

All You Want To Know About CNNs

...

Yukun Zhu

Deep Learning

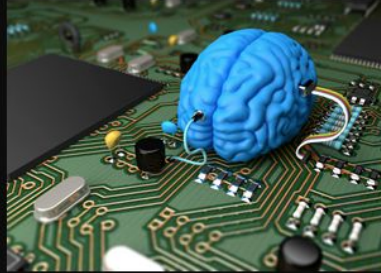
Deep Learning



Deep Learning



What society thinks I do

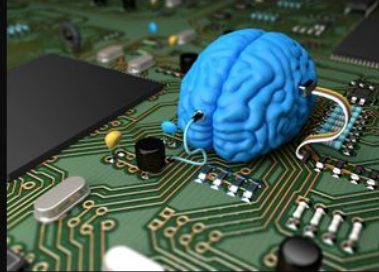


What my friends think I do

Deep Learning



What society thinks I do



What my friends think I do

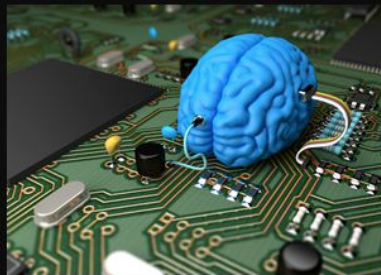


**What other computer
scientists think I do**

Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do

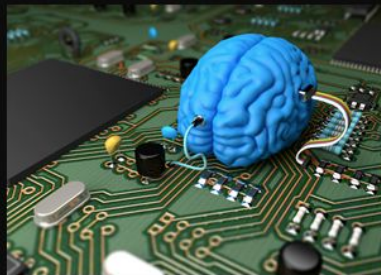


What mathematicians think I do

Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do



What mathematicians think I do



What I think I do

Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do



What mathematicians think I do



What I think I do

```
from theano import *
```

What I actually do

Deep Learning in Vision

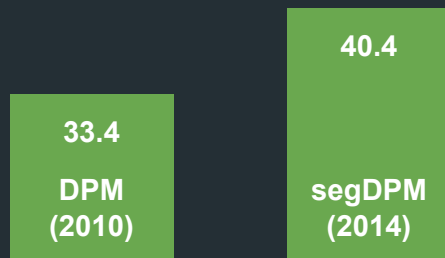
Object detection performance, PASCAL VOC 2010

33.4

DPM
(2010)

