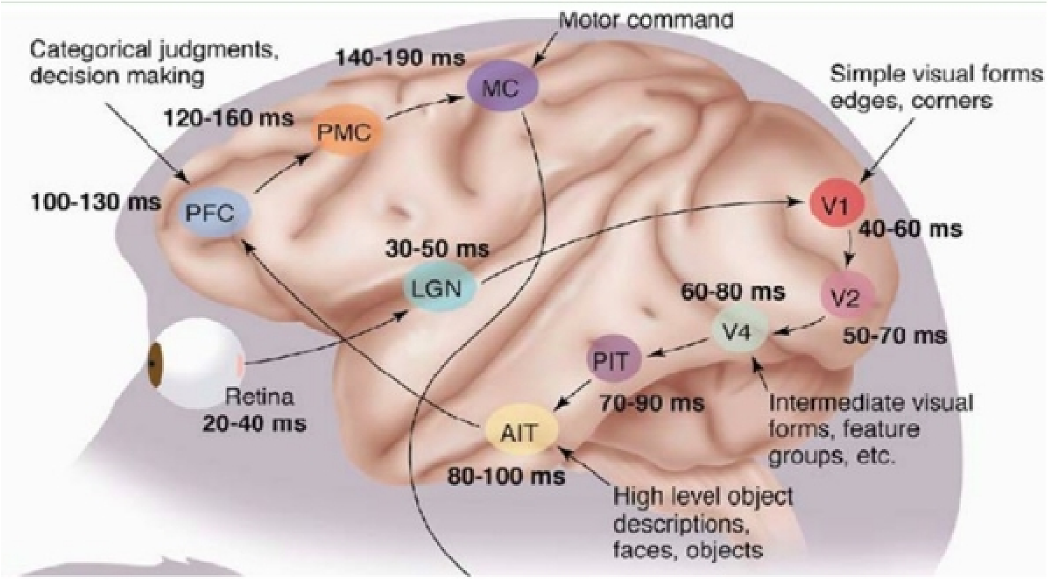


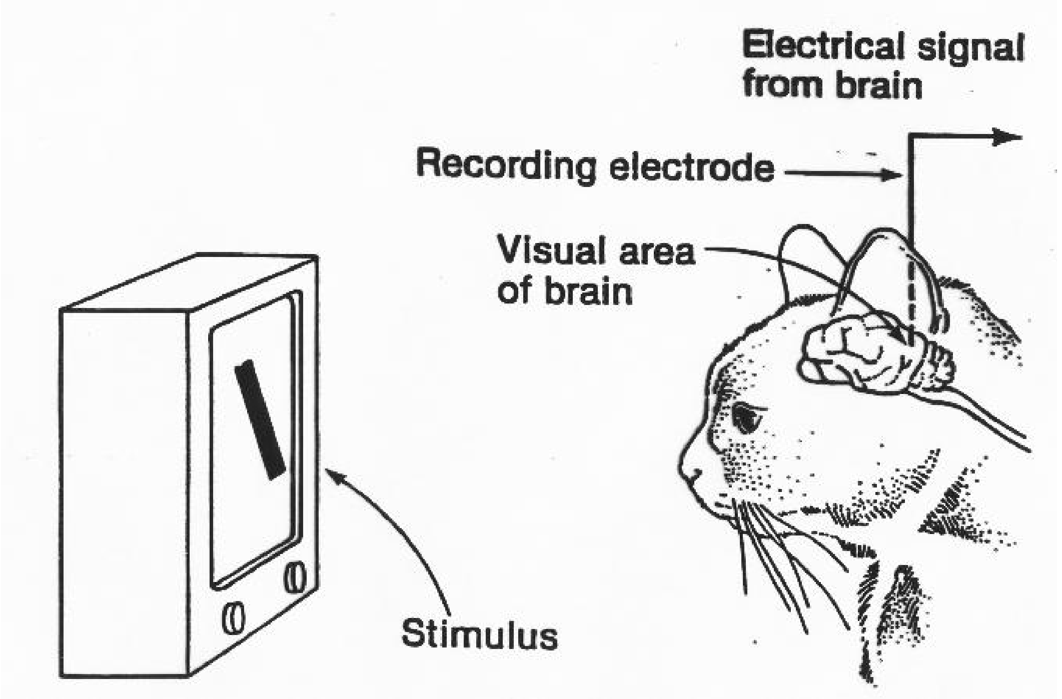
GAN training overview.

Types of NN: Convolutional Neural Net (CNN)

CNN is the most powerful approach for image recognition so far.

