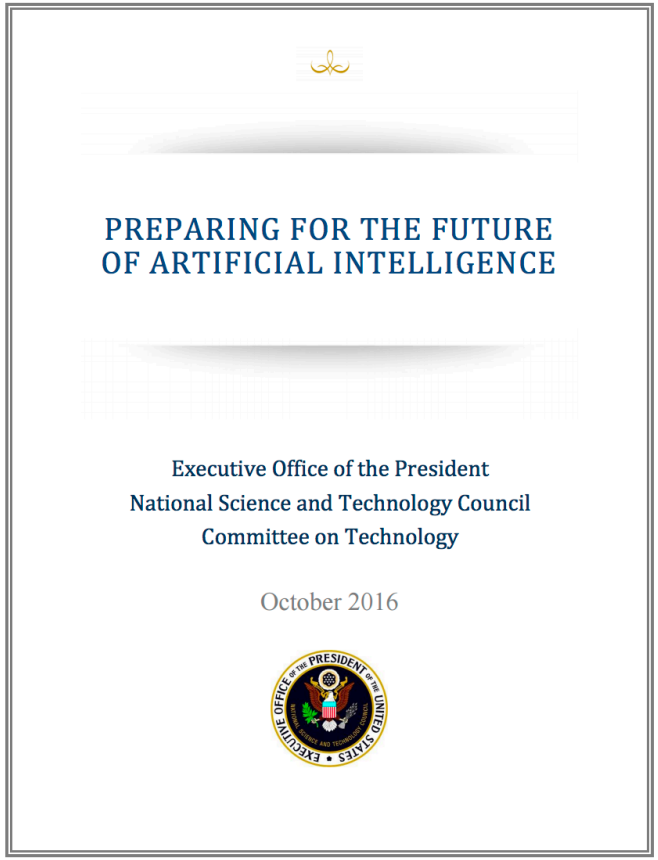
EECS 445: Intro to Machine Learning

11/28/2016, Deep Learning



Deep Learning

Deep Learning

In recent years, some of the most impressive advancements in machine learning have been in the subfield of deep learning, also known as deep network learning. Deep learning uses structures loosely inspired by the human brain, consisting of a set of units (or "neurons"). Each unit combines a set of input values to produce an output value, which in turn is passed on to other neurons downstream. For example, in an image recognition application, a first layer of units might combine the raw data of the image to recognize simple patterns in the image; a second layer of units might combine the results of the first layer to recognize patterns-of-patterns; a third layer might combine the results of the second layer; and so on.

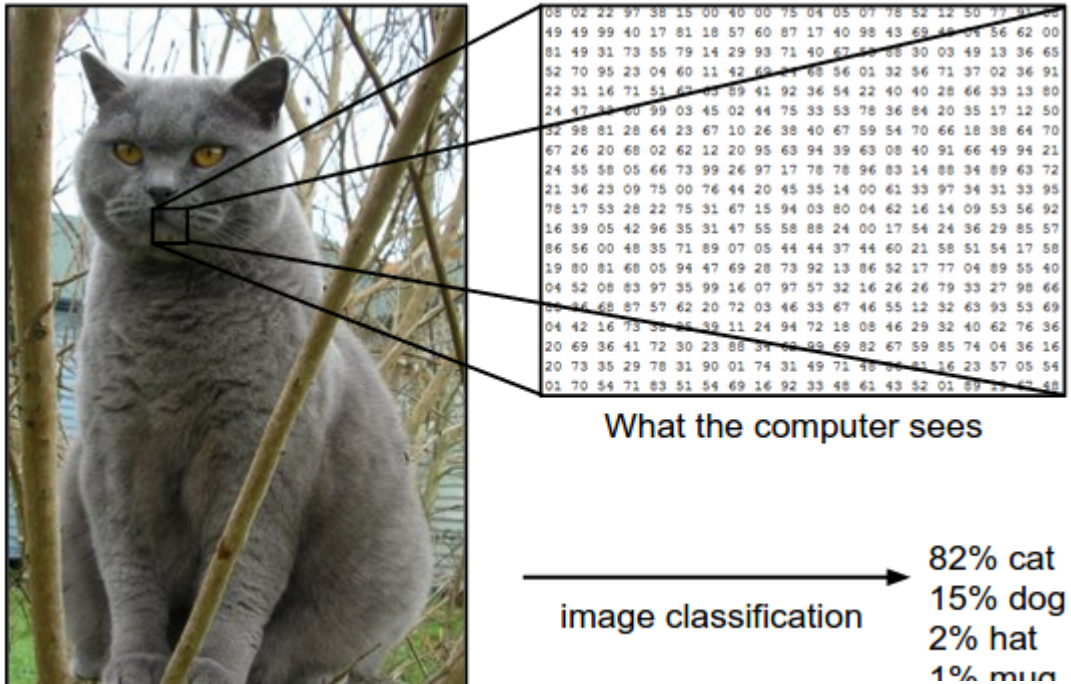
Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

Source: Andrej Karpathy & Fei-Fei Li

The problem:
semantic gap



Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)

Source: Andrej Karpathy & Fei-Fei Li

Linear Classification

1. define a **score function**

(assume CIFAR-10 example so
32 x 32 x 3 images, 10 classes)

data (image)
[3072 x 1]

$$f(x_i, W, b) = Wx_i + b$$

class scores
[10 x 1]

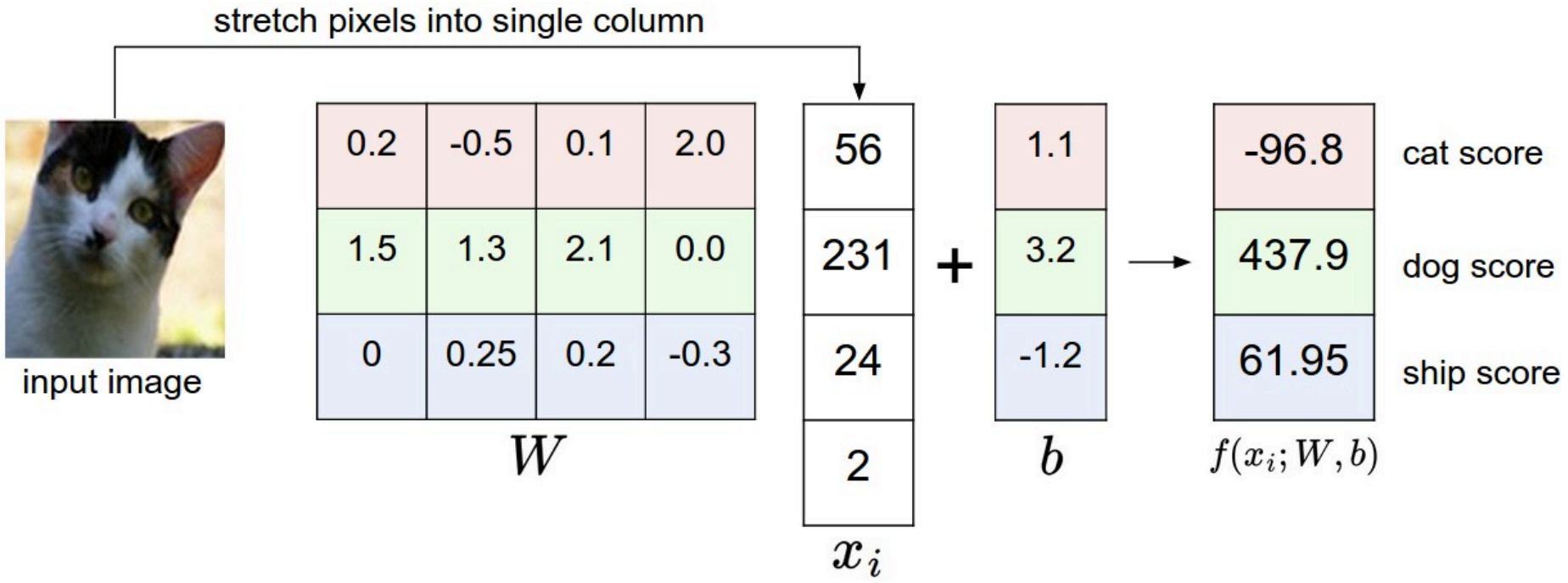
weights
[10 x 3072]

bias vector
[10 x 1]

Source: Andrej Karpathy & Fei-Fei Li

Linear Classification

$$f(x_i, W, b) = Wx_i + b$$

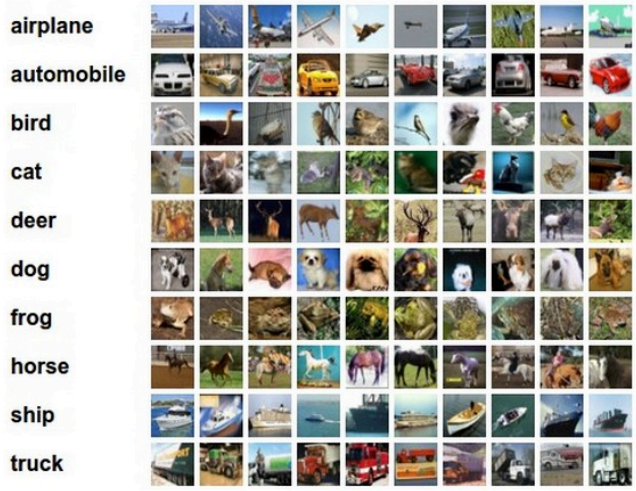


Source: Andrej Karpathy & Fei-Fei Li

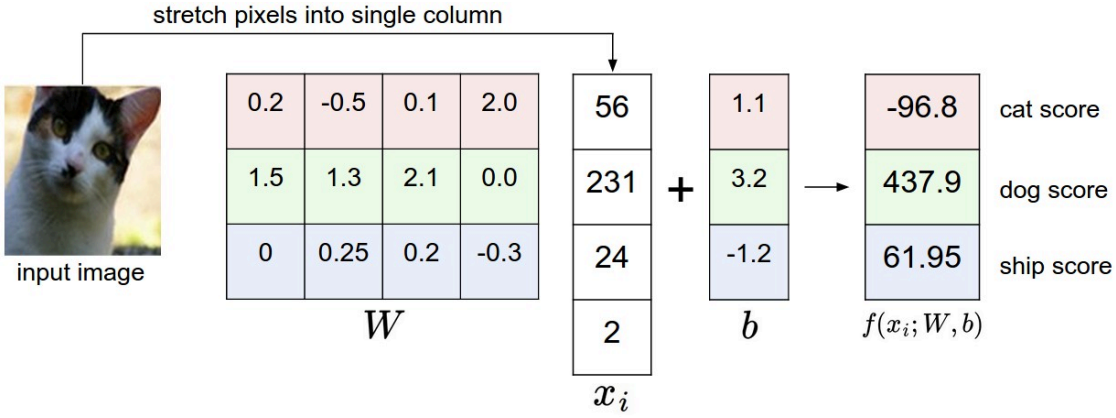
Interpreting a Linear Classifier

Question:

what can a linear classifier do?

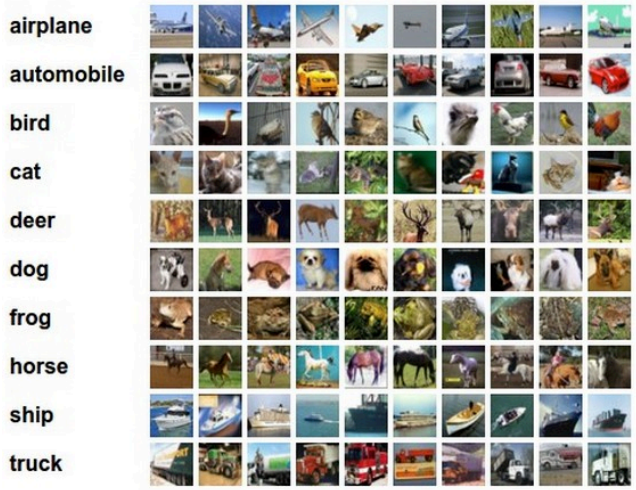


$$f(x_i, W, b) = Wx_i + b$$



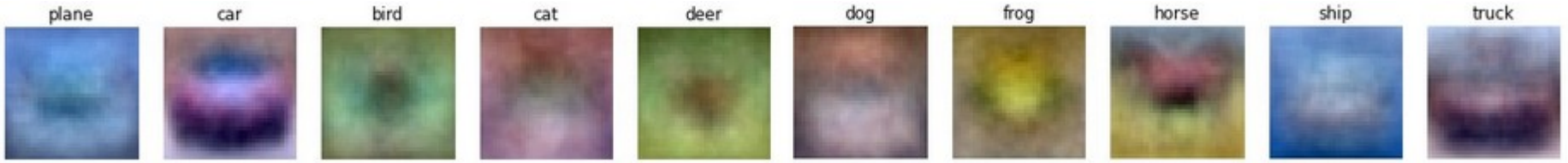
Source: Andrej Karpathy & Fei-Fei Li

Interpreting a Linear Classifier



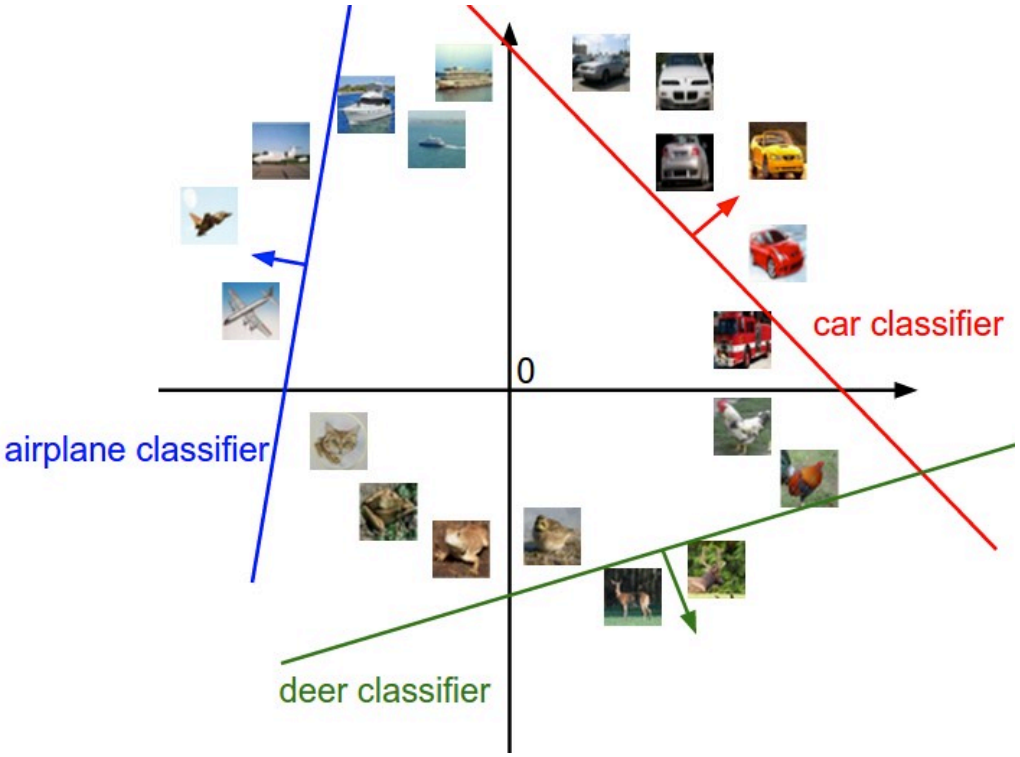
$$f(x_i, W, b) = Wx_i + b$$

Example training classifiers on CIFAR-10:



