

Everything You Always Wanted to Know About

Neural Networks

But Were Afraid to Ask

Ilya Ezepov

Agenda

- Linear models
- Perceptron
- Deep NN
- Convolutional NN
- Modern NN techniques
- When to use NN and when to not

18. ... Bxe7



White to move

19.c4





Garry Kasparov



Garry Kasparov



Deep Blue

Checkers: completely solved (2007)

Checkers: completely solved (2007)

Chess: February 1996
(first win by computer against top human)

Checkers: completely solved (2007)

Chess: February 1996
(first win by computer against top human)

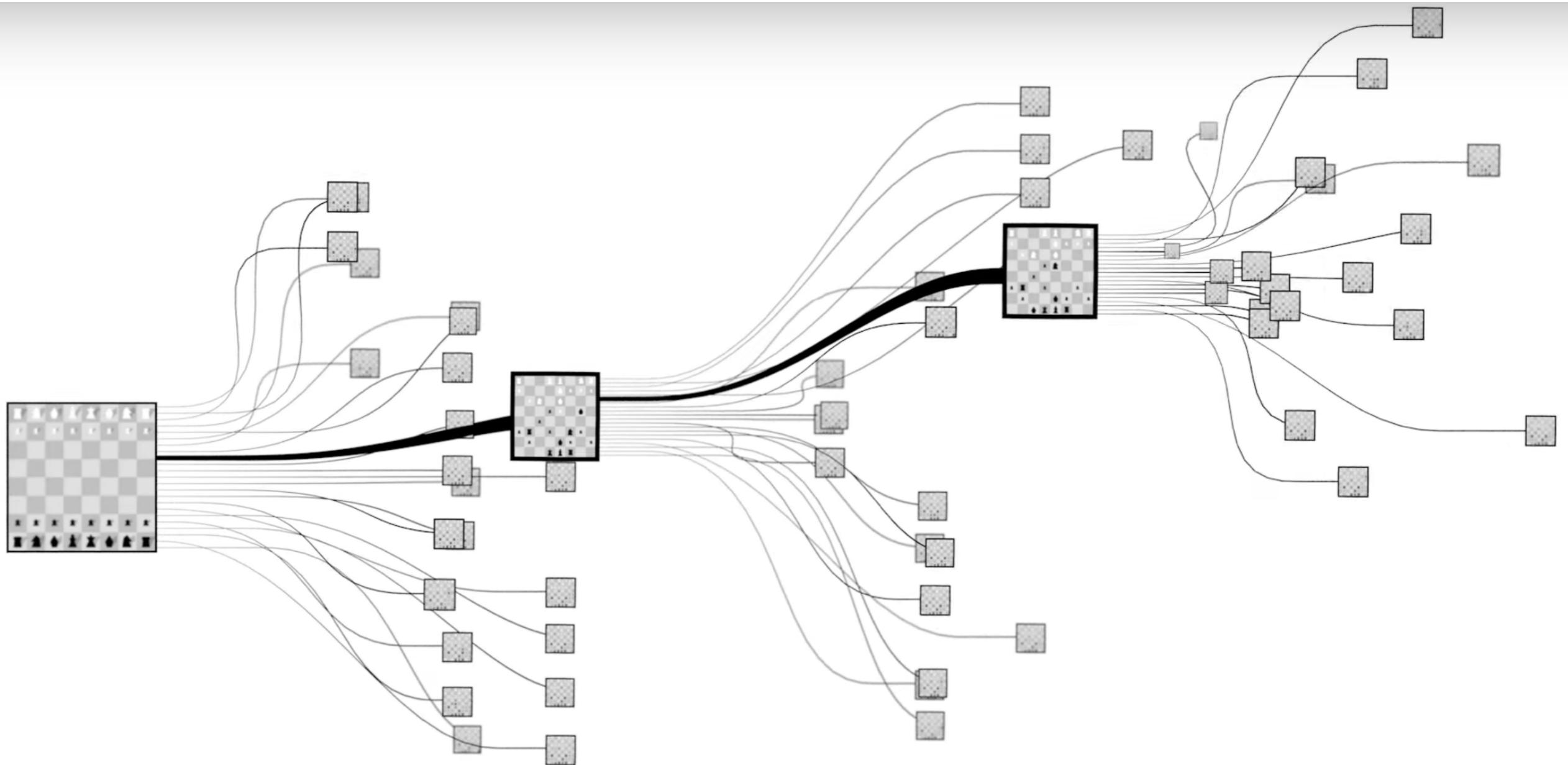
Chess: November 2005
(last win by human against top computer)

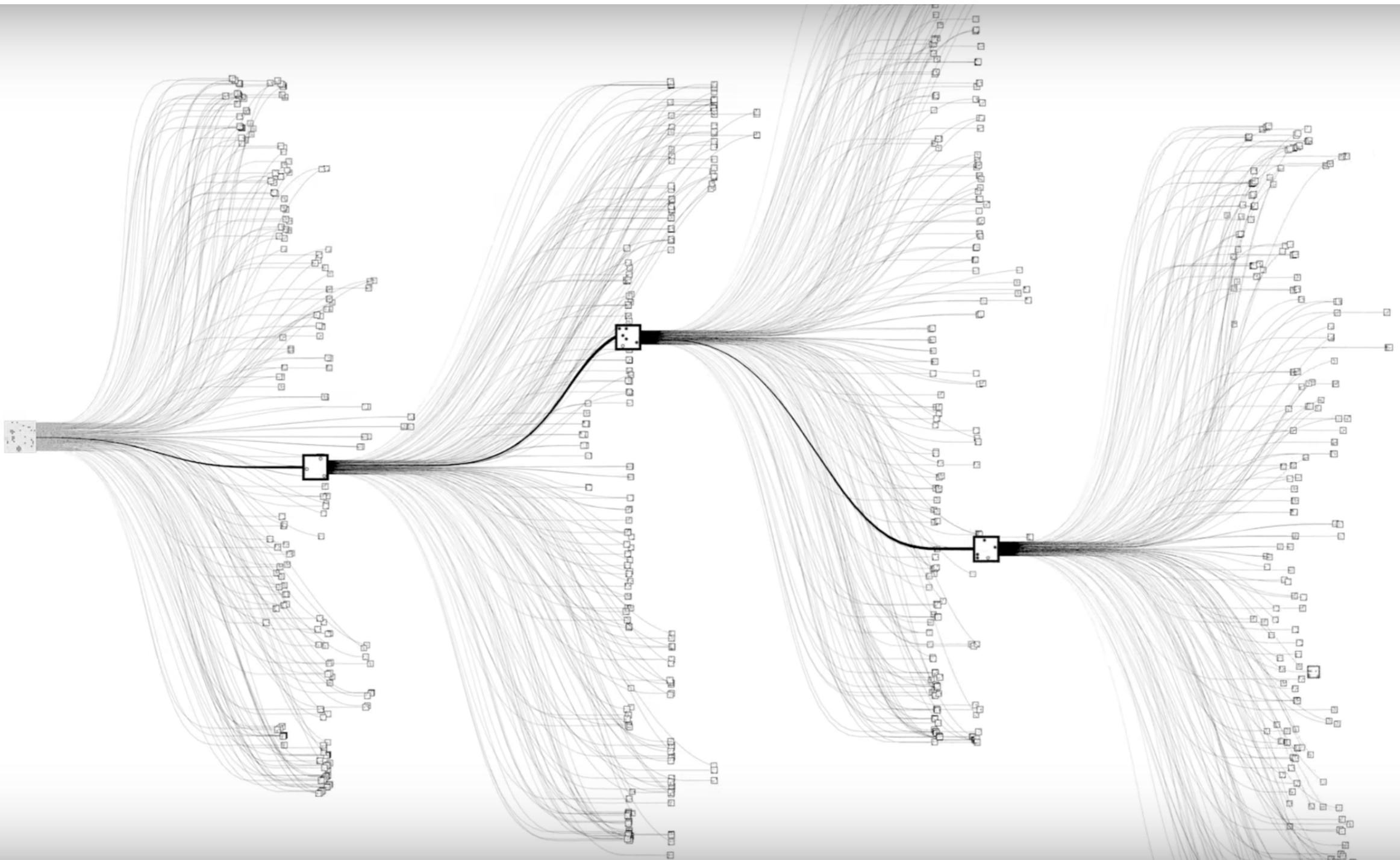
Checkers: completely solved (2007)

Chess: February 1996
(first win by computer against top human)

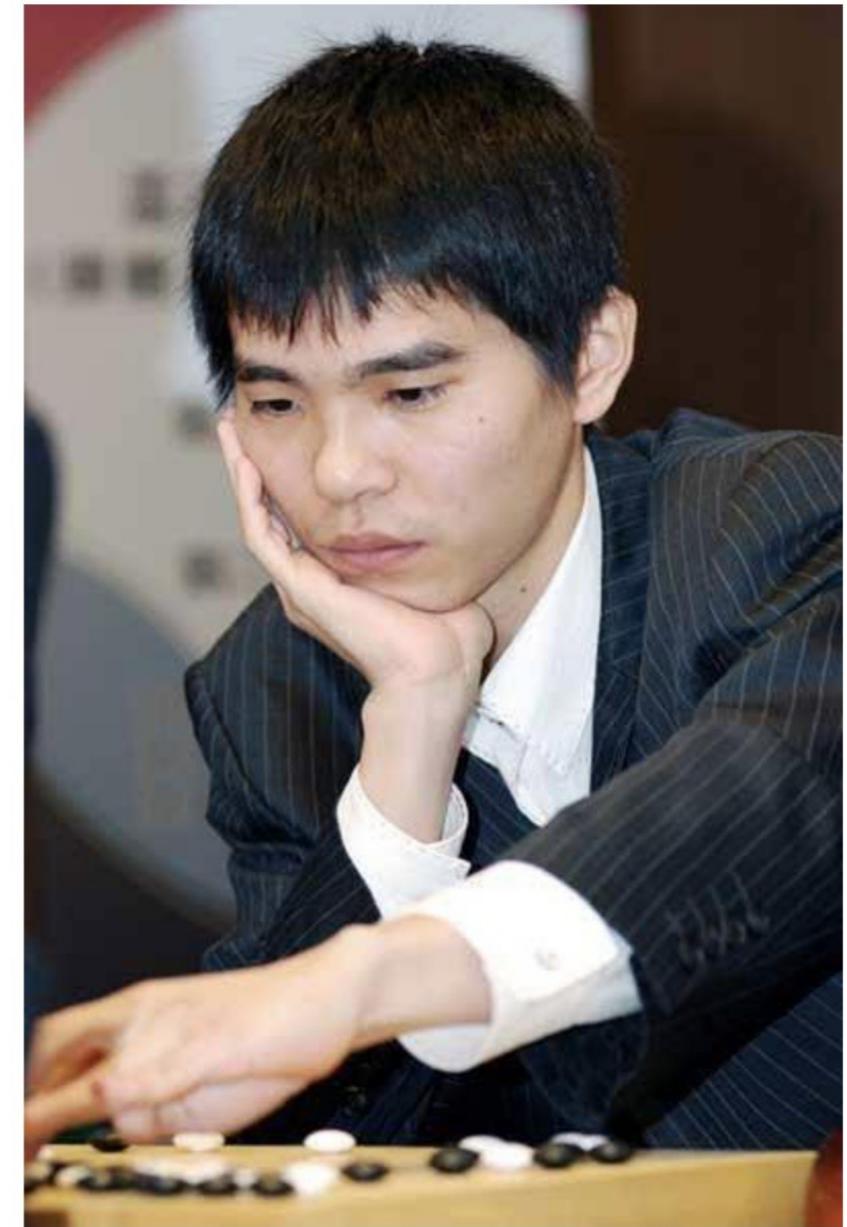
Chess: November 2005
(last win by human against top computer)

Go: Uncracked for a long time



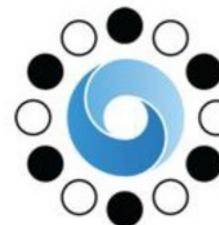


AlphaGo (by Google)

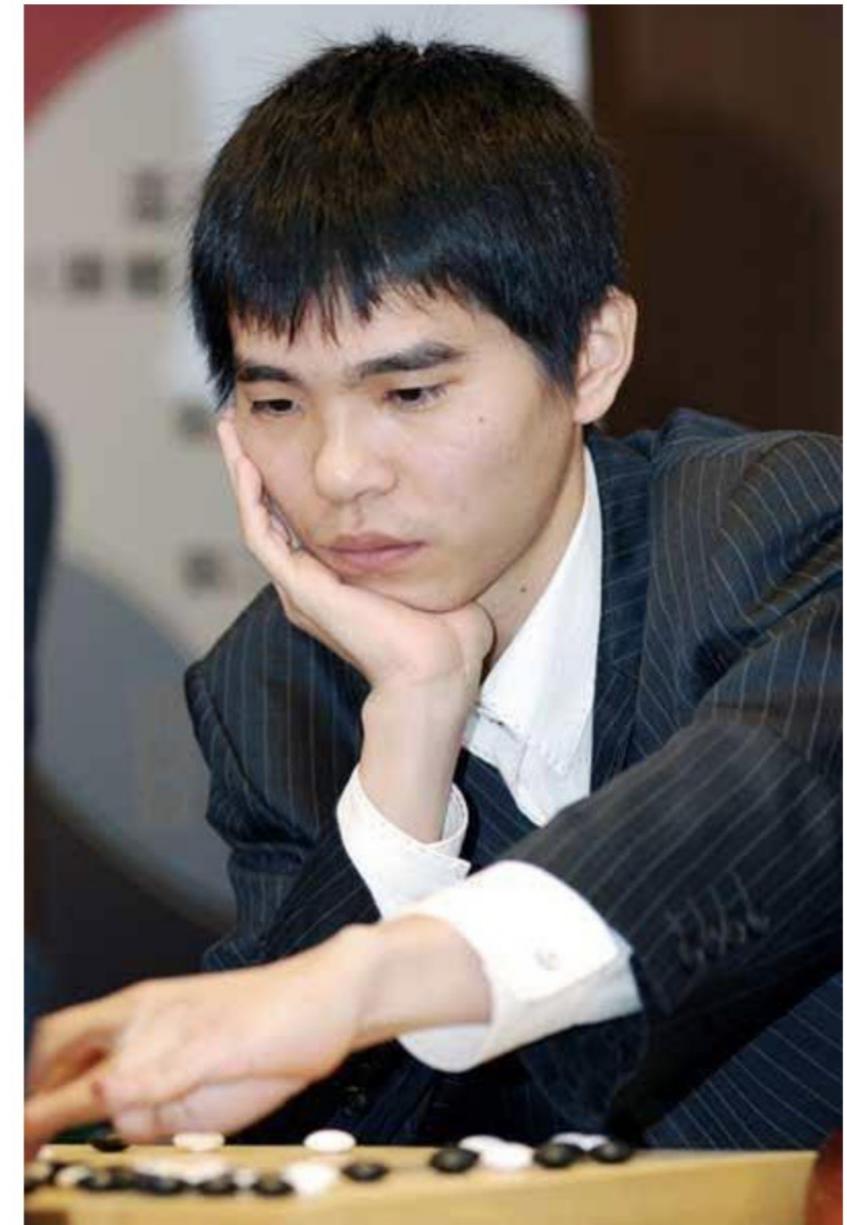


Lee Sedol,
Last decade top player

AlphaGo (by Google)

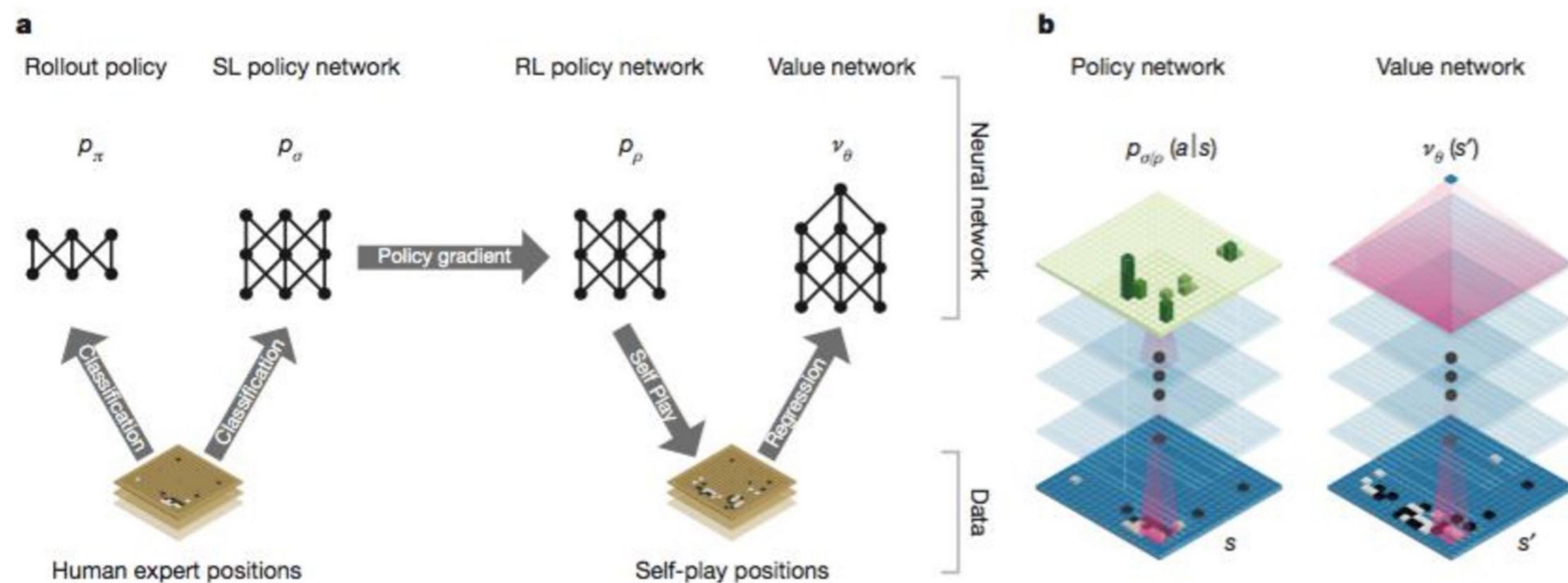
 AlphaGo

4:1



Lee Sedol,
Last decade top player

It works on neural networks!



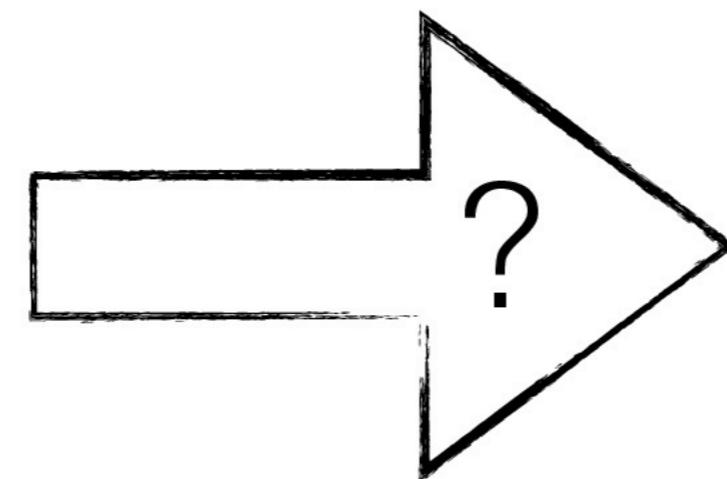
Mastering the game of Go with deep neural networks and tree search
Nature 529, 484–489 (28 January 2016) doi:10.1038/nature16961

Linear Models

The Problem

Feature matrix

Correct answer



The Problem

Model:

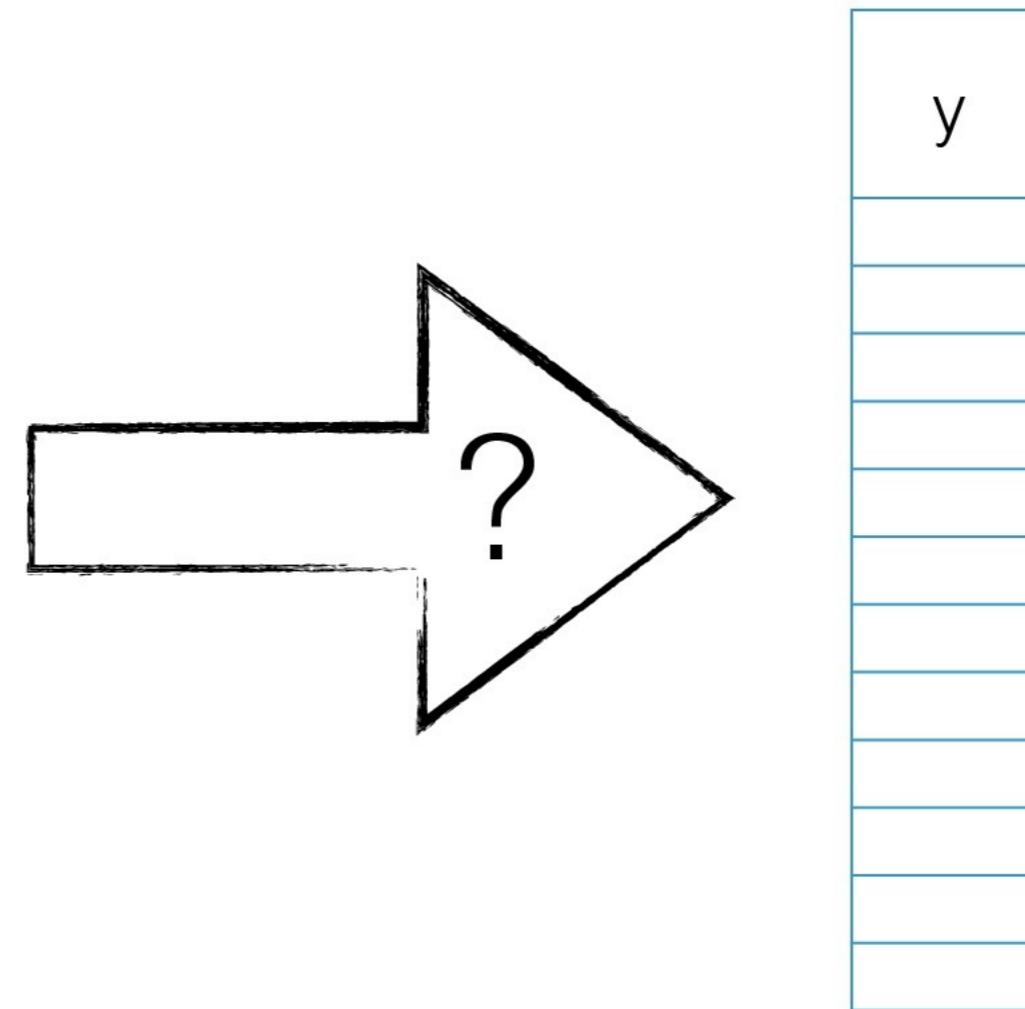
$$X_1W_1 + X_2W_2 + X_3W_3 + X_4W_4 + b = \hat{y}$$

How to find W, b?

