ENGN6528

Lecture 10A: Deep Learning for Computer Vision

Thalaiyasingam Ajanthan

Slides from Fei-Fei Li, Andrew Ng, Miaomiao Liu and others.

Lecture schedule

- Logistic regression (last week)
- Deep learning for computer vision (today)
  - Perceptron
  - Backpropagation
  - CNNs
- Deep learning applications in CV (tomorrow)
- Generative models (next week)
- Training neural networks: tricks of the trade (week-12)

Deep learning for classification

- Classifier: (binary/multi-class) logistic regression
- ~~Uses hand designed features~~ Learn features from data
- Optimized via (stochastic) gradient descent
- Use regularization, data augmentation, etc. to reduce overfitting

Recap: Logistic regression

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Recap: Logistic regression

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Loss function: $L(\theta; \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta)$

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

$$h_\theta(x) = g(\theta^T x) \quad \text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

Recap: Logistic regression

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Loss function: $L(\theta; \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta)$

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

$$h_\theta(x) = g(\theta^T x) \quad \text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

Objective: $\min_\theta L(\theta; \mathcal{D})$

Recap: Logistic regression

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Loss function: $L(\theta; \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta)$

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

$$h_\theta(x) = g(\theta^T x) \quad \text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

Objective: $\min_\theta L(\theta; \mathcal{D})$

Optimization: $\theta^{t+1} = \theta^t - \alpha \nabla_\theta L(\theta^t; \mathcal{B}^t) \quad \text{where} \quad \mathcal{B}^t \subset \mathcal{D}$

Recap: Logistic regression

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Loss function: $L(\theta; \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta)$

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

$$h_\theta(x) = g(\theta^T x) \quad \text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

Objective: $\min_\theta L(\theta; \mathcal{D})$

Optimization: $\theta^{t+1} = \theta^t - \alpha \nabla_\theta L(\theta^t; \mathcal{B}^t) \quad \text{where} \quad \mathcal{B}^t \subset \mathcal{D}$

Prediction: $\text{if} \quad h_\theta(x) \geq 0.5, \quad \text{predict} \quad y = 1$

Deep learning

Training set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Loss function: $L(\theta; \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta)$

Eg: $\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$

$h_\theta(x) = g\left(f_\theta(x)\right)$ Learn the features too!

Objective: $\min_\theta L(\theta; \mathcal{D})$

Optimization: $\theta^{t+1} = \theta^t - \alpha \nabla_\theta L(\theta^t; \mathcal{B}^t)$ where $\mathcal{B}^t \subset \mathcal{D}$

Prediction: if $h_\theta(x) \geq 0.5$, predict $y = 1$

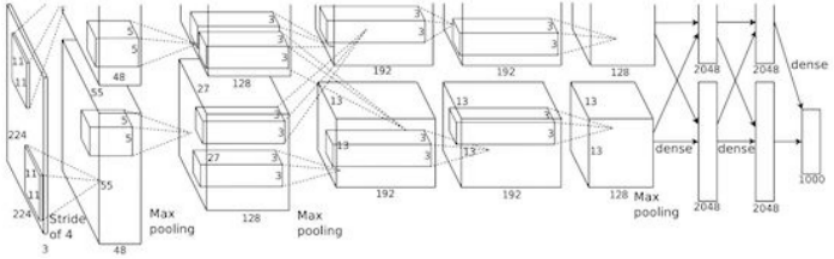
Image features vs ConvNets

Feature Extraction

f

10 numbers giving scores for classes

training



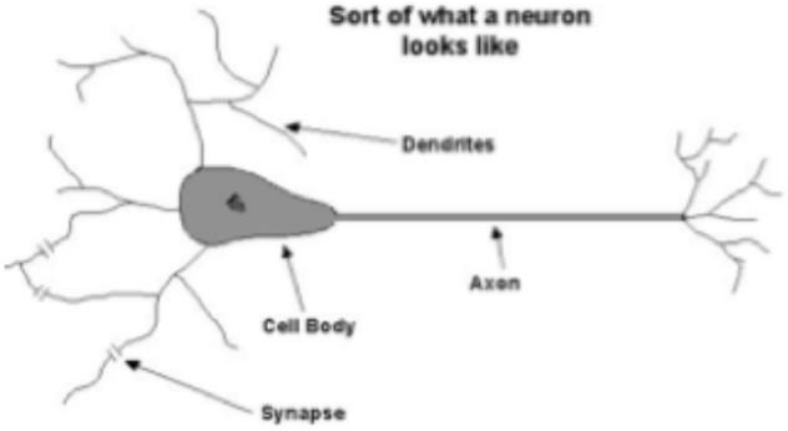
Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

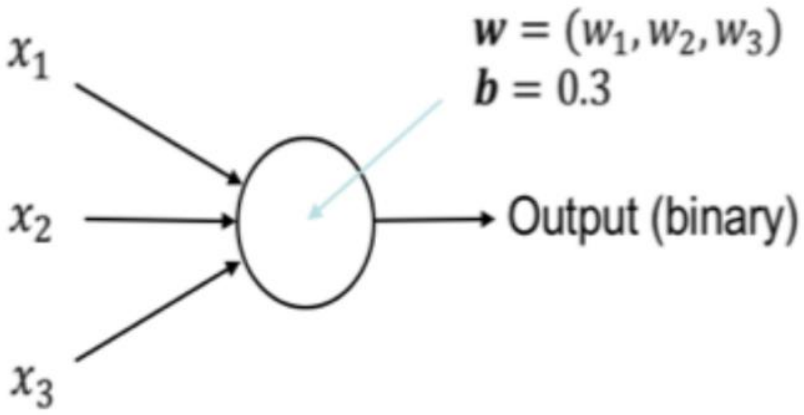
10 numbers giving scores for classes

training

Inspiration: Biological Neurons

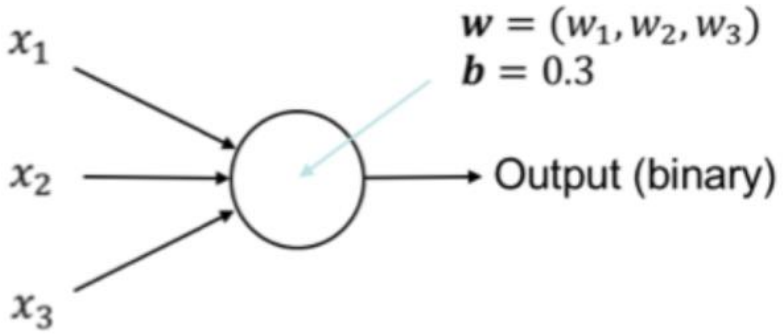
- Neurons
  - accept information from multiple inputs,
  - transmit (process and relay) information to other neurons.
- Multiply inputs by weights along edges
- Apply some **activation function** to the output of each node





Perceptron as linear classifier

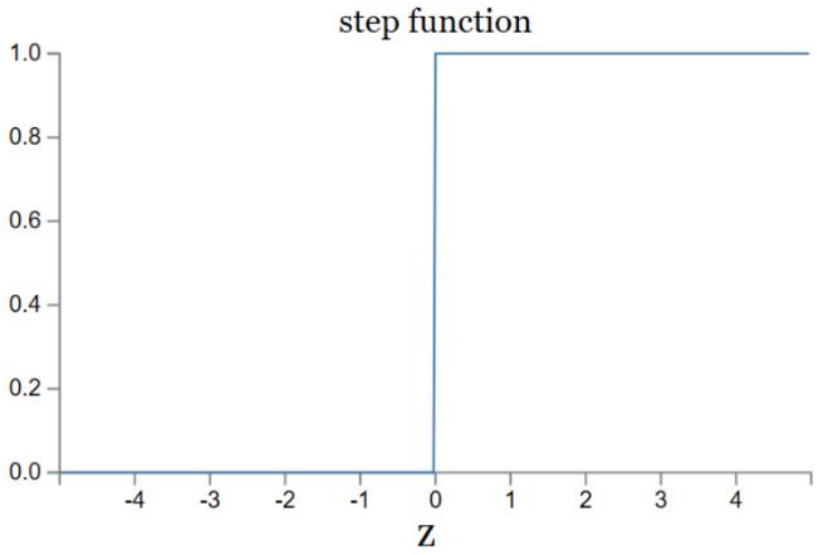
- Basic building block for composition is **perceptron** (Rosenblatt c.1960)
- Linear neuron classifier – vector of weights *w* and a 'bias' *b*



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

Perceptron's activation function

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

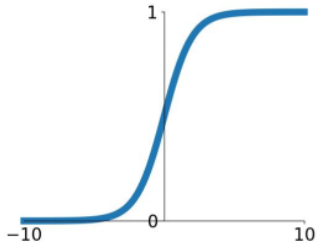


In the context of neural networks, a perceptron is an artificial neuron using the step function as the activation function.

Activation functions

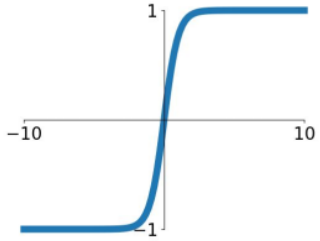
Sigmoid

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



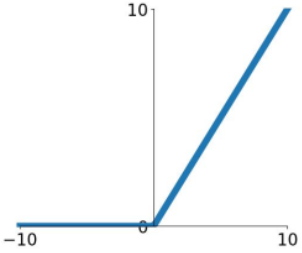
tanh

$$\tanh(x)$$



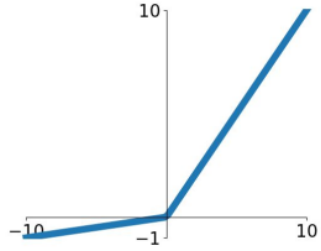
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

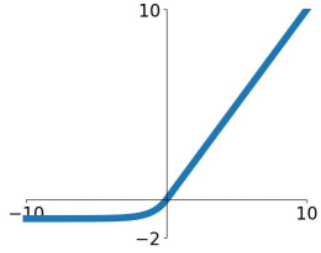


Maxout

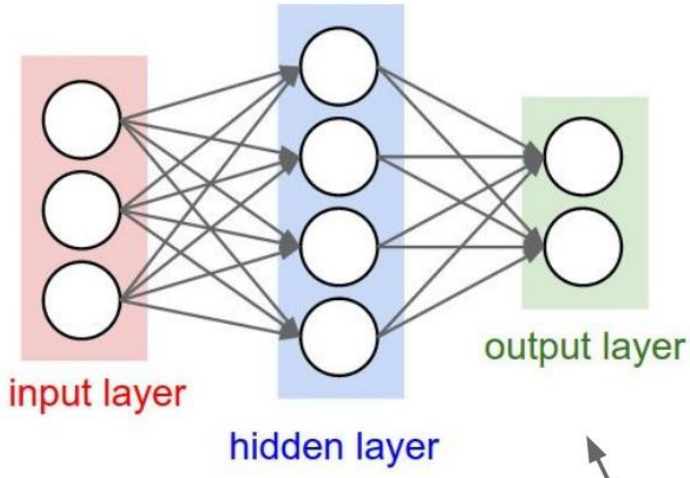
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

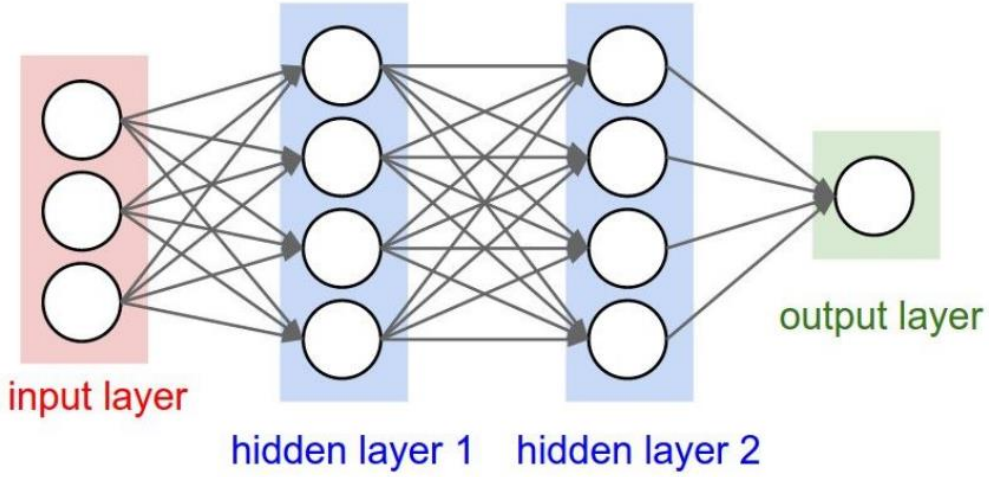
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural networks: Architectures



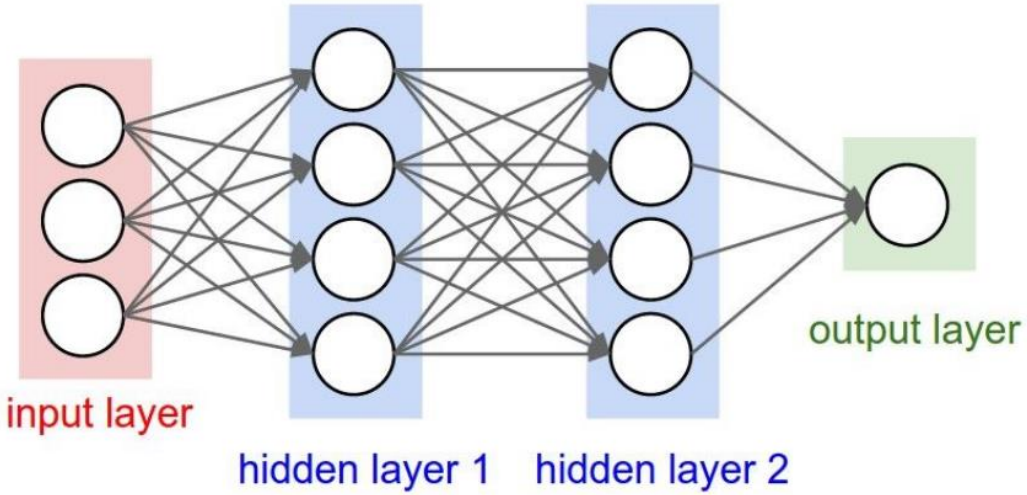


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

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

"Fully-connected" layers

Example feed-forward computation of a neural network




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Why nonlinear activations?

Problems with using all-linear functions in the hidden layers

- We have formed multiple layer or chains of linear functions.
- We know that linear function of linear functions is a linear function, i.e. they can be reduced:
  - $g = f(h(x))$
  - Eg. $O_1 = xw_1+b_1$; $O_2 = o_1w_2+b_2$; $=>O_2 = (xw_1+b_1)w_2+b_2=xw_1w_2+(b_1w_2+b_2)$

Our multi-layer composition of functions is really just a single linear function : (

Mathematical Theorem for MLP

- The application of MLP is grounded in theory.
  - **Universal Approximation Theorem.**

With three (or more) layers and enough parameters, MLP can approximate any function.

Proof(Michael Nielson):

http://neuralnetworksanddeeplearning.com/chap4.html

Why go **deeper** ?

- A three-layer network is already a “universal approximator”,
- Only if the number of neurons in the hidden layer is sufficiently many.
- The identification of the weights is troublesome (possible convergence into wrong local minima)
- Deep neural networks
  - Have many levels of non-linearity
  - Compactly represent highly non-linear functions
  - Layered structure facilitates convergence to a good optimum
  - **There are many local optima of approximately same (and good) quality**

Interpretation

Question: What does a hidden unit do?

Answer: It can be thought of as a classifier or feature detector.

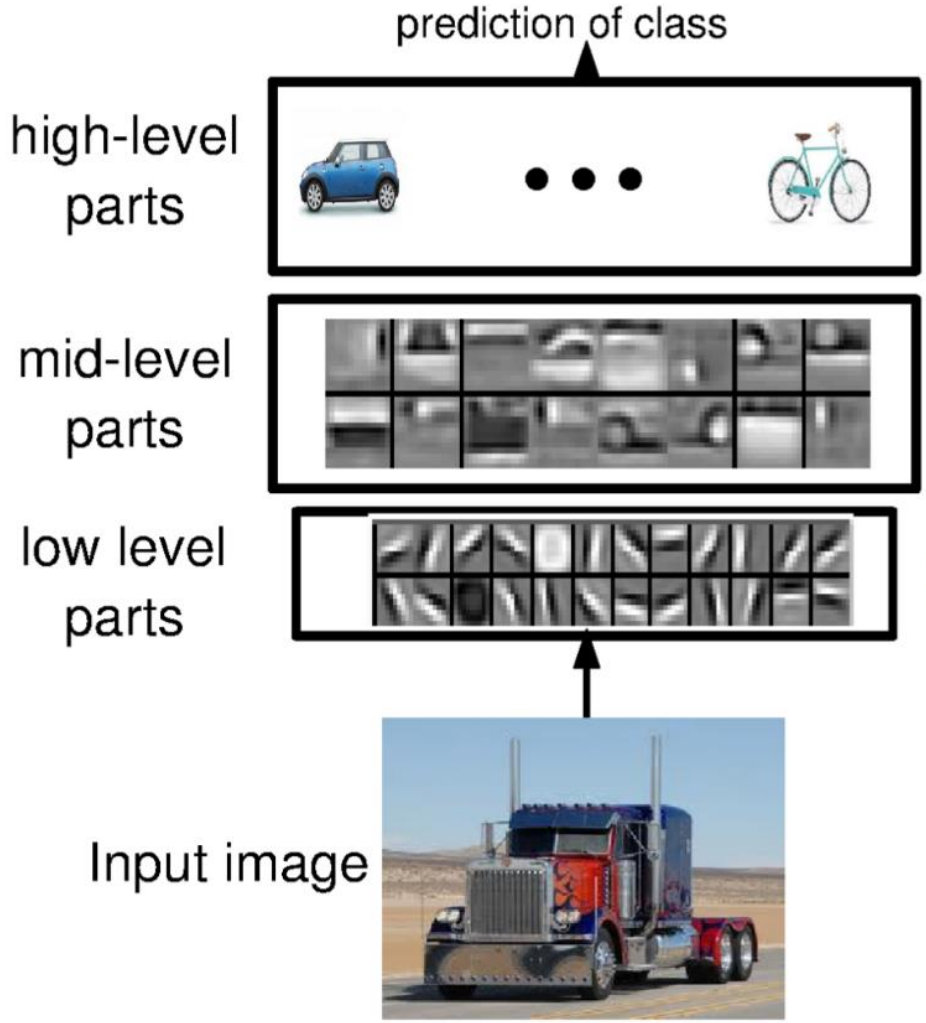
Question: How many layers? How many hidden units?

Answer: Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

Question: How do I set the weight matrices?

Answer: Weight matrices and biases are learned.
First, we need to define a measure of quality of the current mapping.
Then, we need to define a procedure to adjust the parameters.

Interpretation



- distributed representations
- feature sharing
- compositionality

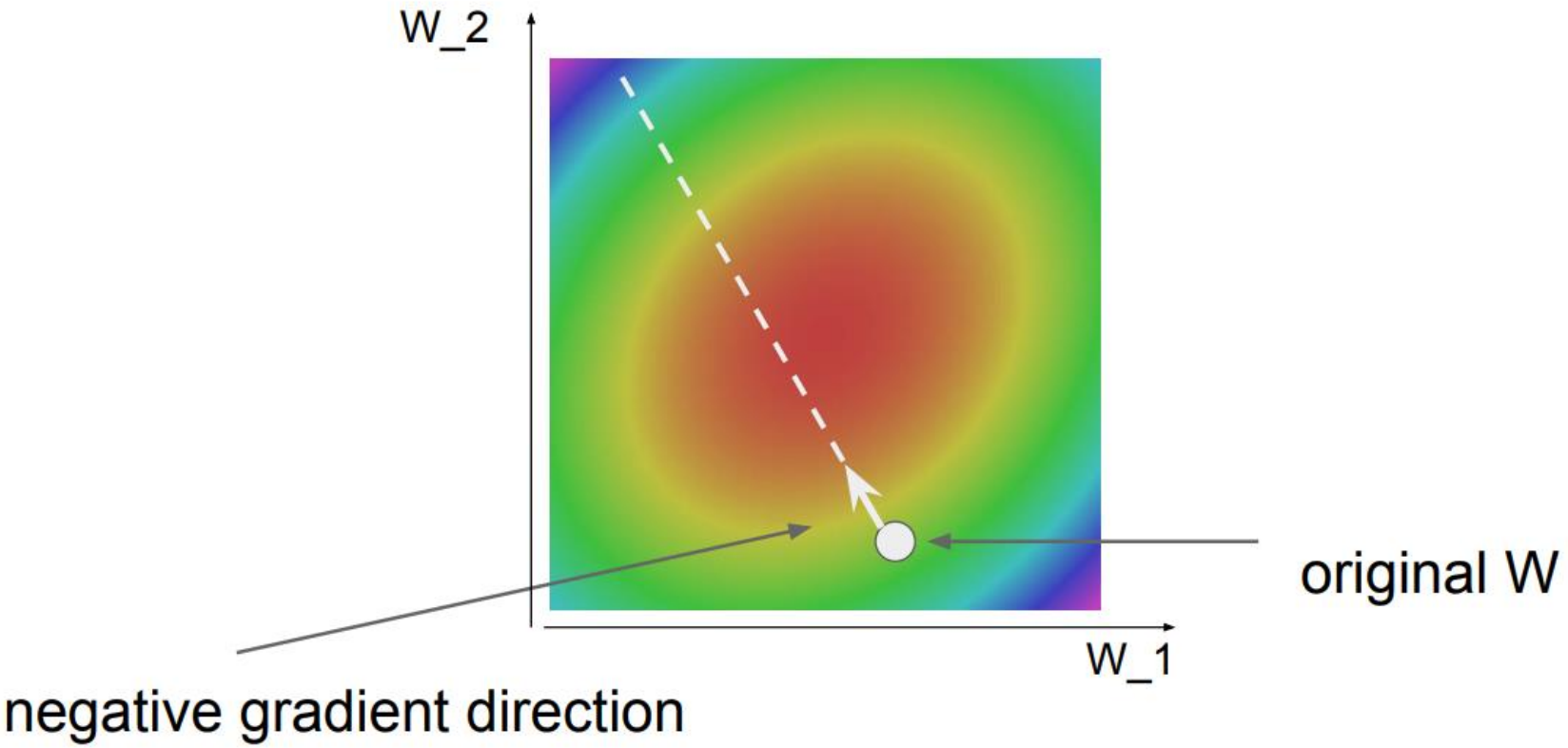
NOTE: Not actually the weights; a demonstrative visualization!

