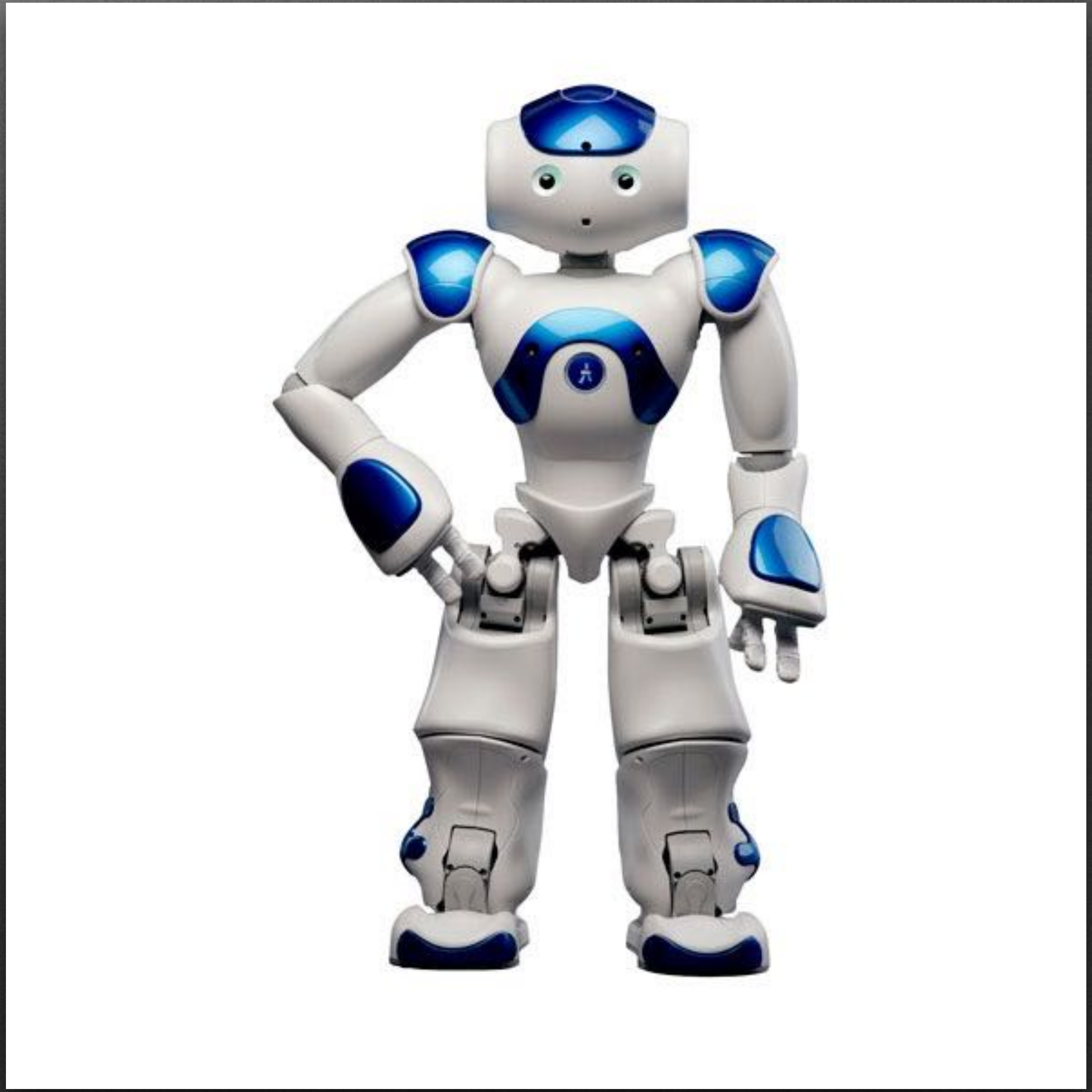
Neural networks across space & time

Dave Snowdon
@davesnowdon
https://www.linkedin.com/in/davesnowdon/

About me

- Java & javascript by day
- Python & clojure by night
- Amateur social roboticist
- Been learning about deep learning for 18 months



Agenda

- Why neural networks
- How do neural networks work
- Convolutional neural networks
- Recurrent neural networks

Why neural networks?

Why care about deep learning?

- Impressive results in a wide range of domains
  - image classification, text descriptions of images, language translation, speech generation, speech recognition…
- Predictable execution (inference) time
- Amenable to hardware acceleration
- Automatic feature extraction

What are features?

Average statement length

Number of variables

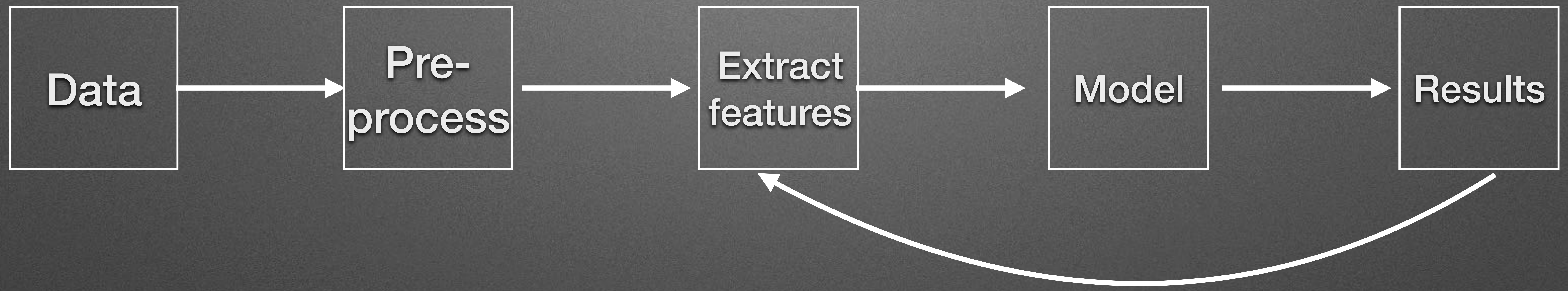
```
10 PRINT "Hello QCon London"
20 GOTO 10
```

Number of statements

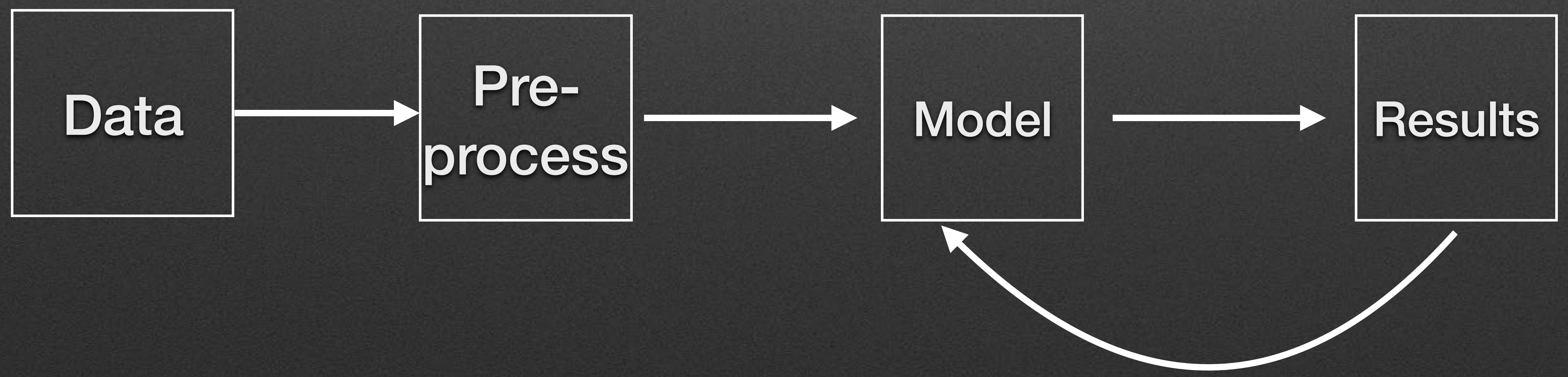
Cyclomatic complexity

Feature extraction

Traditional machine learning process



Deep learning process

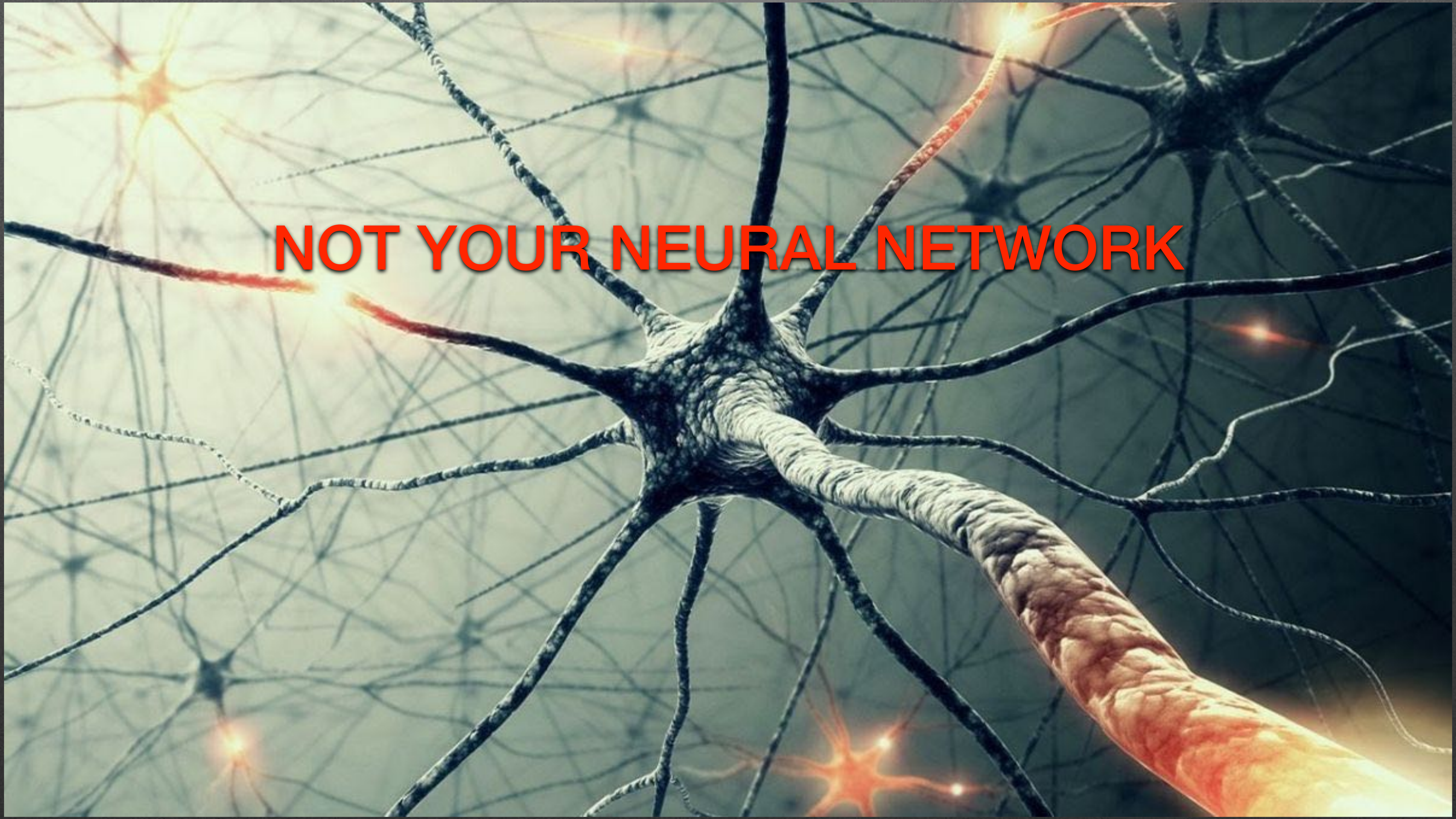


Neural network downsides

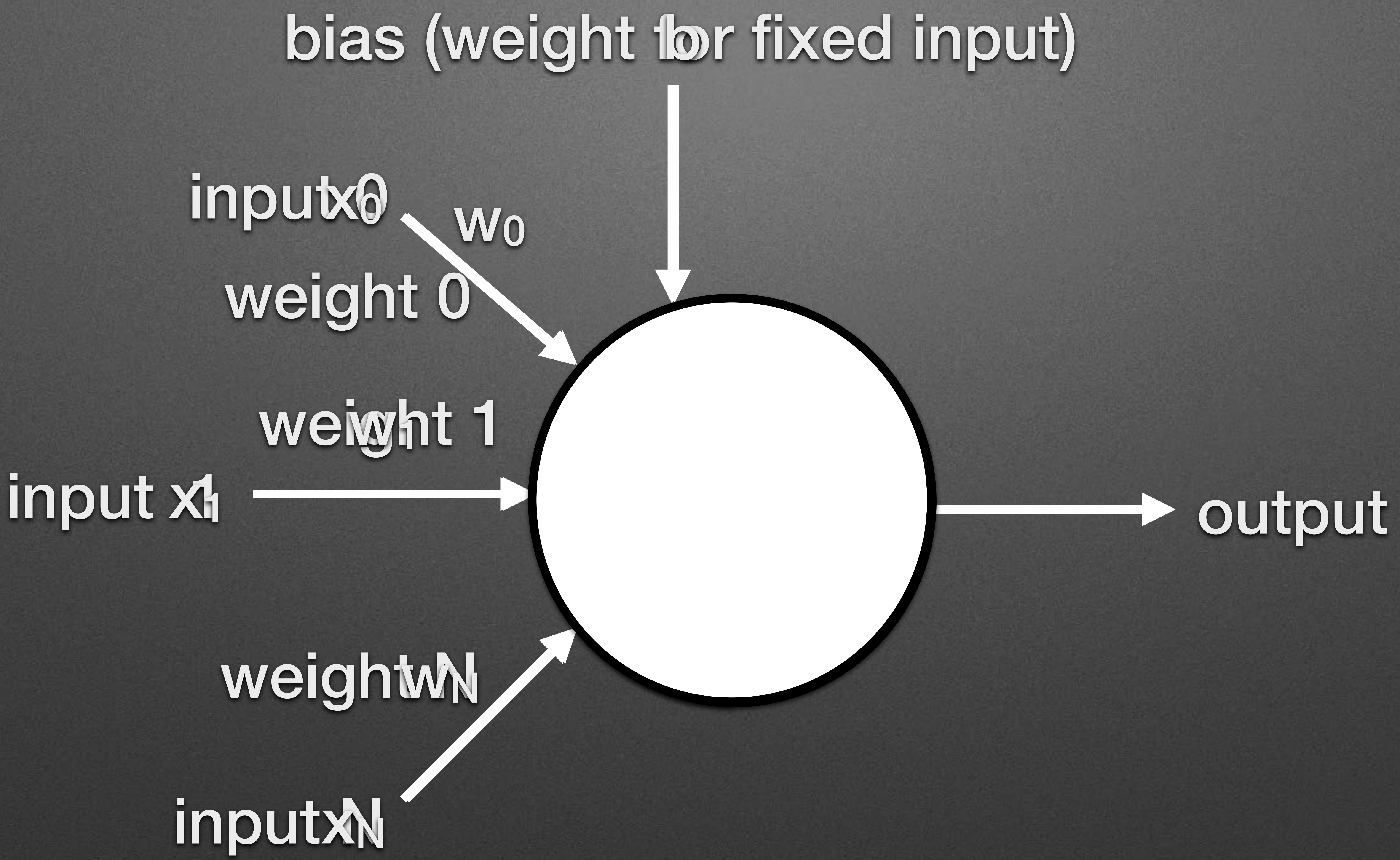
- Need to define the model and it’s training parameters
- Large models can take days or weeks to train
- May need a lot of data. > 10K examples