Source: Andrej Karpathy & Fei-Fei Li

Interpreting a Linear Classifier



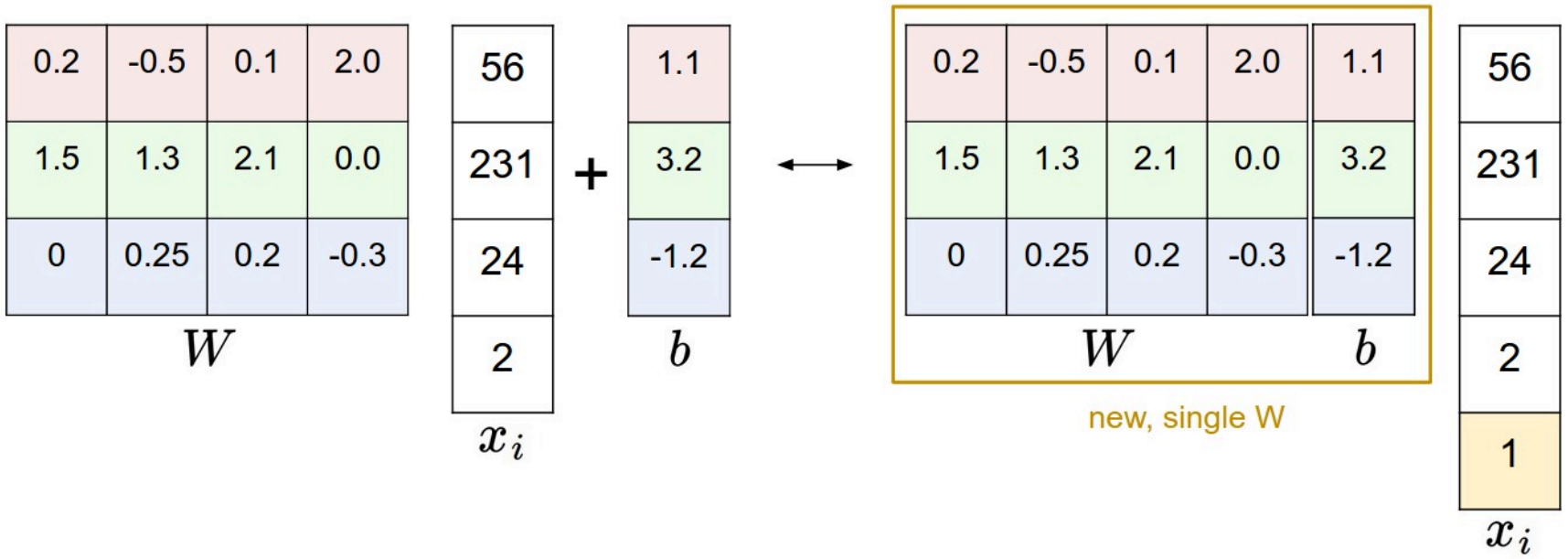
$$f(x_i, W, b) = Wx_i + b$$

Source: Andrej Karpathy & Fei-Fei Li

Bias trick

$$f(x_i, W, b) = Wx_i + b$$

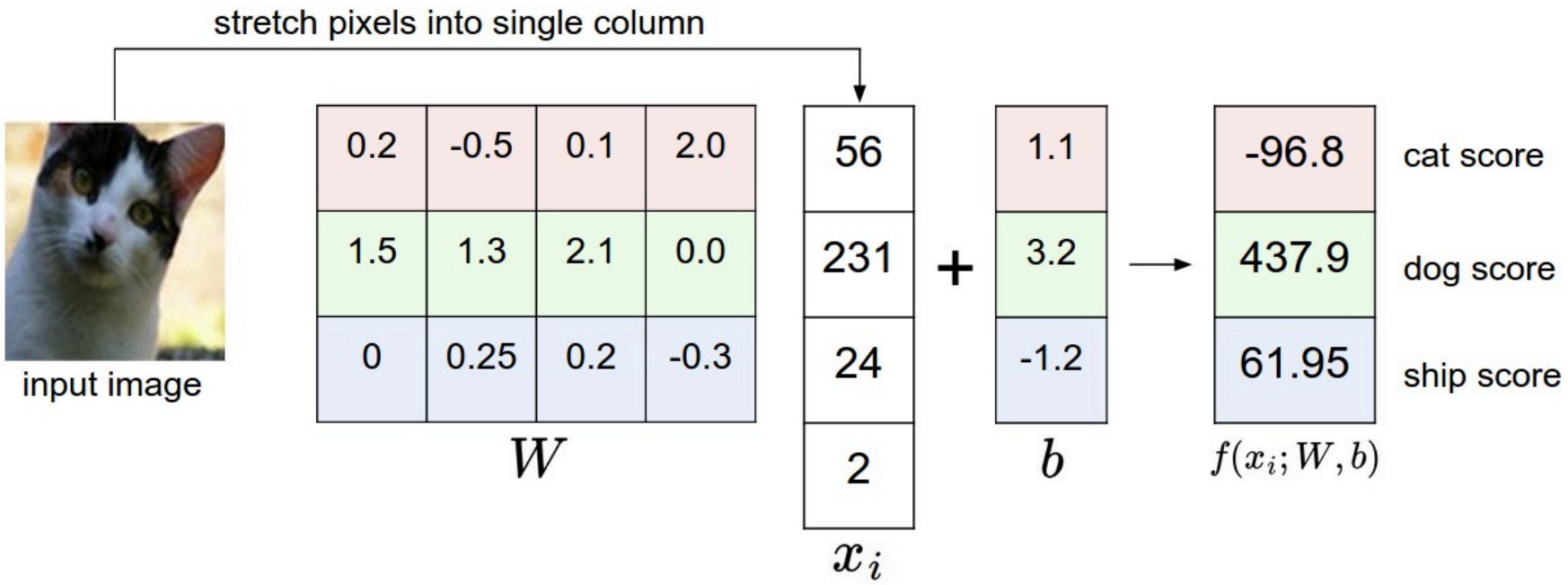
$$f(x_i, W) = Wx_i$$



Source: Andrej Karpathy & Fei-Fei Li

So far:

We defined a (linear) **score function**: $f(x_i, W, b) = Wx_i + b$



Source: Andrej Karpathy & Fei-Fei Li

2. Define a **loss function** (or cost function, or objective)

- scores, label → *loss*.

$f(x_i, W)$ y_i L_i

Example:

$$f(x_i, W) = [13, -7, 11]$$

$$y_i = 0$$

Question: if you were to assign a single number to how "unhappy" you are with these scores, what would you do?

Source: Andrej Karpathy & Fei-Fei Li

2. Define a **loss function** (or cost function, or objective)
One (of many ways) to do it: **Multiclass SVM Loss**

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

(One possible generalization of Binary Support Vector Machine to multiple classes)

$$L_i = C \max(0, 1 - y_i w^T x_i) + R(W)$$

Source: Andrej Karpathy & Fei-Fei Li

2. Define a **loss function** (or cost function, or objective)
One (of many ways) to do it: **Multiclass SVM Loss**

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

loss due to example i

sum over all incorrect labels

difference between the correct class score and incorrect class score

Source: Andrej Karpathy & Fei-Fei Li

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

loss due to example i

sum over all incorrect labels

difference between the correct class score and incorrect class score

Source: Andrej Karpathy & Fei-Fei Li

Example: $$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

e.g. 10

$$f(x_i, W) = [13, -7, 11]$$

$$y_i = 0$$

---

loss = ?

Example: $L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$

e.g. 10

$$f(x_i, W) = [13, -7, 11]$$

$$y_i = 0$$

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

Source: Andrej Karpathy & Fei-Fei Li

There is a bug with the objective…

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right]$$

Source: Andrej Karpathy & Fei-Fei Li

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W)$$

Regularization strength

Source: Andrej Karpathy & Fei-Fei Li

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Source: Andrej Karpathy & Fei-Fei Li

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W)$$

Source: Andrej Karpathy & Fei-Fei Li

So far…

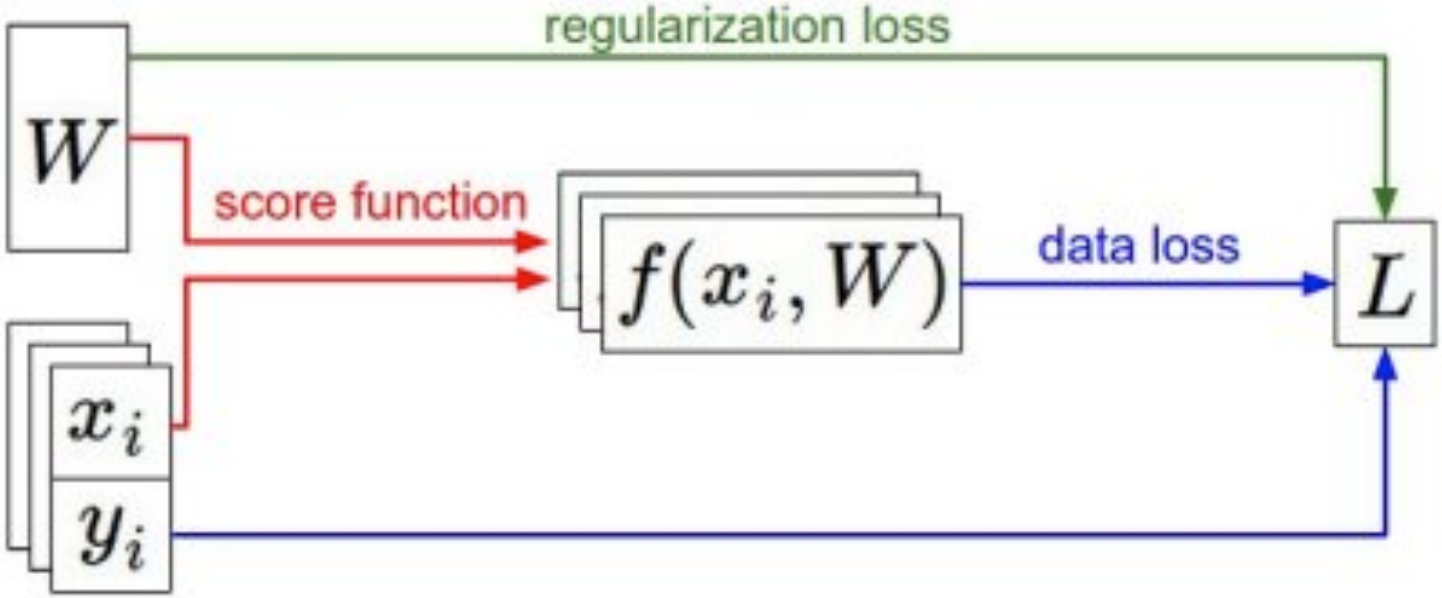
1. Score function

$$f(x_i, W, b) = Wx_i + b$$

2. Loss function

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W)$$

Source: Andrej Karpathy & Fei-Fei Li



Source: Andrej Karpathy & Fei-Fei Li

Softmax Classifier

$$f(x_i, W) = Wx_i$$

score function is the same

(extension of Logistic Regression to multiple classes)

Source: Andrej Karpathy & Fei-Fei Li

Softmax Classifier

$$f(x_i, W) = Wx_i$$

score function is the same

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

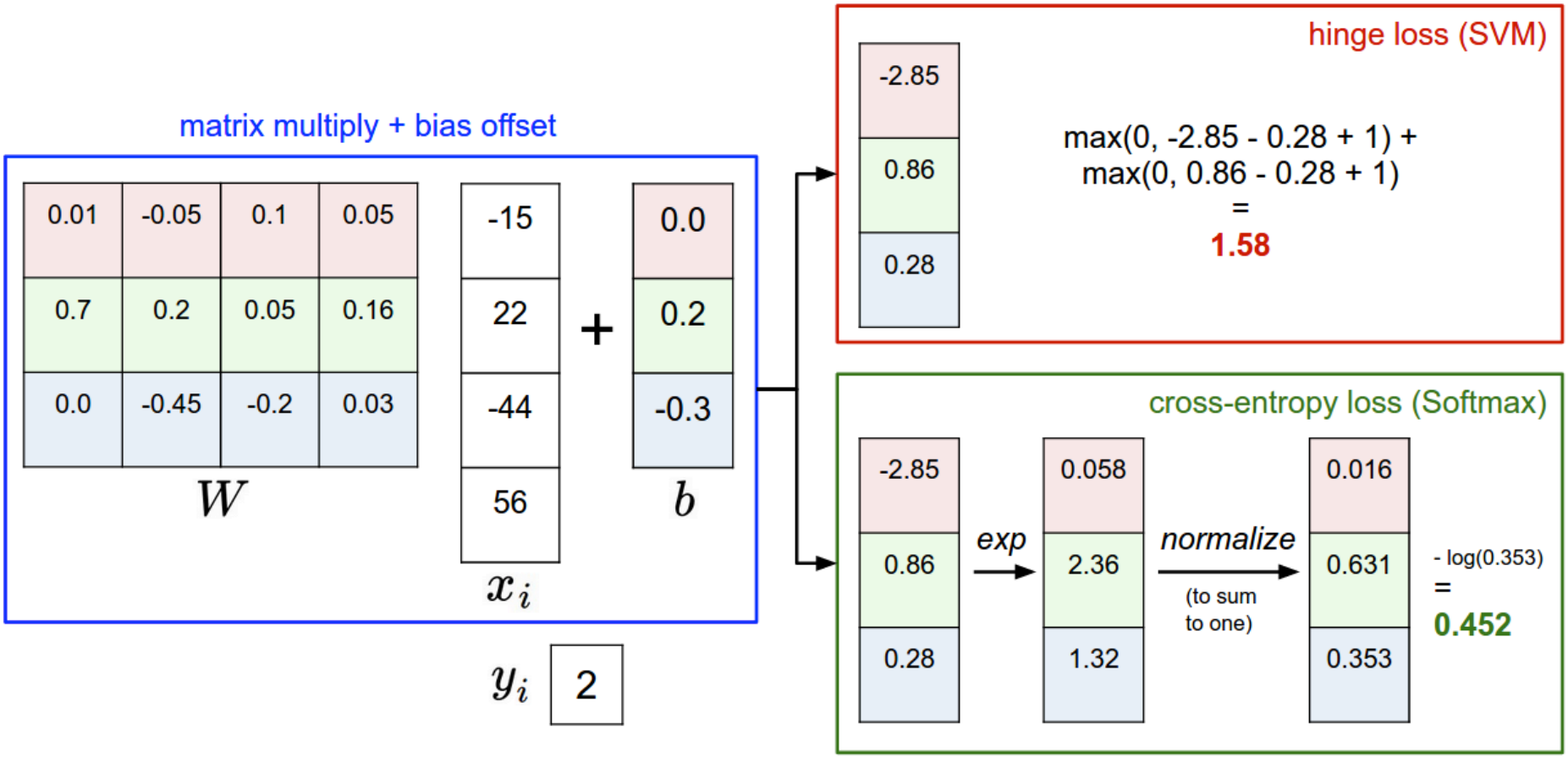
softmax function

$$P(y_i \mid x_i; W)$$

i.e. we're minimizing the negative log likelihood.