Lee et al. "Convolutional DBN's ..." ICML 2009

Full implementation of training a 2-layer Neural Network needs ~20 lines:

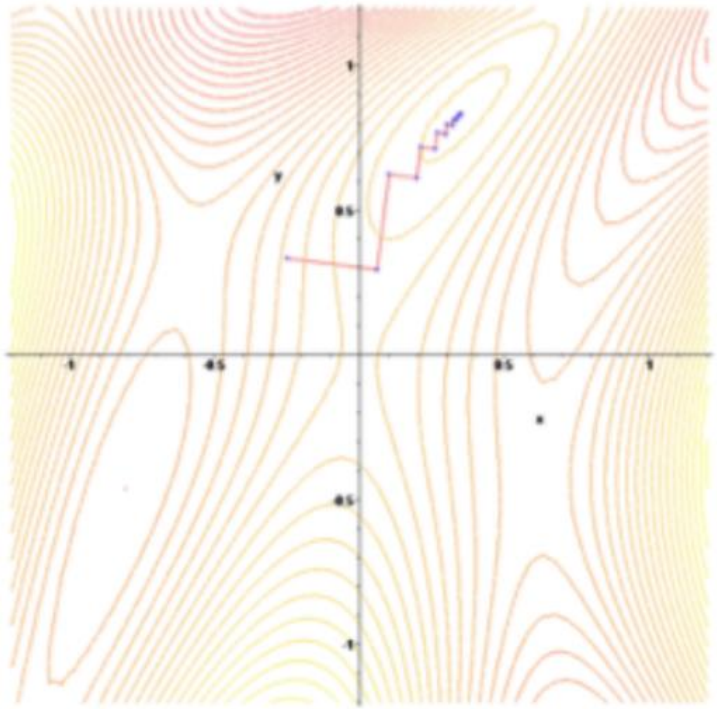
```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

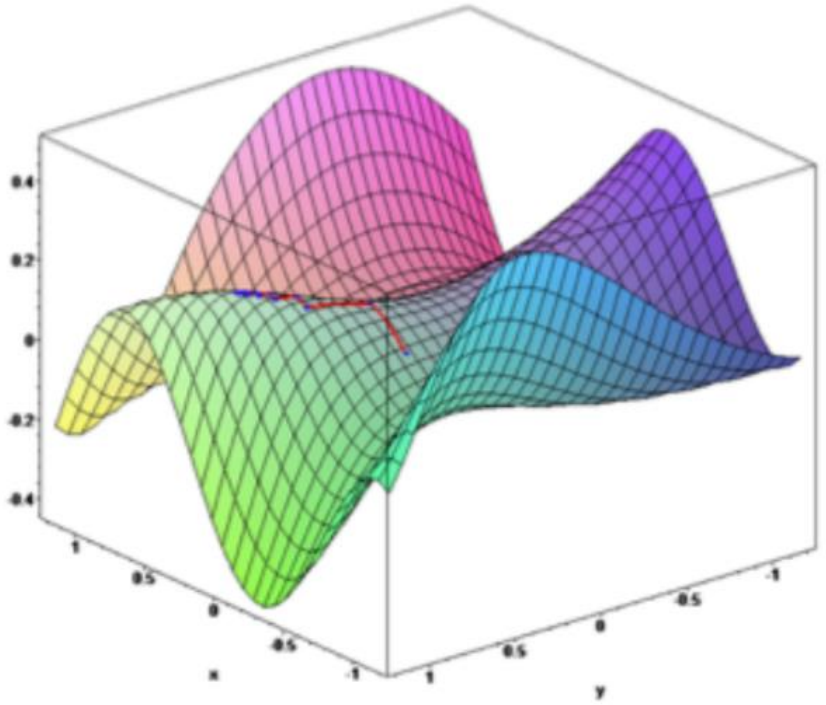
Gradient descent



Gradient descent

- Works in arbitrary dimensions





Training deep neural networks

- Outlook:
  - Computing gradients is hard (many parameters)
    → Back-propagation!

- Network represents a chain of function calls (one per layer):

$$\boldsymbol{f}(\mathbf{x}) = \boldsymbol{f}_{L+1} \circ \boldsymbol{f}_L \circ \cdots \circ \boldsymbol{f}_1(\mathbf{x})$$
$$= \boldsymbol{f}_{L+1}(\boldsymbol{f}_L(\ldots \boldsymbol{f}_1(\mathbf{x})))$$

- Gradients: Chain rule can be applied!

$$\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{f}_{L+1}}{\partial \boldsymbol{f}_L} \cdot \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \cdot \ldots \cdot \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\theta}}$$

How to compute gradient?

Symbolic Differentiation

- Input formulae is a symbolic expression tree (computation graph).
- Implement differentiation rules, e.g., sum rule, product rule, chain rule

$$\frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx} \qquad \frac{d(fg)}{dx} = \frac{df}{dx}g + f\frac{dg}{dx} \qquad \frac{d(h(x))}{dx} = \frac{df(g(x))}{dx} \cdot \frac{dg(x)}{x}$$

✗ For complicated functions, the resultant expression can be exponentially large.

✗ Wasteful to keep around intermediate symbolic expressions if we only need a numeric value of the gradient in the end

✗ Prone to error

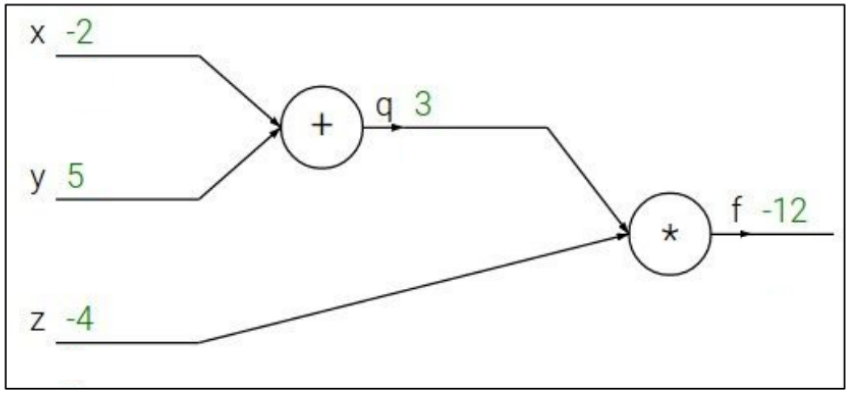
Backpropagation for computing gradients

Backpropagation for computing gradients

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Backpropagation for computing gradients

