CMSC 491/691

Lecture 9

Neural Network Optimization

CMSC 491/691

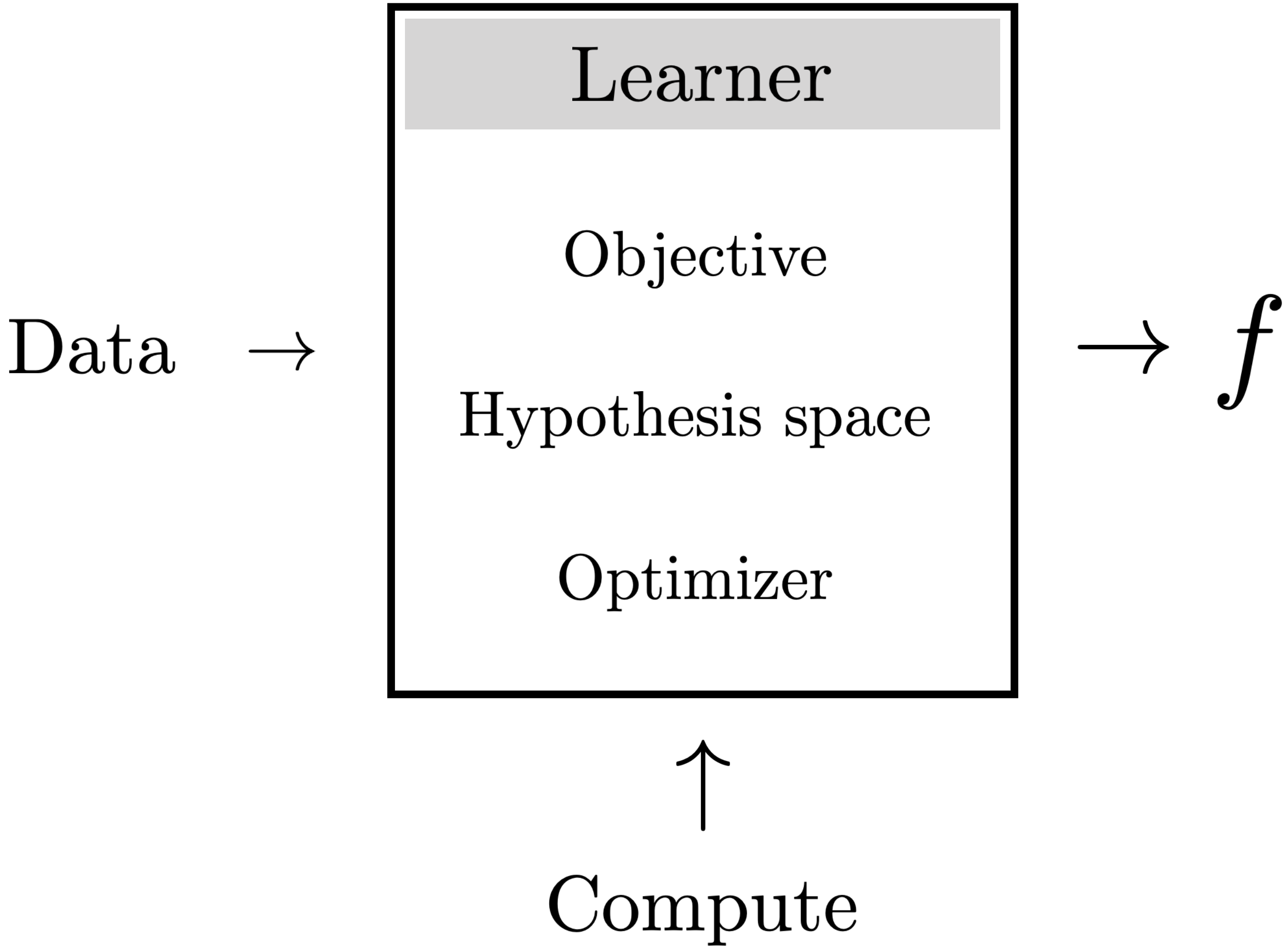
Lecture 9