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

Source: Andrej Karpathy & Fei-Fei Li



Source: Andrej Karpathy & Fei-Fei Li

Recap

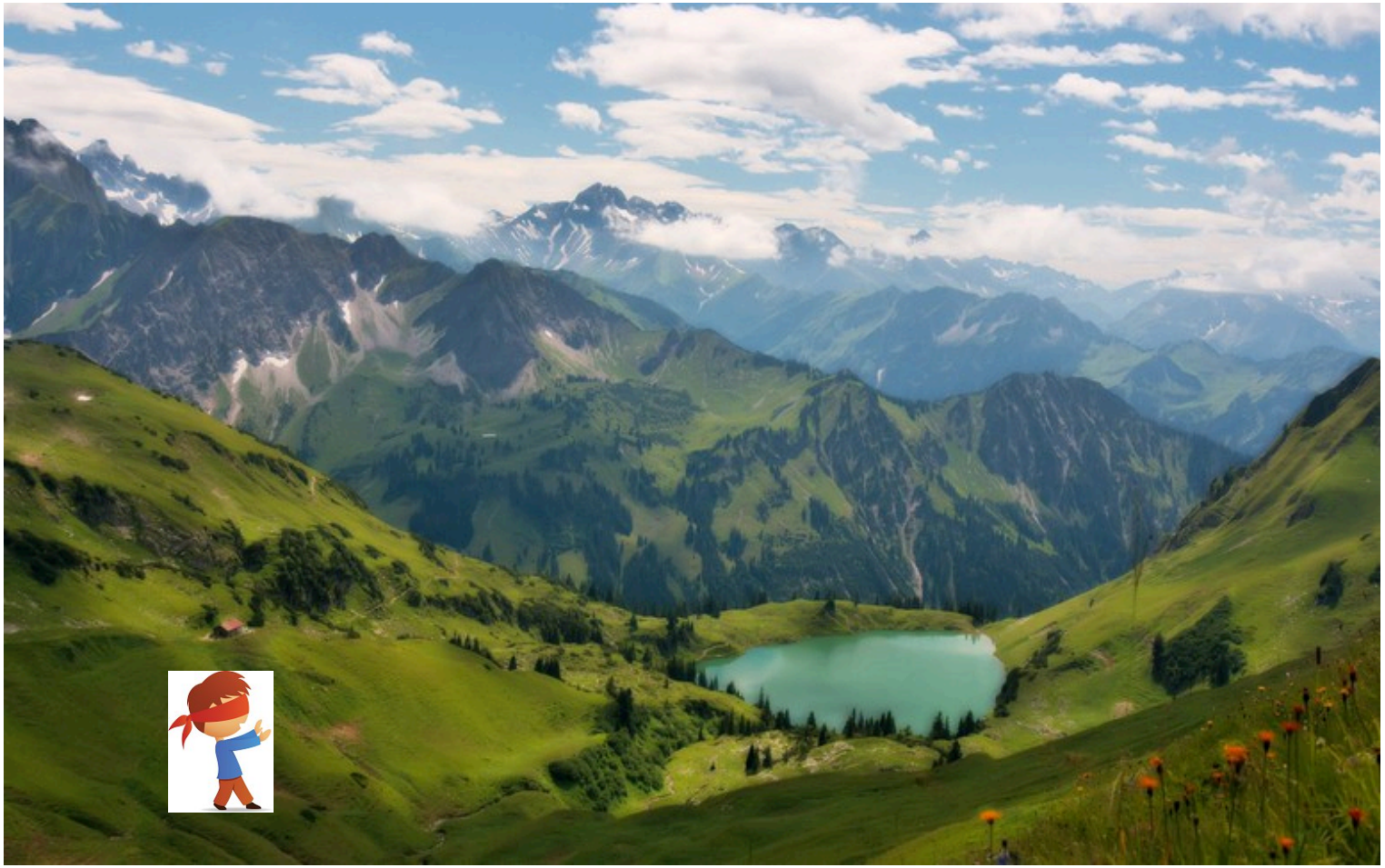
- We introduced a **parametric approach** to image classification
- We defined a **score function** (linear map)
- We defined a **loss function** (SVM / Softmax)

One problem remains: How to find W,b ?

Source: Andrej Karpathy & Fei-Fei Li

Optimization

Source: Andrej Karpathy & Fei-Fei Li



Source: Andrej Karpathy & Fei-Fei Li

Following the gradient

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimension, the **gradient** is the vector of (partial derivatives).

Source: Andrej Karpathy & Fei-Fei Li

Evaluation the gradient numerically

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

"finite difference approximation"

Source: Andrej Karpathy & Fei-Fei Li

Evaluation the gradient numerically

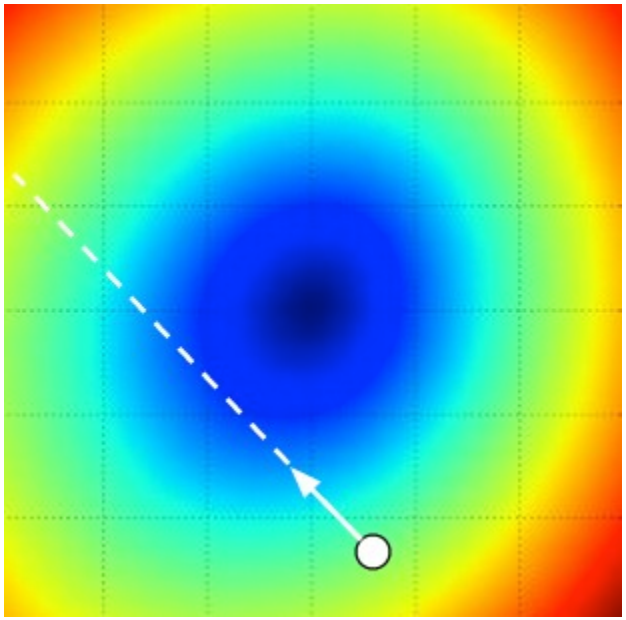
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

in practice:

$$[f(x+h) - f(x-h)]/2h$$

“centered difference formula”

Source: Andrej Karpathy & Fei-Fei Li



original W

negative gradient direction

Source: Andrej Karpathy & Fei-Fei Li

The problems with numerical gradient:

- approximate
- very slow to evaluate

We need something better...

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W)$$

Calculus

Source: Andrej Karpathy & Fei-Fei Li

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \right]$$

$$\begin{aligned} L_i = & \max\,(0, w_1^T x_i - w_0^T x_i + 1) \\ & + \max\,(0, w_2^T x_i - w_0^T x_i + 1) \\ & + \max\,(0, w_3^T x_i - w_0^T x_i + 1) \end{aligned}$$

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

$$\nabla_{w_{y_i}} L_i = - \left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

Source: Andrej Karpathy & Fei-Fei Li

In summary:

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

=>

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

Source: Andrej Karpathy & Fei-Fei Li

Gradient Descent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Source: Andrej Karpathy & Fei-Fei Li

Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are ~100 examples.
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

Source: Andrej Karpathy & Fei-Fei Li

Stochastic Gradient Descent (SGD)

- use a single example at a time

1

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

(also sometimes called **on-line** Gradient Descent)

Source: Andrej Karpathy & Fei-Fei Li

Summary

- Always use mini-batch gradient descent
- Incorrectly refer to it as “doing SGD” as everyone else
  (or call it batch gradient descent)
- The mini-batch size is a hyperparameter, but it is not very common to cross-validate over it (usually based on practical concerns, e.g. space/time efficiency)

Source: Andrej Karpathy & Fei-Fei Li

Fun question: Suppose you were training with mini-batch size of 100, and now you switch to mini-batch of size 1. Your learning rate (step size) should:

- increase
- decrease
- stay the same
- become zero

Source: Andrej Karpathy & Fei-Fei Li

The dynamics of Gradient Descent

pull some weights up and some down

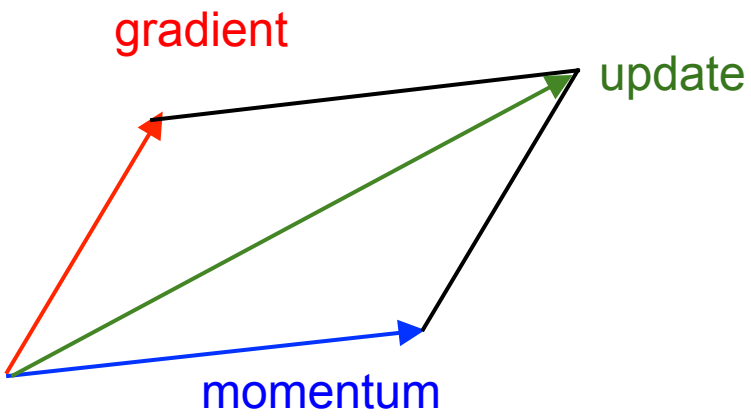
$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda \sum_k \sum_l W_{k,l}^2$$

$$L = \frac{1}{N} \sum_i -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) + \lambda \sum_k \sum_l W_{k,l}^2$$

always pull the weights down

Source: Andrej Karpathy & Fei-Fei Li

Momentum Update




```
weights_grad = evaluate_gradient(loss_fun, data, weights)
vel = vel * 0.9 - step_size * weights_grad
weights += vel
```

Source: Andrej Karpathy & Fei-Fei Li

Many other ways to perform optimization…

- Second order methods that use the Hessian (or its approximation): BFGS, **LBFGS**, etc.

- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for neural networks (to be introduced later).

Source: Andrej Karpathy & Fei-Fei Li

Recap:

- We explored intuition about what the **loss surfaces** of linear classifiers look like
- We introduced **gradient descent** as a way of optimizing loss functions, as well as **batch gradient descent** and **SGD**.
- **Numerical gradient**: slow :(, approximate :(, easy to write :)
- **Analytic gradient**: fast :), exact :), error-prone :(
- In practice: **Gradient check** (but be careful)

Source: Andrej Karpathy & Fei-Fei Li

Backprop

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h \frac{df(x)}{dx}$$

Source: Andrej Karpathy & Fei-Fei Li

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h \frac{df(x)}{dx}$$

Example: x = 4, y = -3. => f(x,y) = -12

$$\frac{\partial f}{\partial x} = -3$$

$$\frac{\partial f}{\partial y} = 4$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

partial derivatives

gradient

Question: If I increase x by h, how would the output of f change?

Source: Andrej Karpathy & Fei-Fei Li

Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Source: Andrej Karpathy & Fei-Fei Li

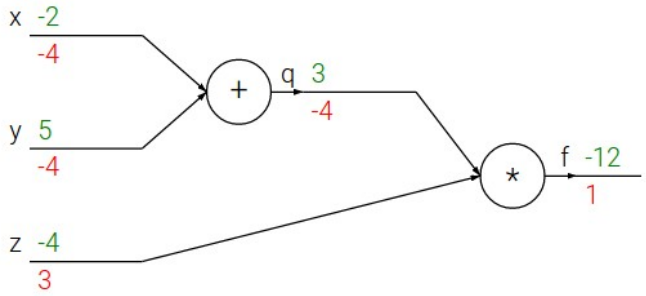
Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

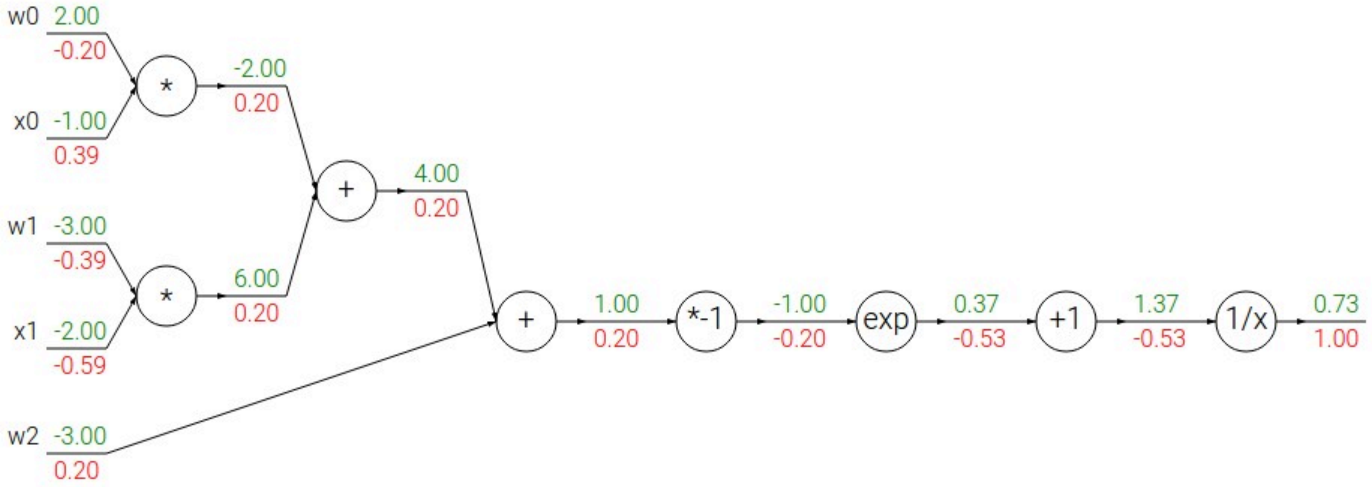
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

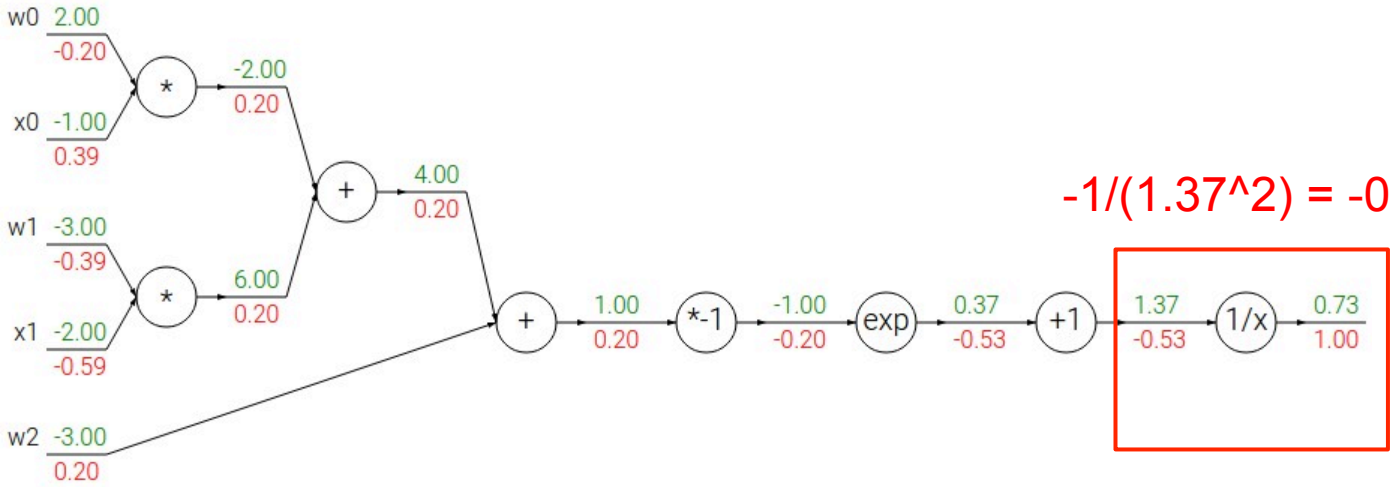
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