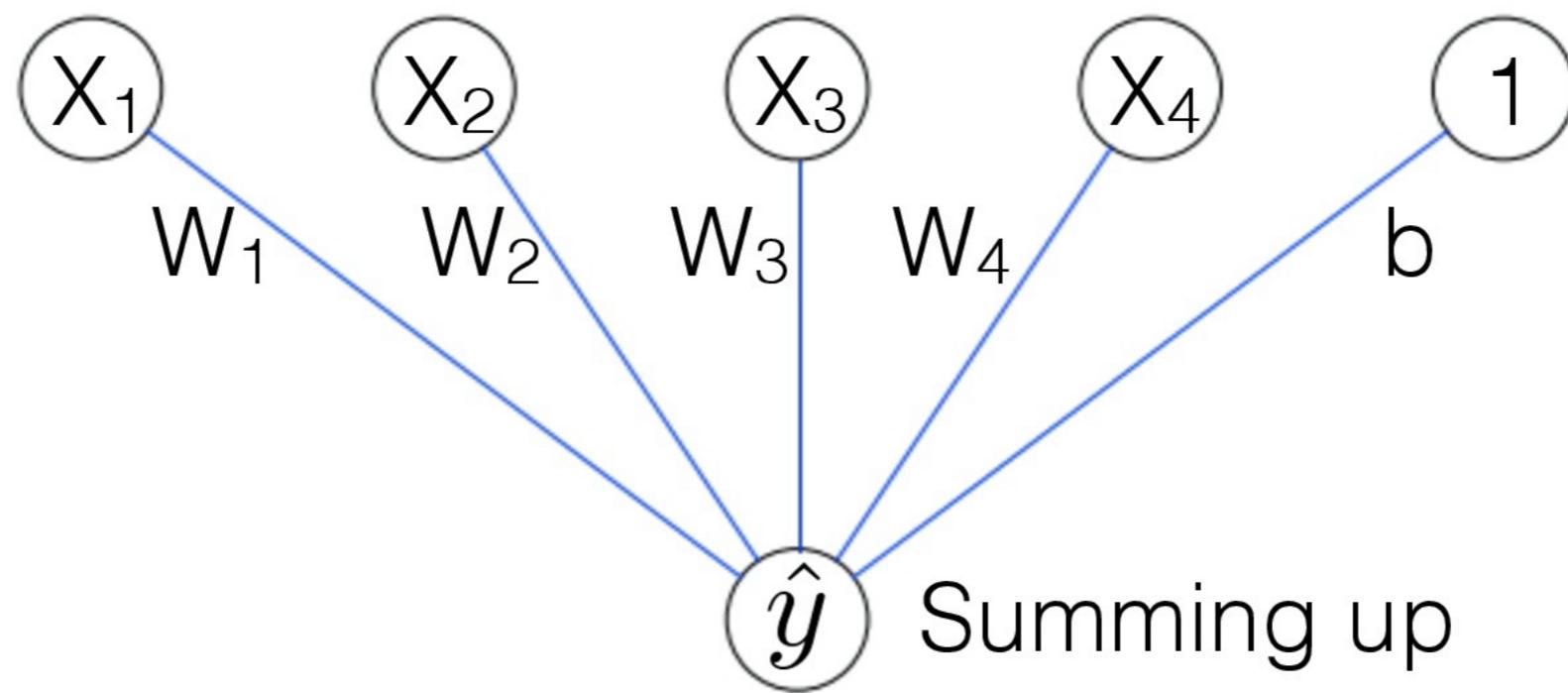
Adding bias

Feature matrix (X)

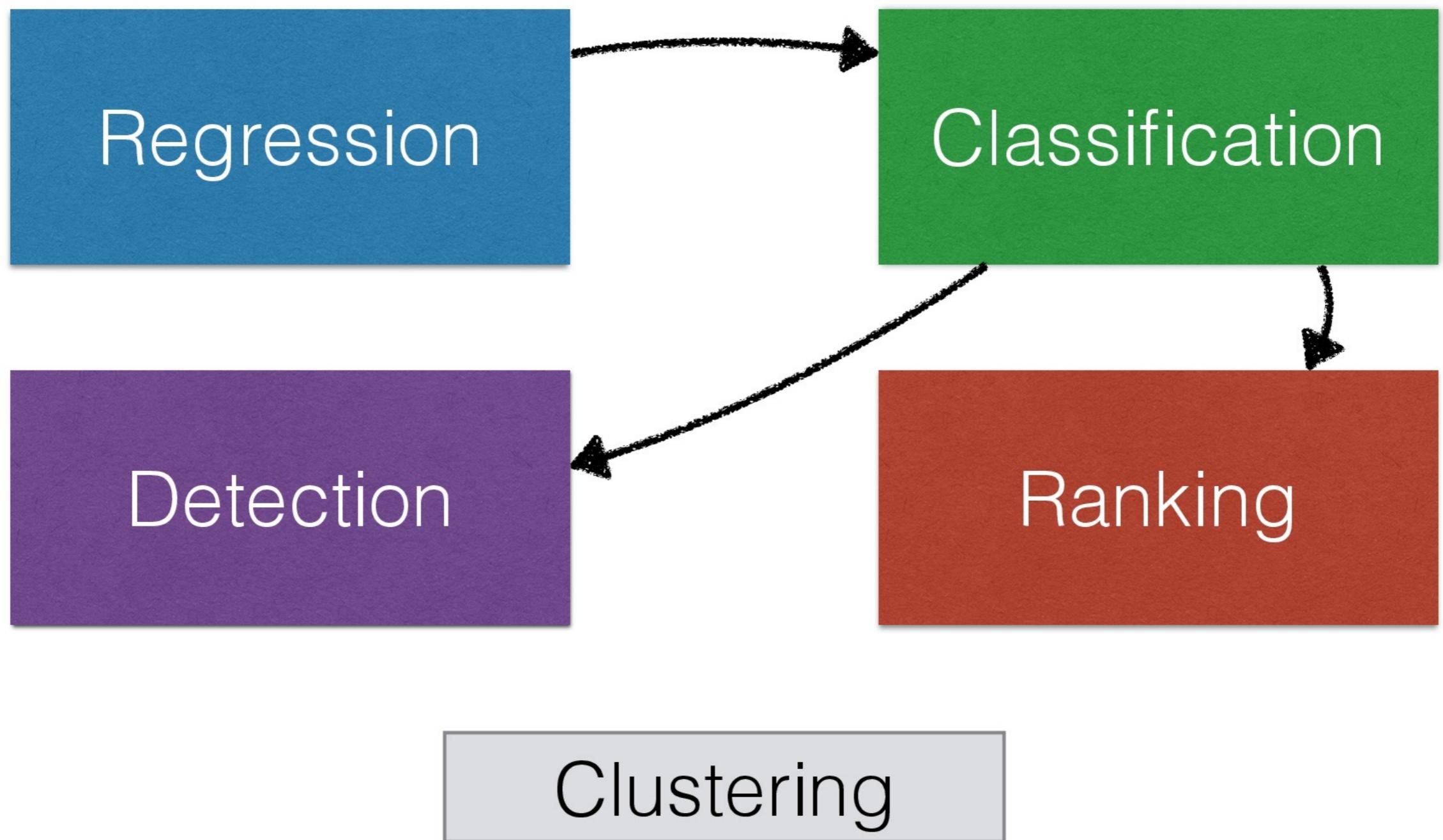
Correct answer (y)



Simple linear model



Machine learning types of problems



How to train?

- “Training” = Finding optimal W , b

How to train?

- “Training” = Finding optimal W , b
- “Optimal” = Minimize loss function (a.k.a. cost function)

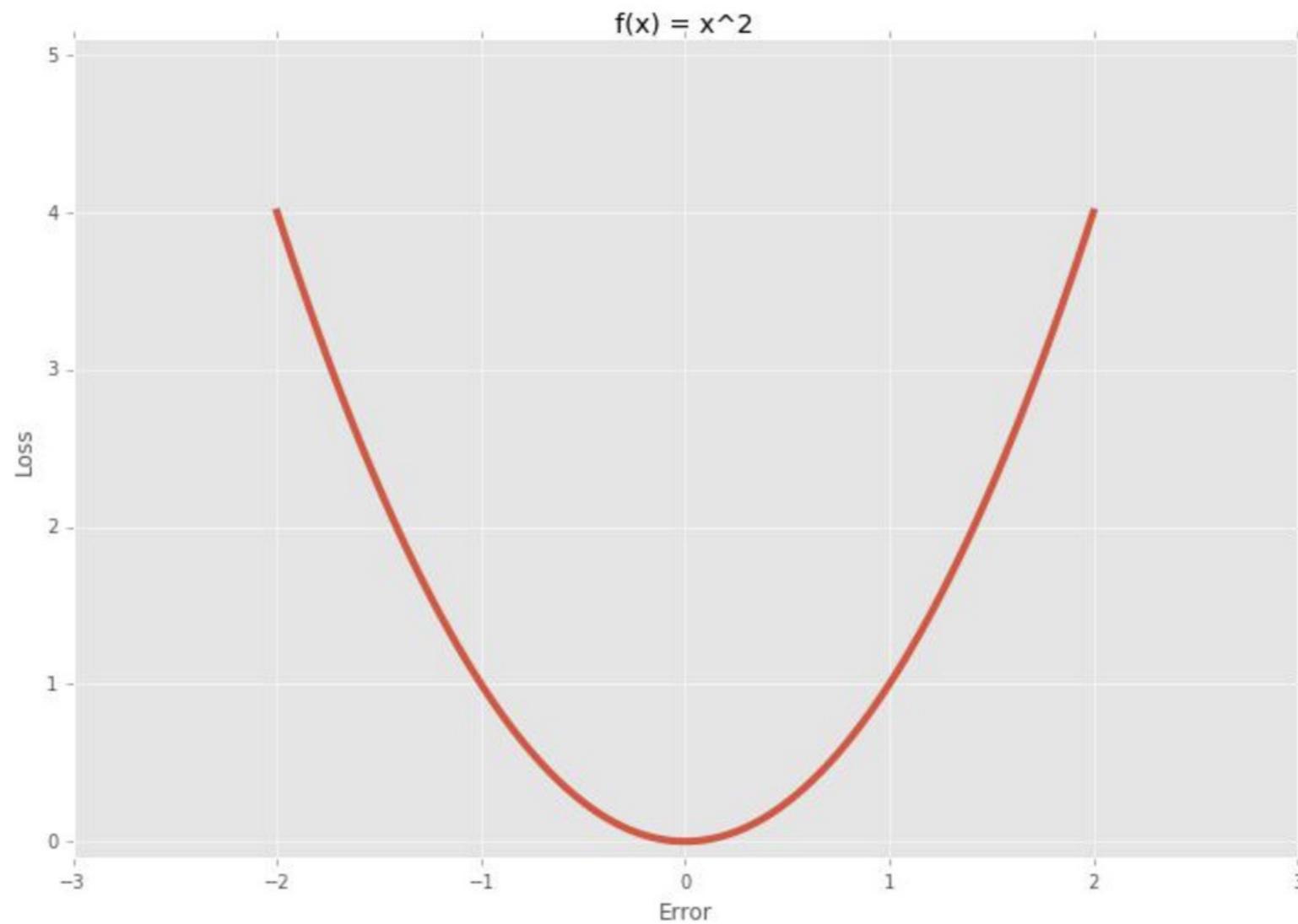
How to train?

- “Training” = Finding optimal W , b
- “Optimal” = Minimize loss function (a.k.a. cost function)
- The must-know losses:
 - MSE (mean squared error)
 - Log loss
 - Hinge loss
 - Huber loss

Loss functions

Mean Squared Error (MSE) for regression

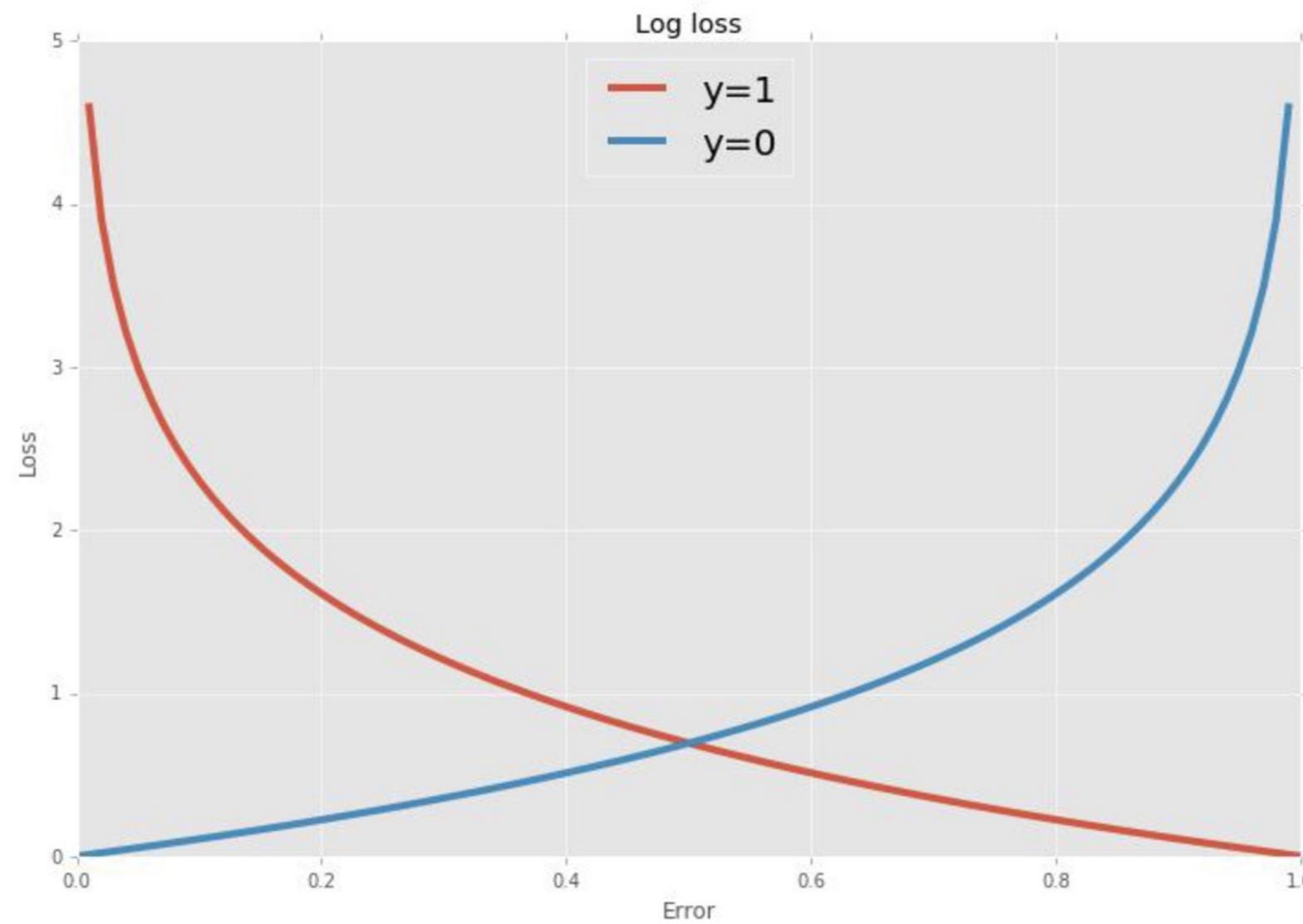
$$\frac{1}{N} \sum_{i=1}^N \frac{1}{2} (y_i - \hat{y}_i)^2$$



Loss functions

Logarithmic loss for classification

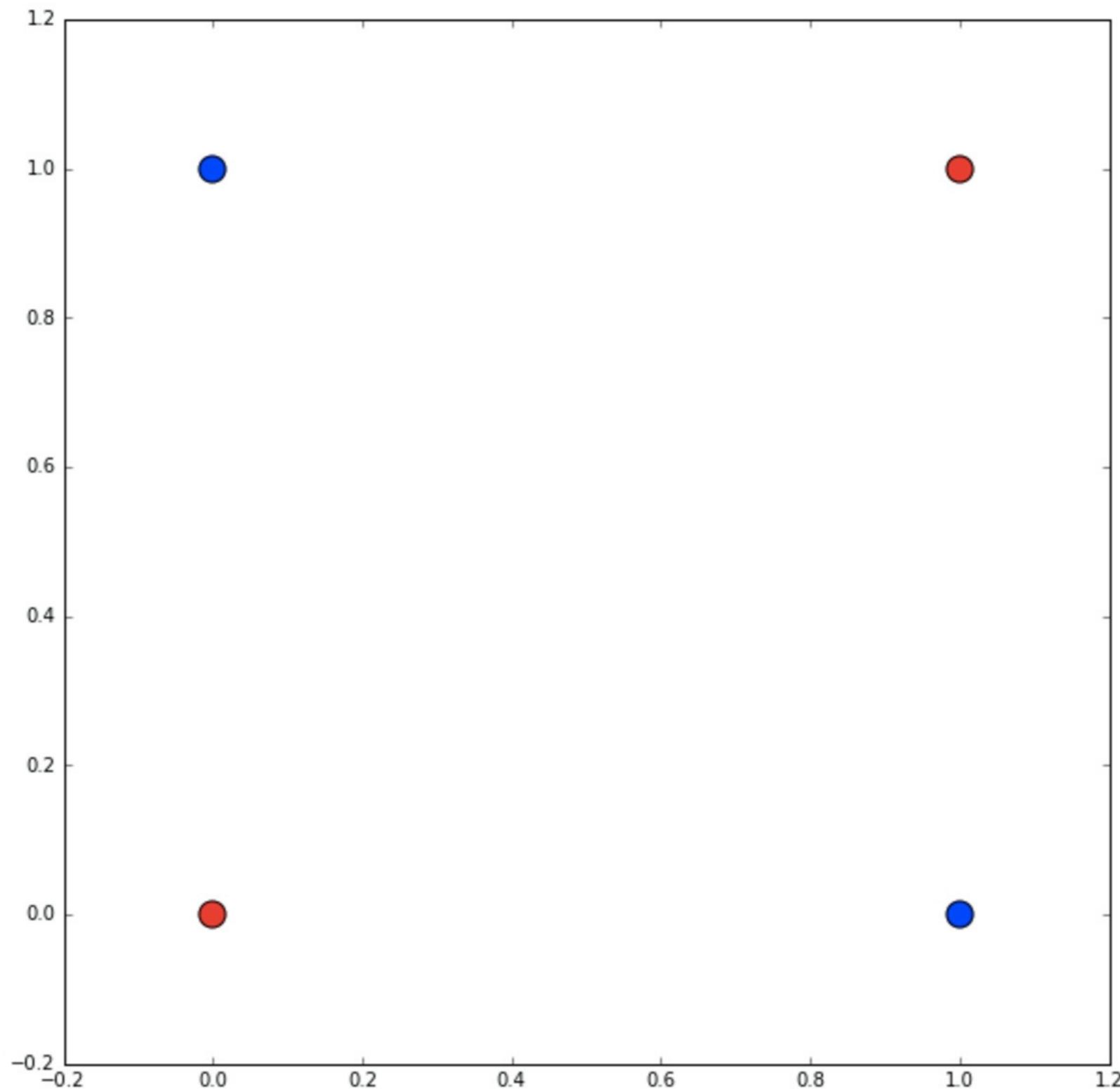
$$-\frac{1}{N} \sum_{i=1}^N (y_i \ln p_i + (1 - y_i) \ln(1 - p_i))$$



Optimization

- Explicit solution
- Gradient descent
- SGD
- Adam/RMSprop

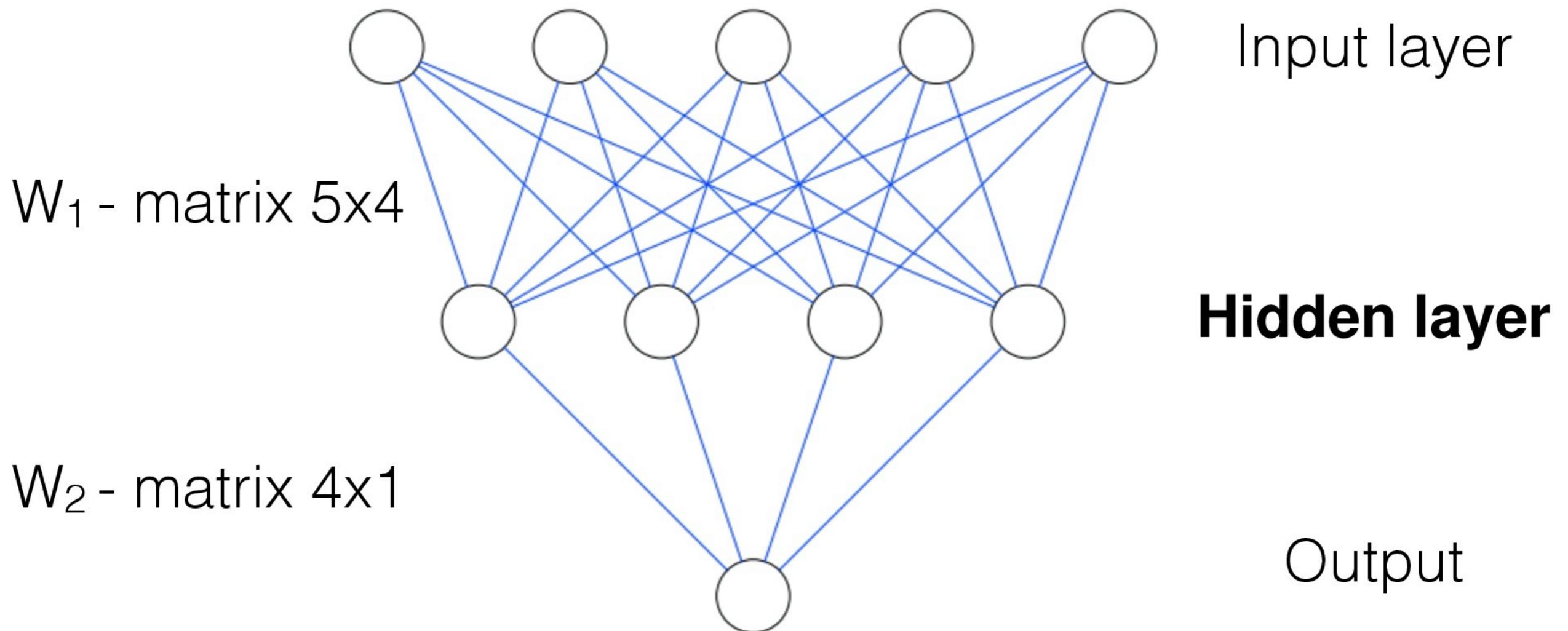
XOR problem



Perceptron

Perceptron model

Published in 1957, implemented in 1960



Output calculation

Linear model: $\hat{y} = XW$

Output calculation

Linear model: $\hat{y} = XW$

Hidden: $h = XW_1$

Perceptron:

Output calculation

Linear model: $\hat{y} = XW$

Hidden: $h = XW_1$

Perceptron:

Output: $\hat{y} = hW_2$

Output calculation

Linear model: $\hat{y} = XW$

Hidden: $h = XW_1$

Perceptron:

Output: $\hat{y} = hW_2$

But: $\hat{y} = hW_2 = XW_1W_2 = XA$

**Linear
again!
damn.**

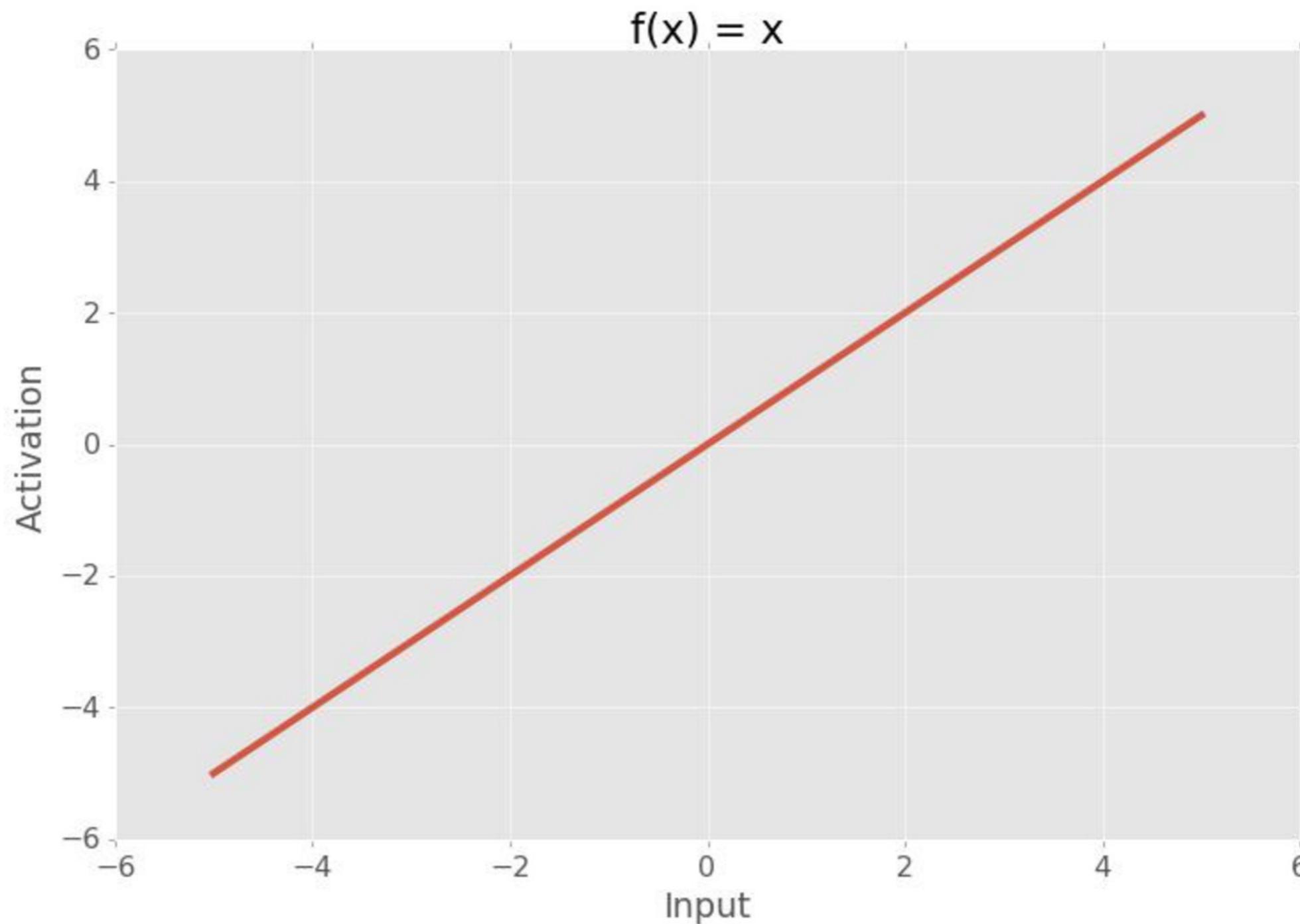
Idea

Add some non-linear activation in the hidden layer

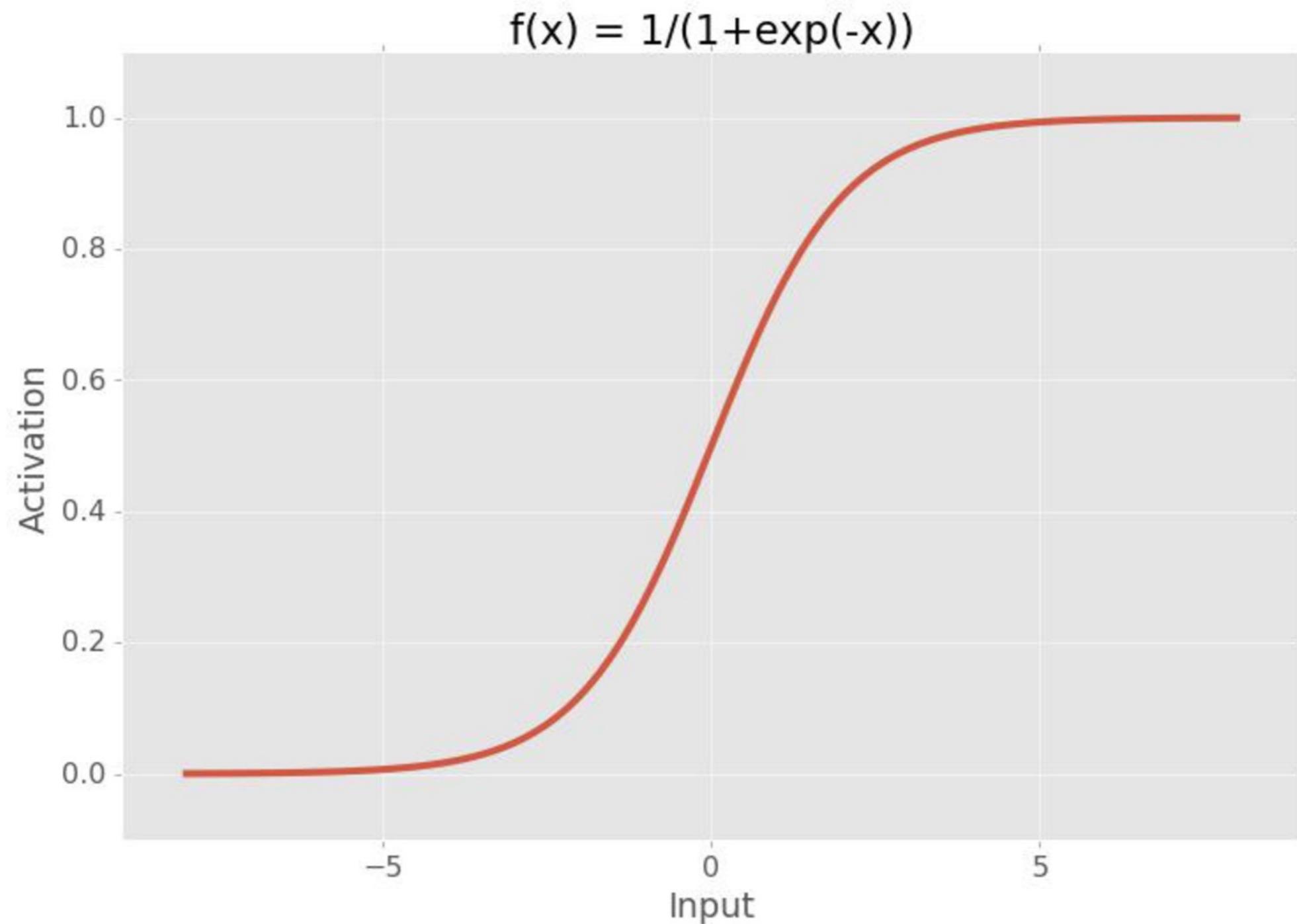
$$h = f(XW_1)$$

$$\hat{y} = hW_2 = f(XW_1)W_2 \neq XA$$

Activation function

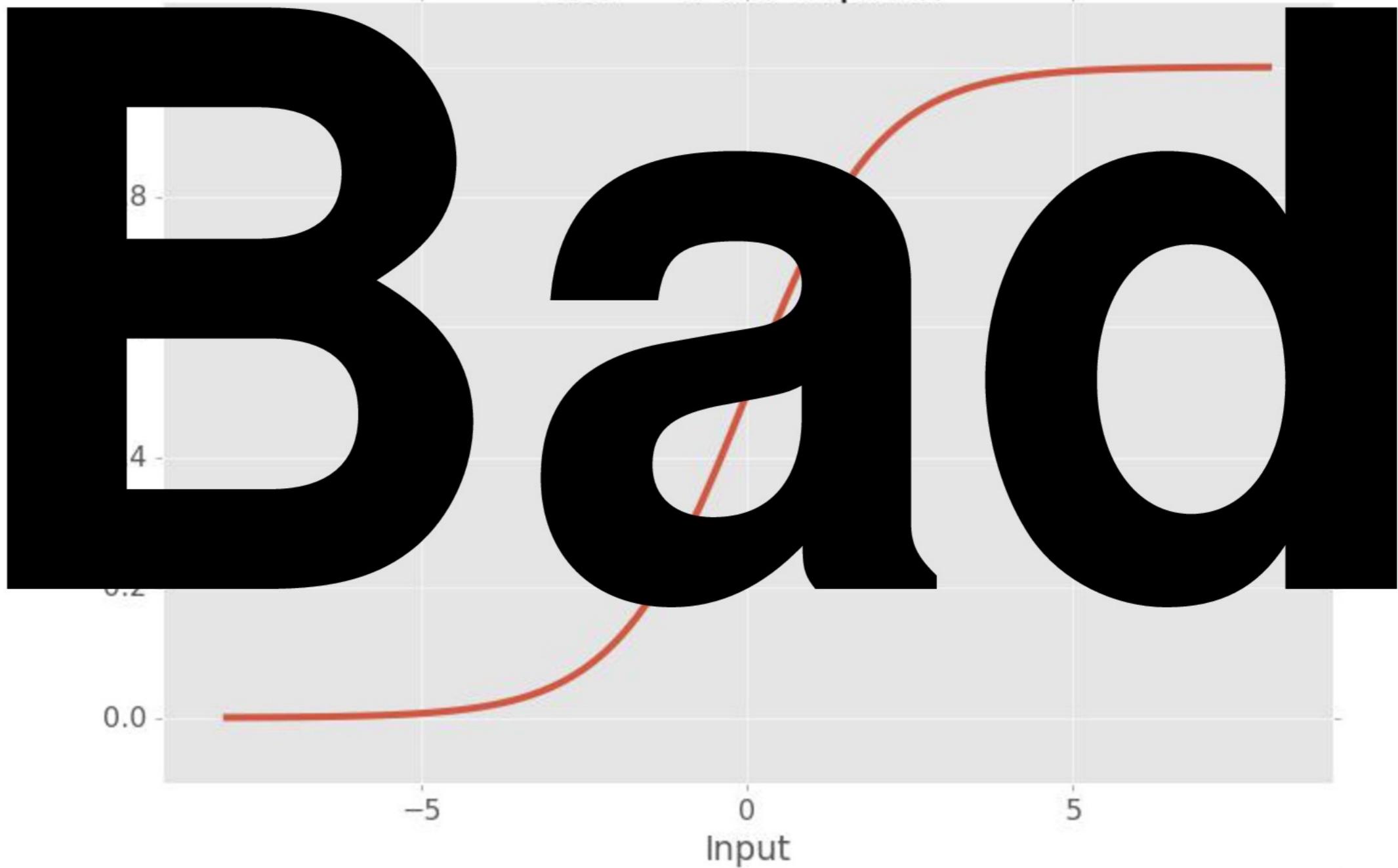


Sigmoid

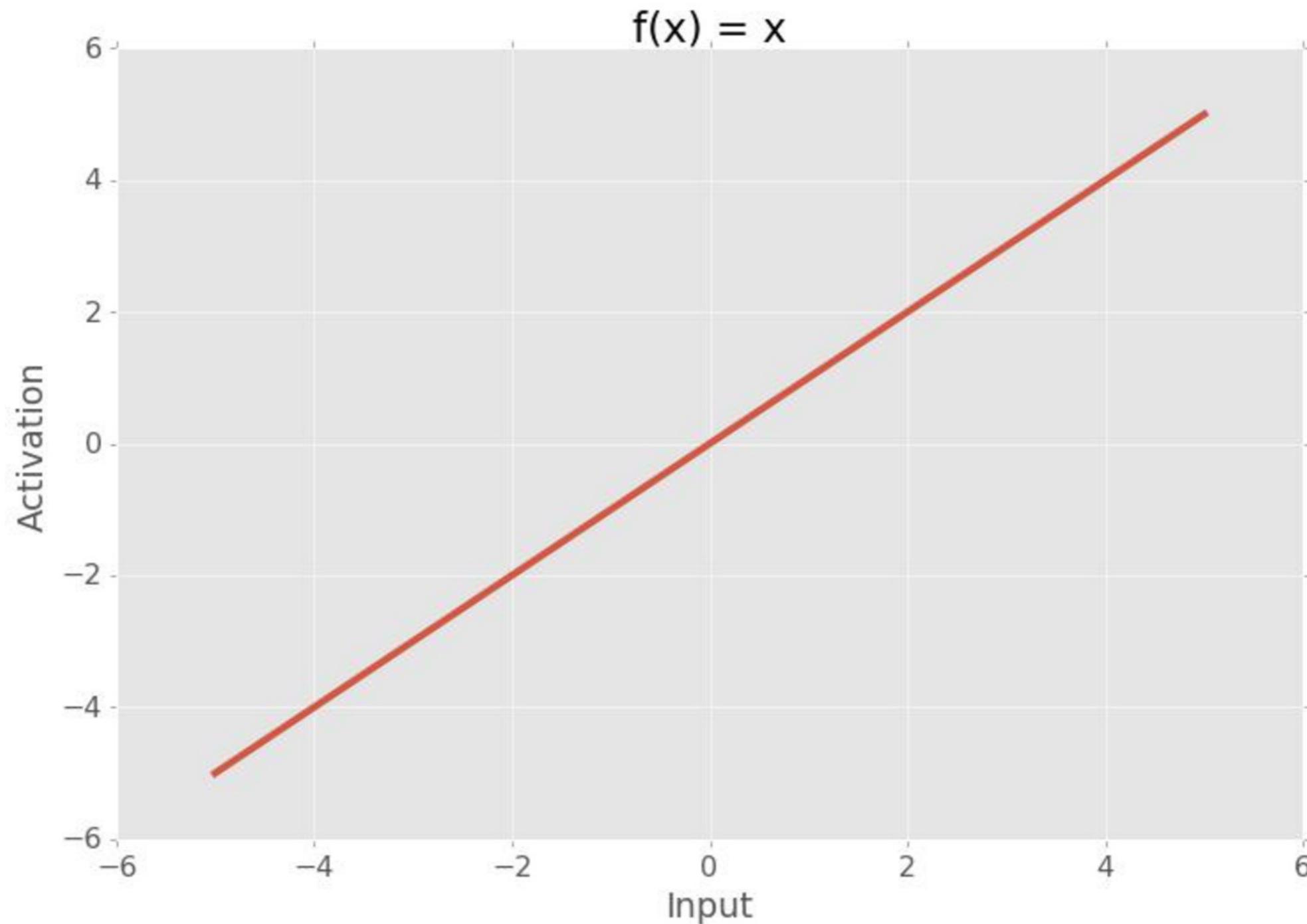


Sigmoid

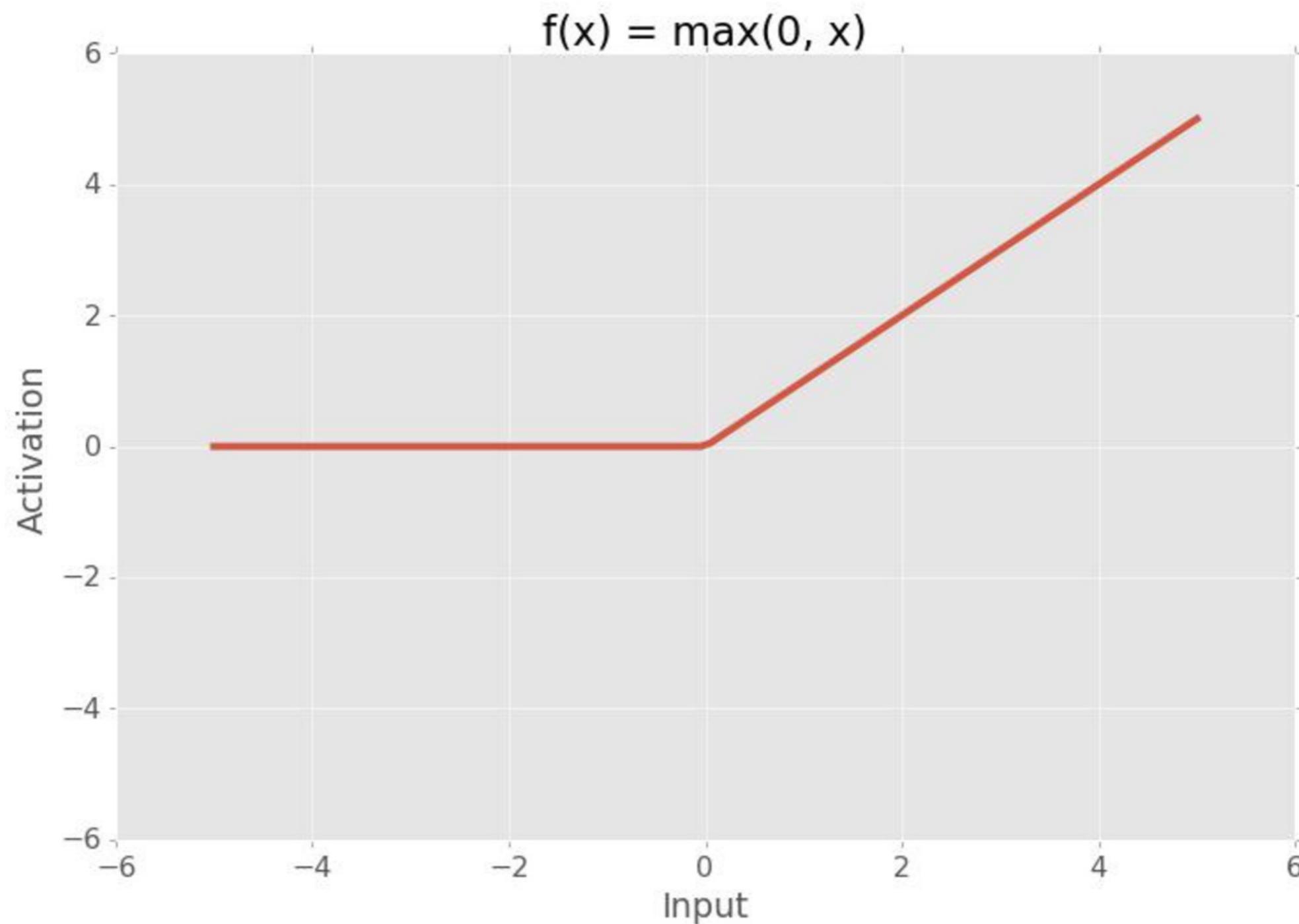
$$f(x) = 1/(1+\exp(-x))$$



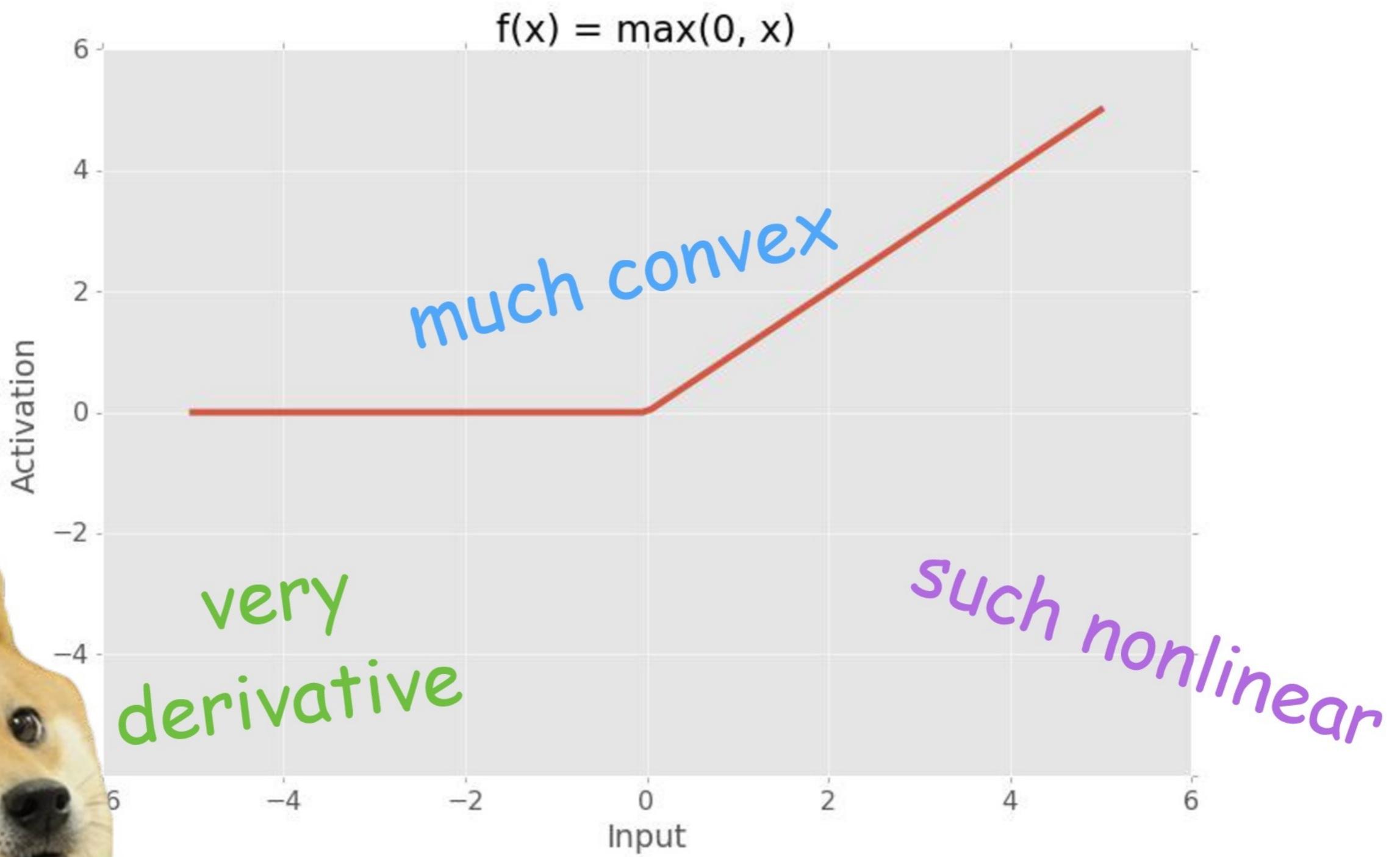
From linear activation . . .



... to nonlinear



ReLU



How to train perceptron?

$$Loss(W_1, W_2) = \frac{1}{2} (y - f(XW_1)W_2)^2$$

The key point is chain rule:

$$f(g(x))' = f'(g(x)) * g(x)'$$

How to train perceptron?

$$Loss(W_1, W_2) = \frac{1}{2} (y - f(XW_1)W_2)^2$$

The key point is chain rule:

$$f(g(x))' = f'(g(x)) * g(x)'$$

$$\frac{\partial Loss}{\partial W_2} = -(y - f(XW_1)W_2)f(XW_1)$$

$$\frac{\partial Loss}{\partial W_1} = -(y - f(XW_1)W_2)W_2f'(XW_1)W_1$$

The Perceptron Convergence Theorem

Frank Rosenblatt, 1965

- Perceptron can solve **any** problem
- Perceptron will converge in finite time
- Result is independent of initial weights

Great. Why to implement anything else?

The background of the image is a deep blue ocean surface. Sunlight filters down from the top right, creating bright, white, wavy patterns that resemble ripples or reflections on the water's surface. The overall effect is a sense of depth and tranquility.