Backpropagation: a simple example

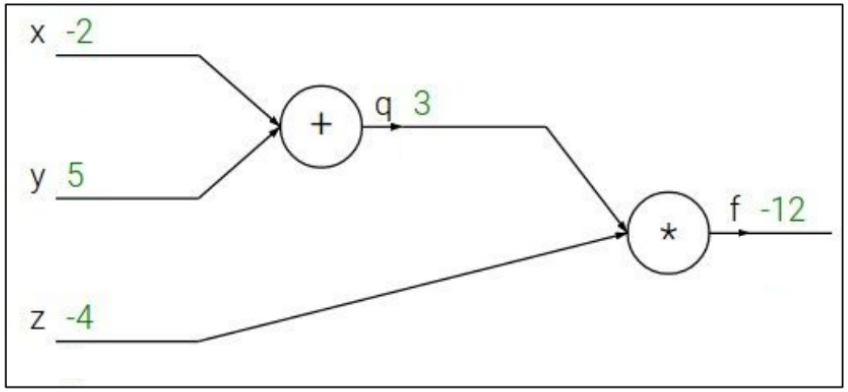
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation for computing gradients

Backpropagation: a simple example

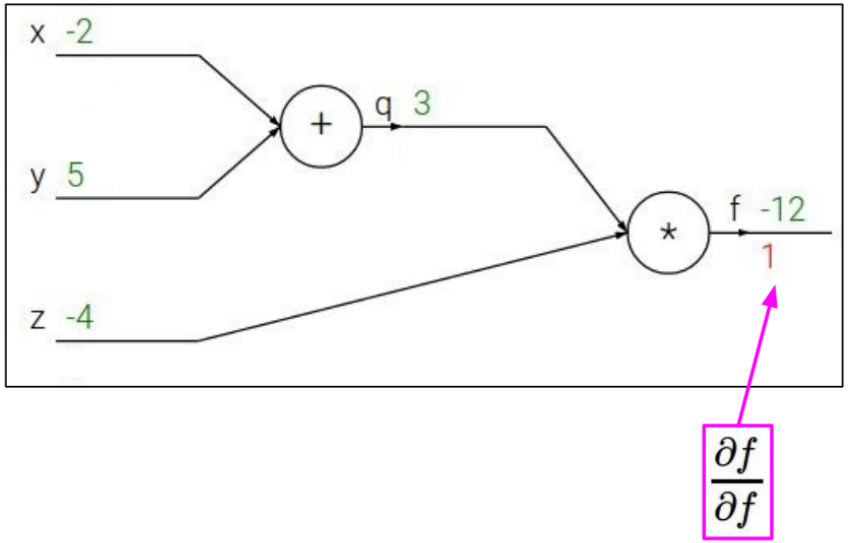
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation for computing gradients

Backpropagation: a simple example

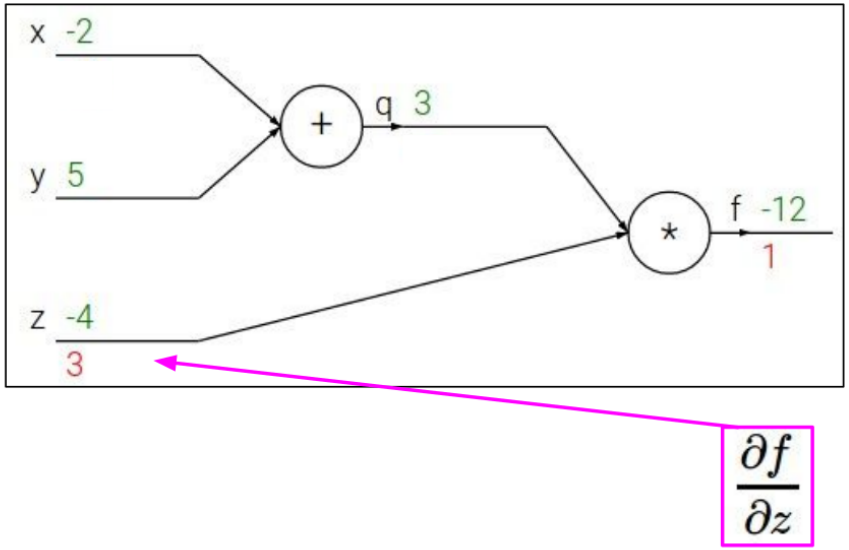
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation for computing gradients

Backpropagation: a simple example

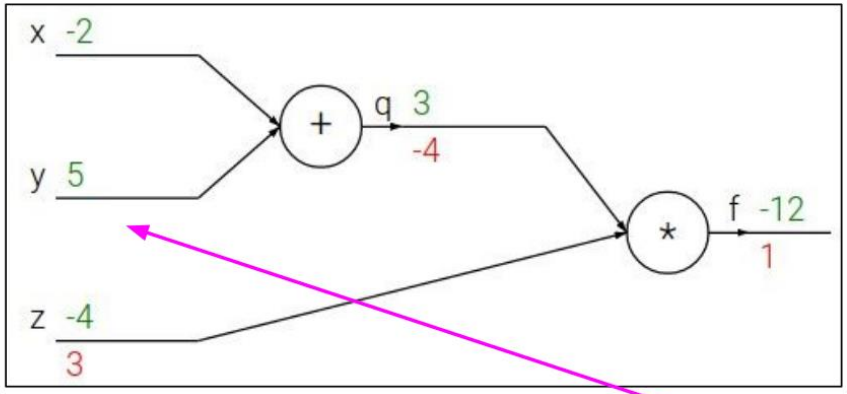
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

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

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

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



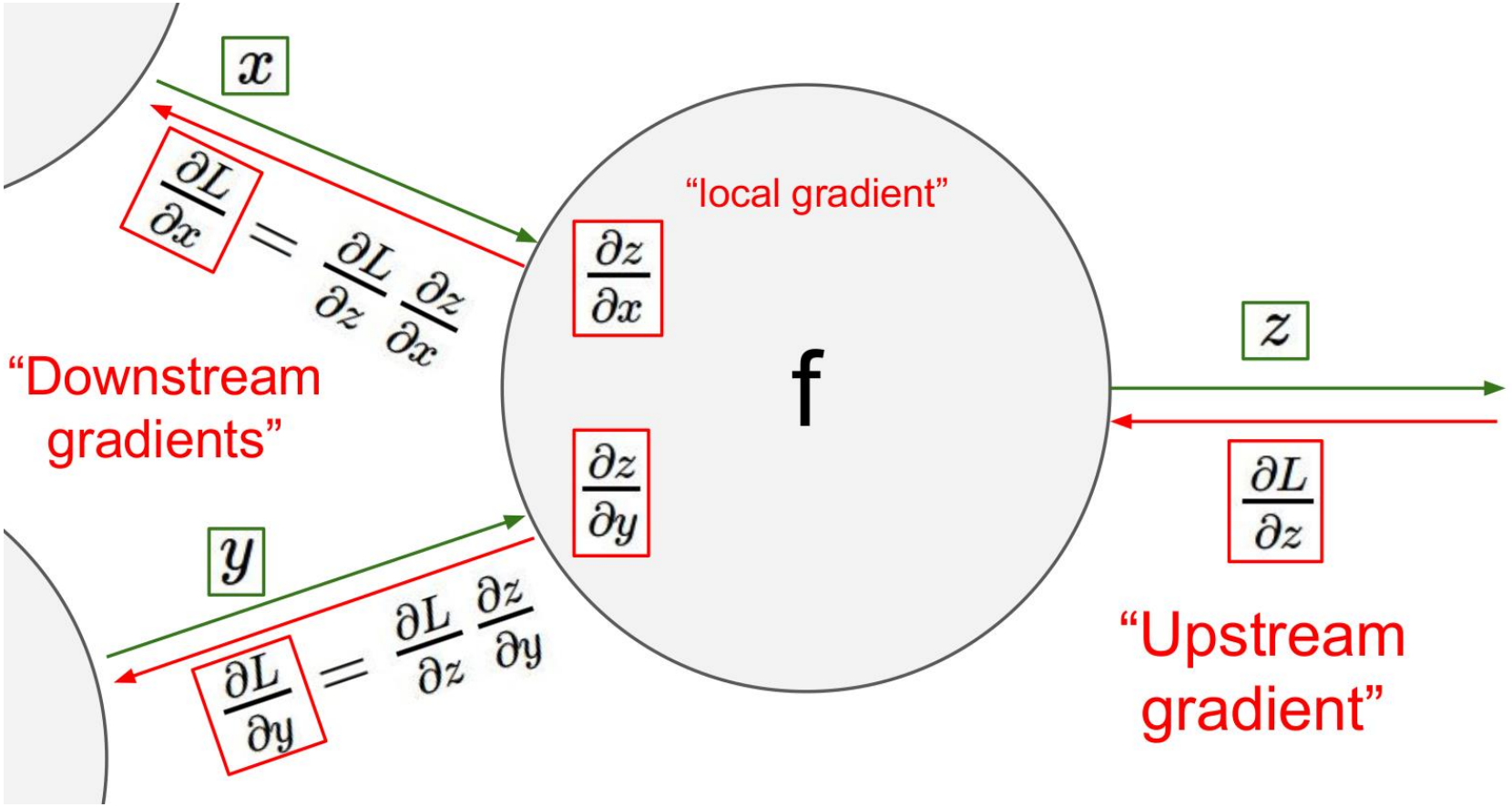
Chain rule:

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

Upstream gradient

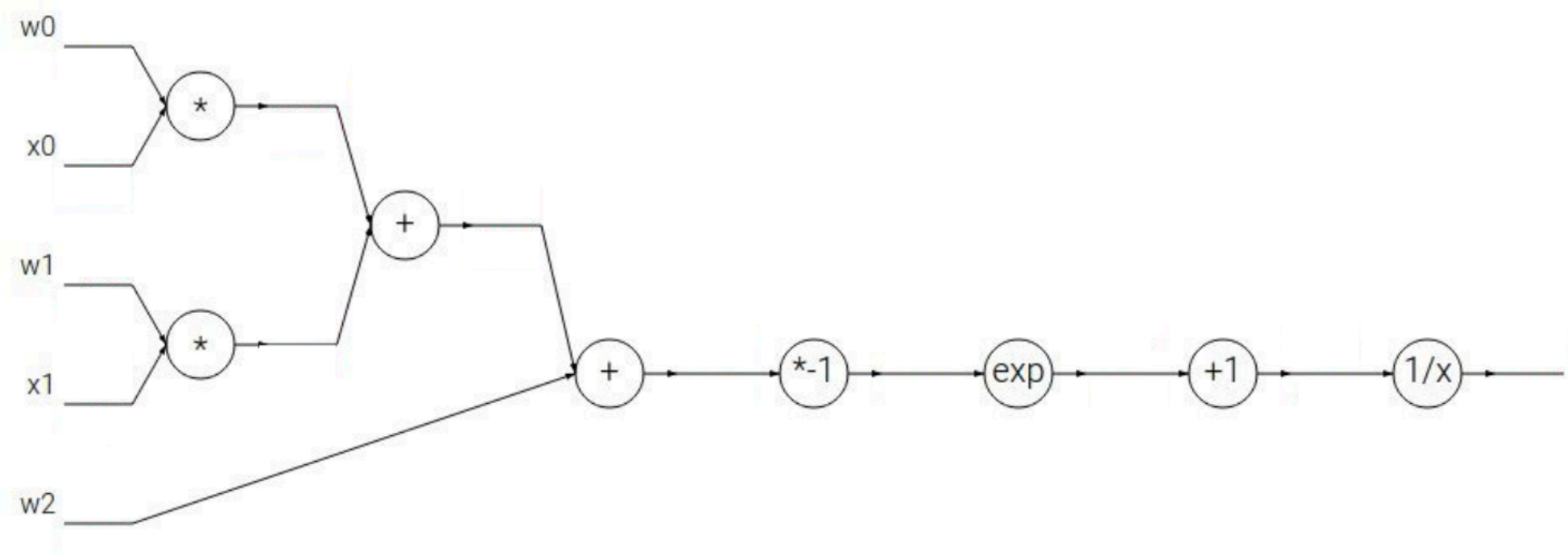
Local gradient

Backpropagation for computing gradients



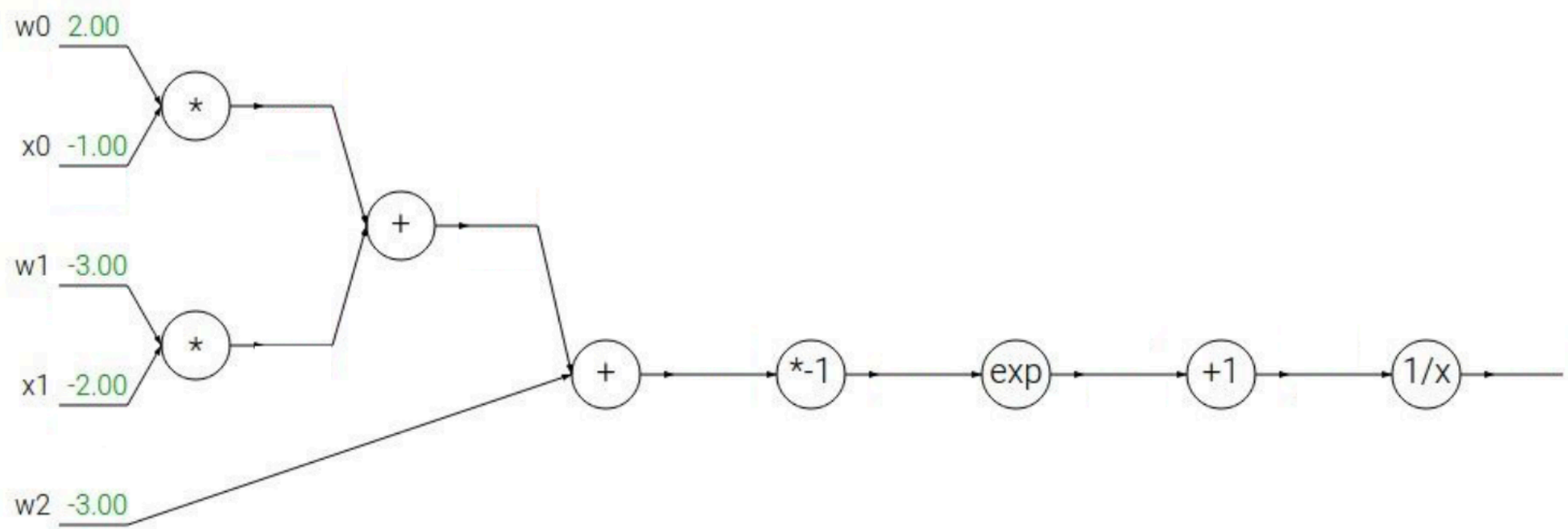
Another example:

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



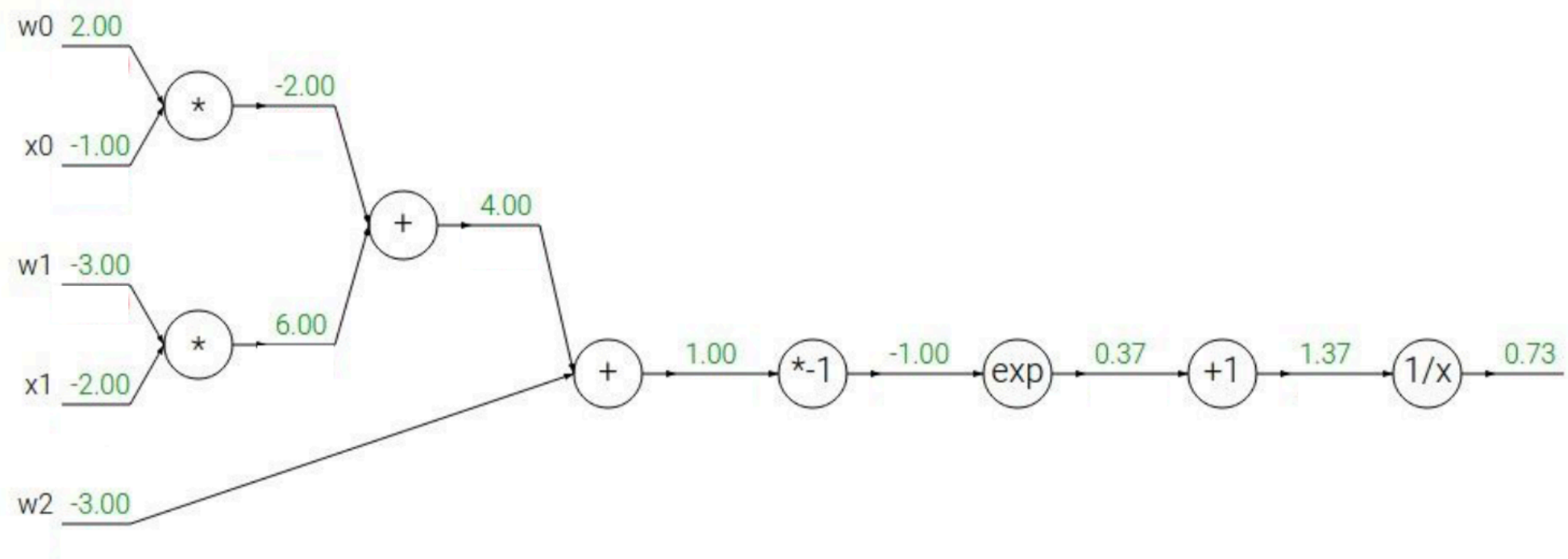
Another example:

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

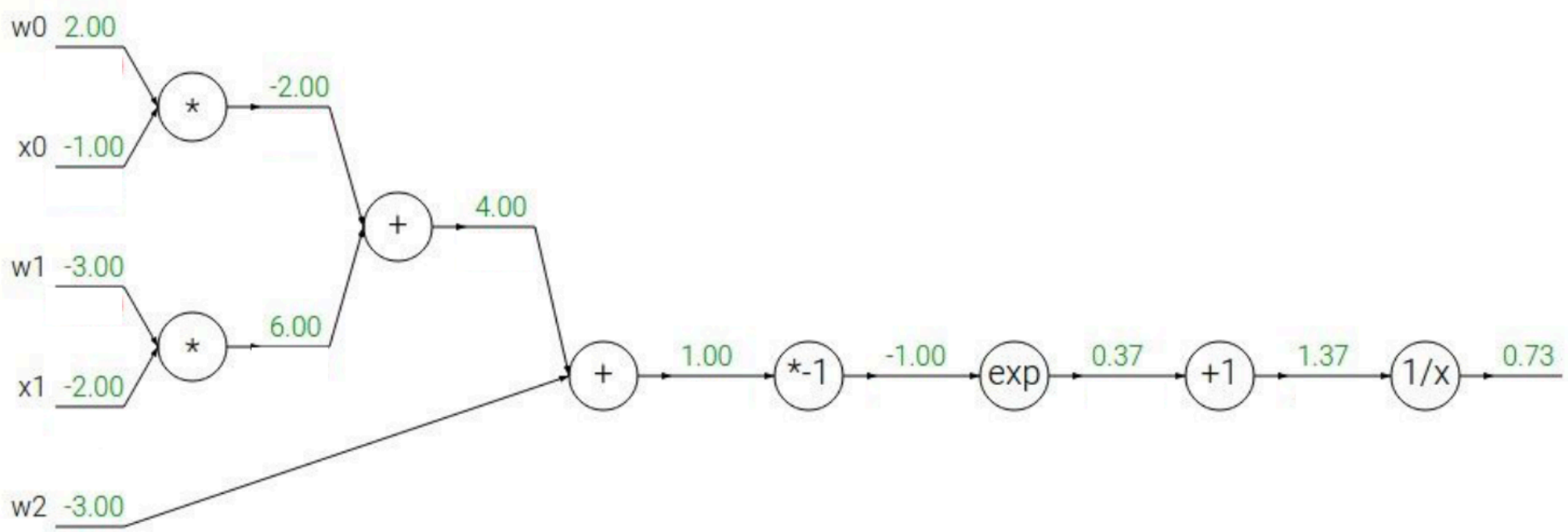


Another example:

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



Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



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

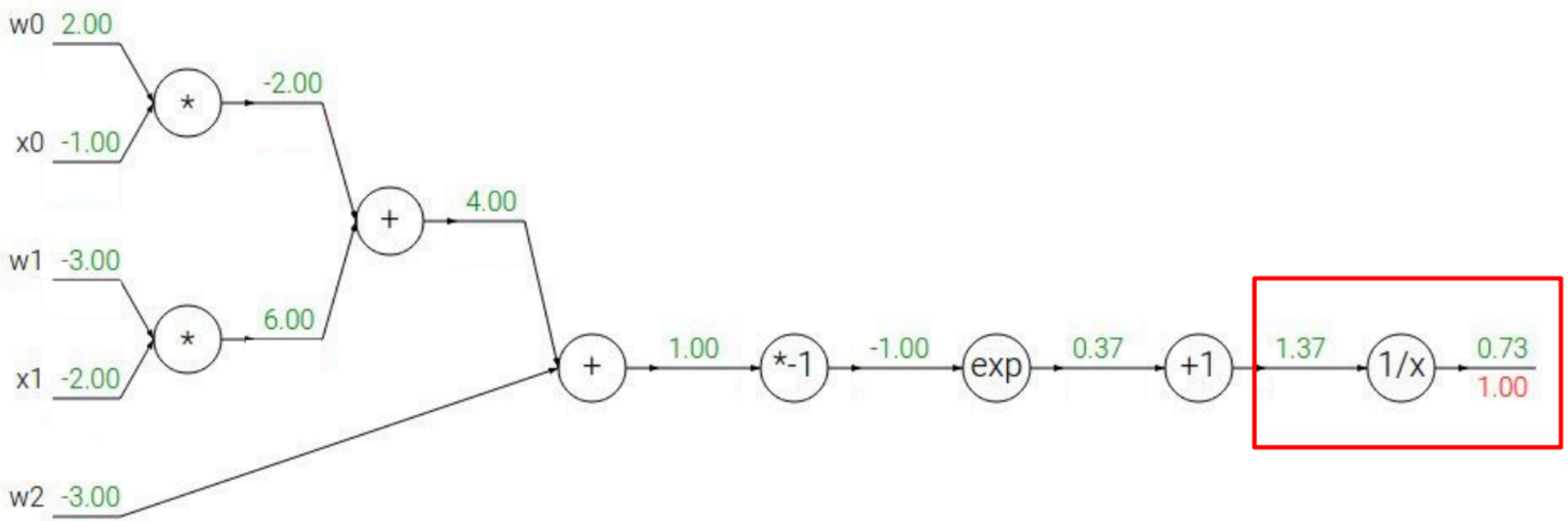
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

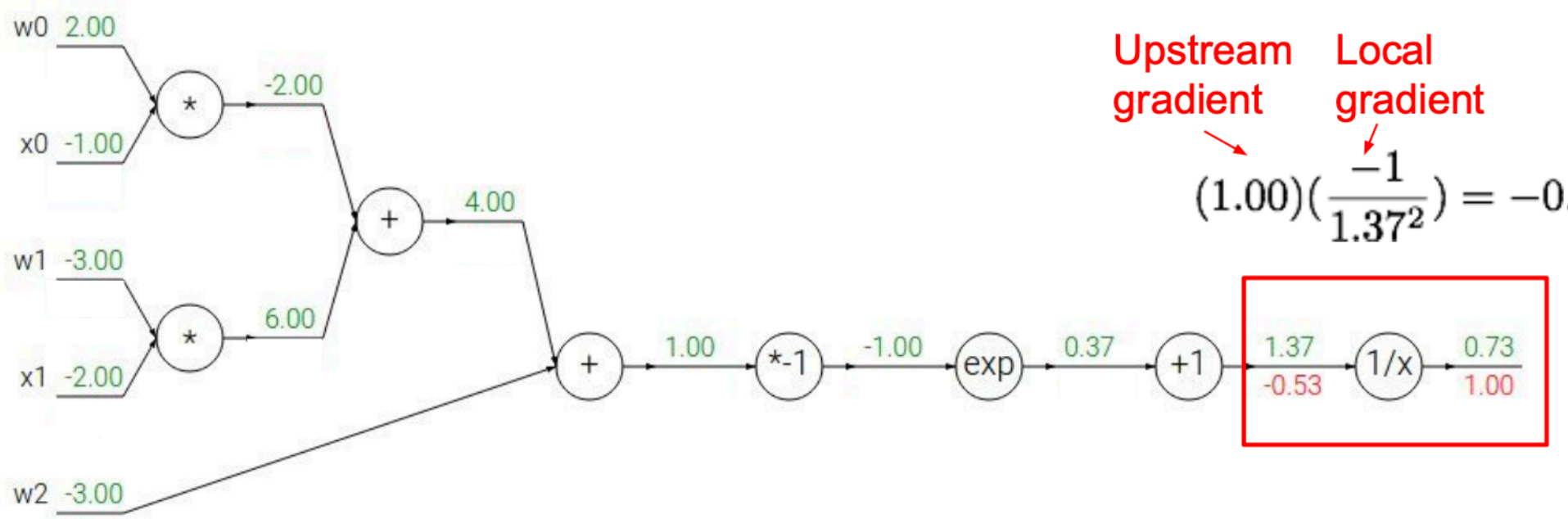
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

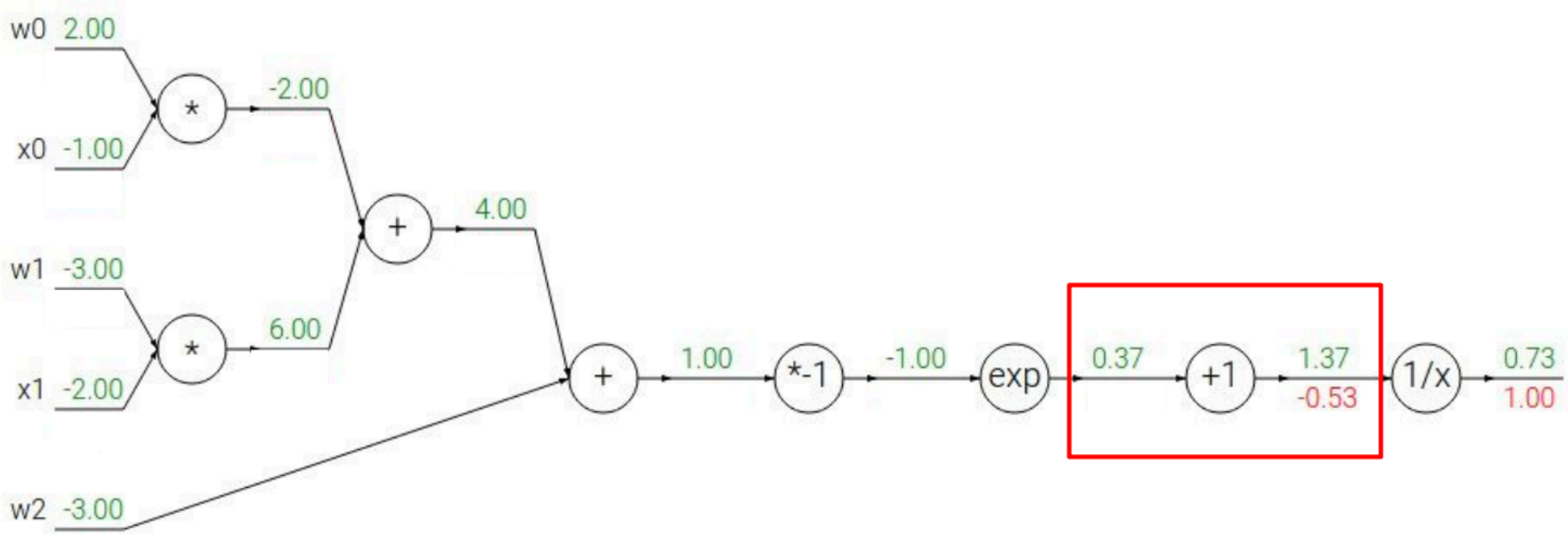
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

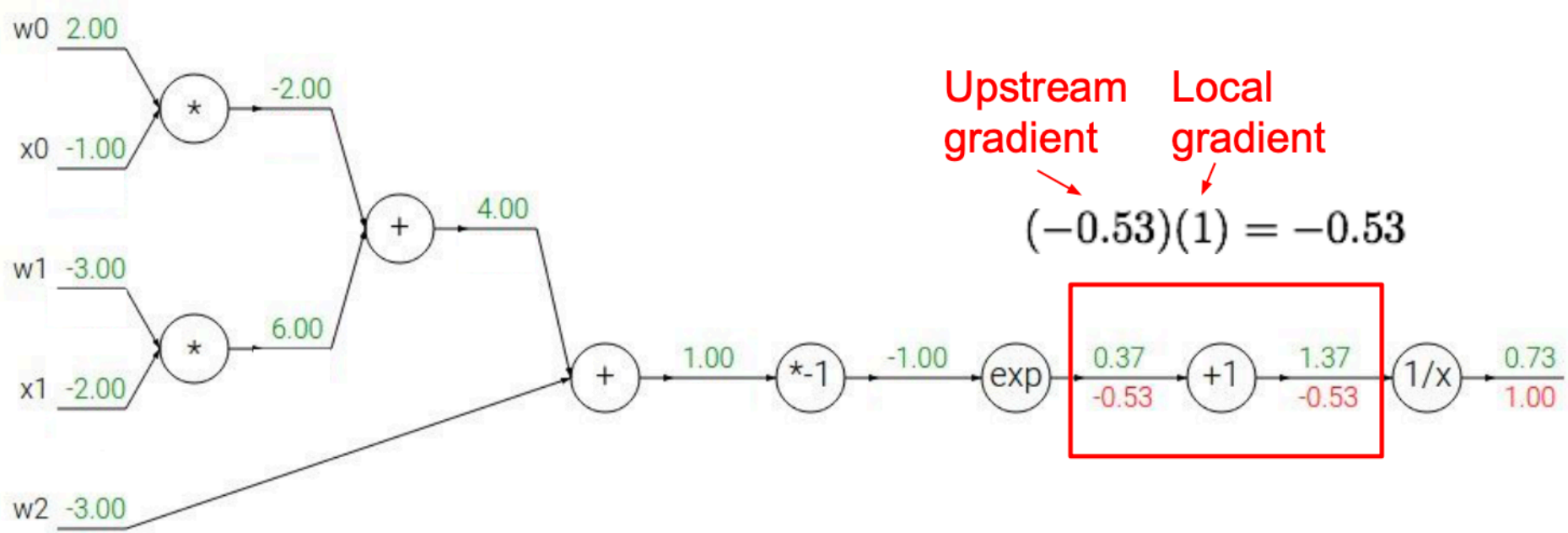