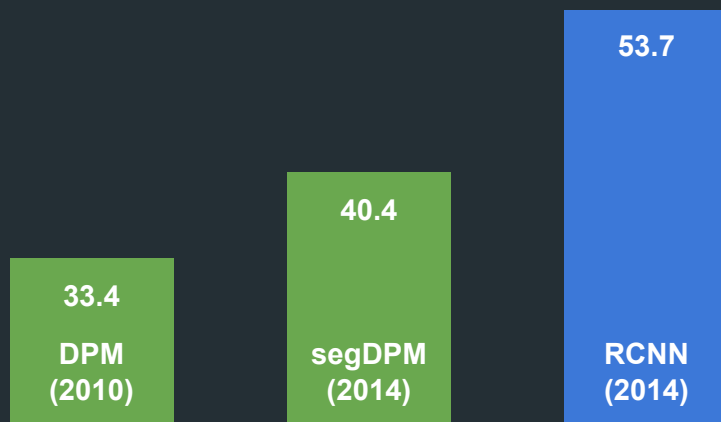
Deep Learning in Vision

Object detection performance, PASCAL VOC 2010



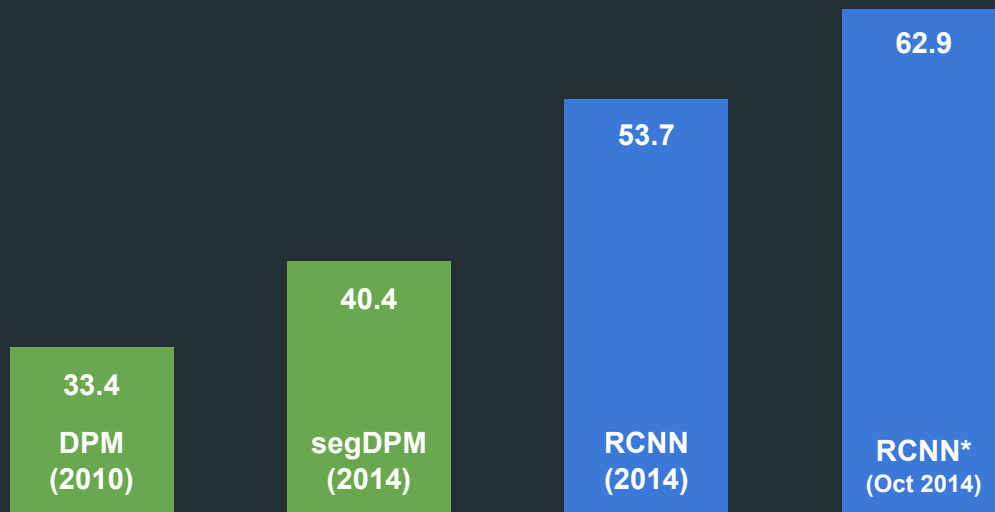
Deep Learning in Vision

Object detection performance, PASCAL VOC 2010



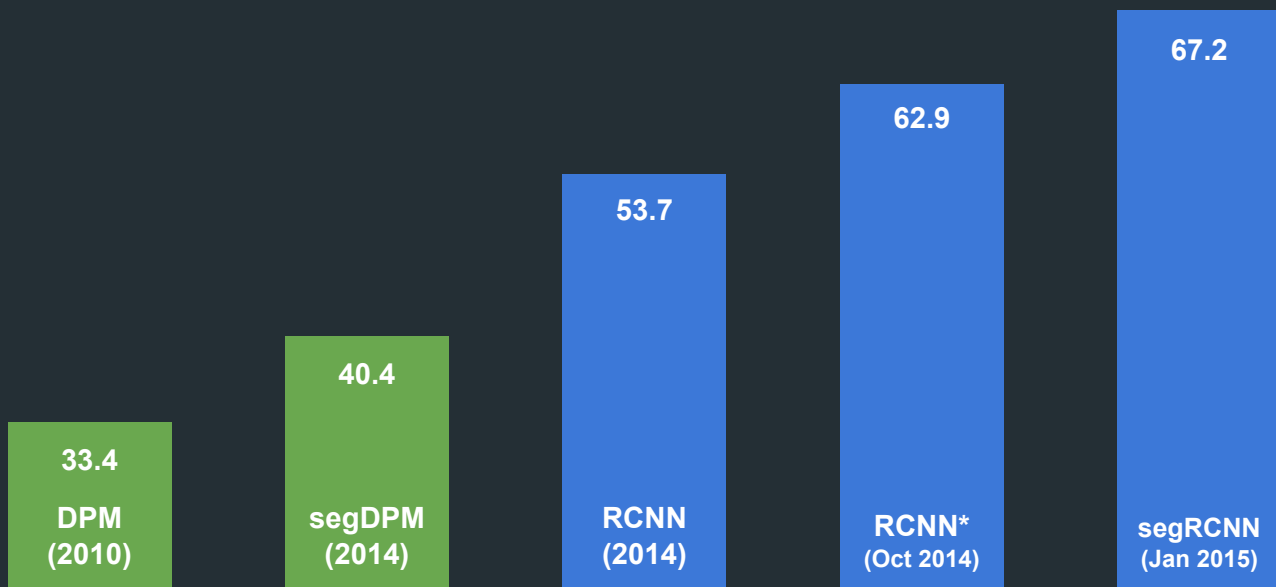
Deep Learning in Vision

Object detection performance, PASCAL VOC 2010



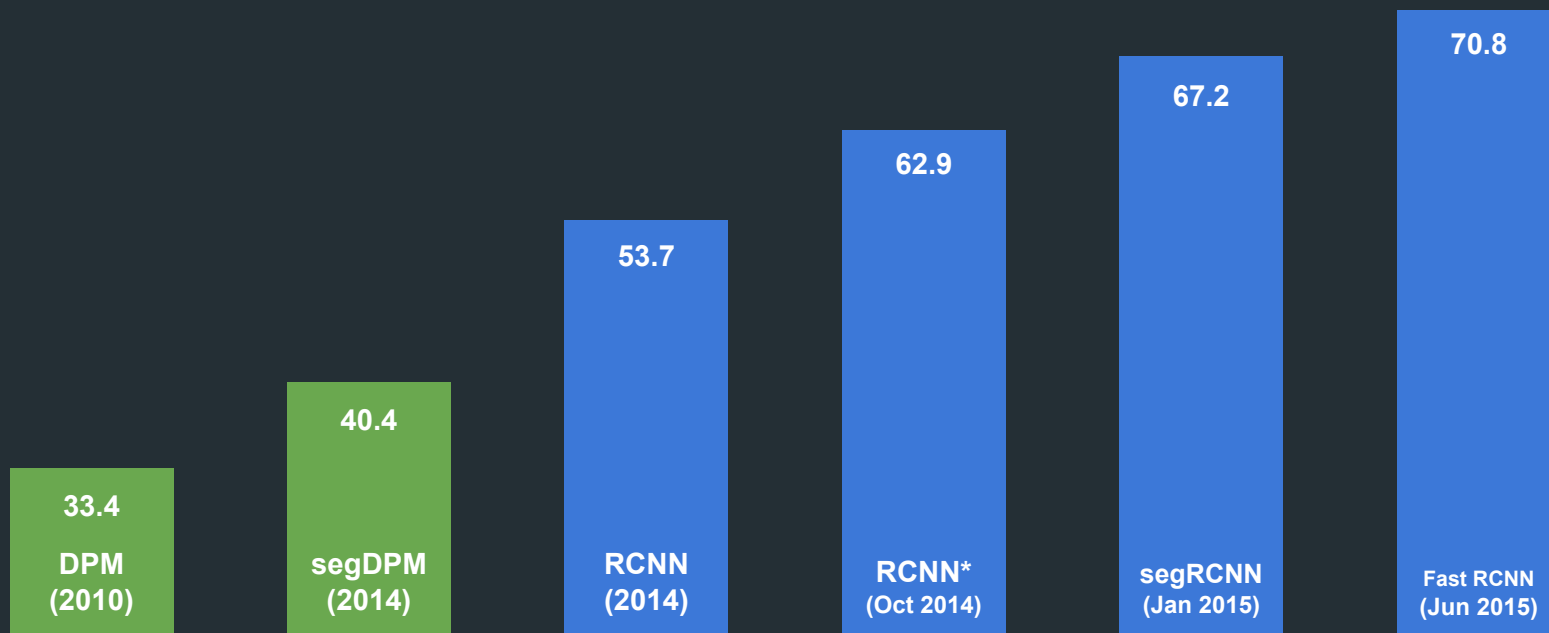
Deep Learning in Vision

Object detection performance, PASCAL VOC 2010

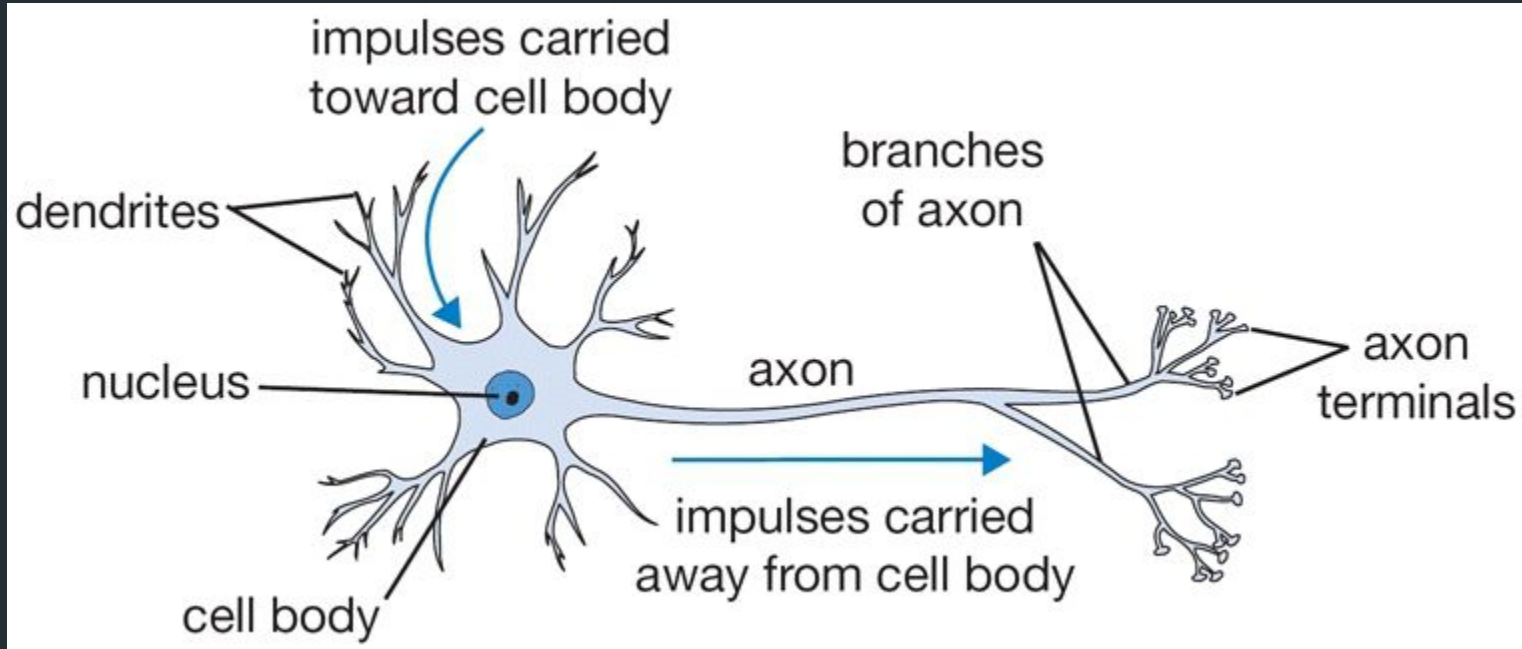


Deep Learning in Vision

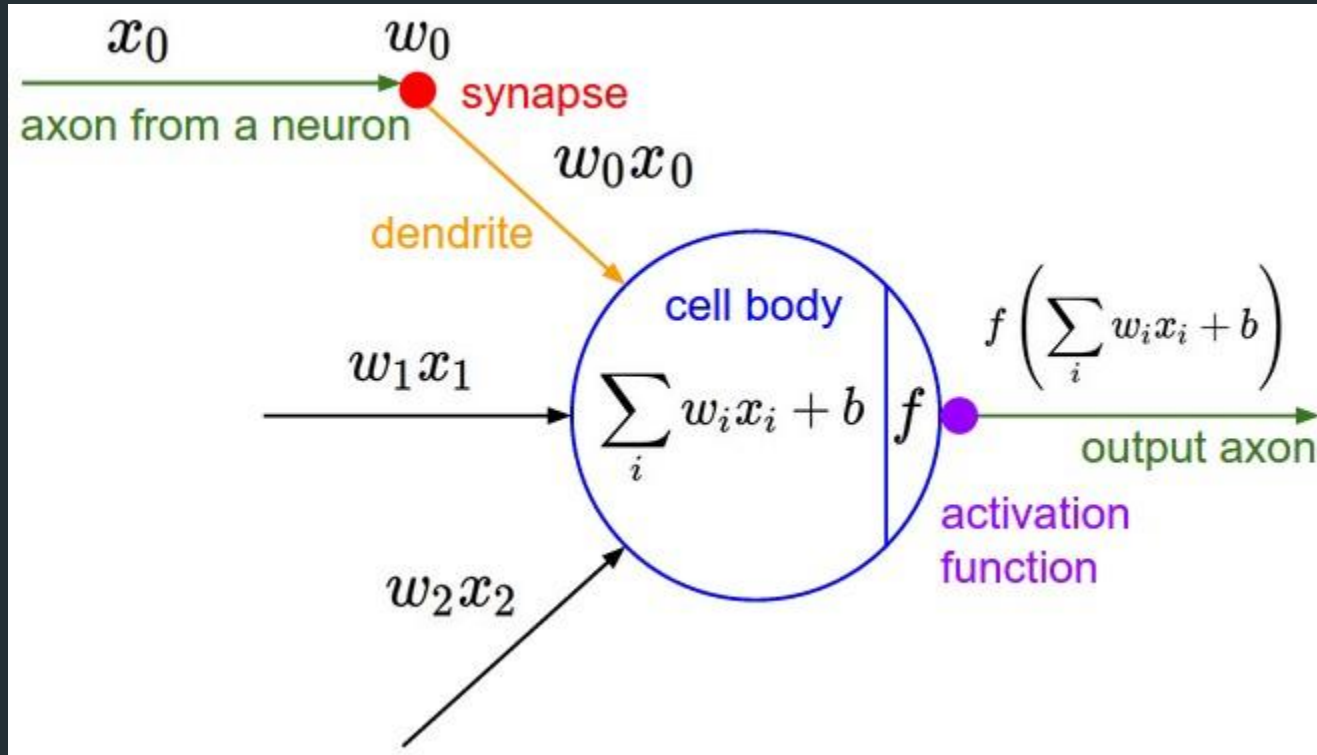
Object detection performance, PASCAL VOC 2010



A Neuron



A Neuron in Neural Network

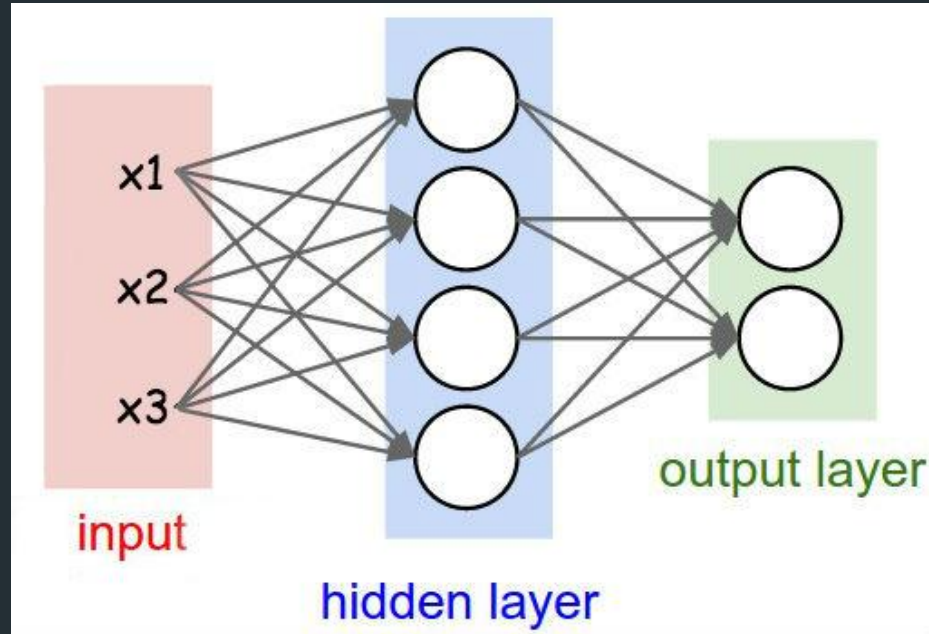


Activation Functions

- Sigmoid: $f(x) = 1 / (1 + e^{-x})$
- ReLU: $f(x) = \max(0, x)$
- Leaky ReLU: $f(x) = \max(ax, x)$
- Maxout: $f(x) = \max(w_0x + b_0, w_1x + b_1)$
- and many others...

Neural Network (MLP)

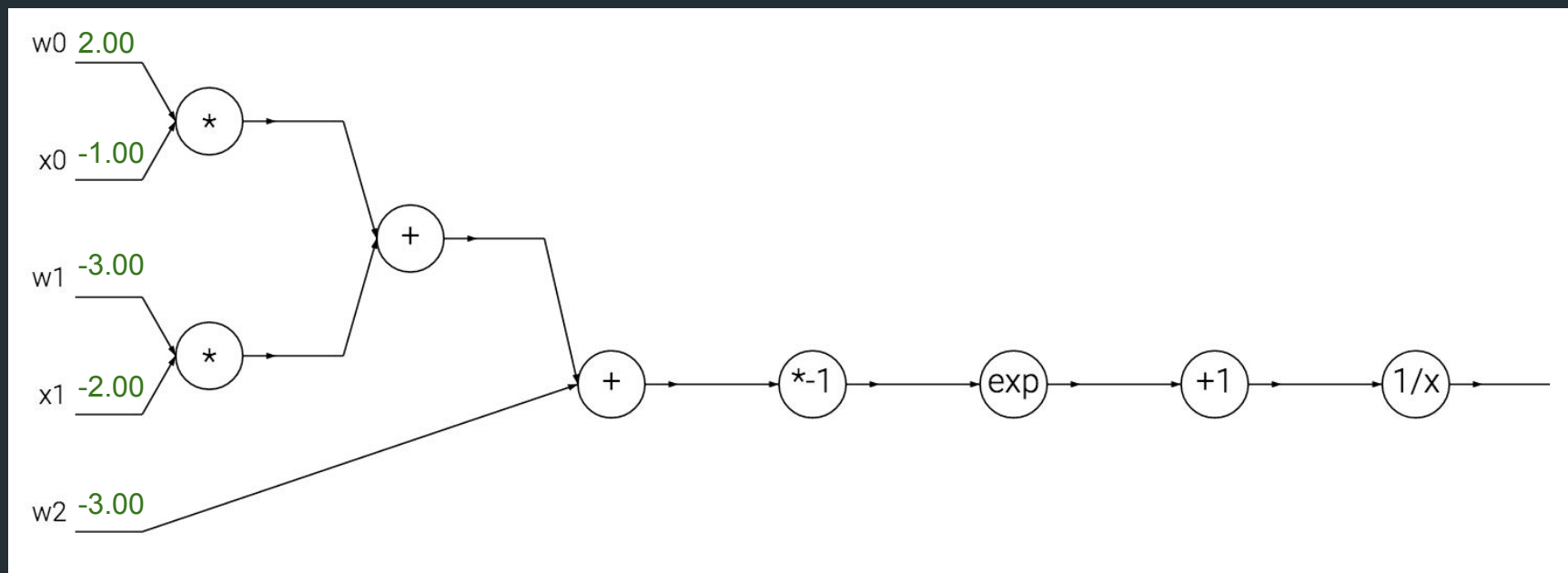
The network simulates a function $y = f(x; w)$



Forward Computation



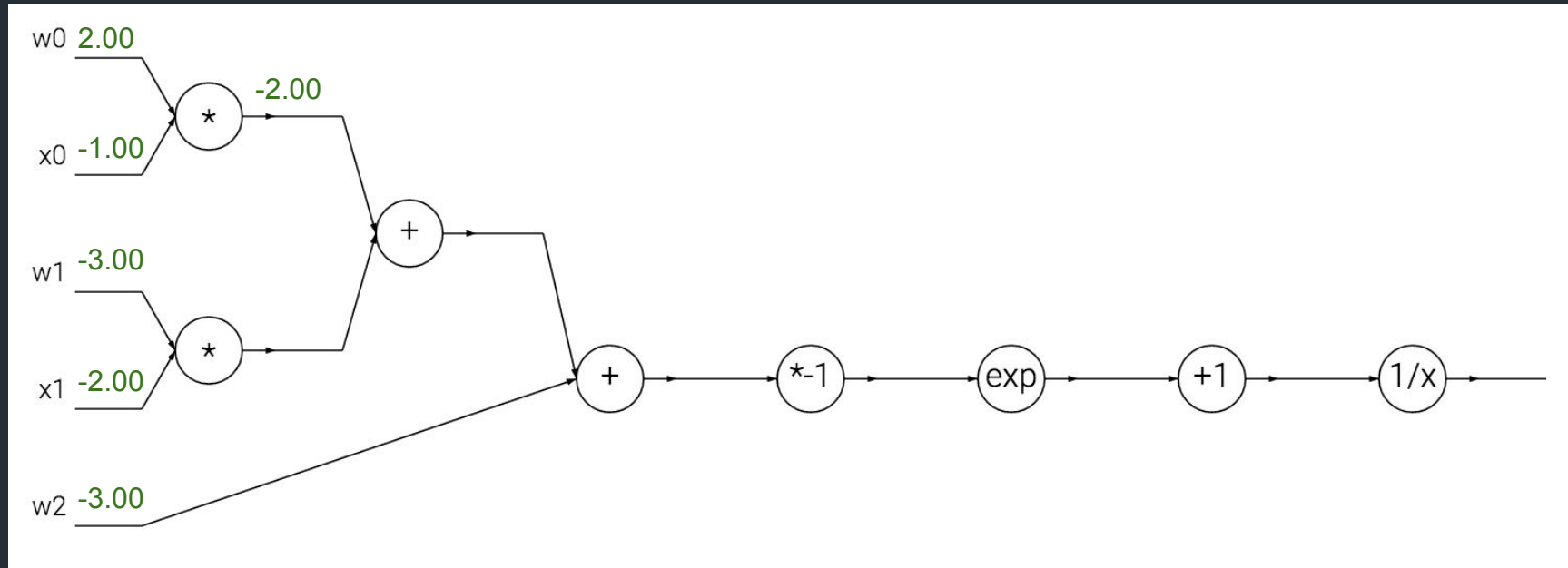
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



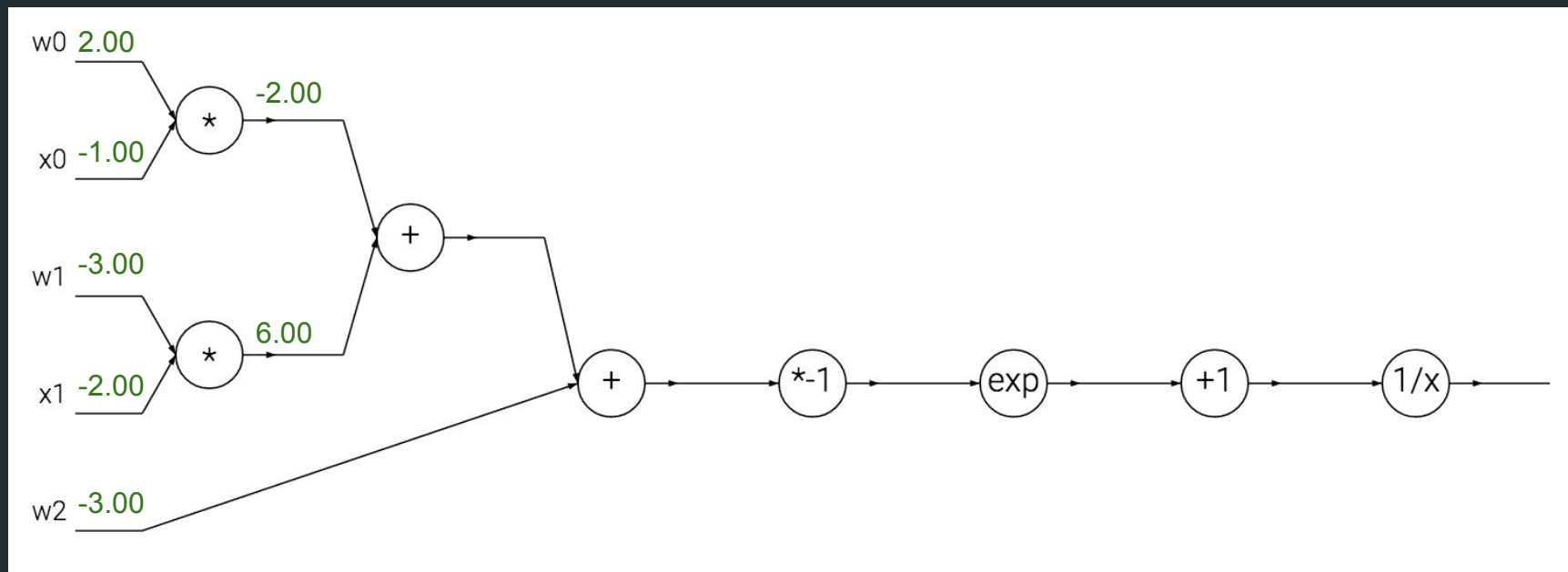
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