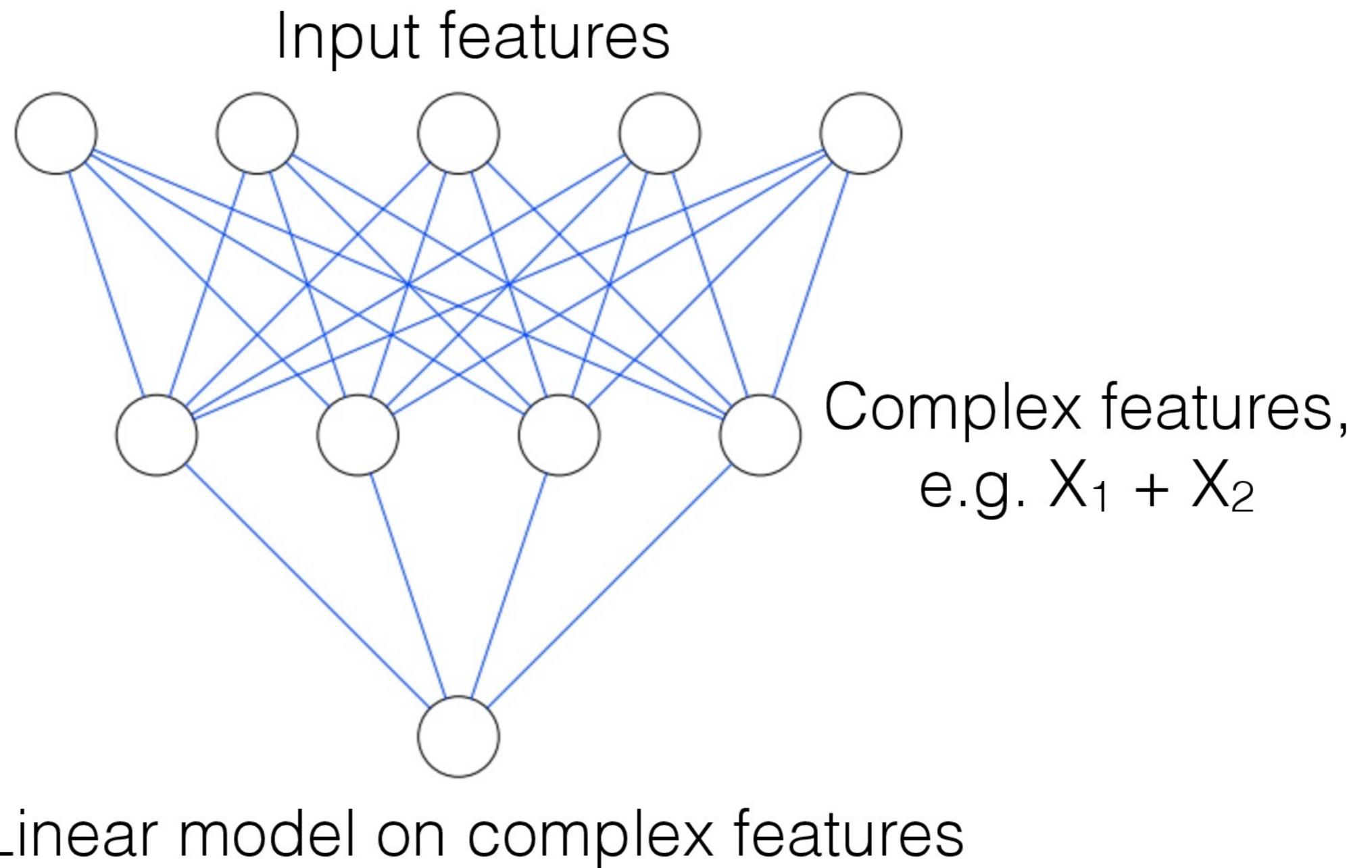
Going Deeper

Motivation: computational cost

Wide enough perceptron can solve any problem, but
the cost will be very high

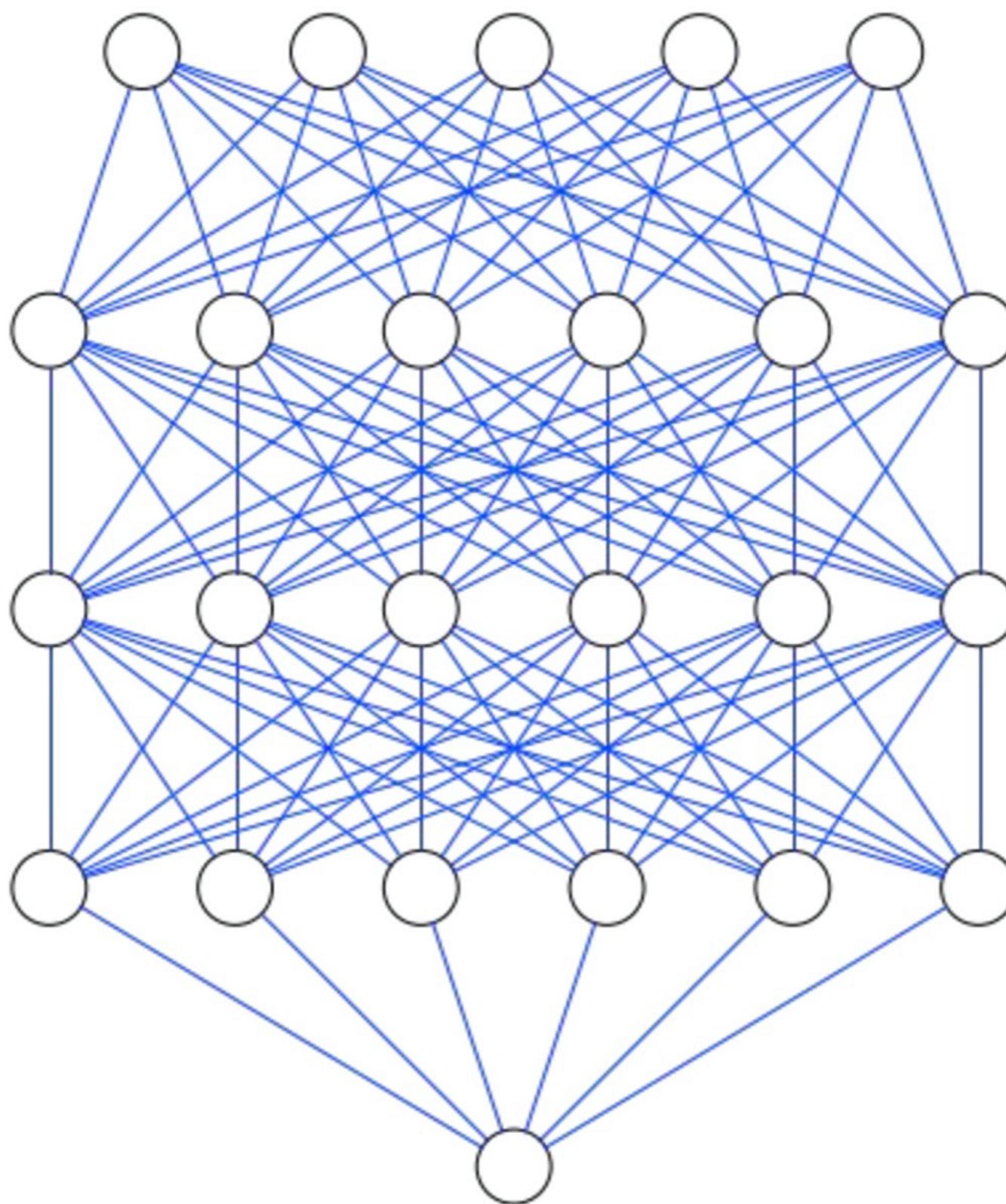
It's cheaper to go deeper than wider

Motivation

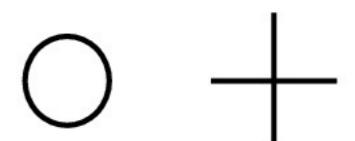


E.g.

Input features



Complex features



Very
Complex features



Extremely
Complex features



Face recognition system

How to train?

Forward propagate your data:

$$X \Rightarrow XW_1 \Rightarrow f(XW_1) \Rightarrow f(XW_1)W_2 \Rightarrow \dots \Rightarrow \hat{y}$$

How to train?

Forward propagate your data:

$$X \Rightarrow XW_1 \Rightarrow f(XW_1) \Rightarrow f(XW_1)W_2 \Rightarrow \dots \Rightarrow \hat{y}$$

Calculate loss:

$$Loss = \frac{1}{2}(y - \hat{y})^2$$

How to train?

Forward propagate your data:

$$X \Rightarrow XW_1 \Rightarrow f(XW_1) \Rightarrow f(XW_1)W_2 \Rightarrow \dots \Rightarrow \hat{y}$$

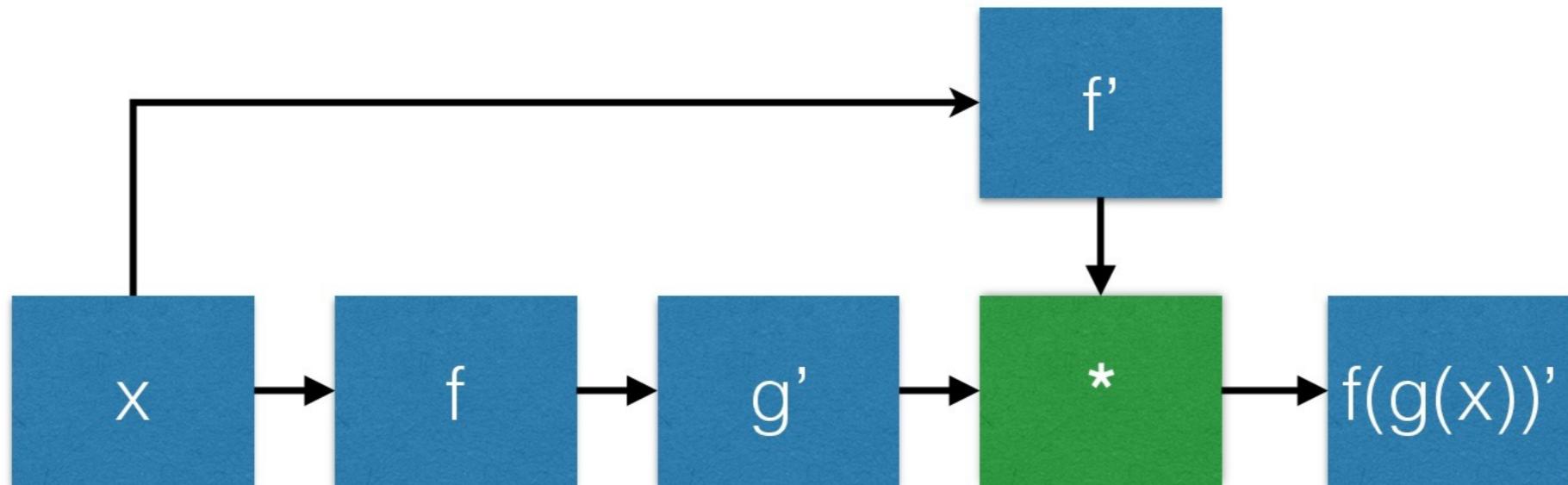
Calculate loss:

$$Loss = \frac{1}{2}(y - \hat{y})^2$$

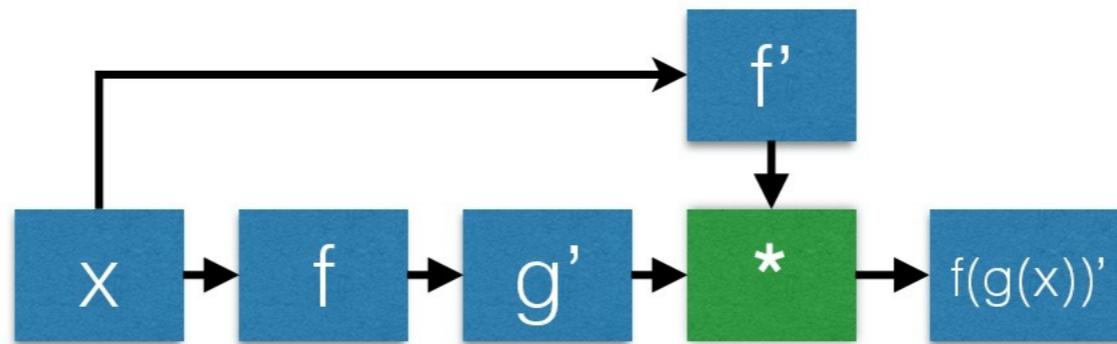
Calculate derivatives:

As always, chain rule

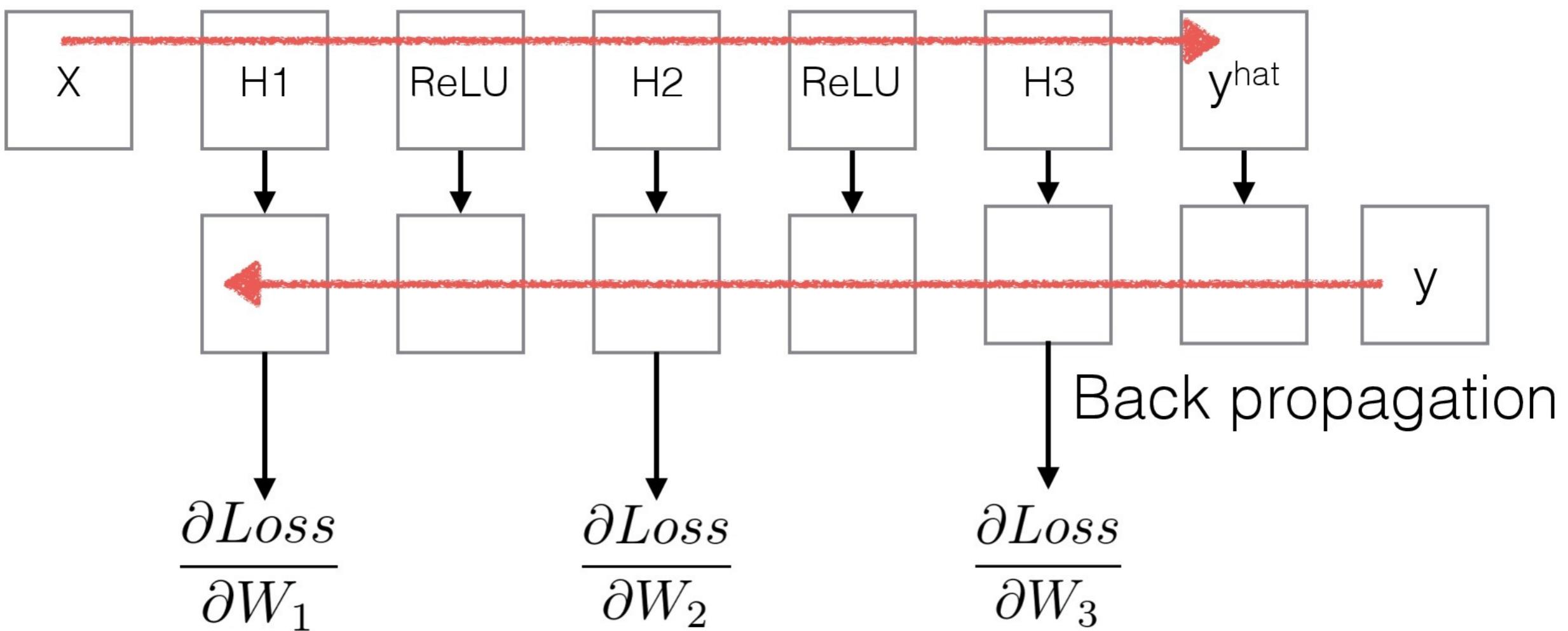
So much of chain rule



So much of chain rule

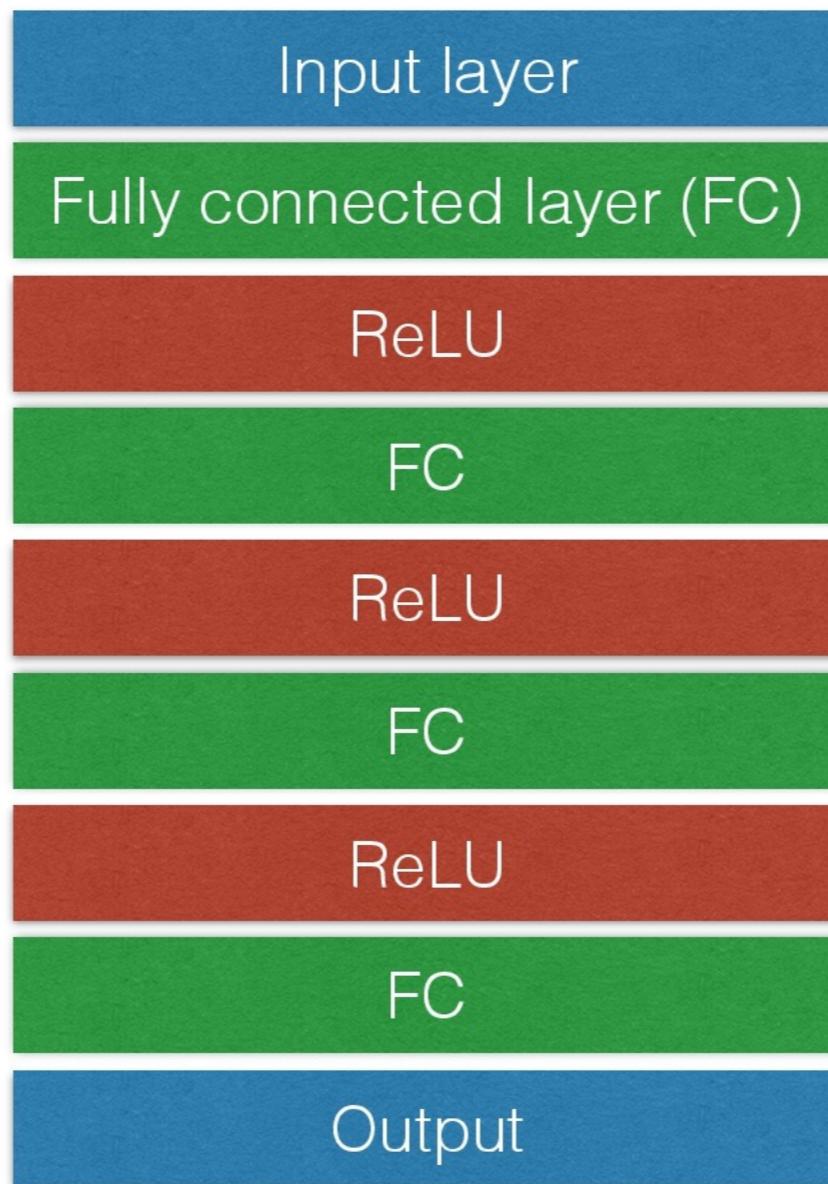
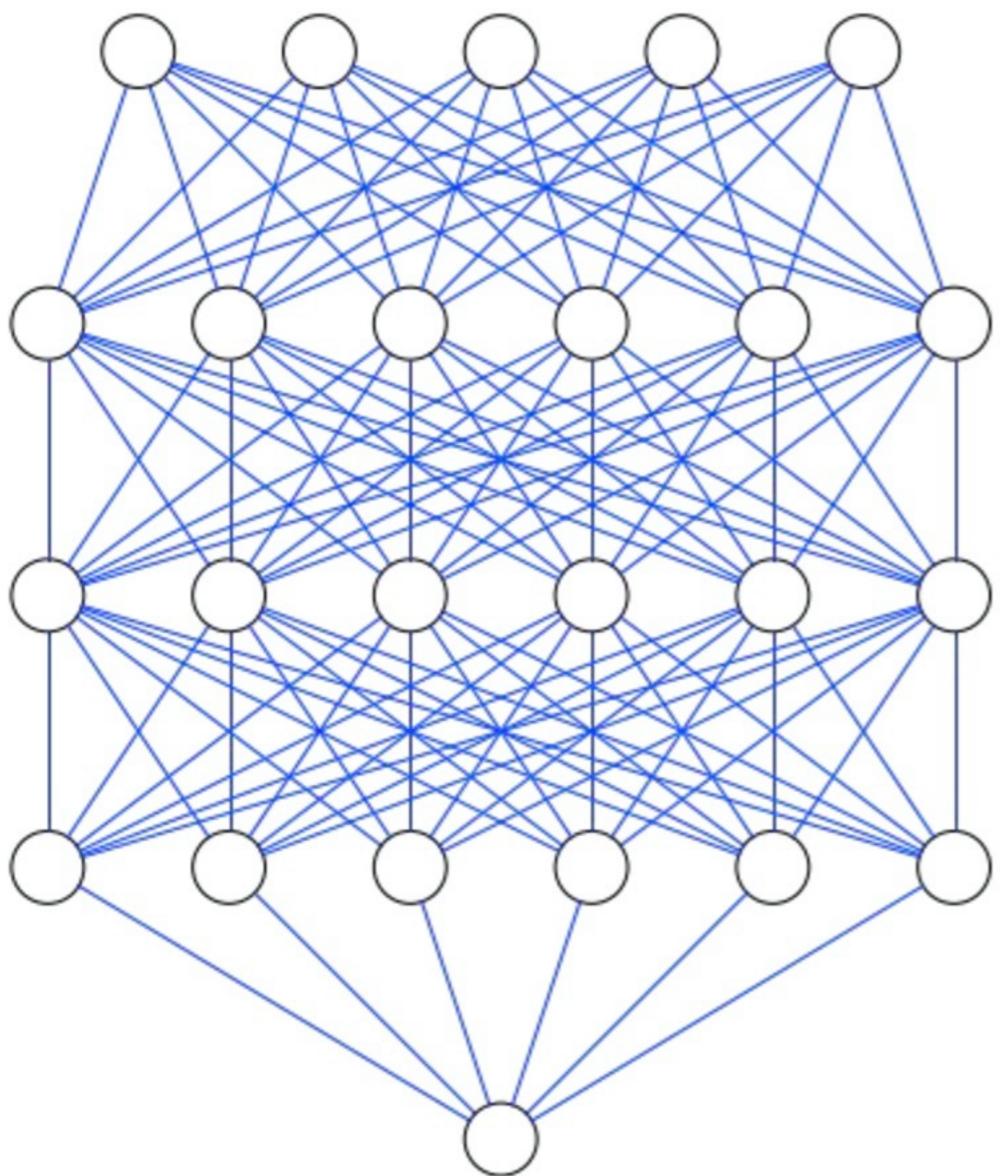


Forward propagation



Modern visualisation

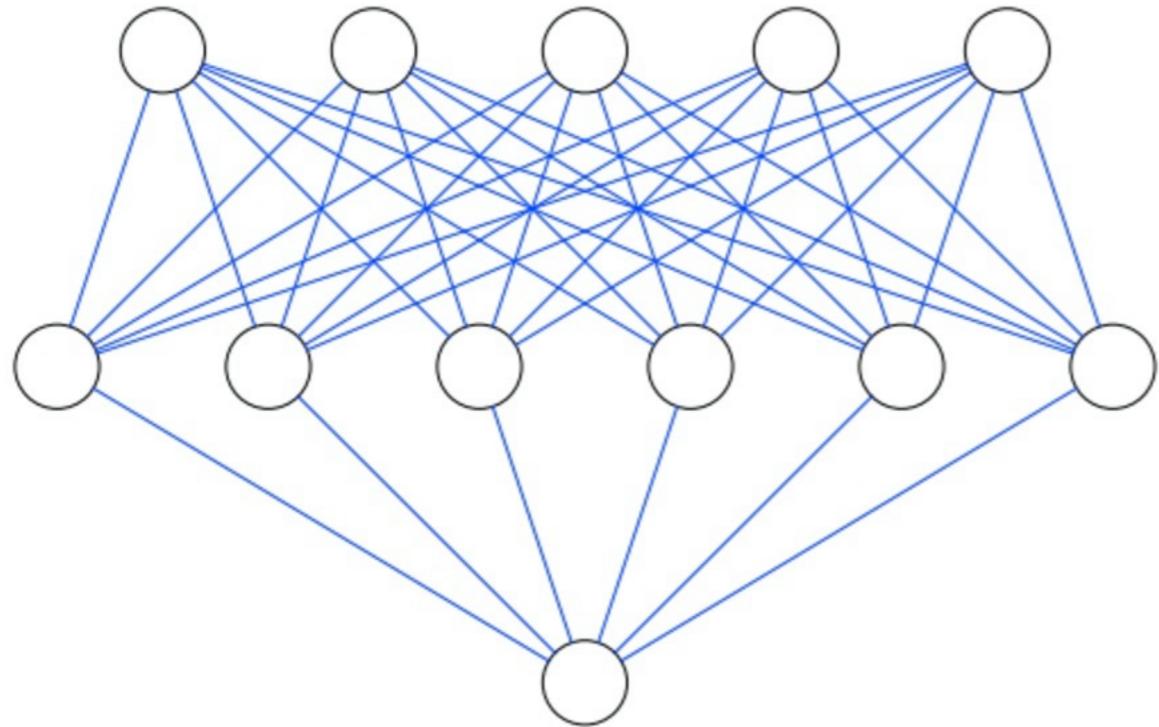
Every layer is just a function $f(X)$



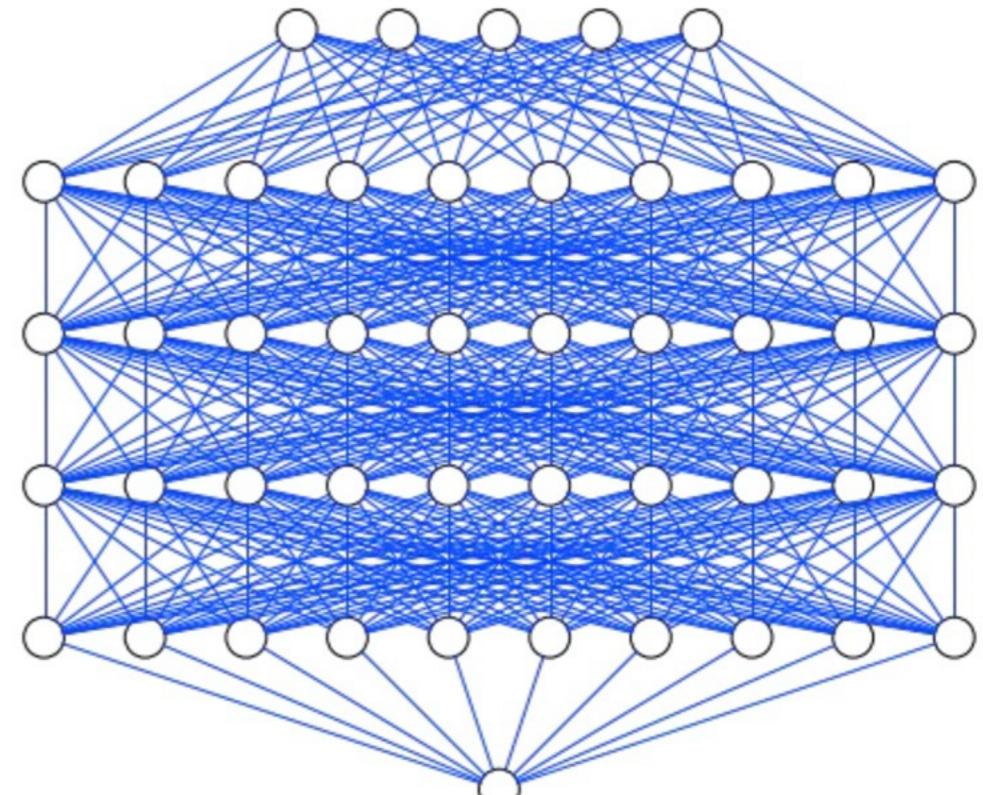
$XW + b$
 $\max(0, X)$
 $XW + b$
 $\max(0, X)$
 $XW + b$
 $\max(0, X)$
 $XW + b$

Non-convex

- With great number of weights comes great number of local minimums. Correct?
- In which case it should be the problem: in the shallow or deep network?