How neural networks work

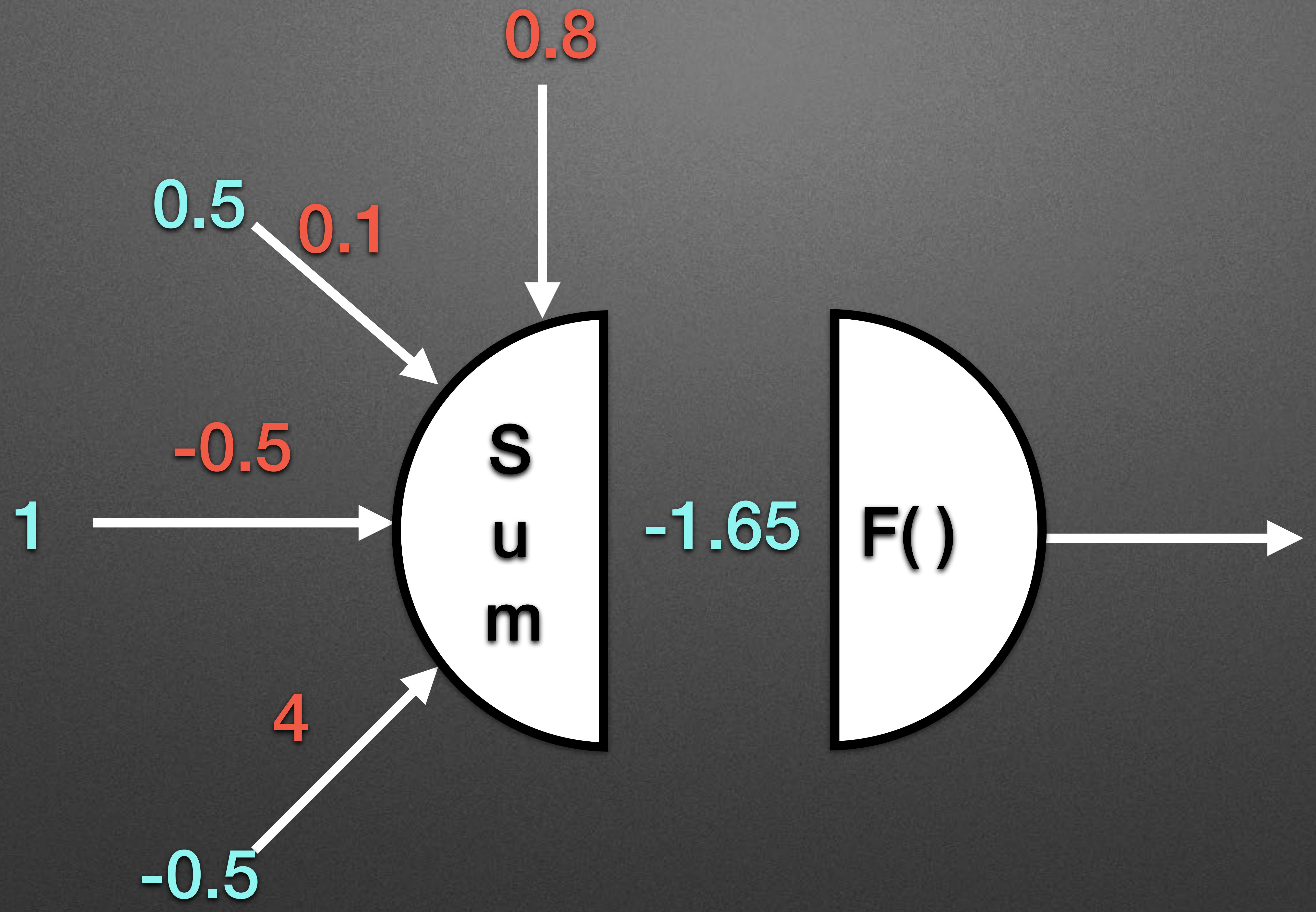
Deep learning != your brain



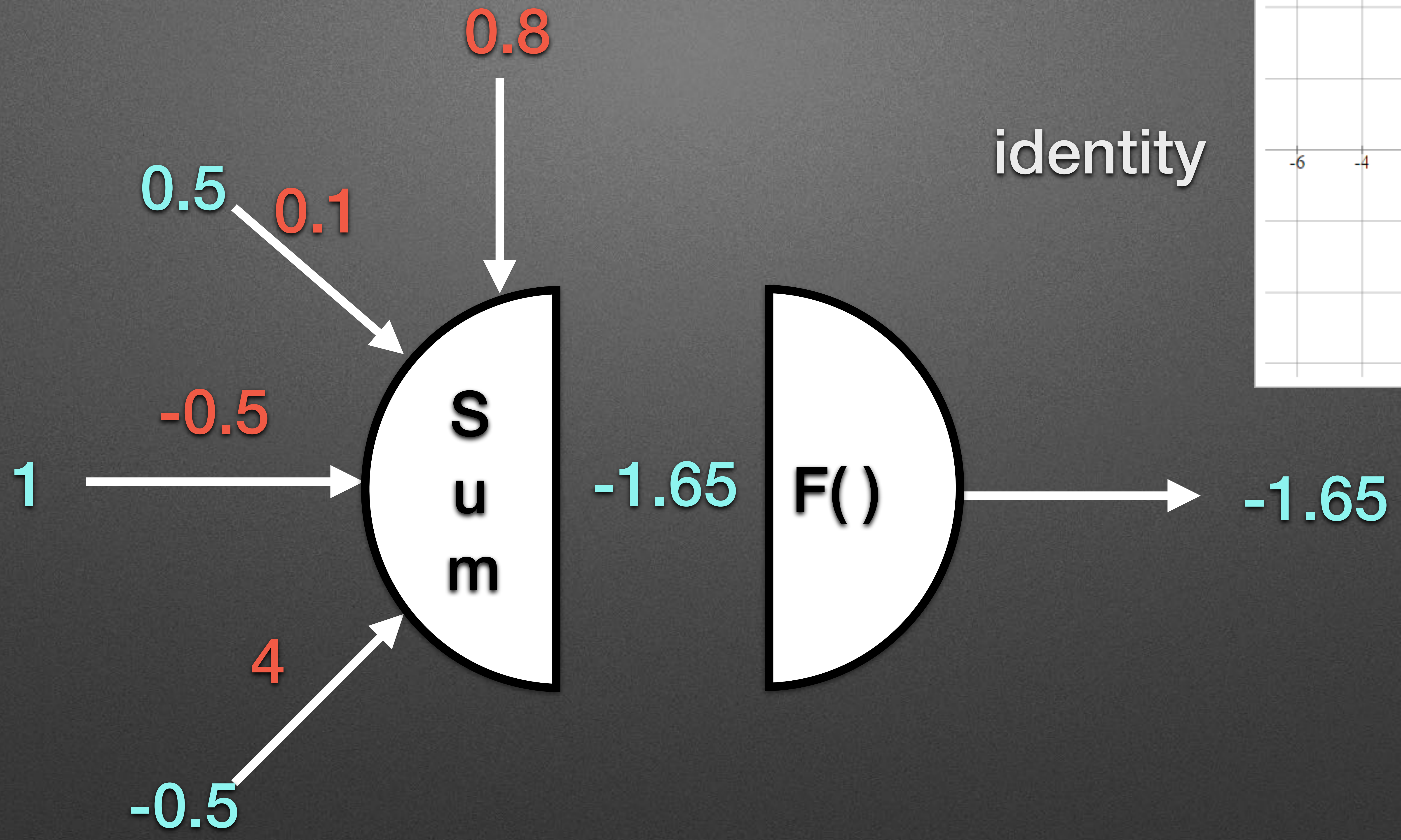
Neuron model



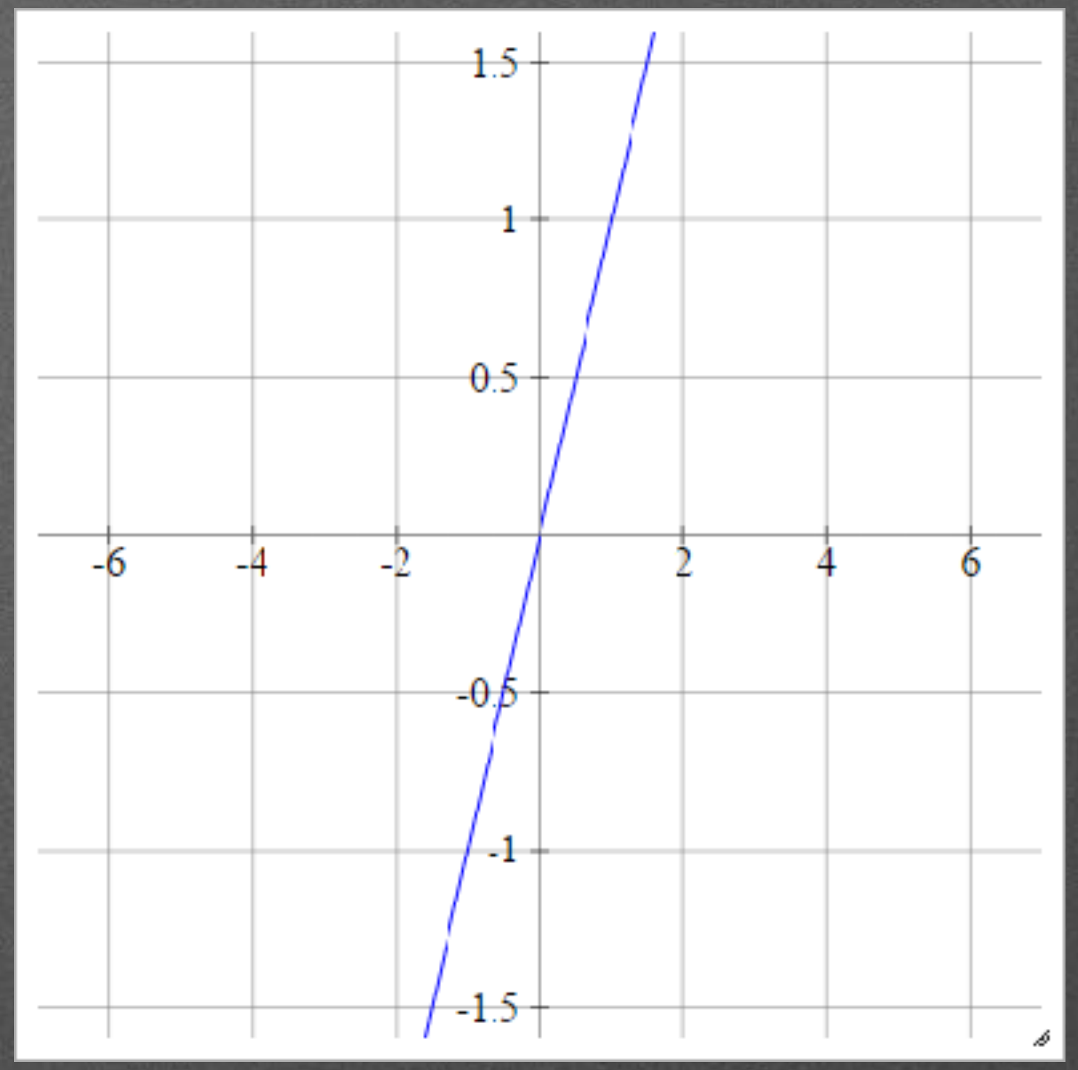
Neuron model



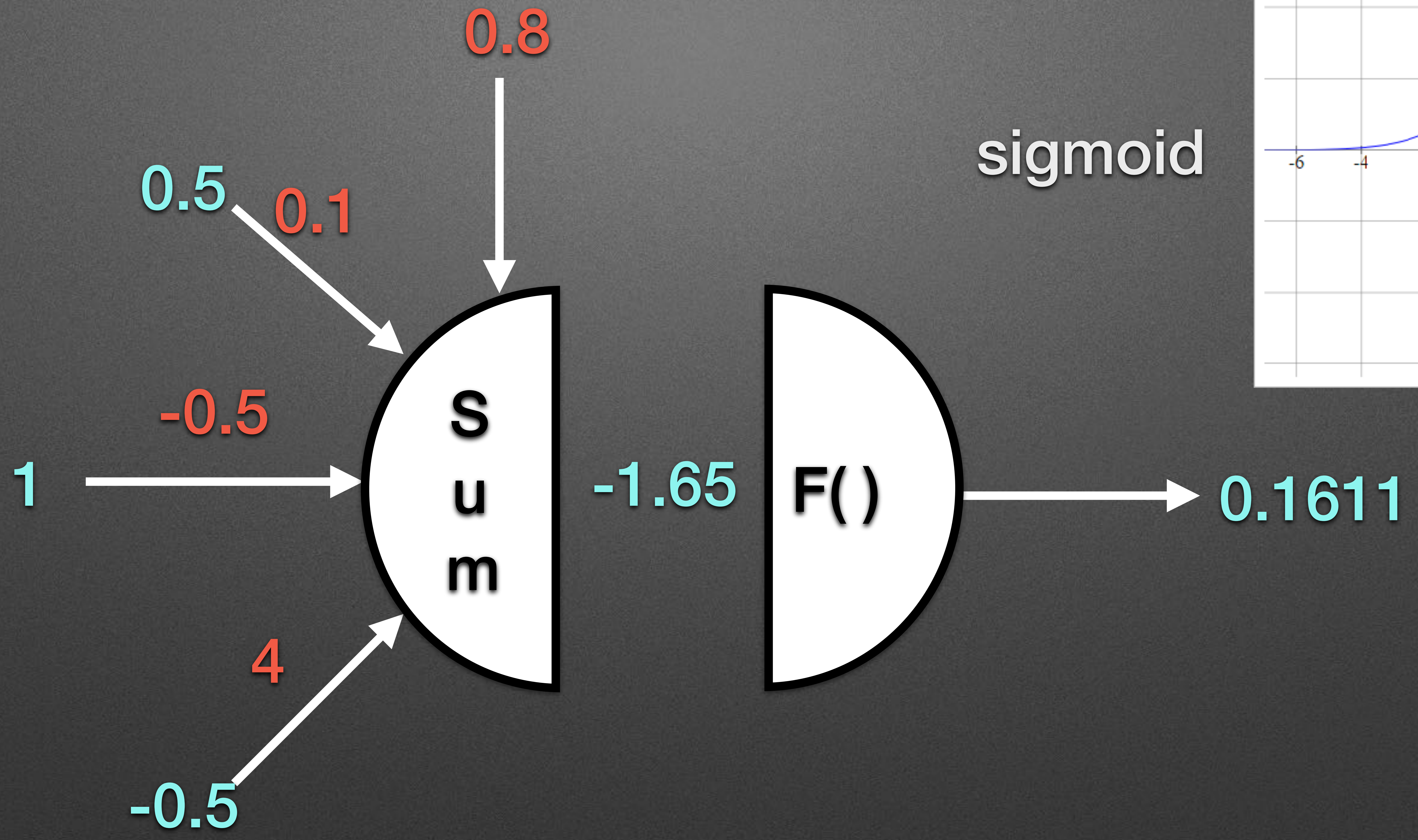
Neuron model



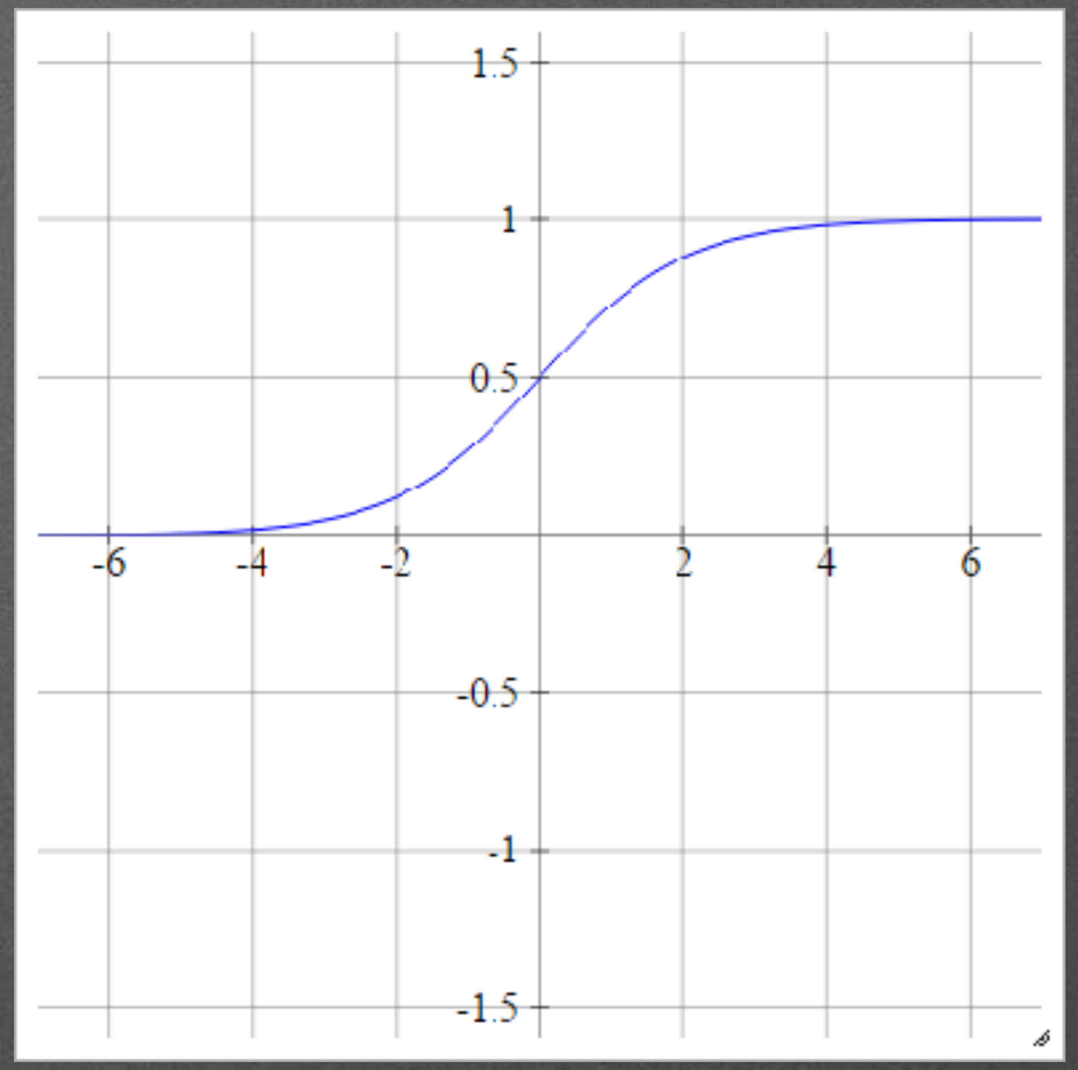
identity



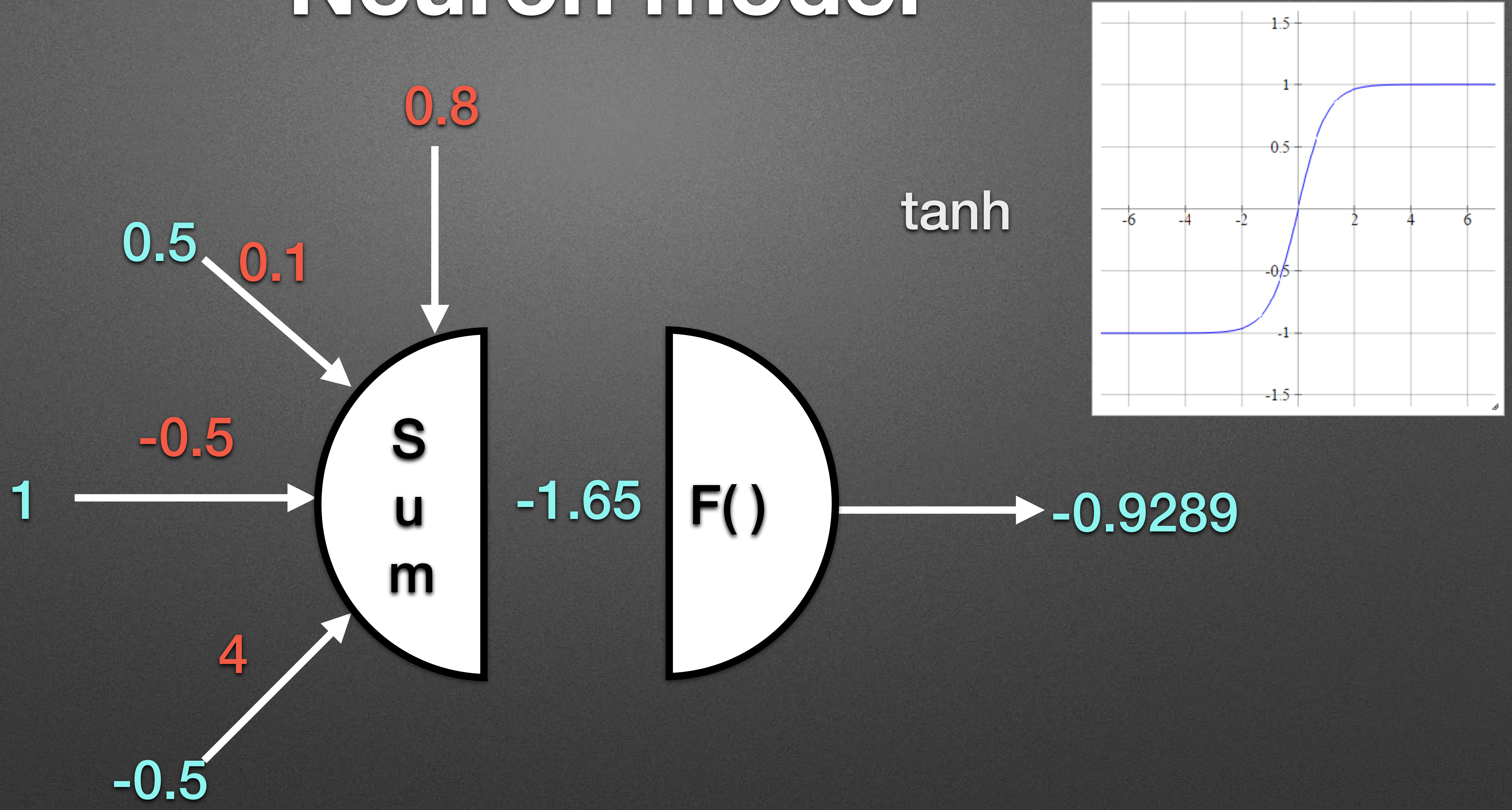
Neuron model



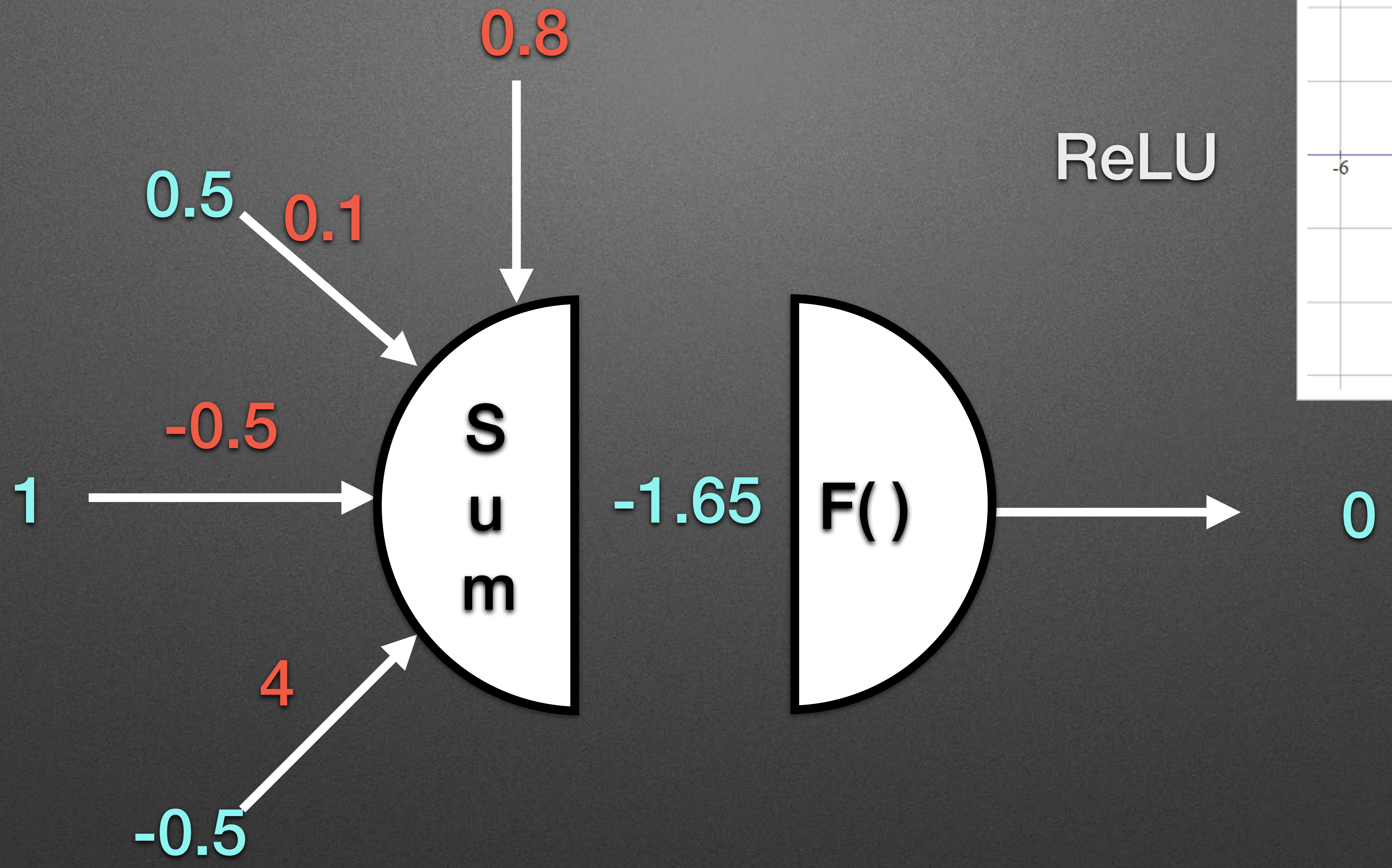