-1/(1.37^2) = -0.53

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

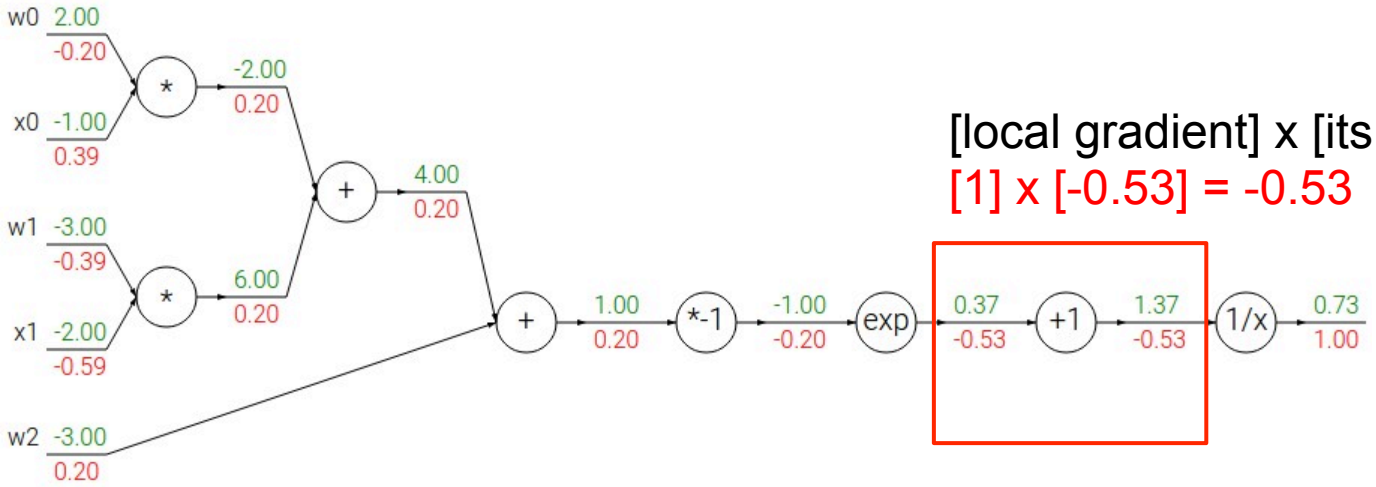
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
[1] x [-0.53] = -0.53

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

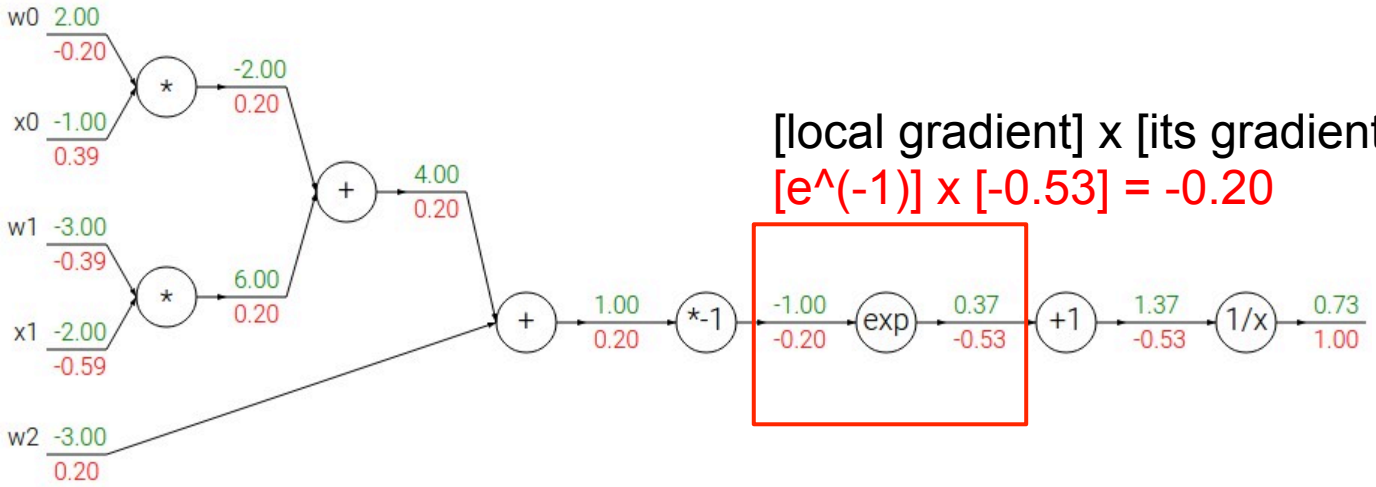
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
[e^(-1)] x [-0.53] = -0.20

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

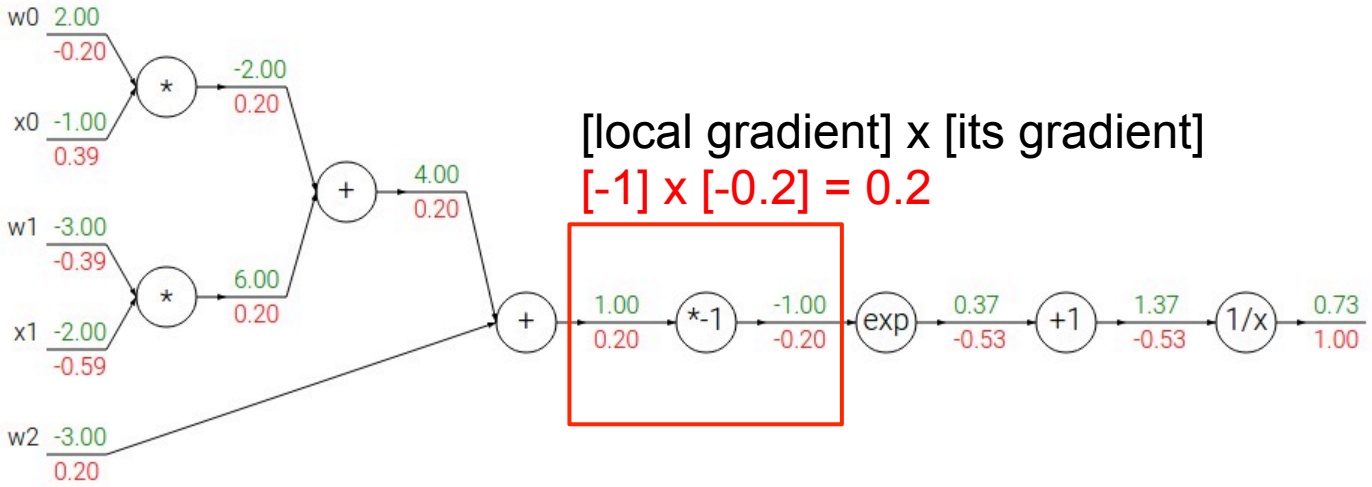
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
[-1] x [-0.2] = 0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

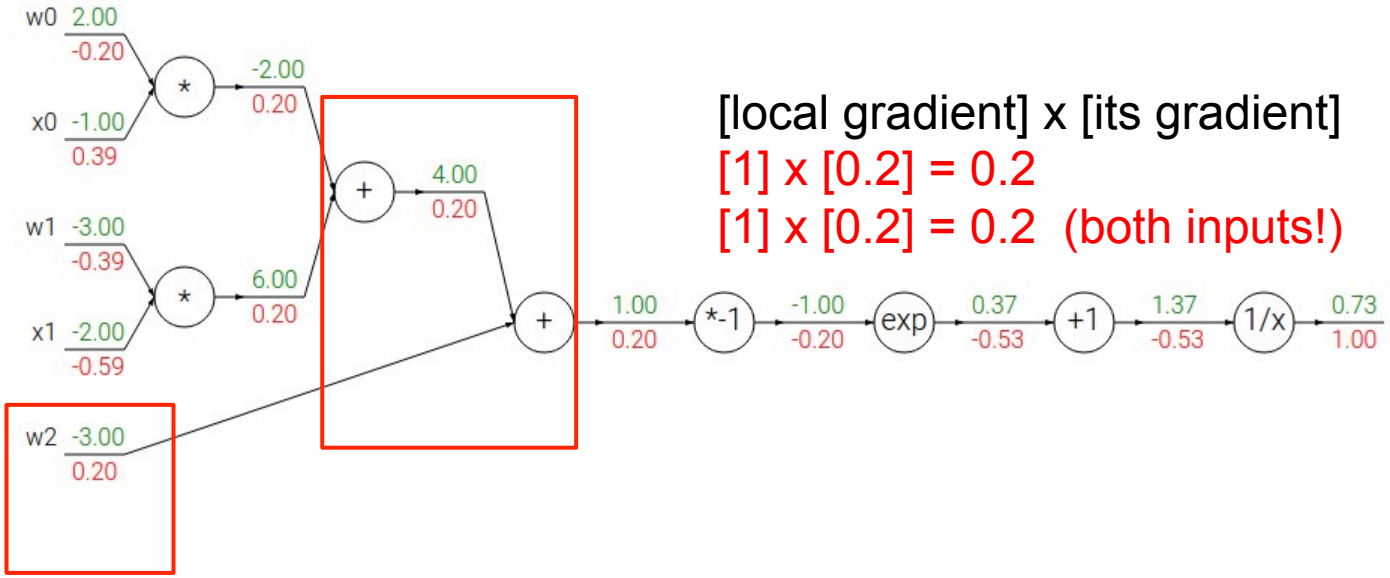
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
[1] x [0.2] = 0.2
[1] x [0.2] = 0.2 (both inputs!)

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

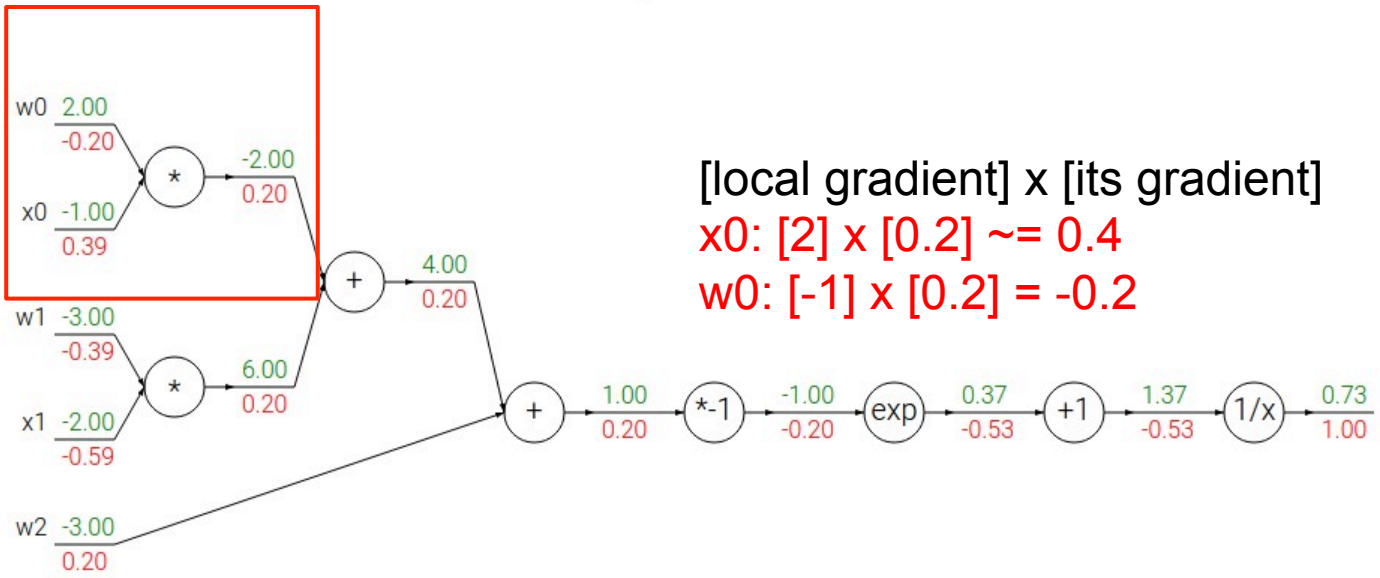
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
x0: [2] x [0.2] ~= 0.4
w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Source: Andrej Karpathy & Fei-Fei Li

Every gate during backprop computes, for all its inputs:

[LOCAL GRADIENT] x [GATE GRADIENT]