36 weights



360 weights

Non-convex

- Great number of dimensions allow you to pass by any “mountains” in the loss function!
- Yann LeCun “Who is afraid of non-convex loss functions”

Non-convex

- Great number of dimensions allow you to pass by any “mountains” in the loss function!
- Yann LeCun “Who is afraid of non-convex loss functions”

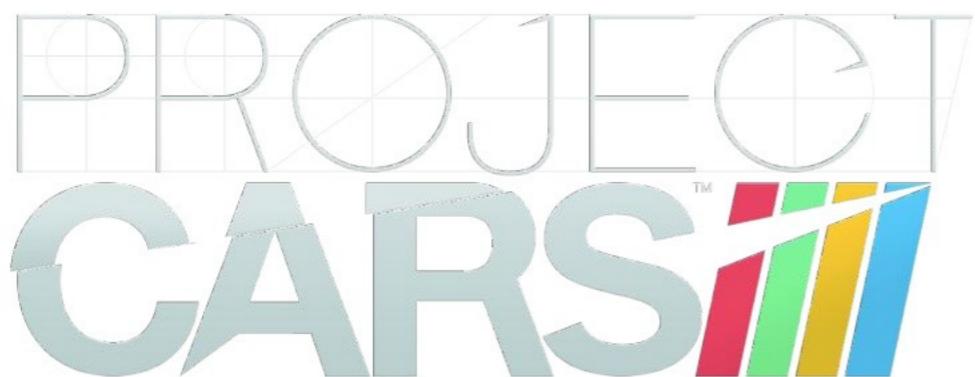
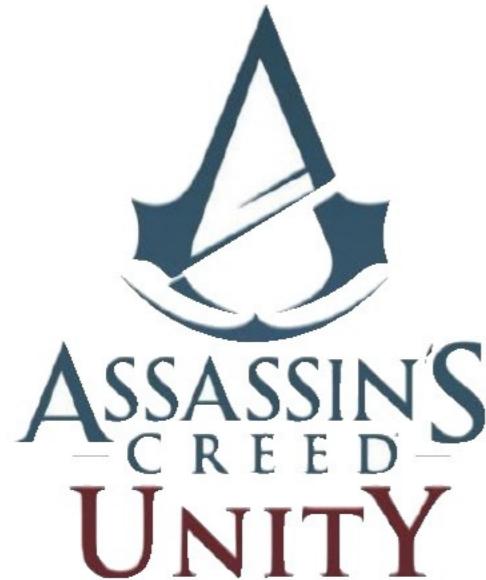
The screenshot shows a YouTube search results page. The search query "who is afraid of non-convex loss" is entered in the search bar. The results list includes a video titled "Eminem - Not Afraid" by EminemVEVO, which has been viewed 737,152,543 times. The video thumbnail features a portrait of Eminem. The sidebar on the left shows navigation links such as Главная, Мой канал, Набирающие популярность, Подписки (32), Просмотренные, and Посмотреть позже (11).

Why is it possible?

I mean, perceptron was created in 1960

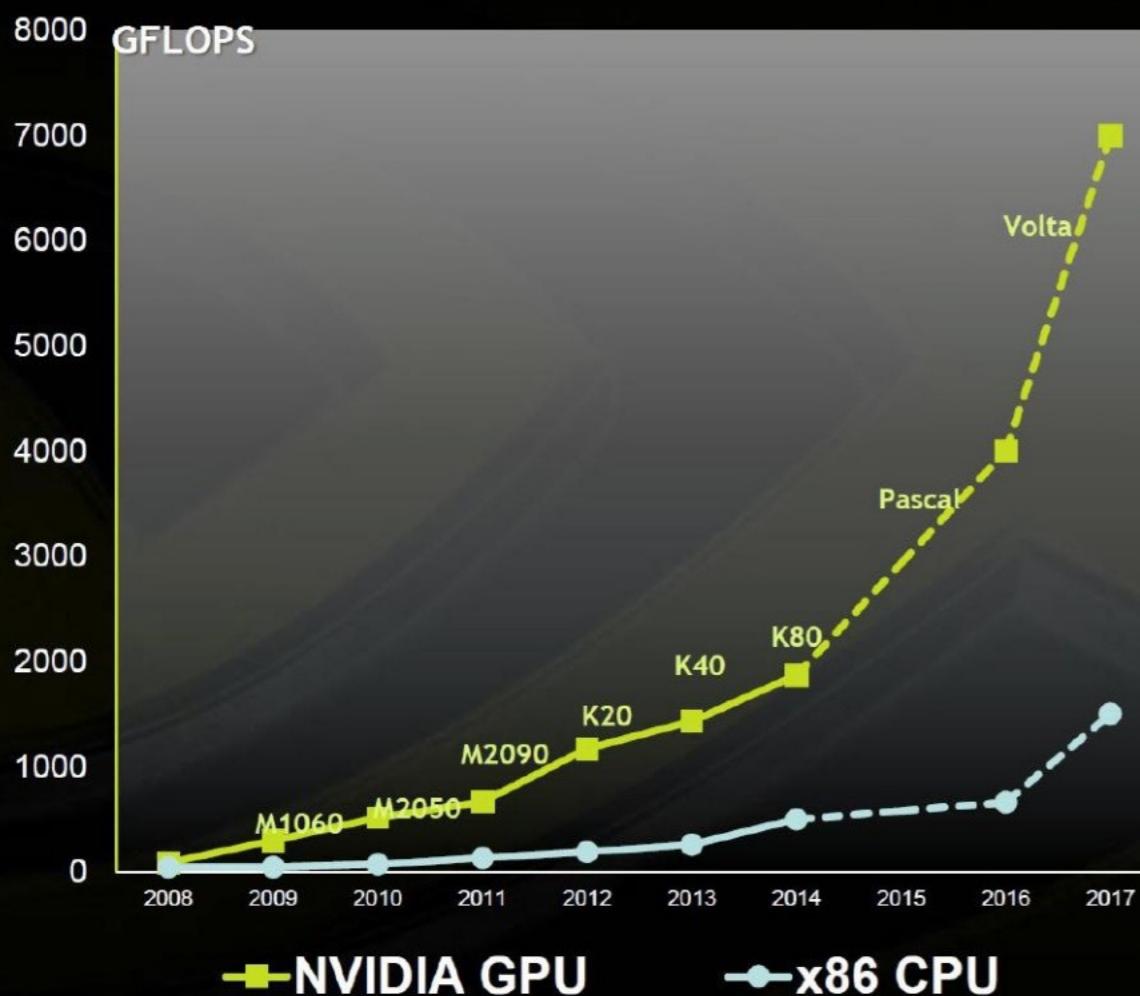
Deep neural nets became popular only in mid-2000s

Why is it possible?

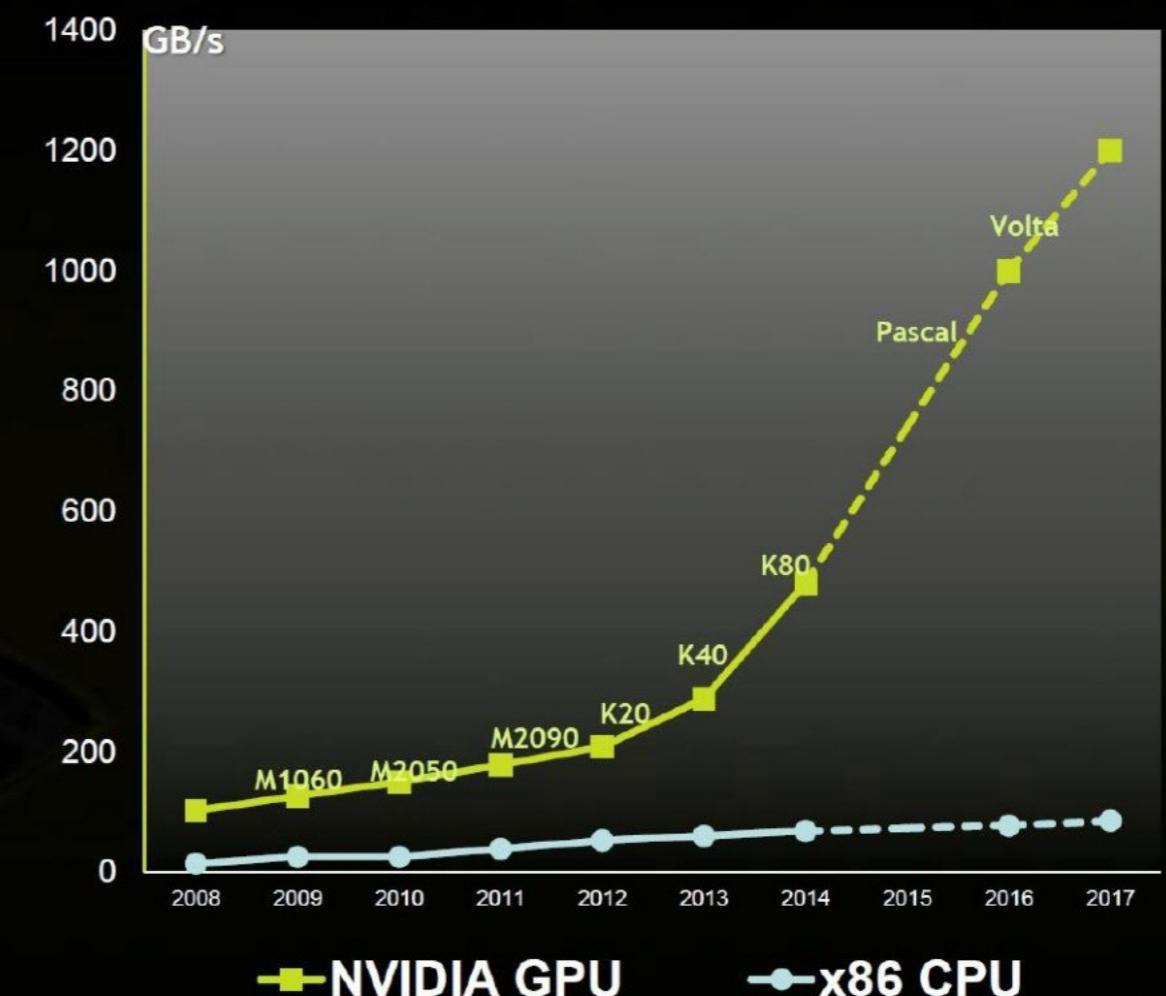


GPU Motivation (I): Performance Trends

Peak Double Precision FLOPS



Peak Memory Bandwidth



Convolution NN

Image recognition



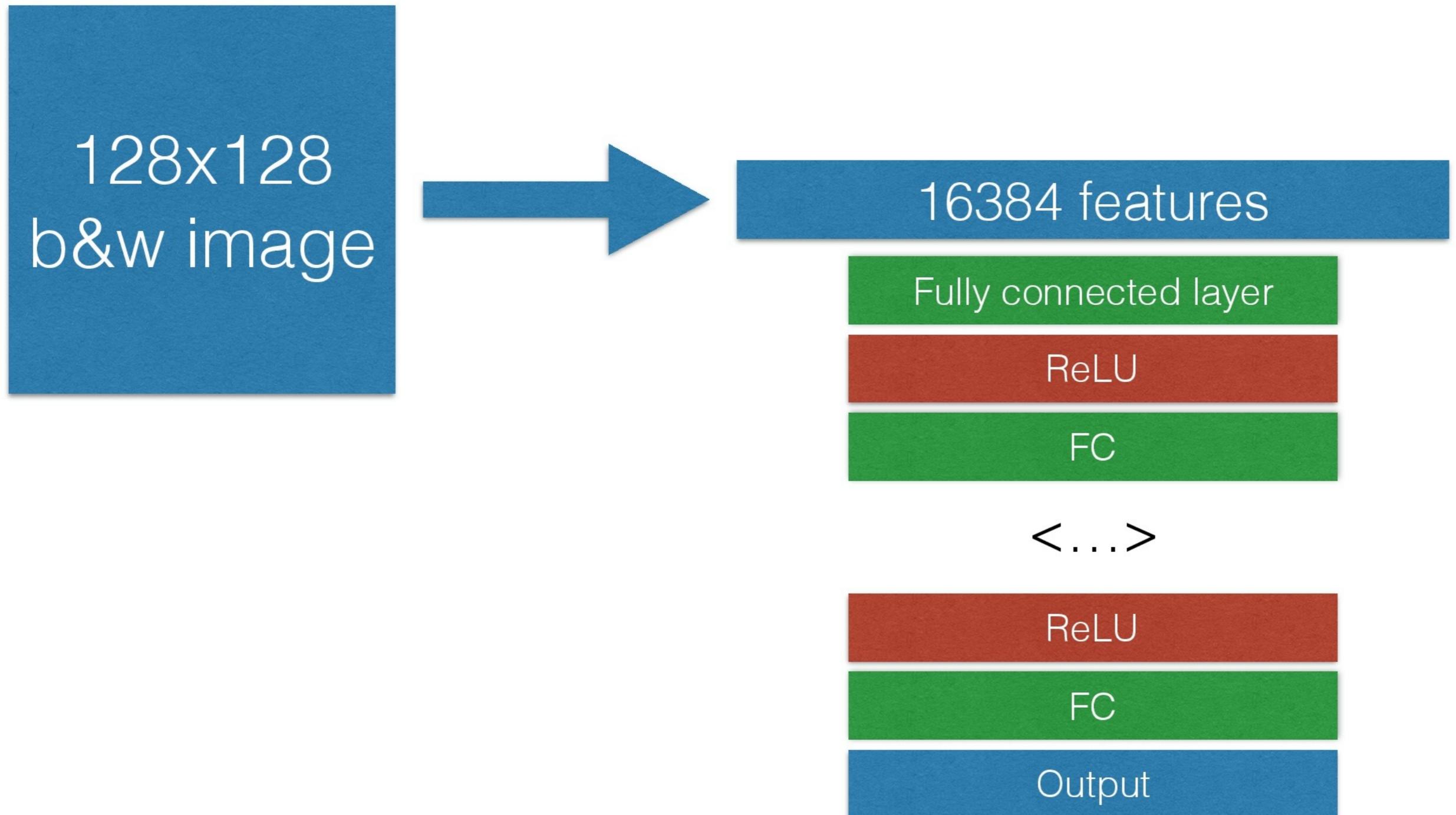
Image recognition



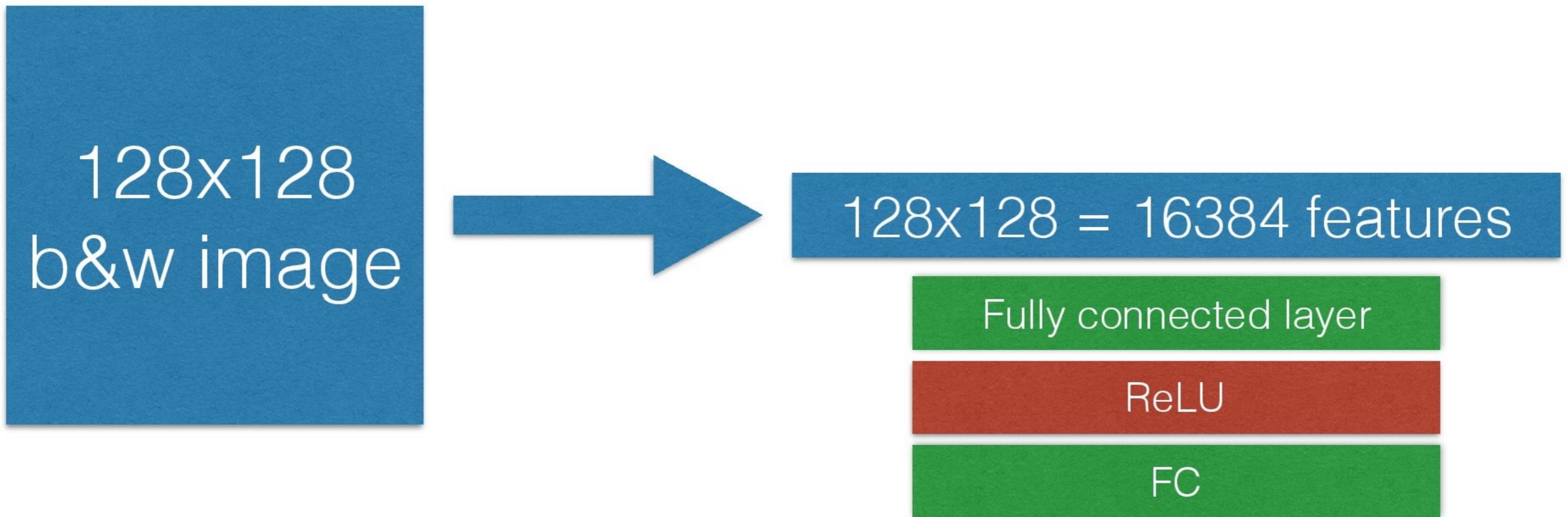
Image recognition — 2



Pixel remembering



Pixel remembering



- A lot of weights!
- Information about relative position is lost
- No translation invariance

<...>



Transitional invariance



Transitional invariance

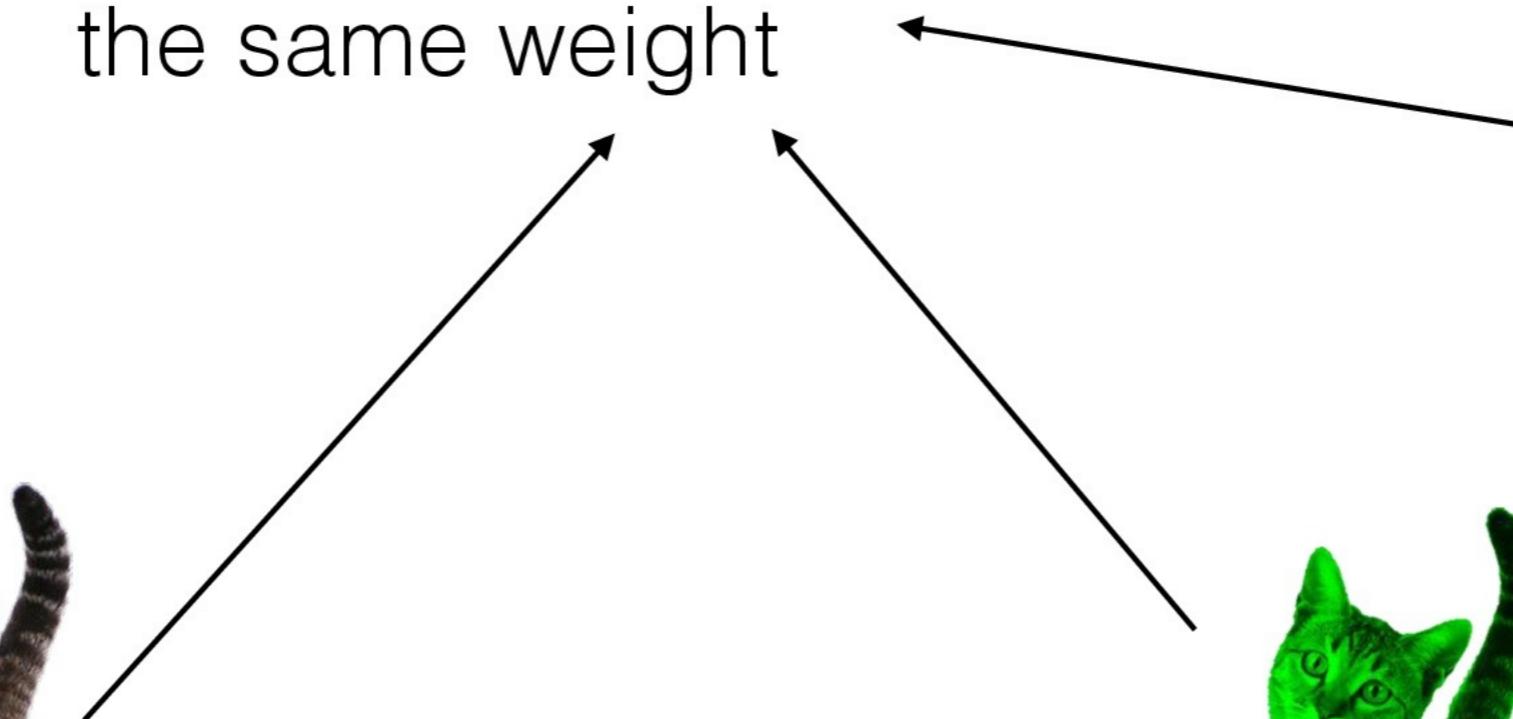


Color invariance

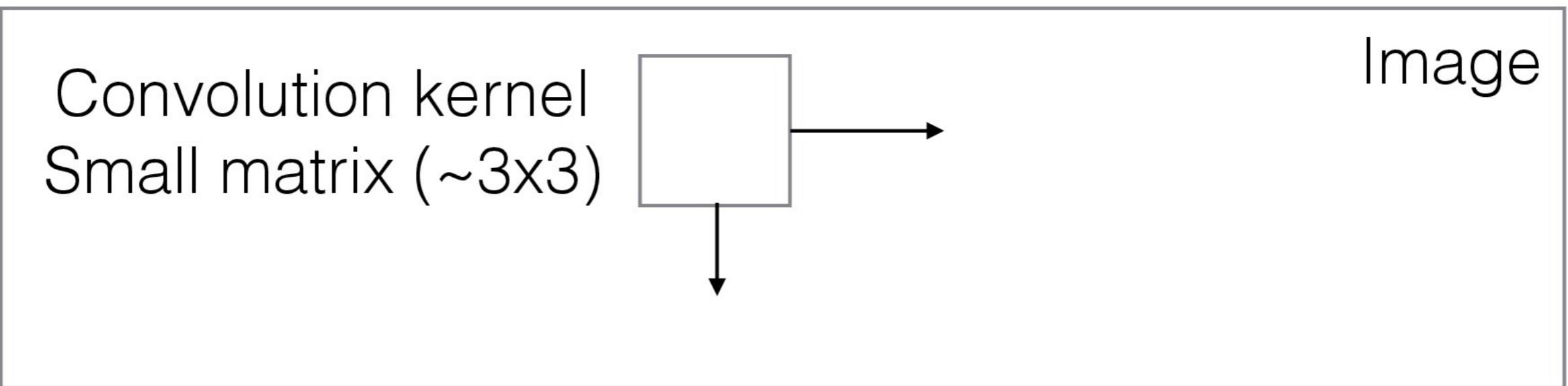


Weight sharing

They all should have
the same weight

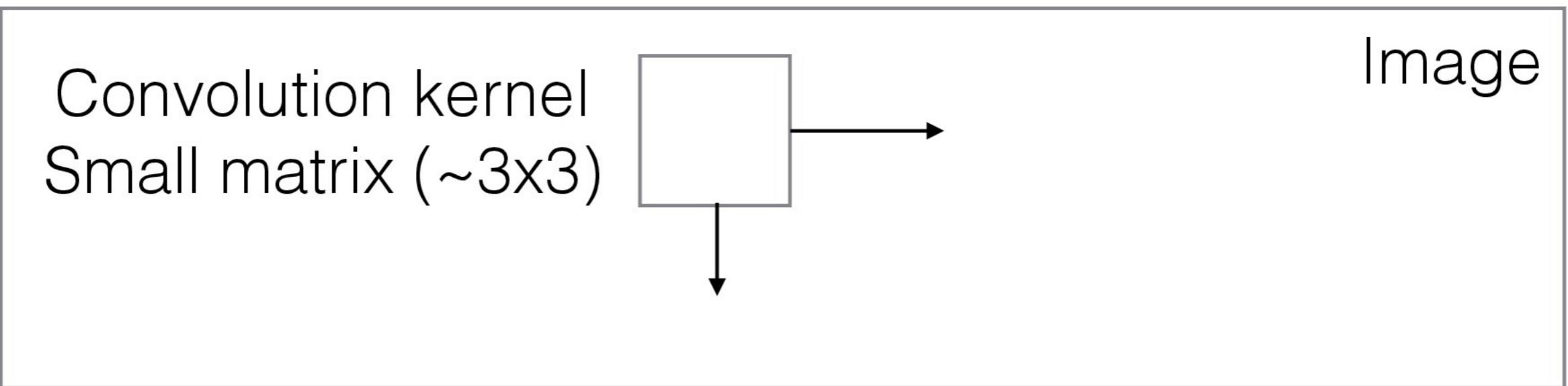


Convolution



In every point of image we multiply kernel on the pixels intensity and sum up.