sigmoid



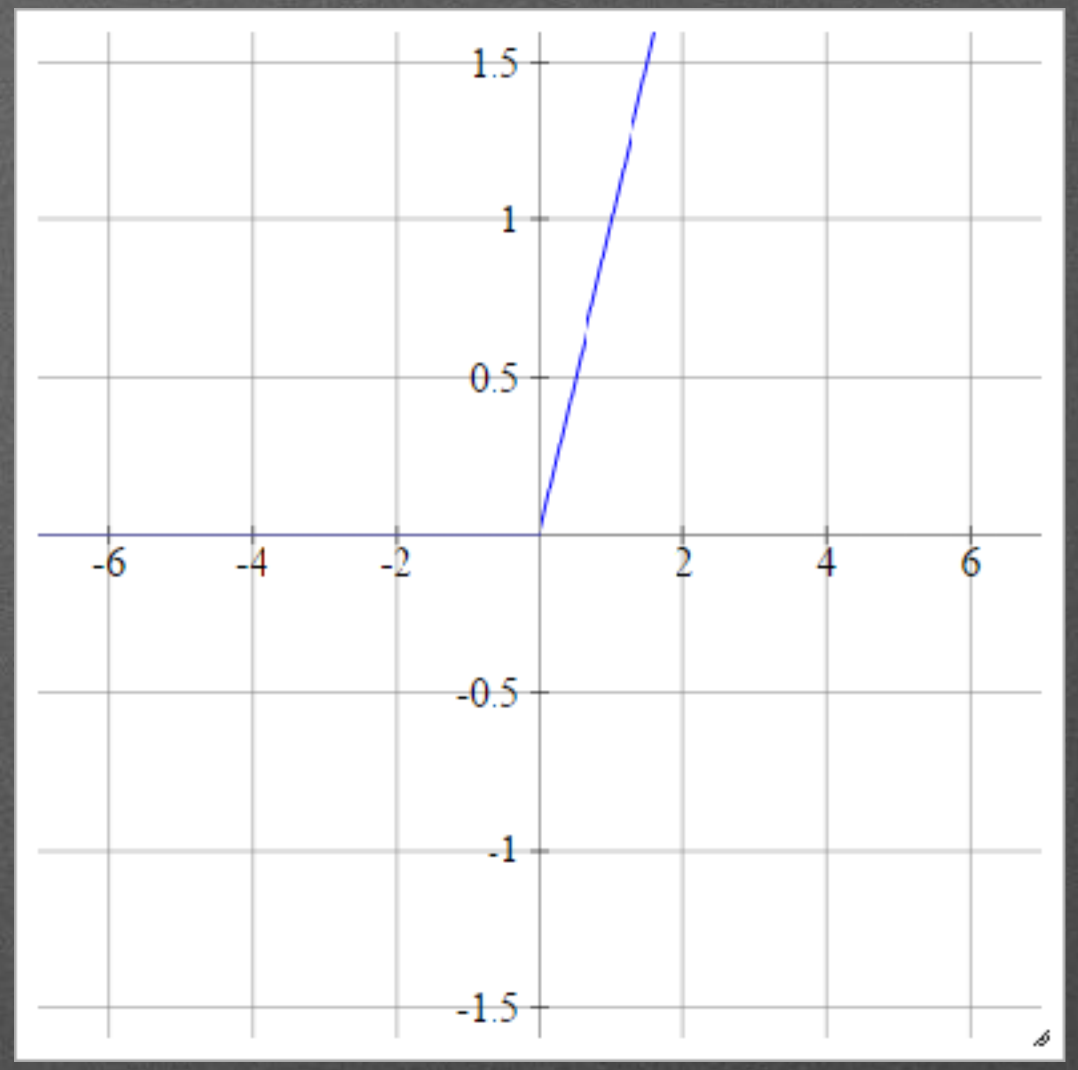
Neuron model



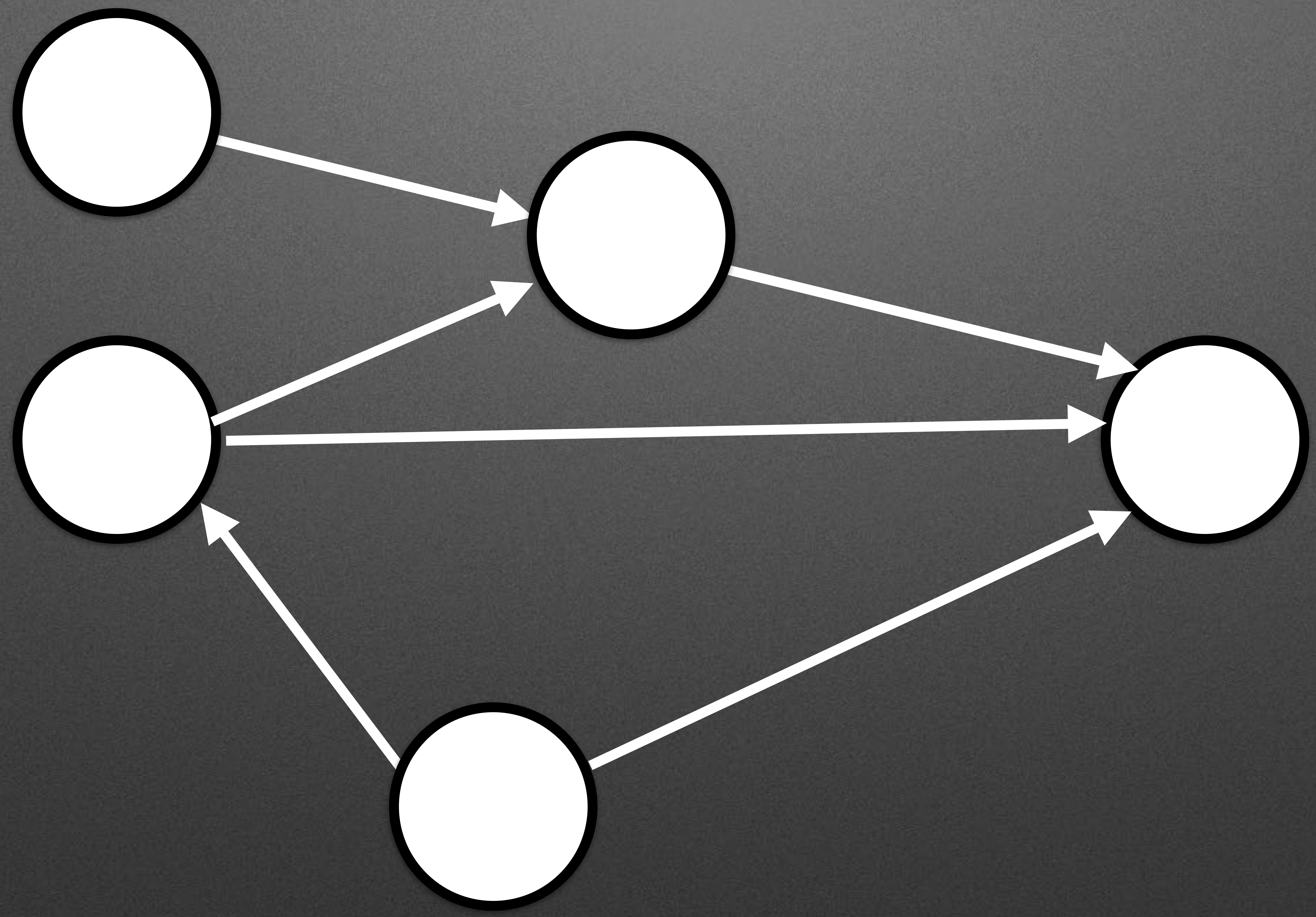
Neuron model



ReLU

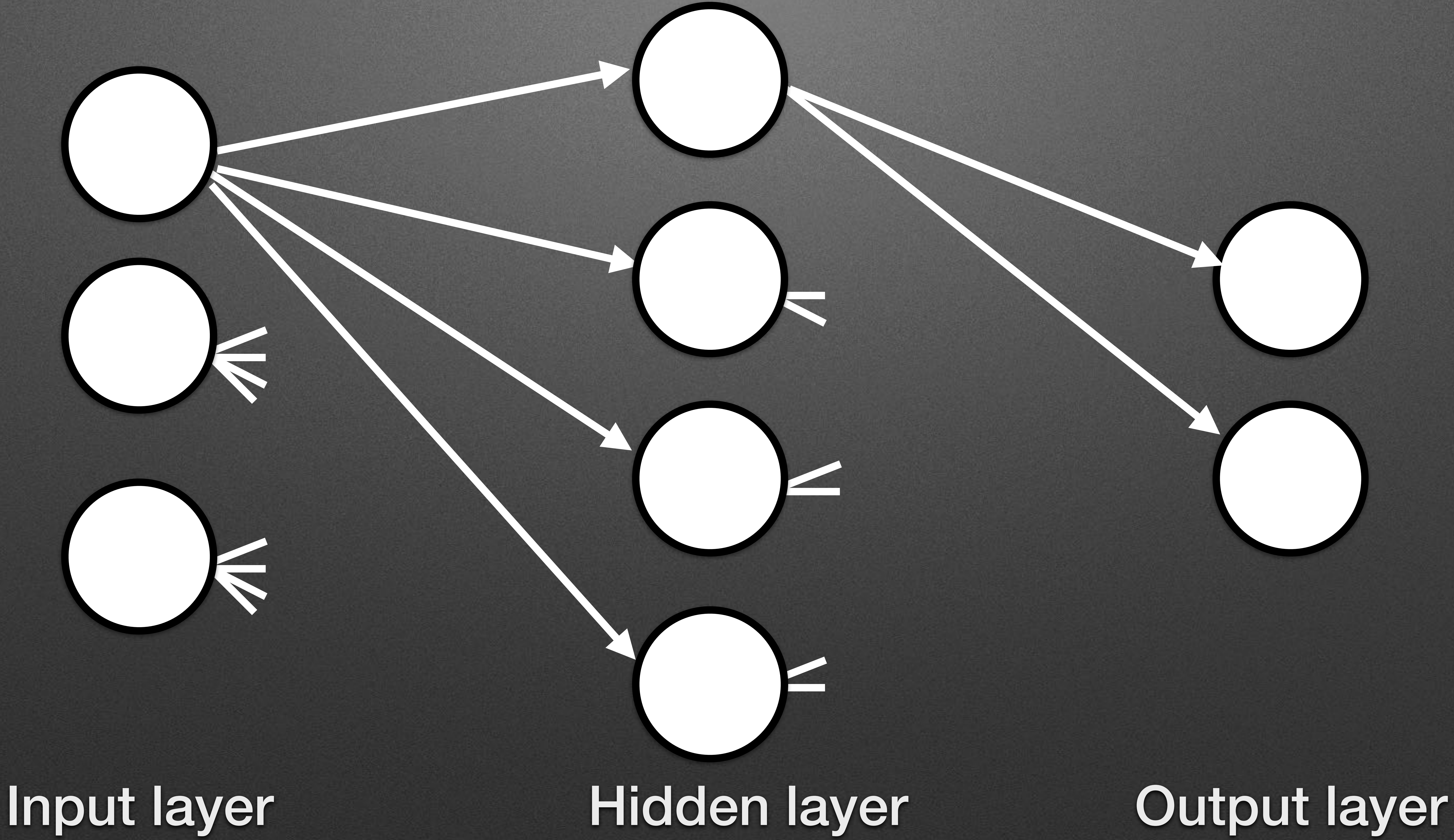


Neural networks are not graphs

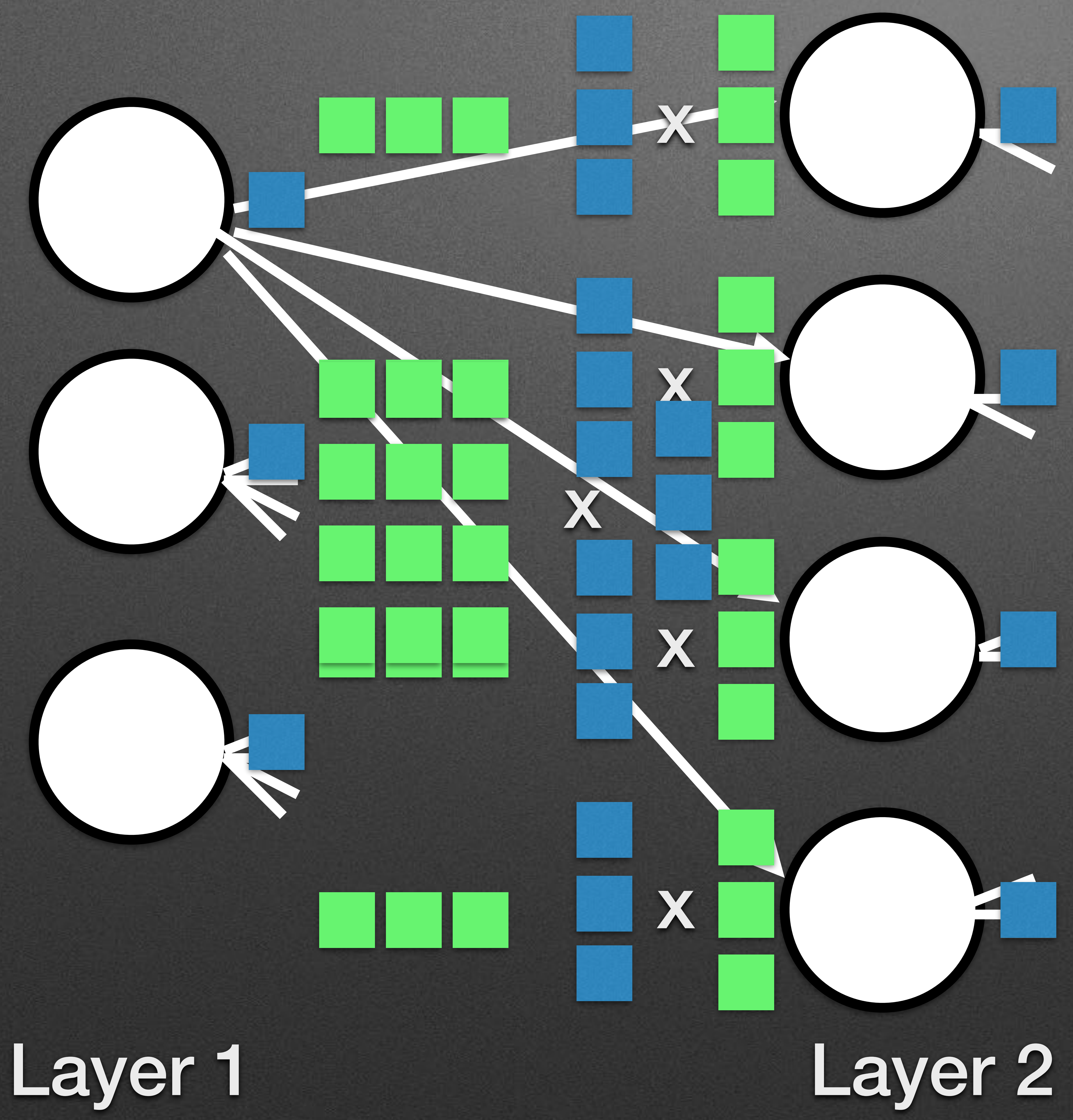


Neural networks are like onions

(they have layers and can make you cry)

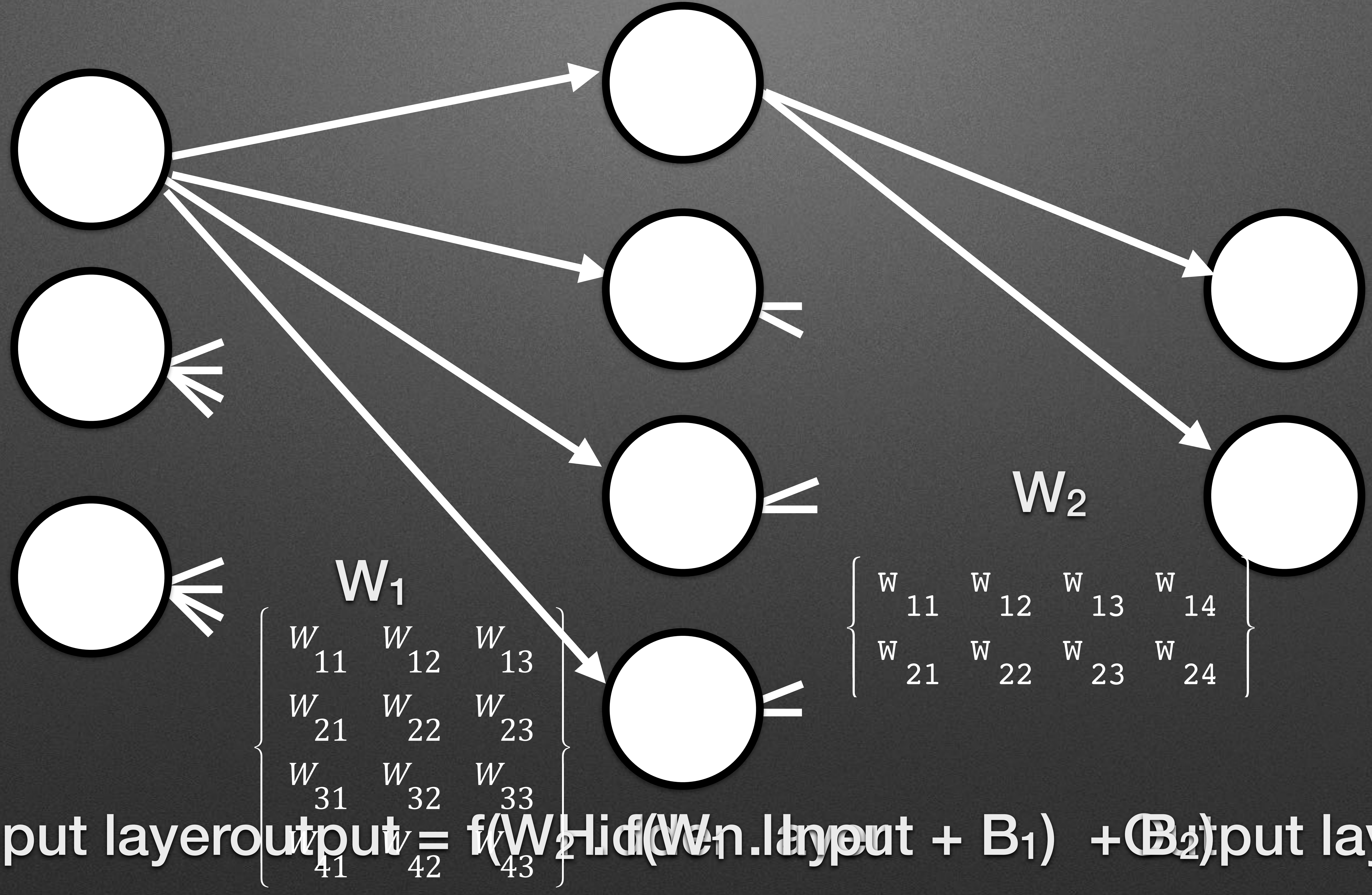


Why layers?