Visual system

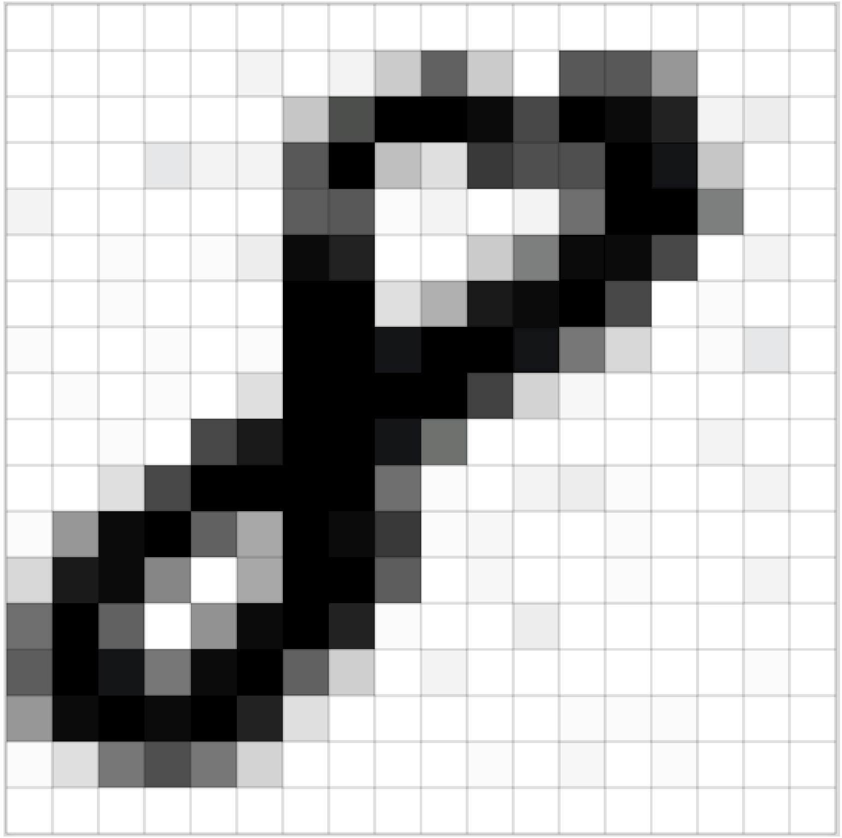




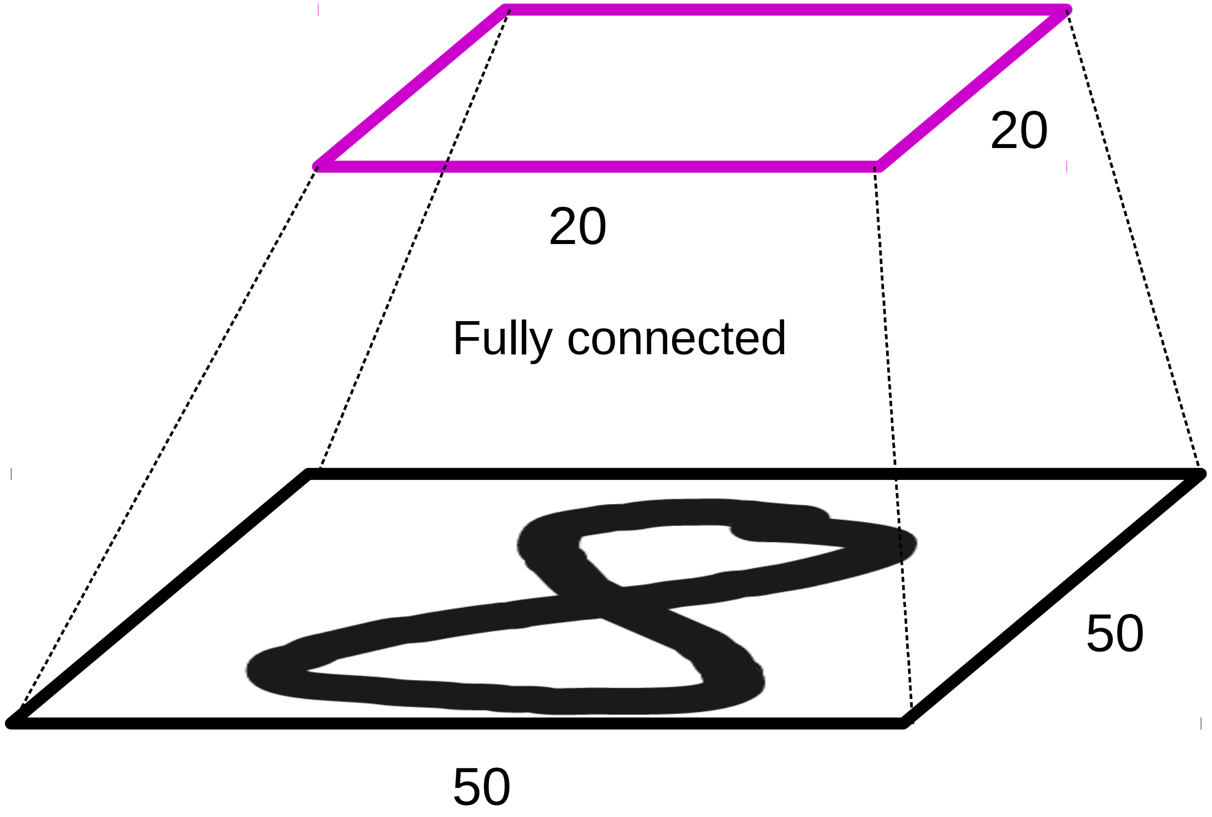
V1 cortex tested in this experiment was only active in response to one simple pattern.
Many identical cells detect the same pattern, which are connected to different parts of the retina.