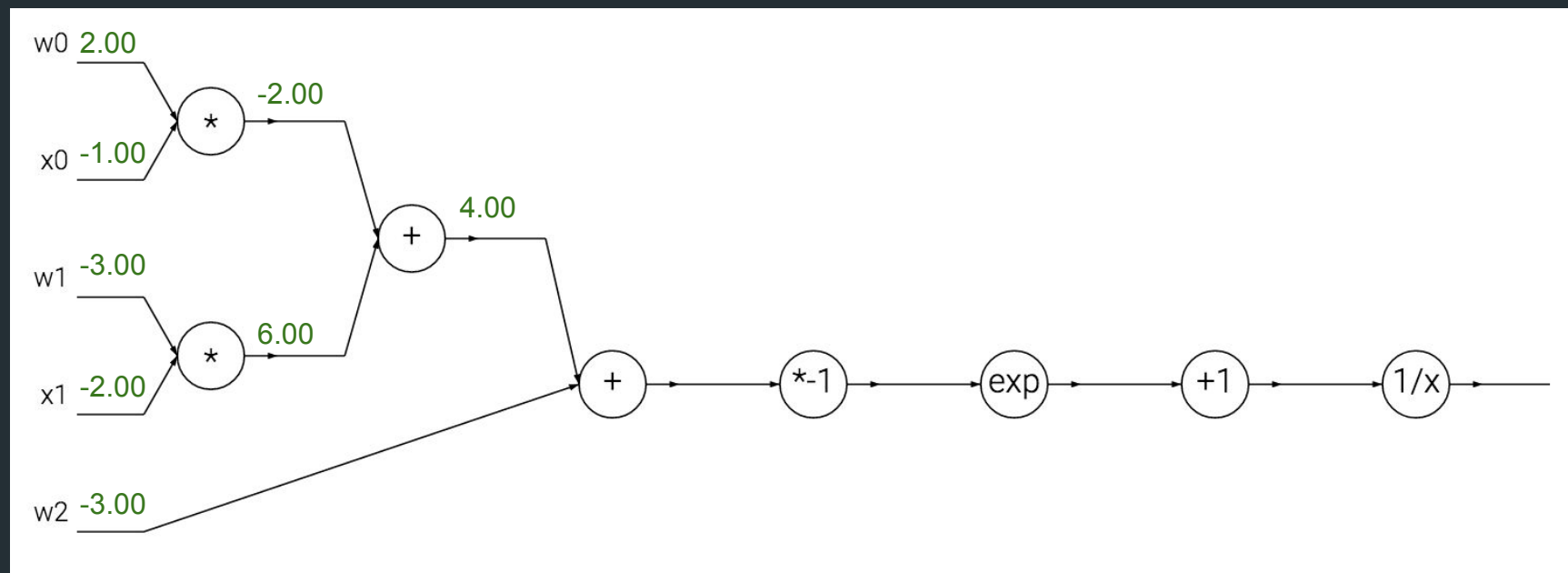
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



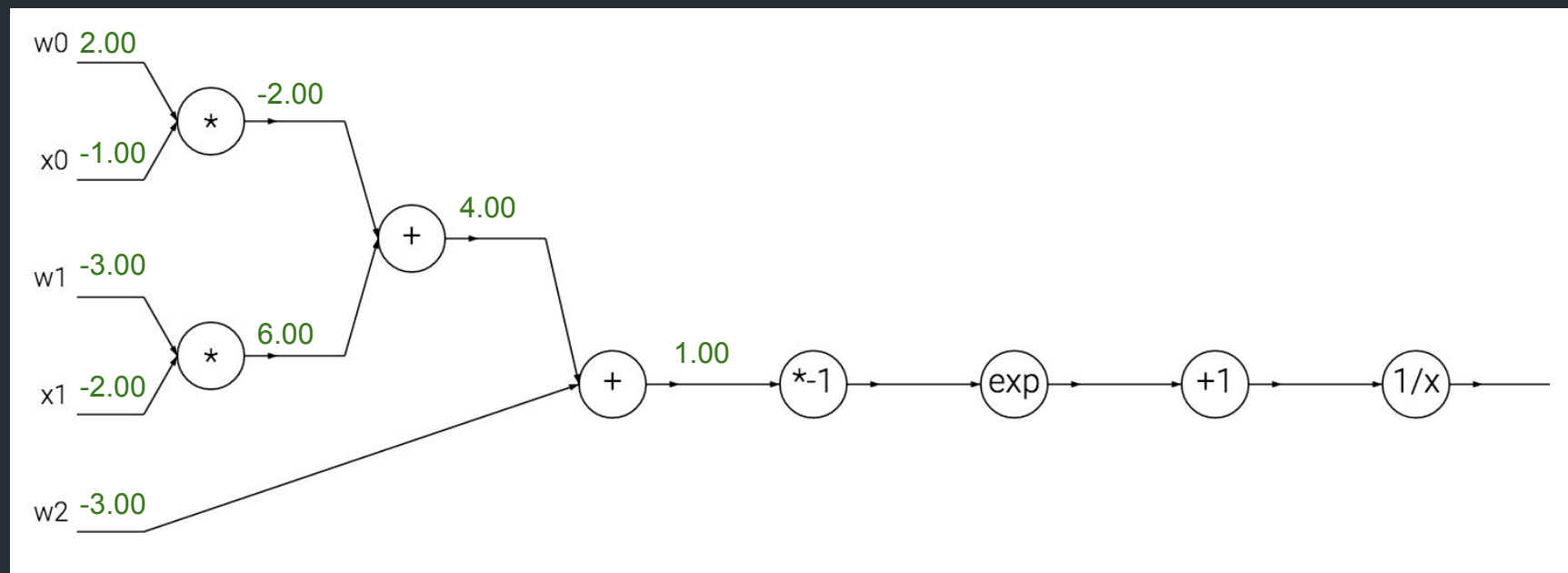
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$$



Forward Computation



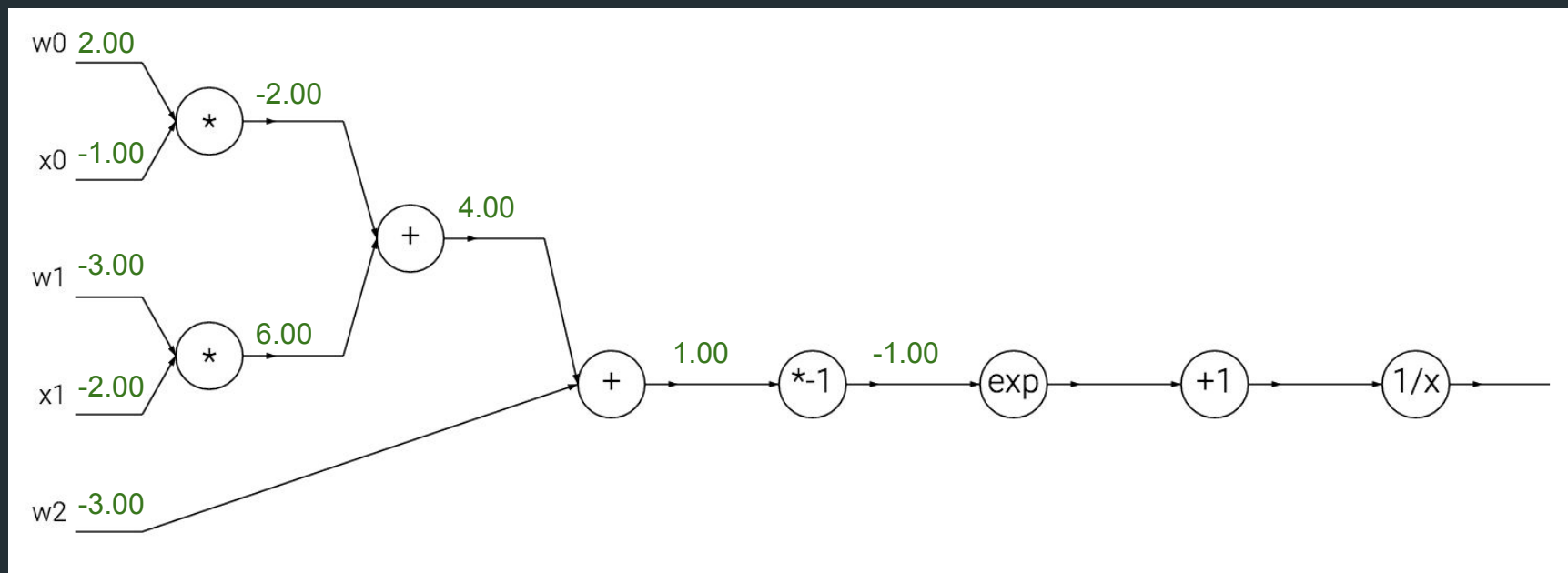
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



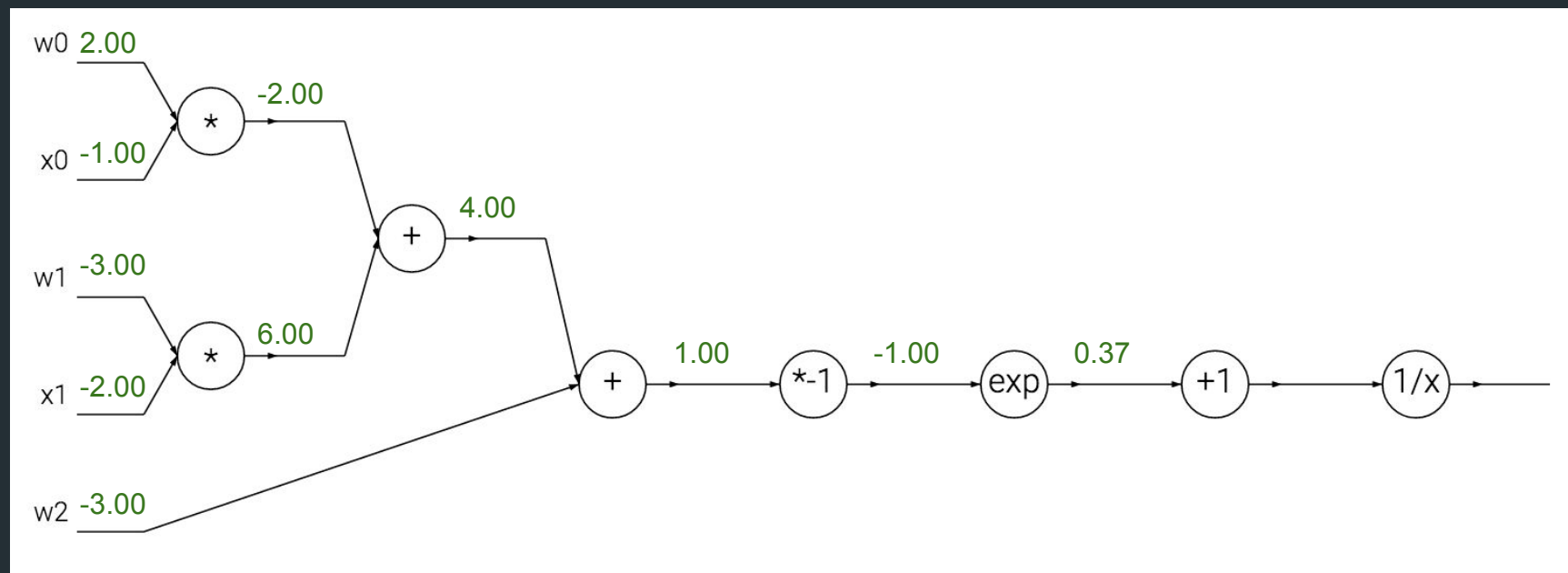
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