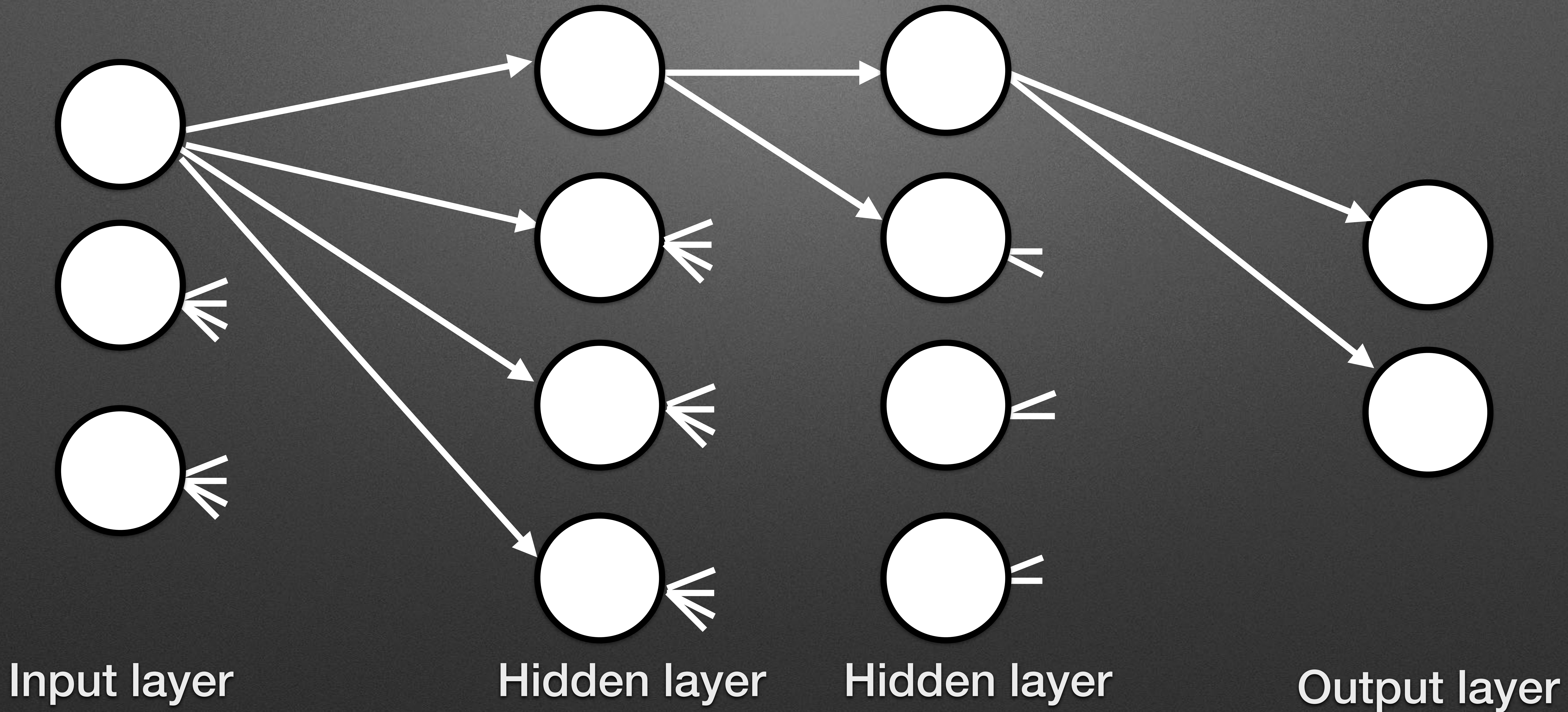
Neural networks are like onions

(they have layers and can make you cry)

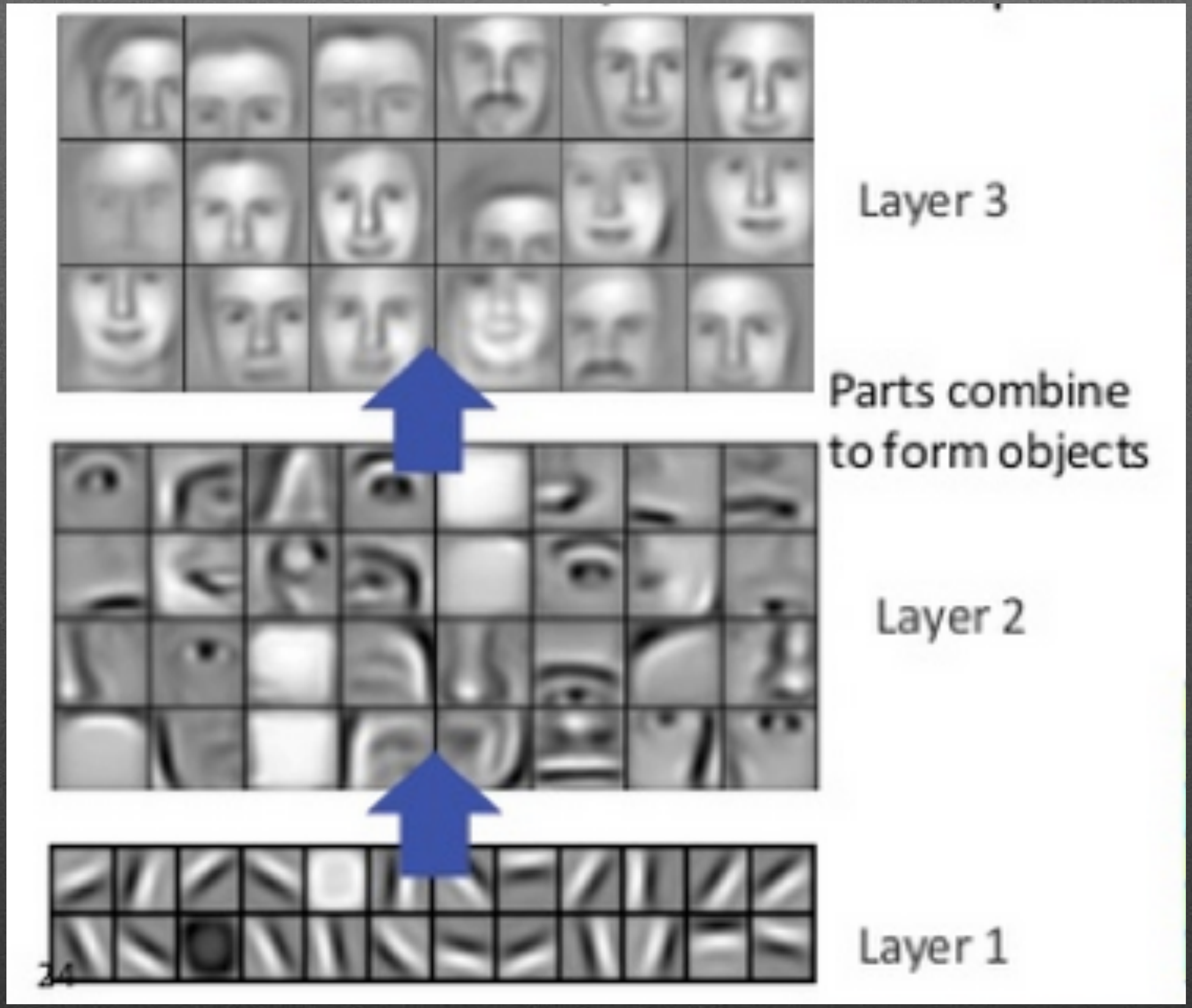


output = $f(W_2 . f(W_1 . \text{Input} + B_1) + B_2)$

Going deeper



What do the layers do?



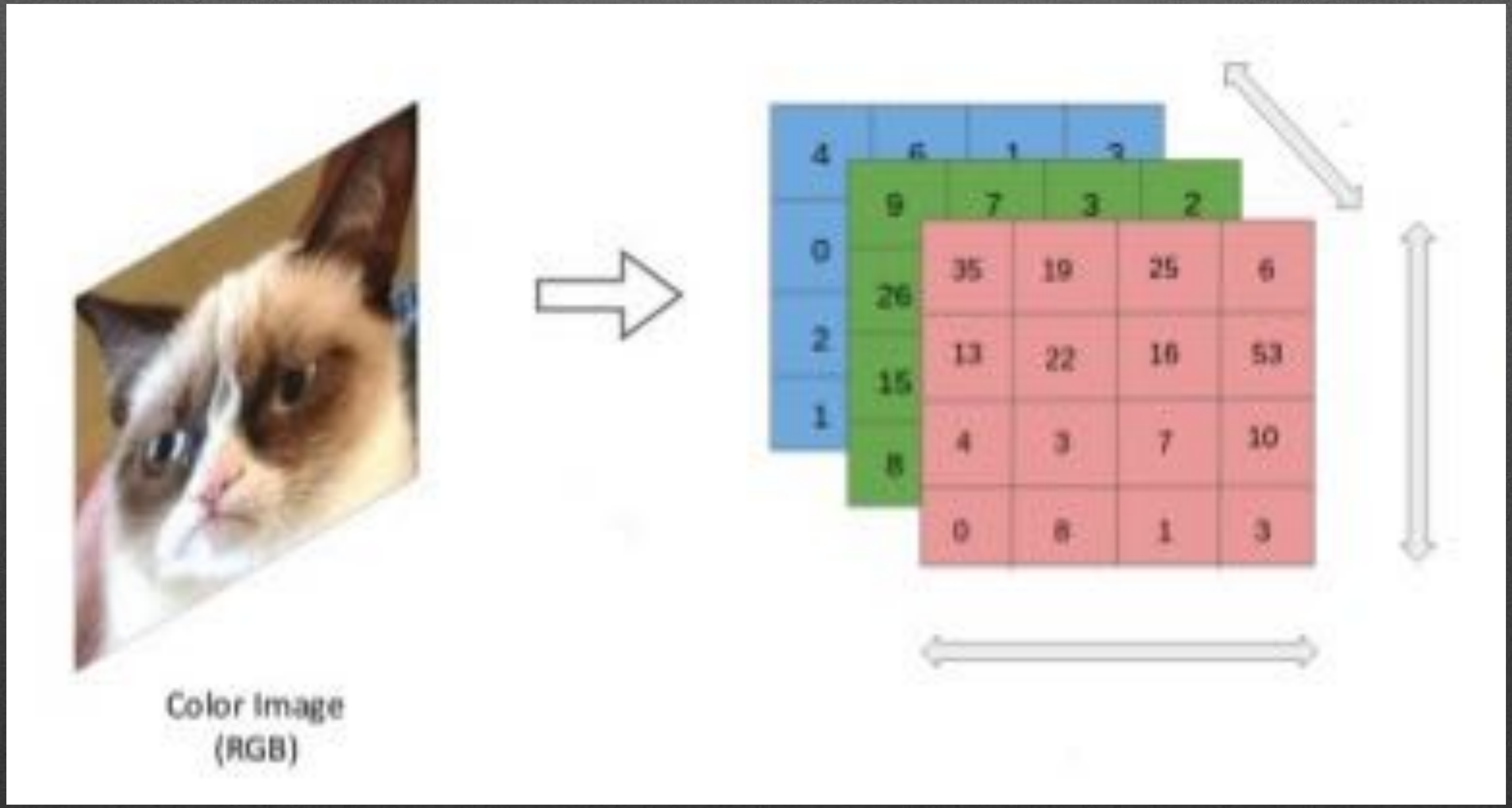
Successive layers model higher level features

What input can a network accept?

- Anything you like as long as it’s a tensor
- Tensor = general multi-dimensional numeric quantity
  - scalar = tensor of 0 dimensions (AKA rank 0)
  - vector = 1 dimensional tensor (rank 1)
  - matrix = 2 dimensional tensor (rank 2)
  - tensor = N dimensional tensor (rank > 2)

Images

Can represent image as tensor of rank 3



Source: https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors

One-hot encoding : input

"enums"

	FAVOURITE PROGRAMMING LANGUAGE			
	JAVA	CLOJURE	PYTHON	JAVASCRIPT
BARRY	1	0	0	0
BRUCE	0	1	0	0
RUSSEL	0	0	1	0

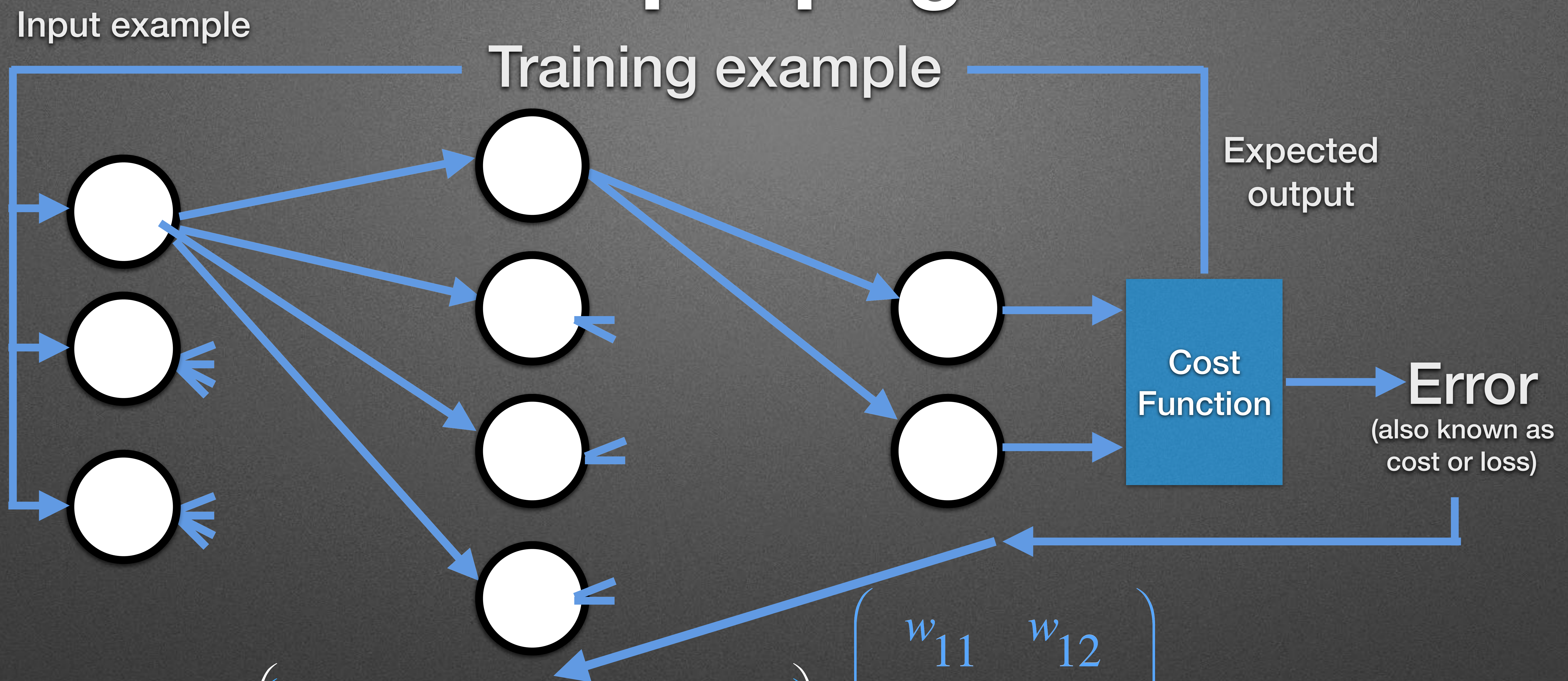
One-hot encoding: output

Also useful for output

Probability distribution

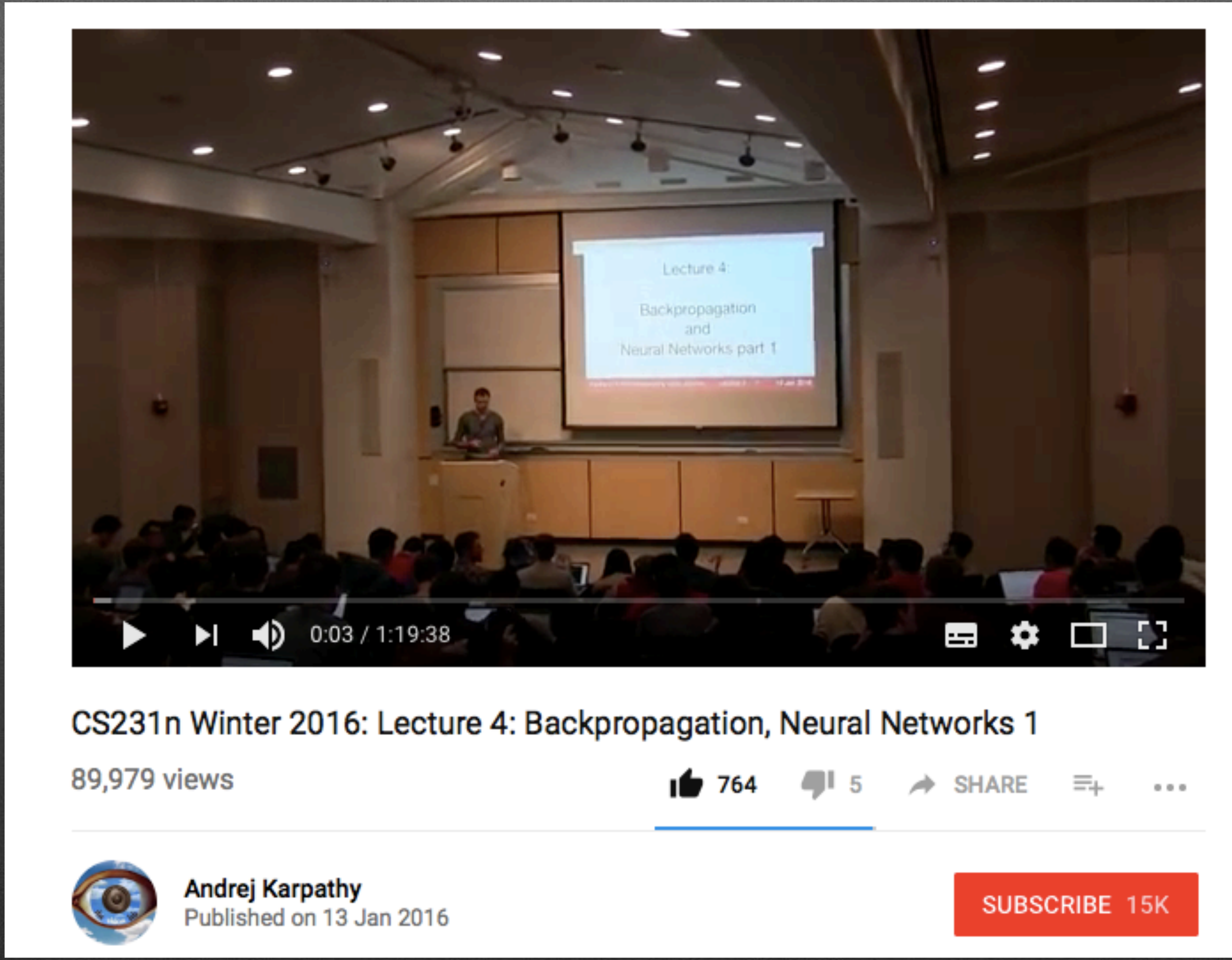
	JAVA	CLOJURE	PYTHON	JAVASCRIPT
BARRY	0.6	0.1	0.1	0.2
BRUCE	0.15	0.75	0.05	0.05
RUSSEL	0.34	0.05	0.6	0.01

Back propagation



$$
\begin{pmatrix}
w_{11} & w_{12} & w_{13} & w_{14} \\
w_{21} & w_{22} & w_{23} & w_{24} \\
w_{31} & w_{32} & w_{33} & w_{34}
\end{pmatrix}
\qquad
\begin{pmatrix}
w_{11} & w_{12} \\
w_{21} & w_{22} \\
w_{31} & w_{32} \\
w_{41} & w_{42}
\end{pmatrix}
$$