Convolutional Neural Net

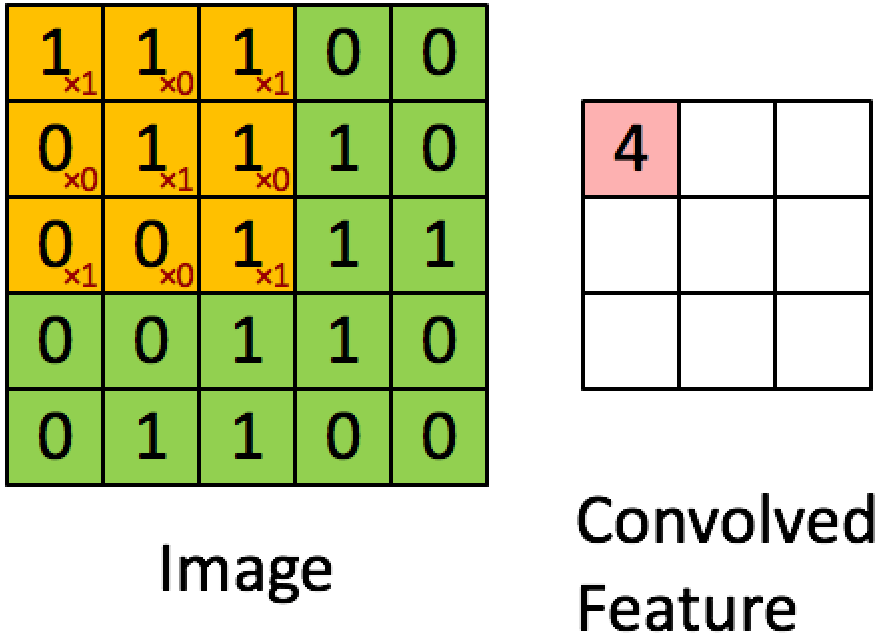
Why not fully connected?

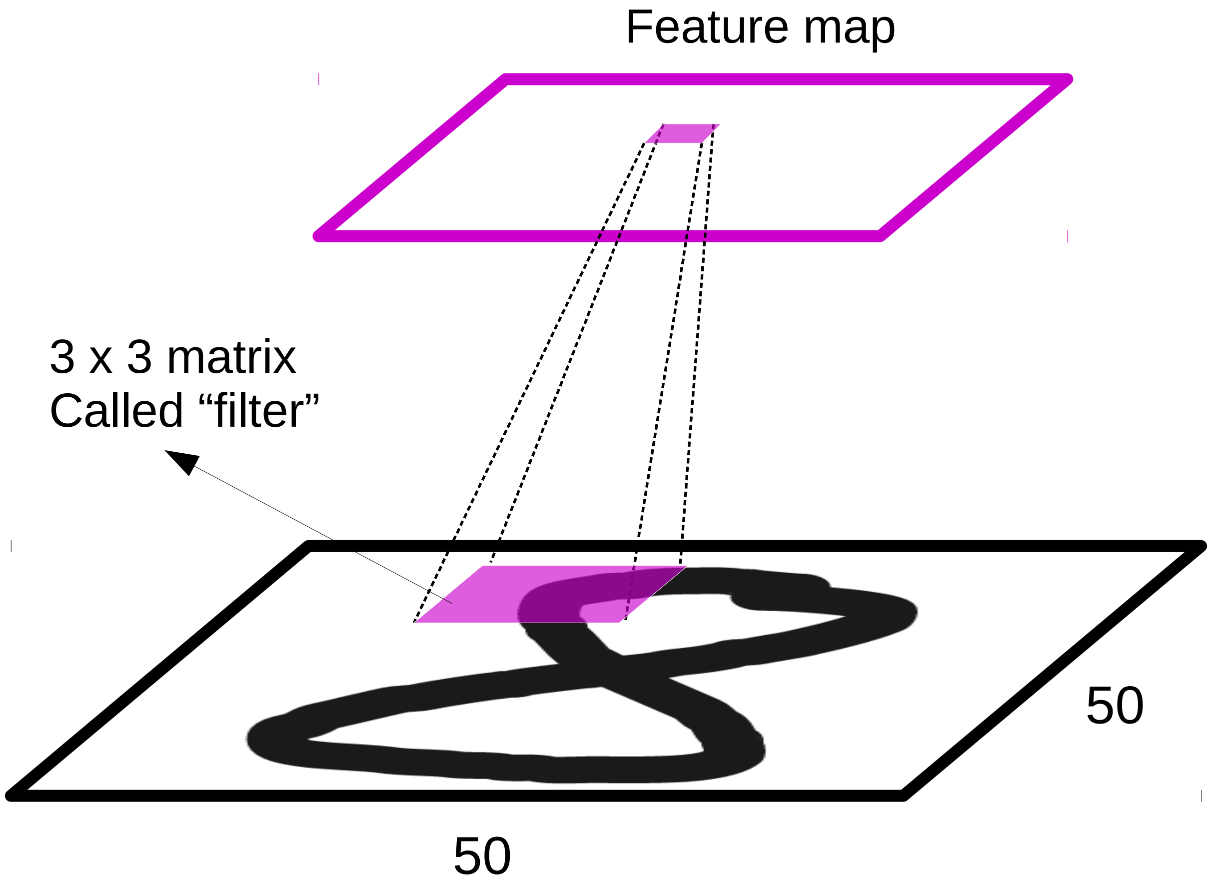


Parameters for one single layer:
50*50*20*20 = 1 million!