Convolution



In every point of image we multiply kernel on the pixels intensity and sum up.

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

Image

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

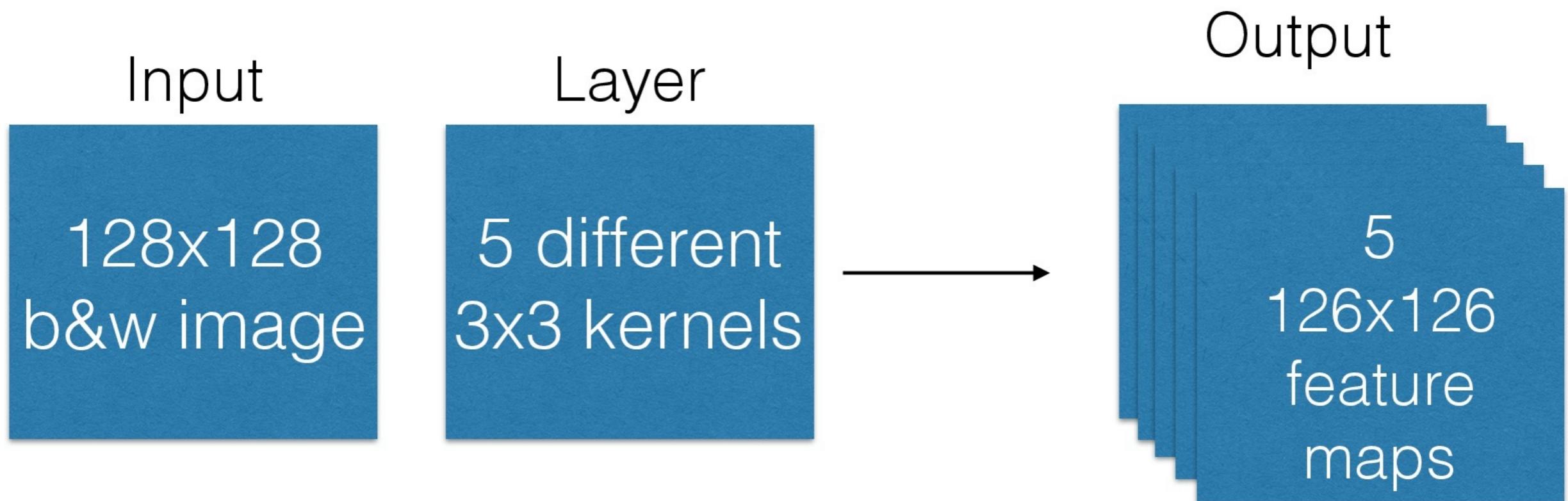
Kernel

2	2	-2
1	0	-1
2	0	-2

Result:
Feature map (5-2)x(5-2)

Convolutional layer

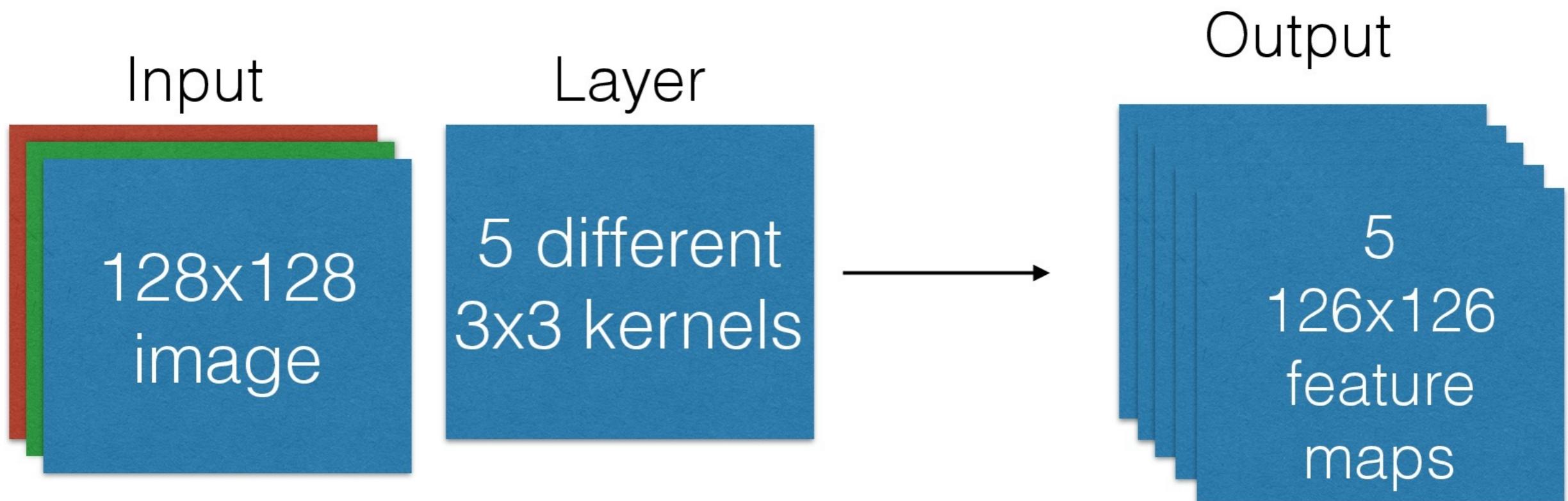
Only 45 weights!



May be considered as encoding 1-channel image
to 5-channel image

Convolutional layer

Only 135 weights!



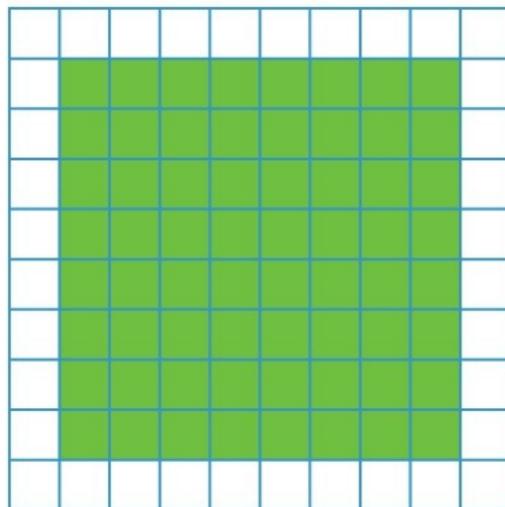
May be considered as encoding 3-channel image
to 5-channel image

Reduce the output

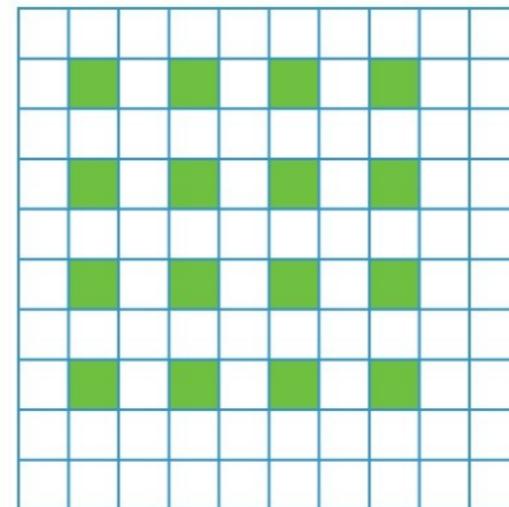
5x126x126 output from convolution is pretty big

Strategy 1: Strides

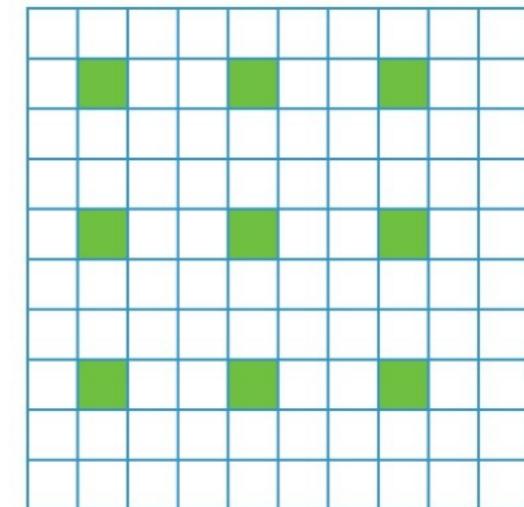
We calculated convolution
at green points



Stride = 1



Stride = 2



Stride = 3

Very aggressive

Reduce the output

5x126x126 output from convolution is pretty big

Strategy 2: Pooling

2	4	5	7	3	-2
-2	0	0	4	9	9
1	0	-1	2	1	1
1	1	6	3	7	2
3	4	0	-2	3	0
3	0	5	1	0	0

Feature Map

2	4	5	7	3	-2
-2	0	0	4	9	9
1	0	-1	2	1	1
1	1	6	3	7	2
3	4	0	-2	3	0
3	0	5	1	0	0

Pool size=2

Max
Pooling

Average
Pooling

4	7	9
1	6	7
4	5	3

1	4	4,8
0,8	2,5	2,8
2,5	1	0,8

Typical DNN

Input	3-channel image	3x128x128	49152
Convolution	6 kernels, 5x5	6x124x124	92256
MaxPool	Pool size = 2	6x62x62	19220
Convolution	12 kernels, 3x3	12x60x60	43200
MaxPool	Pool size = 2	12x30x30	43200
Convolution	20 kernels, 3x3	20x28x28	10800
MaxPool	Pool size = 2	20x14x14	3920
Convolution	40 kernels, 3x3	40x12x12	5760
MaxPool	Pool size = 2	40x6x6	1440
FC	512 hidden neurones		
ReLU			
FC	256 hidden neurones		
ReLU			
FC	1 neuron		

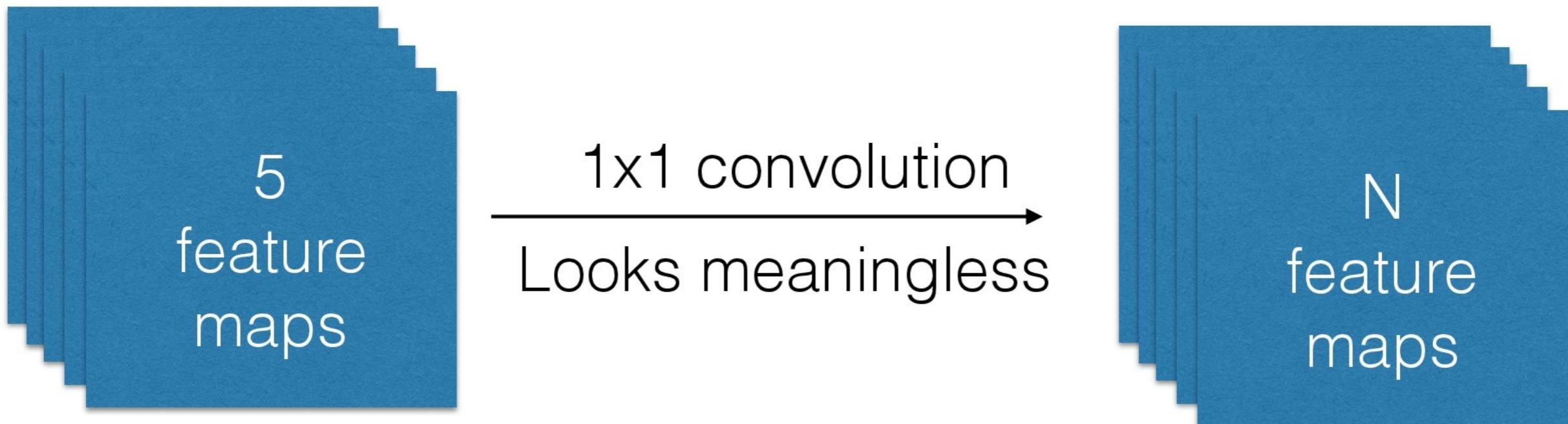
Typical DNN

Input	3-channel image	3x128x128	49152
Convolution	6 kernels, 5x5	6x124x124	92256
MaxPool	Pool size = 2	6x62x62	19220
Convolution	12 kernels, 3x3	12x60x60	43200
MaxPool	Pool size = 2	12x30x30	43200
Convolution	20 kernels, 3x3	20x28x28	10800
MaxPool	Pool size = 2	20x14x14	3920
Convolution	40 kernels, 3x3	40x12x12	5760
MaxPool	Pool size = 2	40x6x6	1440
FC	512 hidden neurones		
ReLU			
FC	256 hidden neurones		
ReLU			
FC	1 neuron		

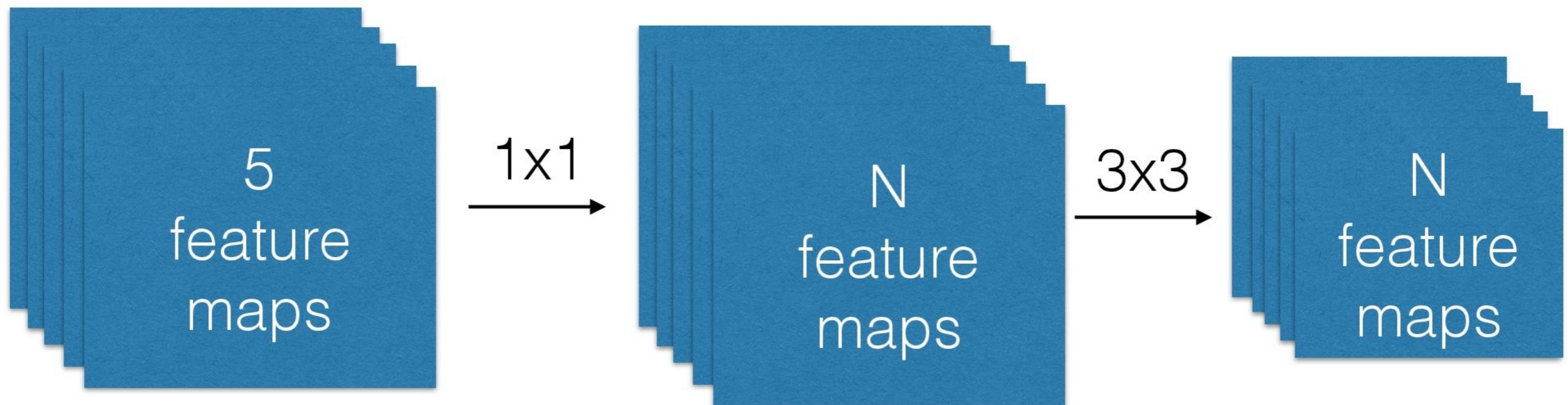
Nonlinearity everywhere

The diagram illustrates the architecture of a typical Deep Neural Network (DNN). The flow of data is shown by red arrows connecting the layers. The input is a 3-channel image (3x128x128). It passes through several layers: Convolution (6 kernels, 5x5), MaxPool (pool size = 2), Convolution (12 kernels, 3x3), MaxPool (pool size = 2), Convolution (20 kernels, 3x3), MaxPool (pool size = 2), Convolution (40 kernels, 3x3), MaxPool (pool size = 2), FC (512 hidden neurones), ReLU, FC (256 hidden neurones), ReLU, and finally FC (1 neuron). Nonlinearity is applied at every stage.

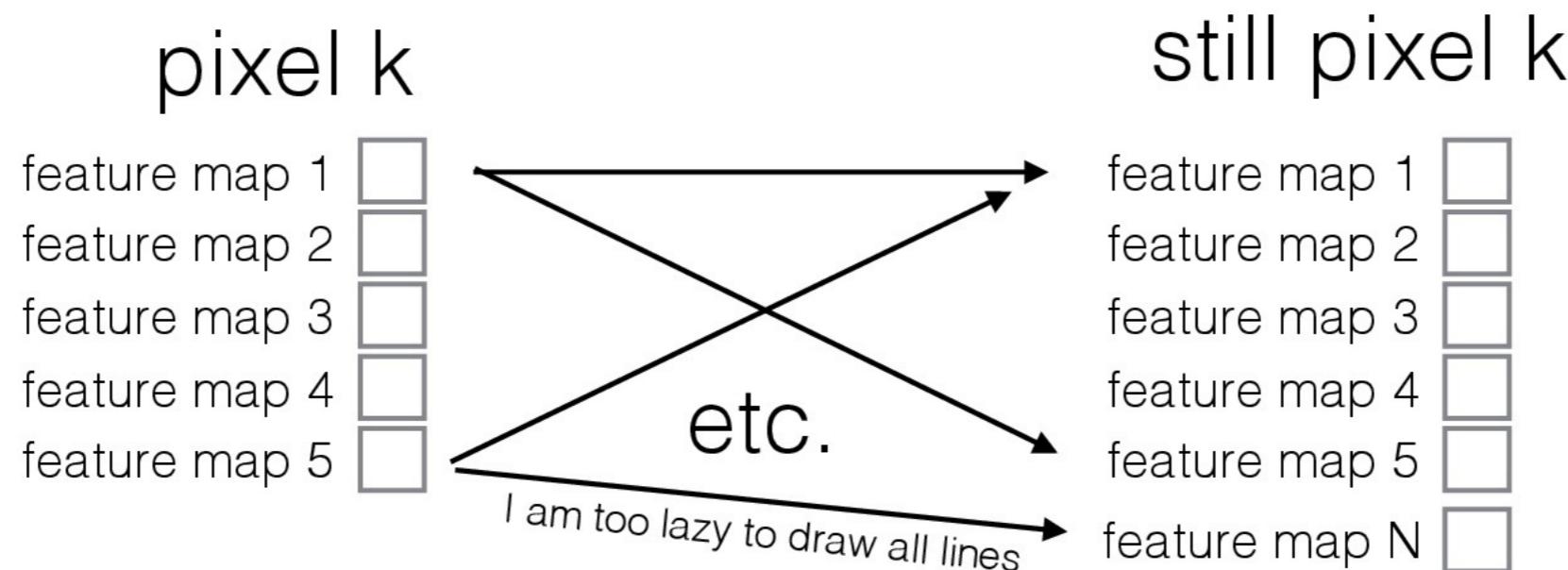
1×1 convolution



1×1 convolution



Now it's deep even on convolution!



Overfitting

Thousands of parameters may (and will) cause overfitting

- Data augmentation if possible
- L2 penalty (add sum of weights squares to loss)
- Dropout

We need more data

Rule of thumb:

at least 10 points per one degree of freedom

Google images use DNN with 37M weights =>
at least 370M images to train

Data augmentation



Cat

Data augmentation



Cat



Still a cat — we can learn on them!