More on back propagation



Frameworks

Keras

mxnet

TensorFlow

Caffe2

DL4J
DEEPLEARNING4J

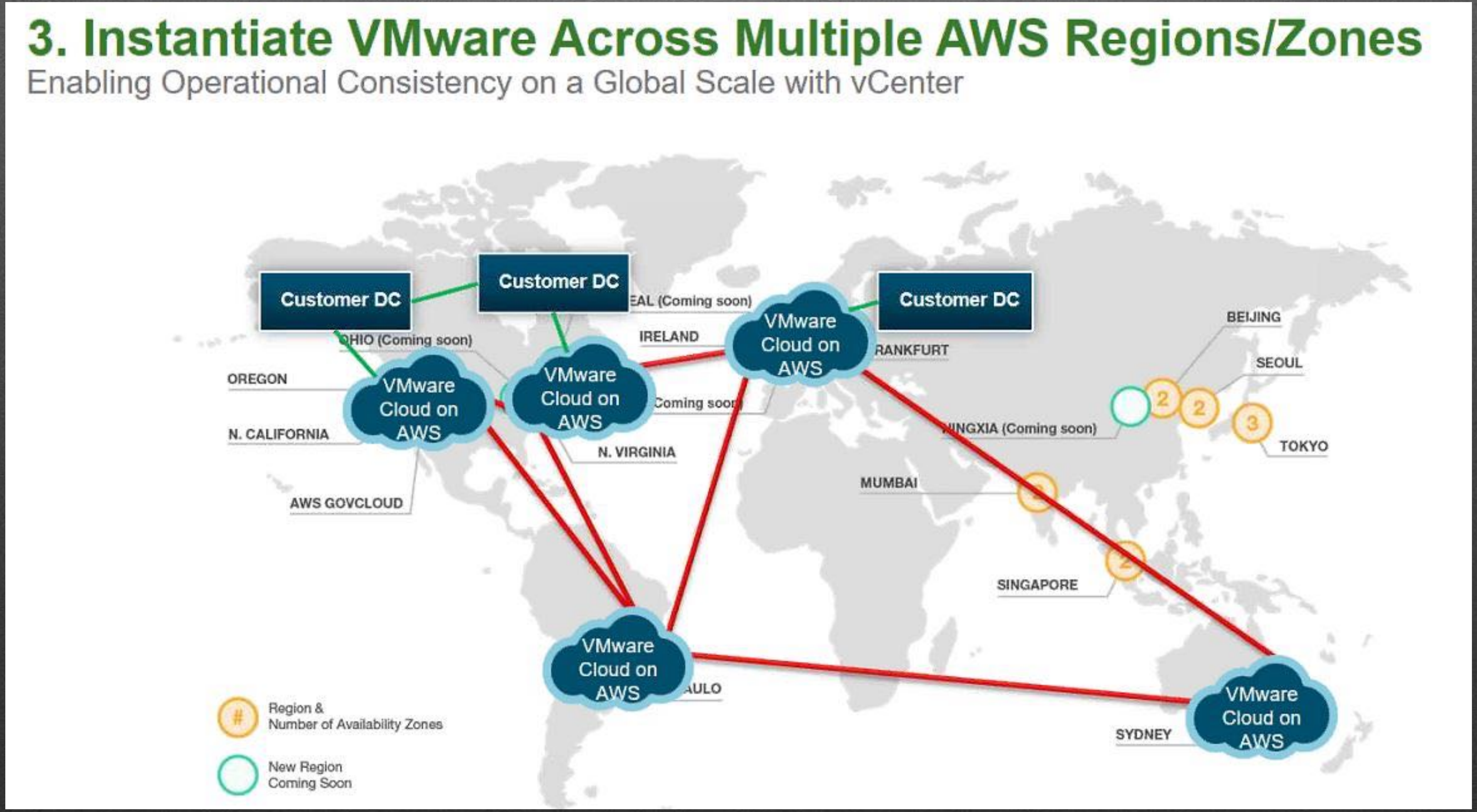
PYTORCH

Summary so far

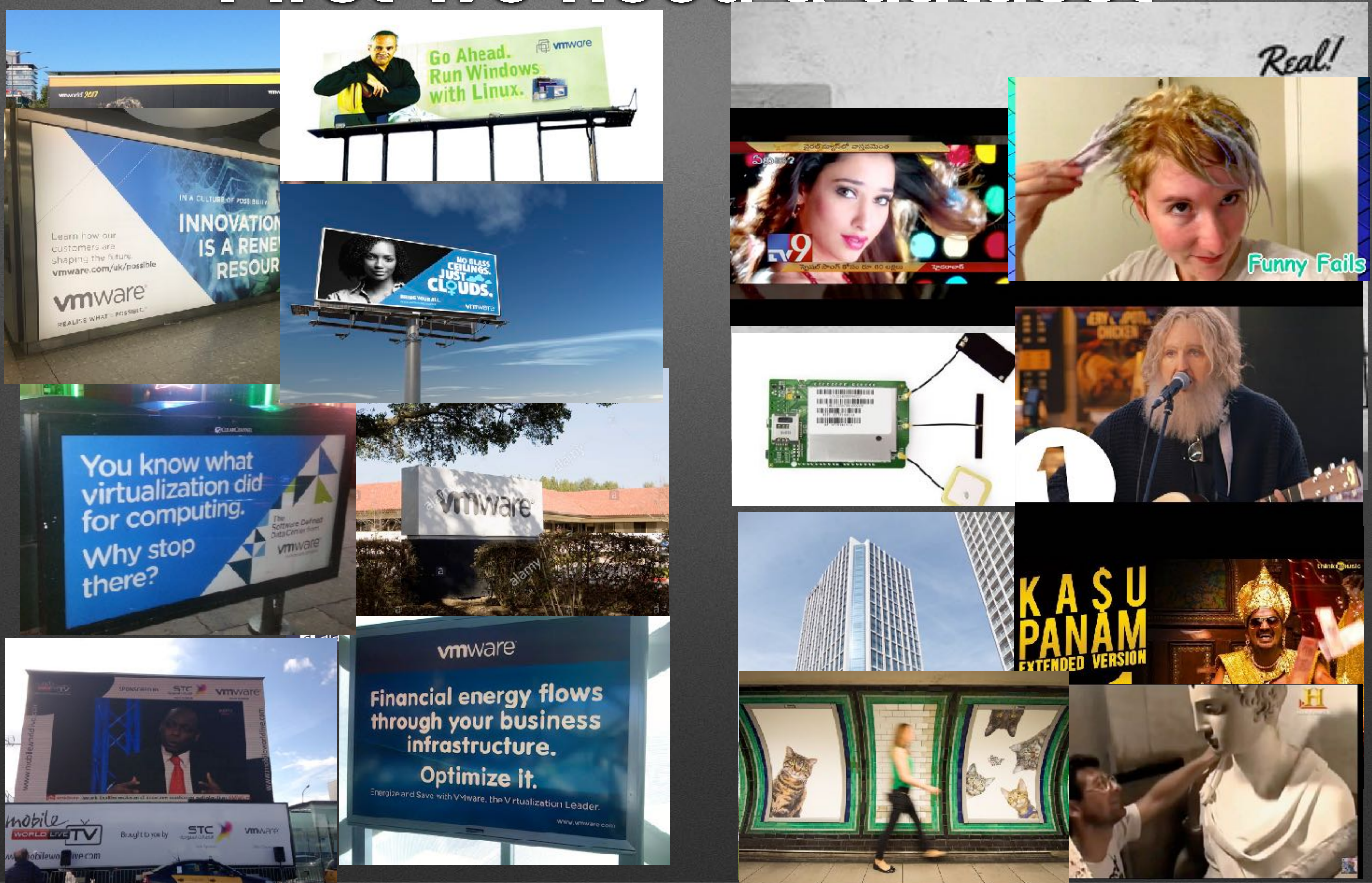
- Neural networks are NOT like your brain
- Networks are arranged as layers
- Forward pass compute output of network
- Backward pass compute gradients & adjust weights
- Frameworks take care of the math for you
    - but still good to understand what’s going on

A request from marketing

Images that mention VMware



First we need a dataset



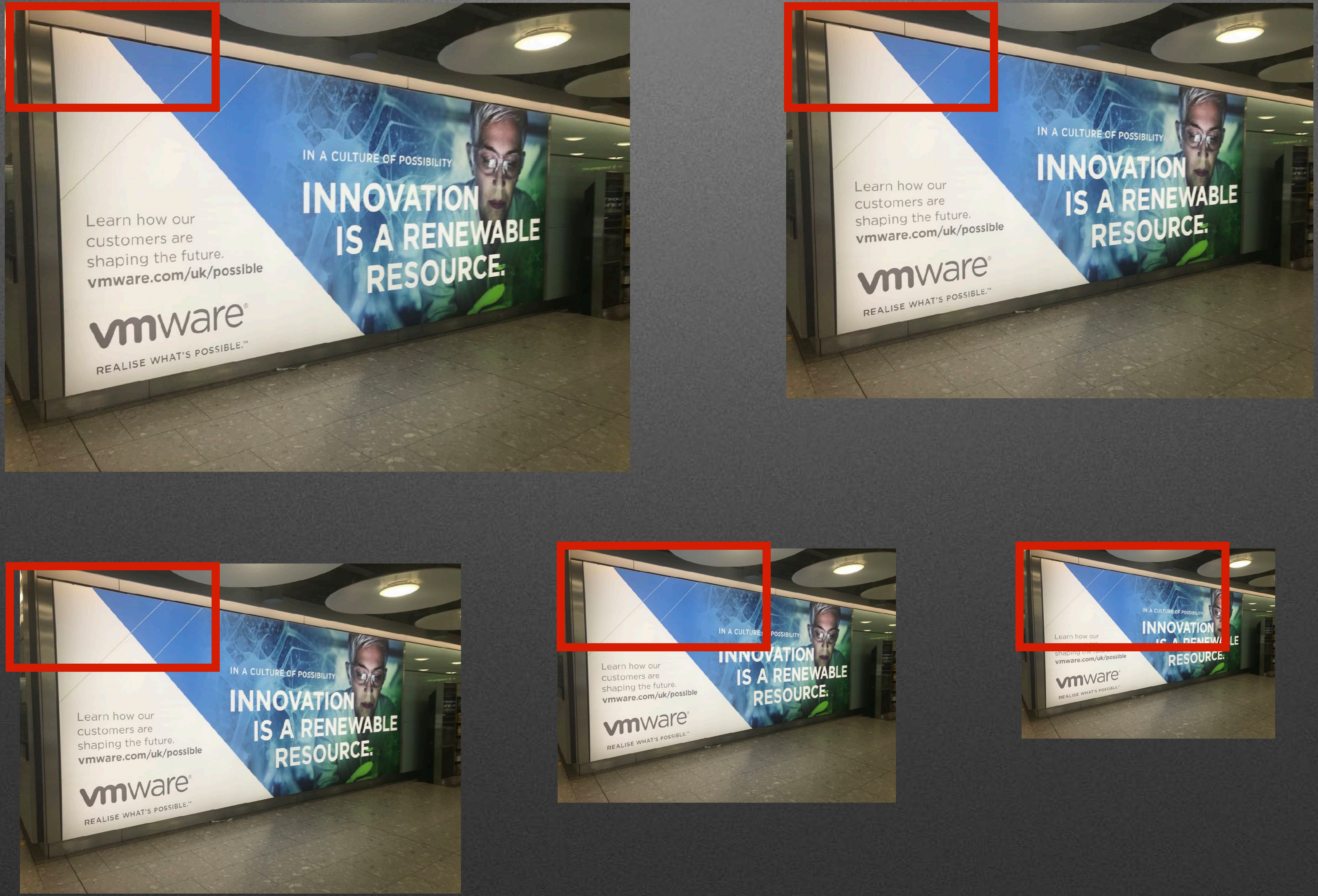
Highlight the parts for training

Creating the dataset

- Grab images from google image search
  - PyImageSearch “How to create a deep learning dataset using Google Images”
- Use dlib imglab tool to draw bounding boxes around logos / not Logos
  - https://github.com/davisking/dlib/tree/master/tools/imglab
- Wrote python script to read imglab XML and produce cropped images using OpenCV

Sliding windows

Multiple scales

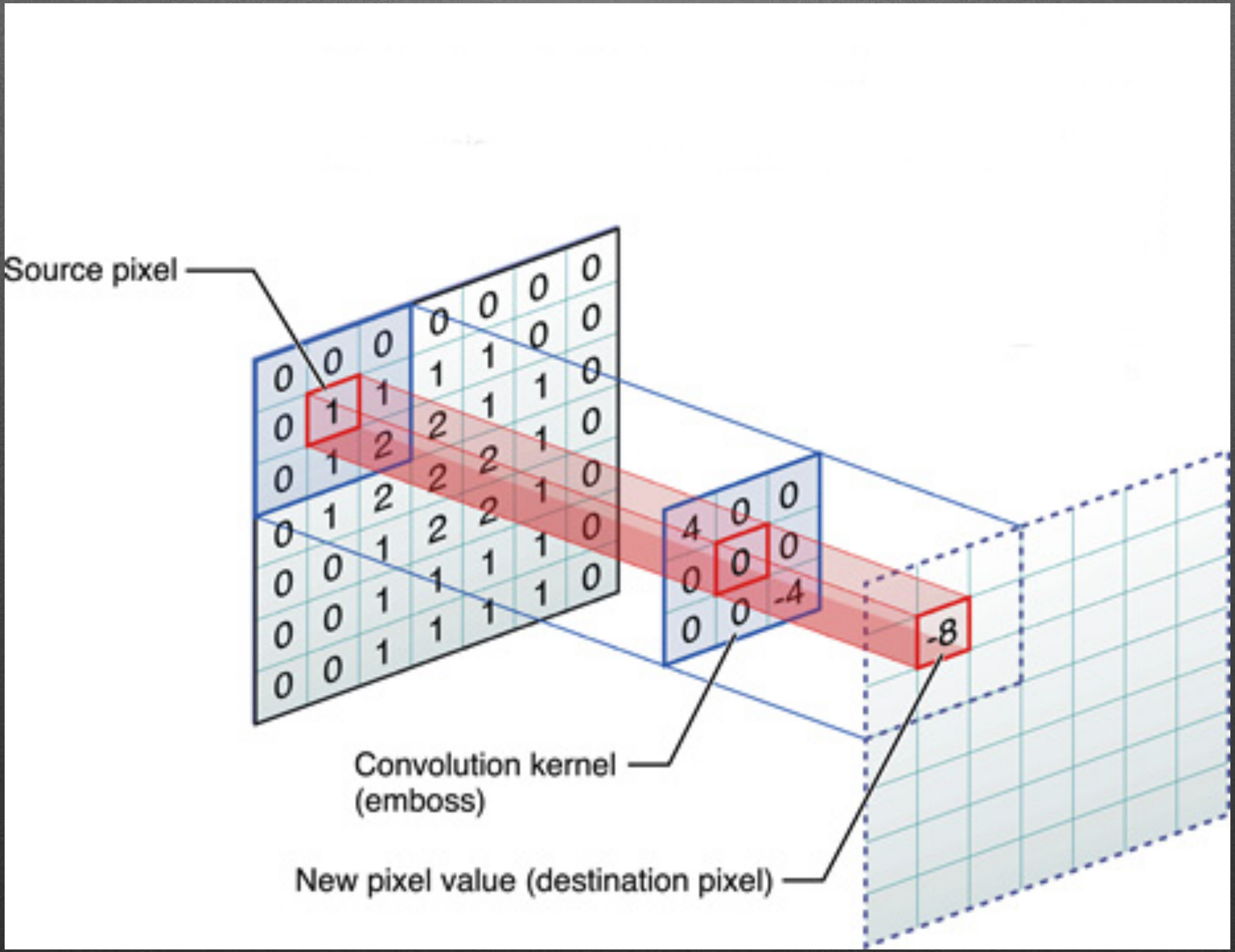


How it all adds up

- 5501 images total
  - 883 VMware
  - 4318 not VMware
- Scaled to 75x22x3 -> 4950 inputs
- Easily 4,950,000 weights in first layer alone
- Maybe we need another neural network architecture

Convolutional Neural Networks

Convolution



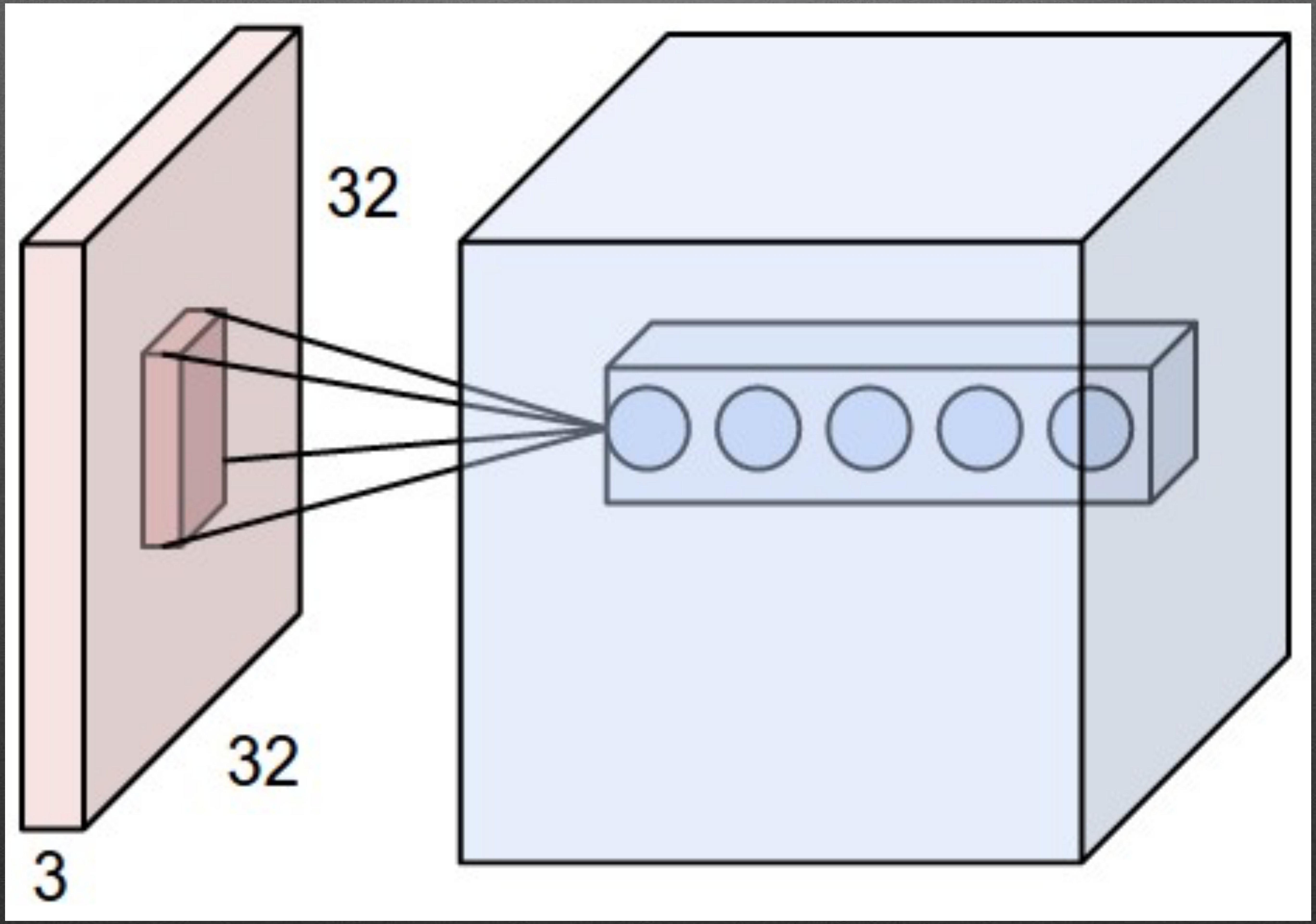
Convolution example(s)