Can be computed right away, even during forward pass

The gate receives this during backpropagation

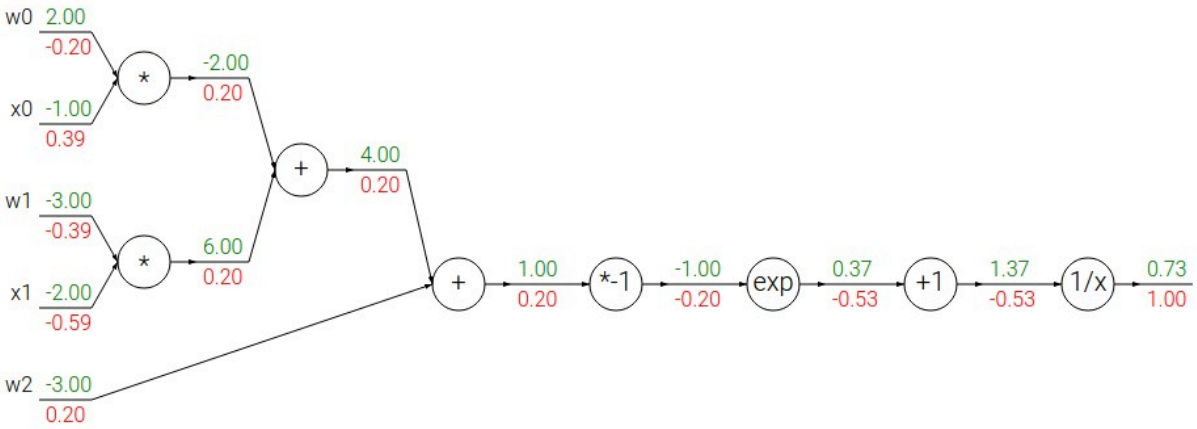
Source: Andrej Karpathy & Fei-Fei Li

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



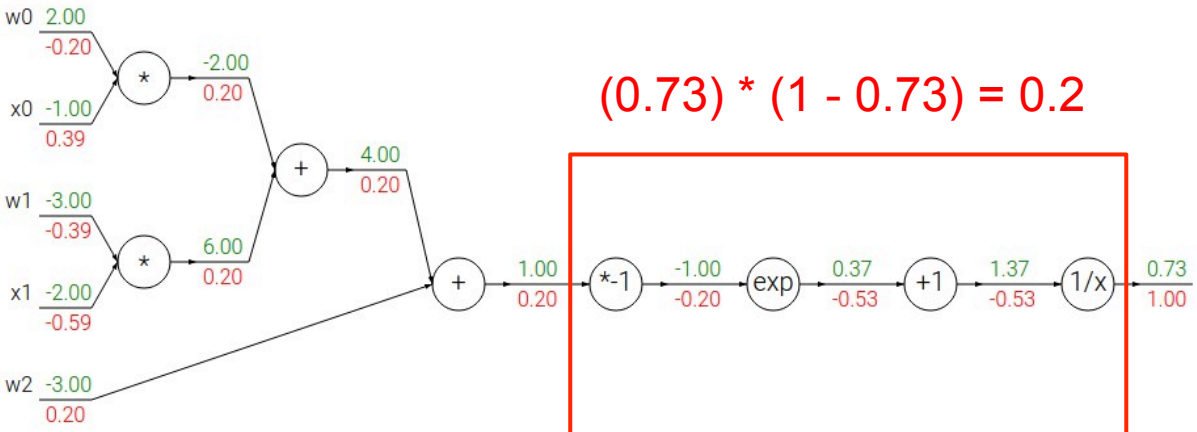
Source: Andrej Karpathy & Fei-Fei Li

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Source: Andrej Karpathy & Fei-Fei Li

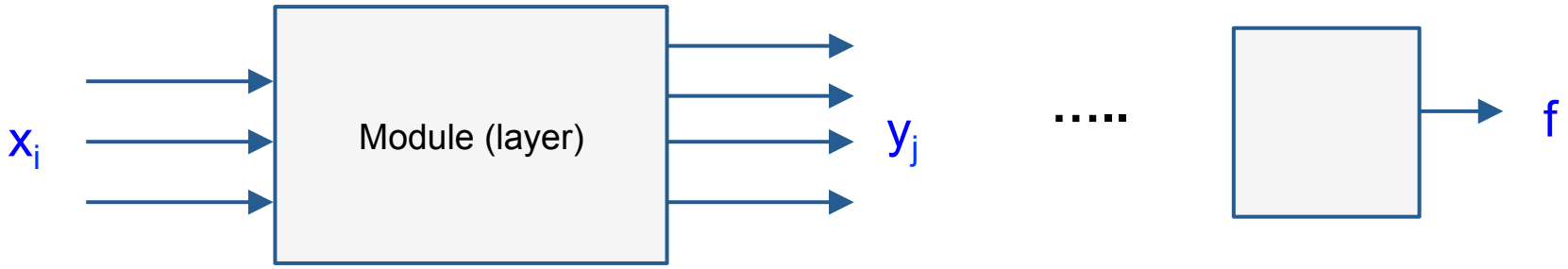
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Source: Andrej Karpathy & Fei-Fei Li



Multiple inputs x_i
Multiple outputs y_j

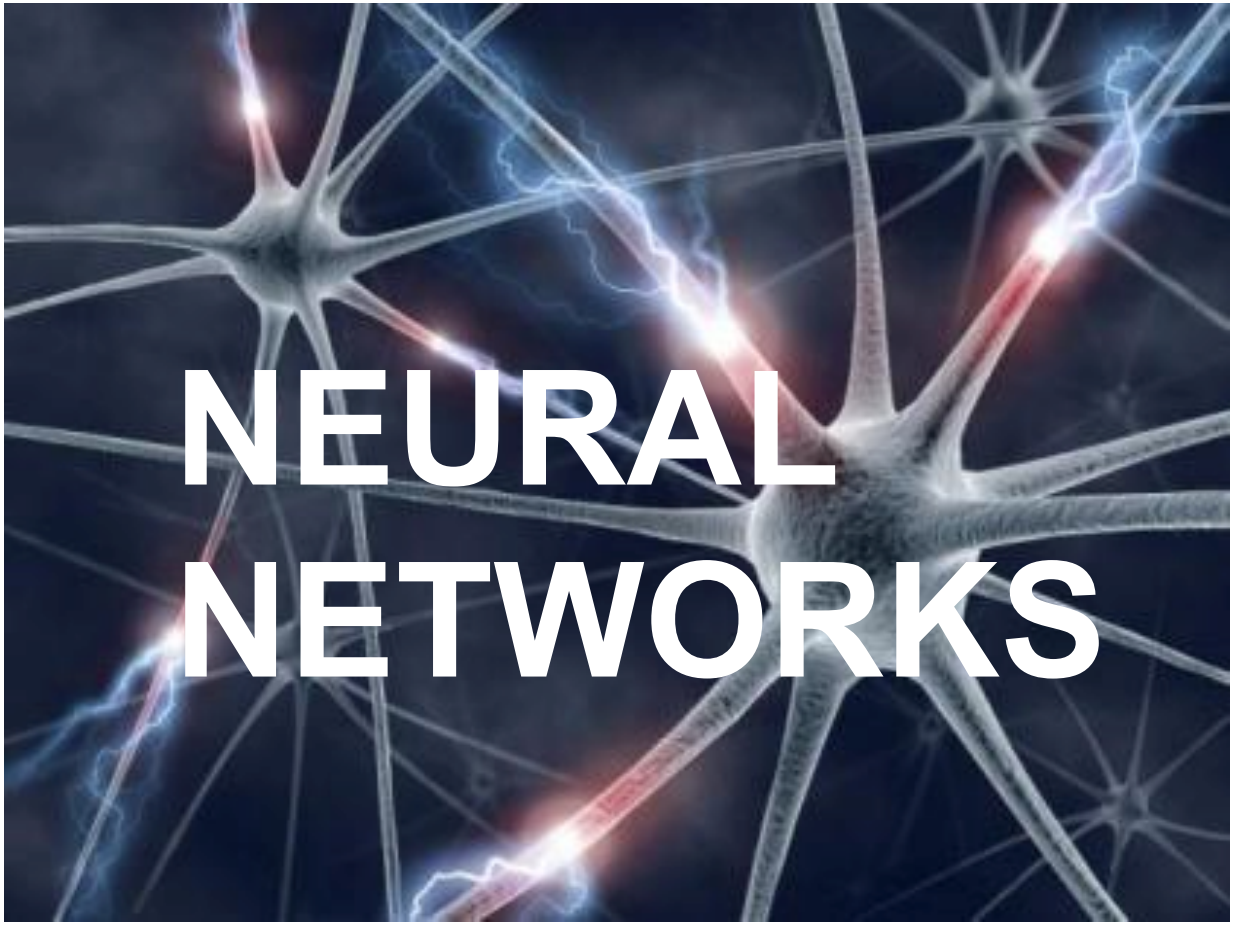
$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Source: Andrej Karpathy & Fei-Fei Li

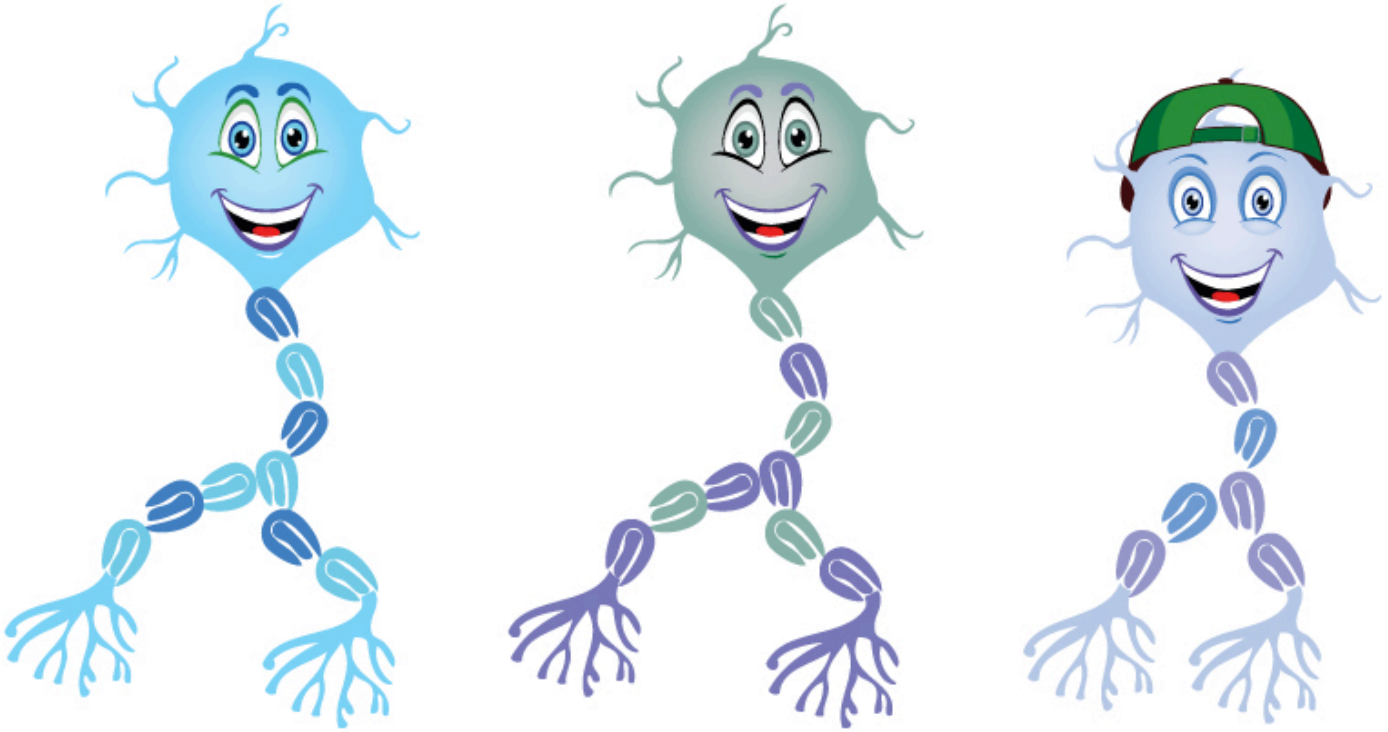
In summary

- in practice it is rarely needed to derive long gradients of variables on pen and paper
- structured your code in **stages** (layers), where you can derive the local gradients, then chain the gradients during **backprop**.
- caveat: sometimes gradients simplify (e.g. for sigmoid, also softmax). Group these.