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

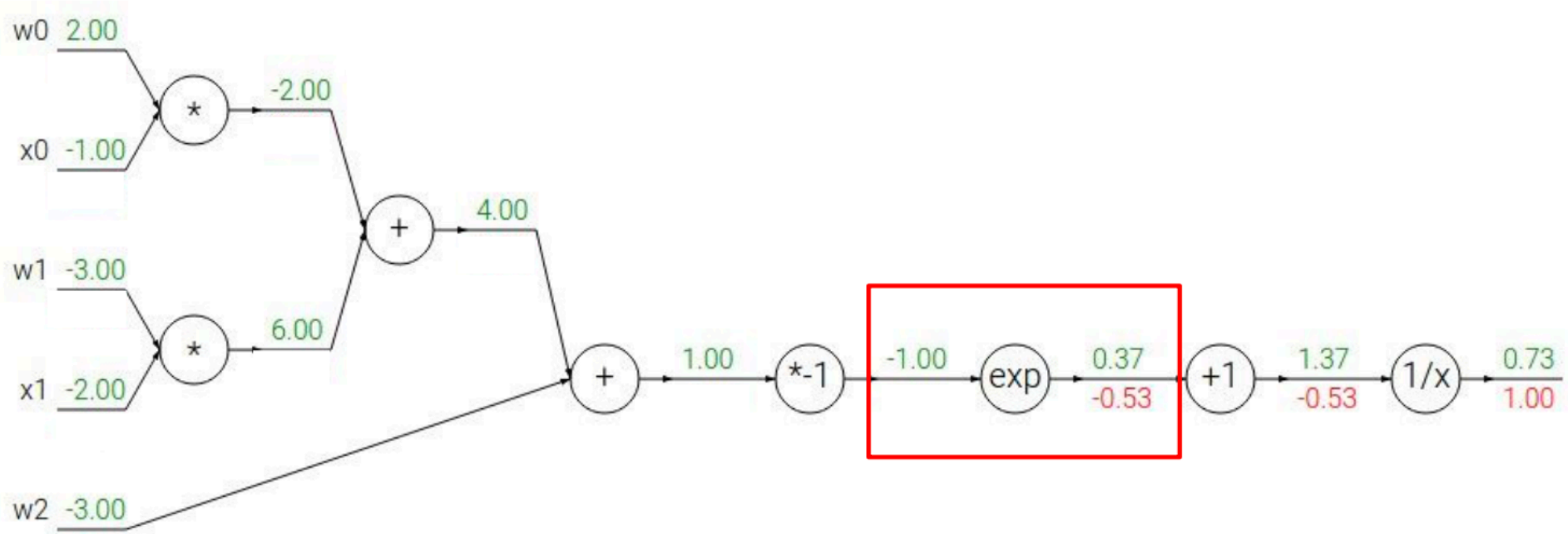
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

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

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

Another example:

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



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

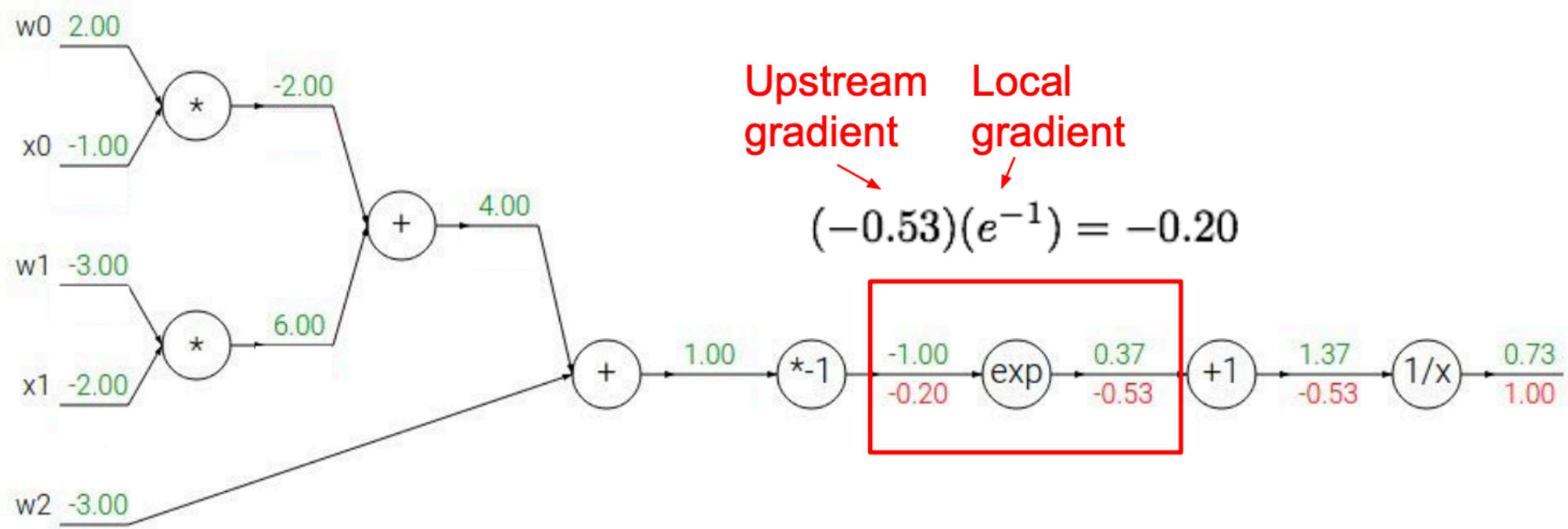
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

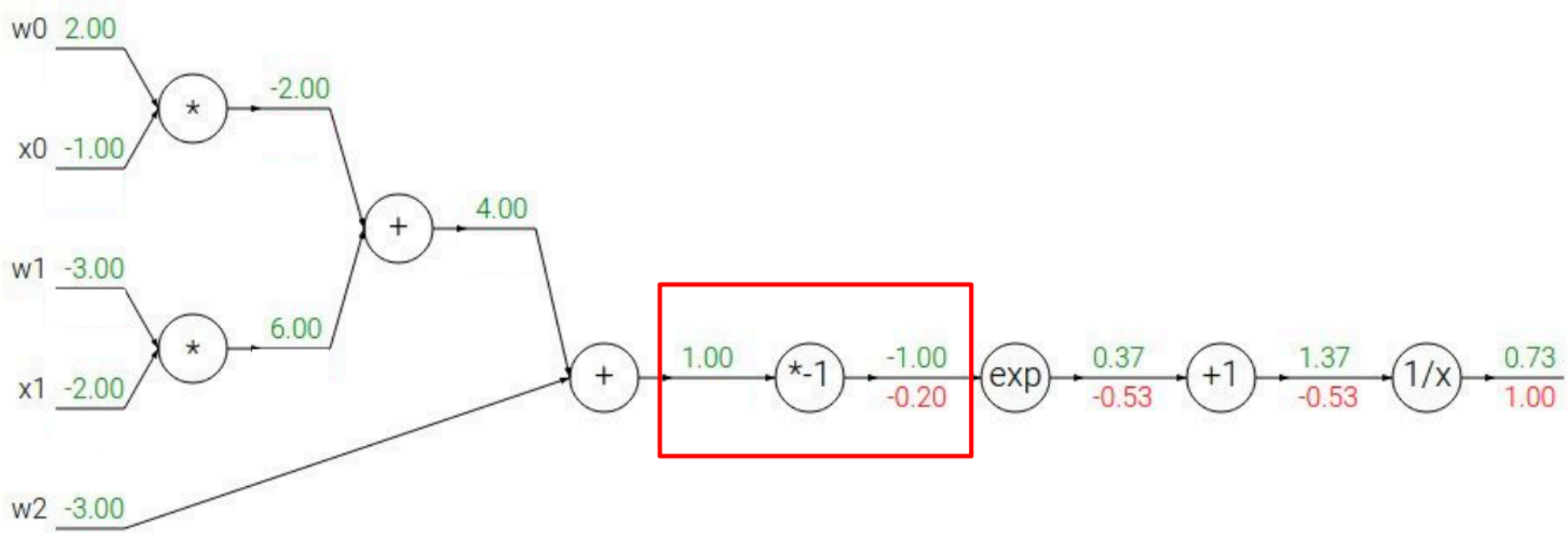
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

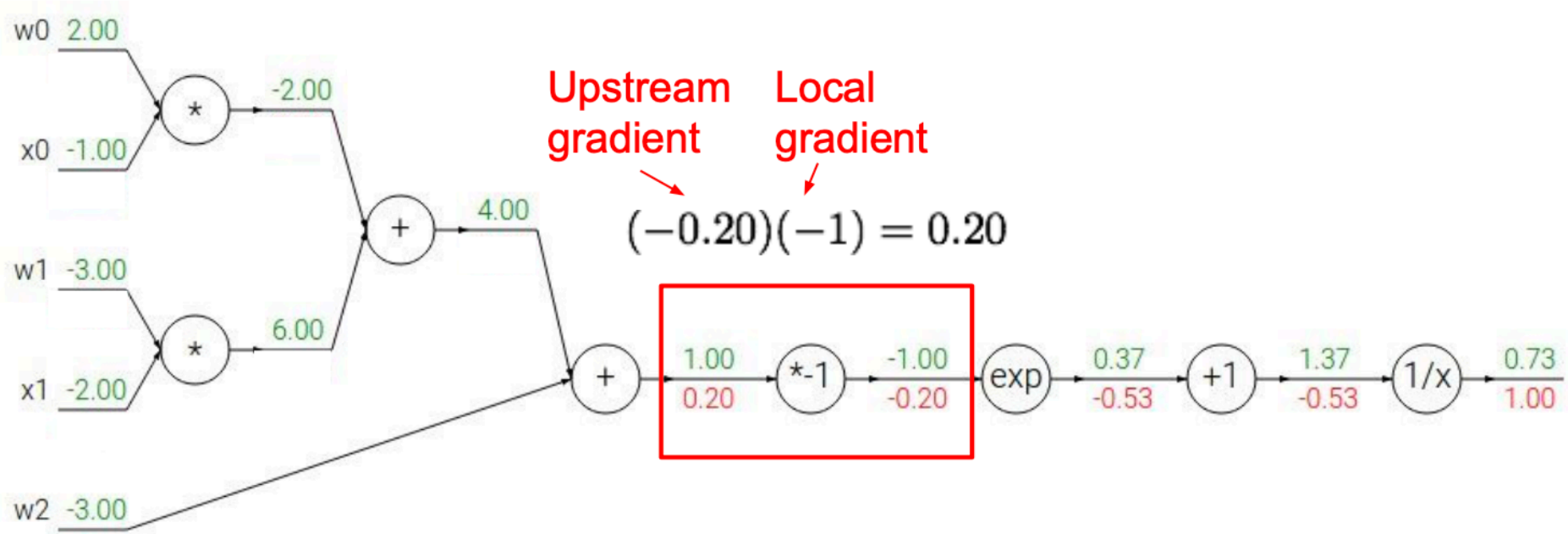
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

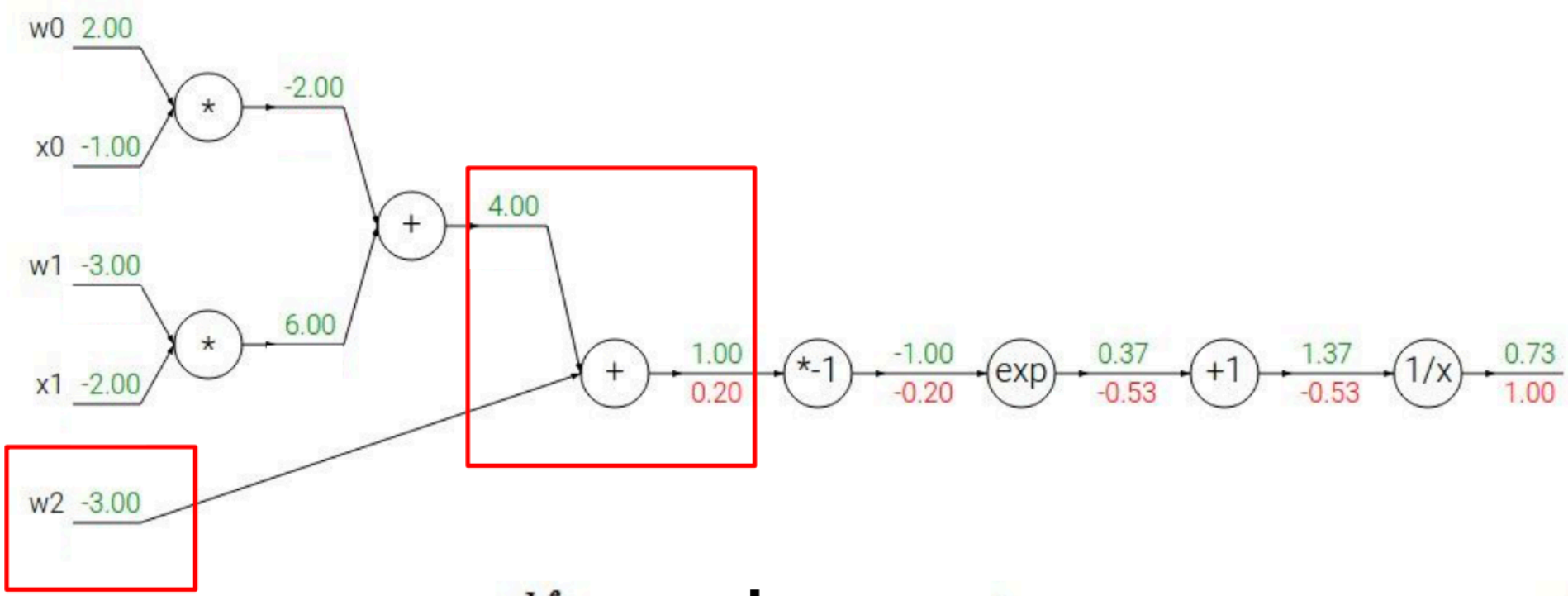
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



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

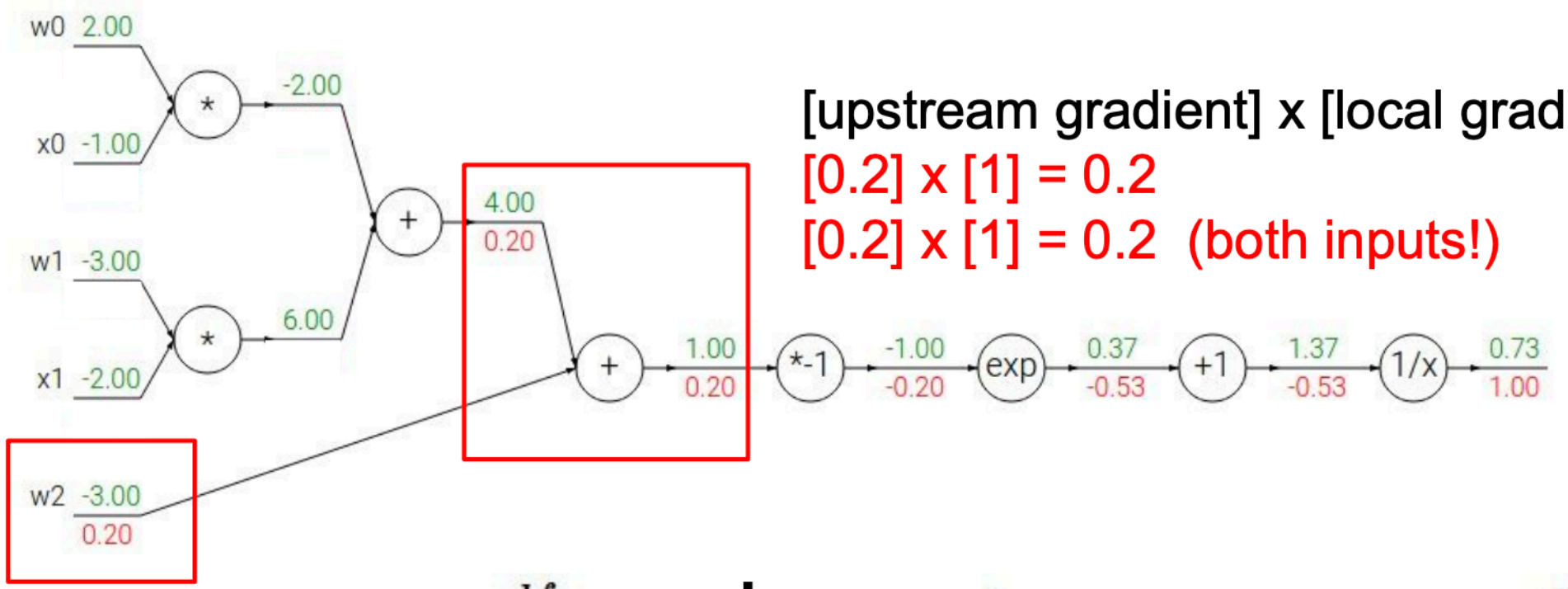
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Another example:

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



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

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

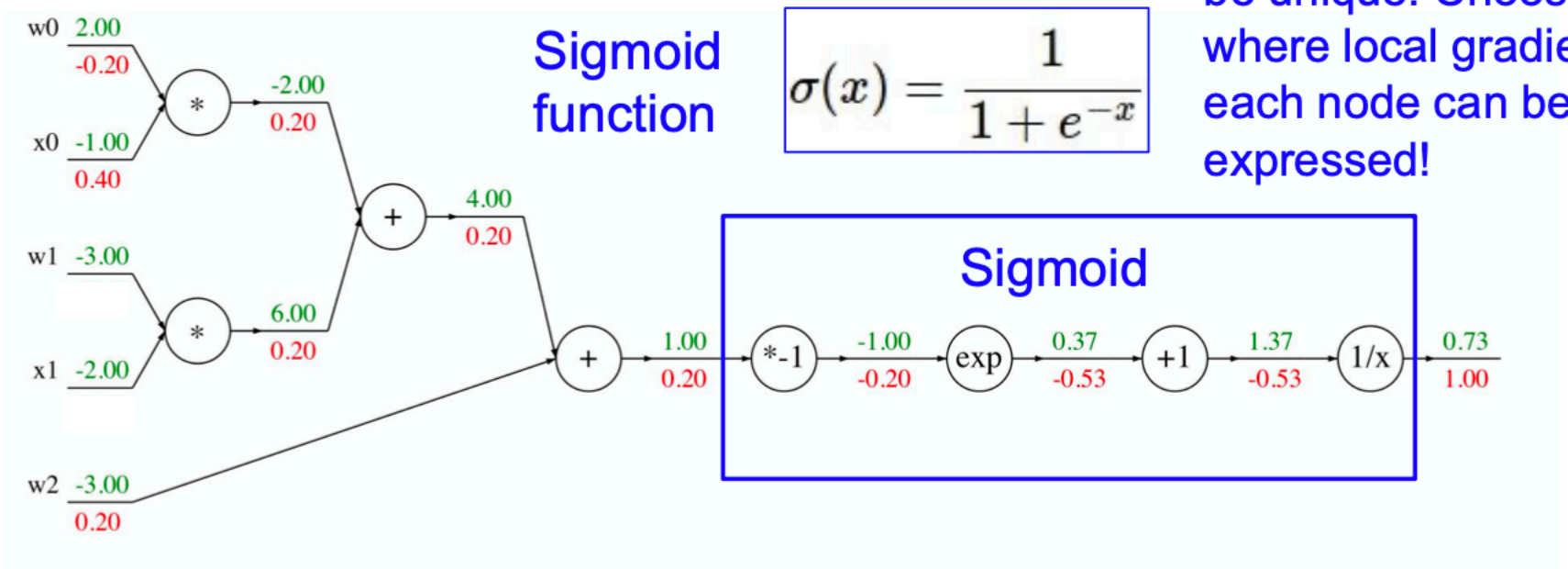
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

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

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

Another example:

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



Sigmoid function

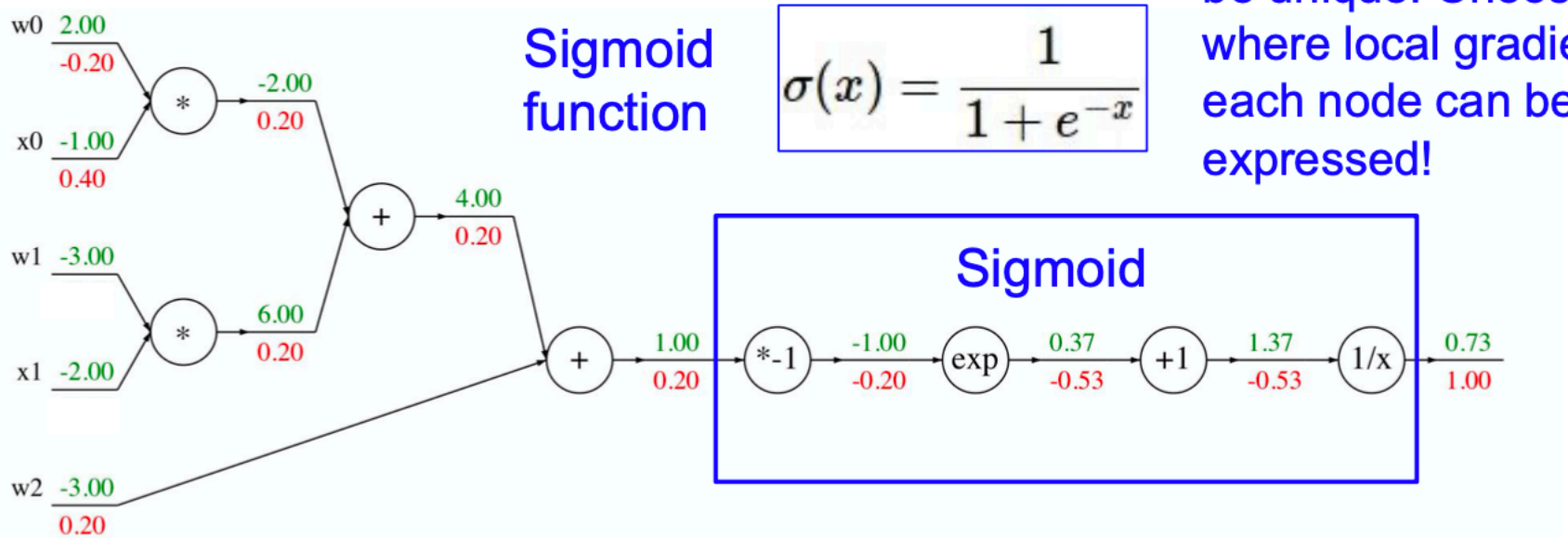
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

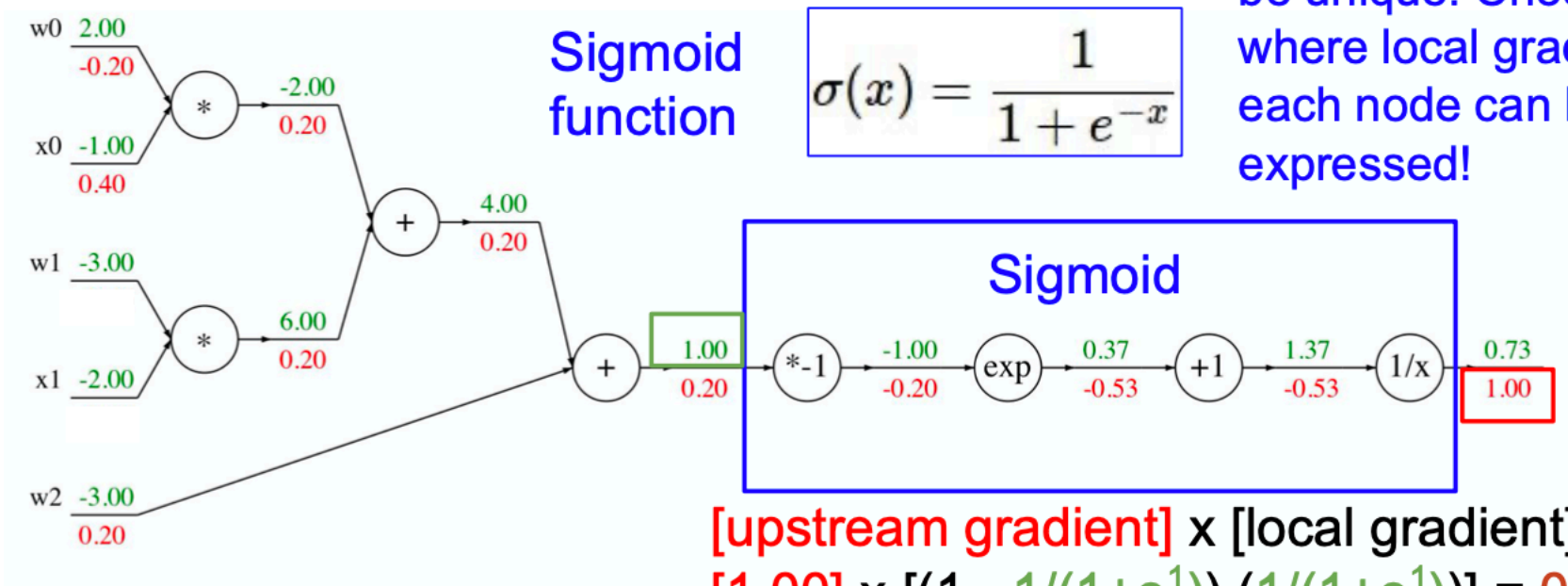
Sigmoid local gradient:

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

Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

[1.00] x [(1 - 1/(1+e^{1})) (1/(1+e^{1}))] = 0.2

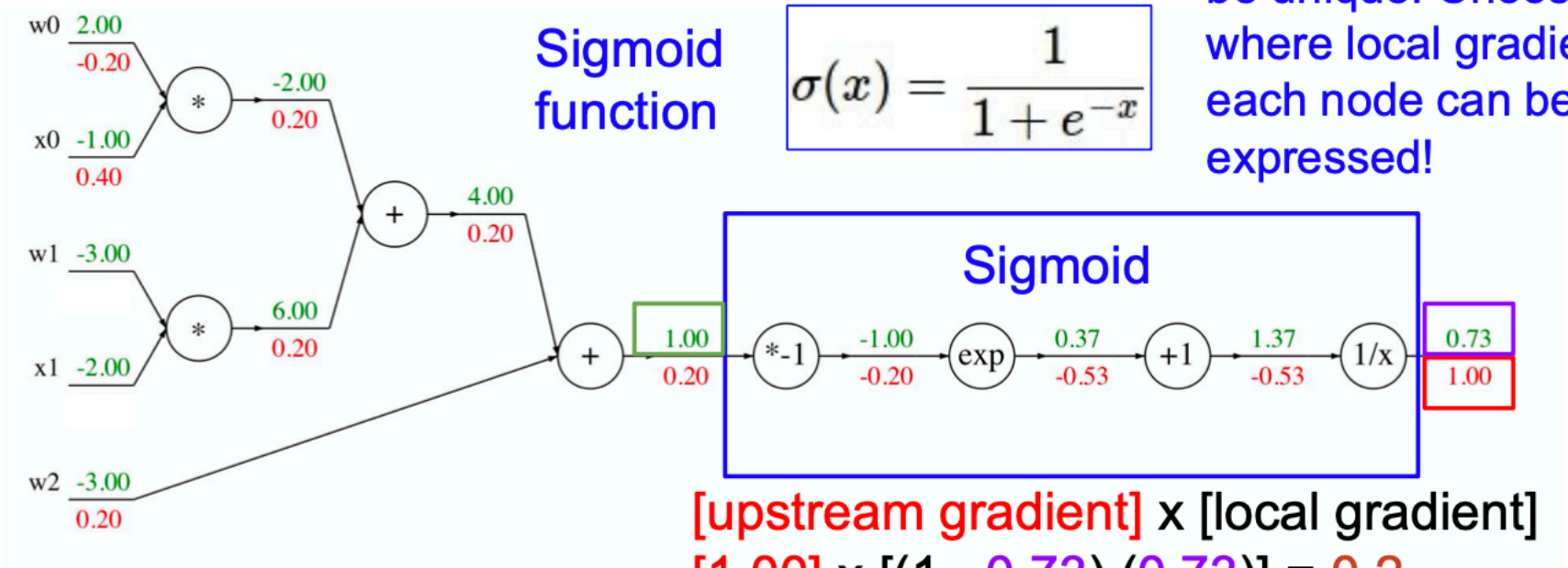
Sigmoid local gradient:

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

Another example:

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

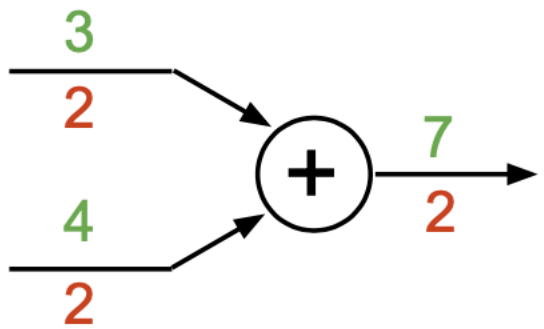
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

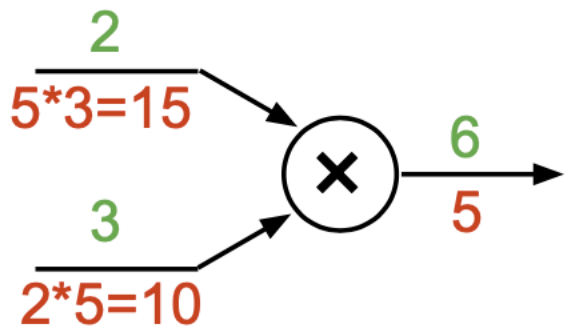
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Patterns in gradient flow

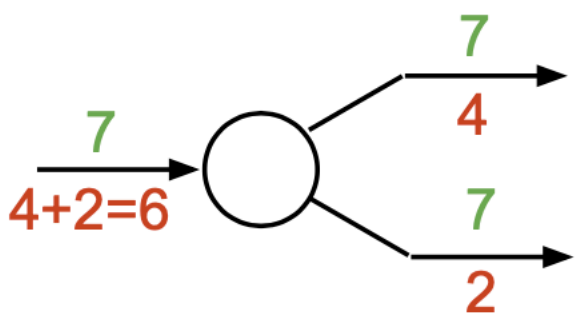
add gate: gradient distributor



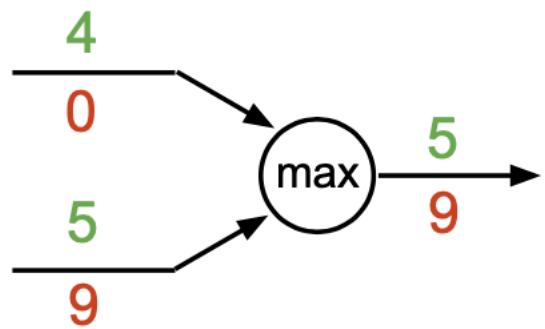
mul gate: “swap multiplier”



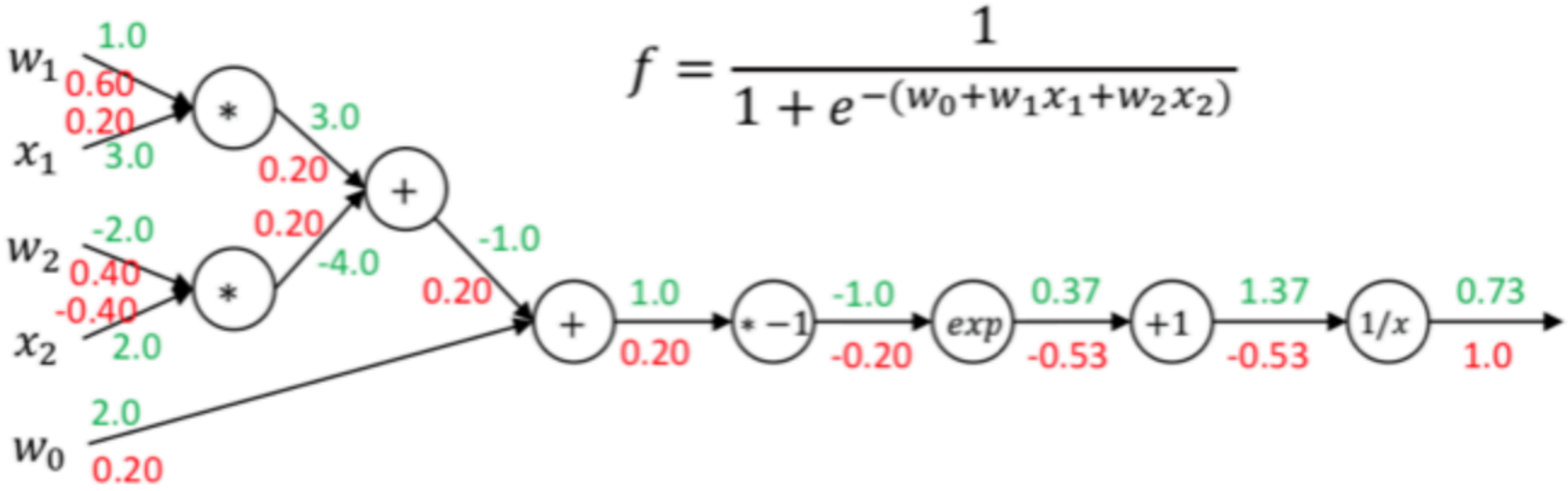
copy gate: gradient adder



max gate: gradient router



Result


w_1
1.0
0.60
x_1
0.20
3.0
*
3.0
0.20
w_2
-2.0
0.40
-0.40
x_2
2.0
*
0.20
-4.0
+
-1.0
0.20
w_0
2.0
0.20
+
1.0
0.20
* −1
-1.0
-0.20
exp
0.37
-0.53
+1
1.37
-0.53
1/x
0.73
1.0
$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$

Recap: Vector derivatives

Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

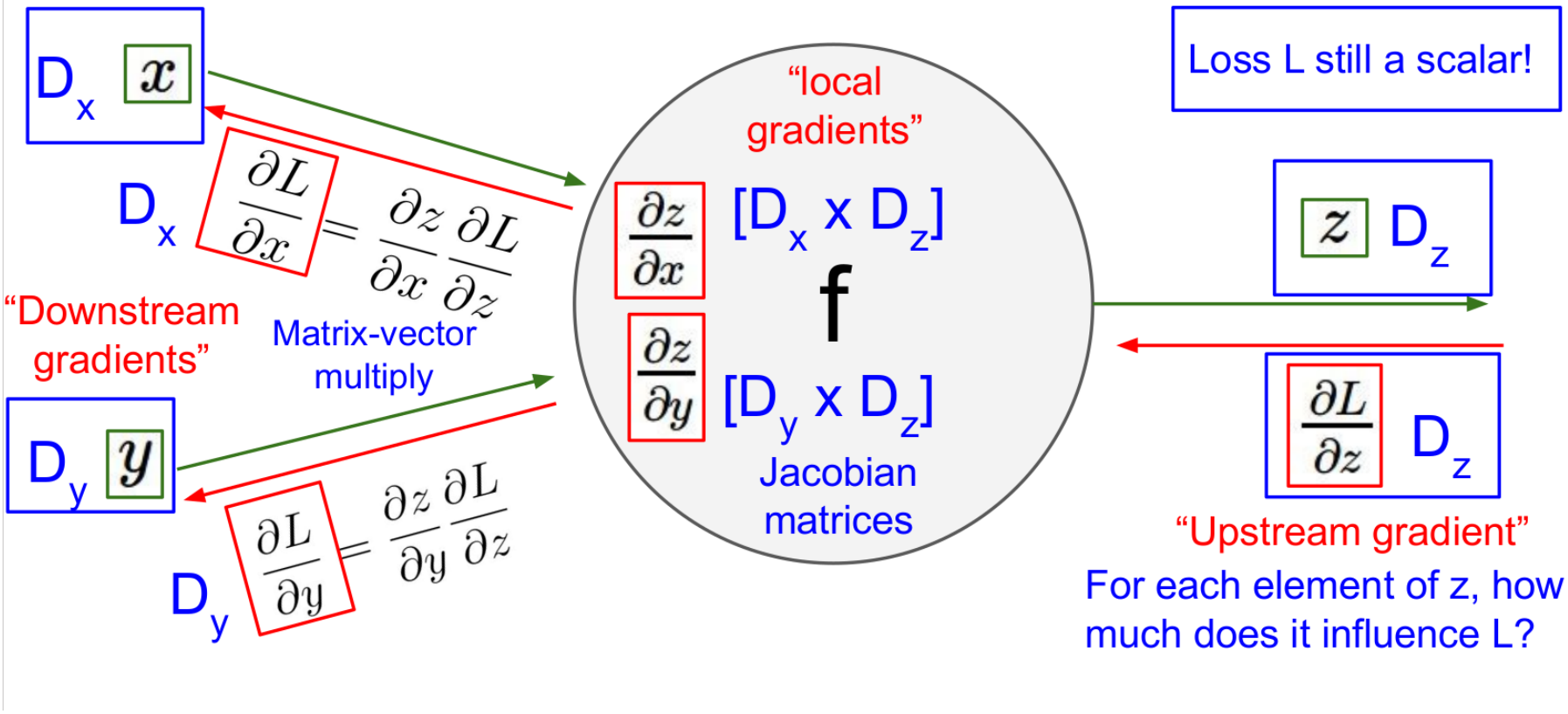
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

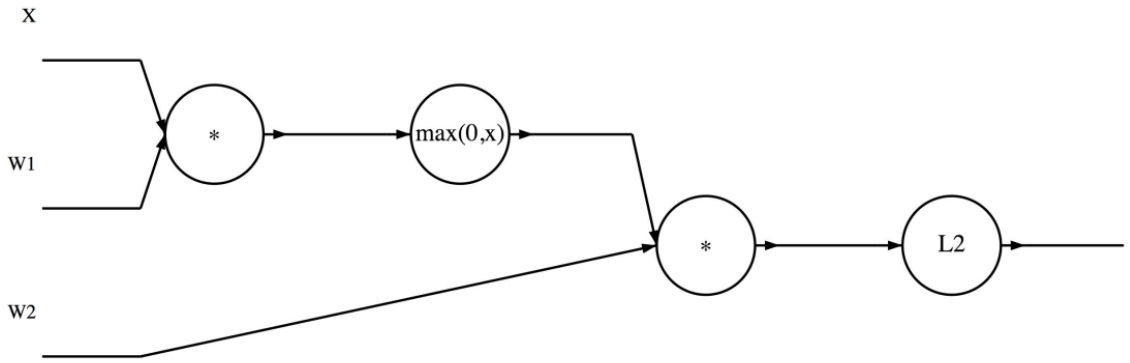
For each element of x, if it changes by a small amount then how much will each element of y change?

Backprop with Vectors



Homework: a matrix example

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$
$$\frac{\partial L}{\partial W_2} = \textbf{?}$$
$$\frac{\partial L}{\partial W_1} = \textbf{?}$$



Multilayer Neural Networks

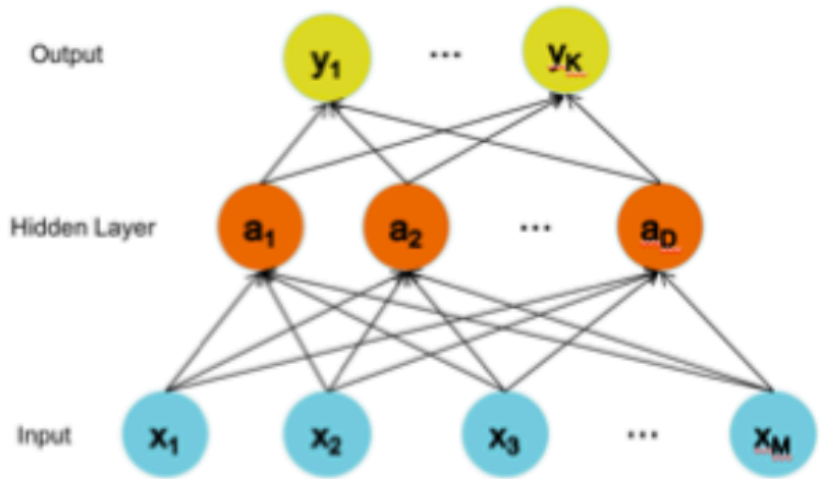
Network Design

Neural Network Architectures

- Even for a basic Multilayer Neural Network, there are many design decisions to make:

1) Number of hidden layers (depth of the network)
2) Number of neurons per hidden layer (width)
3) Type of activation functions
4) Form of objective functions

Example: Neural Network with sigmoid activation functions



(F) Loss