Neural Network Optimization

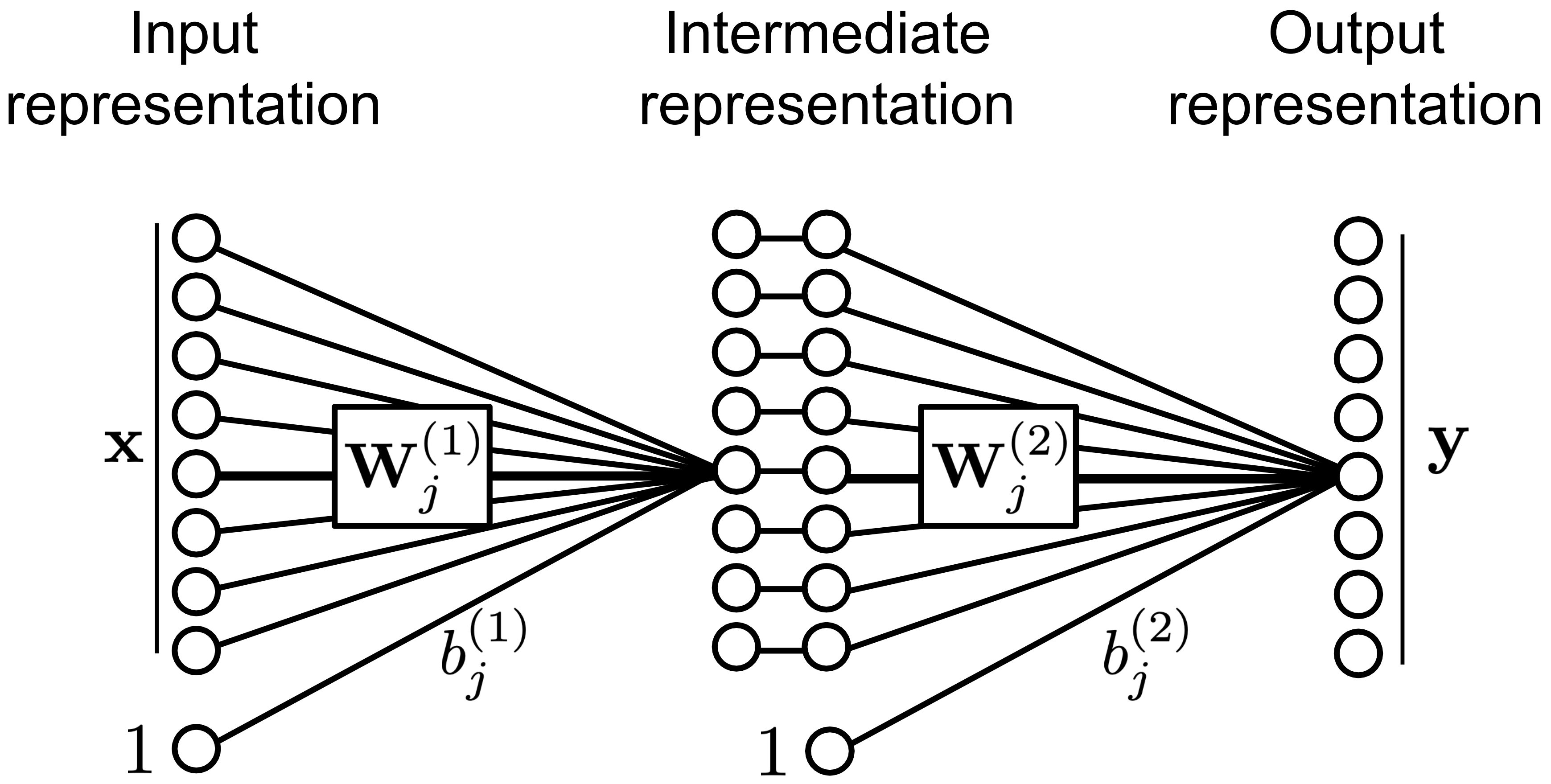
Some slides from Owens, Jayasuriya, Karpathy

PREVIOUSLY ON...