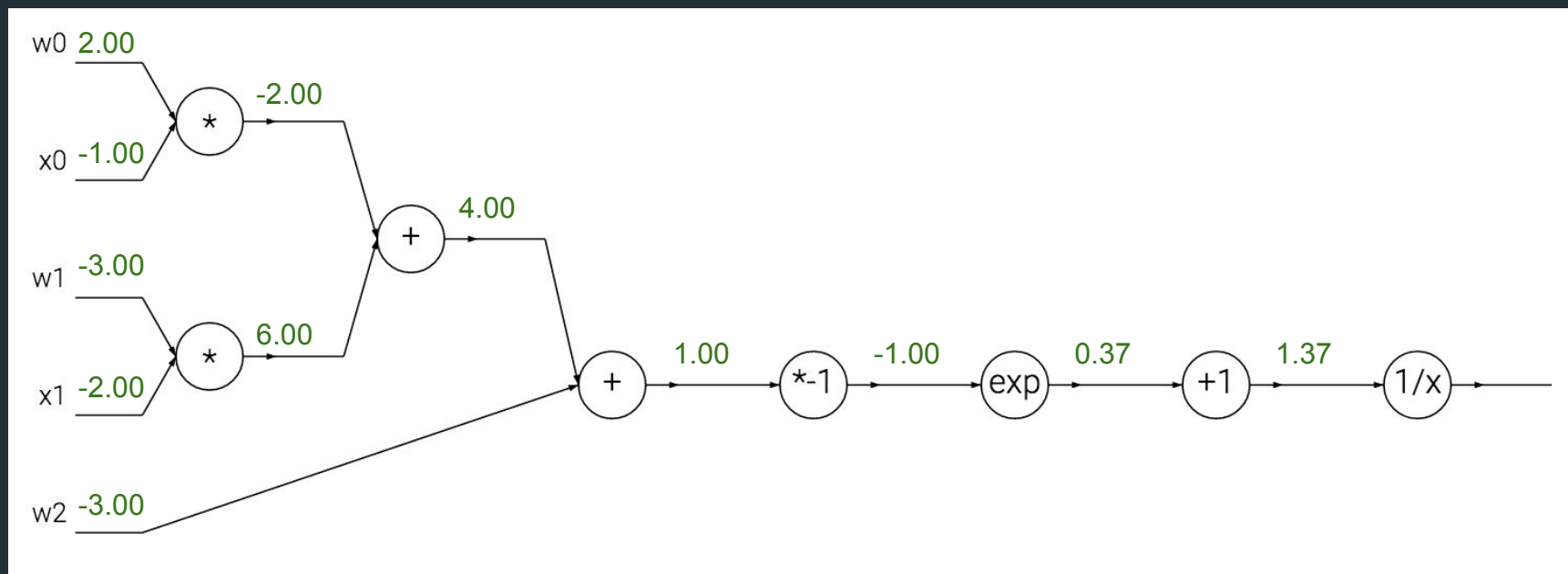
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$$



Forward Computation



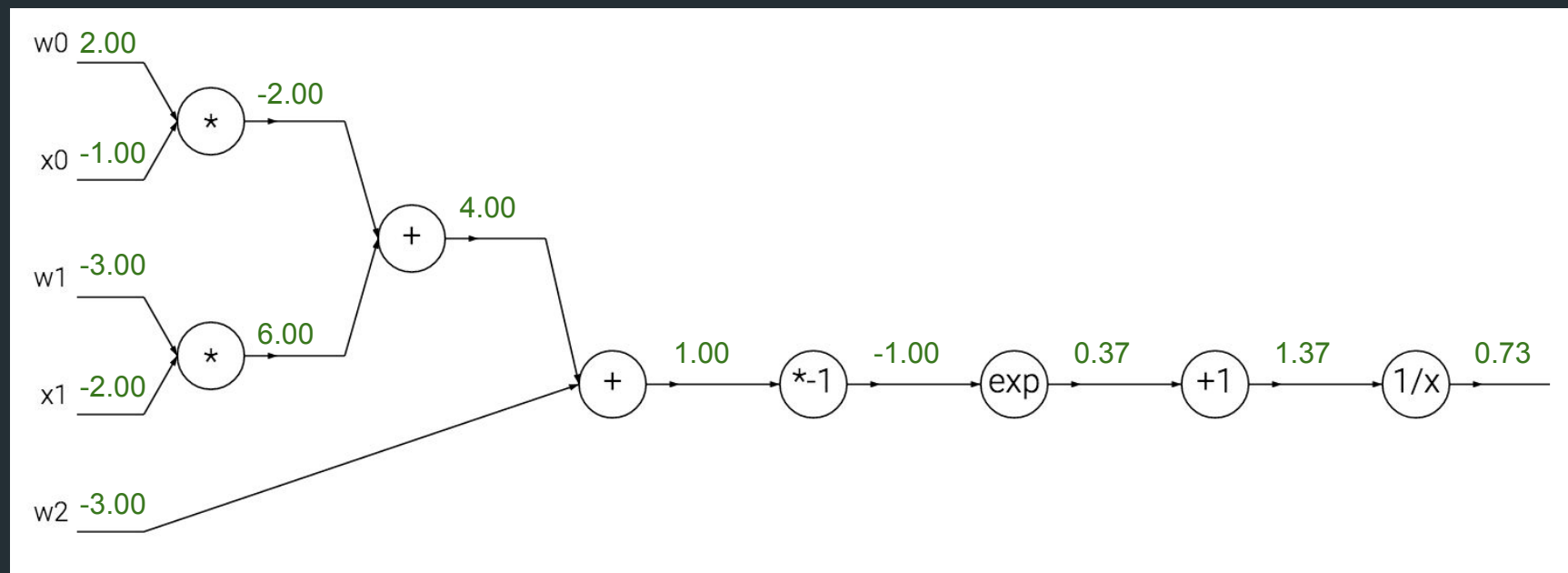
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Forward Computation



$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Loss Function

Loss function measures how well prediction matches true value

Commonly used loss function:

- Squared loss: $(y - y')^2$
- Cross-entropy loss: $-\sum_i (y_i' * \log(y_i))$
- and many others

Loss Function

During training, we would like to minimize the total loss on a set of training data

- We want to find $w^* = \operatorname{argmin}\{\sum_i [\operatorname{loss}(f(x_i; w), y_i)]\}$

Loss Function

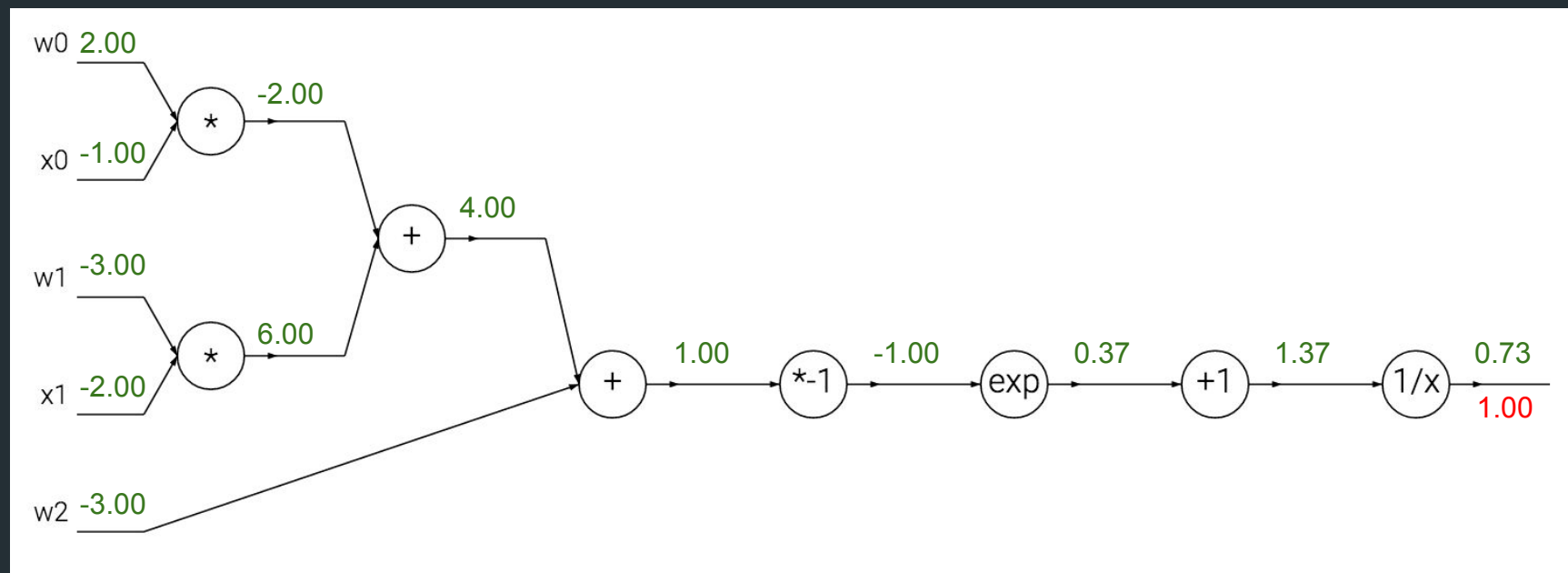
During training, we would like to minimize the total loss on a set of training data

- We want to find $w^* = \operatorname{argmin}\{\sum_i [\operatorname{loss}(f(x_i; w), y_i)]\}$
- Usually we use gradient based approach
 - $w^{t+1} = w^t - a \nabla w$

Backward Computation



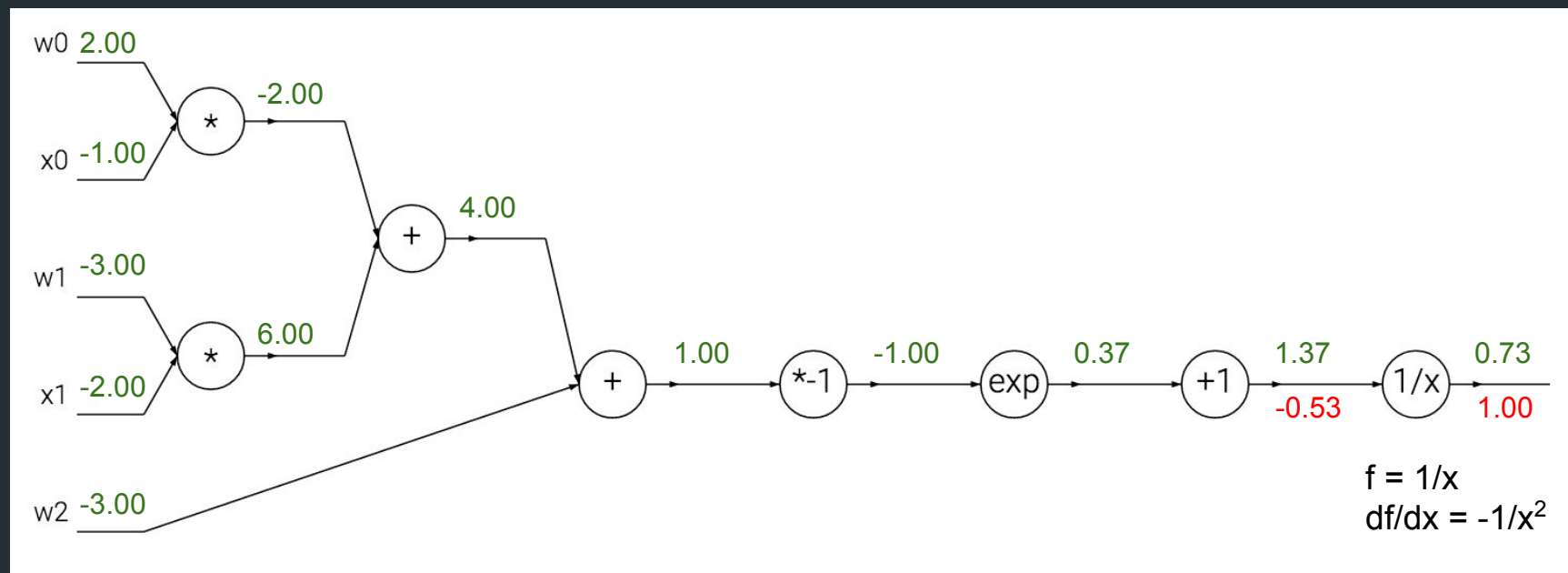
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



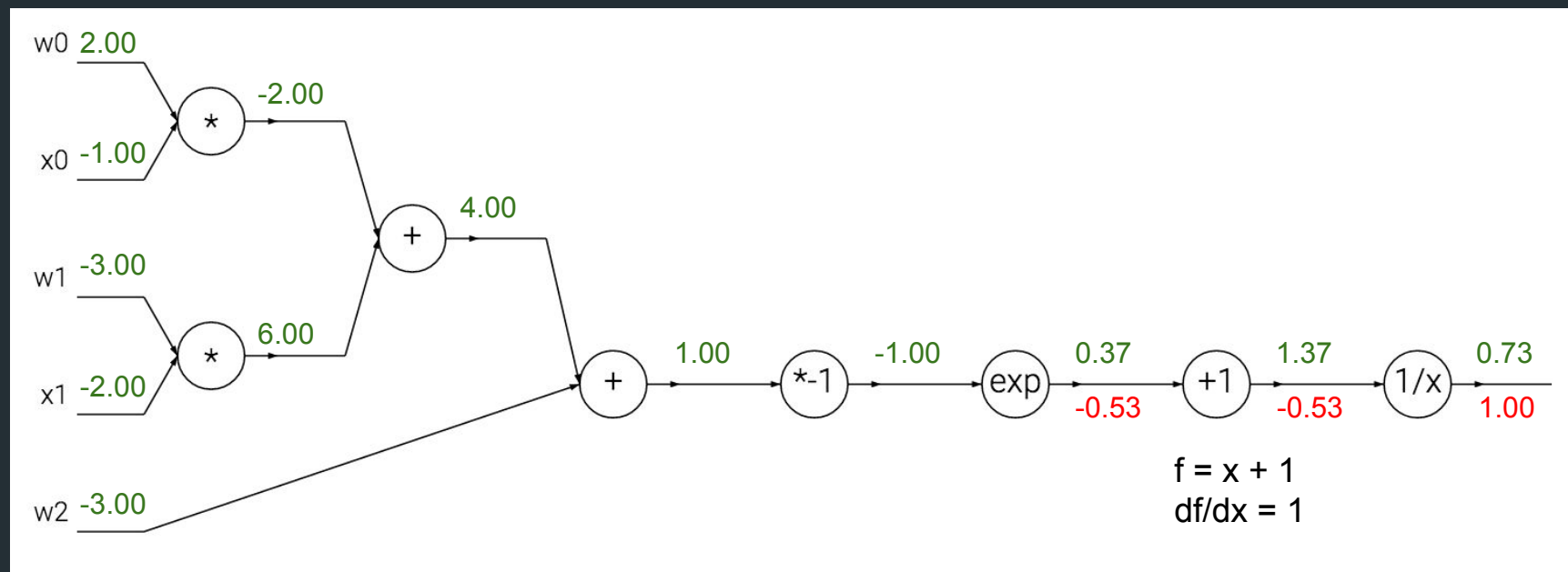
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