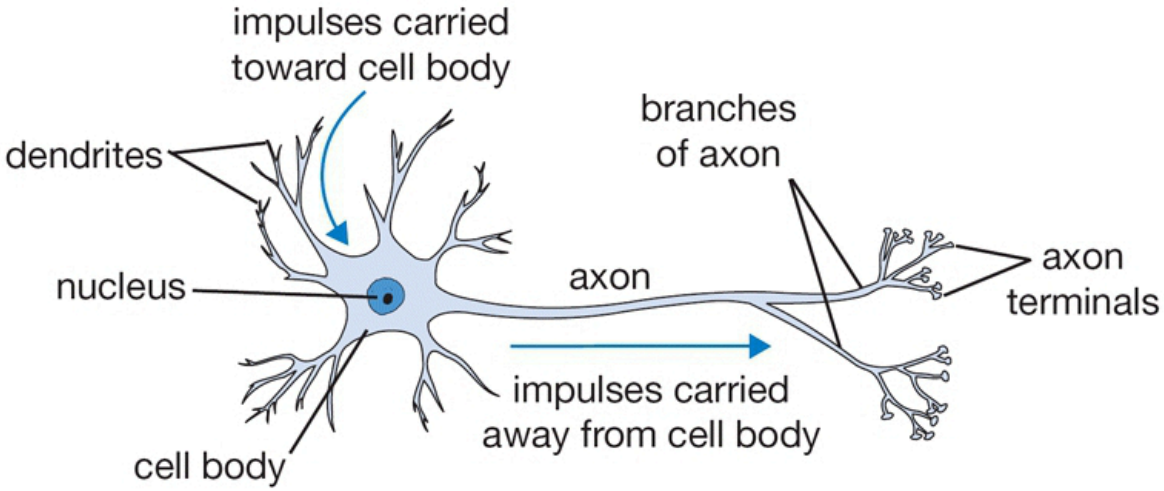
Source: Andrej Karpathy & Fei-Fei Li



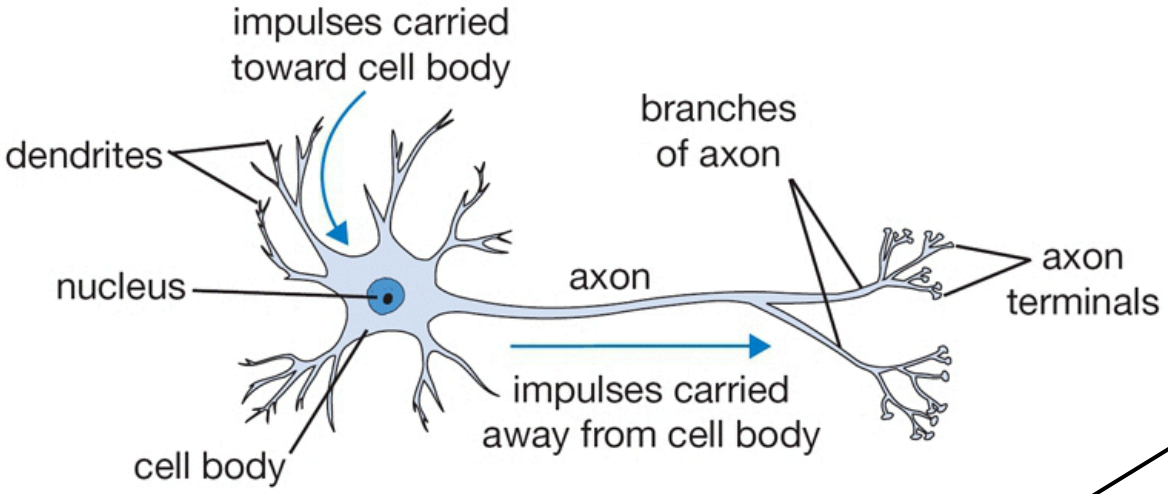
Source: Andrej Karpathy & Fei-Fei Li

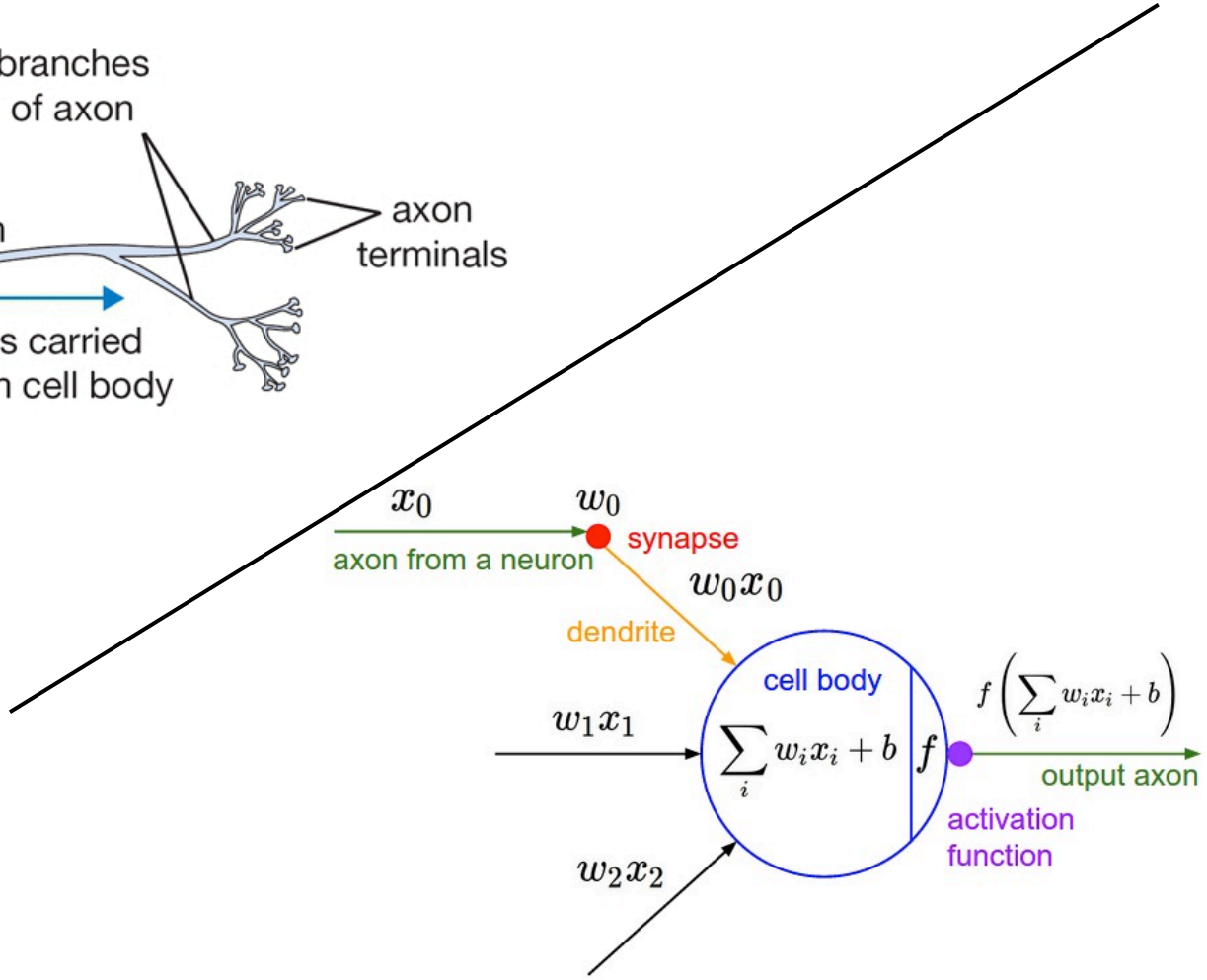


Source: Andrej Karpathy & Fei-Fei Li

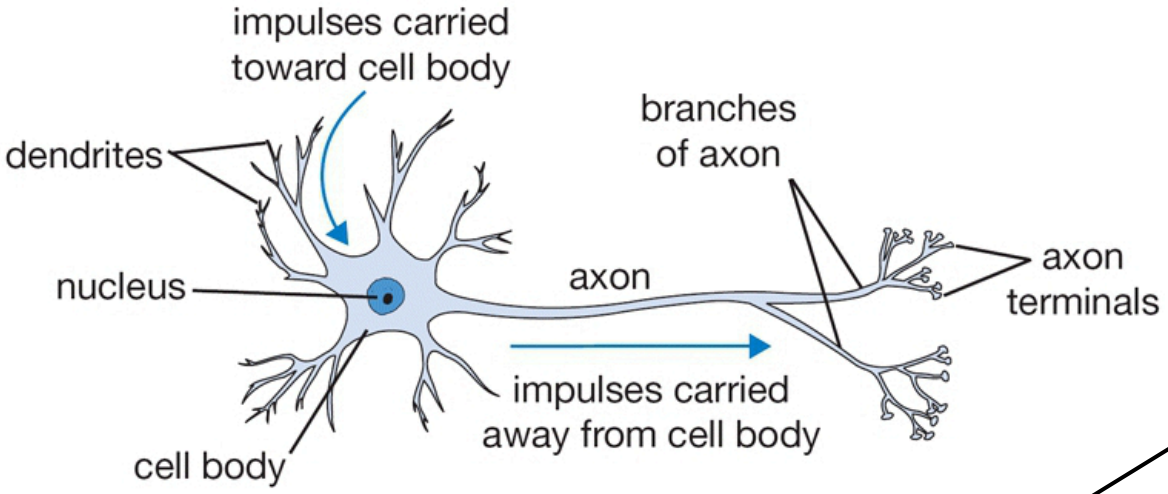


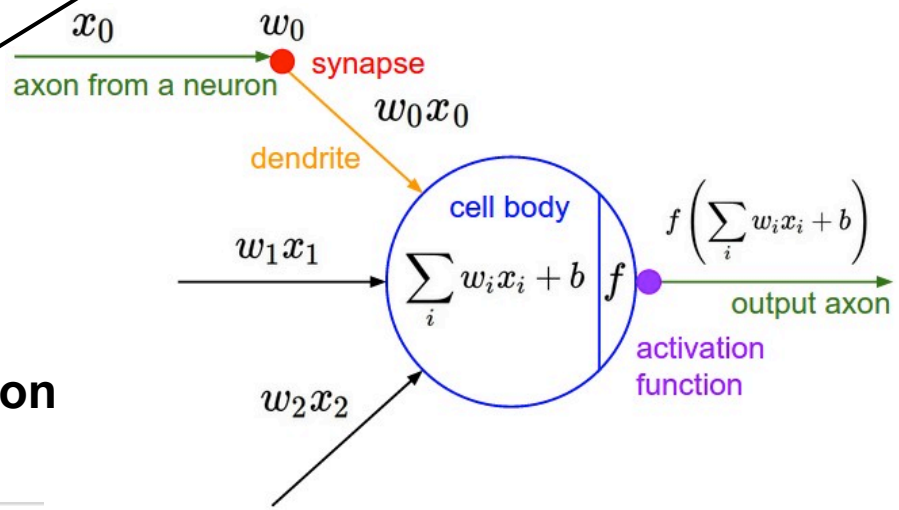
Source: Andrej Karpathy & Fei-Fei Li

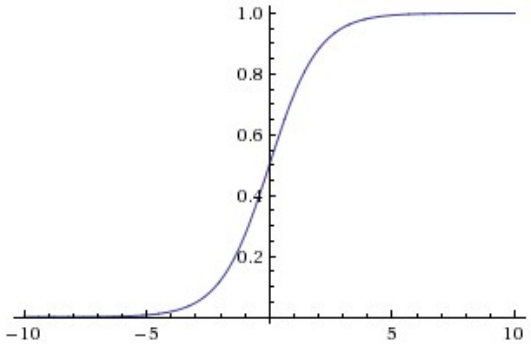




Source: Andrej Karpathy & Fei-Fei Li





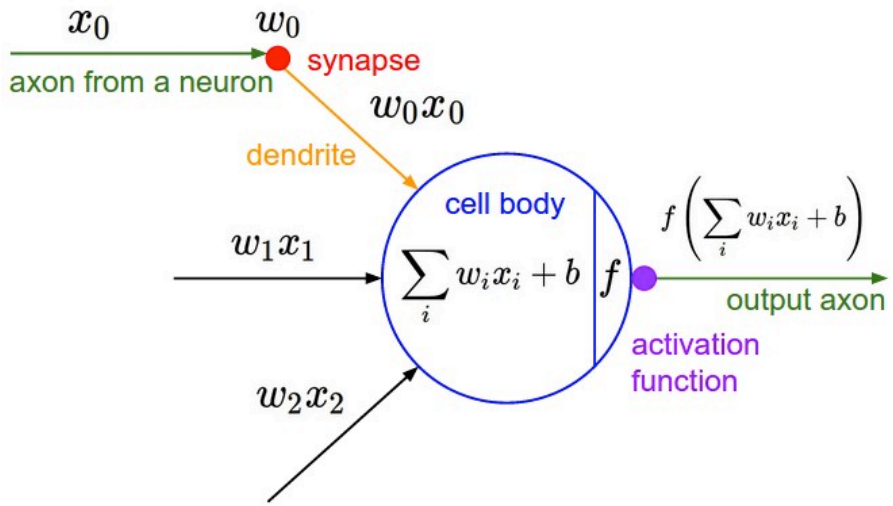


sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

Source: Andrej Karpathy & Fei-Fei Li

A Single Neuron can be used as a binary linear classifier

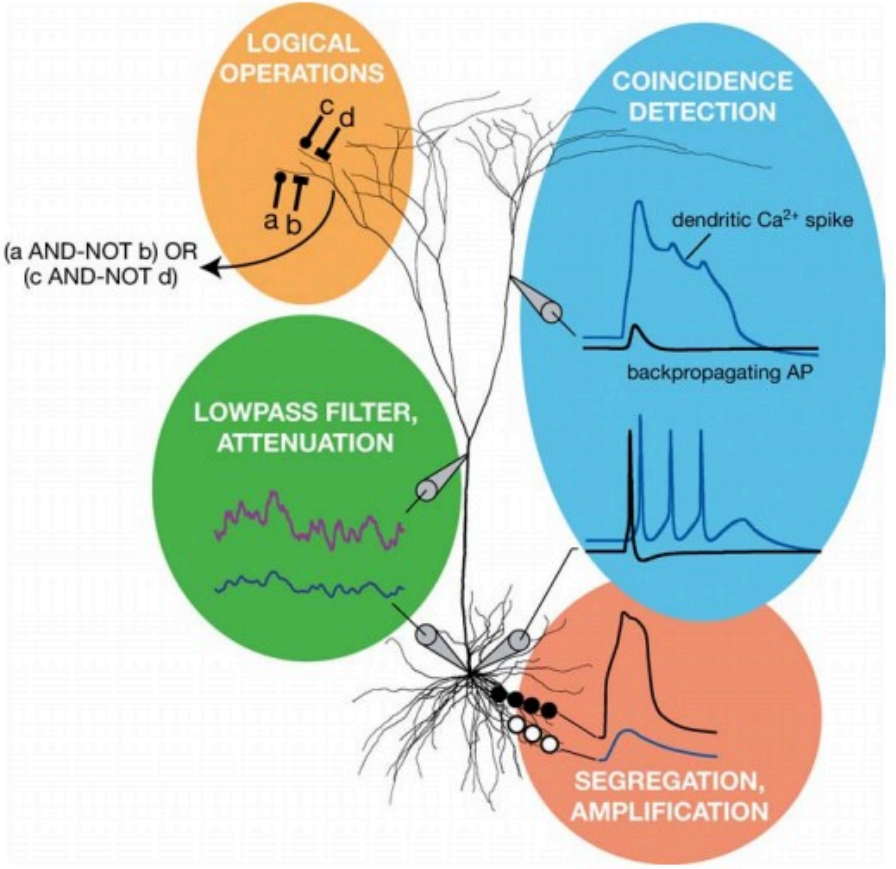


Source: Andrej Karpathy & Fei-Fei Li

Be very careful with your Brain analogies:

Biological Neurons:

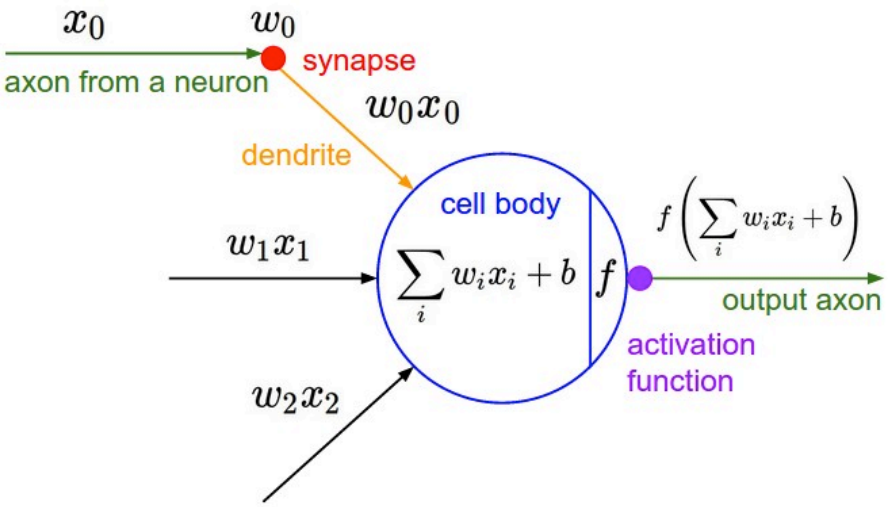
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Rate code may not be adequate



[Dendritic Computation. London and Hausser]

Source: Andrej Karpathy & Fei-Fei Li

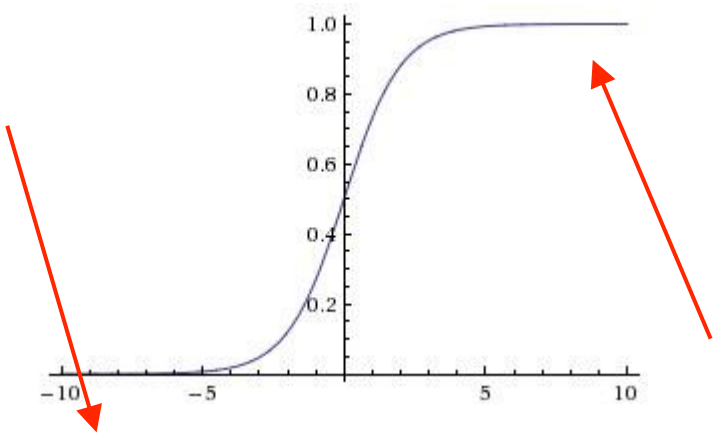
Activation Functions



Source: Andrej Karpathy & Fei-Fei Li

Activation Functions

$$\sigma(x) = 1/(1 + e^{-x})$$



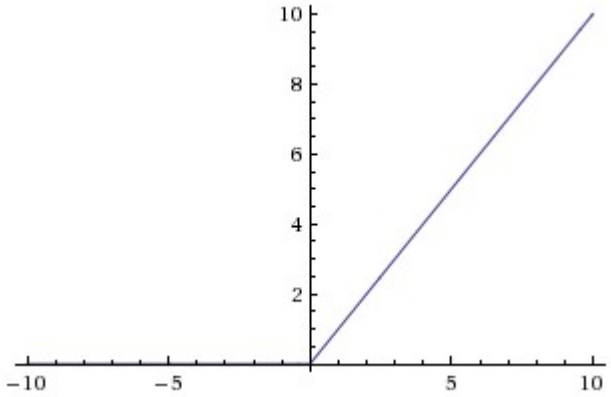
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