$J = \frac{1}{2}(y - y^*)^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

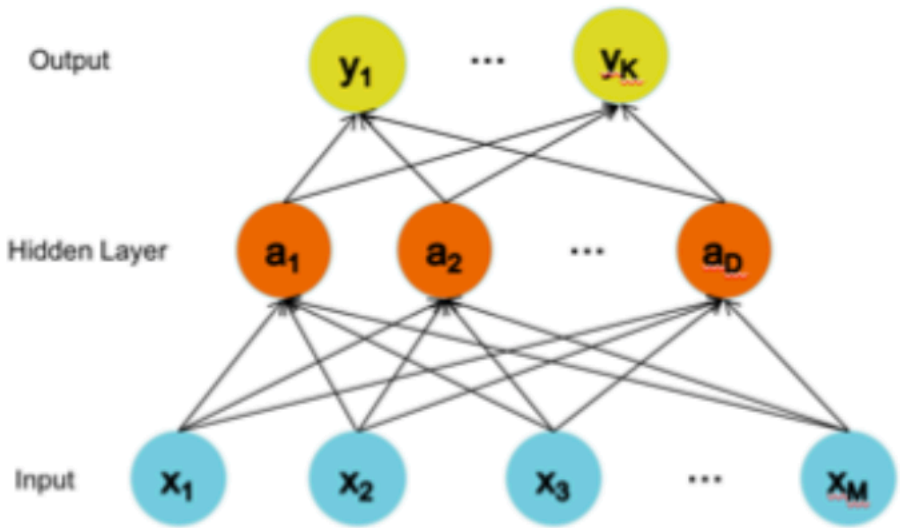
(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Neural Network with arbitrary nonlinear activation functions



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (nonlinear)**
$y = \sigma(b)$

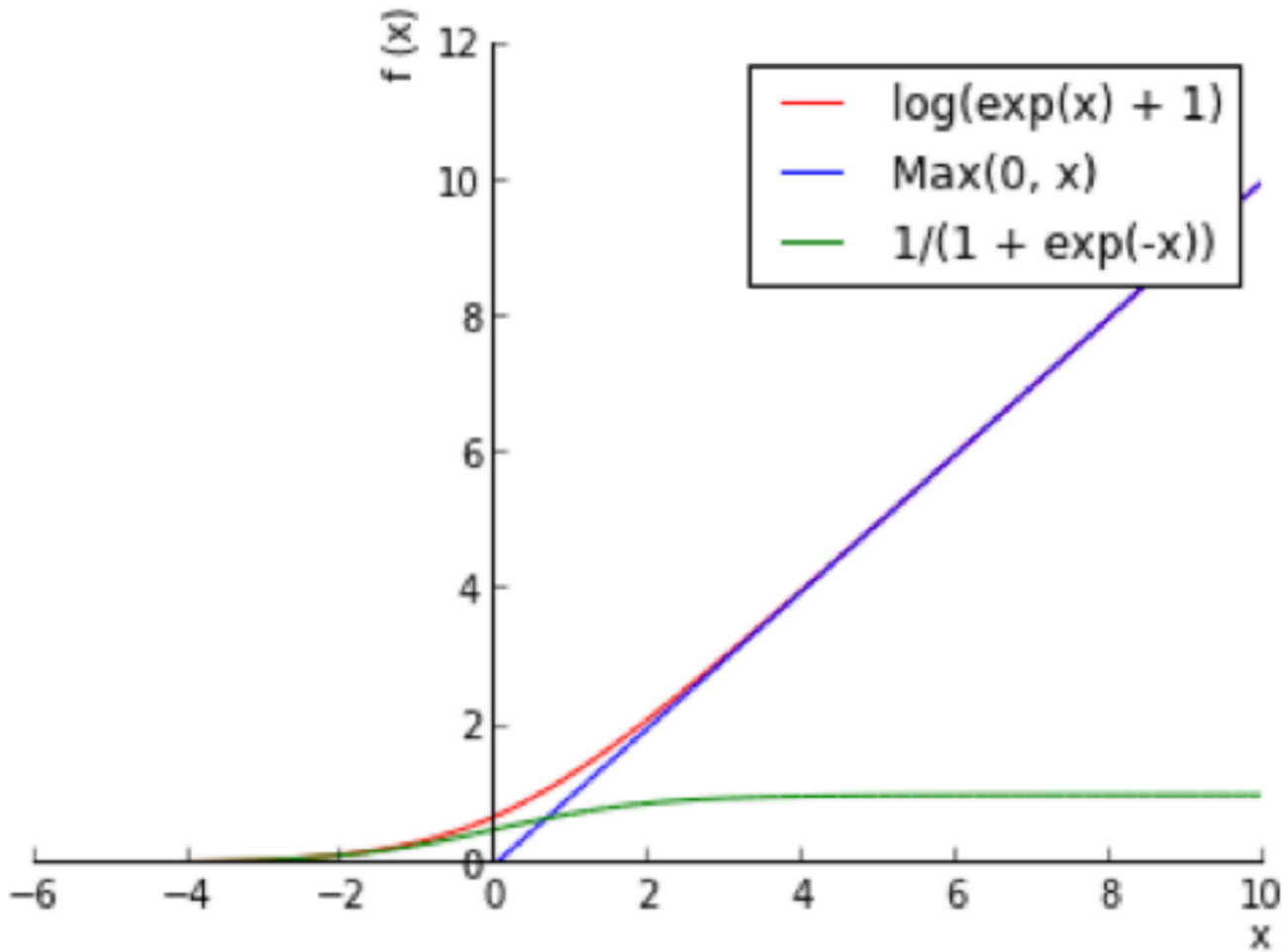
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

ReLU often used in computer vision tasks



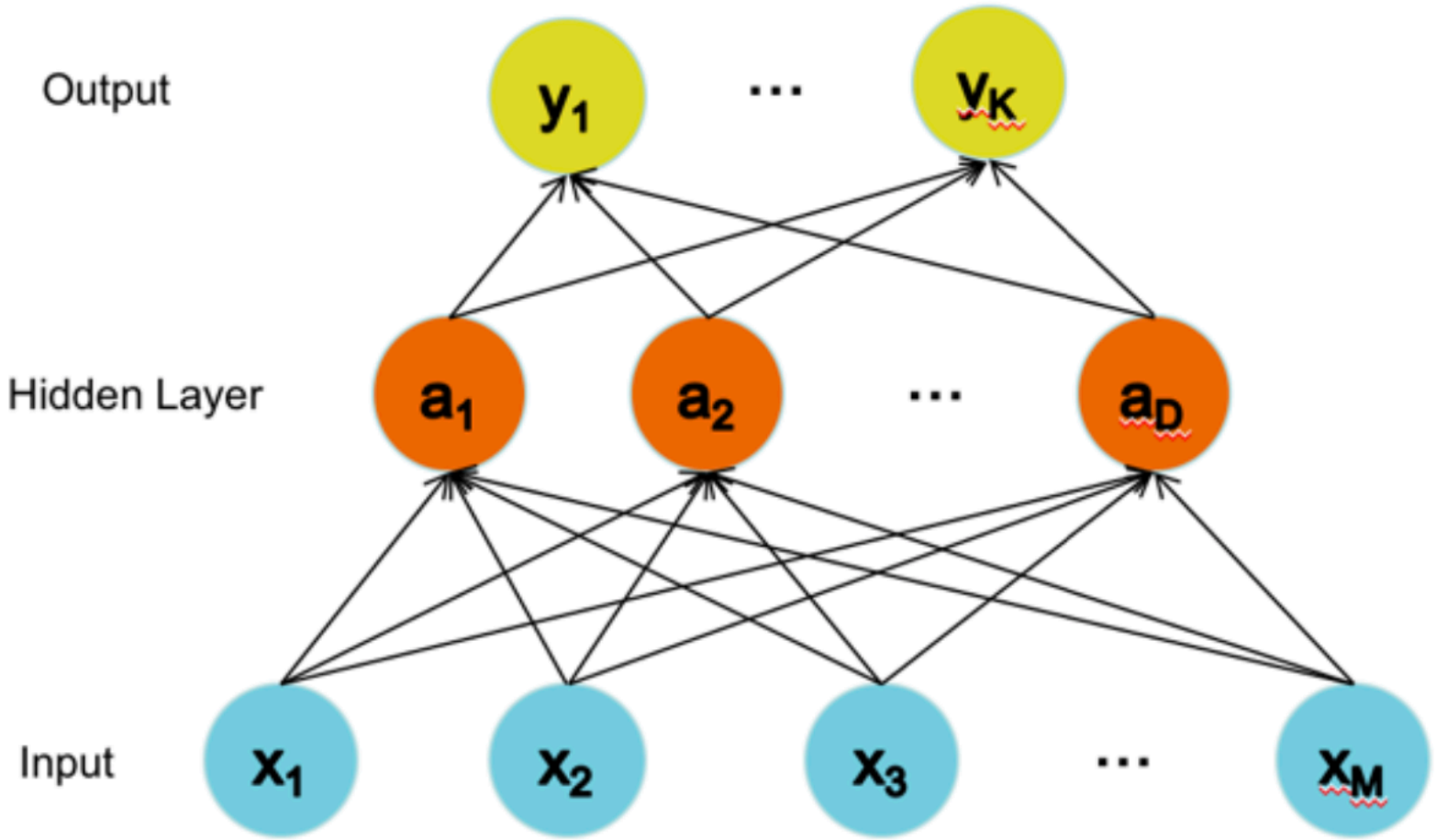
Loss Functions for NN

- **Regression:**
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)

- **Classification:**
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional "softmax" layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

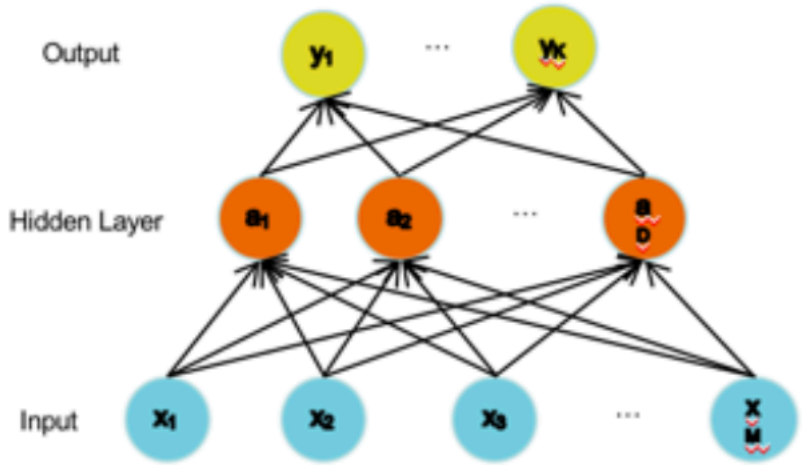
Multi-class Output

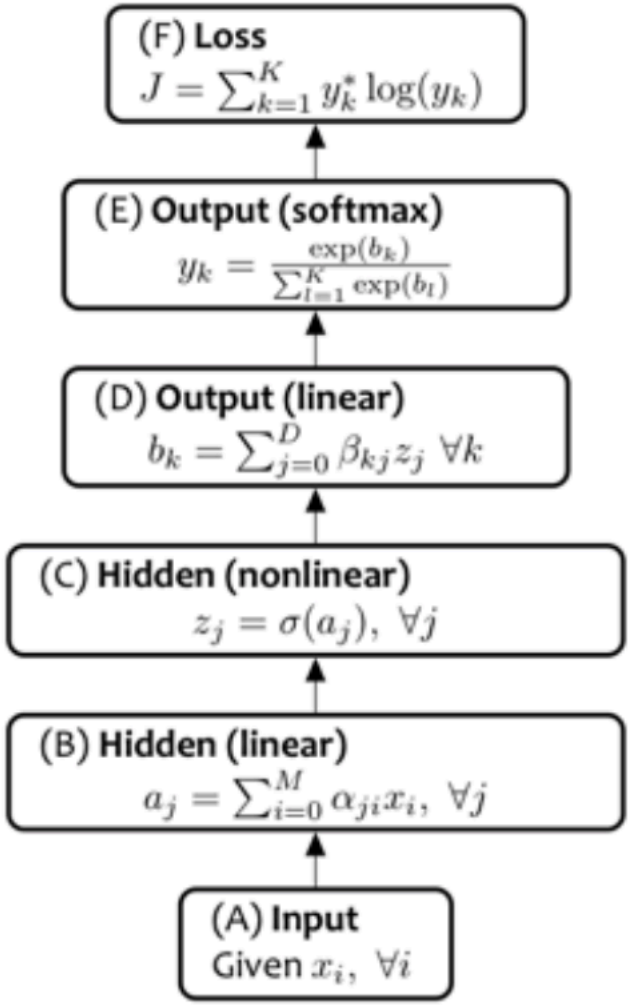


Multi-class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$



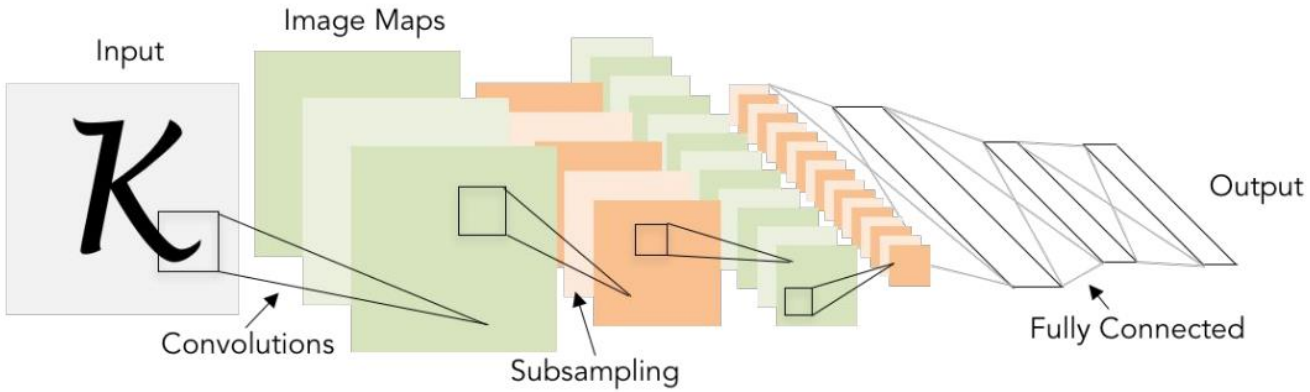


Convolutional Neural Networks (CNNs)

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

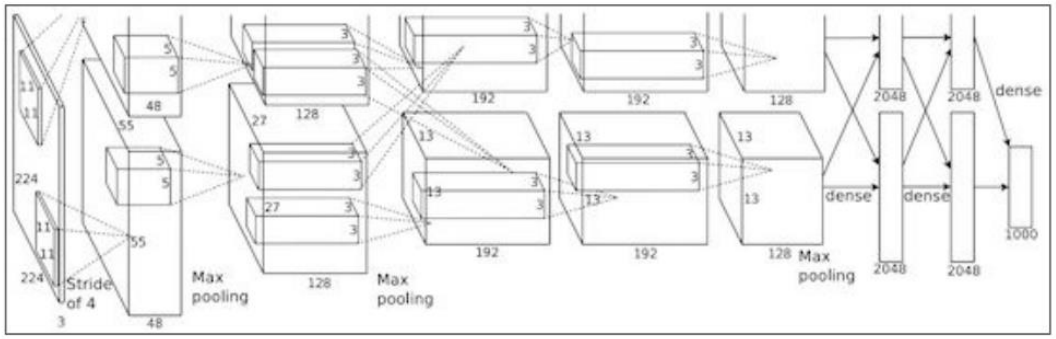


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

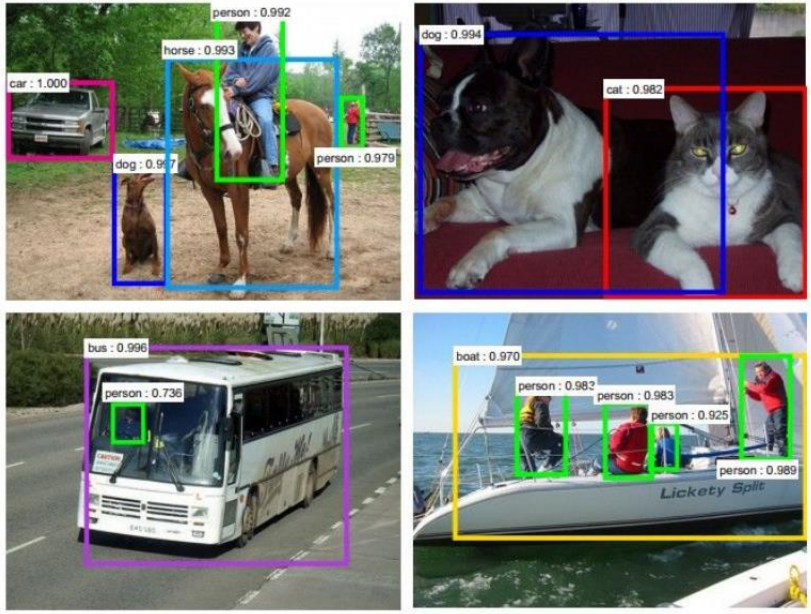
Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

Fast-forward to today: ConvNets are everywhere

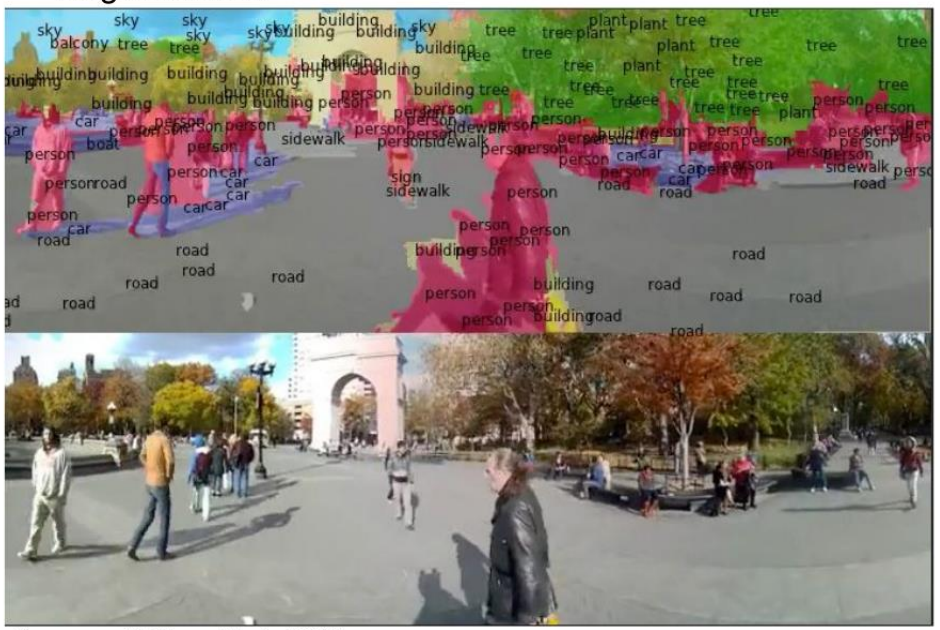
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



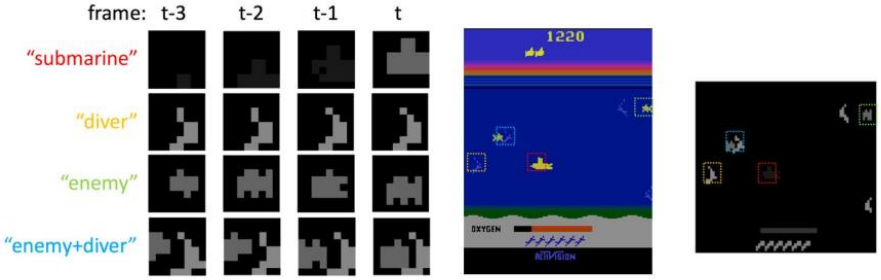
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

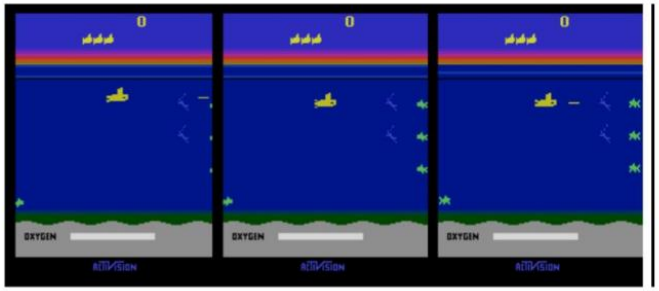
Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Guo et al. 2014]



Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

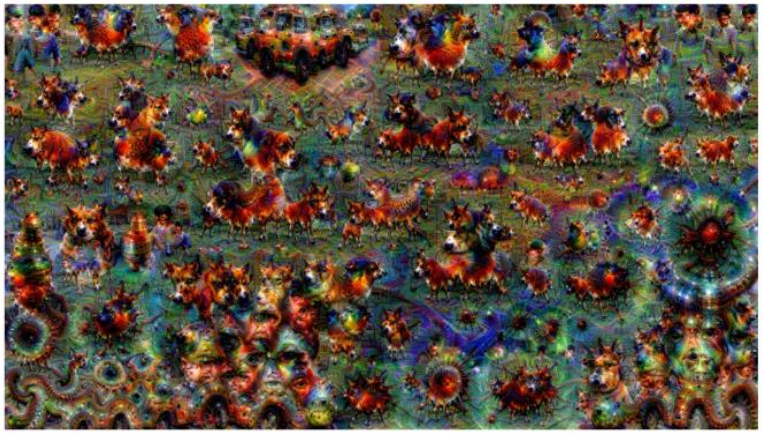
A cat sitting on a suitcase on the floor

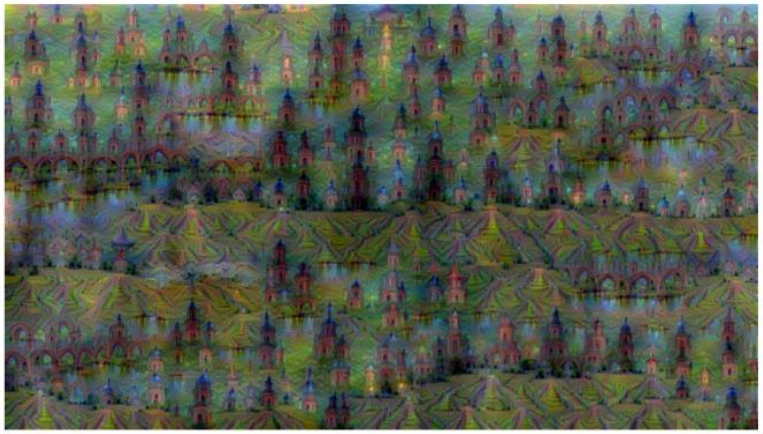
Somewhat related

A woman is holding a cat in her hand