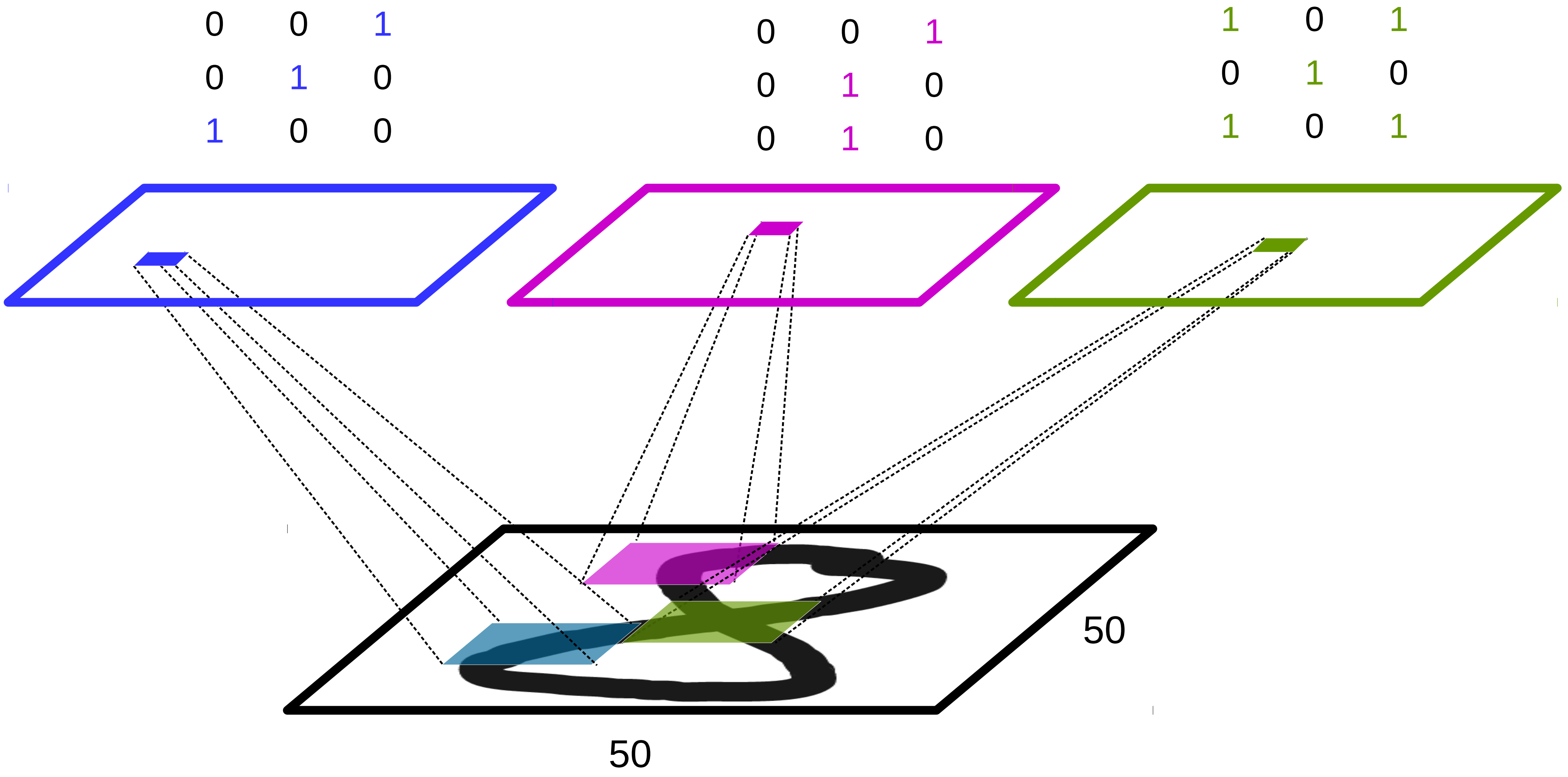


Convolutional Neural Net





Convolutional Neural Net



Convolutional Neural Net

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

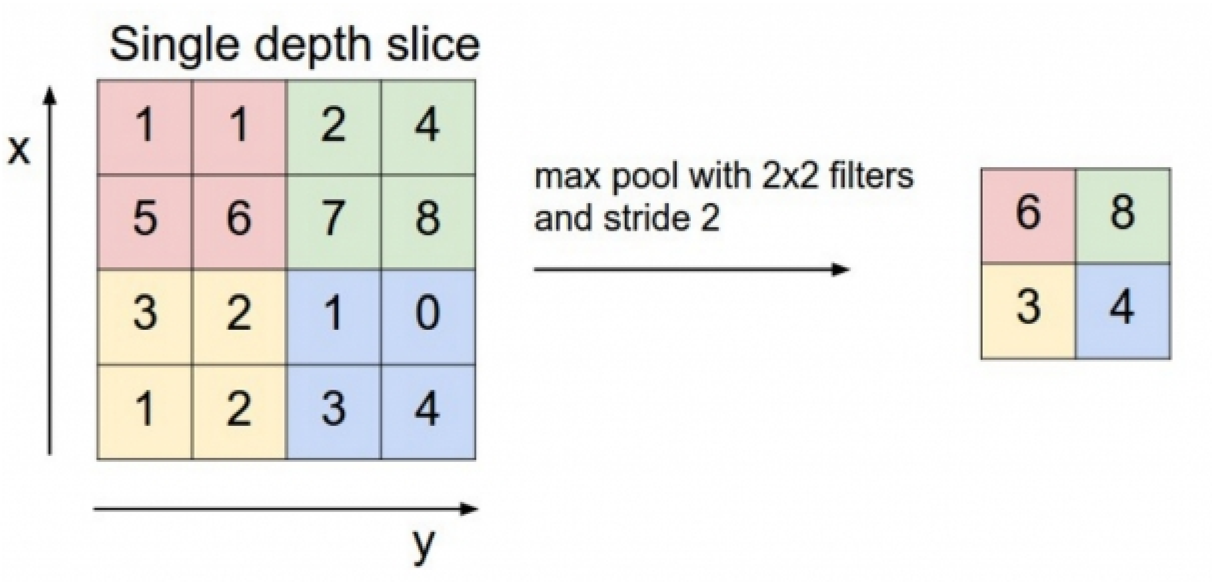
	0	1	0	
	1	-4	1	
	0	1	0	