Sigmoid

A problem: Saturated neurons “kill” the gradients

Source: Andrej Karpathy & Fei-Fei Li

Activation Functions



ReLU

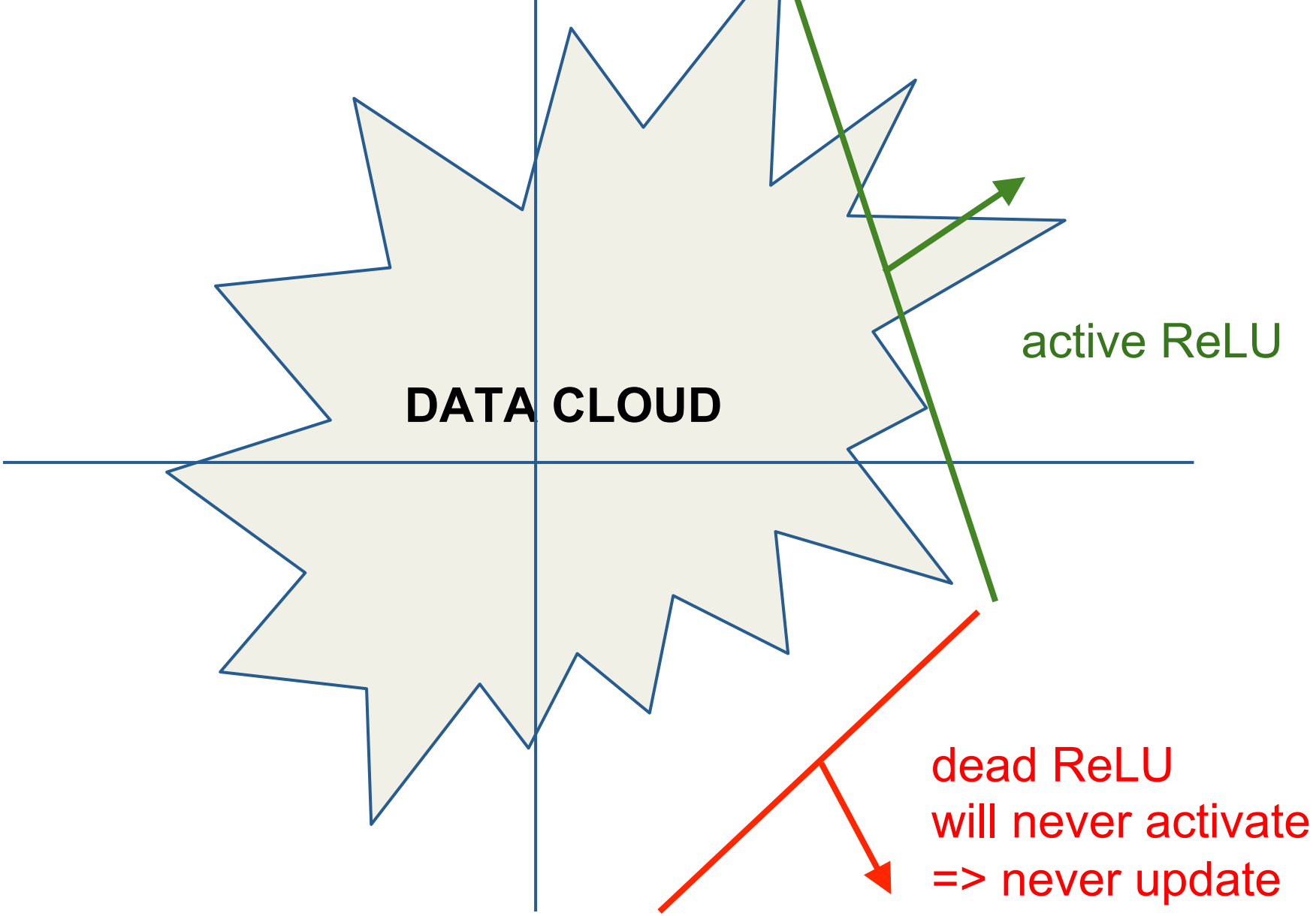
- Computes f(x) = max(0,x)

- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid in practice! (e.g. 6x)

- Just one annoying problem…

hint: what is the gradient when x < 0?

Source: Andrej Karpathy & Fei-Fei Li

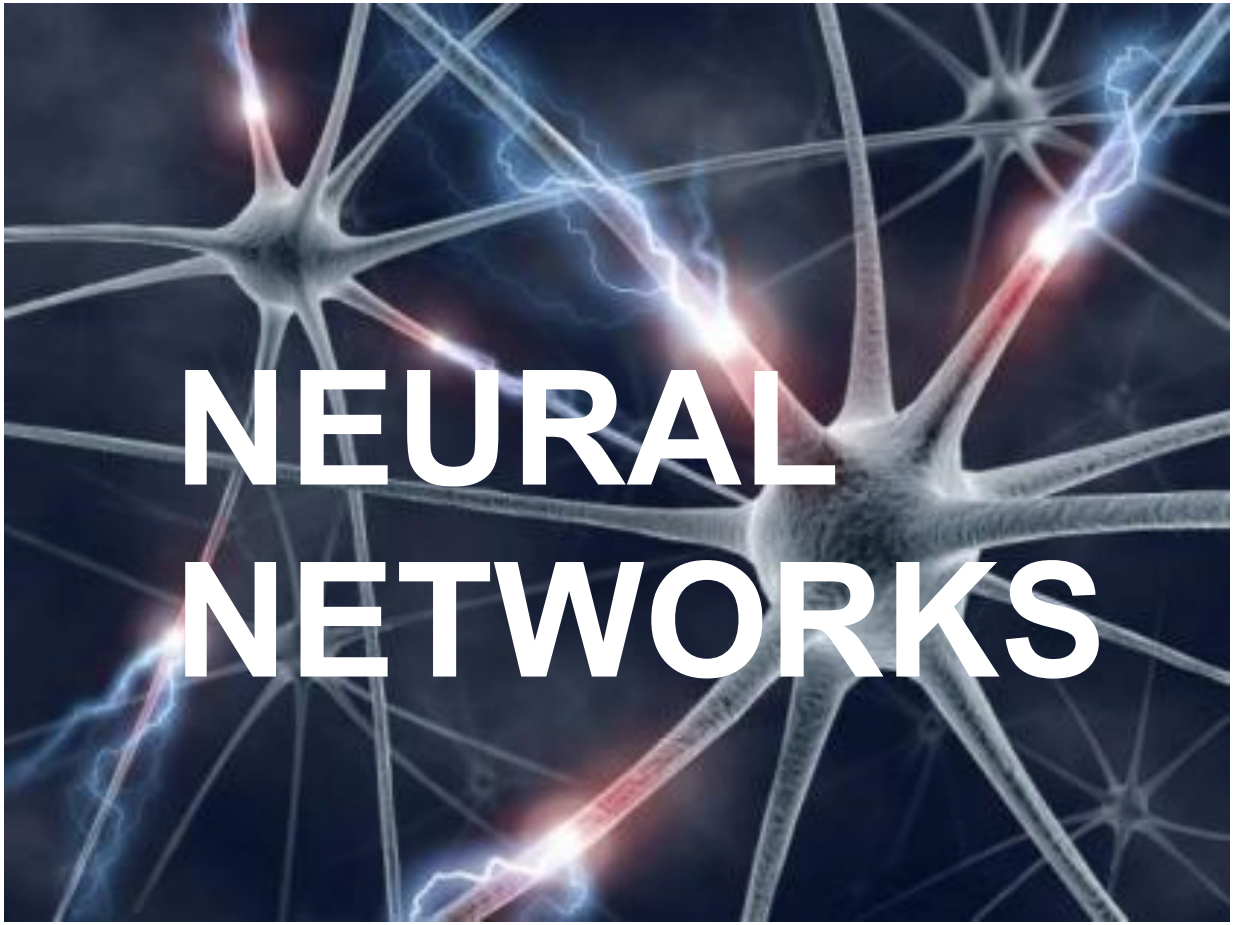


Source: Andrej Karpathy & Fei-Fei Li

TLDR: In practice:

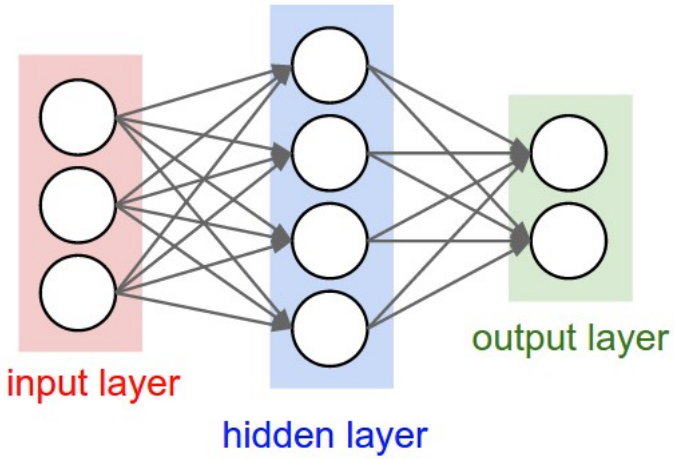
- Use ReLU. Be careful with your learning rates
- Never use sigmoid

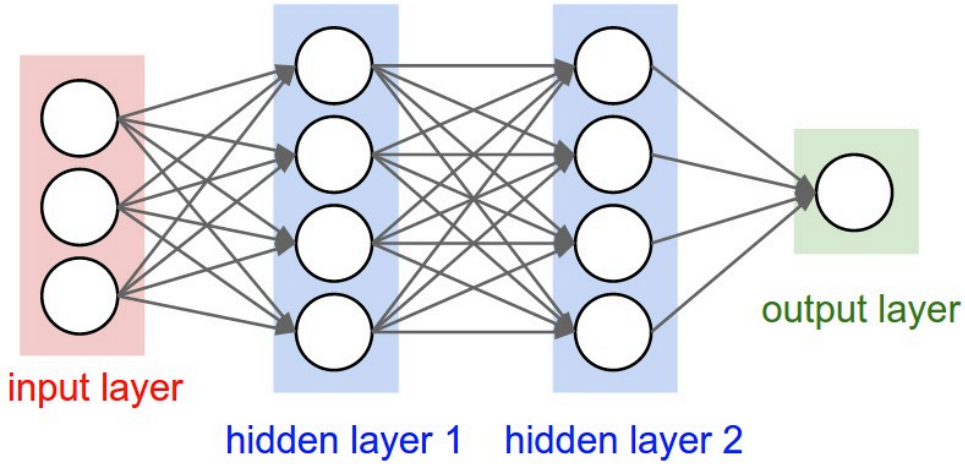
Source: Andrej Karpathy & Fei-Fei Li



Source: Andrej Karpathy & Fei-Fei Li

Neural Networks: Architectures





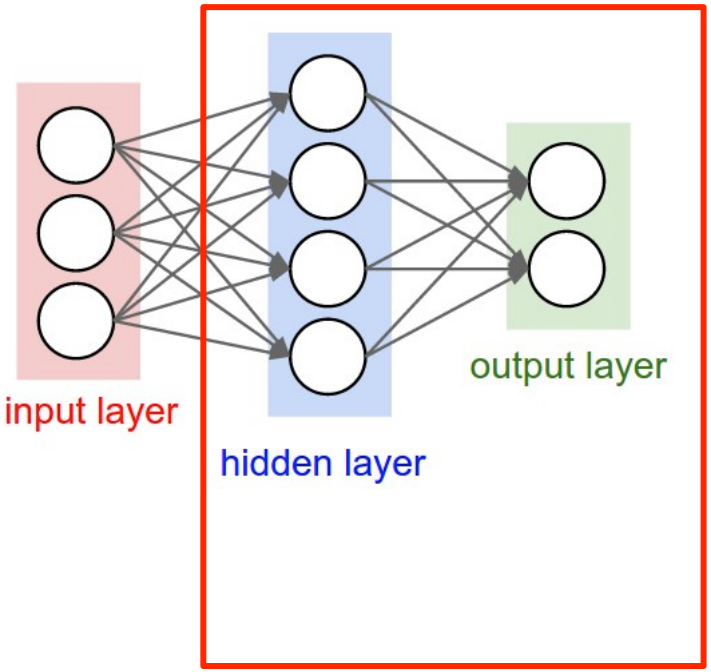
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

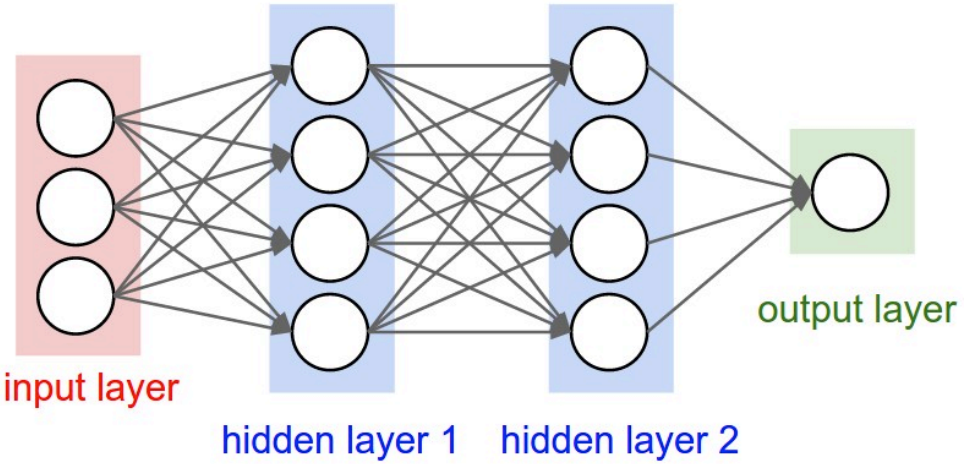
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Source: Andrej Karpathy & Fei-Fei Li