CMSC 491/691

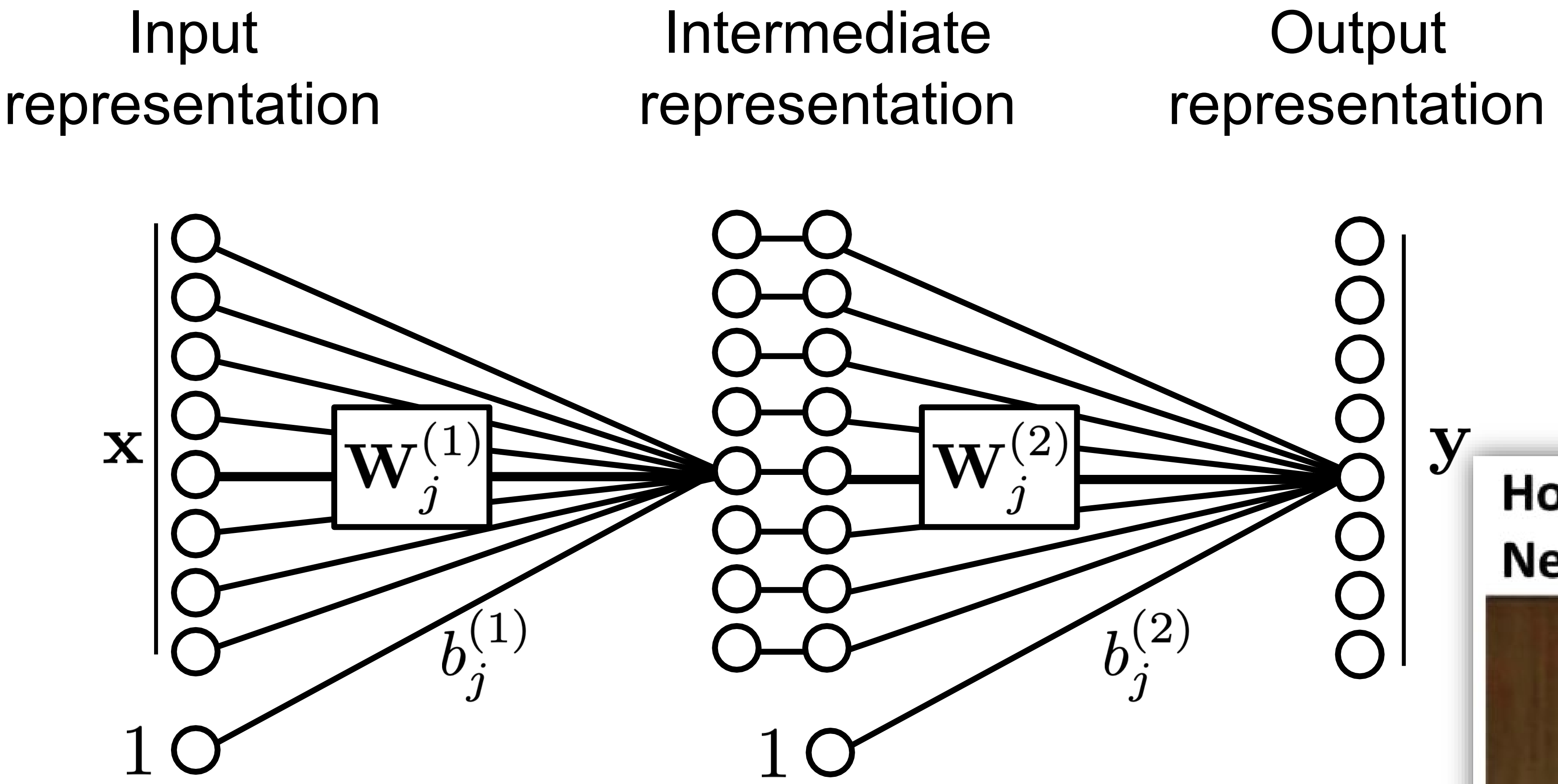

Learner
Objective
Hypothesis space
Optimizer
Data
f
Compute

Stacking layers