Dropout

Introduced in 2014

In terms of layers as functions:

$$f_{train}(x) = \begin{cases} 0, & \text{with probability } p \\ x, & \text{with probability } 1 - p \end{cases}$$

$$f_{test}(x) = (1 - p)x$$

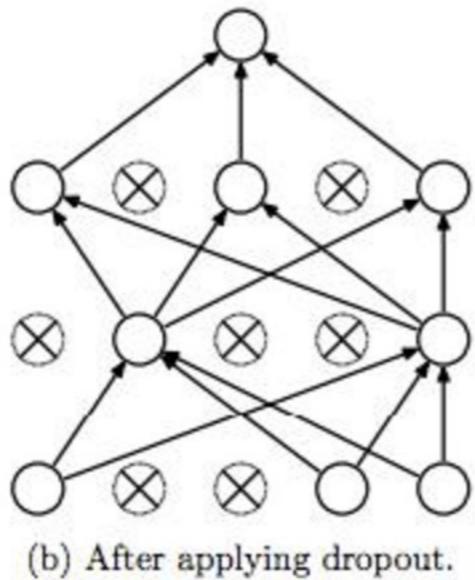
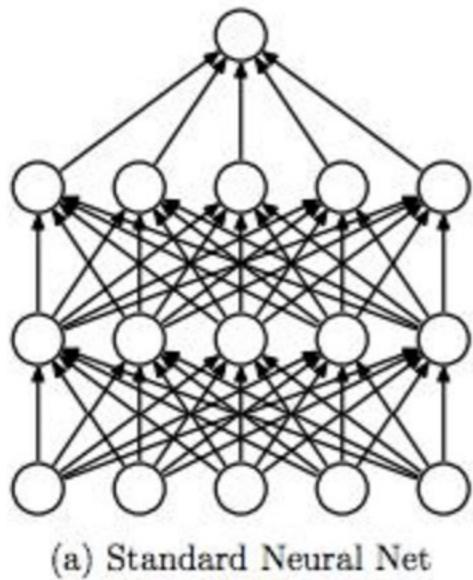
Dropout

Introduced in 2014

In terms of layers as functions:

$$f_{train}(x) = \begin{cases} 0, & \text{with probability } p \\ x, & \text{with probability } 1 - p \end{cases}$$

$$f_{test}(x) = (1 - p)x$$



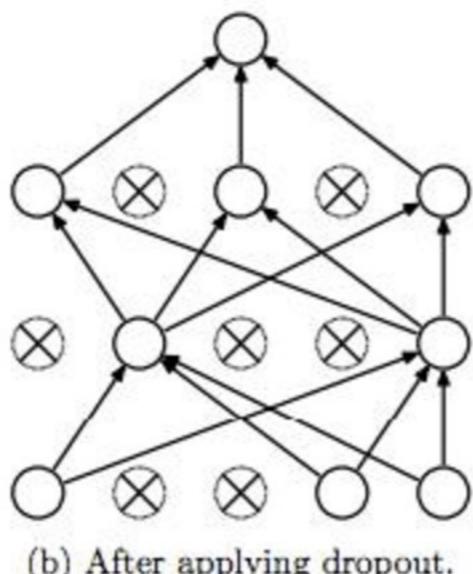
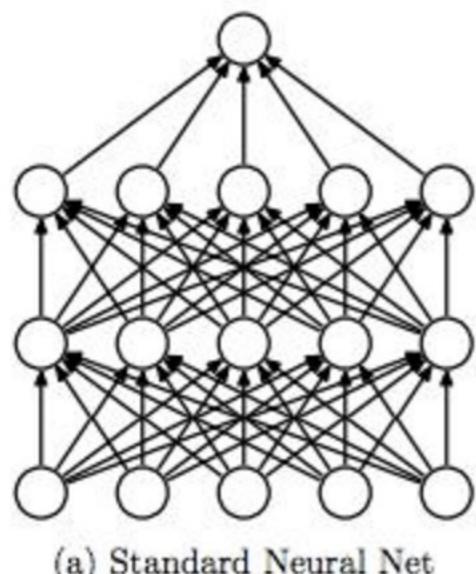
Dropout

Introduced in 2014

In terms of layers as functions:

$$f_{train}(x) = \begin{cases} 0, & \text{with probability } p \\ x, & \text{with probability } 1 - p \end{cases}$$

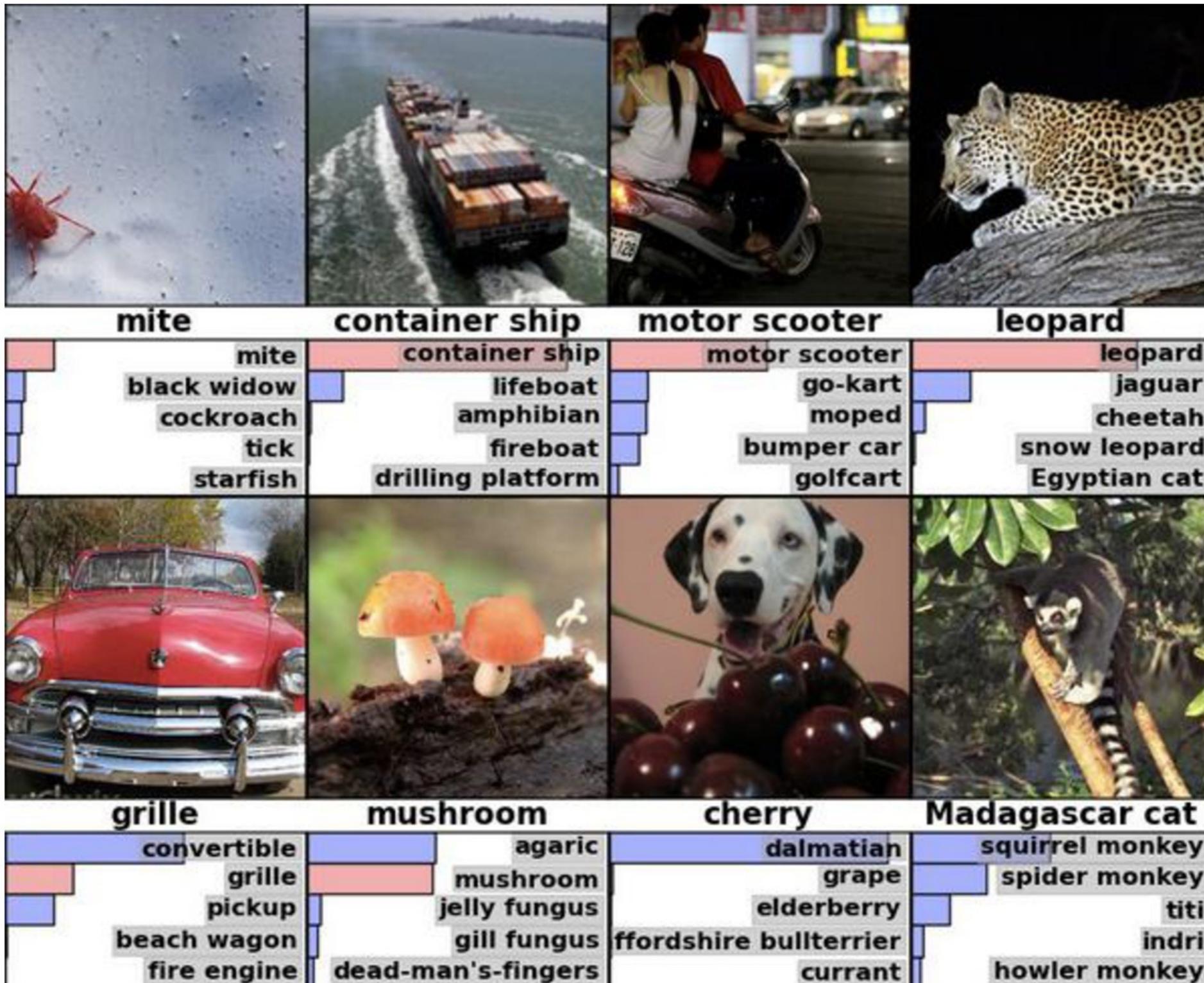
$$f_{test}(x) = (1 - p)x$$



p=0.5 is good choice

Dropout: A Simple Way to Prevent
Neural Networks from Overfitting /
Nitish Srivastava, Geoffrey Hinton,
Alex Krizhevsky, Ilya Sutskever,
Ruslan Salakhutdinov; 15(Jun):
1929–1958, 2014.

DNN performance



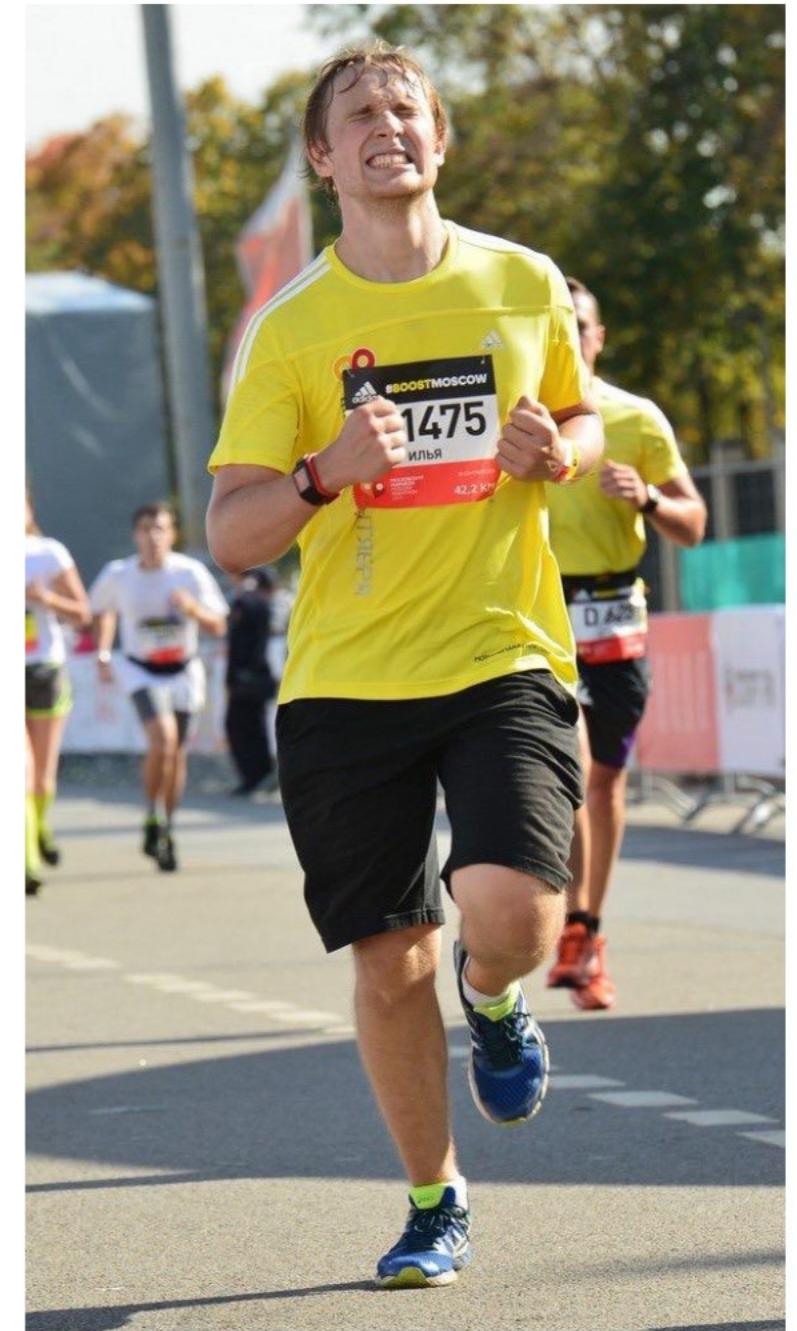
DNN performance



DNN performance



+



The Great Wave off Kanagawa

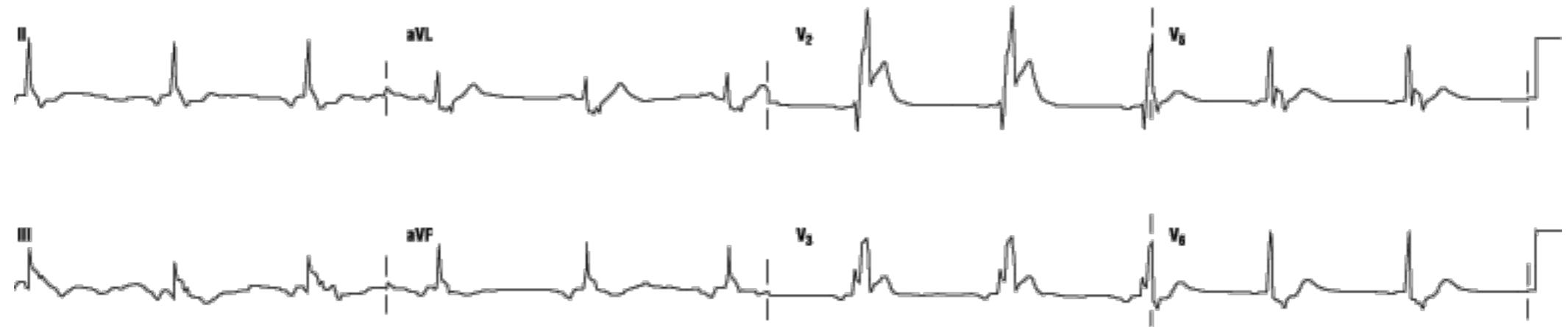
DNN performance



Recurrent Neural Networks



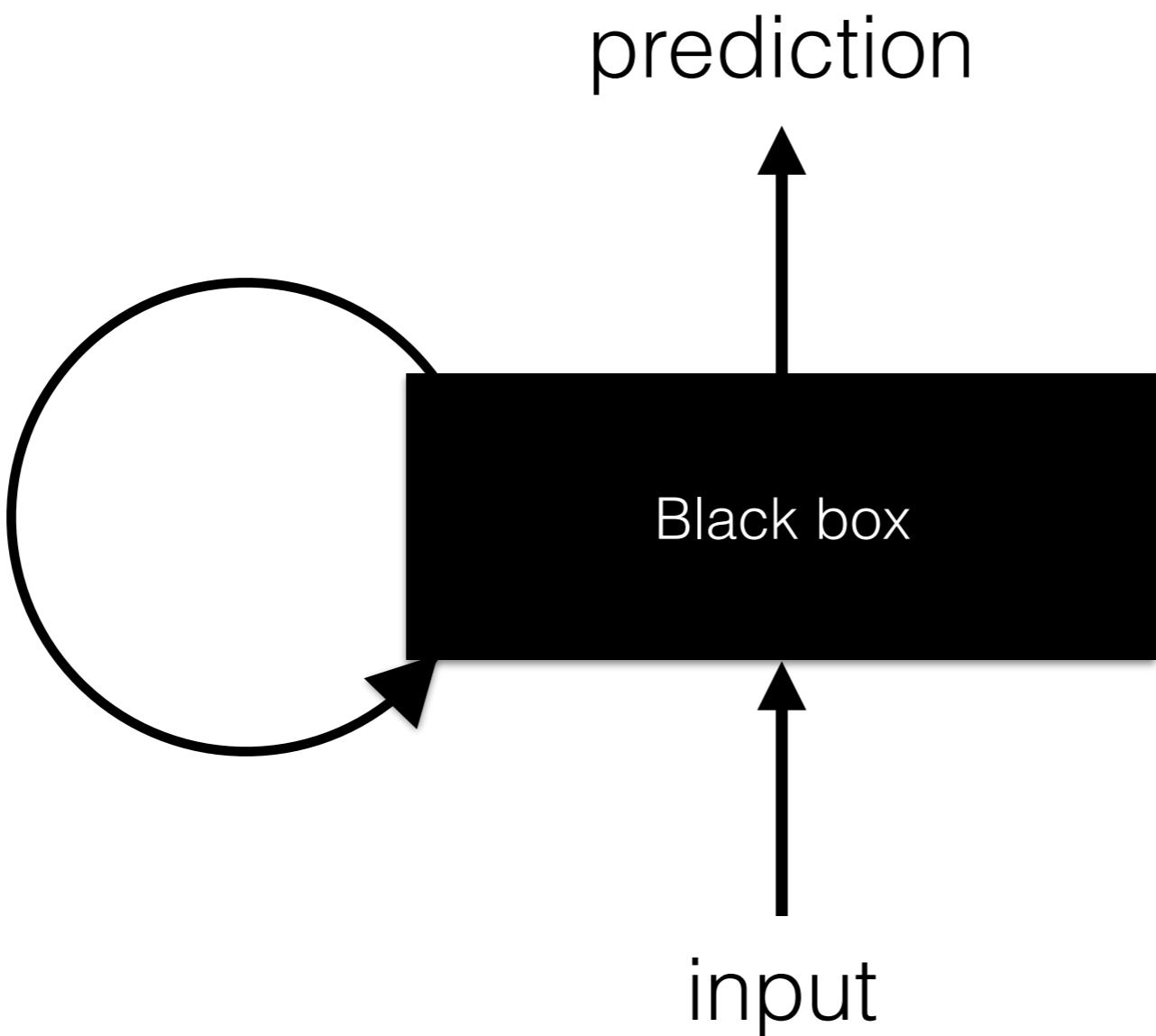
What if I want to process sequences?



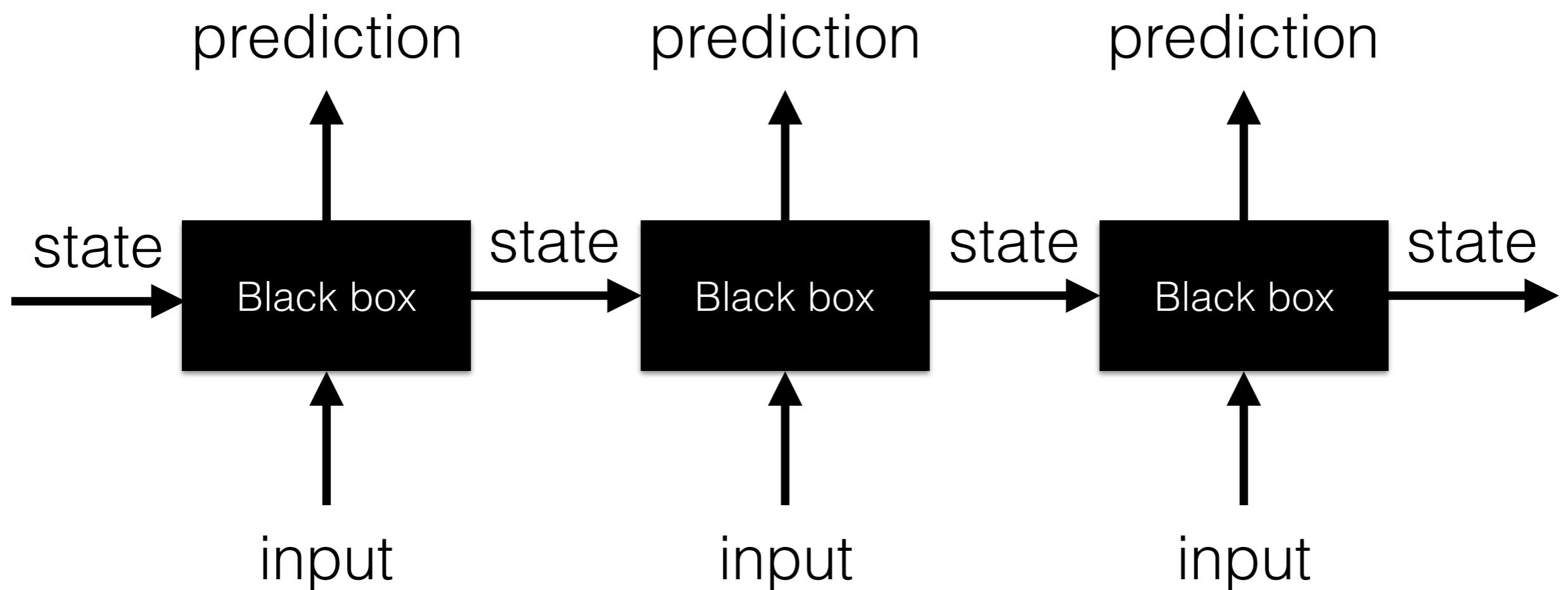
Problem:
We want feed NN
with unfixed
sequence.



Recurrent NN!



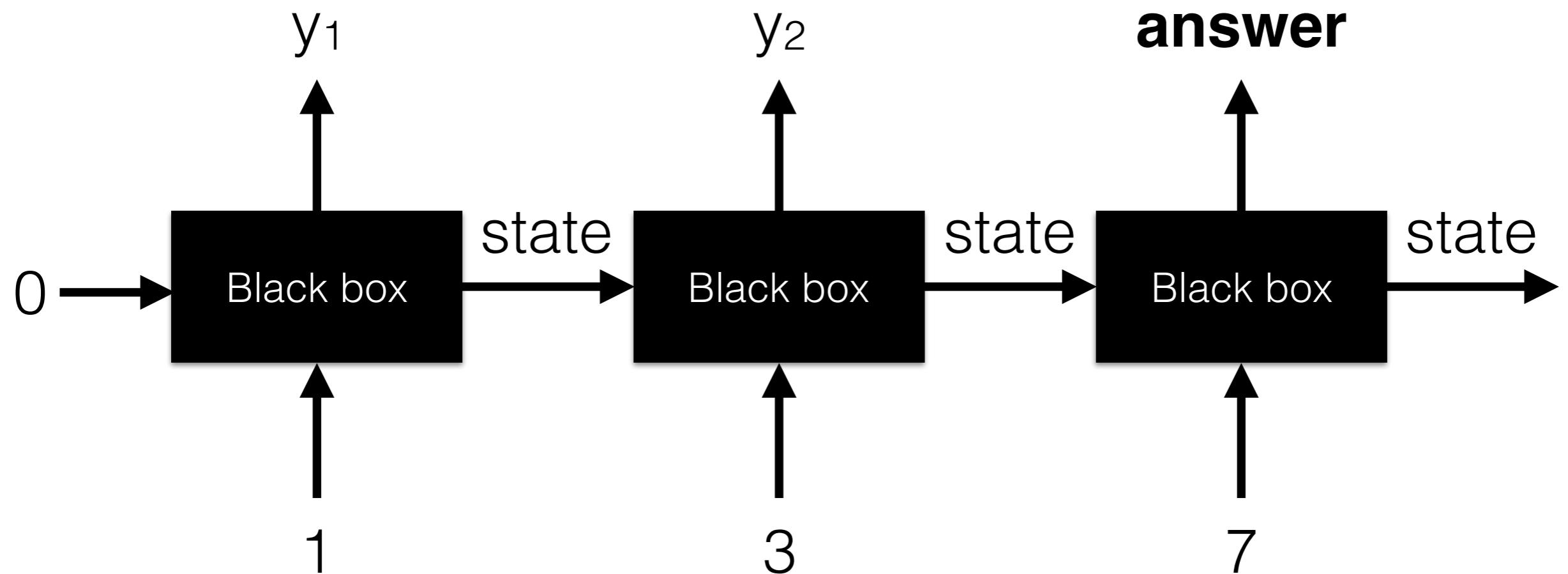
Recurrent NN!



Black Box is the same!