Averaging neighbors blurs the figure

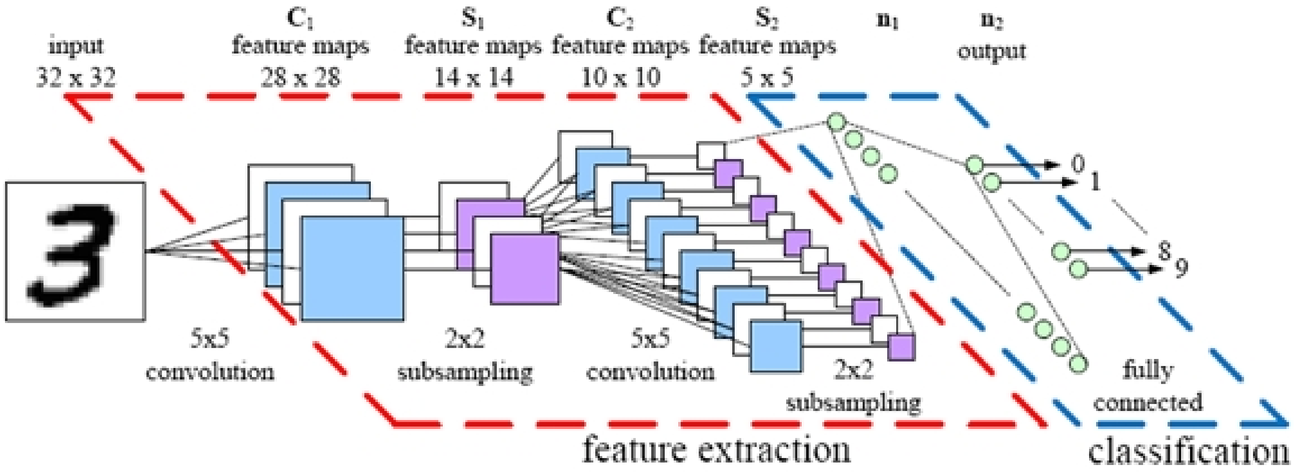
Taking difference with neighbors detects edges

Convolutional Neural Net

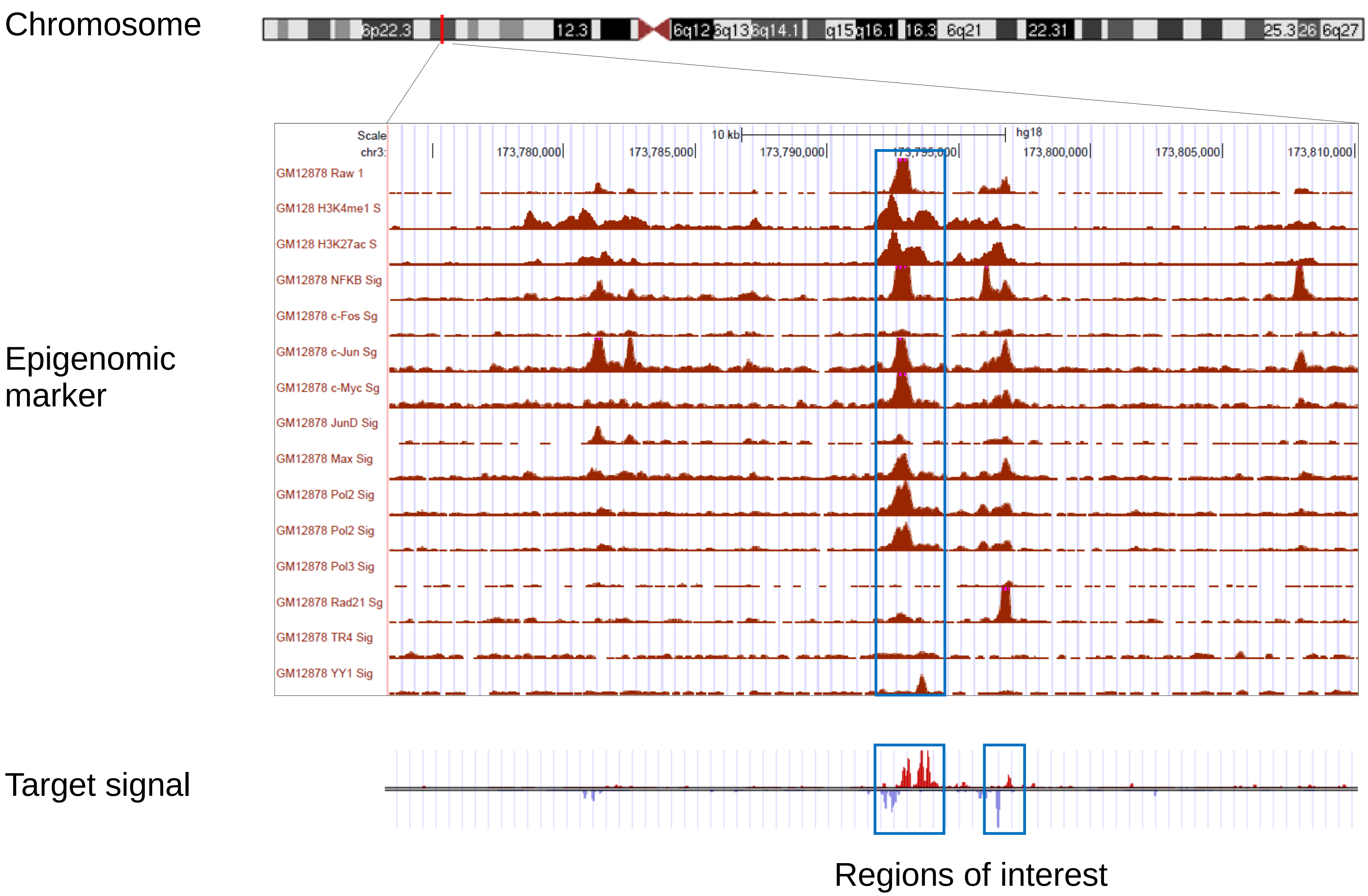
Pooling:
1. Reduces dimensions
2. Allow positional variation