Neural Networks: Architectures

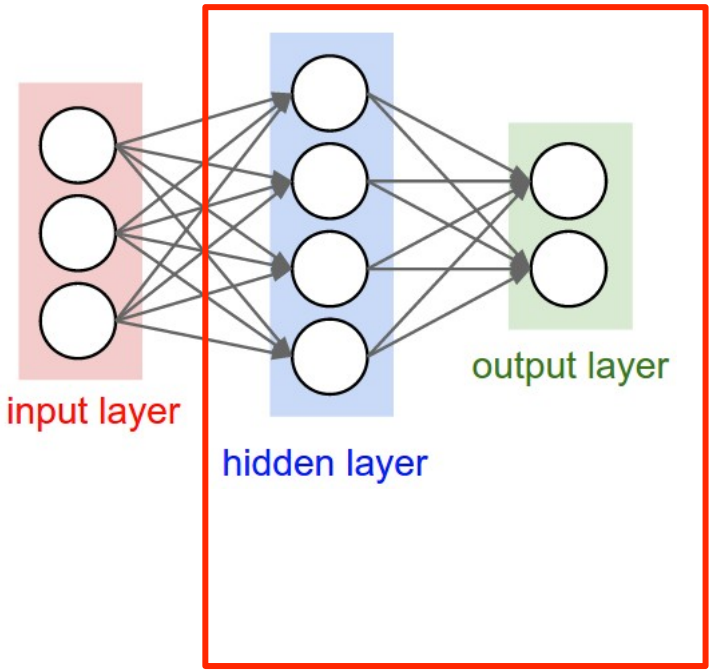


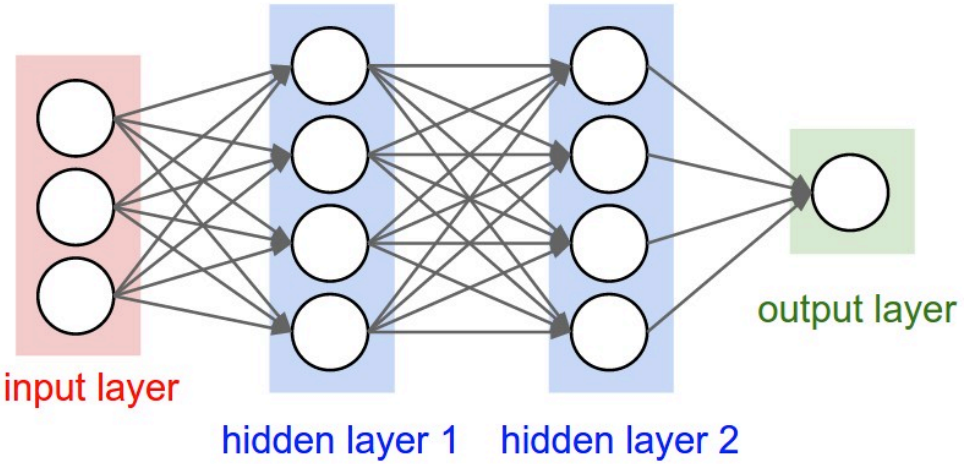


Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Source: Andrej Karpathy & Fei-Fei Li

Neural Networks: Architectures



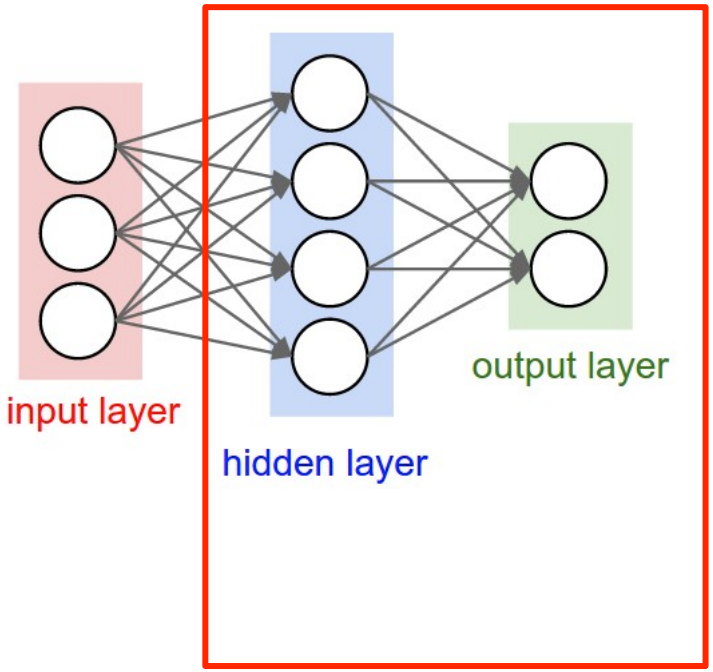


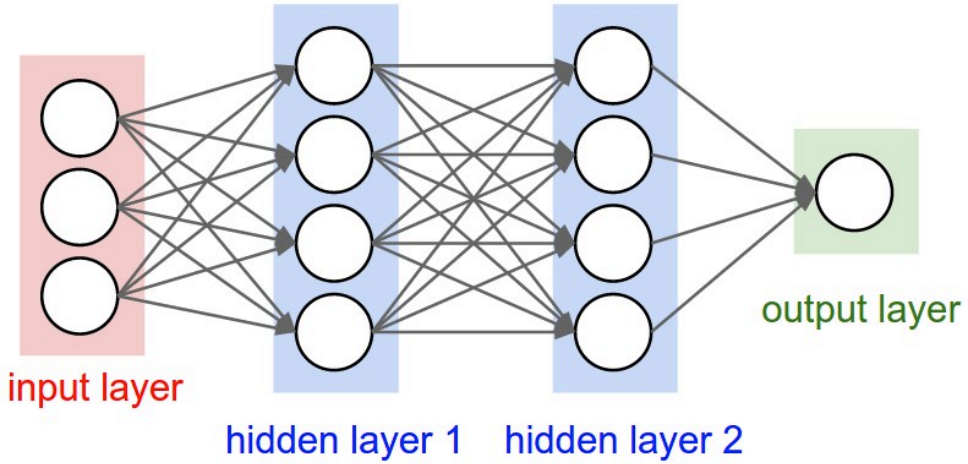
Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Source: Andrej Karpathy & Fei-Fei Li

Neural Networks: Architectures



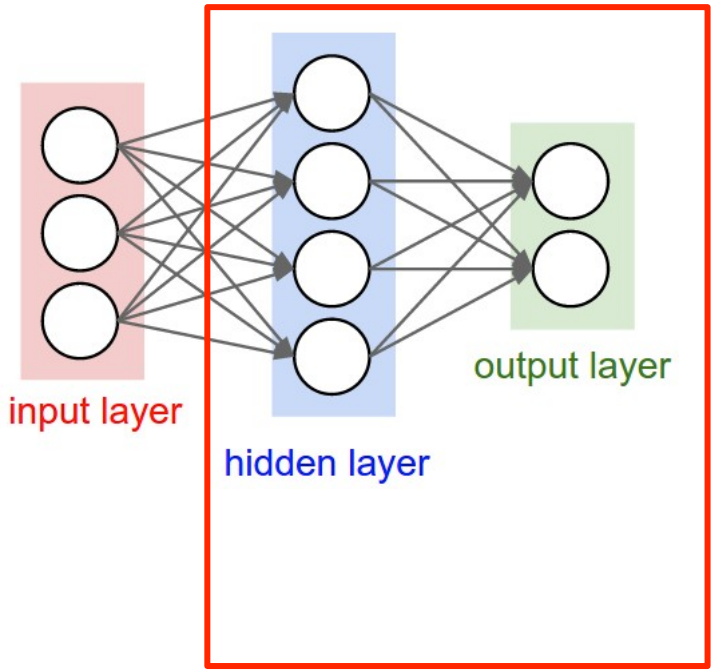


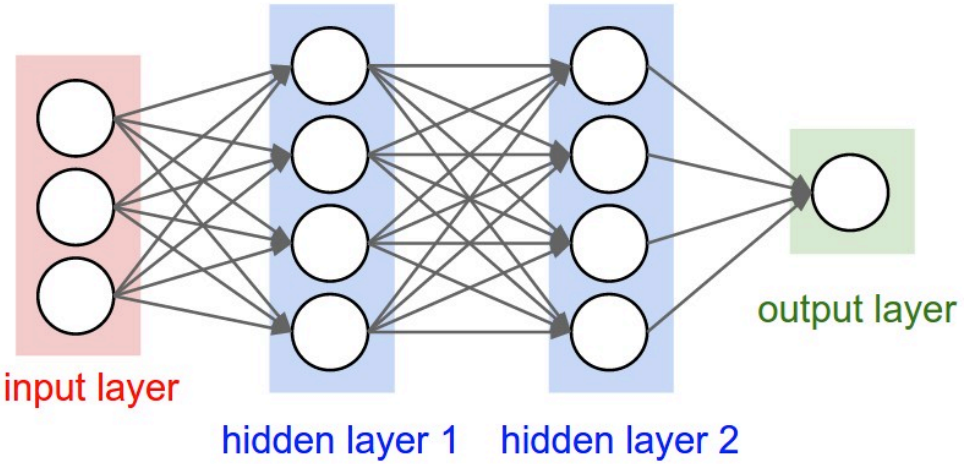
Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: 4 + 4 + 1 = 9
Number of Weights: [4x3+4x4+1x4]=32
Number of Parameters: 32+9 = 41

Source: Andrej Karpathy & Fei-Fei Li

Neural Networks: Architectures

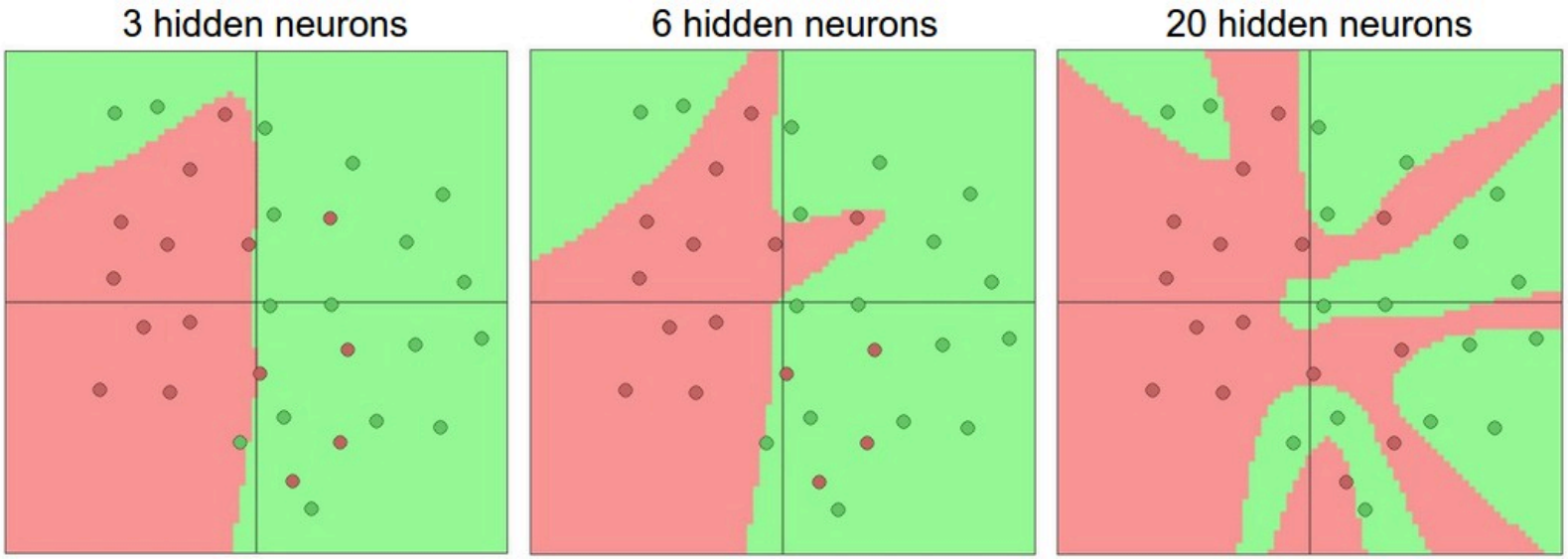




Modern CNNs: ~10 million neurons
Human visual cortex: ~5 billion neurons

Source: Andrej Karpathy & Fei-Fei Li

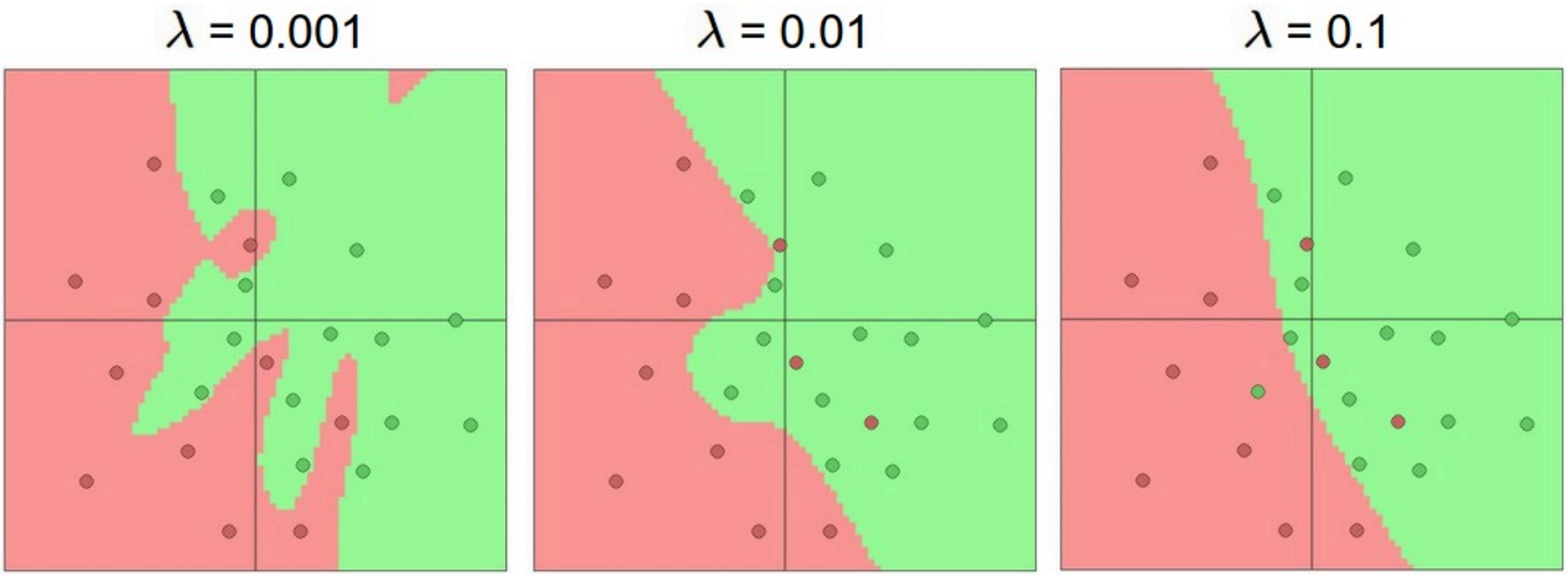
Setting the number of layers and their sizes



more neurons = more capacity

Source: Andrej Karpathy & Fei-Fei Li

Do not use size of neural network as a regularizer. Use stronger regularization instead:



(you can play with this demo over at ConvNetJS:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

Source: Andrej Karpathy & Fei-Fei Li

Summary

- we arrange neurons into fully-connected layers
- the abstraction of a layer has a nice property in that it allows us to use efficient vectorized code (matrix multiplies)
- neural networks are universal function approximators but this doesn’t mean much.
- neural networks are not *neural*

Source: Andrej Karpathy & Fei-Fei Li