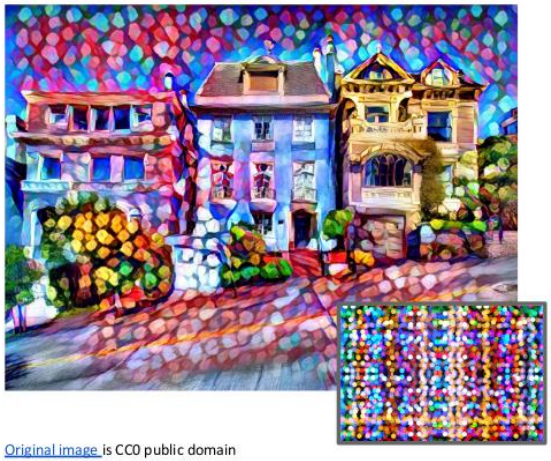
A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2





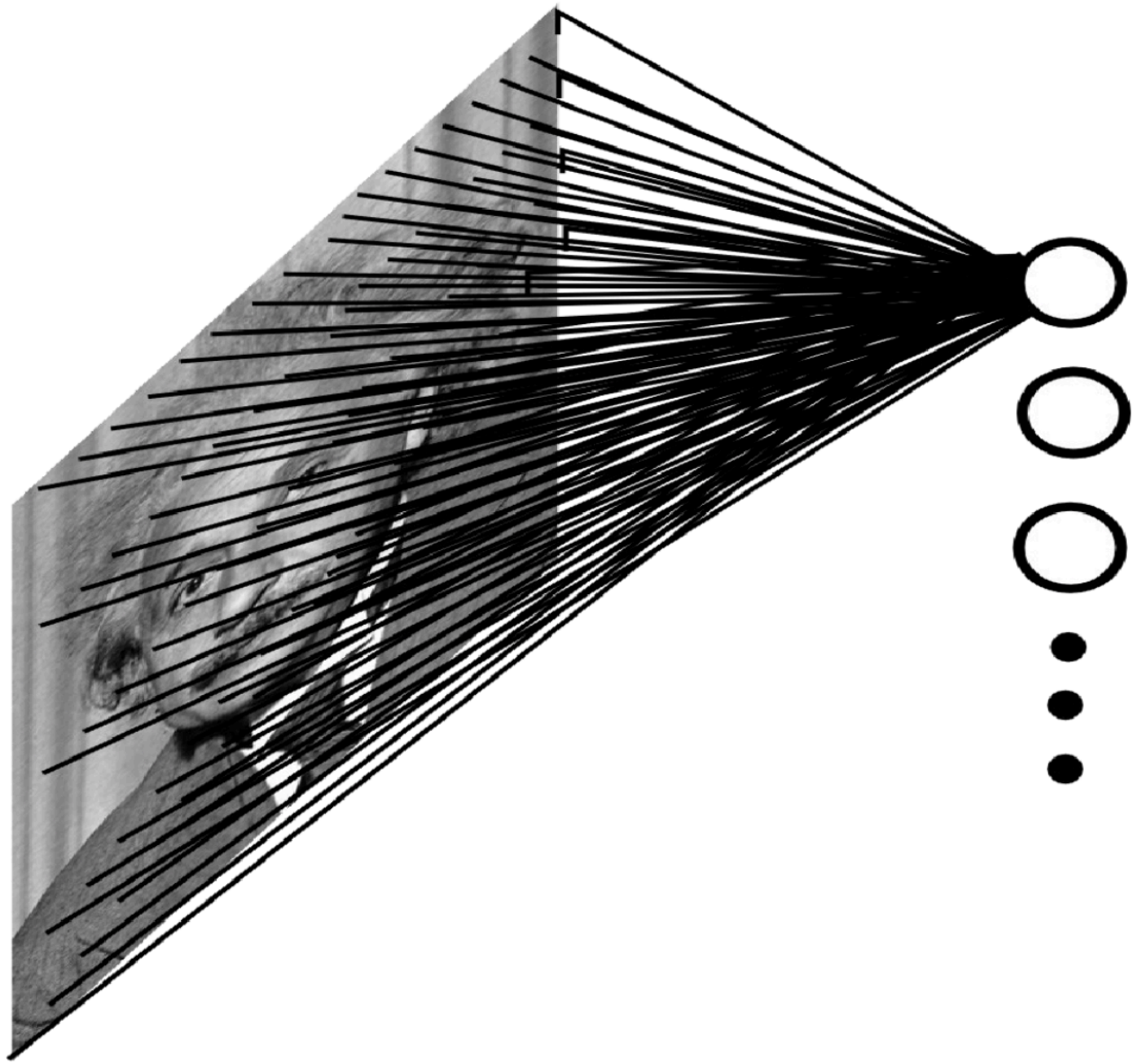
Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



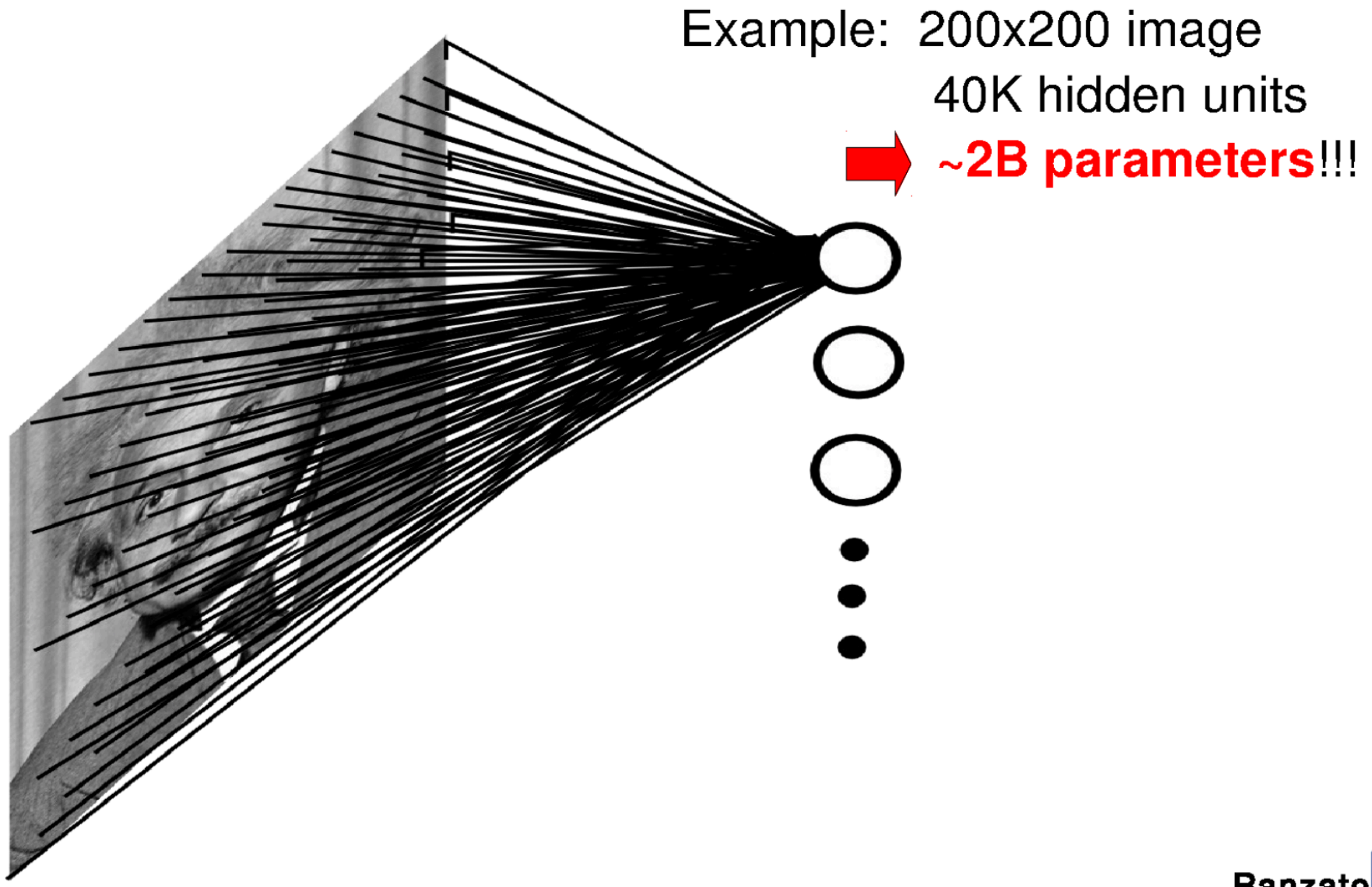
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

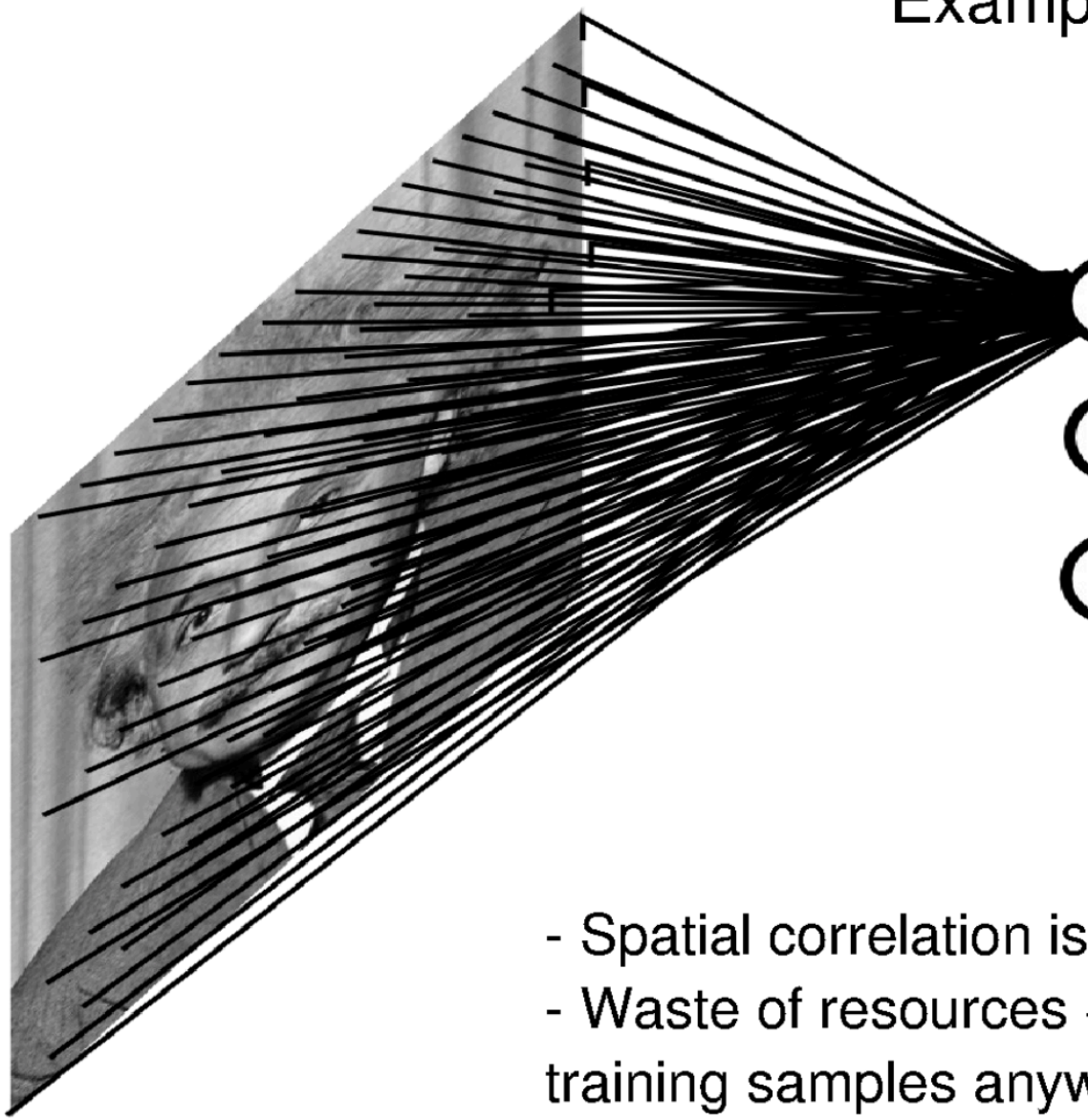
Images as input to neural networks



Images as input to neural networks



Images as input to neural networks



Example: 200x200 image

40K hidden units

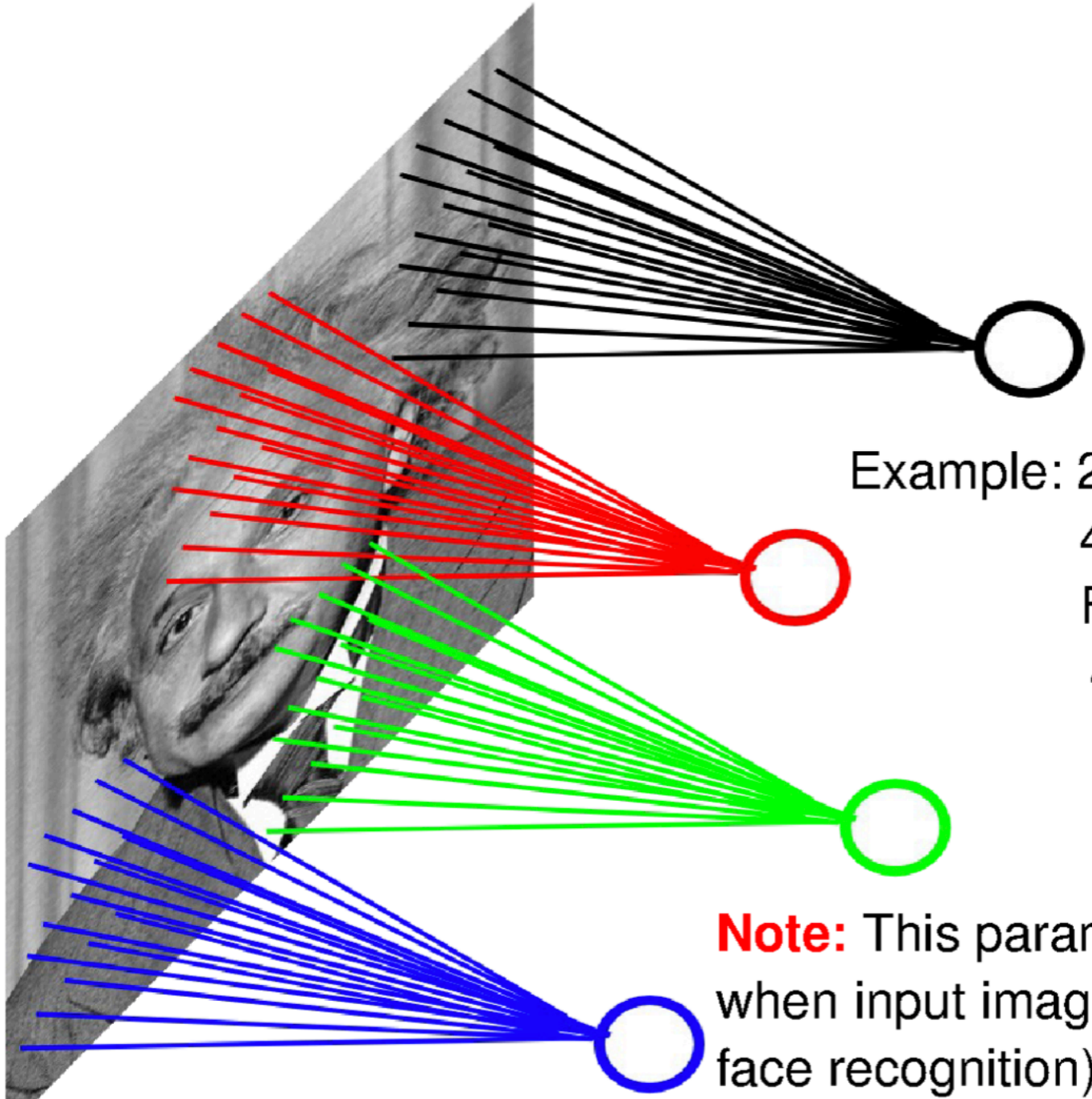
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Ranzato

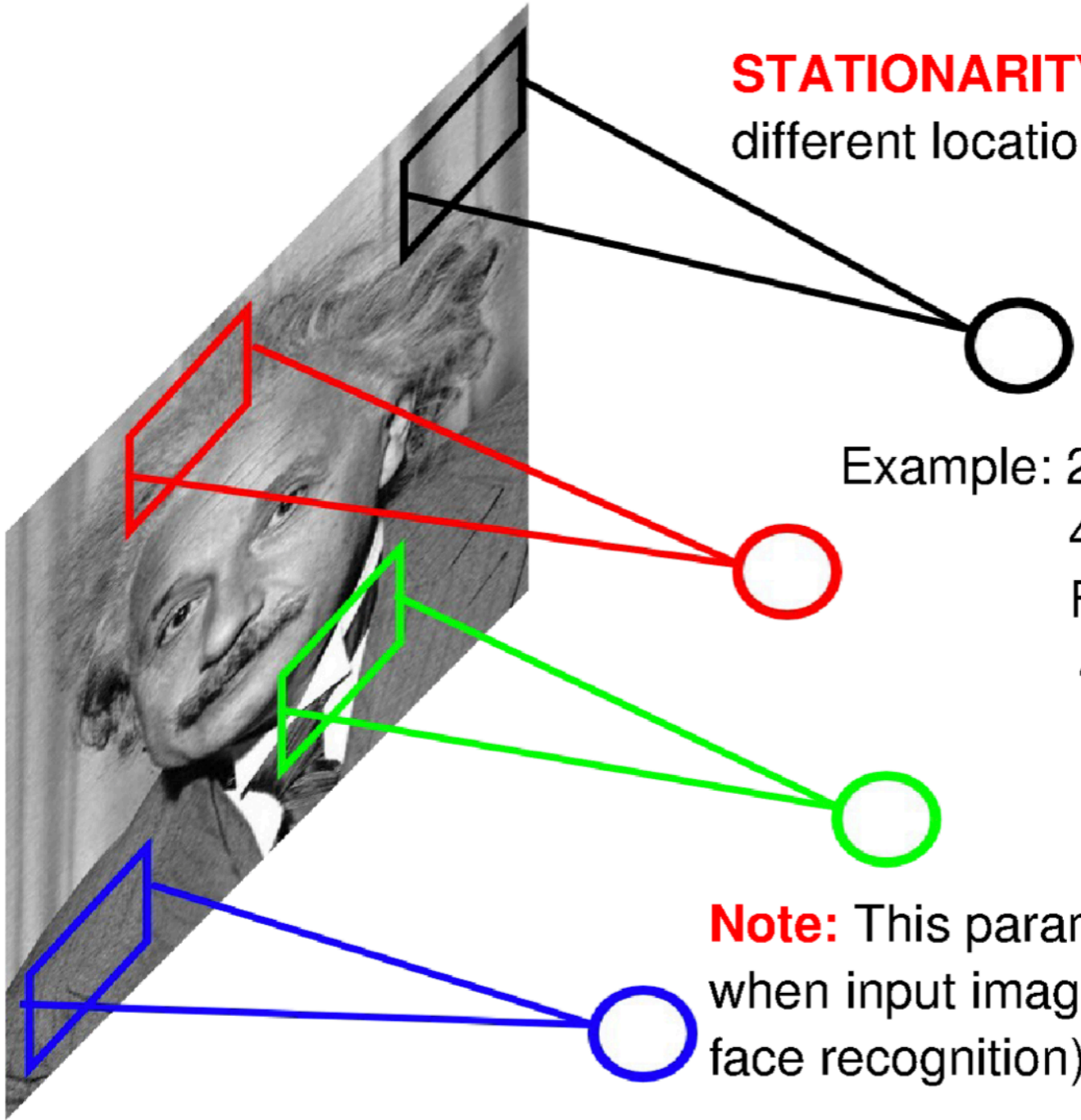
Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local -> global.



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).



STATIONARITY? Statistics is similar at different locations

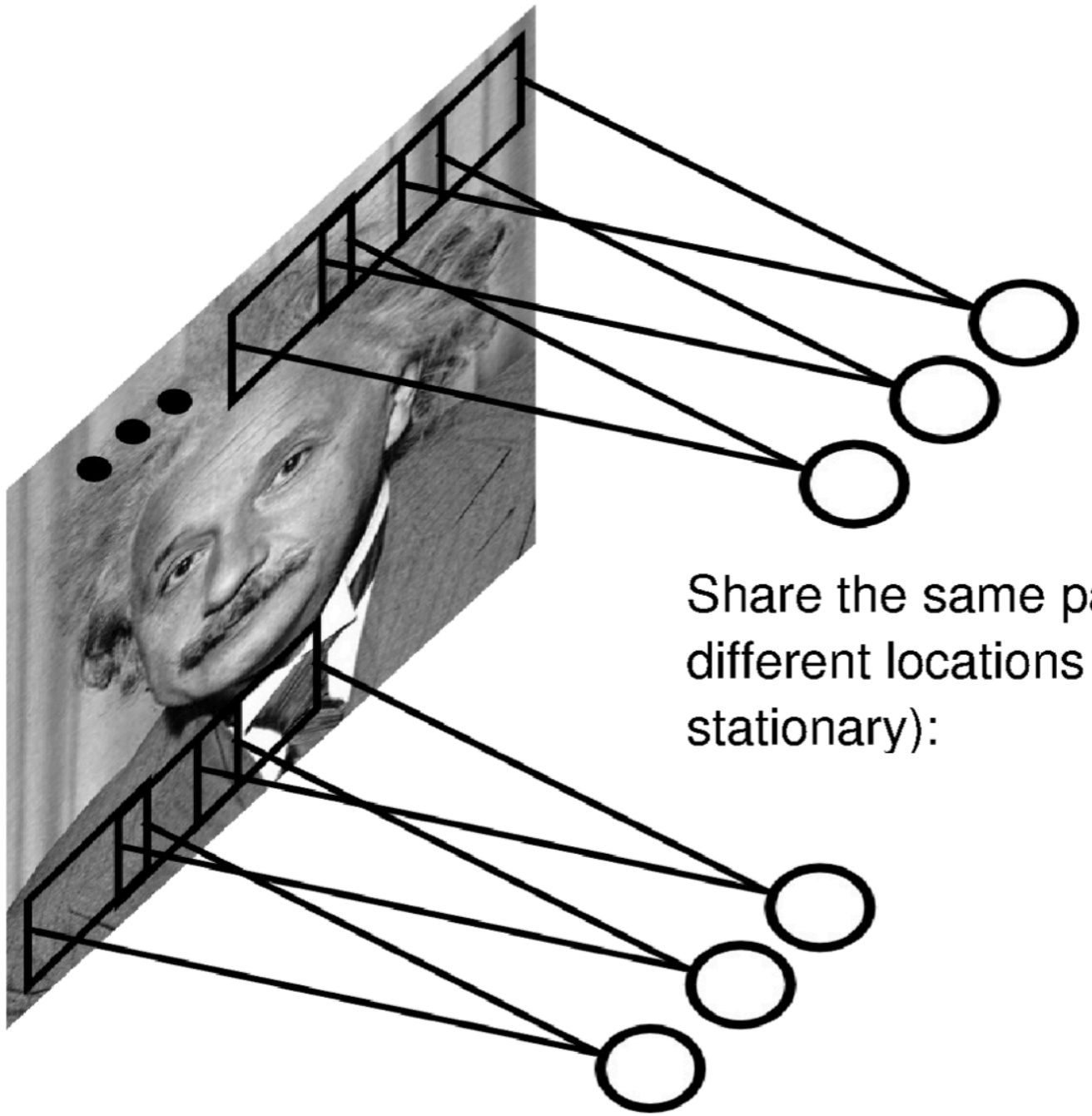
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about 'local neighborhoods' only for a given neural network layer.
  - Composition of layers will expand local -> global.

- Parameter sharing
  - 'Tied weights' – use same weights for more than one perceptron in the neural network.
  - Leads to *equivariant representation*
    - If input changes (e.g., translates), then output changes similarly



Share the same parameters across different locations (assuming input is stationary):

Filtering reminder:
Correlation (rotated convolution)

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

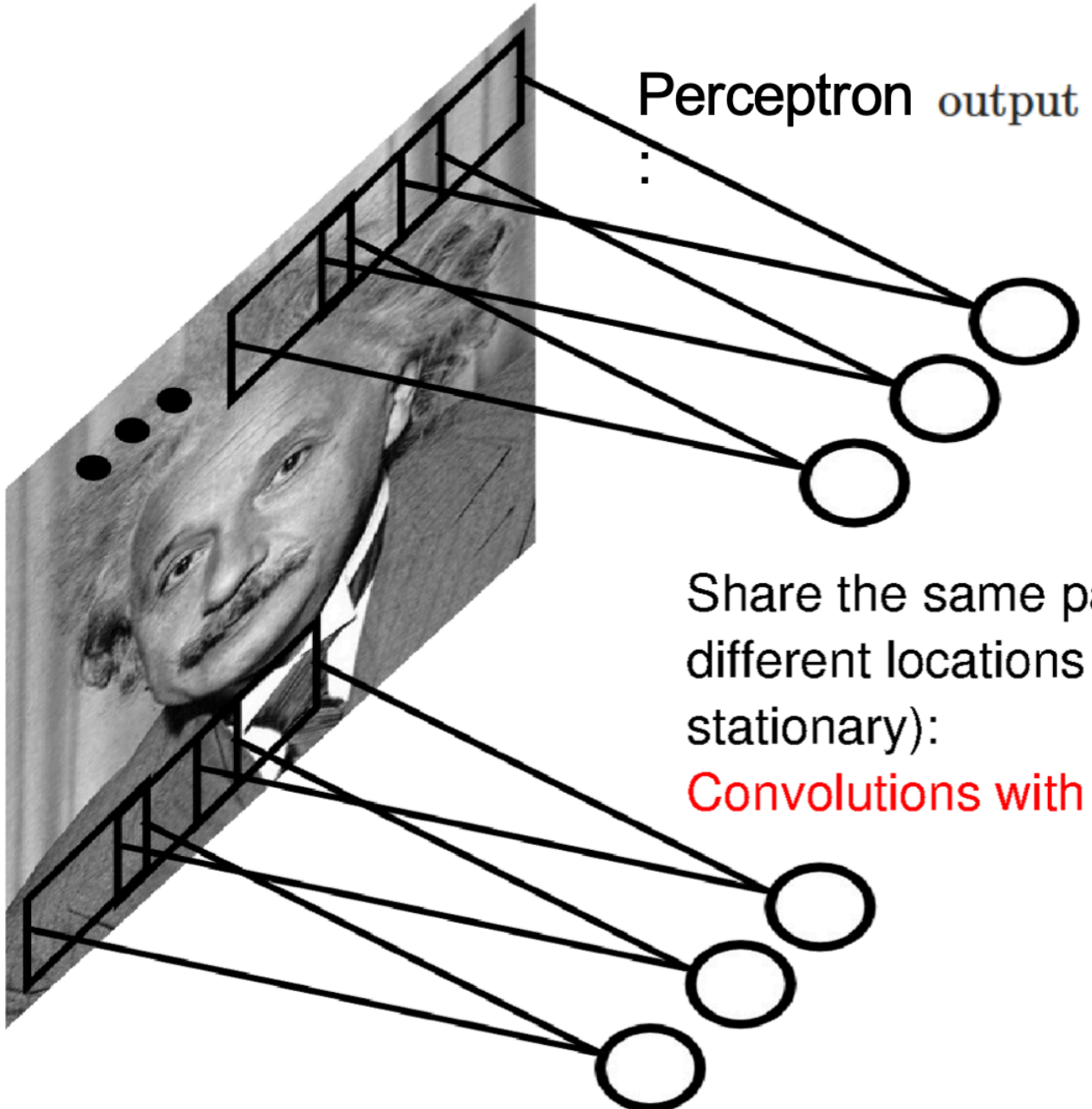
$h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

Convolutional Layer



Perceptron

:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

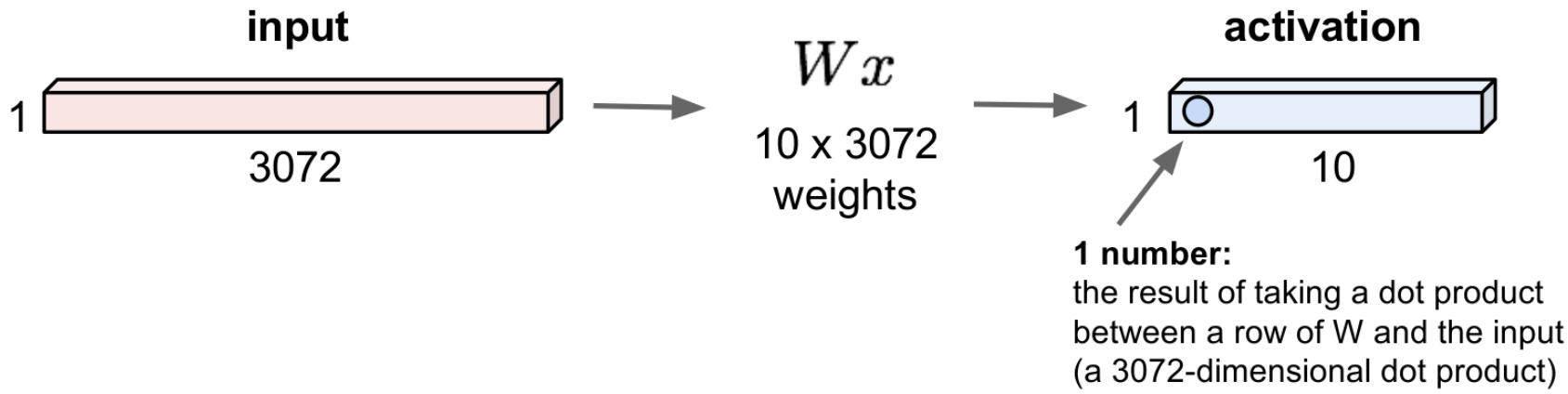
This is convolution!

Share the same parameters across different locations (assuming input is stationary):