1	1	1
0	0	0
-1	-1	-1

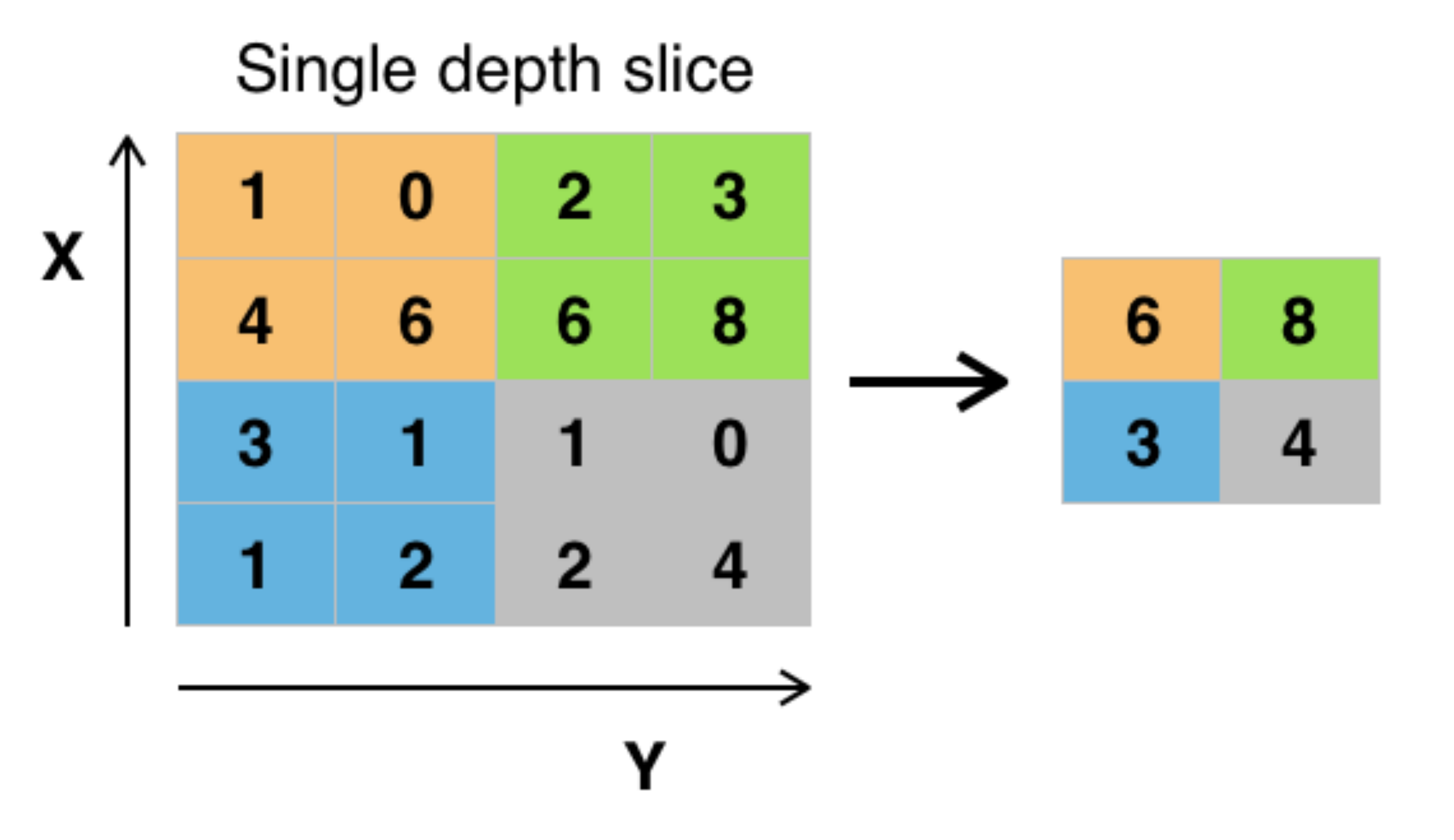
-1	-1	-1
-1	8	-1
-1	-1	-1

1	0	-1
1	0	-1
1	0	-1

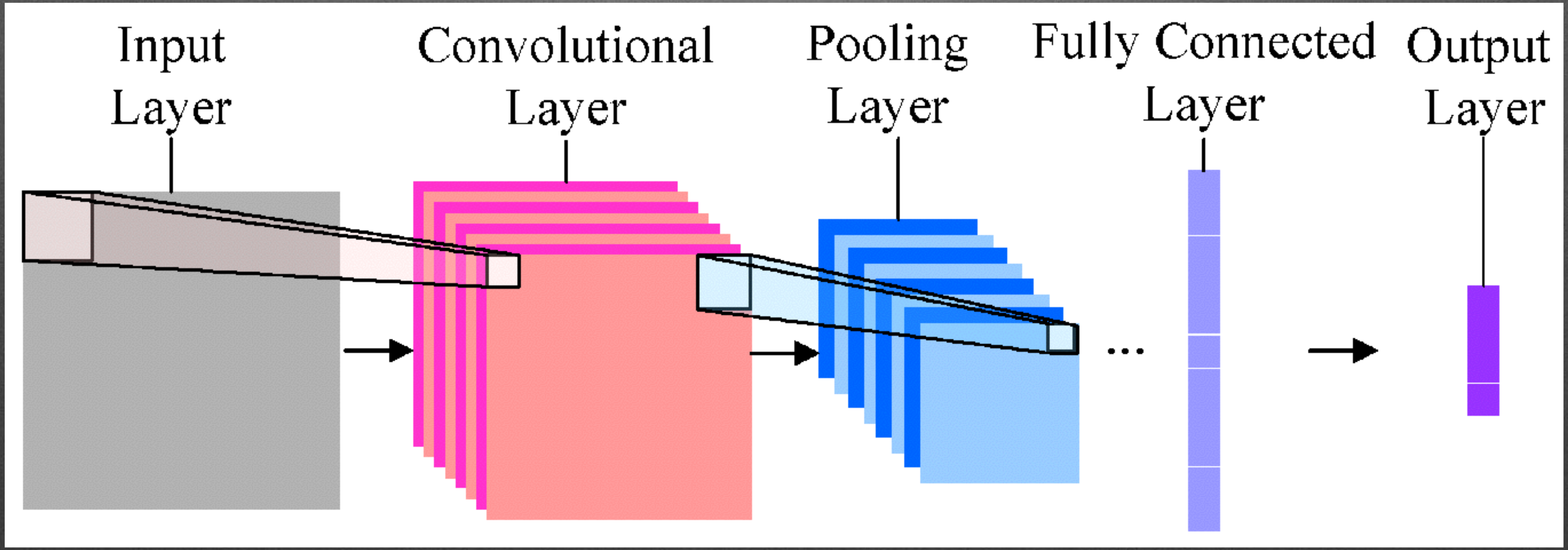
Convolutional layer



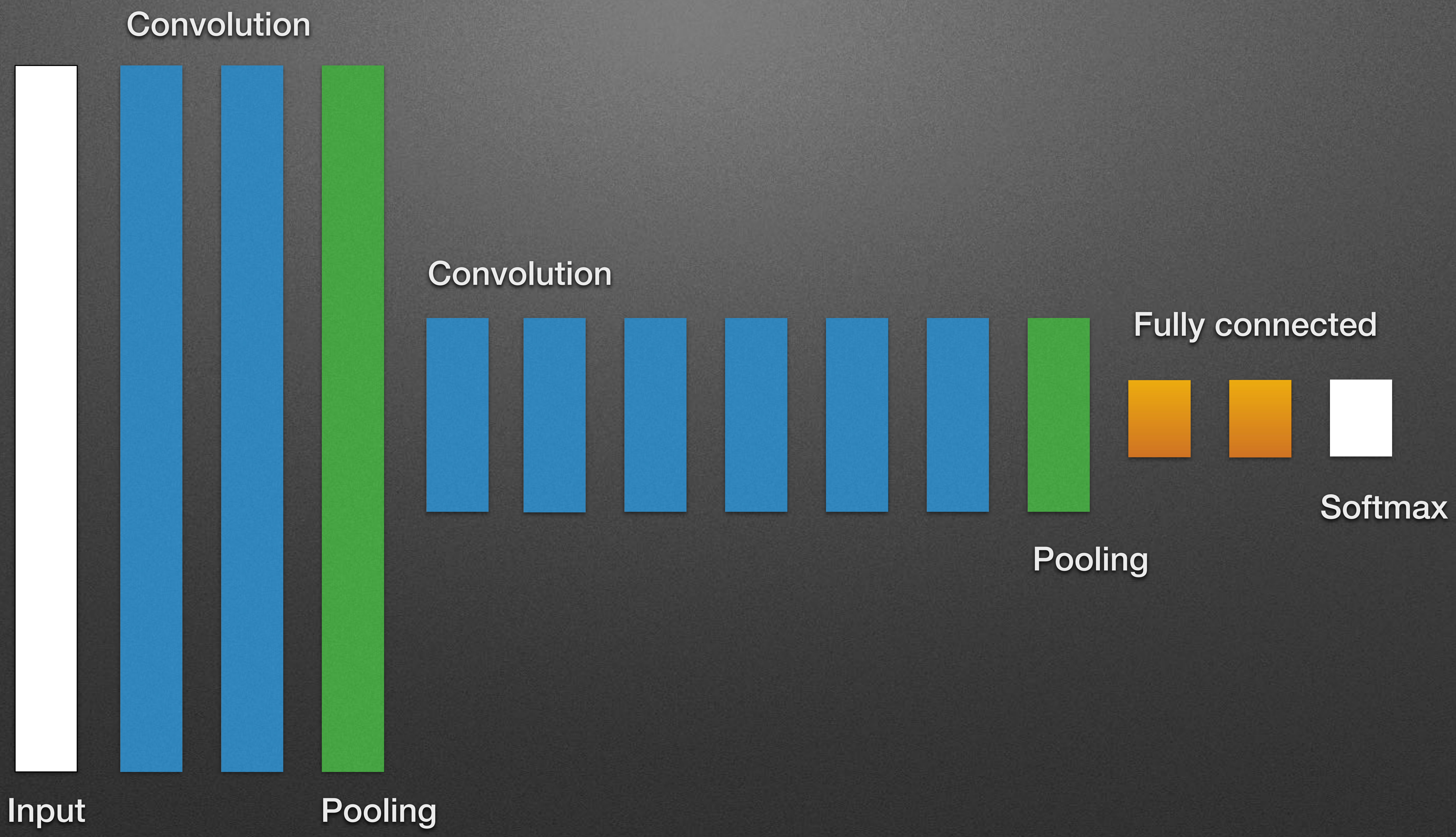
Max Pooling layer



Convolutional network



DL4J model structure



Define the model

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .cacheMode(CacheMode.DEVICE)
    .updater(Updater.ADAM)
    .iterations(iterations)
    .gradientNormalization(GradientNormalization.RenormalizeL2PerLayer) // normalize to prevent vanishing or exploding gradients
    .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
    .l1(1e-4)
    .regularization(true)
    .l2(5 * 1e-4)
    .list()
    .layer(0, new ConvolutionLayer.Builder(new int[]{4, 4}, new int[]{1, 1}, new int[]{0, 0}).name("cnn1").convolutionMode(ConvolutionMode.Same)
            .nIn(3).nOut(64).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)//.learningRateDecayPolicy(LearningRatePolicy.Step)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(1, new ConvolutionLayer.Builder(new int[]{4, 4}, new int[]{1, 1}, new int[]{0, 0}).name("cnn2").convolutionMode(ConvolutionMode.Same)
            .nOut(64).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(2, new SubsamplingLayer.Builder(PoolingType.MAX, new int[]{2, 2}).name("maxpool2").build())

    .layer(3, new ConvolutionLayer.Builder(new int[]{4, 4}, new int[]{1, 1}, new int[]{0, 0}).name("cnn3").convolutionMode(ConvolutionMode.Same)
            .nOut(96).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(4, new ConvolutionLayer.Builder(new int[]{4, 4}, new int[]{1, 1}, new int[]{0, 0}).name("cnn4").convolutionMode(ConvolutionMode.Same)
            .nOut(96).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())

    .layer(5, new ConvolutionLayer.Builder(new int[]{3, 3}, new int[]{1, 1}, new int[]{0, 0}).name("cnn5").convolutionMode(ConvolutionMode.Same)
            .nOut(128).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(6, new ConvolutionLayer.Builder(new int[]{3, 3}, new int[]{1, 1}, new int[]{0, 0}).name("cnn6").convolutionMode(ConvolutionMode.Same)
            .nOut(128).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())

    .layer(7, new ConvolutionLayer.Builder(new int[]{2, 2}, new int[]{1, 1}, new int[]{0, 0}).name("cnn7").convolutionMode(ConvolutionMode.Same)
            .nOut(256).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(8, new ConvolutionLayer.Builder(new int[]{2, 2}, new int[]{1, 1}, new int[]{0, 0}).name("cnn8").convolutionMode(ConvolutionMode.Same)
            .nOut(256).weightInit(WeightInit.XAVIER_UNIFORM).activation(Activation.RELU)
            .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(9, new SubsamplingLayer.Builder(PoolingType.MAX, new int[]{2, 2}).name("maxpool8").build())

    .layer(10, new DenseLayer.Builder().name("ffn1").nOut(1024).learningRate(1e-3).biasInit(1e-3).biasLearningRate(1e-3 * 2).build())
    .layer(11, new DropoutLayer.Builder().name("dropout1").dropOut(0.2).build())
    .layer(12, new DenseLayer.Builder().name("ffn2").nOut(1024).learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2).build())
    .layer(13, new DropoutLayer.Builder().name("dropout2").dropOut(0.2).build())
    .layer(14, new OutputLayer.Builder(LossFunctions.LossFunction.Not-VMwareLOGLIKELIHOOD)
            .name("output")
            .nOut(numLabels)
            .activation(Activation.SOFTMAX)
            .build())
    .backprop(true)
    .pretrain(false)
    .setInputType(InputType.convolutional(height, width, channels))
    .build();

MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
```

code › cnn › src › main › java › com › davesnowdon › qcon › LogoFinder

LogoFinder.java | Word2VecSentimentRNN.java | SentimentExampleIterator.java

```java
    public MultiLayerNetwork createModel() {
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(seed)
                .cacheMode(CacheMode.DEVICE)
                .updater(Updater.ADAM)
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
                .iterations(iterations)
                // normalize to prevent vanishing or exploding gradients
                .gradientNormalization(GradientNormalization.RenormalizeL2PerLayer)
                .l1(1e-4)
                .regularization(true)
                .l2(5 * 1e-4)
                .list()