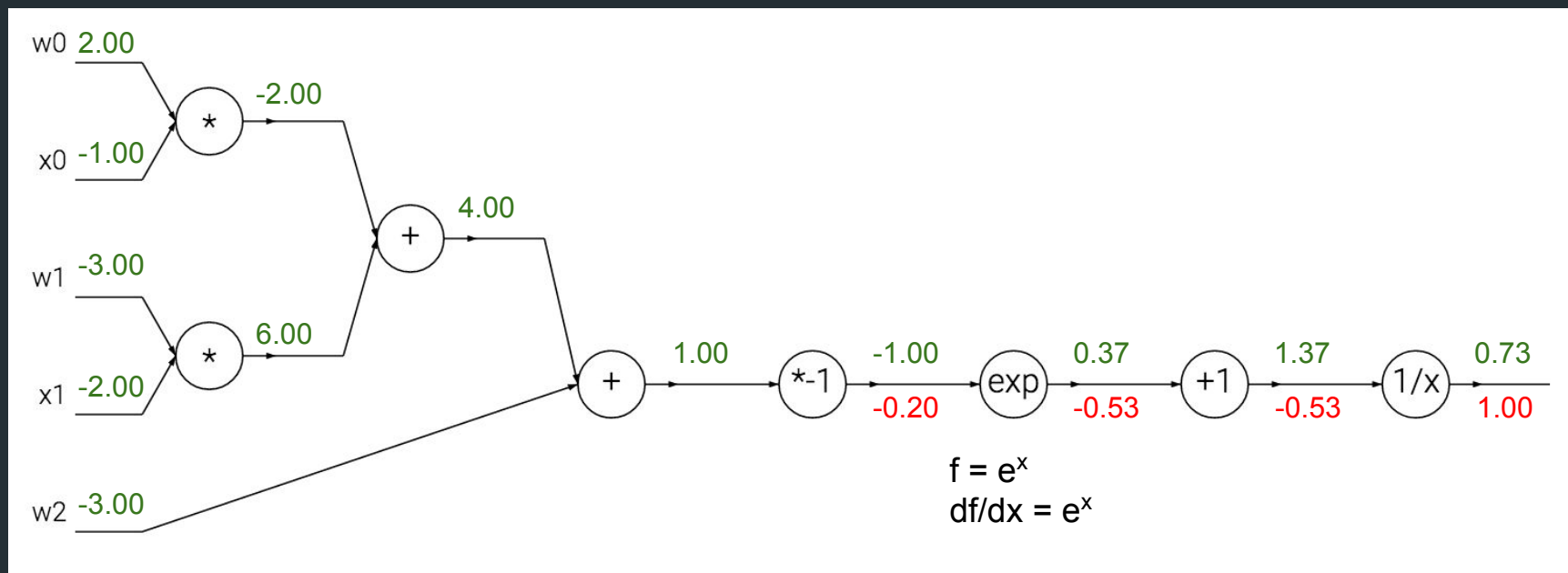
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



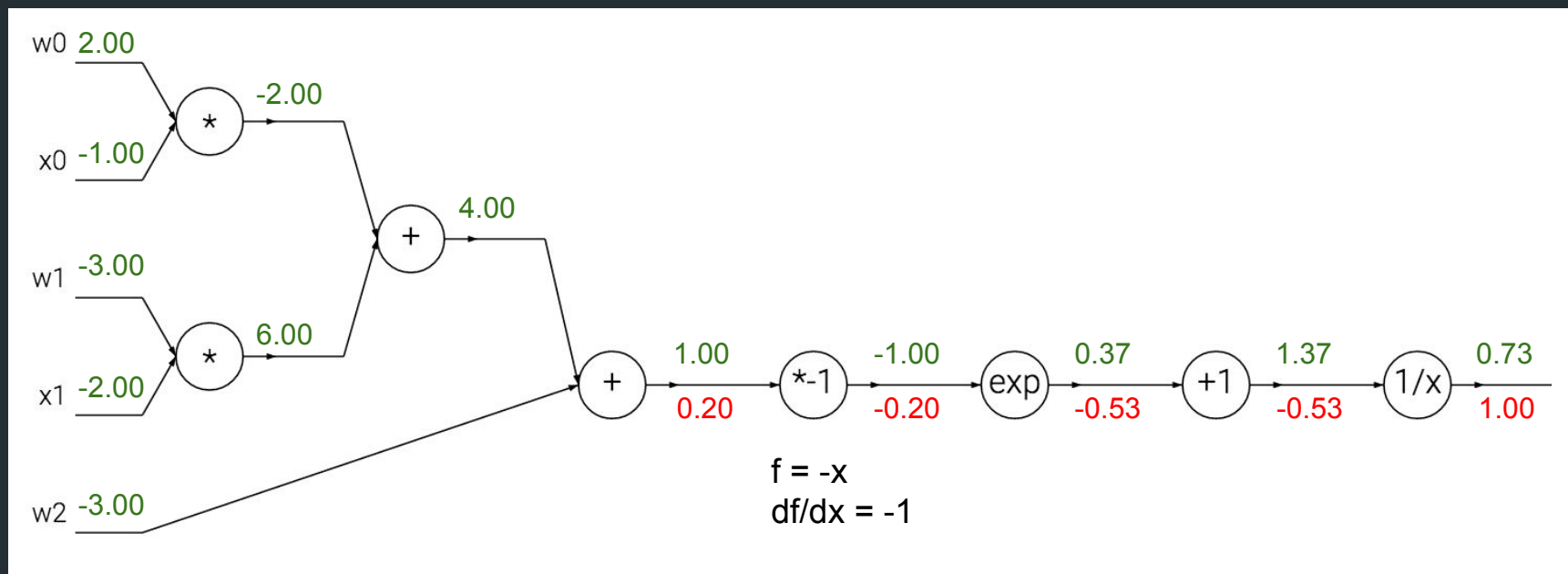
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



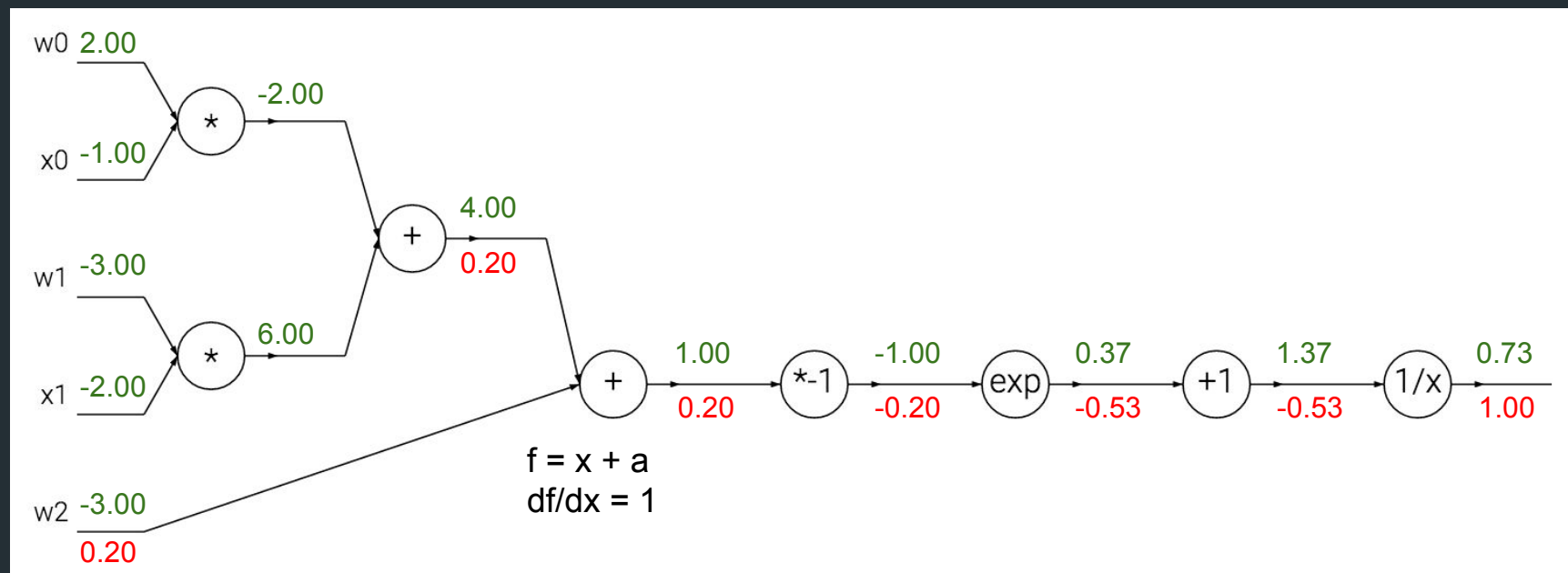
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



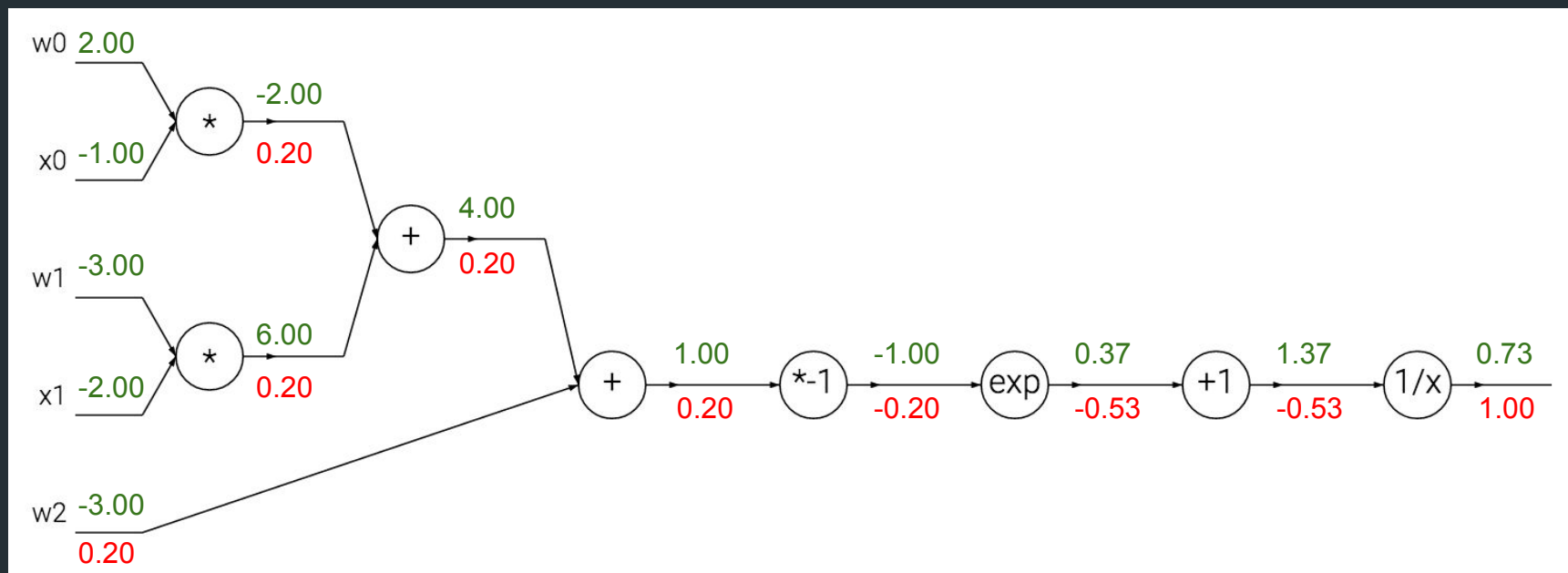
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



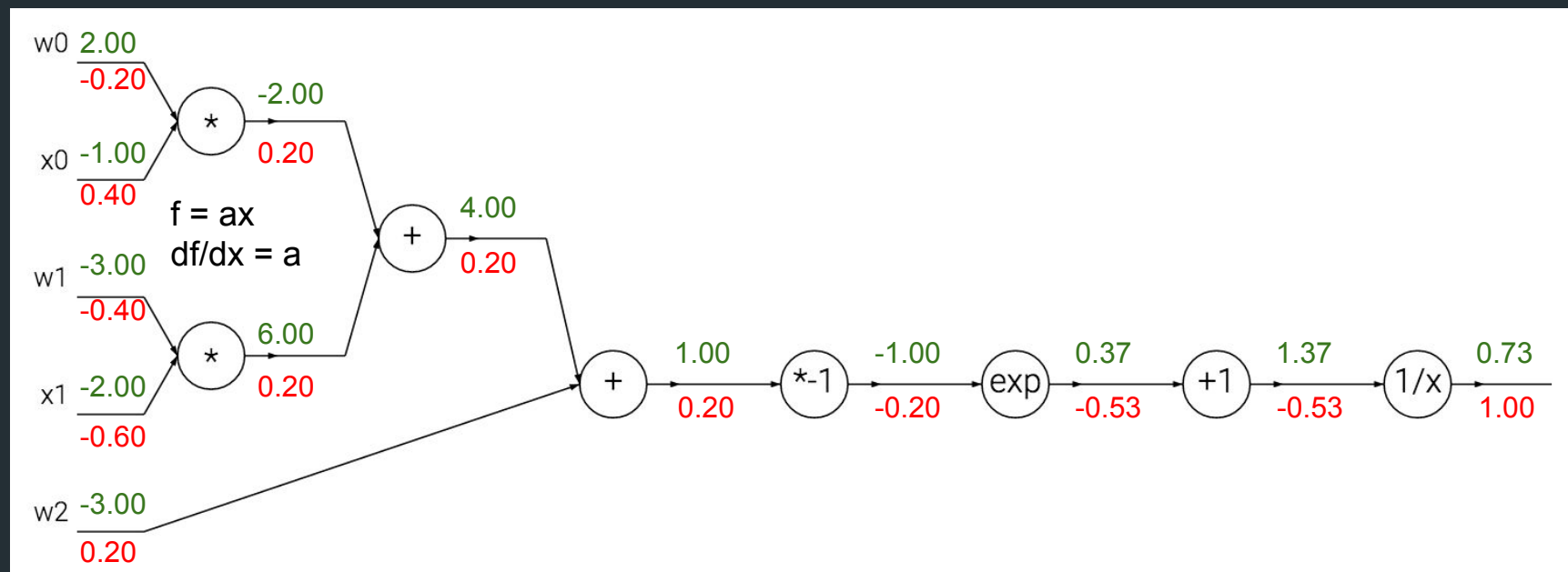
$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Backward Computation



$$f(x_0, x_1) = 1 / (1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2)))$$



Why NNs?

Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of \mathbb{R}^n , under mild assumptions on the activation function.

Stone's Theorem

- Suppose X is a compact Hausdorff space and B is a subalgebra in $C(X, \mathbb{R})$ such that:
 - B separates points.
 - B contains the constant function 1.
 - If $f \in B$ then $af \in B$ for all constants $a \in \mathbb{R}$.
 - If $f, g \in B$, then $f + g, \max\{f, g\} \in B$.
- Then every continuous function defined on $C(X, \mathbb{R})$ can be approximated as closely as desired by functions in B

Why CNNs?

Problems of MLP in Vision

For input as a $10 * 10$ image:

- A 3 layer MLP with 200 hidden units contains ~100k parameters

For input as a $100 * 100$ image:

- A 1 layer MLP with 20k hidden units contains ~200m parameters

Can We Do Better?

Can We Do Better?



Can We Do Better?



Can We Do Better?



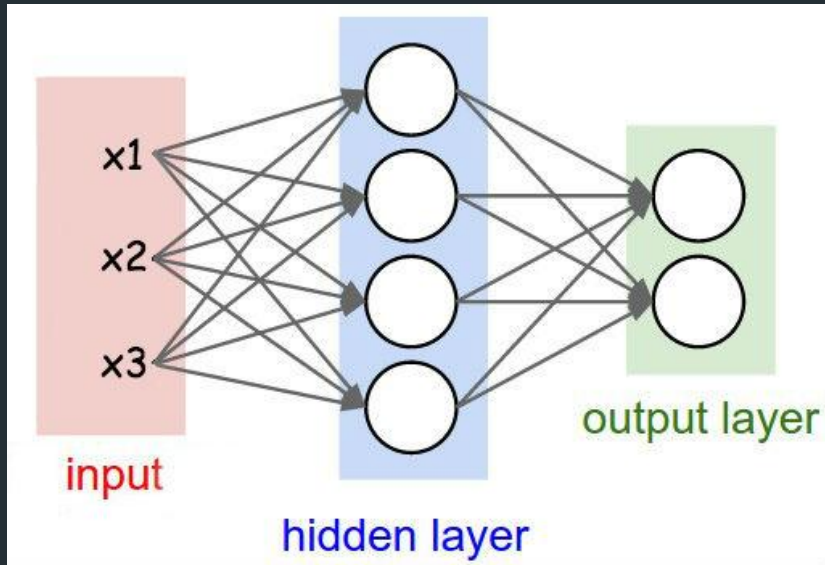
Can We Do Better?

Based on such observation, MLP can be improved in two ways:

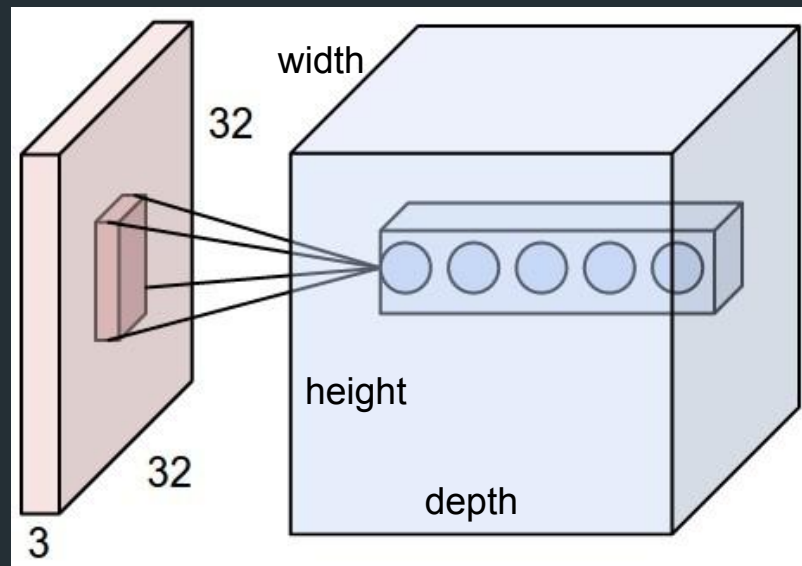
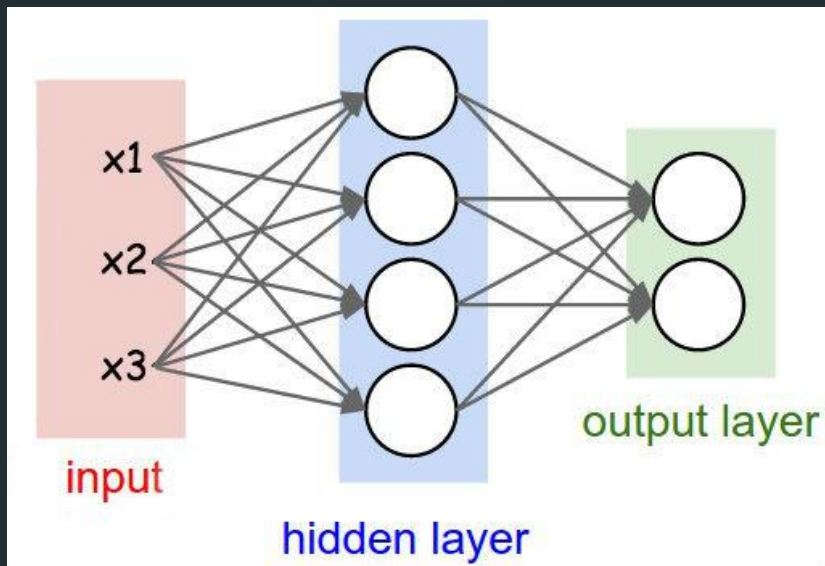
- Locally connected instead of fully connected
- Sharing weights between neurons