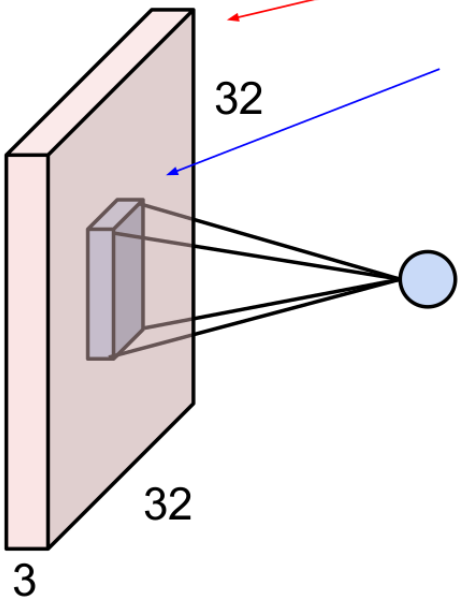
Convolutions with learned kernels

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Convolution Layer



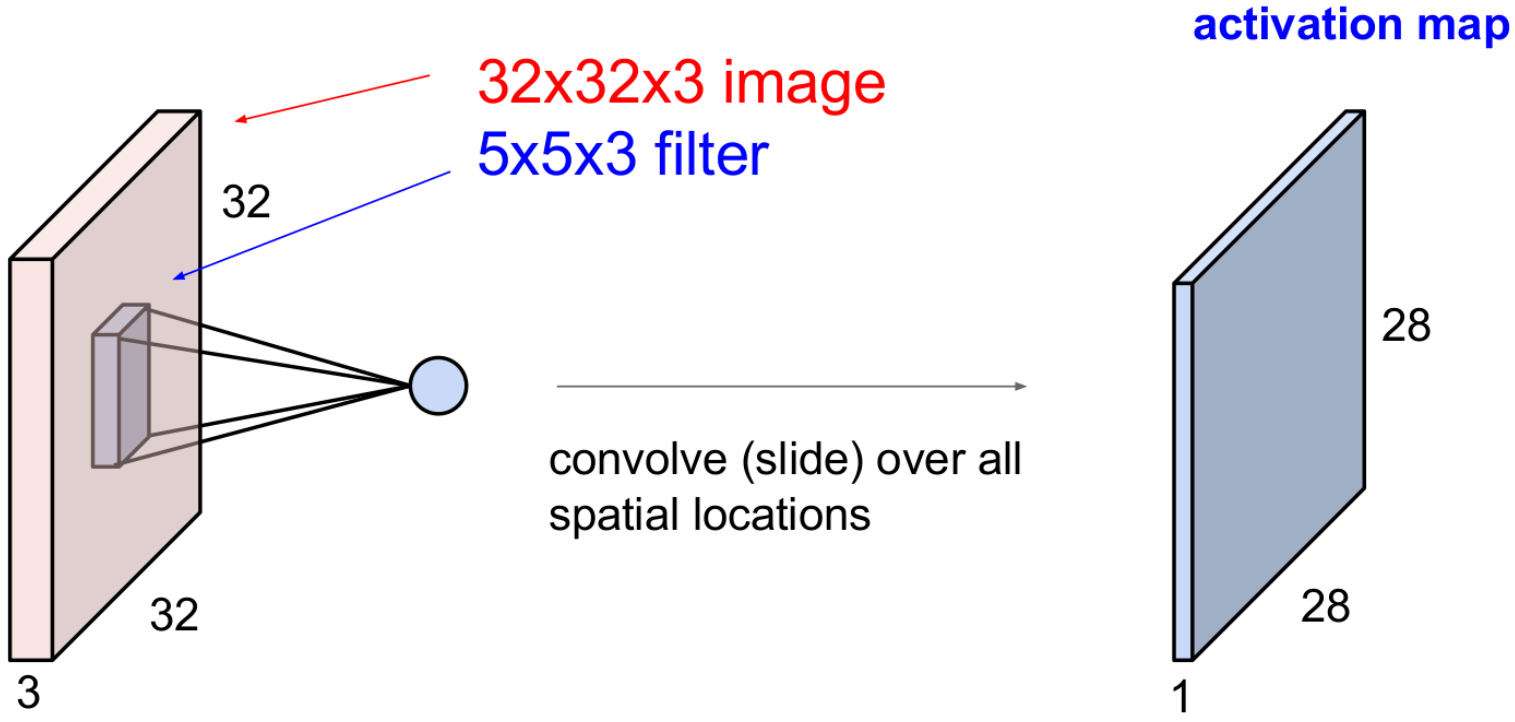
32x32x3 image

5x5x3 filter w

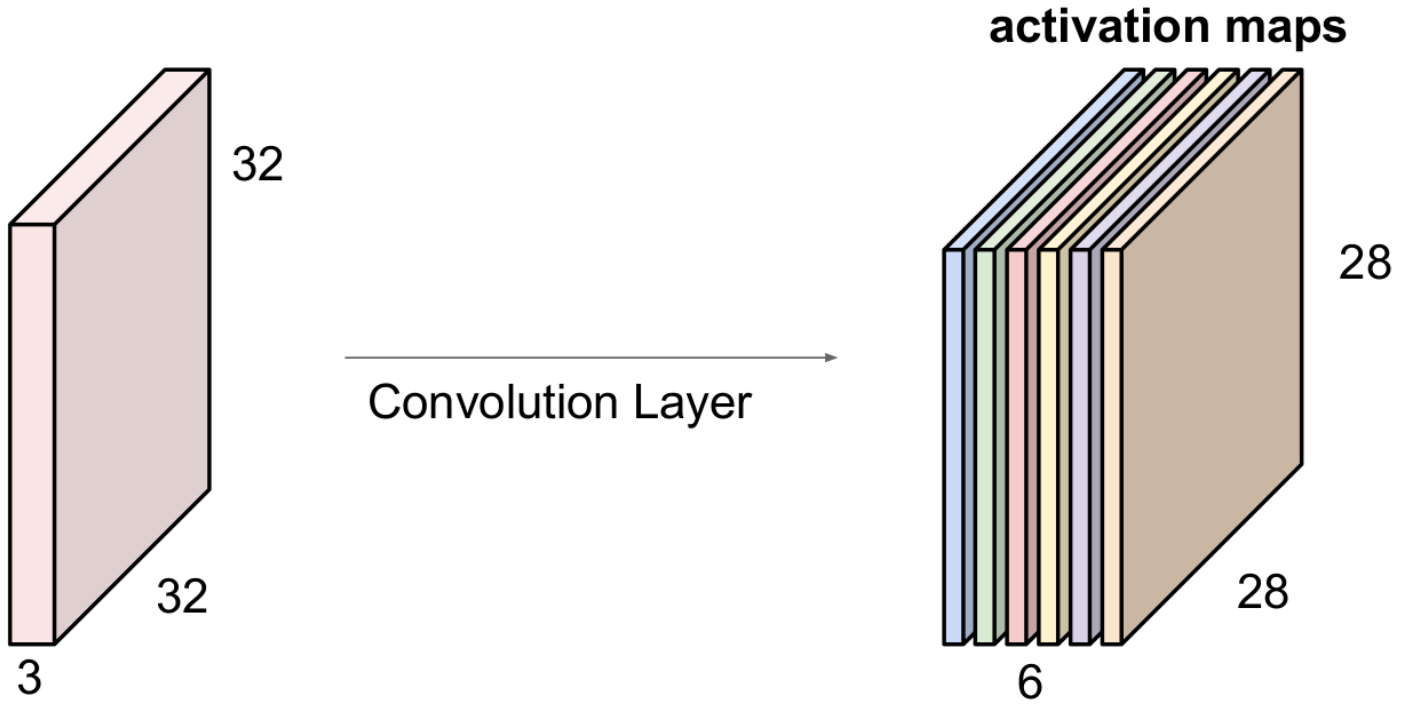
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolutional Layer

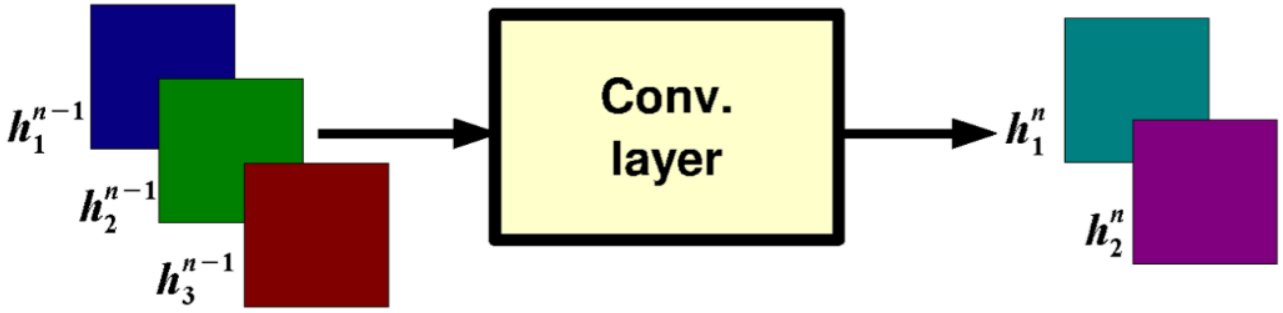
$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

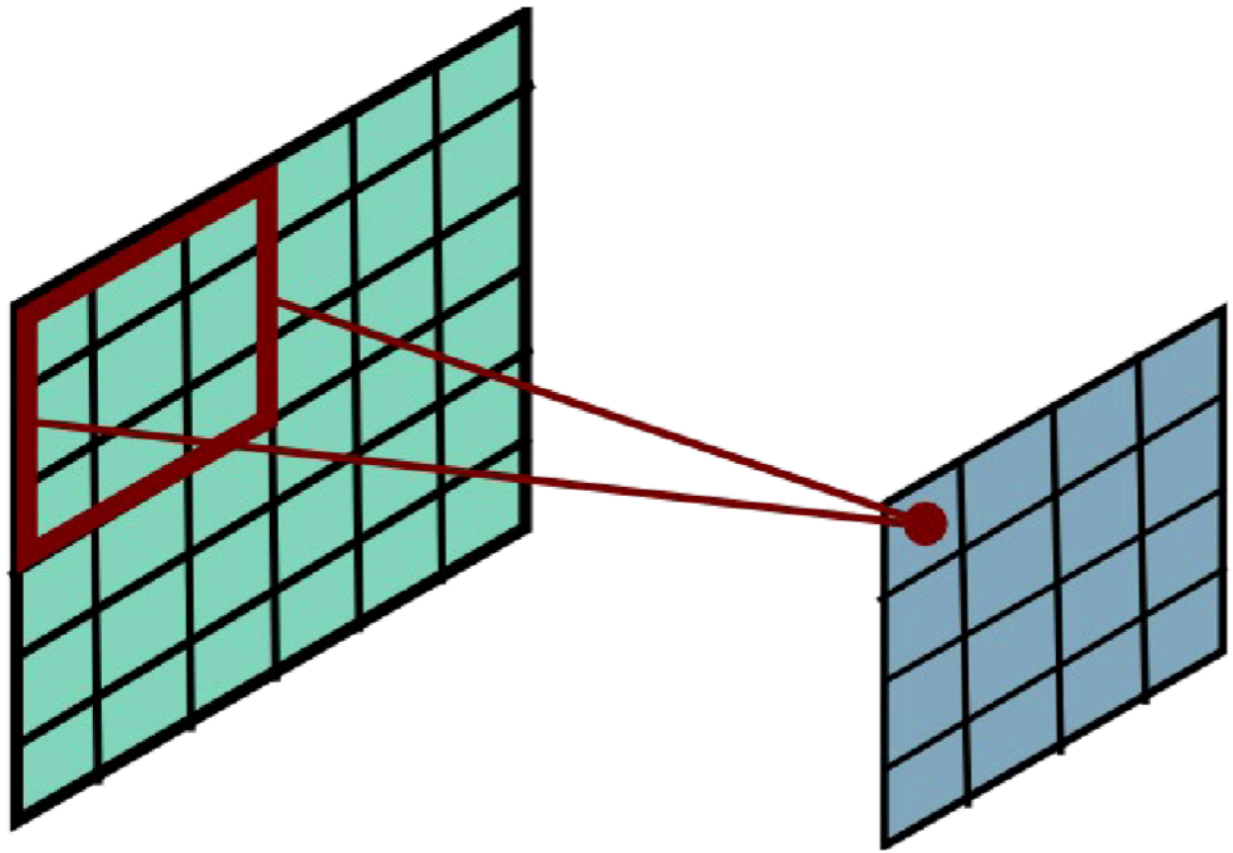
kernel

n = layer number
K = kernel size
j = # channels (input)
or # filters (depth)

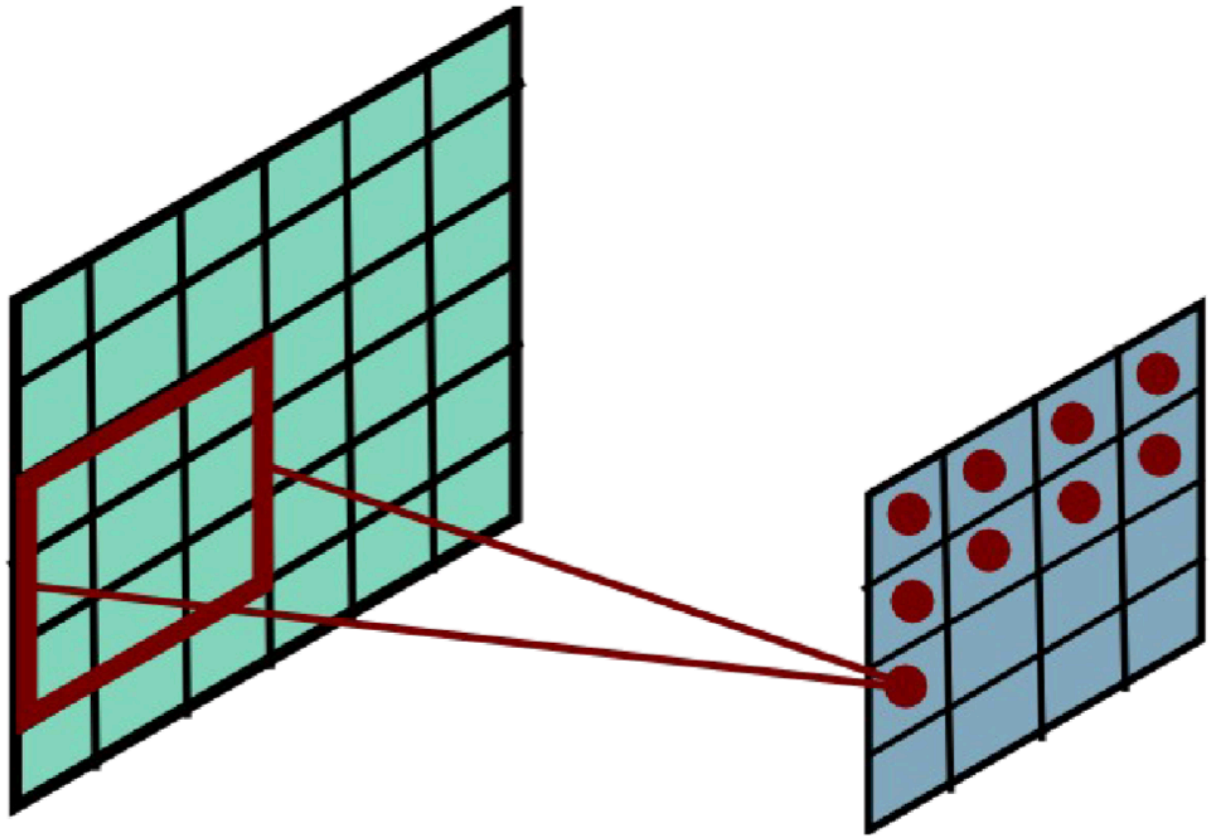


Ranzato

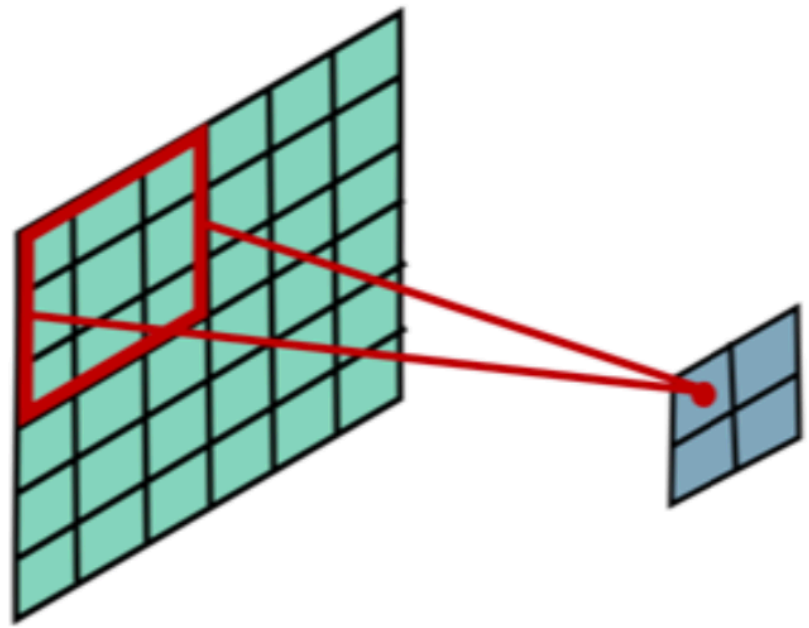
Stride = 1



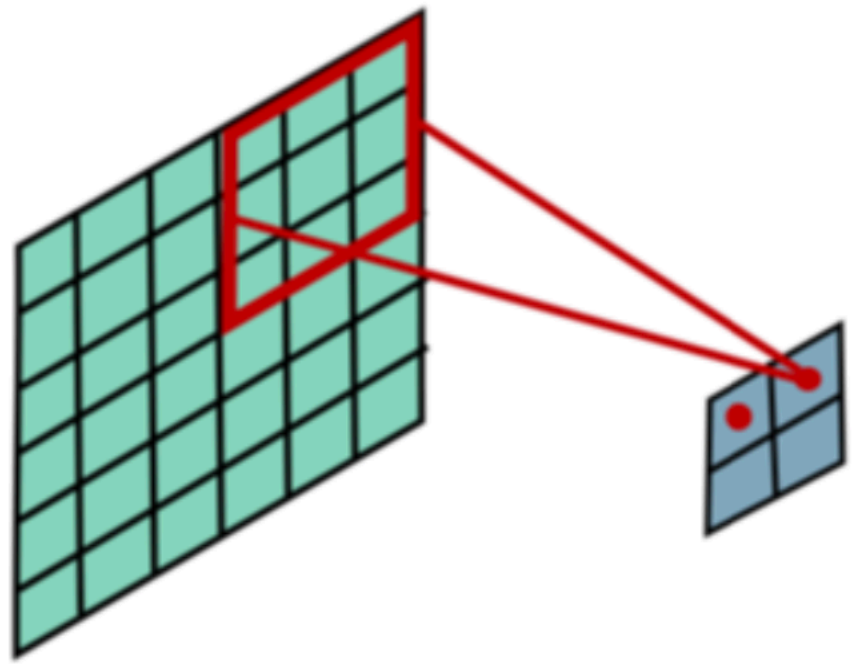
Stride = 1



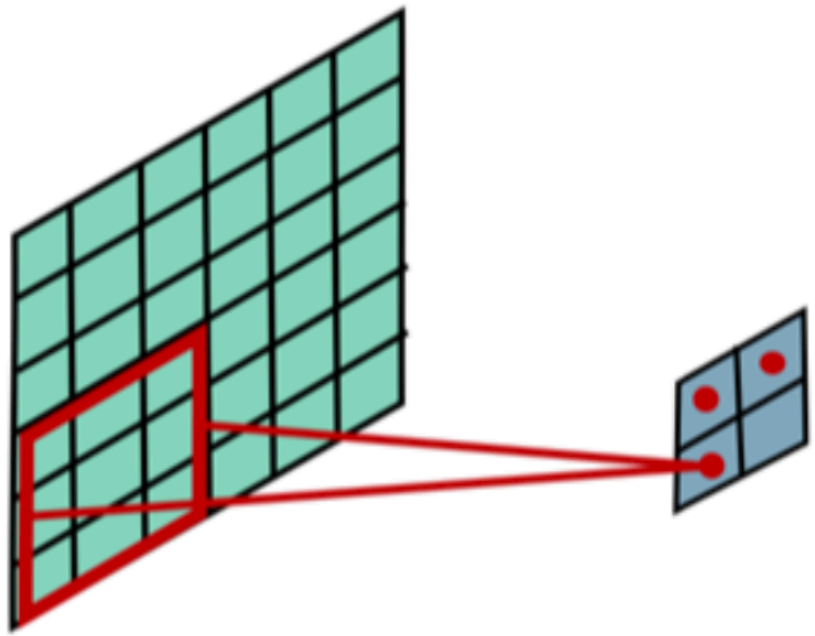
Stride = 3



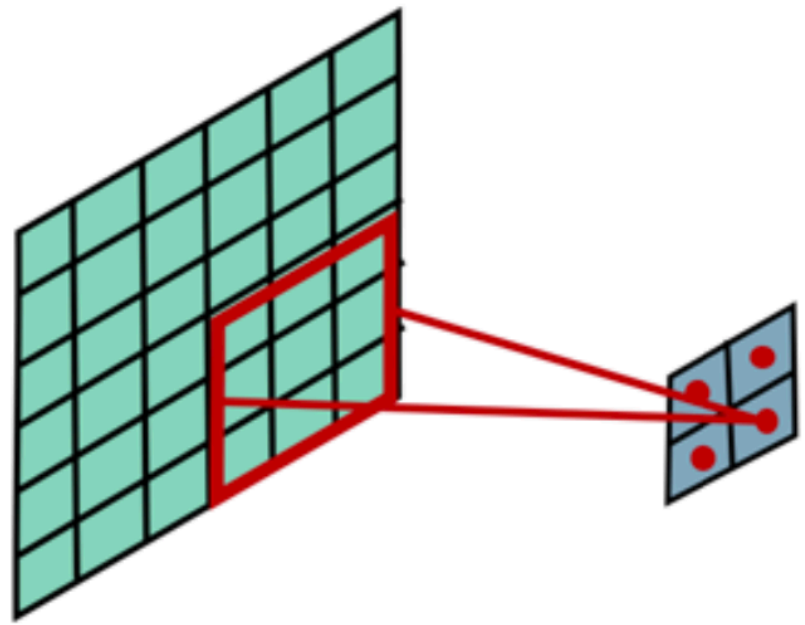
Stride = 3



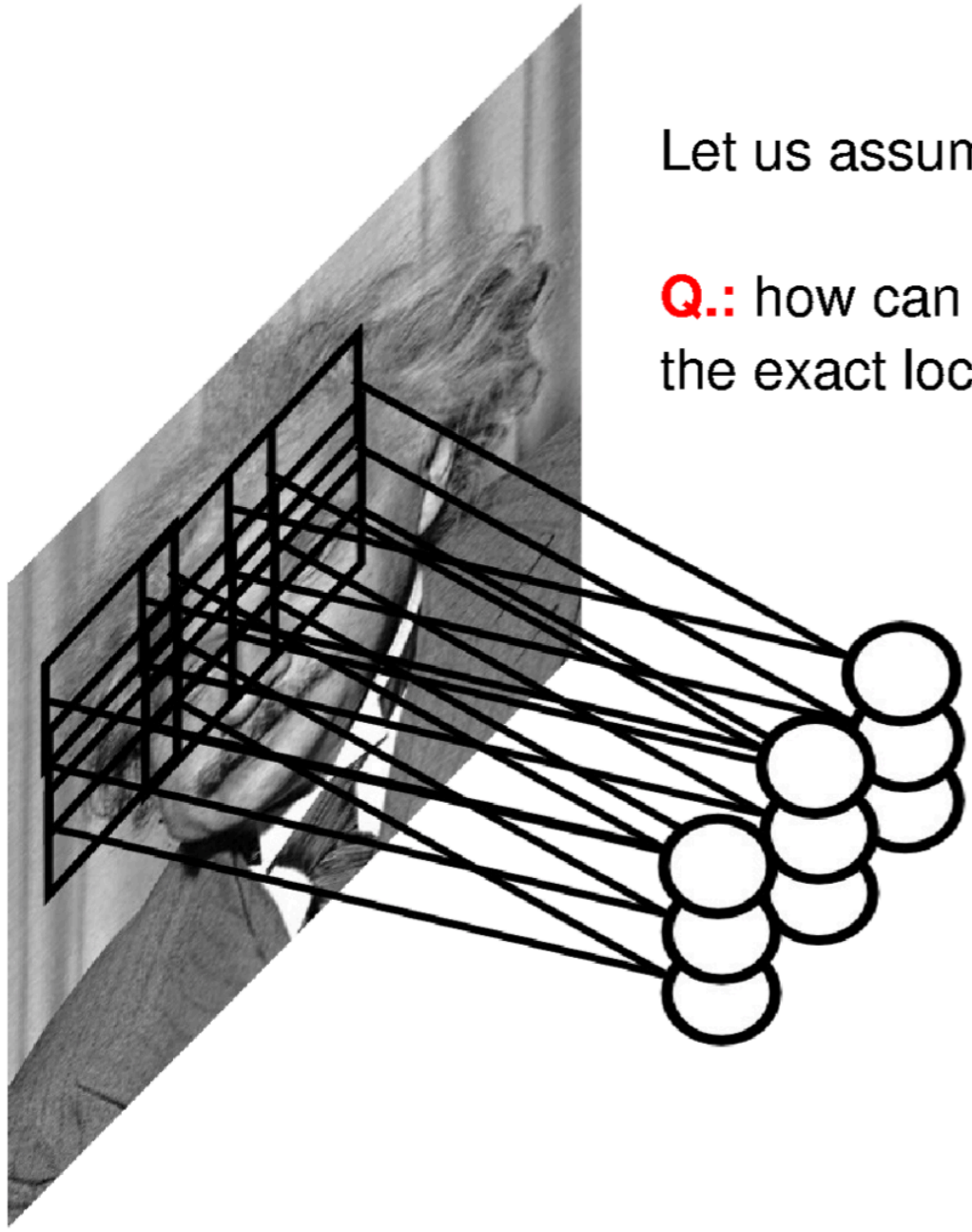
Stride = 3



Stride = 3



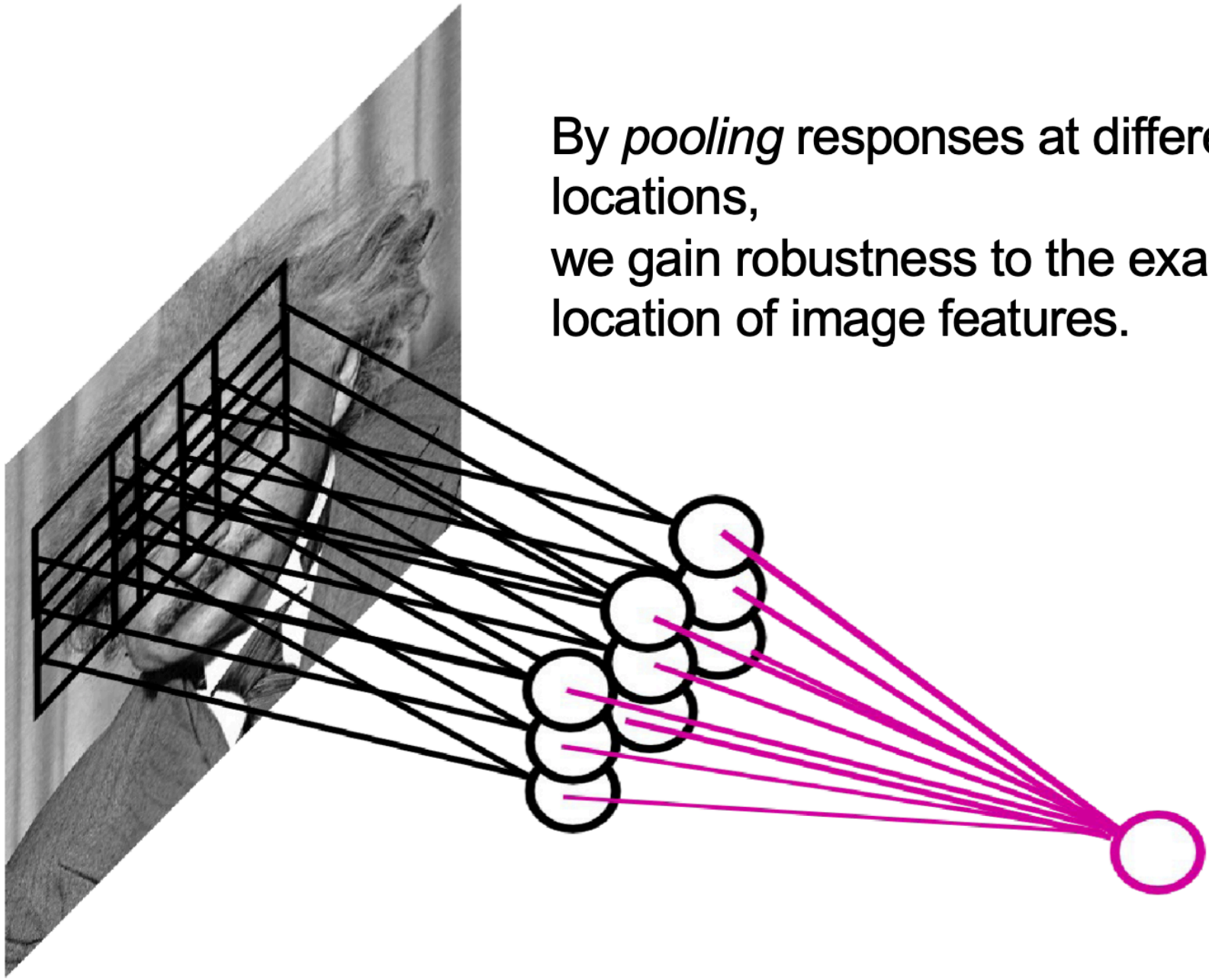
Pooling Layer



Let us assume filter is an “eye” detector.

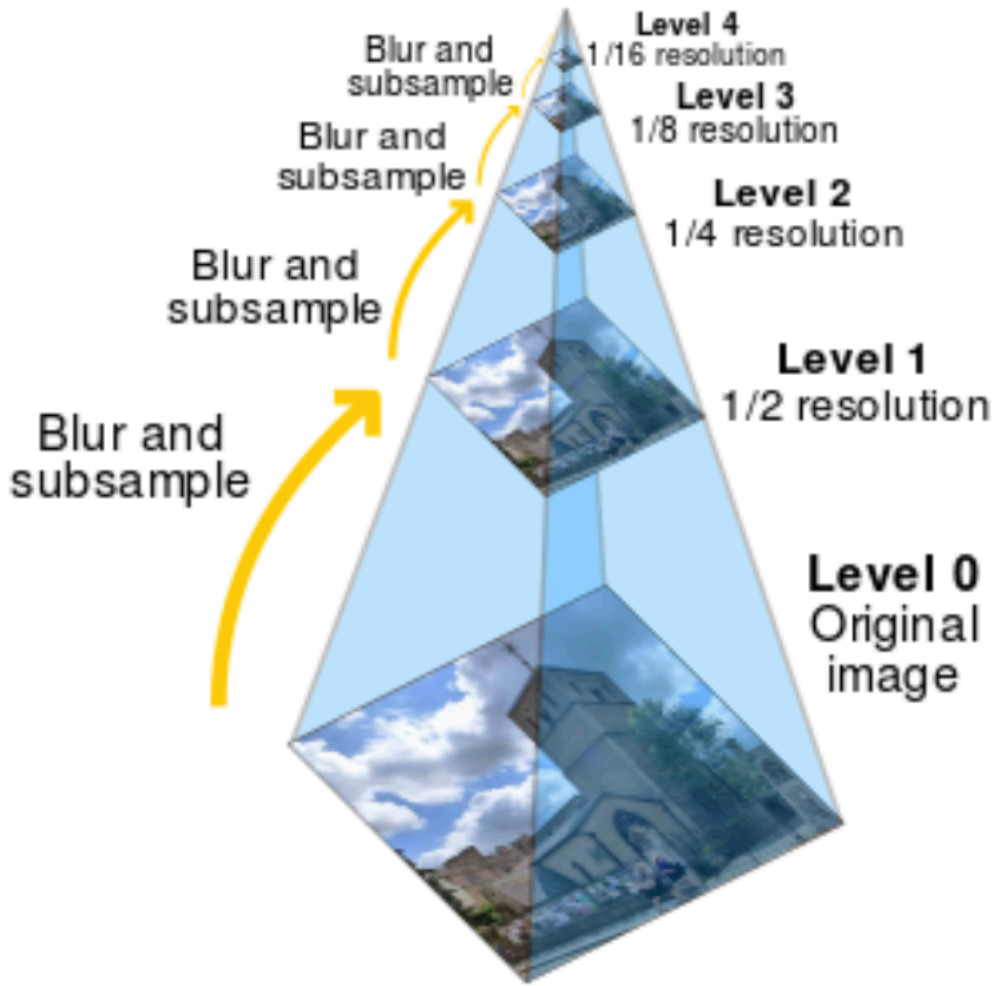
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer

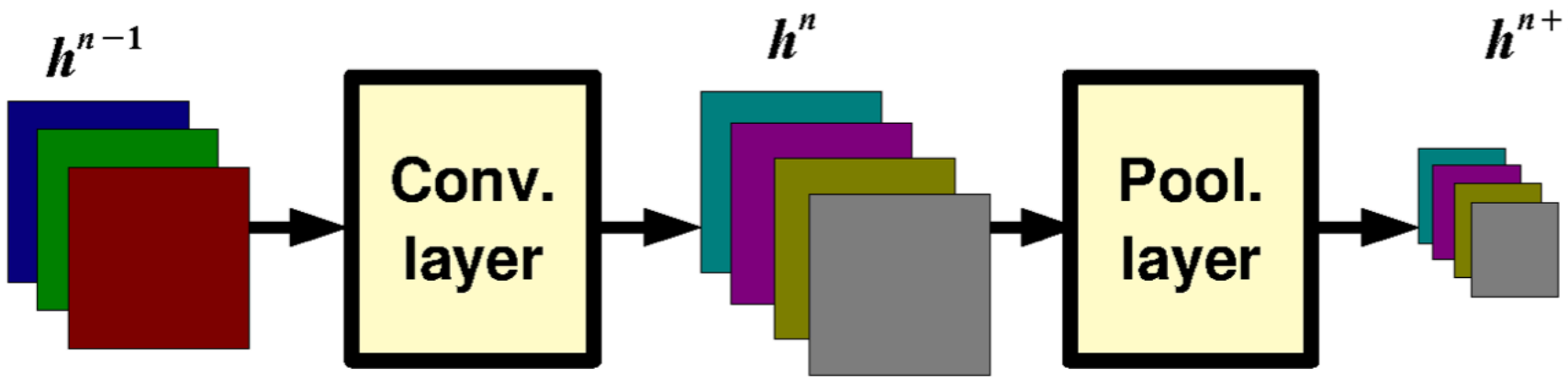


By *pooling* responses at different locations,
we gain robustness to the exact spatial location of image features.

Pooling is similar to pyrimid downsampling



Pooling Layer: Receptive Field Size



Pooling Layer: Examples

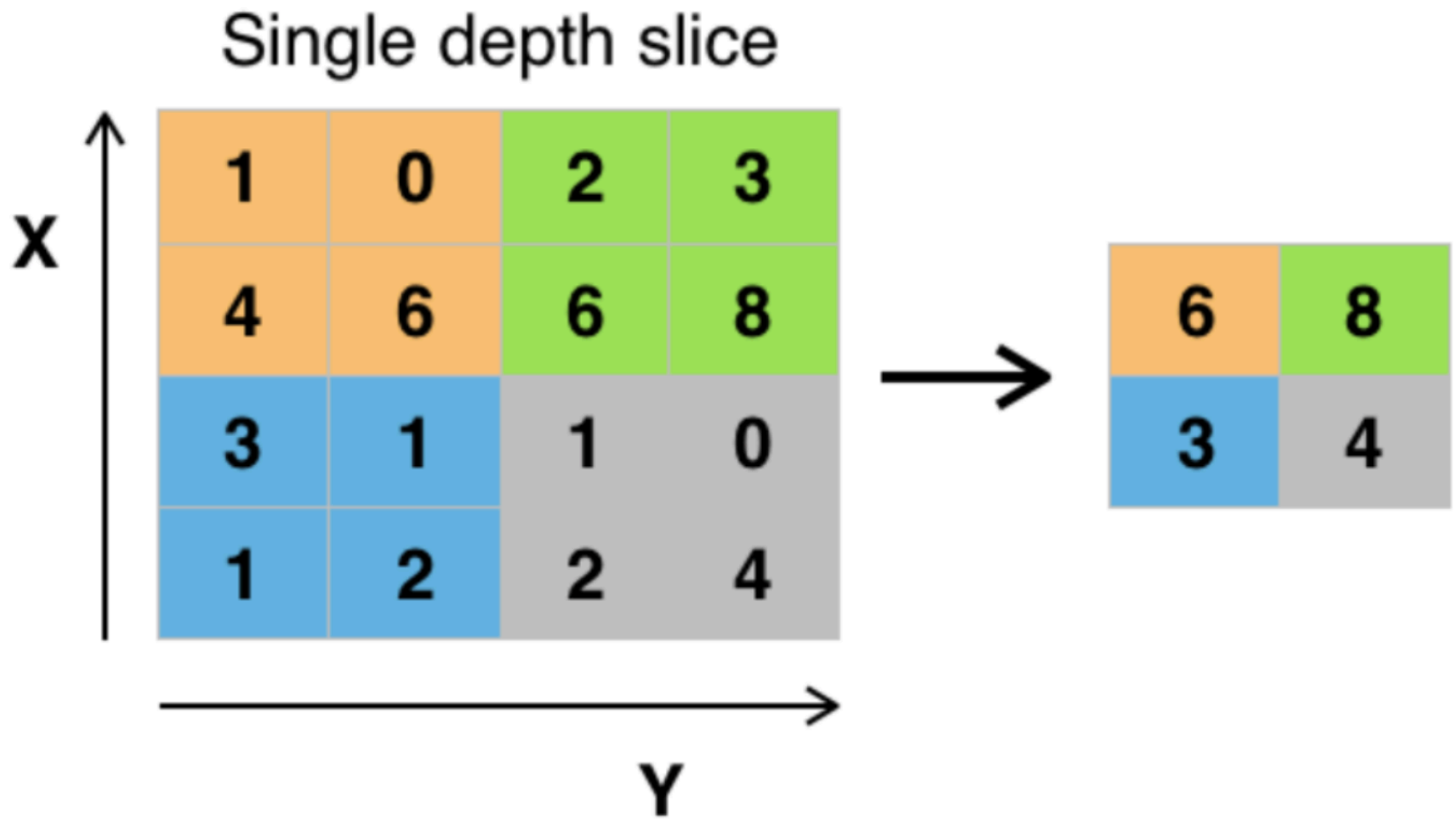
Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Max pooling



Wikipedia

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

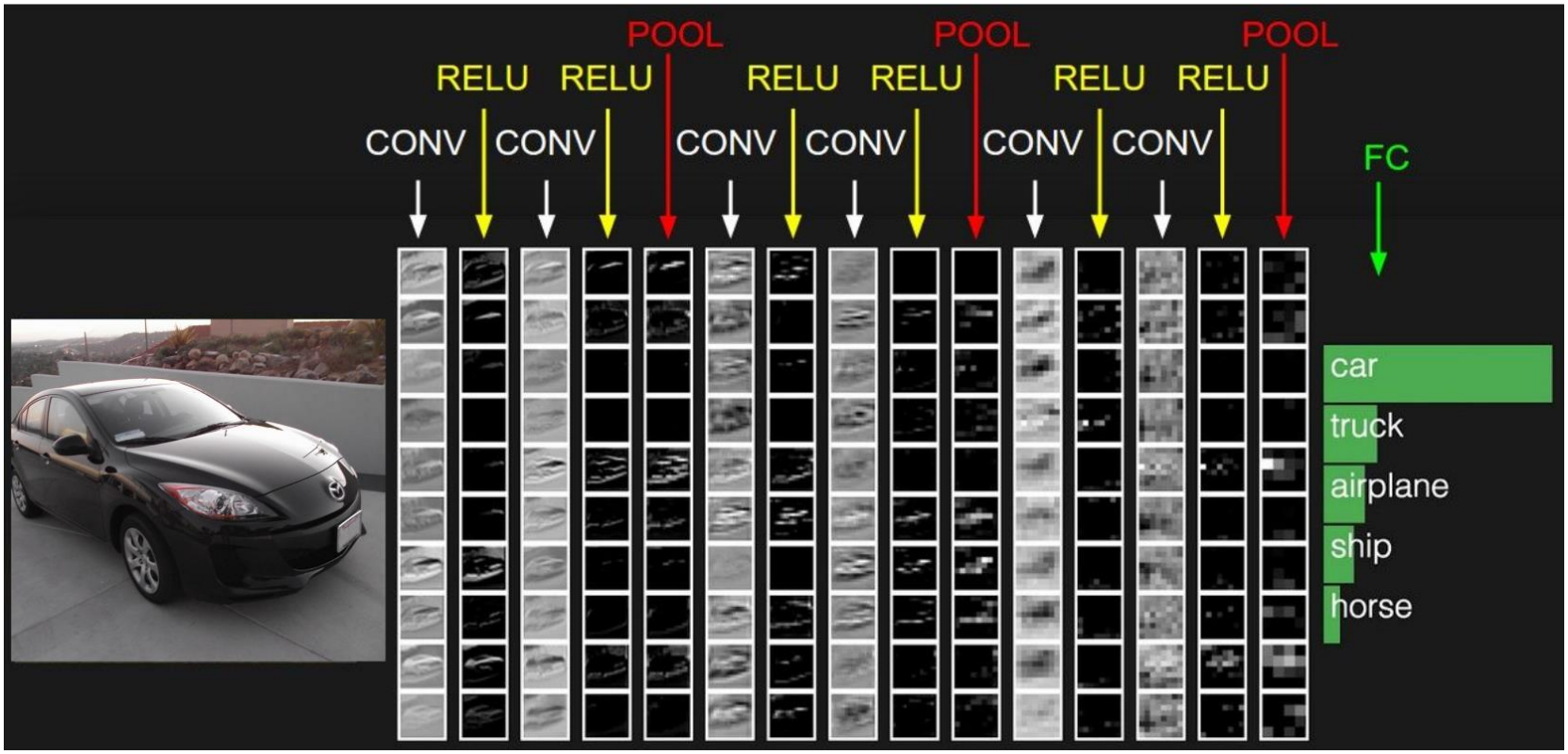
L2-pooling:

$$h_j^n(x,y) = \sqrt{\sum_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x,y) = \sqrt{\sum_{k\in N(j)} h_k^{n-1}(x,y)^2}$$

A CNN example



Next lecture

- Deep learning for computer vision applications