Example:

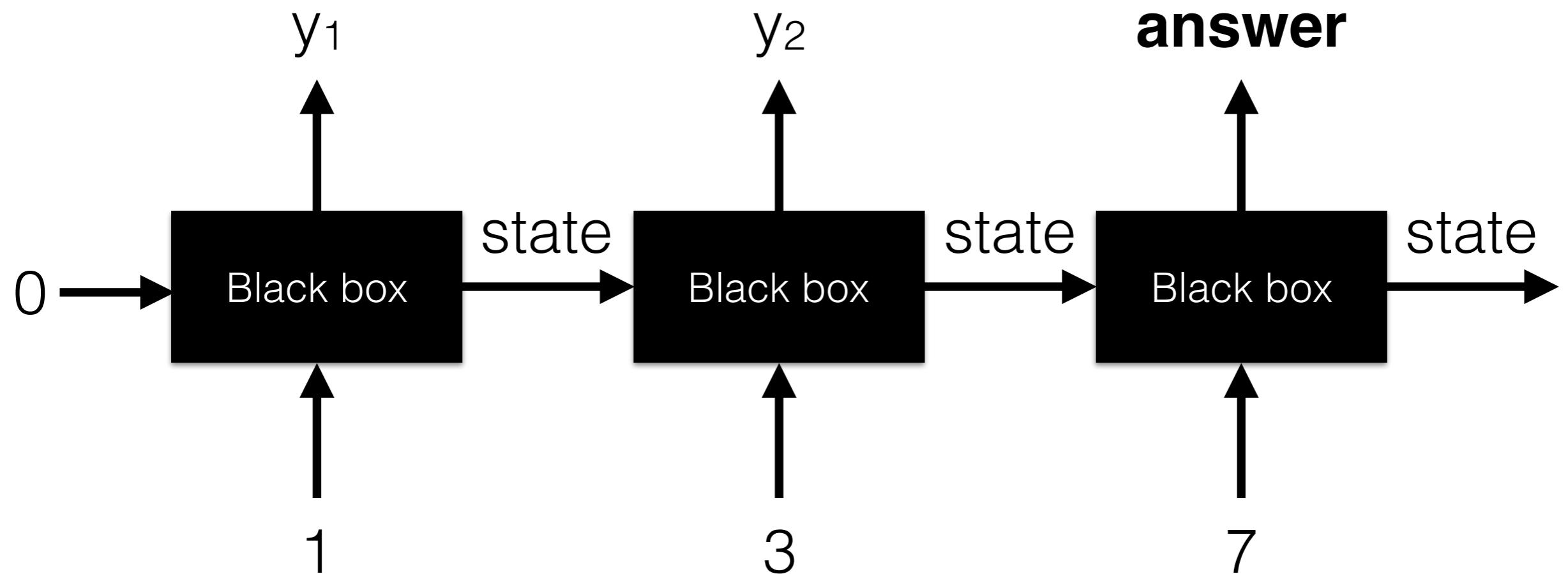
$$X = (1, 3, 7) \quad y=11$$



Example:

$$X = (1, 3, 7) \quad y=11$$

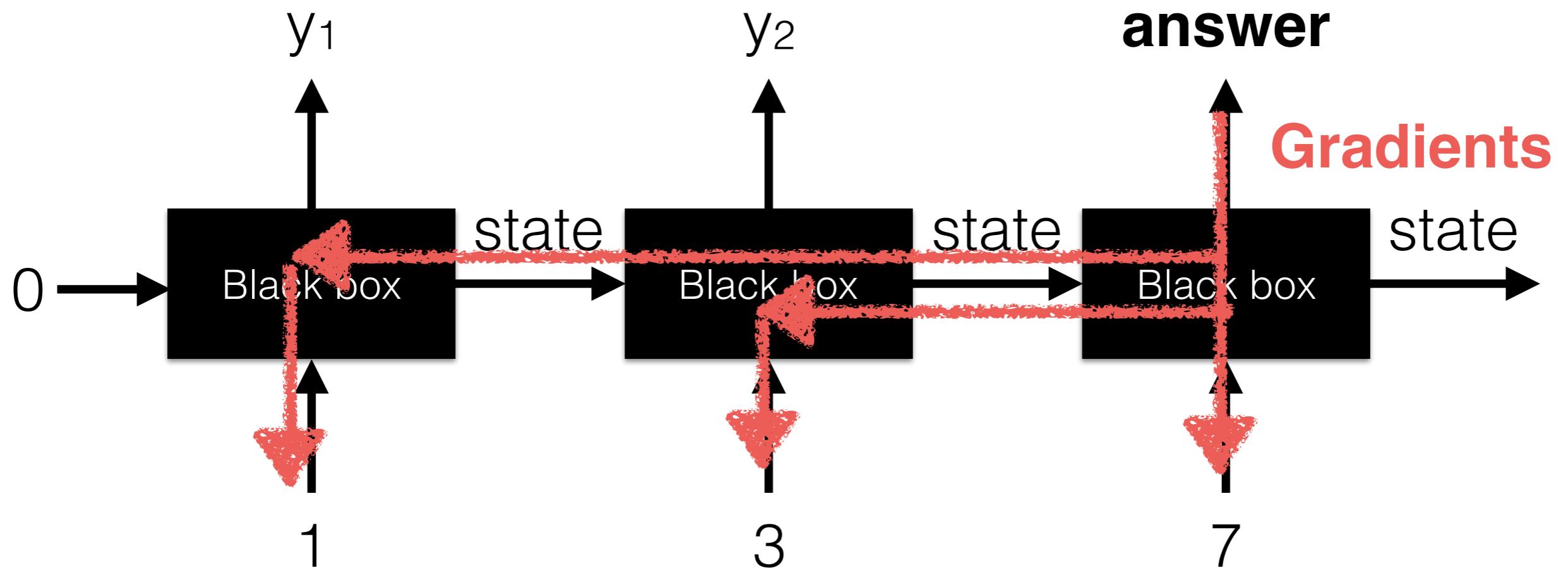
$$\text{Loss} = (\text{answer} - 11)^2$$



Example:

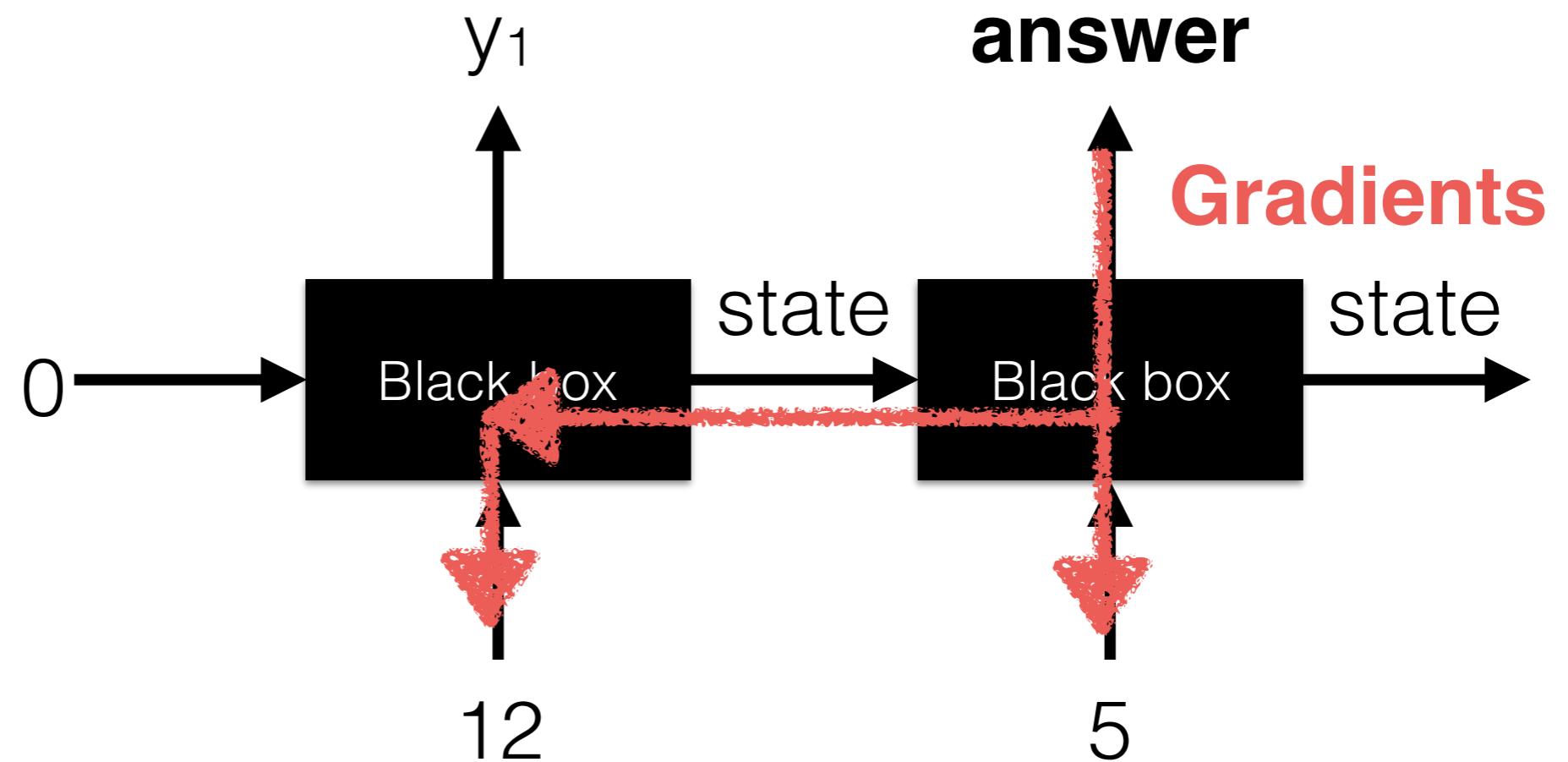
$$X = (1, 3, 7) \quad y=11$$

$$\text{Loss} = (\text{answer} - 11)^2$$

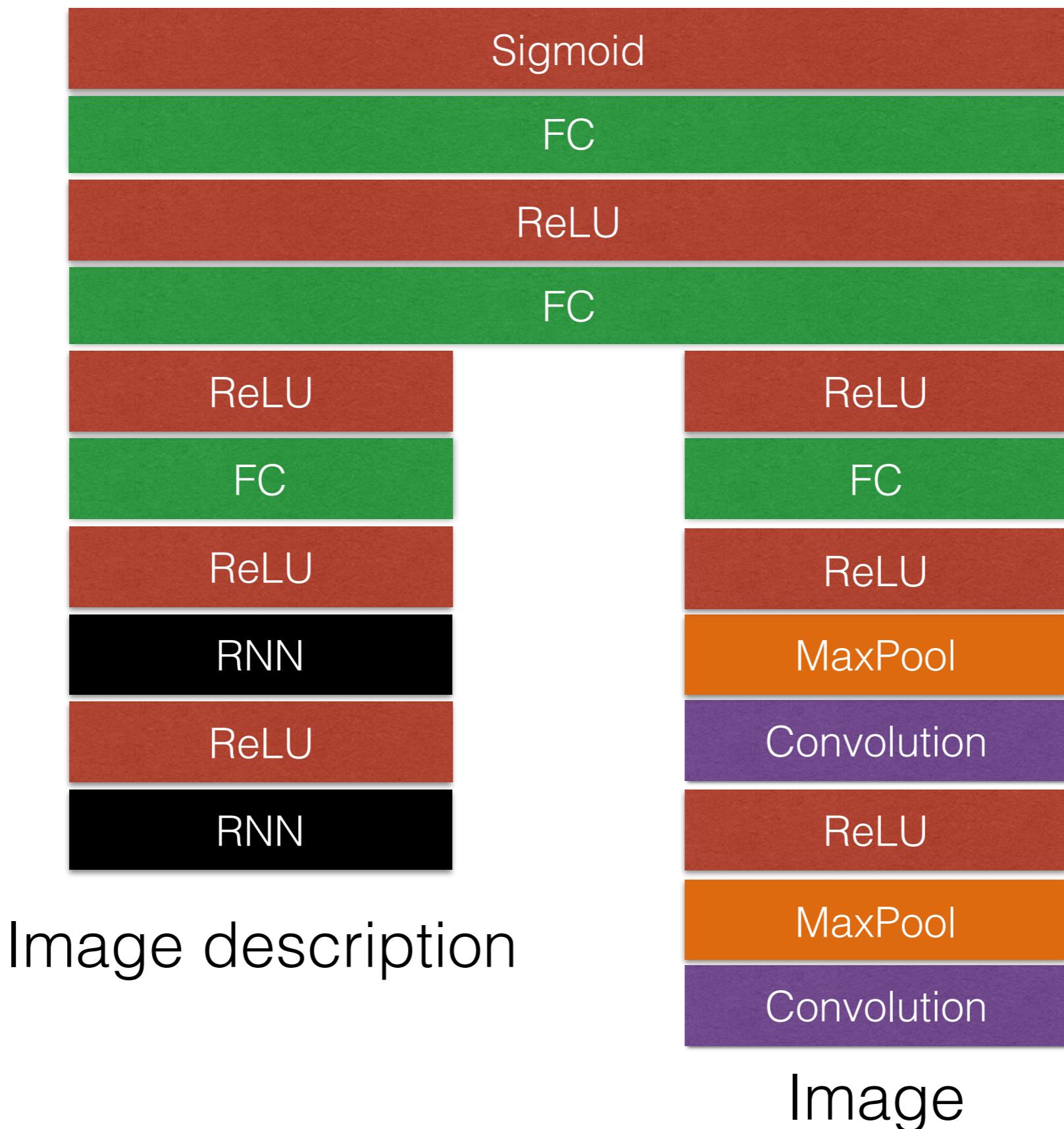


Example:
 $X = (12, 5)$ $y=17$

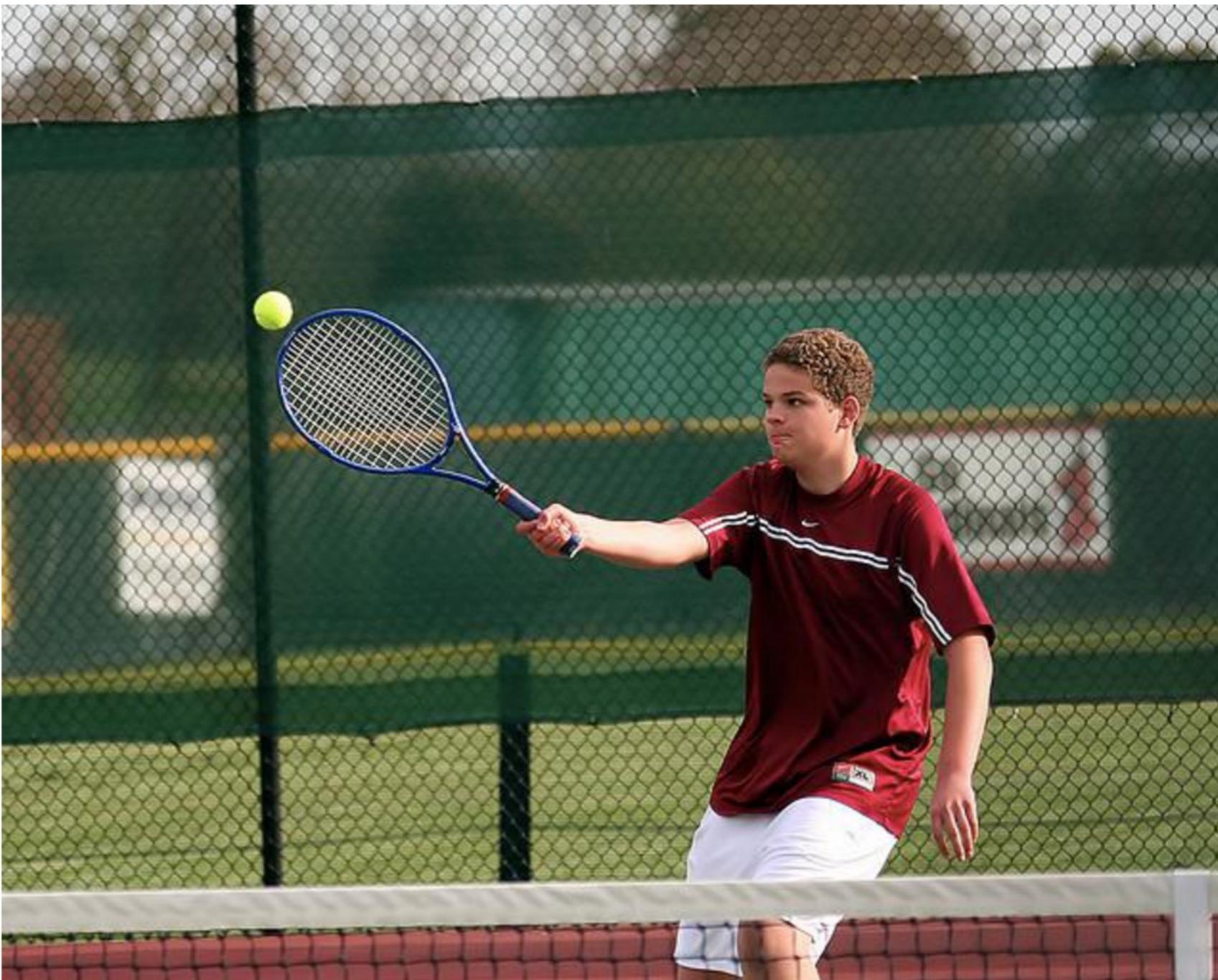
$$\text{Loss} = (\text{answer} - 17)^2$$



We can combine!



DNN performance



a man is playing tennis on a tennis court

DNN performance



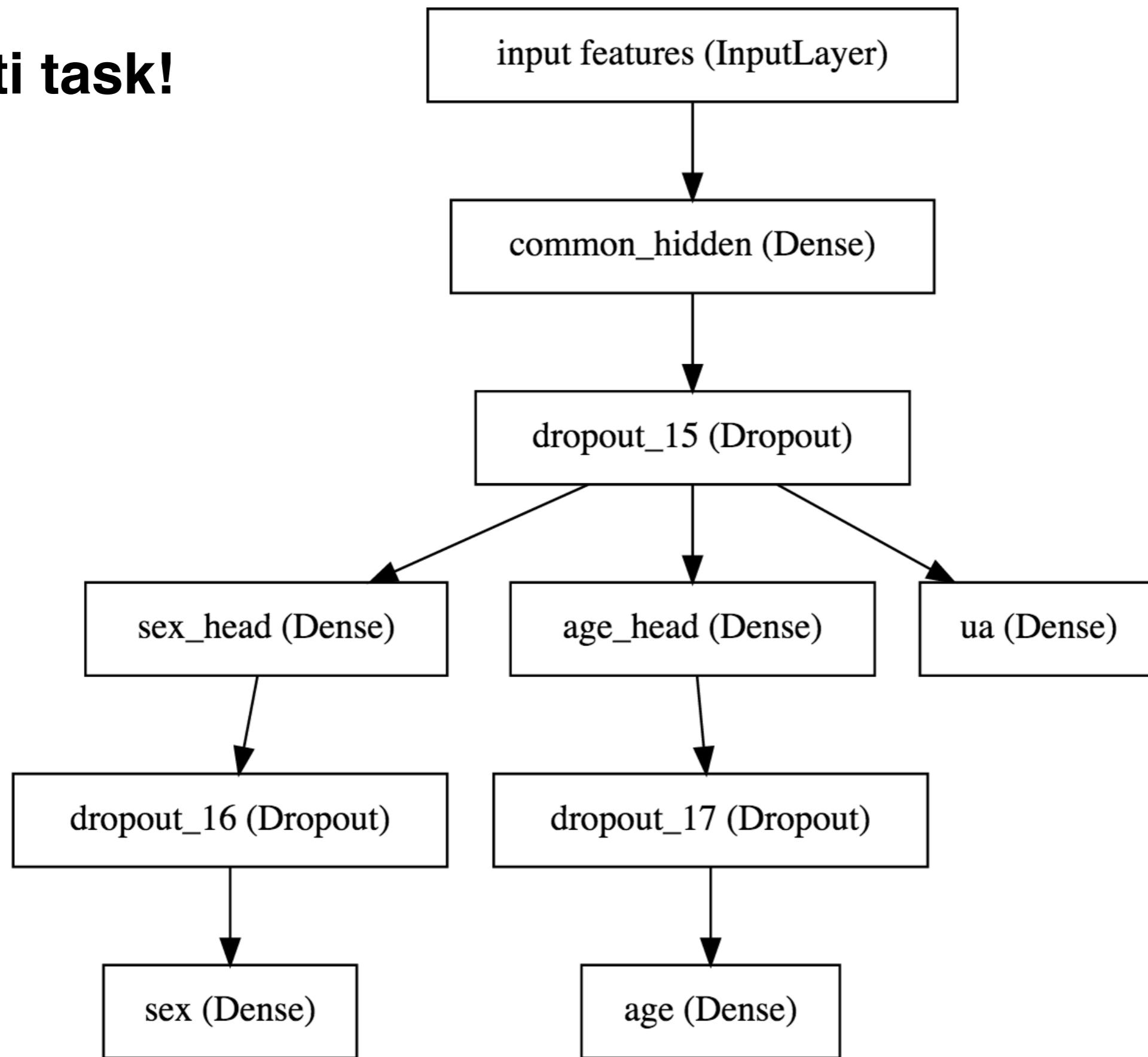
a man sitting on a bus with a dog

DNN performance



a bunch of bananas are hanging from a ceiling

Multi task!



SGD hacks

Weights initialization

W is gaussian, b is constant (or uniform [-1;1])

There are a lot of ways to select best properties
of gaussian distribution for W

One of the best is:

$$W \sim N\left(0, \frac{1}{\sqrt{\# \text{ input neurones}}}\right)$$

Batch norm

Introduced in 2015

Instead of batch X use $\frac{X - \mu_X}{\sigma_X}$

Batch norm

Introduced in 2015

Instead of batch

$$X \quad \text{use} \quad \frac{X - \mu_X}{\sigma_X}$$

It's a layer too!

$$f(X) = \frac{X - \mu_X}{\sigma_X}$$

Really speeds up learning!

Lego

FC

ReLU

Dropout

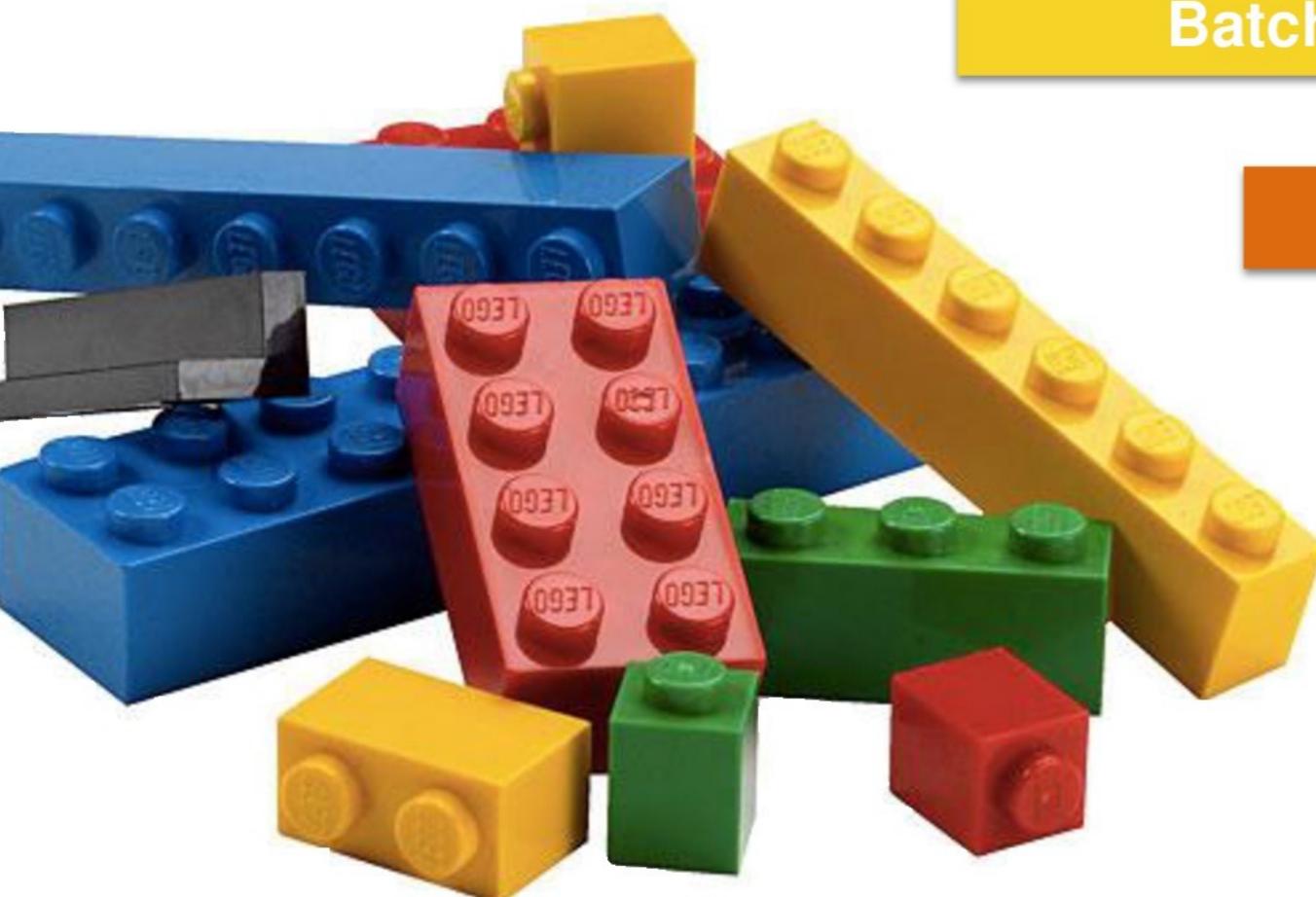
Batch Norm

MaxPool

Convolution

1x1 Convolution

LSTM (RNN)



When to use?

- There are a lot of data
- The data is persistent
- There is some real structure in the data
- All types of images
- You need to complex interaction
- Features are strongly correlated

What to do next?

- Write your own Neural Network?

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity
- Take a look at low-level frameworks:
especially **TensorFlow**

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity
- Take a look at low-level frameworks:
especially **TensorFlow**
- Take a look at high-level **Keras**

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity
- Take a look at low-level frameworks:
especially **TensorFlow**
- Take a look at high-level **Keras**
- Just watch Yann LeCun lectures

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity
- Take a look at low-level frameworks:
especially **TensorFlow**
- Take a look at high-level **Keras**
- Just watch Yann LeCun lectures
- Experiment! (buy GPU before)

What to do next?

- Write your own Neural Network?
- Take Deep Learning course @ udacity
- Take a look at low-level frameworks:
especially **TensorFlow**
- Take a look at high-level **Keras**
- Just watch Yann LeCun lectures
- Experiment! (buy GPU before)
- **Decrease your learning rate!**

“Deep learning is shallow
next to the depth of your greatness.”

–Kaggle