We achieve those by using convolution neurons

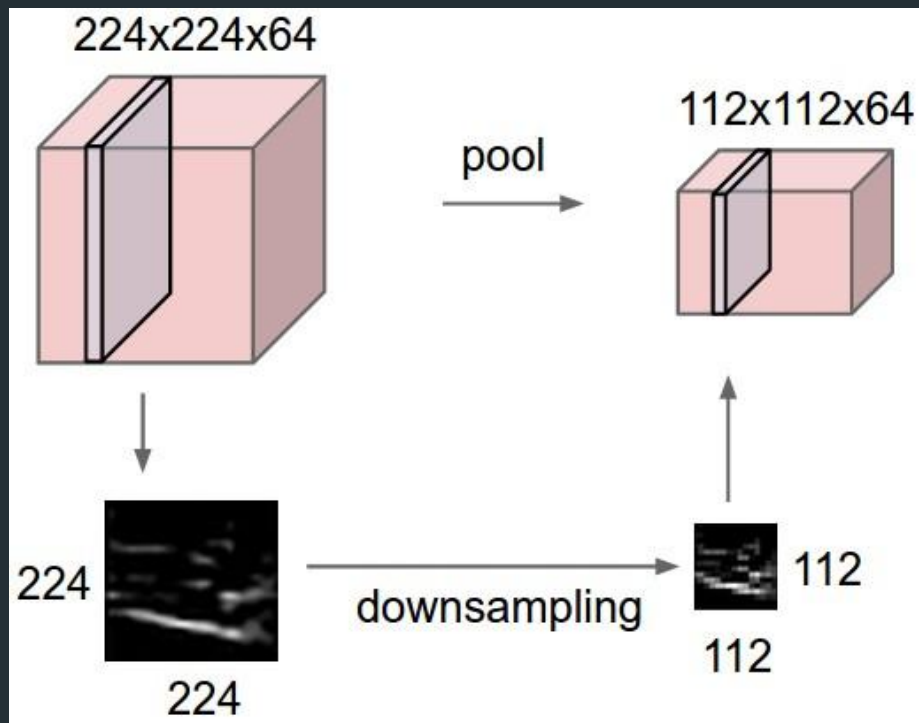
Convolutional Layers



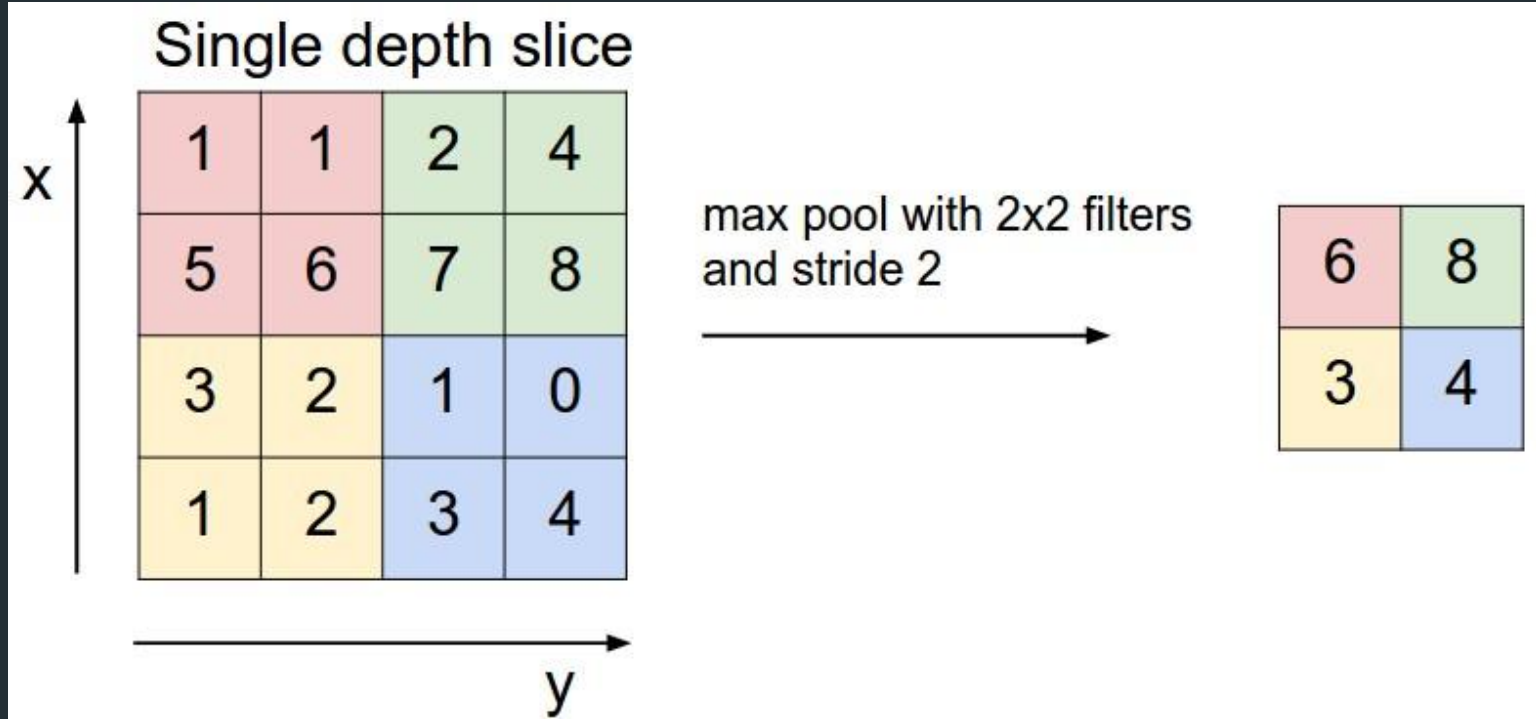
Convolutional Layers



Pooling Layers



Pooling Layers Example: Max Pooling



Pooling Layers

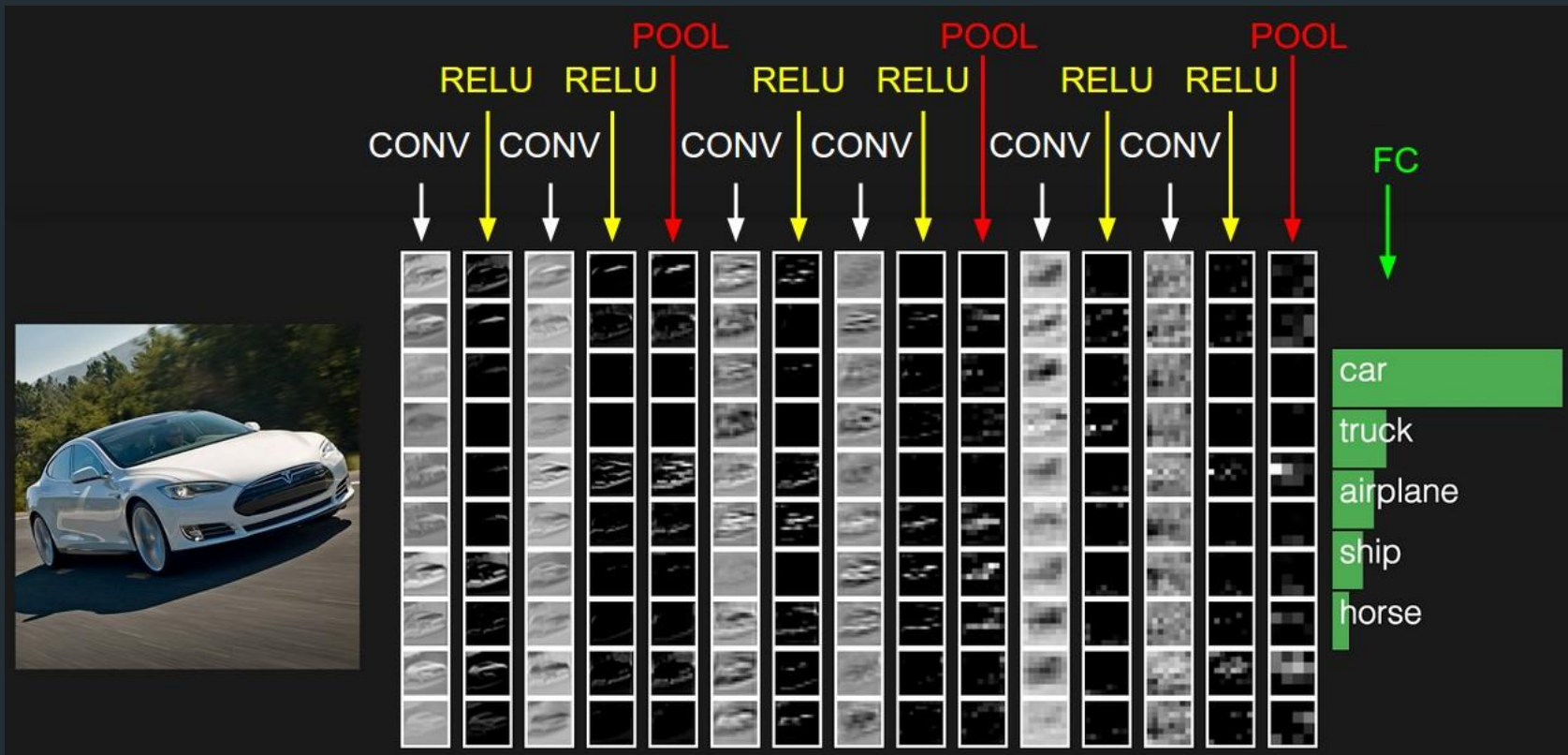
Commonly used pooling layers:

- Max pooling
- Average pooling

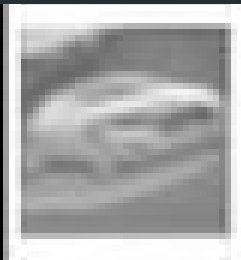
Why pooling layers?

- Reduce activation dimensionality
- Robust against tiny shifts

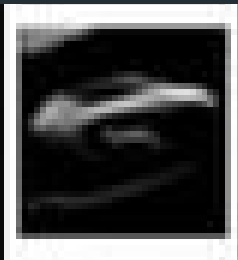
CNN Architecture: An Example



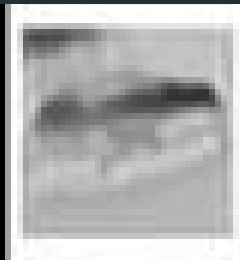
Layer Activations for CNNs



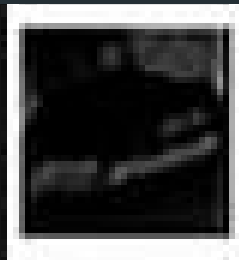
Conv:1



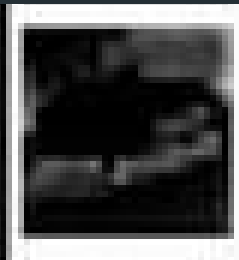
ReLU:1



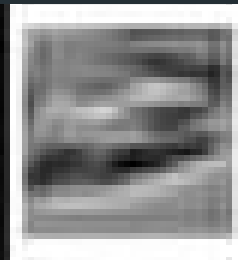
Conv:2



ReLU:2

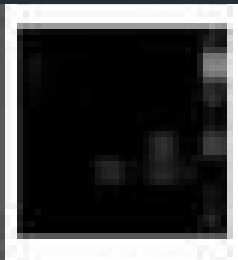


MaxPool:1

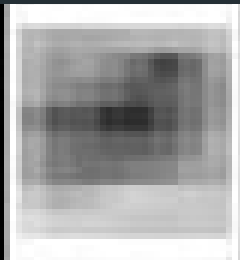


Conv:3

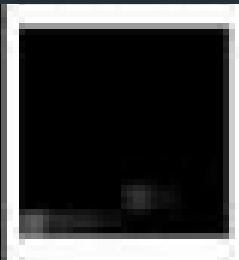
Layer Activations for CNNs



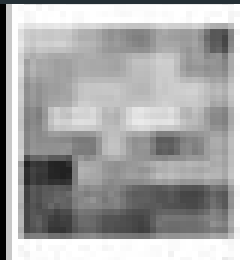
MaxPool:2



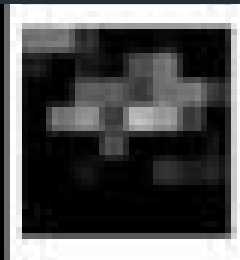
Conv:5



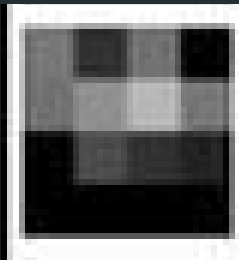
ReLU:5



Conv:6

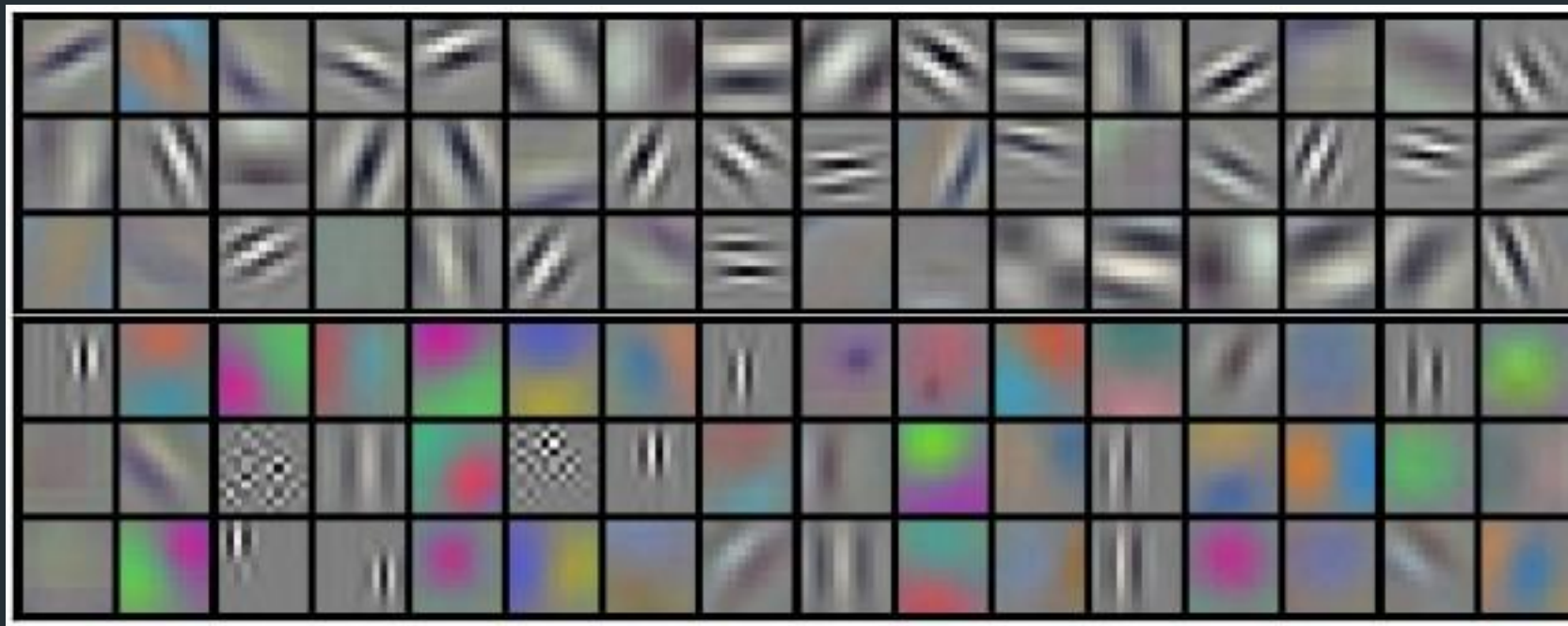


ReLU:6



MaxPool:3

Learnt Weights for CNNs: First Conv Layer of AlexNet

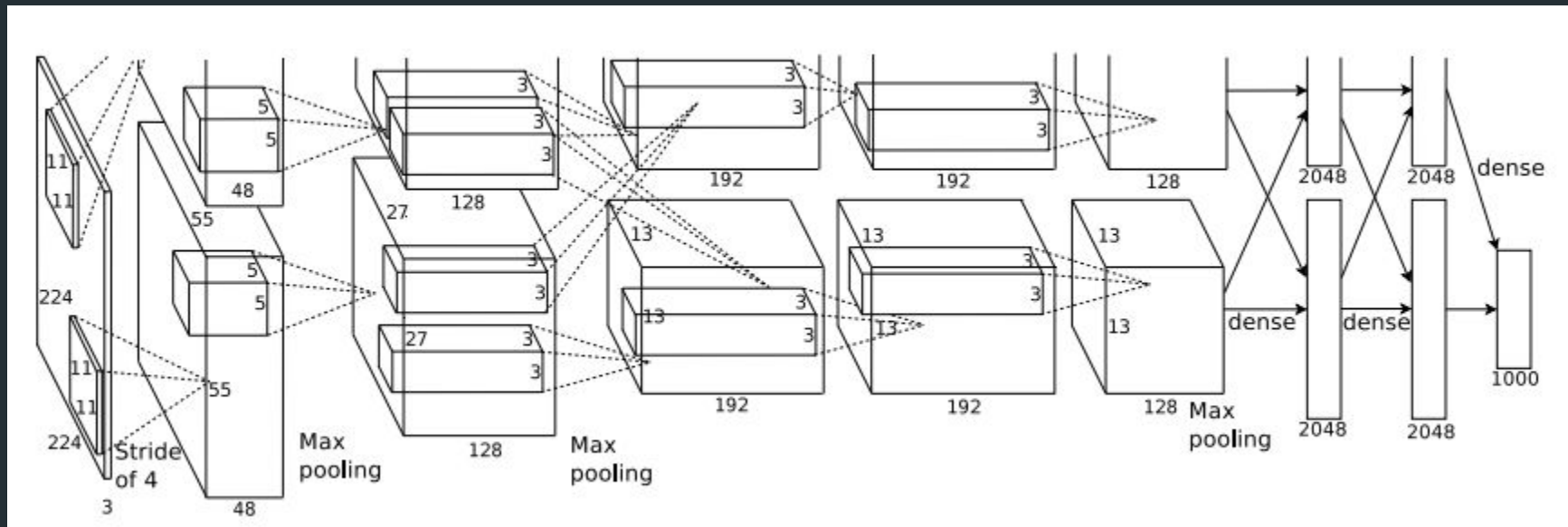


Why CNNs Work Now?

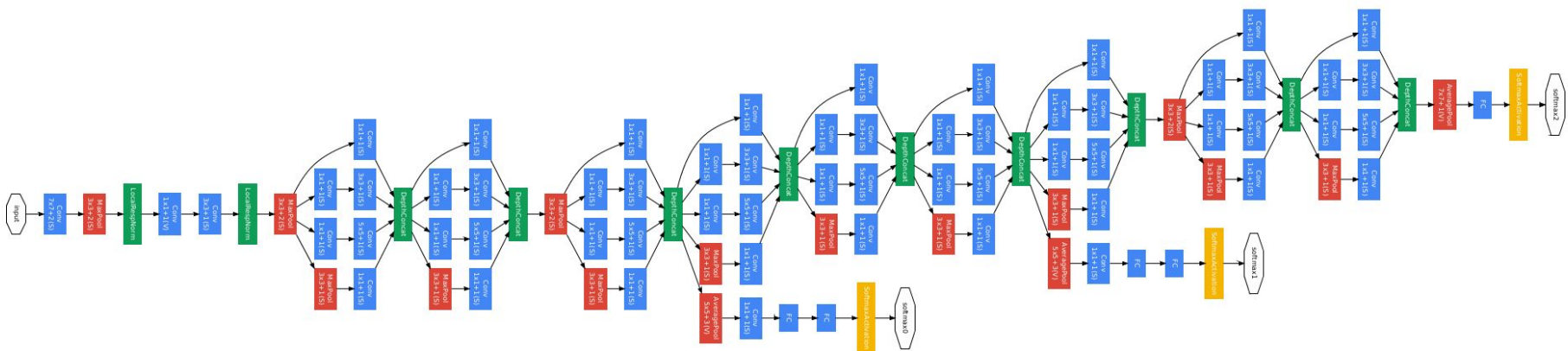
Convolutional Neural Networks

- Faster heterogeneous parallel computing
 - CPU clusters, GPUs, etc.
- Large dataset
 - ImageNet: 1.2m images of 1,000 object classes
 - CoCo: 300k images of 2m object instances
- Improvements in model architecture
 - ReLU, dropout, inception, etc.