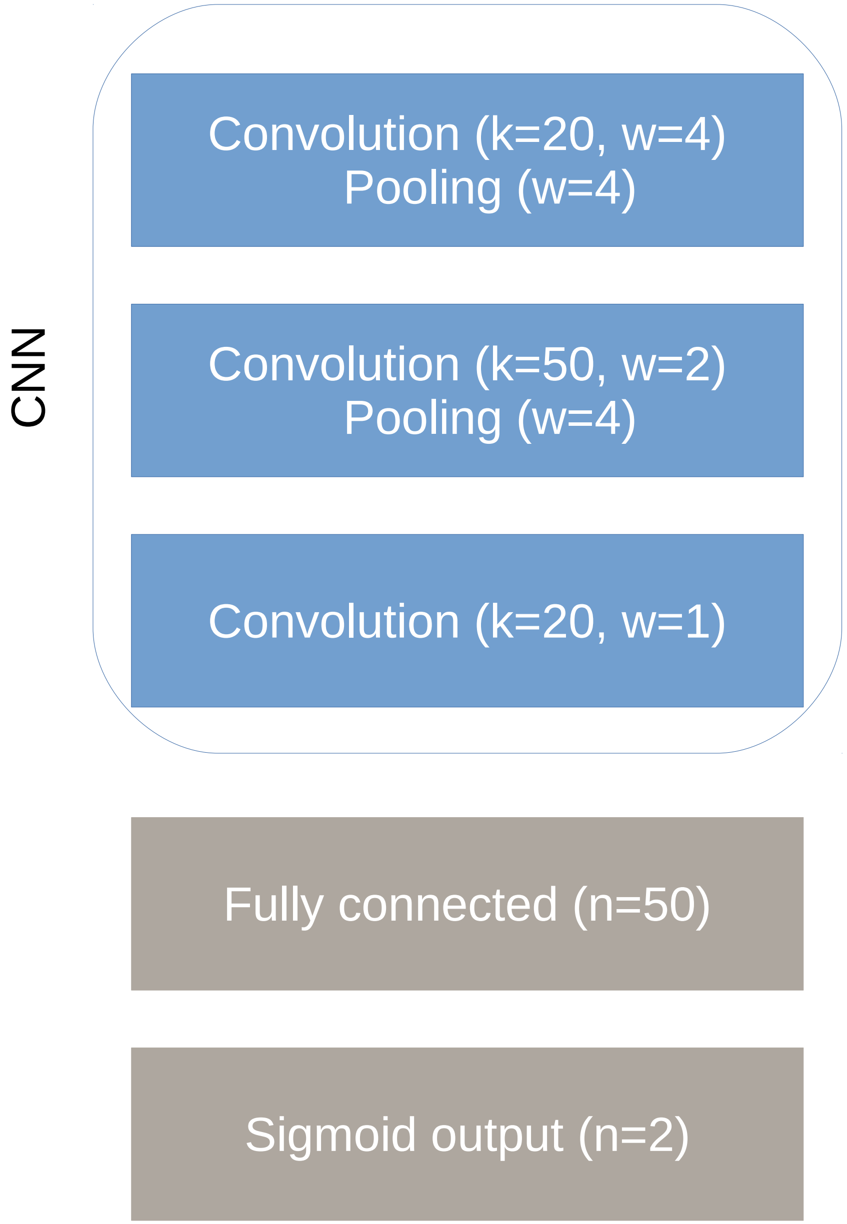
Convolutional Neural Net



Application 1: epigenome reader



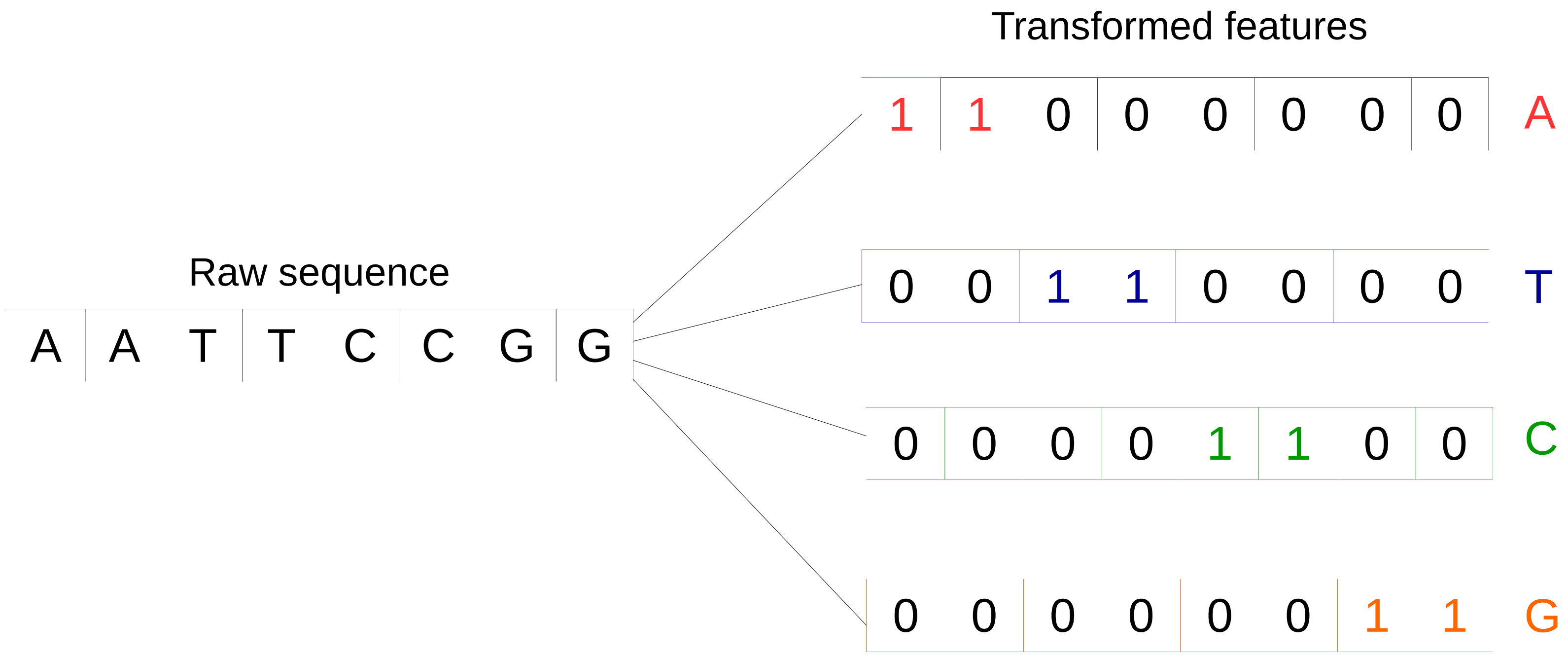
Application 1: epigenome reader



Regularization Parameters:
Dropout proportion
Layer 2: 20%
Layer 4: 20%
Layer 5: 40%
All other layers: 0%

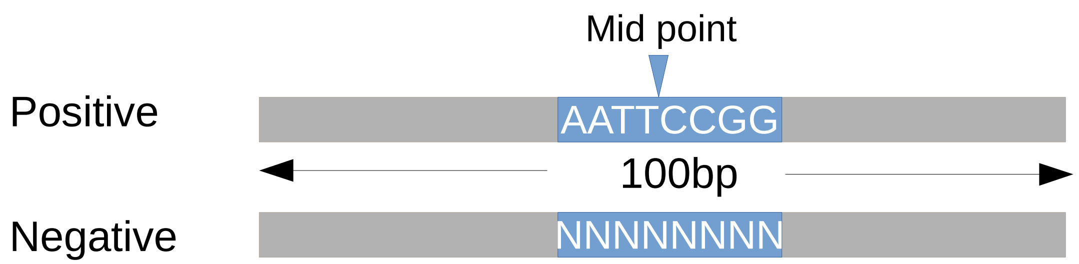