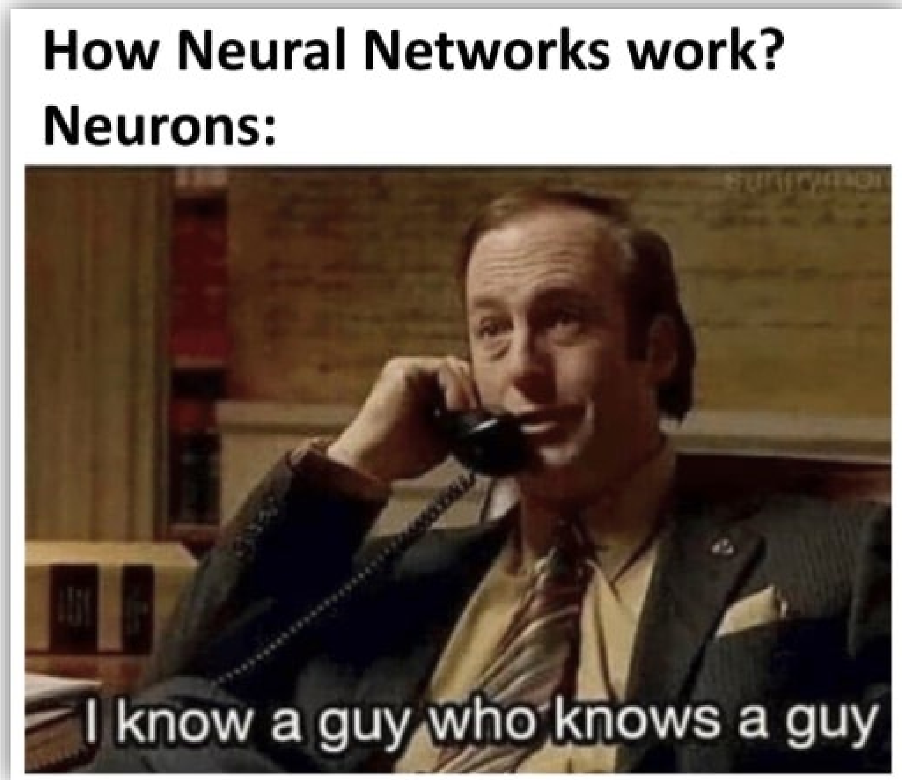


$\mathbf{h}$ = "hidden units"

Stacking layers

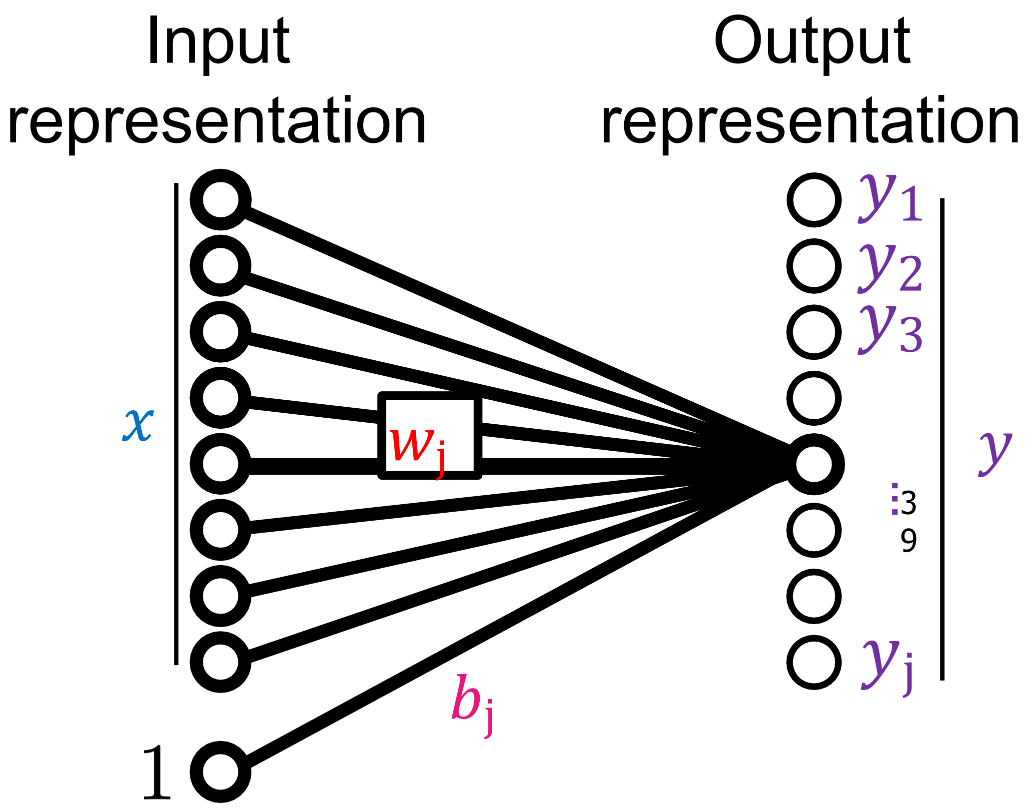


$\mathbf{h}$ = "hidden units"