AlexNet

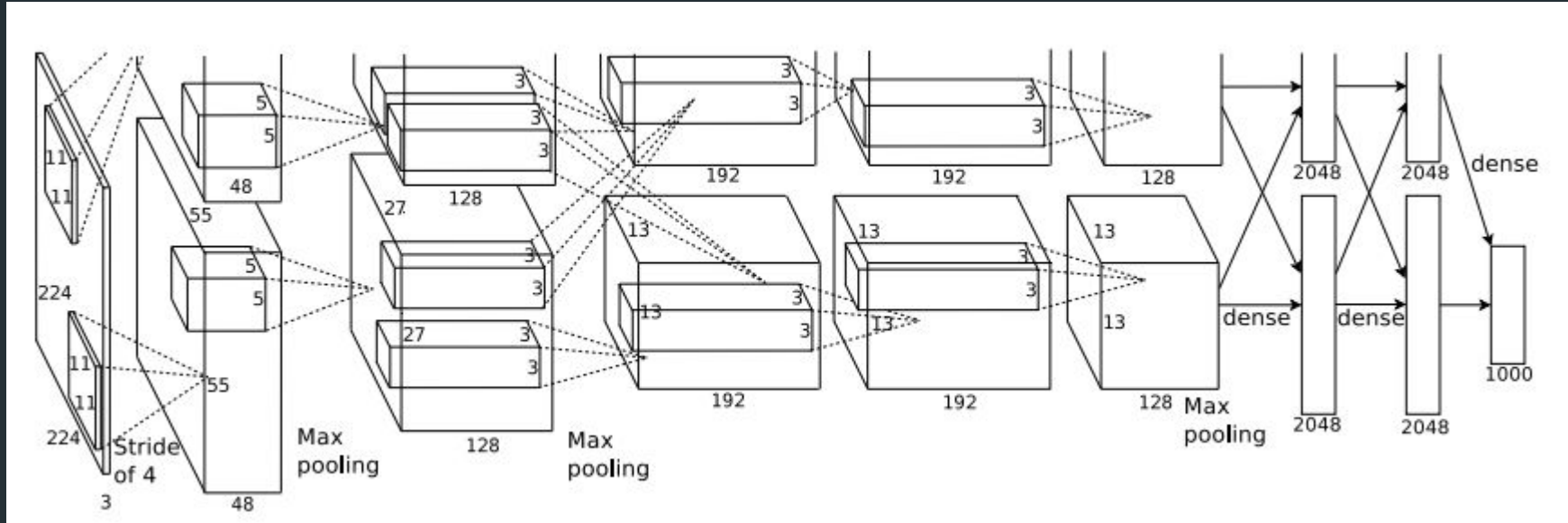


GoogLeNet



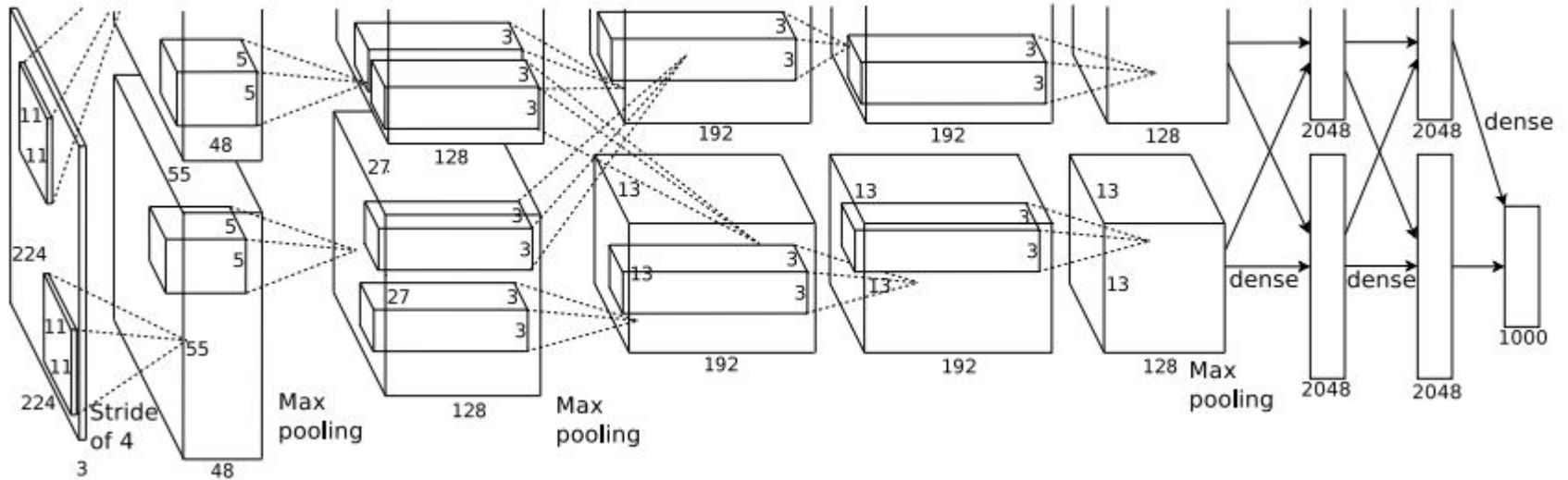
Quiz

of parameters for the first conv layer of AlexNet?



Quiz

of parameters if the first layer is fully-connected?



Quiz

Given a convolution operation written as

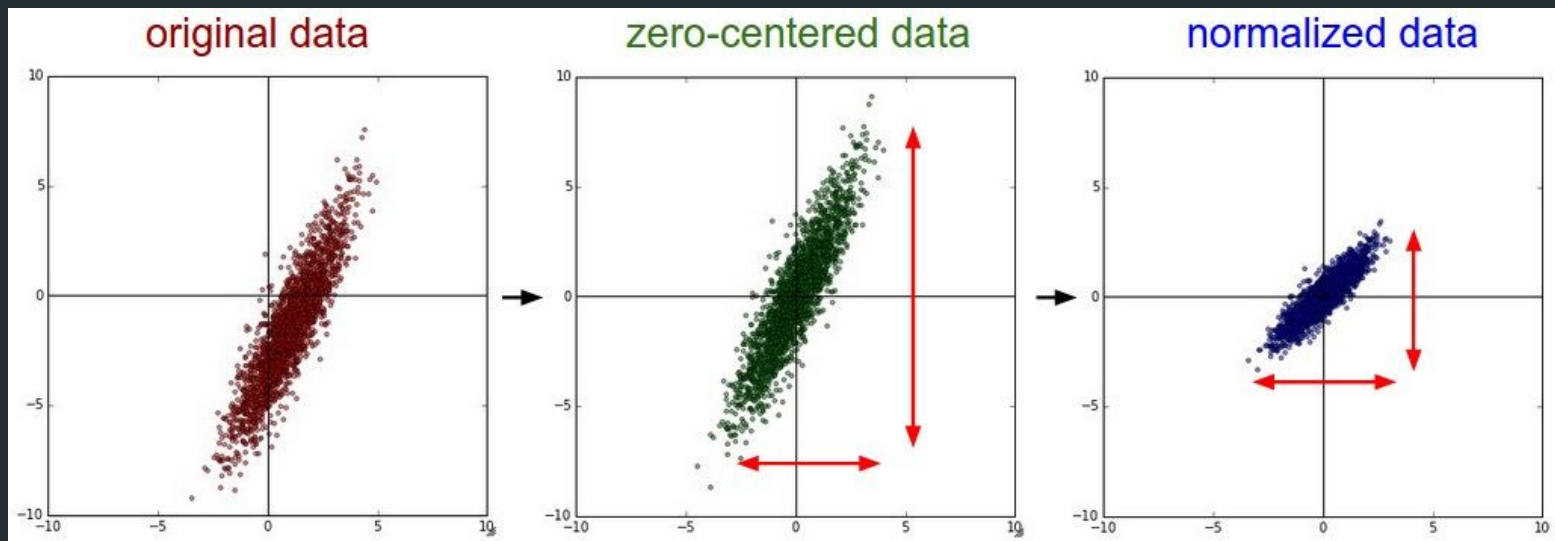
$$f(\mathbf{x}^{3 \times 3}; \mathbf{w}^{3 \times 3}, \mathbf{b}) = \sum_{i,j} (\mathbf{x}_{i,j} \mathbf{w}_{i,j}) + \mathbf{b}$$

Can you derive its gradients ($df/d\mathbf{x}$, $df/d\mathbf{w}$, $df/d\mathbf{b}$)?

Ready to Build Your Own Networks?

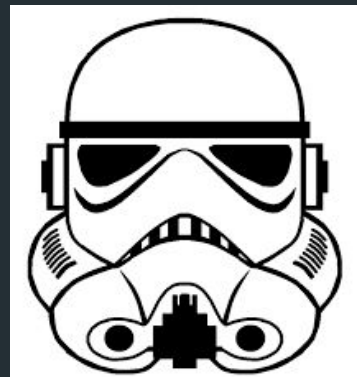
Tips and Tricks for CNNs

- Know your data, clean your data, and normalize your data
 - A common trick: subtract the mean and divide by its std.



Tips and Tricks for CNNs

- Augment your data

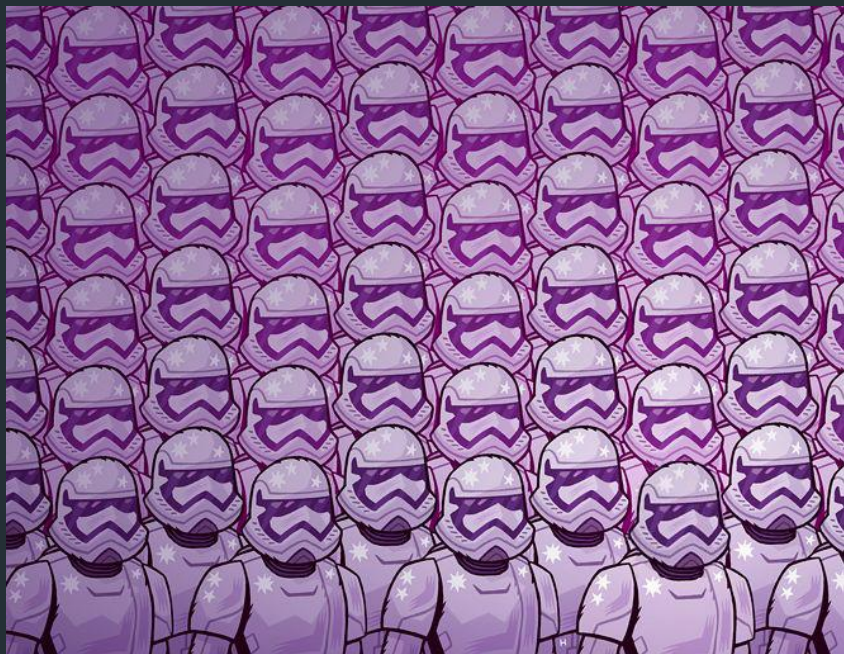


Tips and Tricks for CNNs

- Organize your data:
 - Keep training data balanced
 - Shuffle data before batching

- Feed your data in the correct way
 - Image channel order
 - Tensor storage order

Tips and Tricks for CNNs



FIRST order, IN order.



FIRST order, OUT OF order.

Tips and Tricks for CNNs

Common tensor storage order:

- **BDRC**
 - Used in Caffe, Torch, Theano, supported by CuDNN
 - Pros: faster for convolution (FFT, memory access)
- **BRCD**
 - Used in TensorFlow, limited support by CuDNN
 - Pros: Fast batch normalization, easier batching

Tips and Tricks for CNNs

Designing model architecture

- Convolution, max pooling, then fully connected layers
- Nonlinearity
 - Stay away from sigmoid (except for output)
 - ReLU preferred
 - Leaky ReLU after
 - Use Maxout if most ReLU units die (have zero activation)

Tips and Tricks for CNNs

Setting parameters

- **Weights**
 - Random initialization with proper variance
- **Biases**
 - For ReLU we prefer a small positive bias to activate ReLU

Tips and Tricks for CNNs

Setting hyperparameters

- Learning Rate / Momentum ($\Delta w^{t*} = \Delta w^t + m\Delta w^{t-1}$)
 - Decrease learning rate while training
 - Setting momentum to 0.8 - 0.9
- Batch Size
 - For large dataset: set to whatever fits your memory
 - For smaller dataset: find a tradeoff between instance randomness and gradient smoothness

Tips and Tricks for CNNs

Monitoring your training:

- Split your dataset to training, validation and test
 - Optimize your hyperparameter in val and evaluate on test
 - Keep track of training and validation loss during training
 - Do early stopping if training and validation loss diverge
 - Loss doesn't tell you all. Try precision, class-wise precision, and more

Tips and Tricks for CNNs

Borrow knowledge from another dataset

- Pre-train your CNN on a large dataset (e.g. ImageNet)
- Remove / reshape the last a few layers
- Fix the parameters of first a few layers, or make the learning rate small for them
- Fine-tune the parameters on your own dataset

Tips and Tricks for CNNs

Debugging

- import unittest, not import pdb
- Check your gradient [****deprecated****]
- Make your model large enough, and try overfitting training
- Check gradient norms, weight norms, and activation norms

Talk is Cheap, Show Me Some Code



Fully Convolutional Networks