	Training	Validation	Testing
Accuracy	93.1%	93.7%	92.1%

Input transformation



Simulation

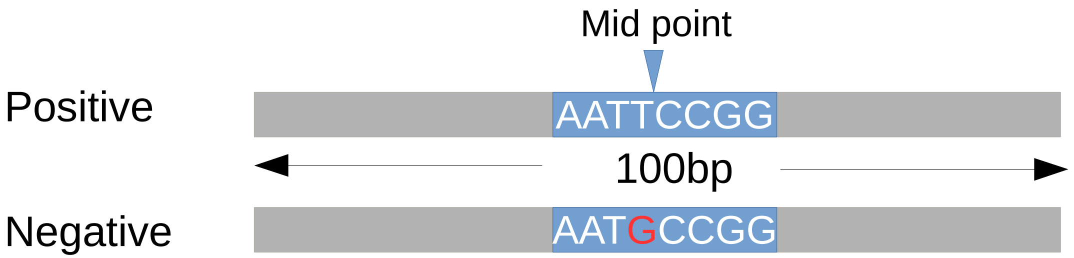
Simulation_1: learning motif sequence



		Training	Validation	Testing
Fixed position	No mutation	0.00%	0.05%	0.00%
	2/8 mutations	0.57%	0.65%	0.57%
	4/8 mutations	5.31%	5.90%	6.95%
	6/8 mutations	47.52%	47.2%	49.98%