                .layer( ind: 0, new ConvolutionLayer.Builder(new int[]{4, 4}, new int[]{1,
                        .name("cnn1").convolutionMode(ConvolutionMode.Same)
                        .nIn(3).nOut(64)
                        .weightInit(WeightInit.XAVIER_UNIFORM)
                        .activation(Activation.RELU)
                        .learningRate(1e-2).biasInit(1e-2).biasLearningRate(1e-2 * 2)
                        .build())
```

LogoFinder › createAndTrainModel()

6: TODO | 9: Version Control | Terminal | Spring | 2 Event Log

Outdated Kotlin Runtime: Version of Kotlin runtime is outdated in several libraries (Gradle: org.jetbrains.kotlin:kotlin-stdlib:1.0.7, Gradle: org.jetbrains.kotlin:kotlin-runtime:1.0.7). Plugi... (yesterday 18:24) 159:73 LF÷ UTF-8÷ Git: master÷

Results

- 10 epochs, 16 minutes to train on NVIDIA GTX 1080
- Inference time: ~20ms
- Precision 0.9661
- Recall 0.8829

		PREDICTED	
		VMWARE	NOT-VMWARE
ACTUAL	VMWARE	140	42
	NOT-VMWARE	3	855

Results

VMware

not-VMware

More efficient object detection

- You Only Look Once (YOLO)
- Single Shot Multibox Detector (SSD)
- Faster R-CNN

“Building a Production Grade Object Detection System with SKIL and YOLO”

https://blog.skymind.ai/building-a-production-grade-object-detection-system-with-skil-and-yolo/

Summary so far

Convolutional networks

- Used mostly for image processing
- Convolution layer applies learnt filter to inputs
- Pooling layer reduces size of inputs
- Fewer weights (parameters) to train compared to fully connected networks

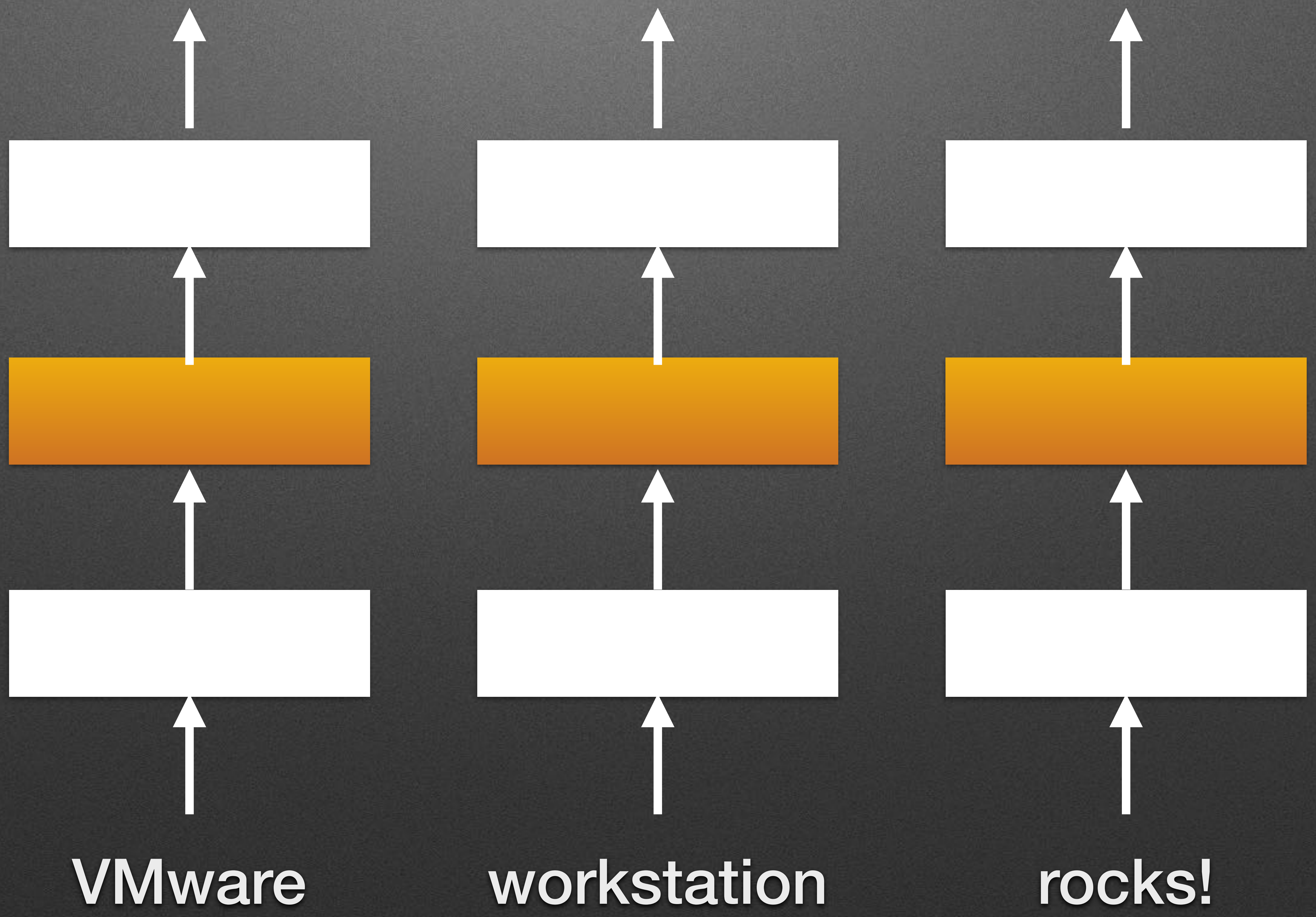
But, are they saying nice things about us?

Variable length input

VMware	workstation	rocks!

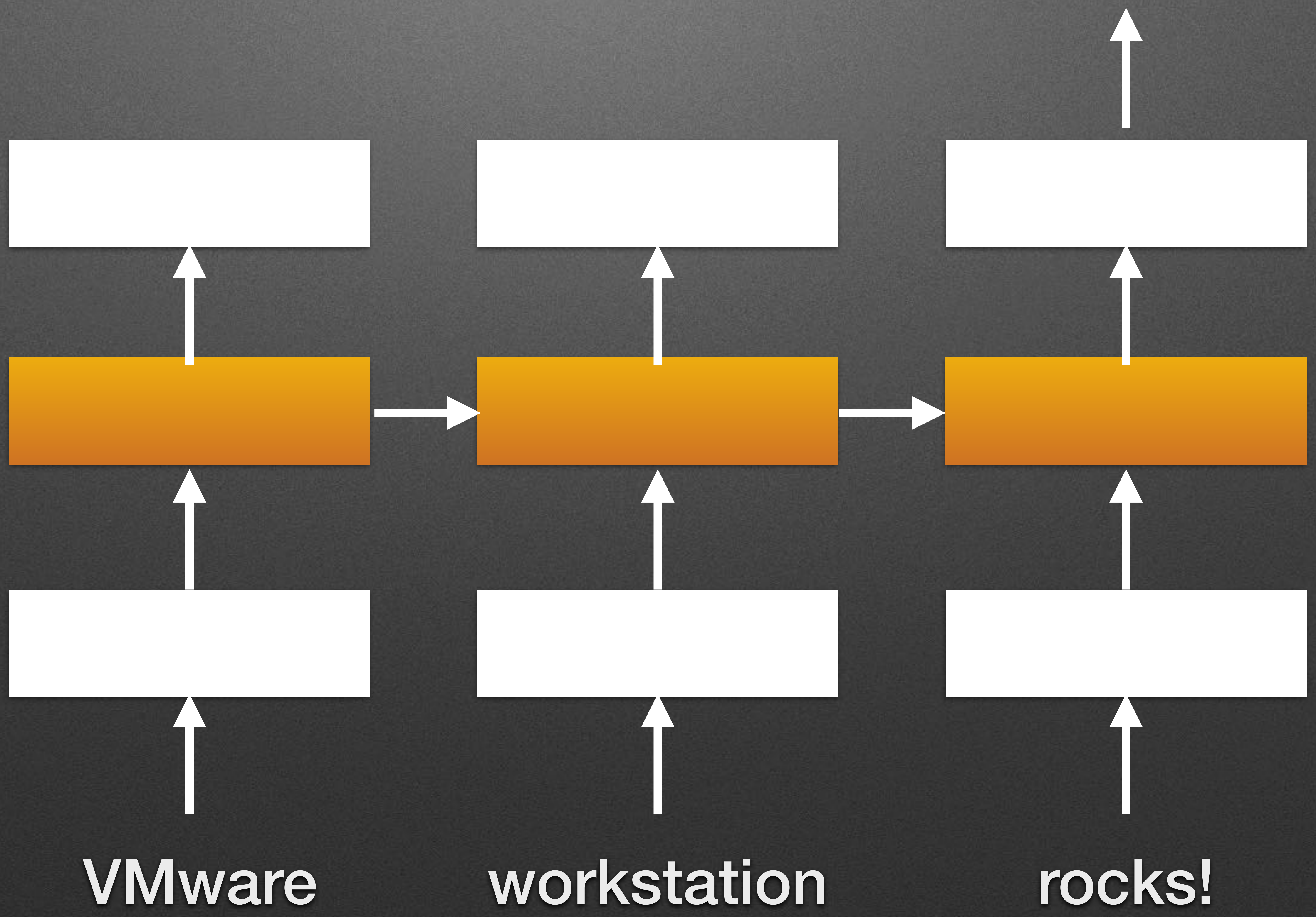
I’ve	used	vSphere	for	a	number	of	years	…

Feed forward networks don’t remember state

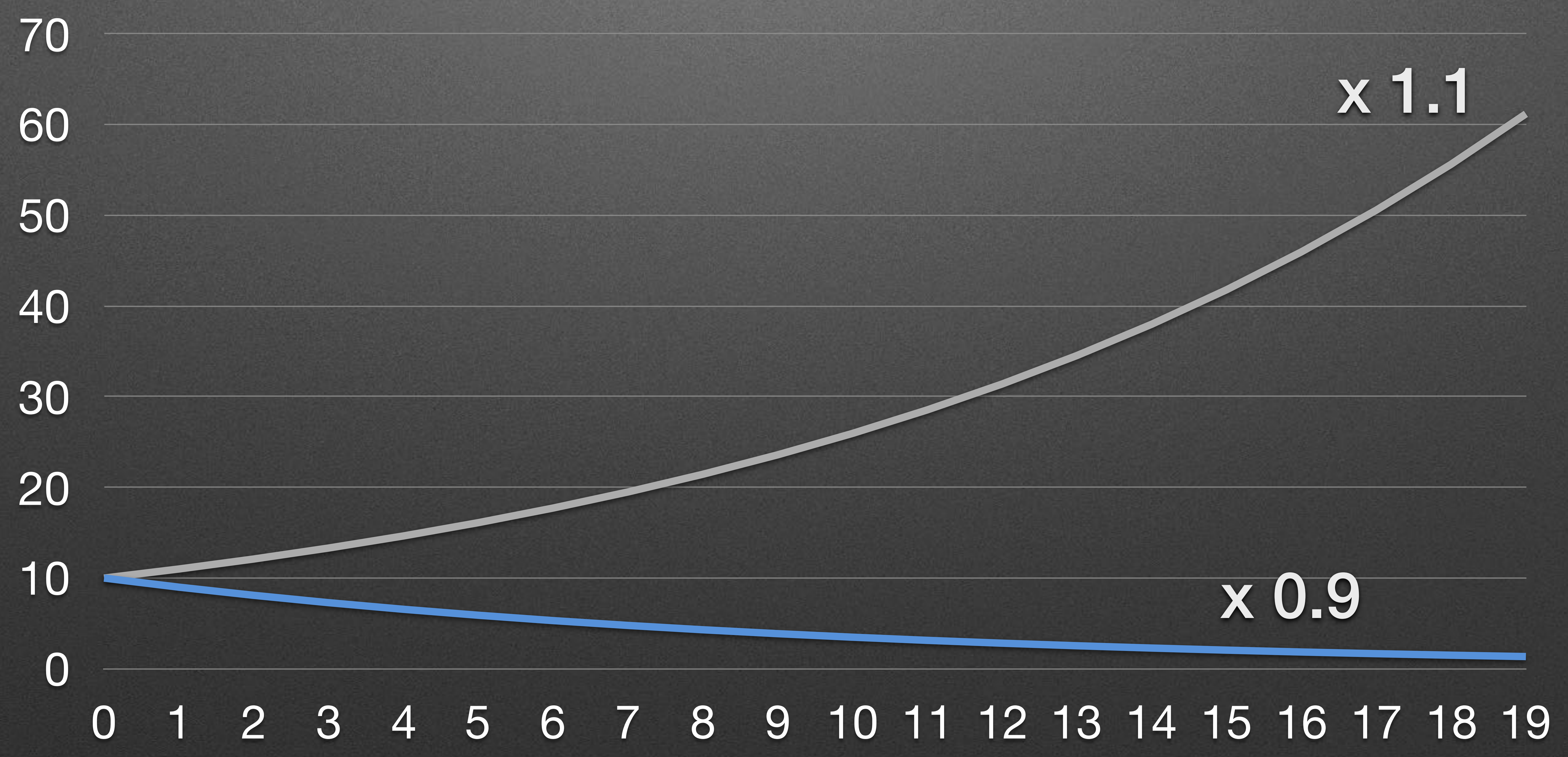


Recurrent neural networks

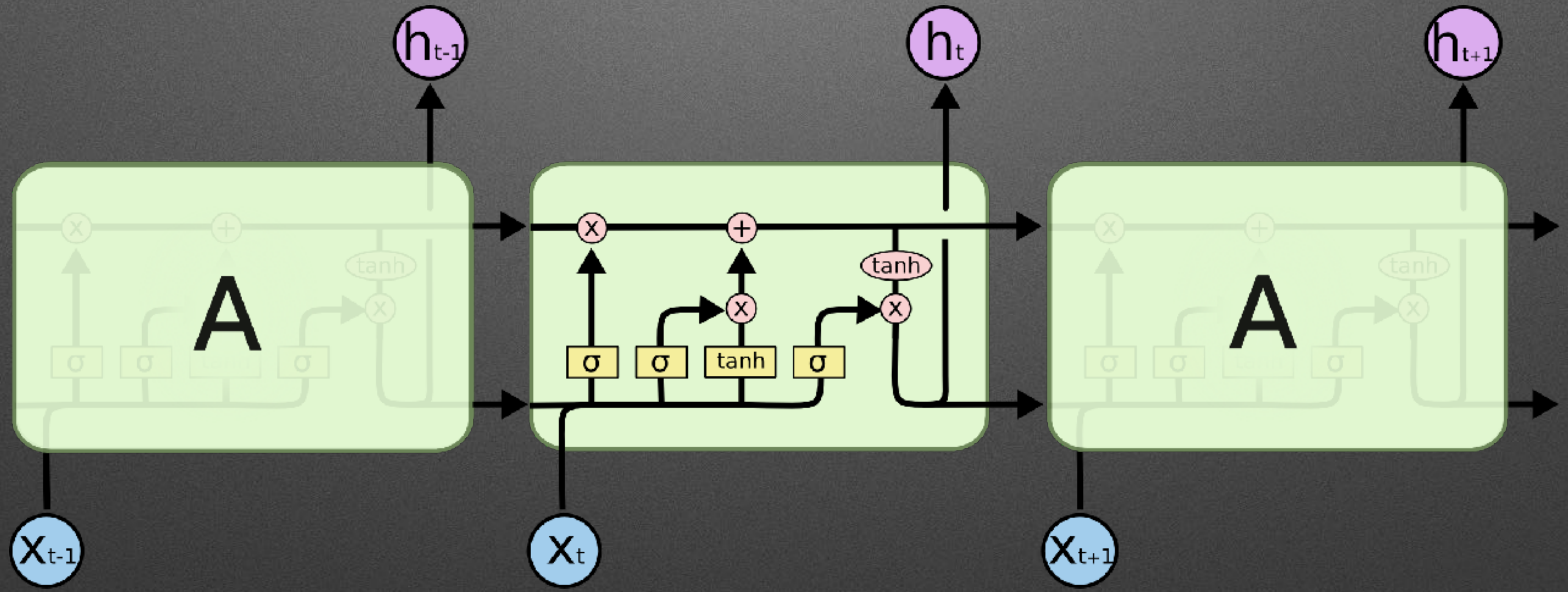
Recurrent network



Vanishing/exploding gradients



Long Short-Term Memory (LSTM) cell



Stanford large movie review dataset

http://ai.stanford.edu/~amaas/data/sentiment/

Homelessness (or Houselessness as George Carlin stated) has been an issue for years but never a plan to help those on the street that were once considered human who did everything from going to school, …

This is the biggest Flop of 2008. I don know what Director has is his mind of creating such a big disaster. The songs have been added without situations, the story have been stretched to fill the 3 hrs gap …

Words -> vectors

cat

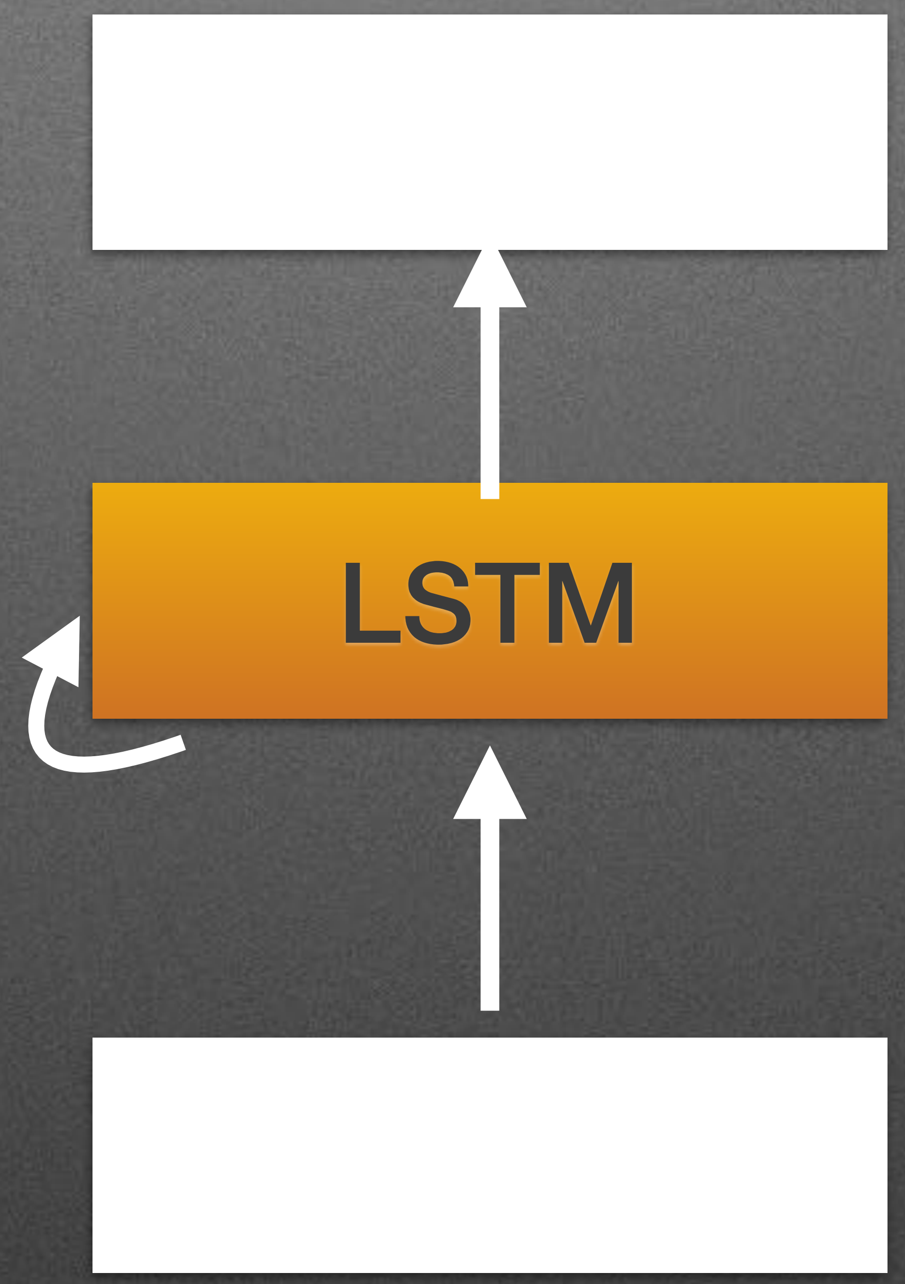
1 hot, 3 million values

0	0	0	0	0	0	0	1	0	0	0	0	0	0	…

word2vec, 300 values

0.67	5.4	0.2	0.46	15	69	23	…

DL4J model



The code

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .seed(seed)
        .updater(Updater.ADAM)  //To configure: .updater(Adam.builder().beta1(0.9).beta2(0.999).build())
        .regularization(true).l2(1e-5)
        .weightInit(WeightInit.XAVIER)
        .gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue).gradientNormalizationThreshol
d(1.0)
        .learningRate(2e-2)
        .trainingWorkspaceMode(WorkspaceMode.SEPARATE).inferenceWorkspaceMode(WorkspaceMode.SEPARAT
E)
        .list()
        .layer(0, new GravesLSTM.Builder().nIn(vectorSize).nOut(256)
                .activation(Activation.TANH).build())
        .layer(1, new RnnOutputLayer.Builder().activation(Activation.SOFTMAX)
                .lossFunction(LossFunctions.LossFunction.MCXENT).nIn(256).nOut(2).build())
        .pretrain(false).backprop(true).build();

MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.init();
```

le › word2vecsentiment › src › main › java › org › deeplearning4j › examples › recurrent › word2vecsentiment › Word2VecSentimentRNN

LogoFinder.java × | Word2VecSentimentRNN.java × | SentimentExampleIterator.java ×

```java
        //Set up network configuration
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(seed)
                .updater(Updater.ADAM)
                .regularization(true).l2(1e-5)
                .weightInit(WeightInit.XAVIER)
                .gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue).
                .learningRate(2e-2)
                .trainingWorkspaceMode(WorkspaceMode.SEPARATE).inferenceWorkspaceMode(Works
                .list()
                .layer( ind: 0, new GravesLSTM.Builder()
                        .nIn(vectorSize).nOut(256)
                        .activation(Activation.TANH)
                        .build())

                .layer( ind: 1, new RnnOutputLayer.Builder()
                        .activation(Activation.SOFTMAX)
                        .lossFunction(LossFunctions.LossFunction.MCXENT)
                        .nIn(256).nOut(2)
                        .build())

```

Word2VecSentimentRNN › createAndTrainModel()

6: TODO | 9: Version Control | Terminal | Spring | 2 Event Log

Outdated Kotlin Runtime: Version of Kotlin runtime is outdated in several libraries (Gradle: org.jetbrains.kotlin:kotlin-stdlib:1.0.7, Gradle: org.jetbrains.kotlin:kotlin-runtime:1.0.7). Plugi... (yesterday 18:24) 163:52 LF: UTF-8: Git: master:

Results

0.976	VMware Horizon rocks!
0.976	I don't like it at all. It does not work the way I think it should
0.0317	It crashes. It's buggy. Don't waste your time on this
0.976	This software is market leading and will change the world

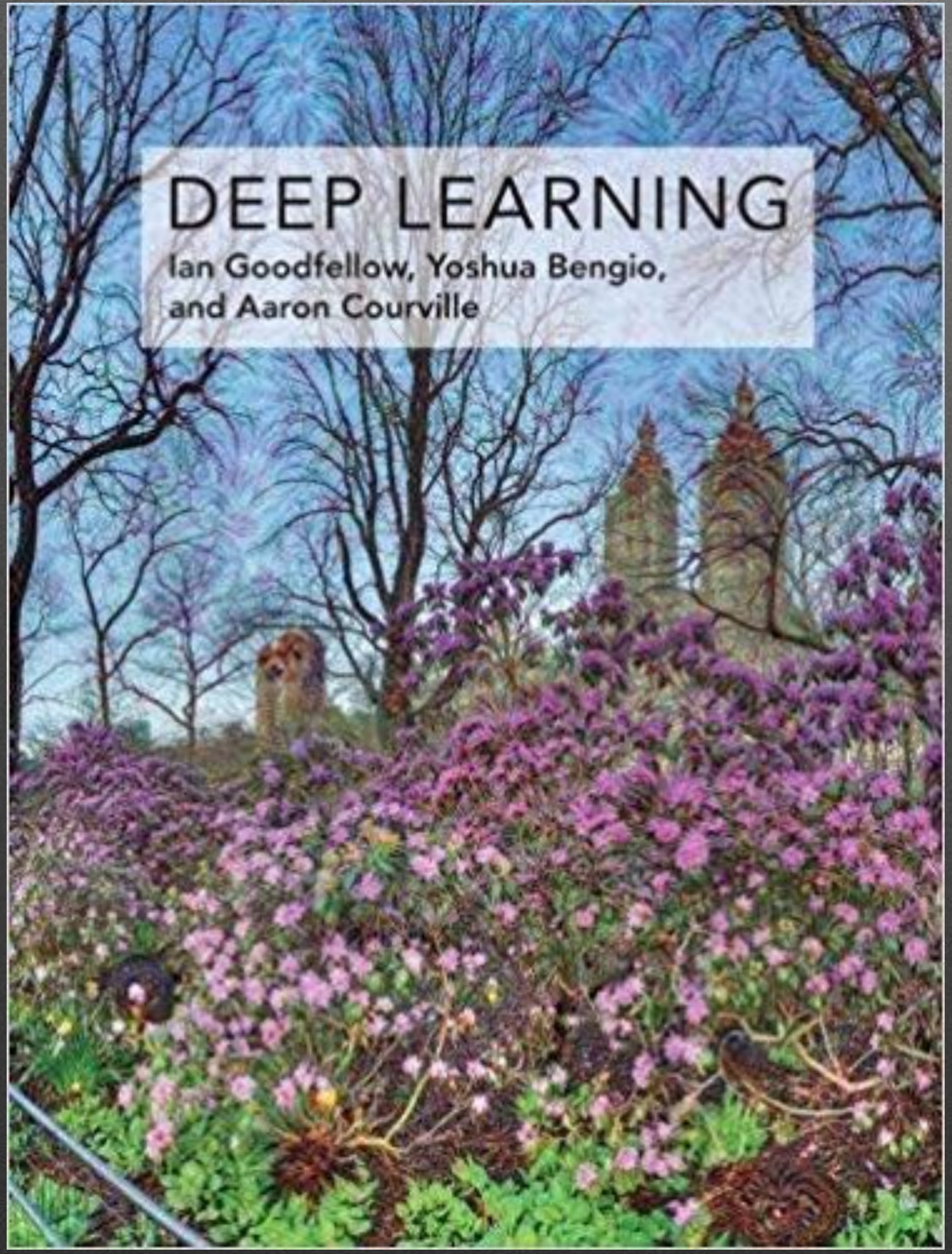
RNN Summary

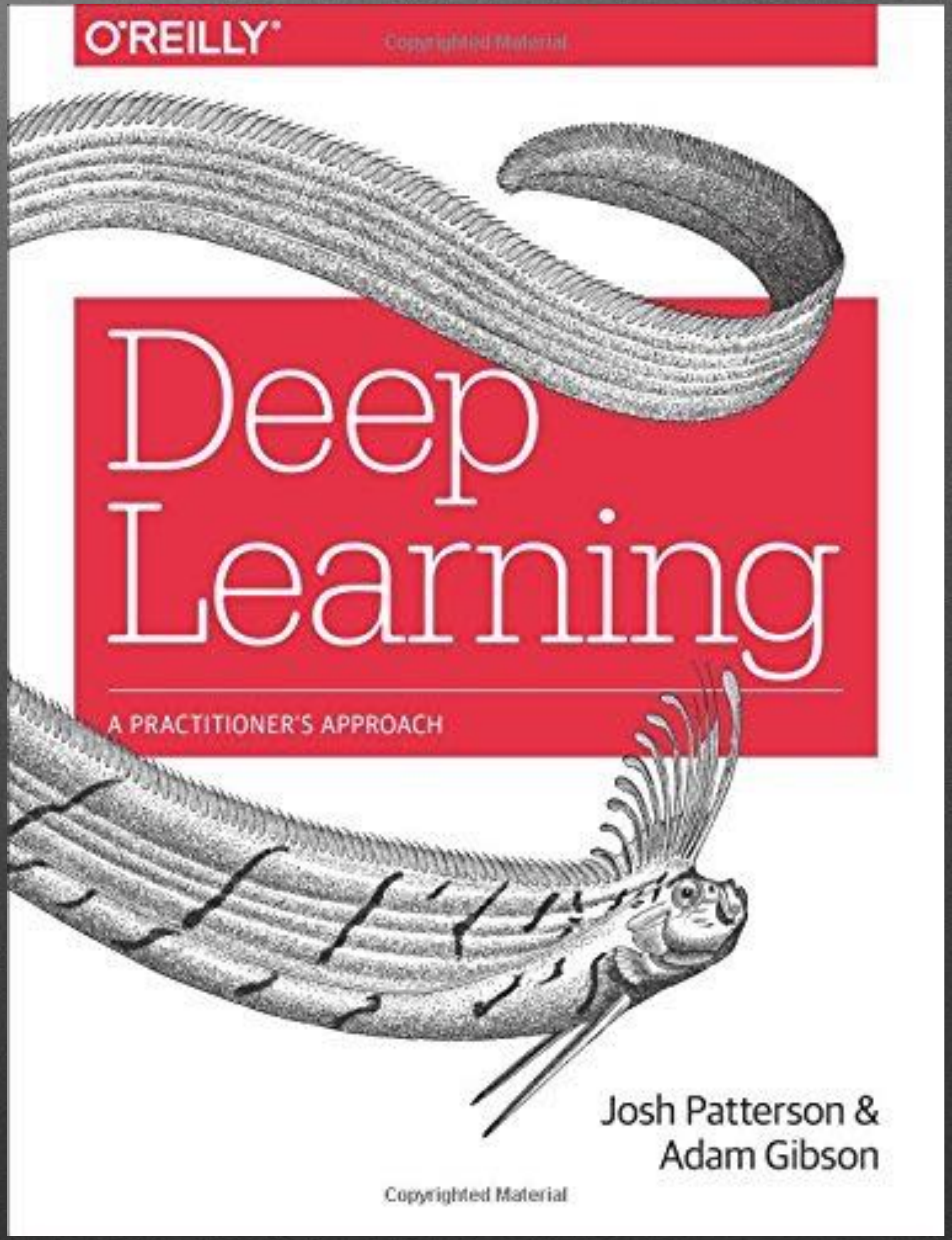
- Use recurrent neural networks (RNNs) for time series or sequential data
- RNNs can consume & generate sequences
- RNNs back propagate through time as well as layers
    - **very deep** networks so increase in training time
- Use LSTM (or GRU) layers