Simulation

Simulation_1.1: learning motif sequence and detect mutations

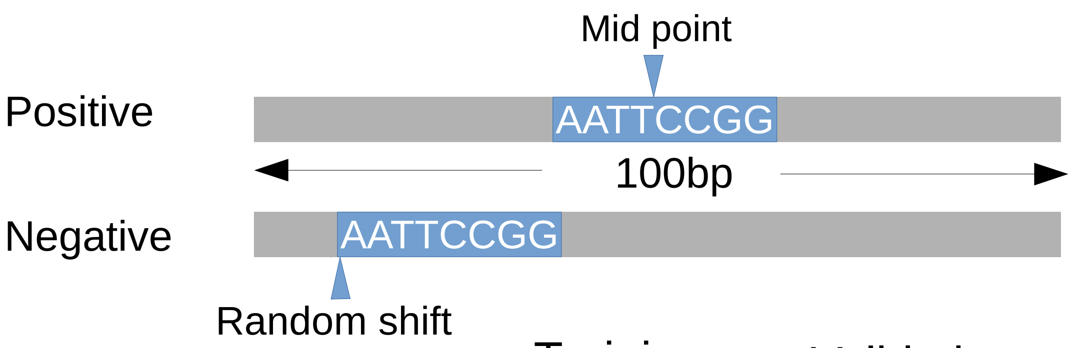


(negative samples with one mutation in motif)

		Training	Validation	Testing
Fixed position	1/8 mutation	0.07%	0.00%	0.08%
	1/8 mutation [25, 75]	0.07%	0.00%	0.13%

Simulation

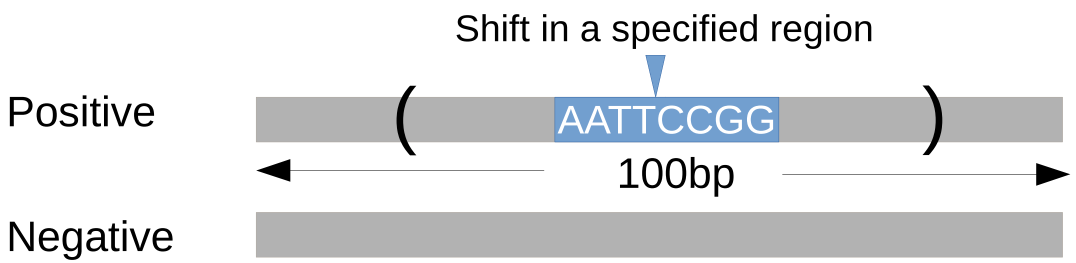
Simulation_2: learning motif position



		Training	Validation	Testing
Position	Positive fixed core Negative randomly-shifted core	0.03%	0.00%	0.10%
	Positive core in region Negative core out of region	0.04%	0.00%	0.17%

Simulation

Simulation_3: testing positional flexibility

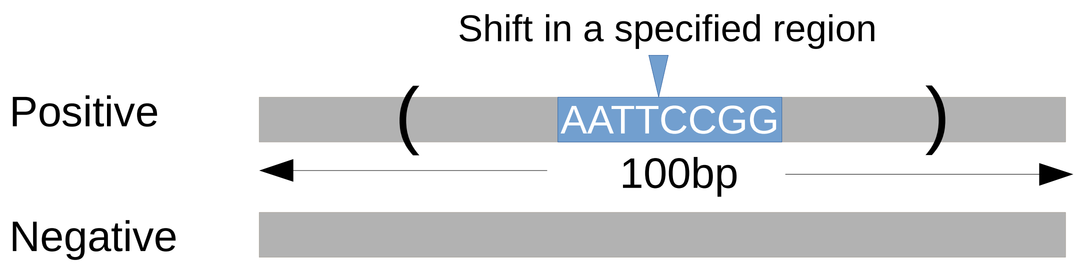


		Training	Validation	Testing
Flexibility	Region size: 50bp	0.14%	0.03%	0.03%
	Region size: 100bp	0.11%	0.08%	0.15%

Conclusion from 1-3:
CNN is able to learn both sequence and positional information, while allowing positional flexibility

Simulation

Simulation_4: mixture

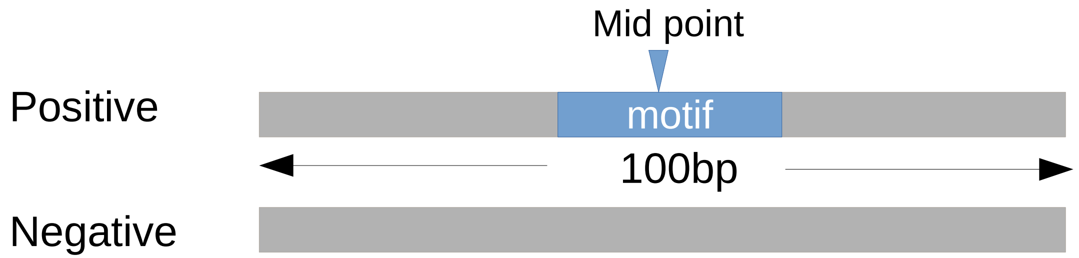


		Training	Validation	Testing
Mixture	2/8 mutations + 50bp flexible region	9.608333%	9.975000%	9.125000%
Fixed position	2/8 mutations	0.57%	0.65%	0.57%

Better alignments of regulatory sequences is helpful for feature detection

Simulation

Simulation_5: learning multiple motifs



		Training	Validation	Testing
Multiple motifs	10	0.13%	0.05%	0.20%
	20	0.74%	0.60%	0.82%
	40	1.25%	1.25%	1.93%
	80	28.76%	22.15%	23.05%
	20 motifs + 50bp region + 1/8 mutations	39.80%	43.25%	43.88%

Summary

- Artificial intelligence should be better than human for reading and understanding biological data.
- Implementing deep learning or training a NN is easier than it seems to be (but harder than understanding it).
- “It's not who has the best algorithm that wins. It's who has the most data.”

Andrew Ng