Summary

- Each layer learns features composed of features from previous layer
- Convolutional neural networks (CNN) well suited for images
- Recurrent Neural Networks (RNN) used for time series data
- Can have networks combining both convolutional & recurrent layers

Further info #1



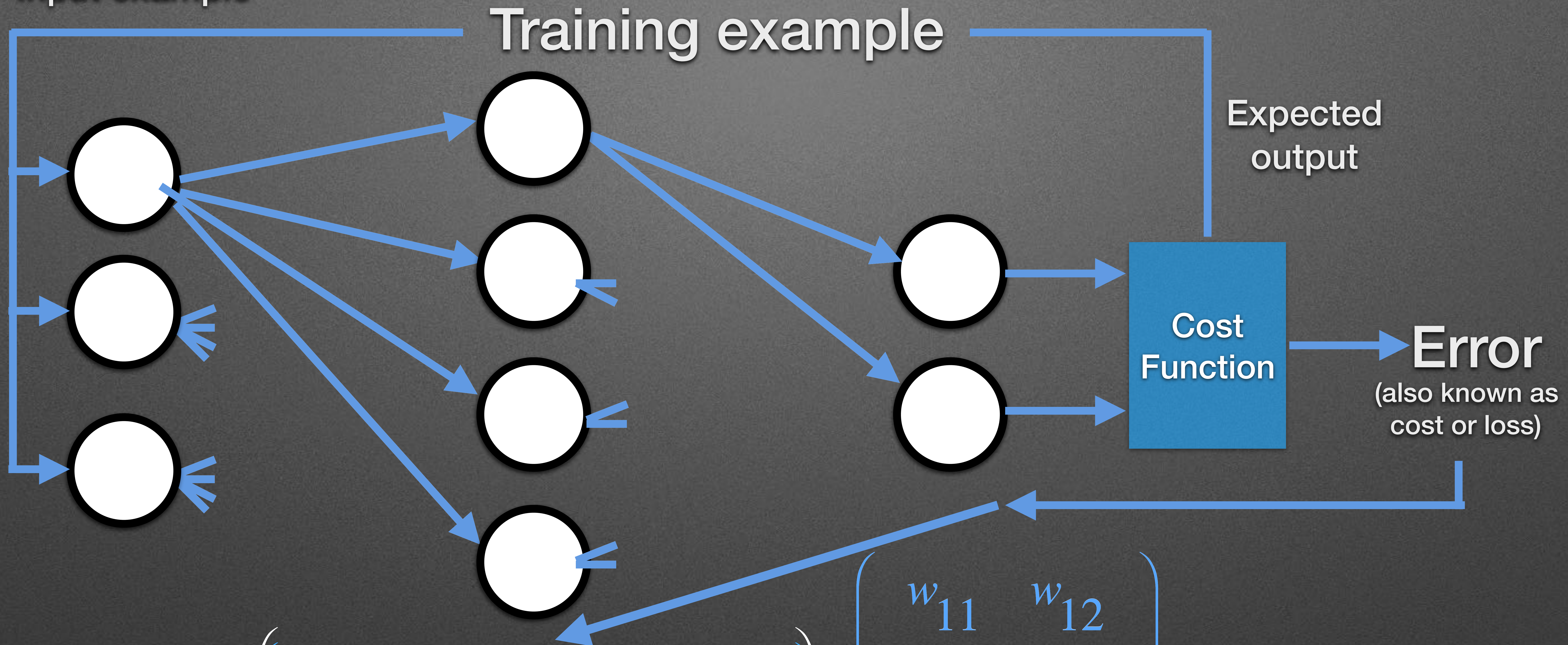


Further info #2

- **Andrew Ng Coursera :** *https://www.coursera.org/specializations/deep-learning*
- **Udacity :** *https://www.udacity.com/course/deep-learning--ud730*
- CS231n Winter 2016 lecture videos
- **Andrej Karpathy's blog :** *http://karpathy.github.io/*
- **Andrew Trask's blog :** *https://iamtrask.github.io/*

Backup / bonus slides

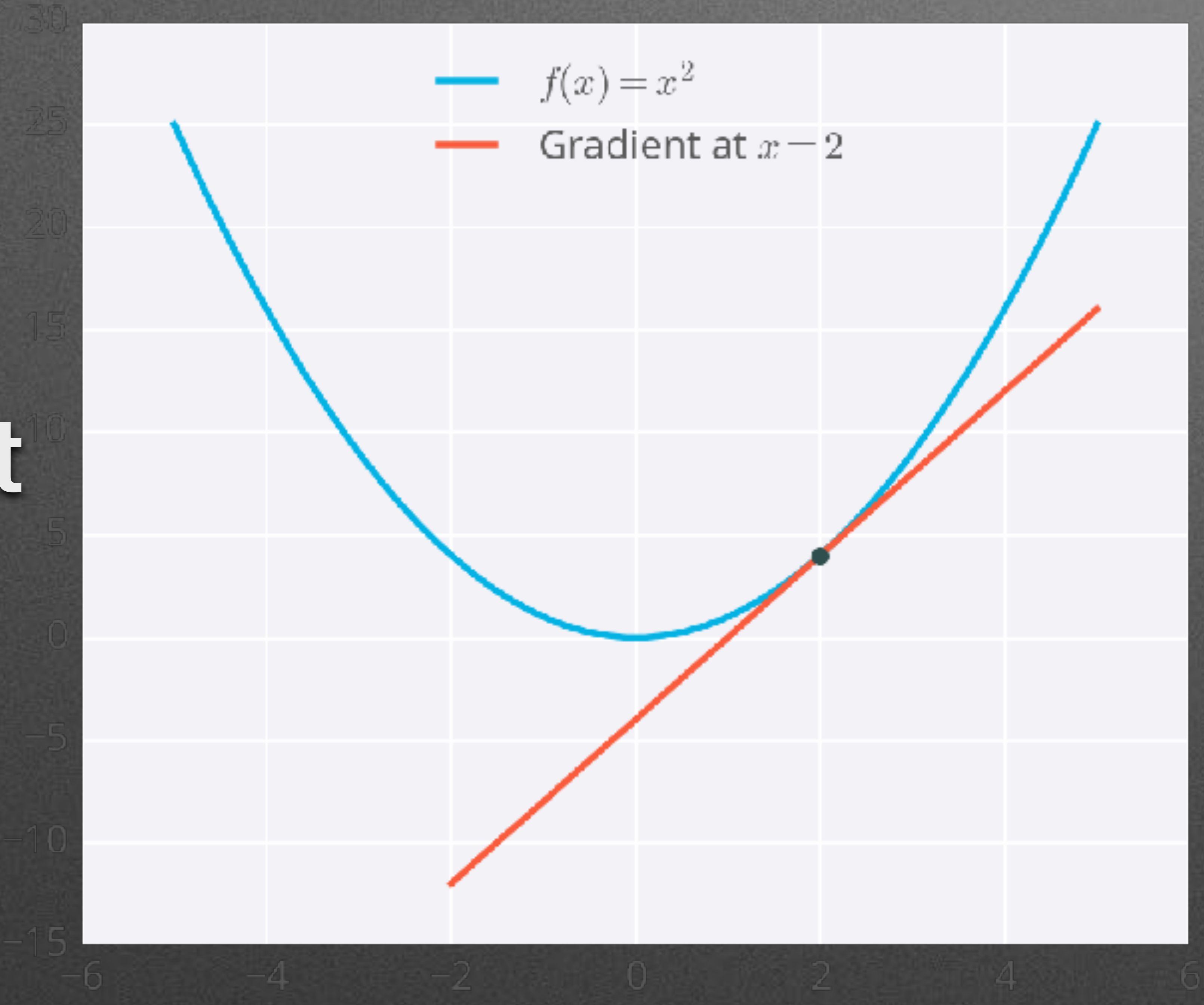
Back propagation

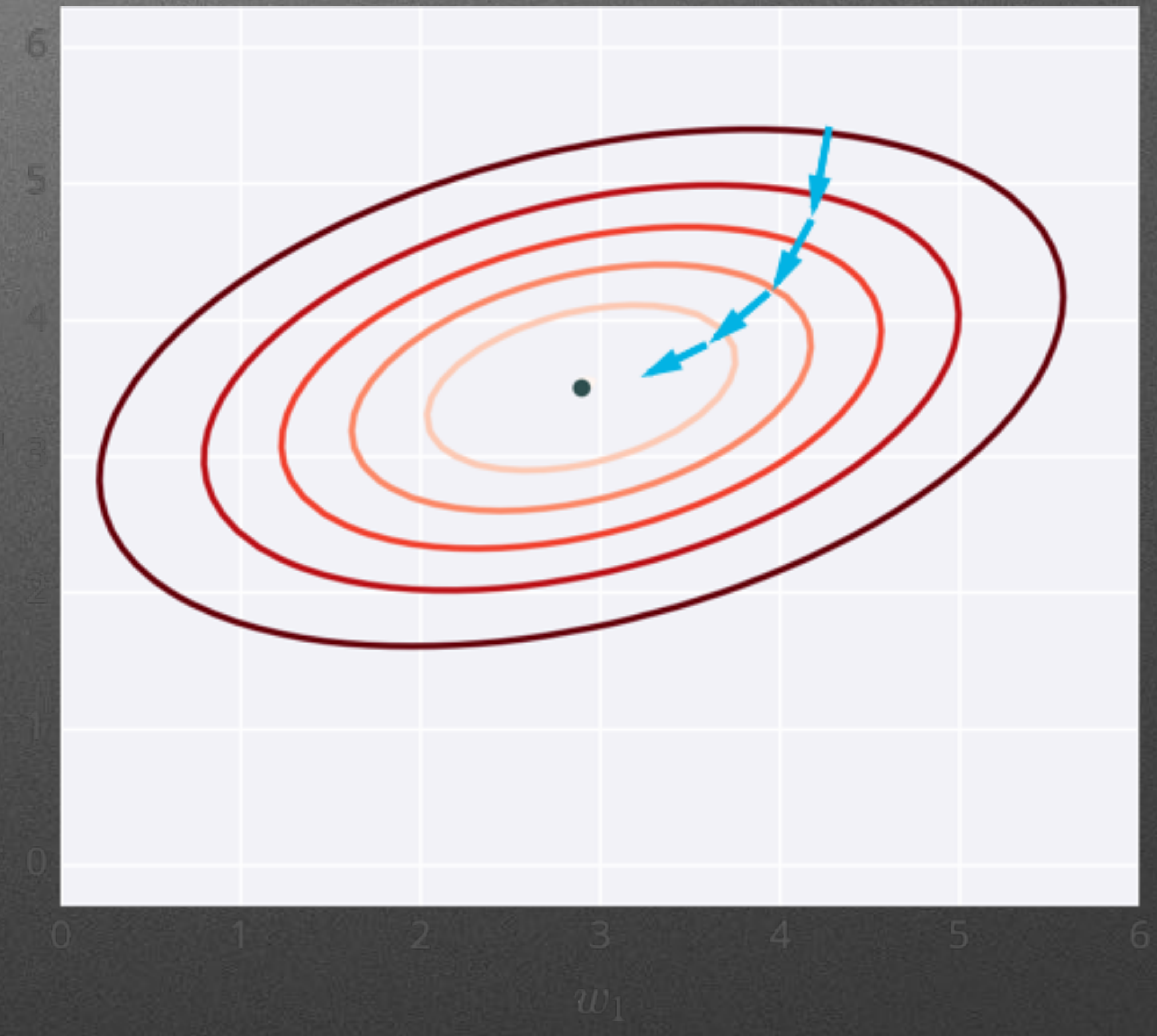


$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix} \quad \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{pmatrix}$$

Gradient descent

Cost



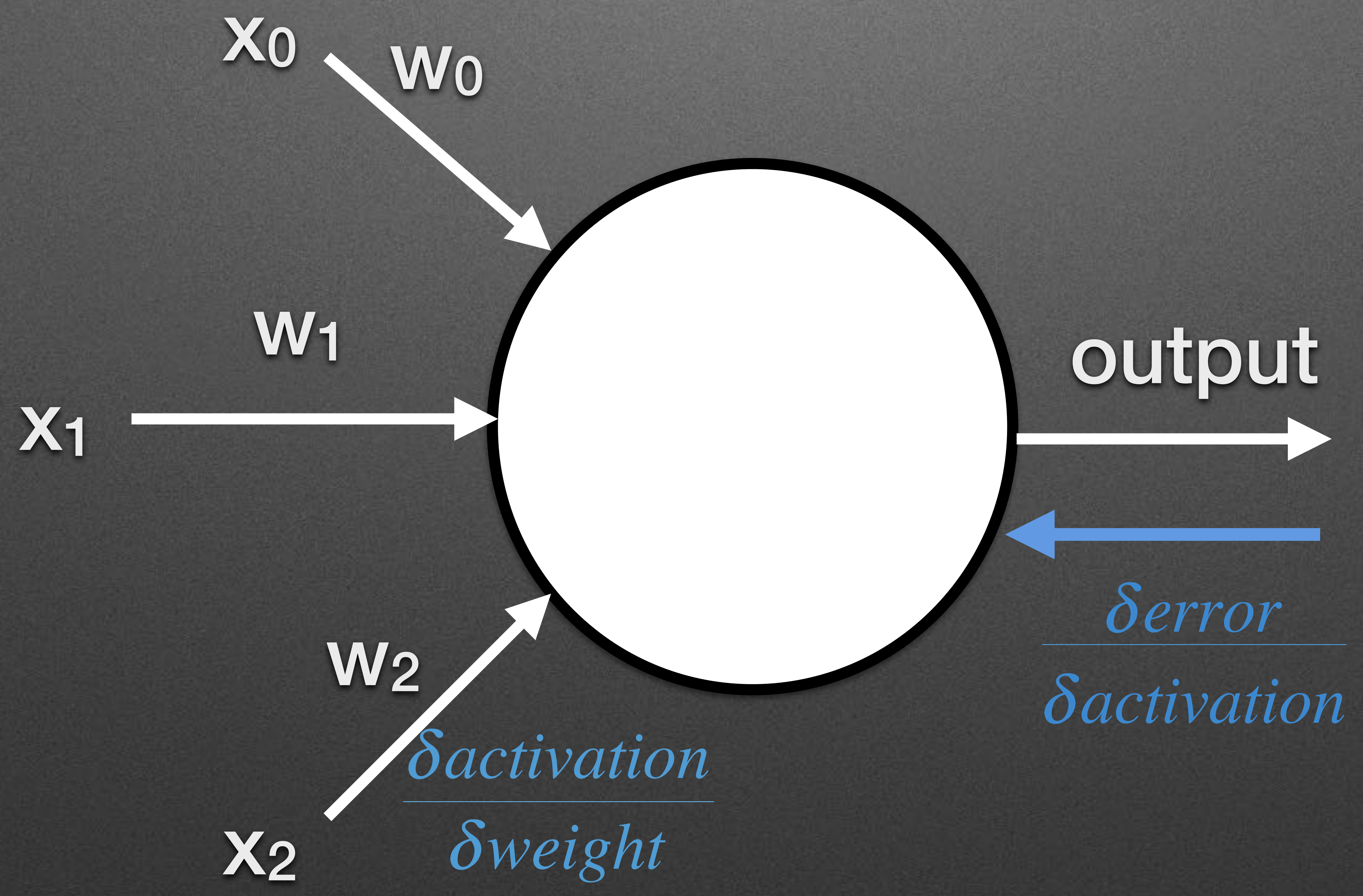


Model parameter (weight)

delta = -gradient * error * learning rate

The Chain Rule

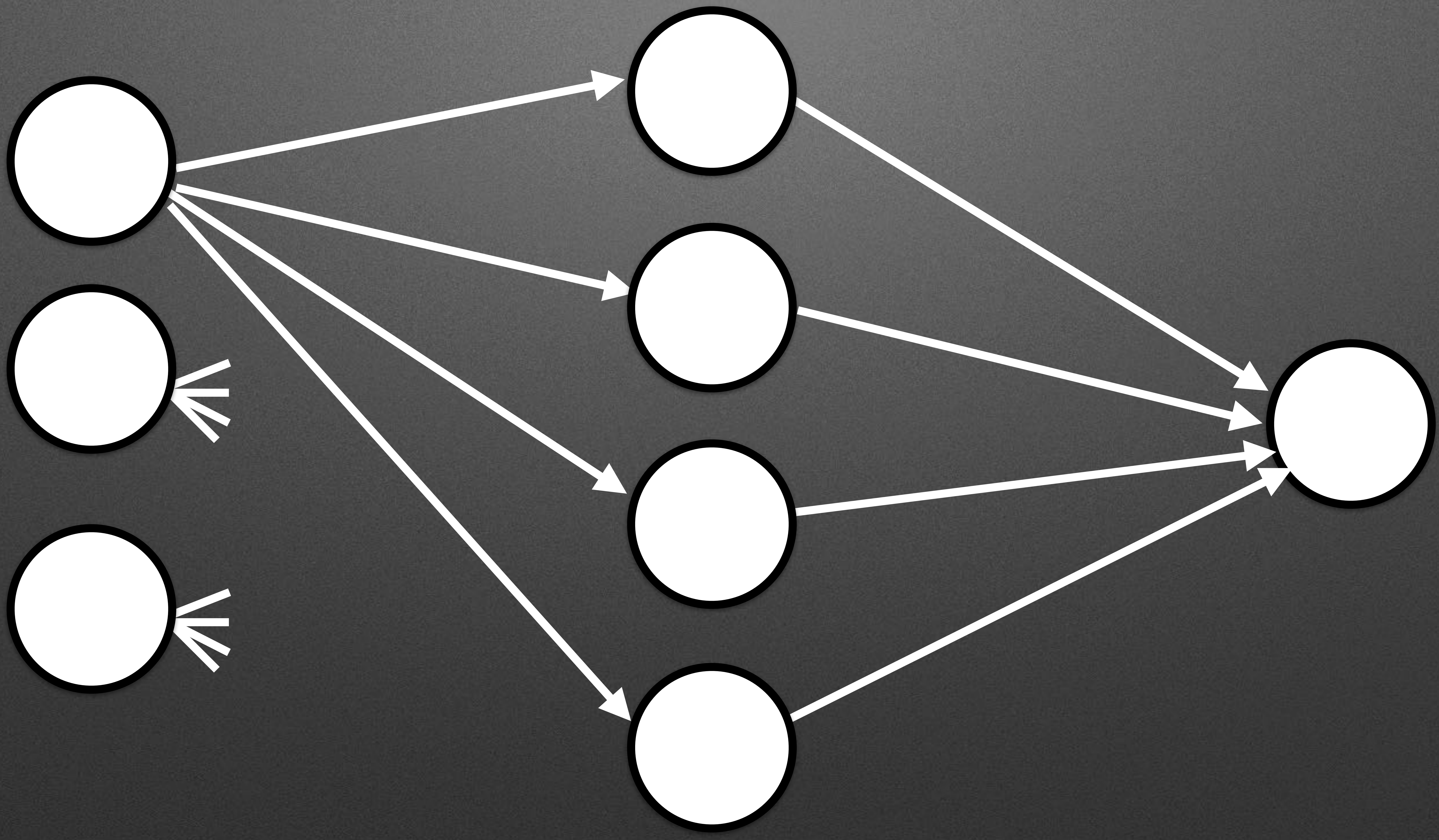
Want error gradient with respect to each weight



$$\frac{\delta error}{\delta weight} = \frac{\delta error}{\delta activation} \times \frac{\delta activation}{\delta weight}$$

Time for some java

Backpropgation



Sigmoid & derivative

```
private INDArray sigmoid(INDArray input) {
return Nd4j.ones(input.shape()).div(exp(input.neg()).add(1));
}

private INDArray sigmoidDerivative(INDArray input) {
```

Inputs & Weights

```
final double[][] inputsArray = {
{0, 0, 1},
{0, 1, 1},
{1, 0, 1},
{1, 1, 1}
```

Forward pass

```
for (int i = 0; i < numIterations; ++i) {
// forward pass
INDArray layer1 = sigmoid(x.mmul(weights1));
INDArray layer2 = sigmoid(layer1.mmul(weights2));
```

Backward pass

```
// backward pass
INDArray delta2 = layer2Error.mul(sigmoidDerivative(layer2));

INDArray layer1Error = delta2.mmul(weights2.transpose());
```

Backward pass

```
weights2 = weights2.add(
    layer1.transpose()      // chain rule
    .mmul(delta2)           // error value
    .mul(learningRate));    // update scale factor
```