Computation in a neural net – Full Layer

Linear layer

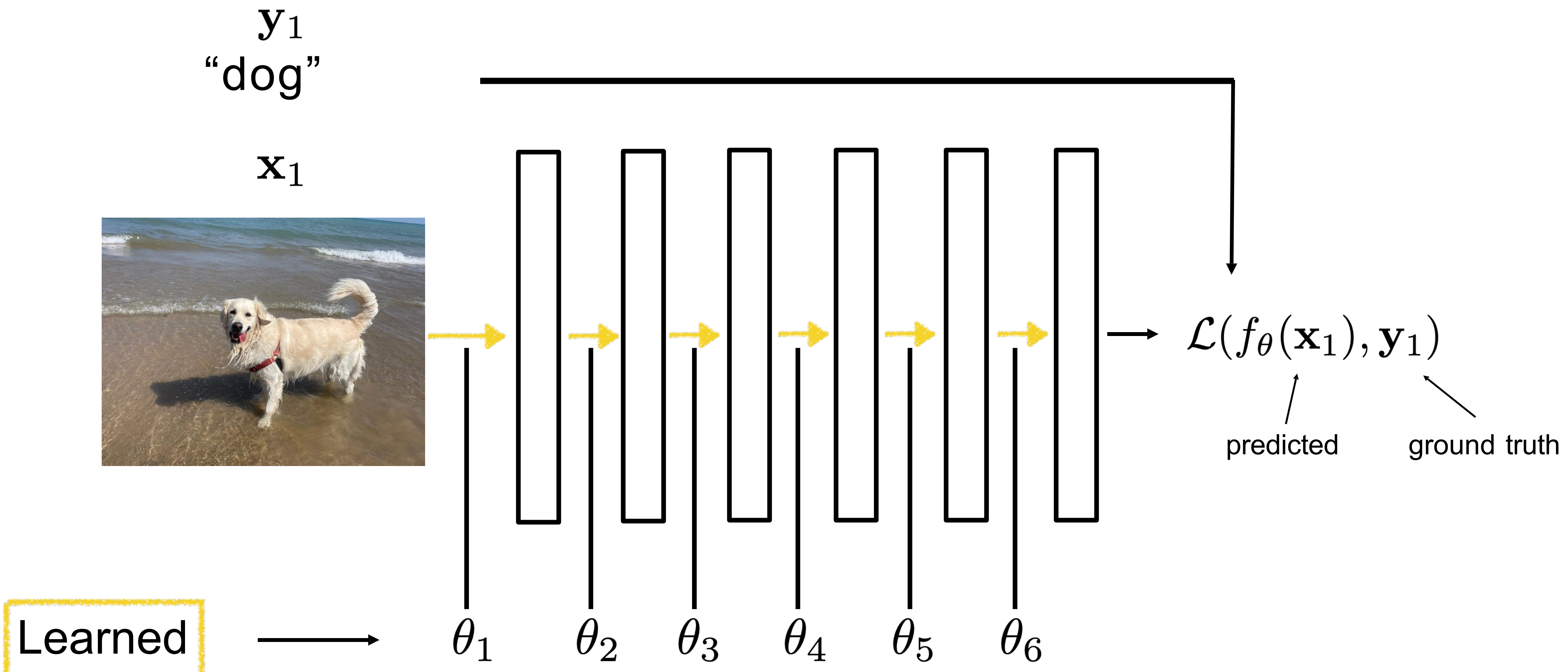


$$y = Wx + b$$

$$\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{jn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

parameters of the model: $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

How do we learn the parameters?



$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Computation has a simple form

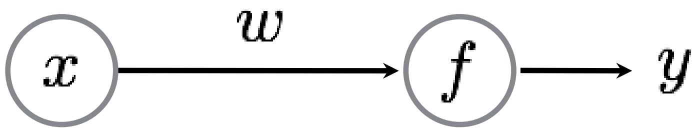
- Composition of linear functions with nonlinearities in between
- E.g. matrix multiplications with ReLU, $max(0, \mathbf{x})$ afterwards
- Do a matrix multiplication, set all negative values to 0, repeat

But where do we get the weights from?

Training neural networks

Let’s start easy

world's smallest neural network!
(a "perceptron")



$$y = wx$$

(a.k.a. line equation, linear regression)

Training a Neural Network

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

Estimate the parameter of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **‘closer’** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

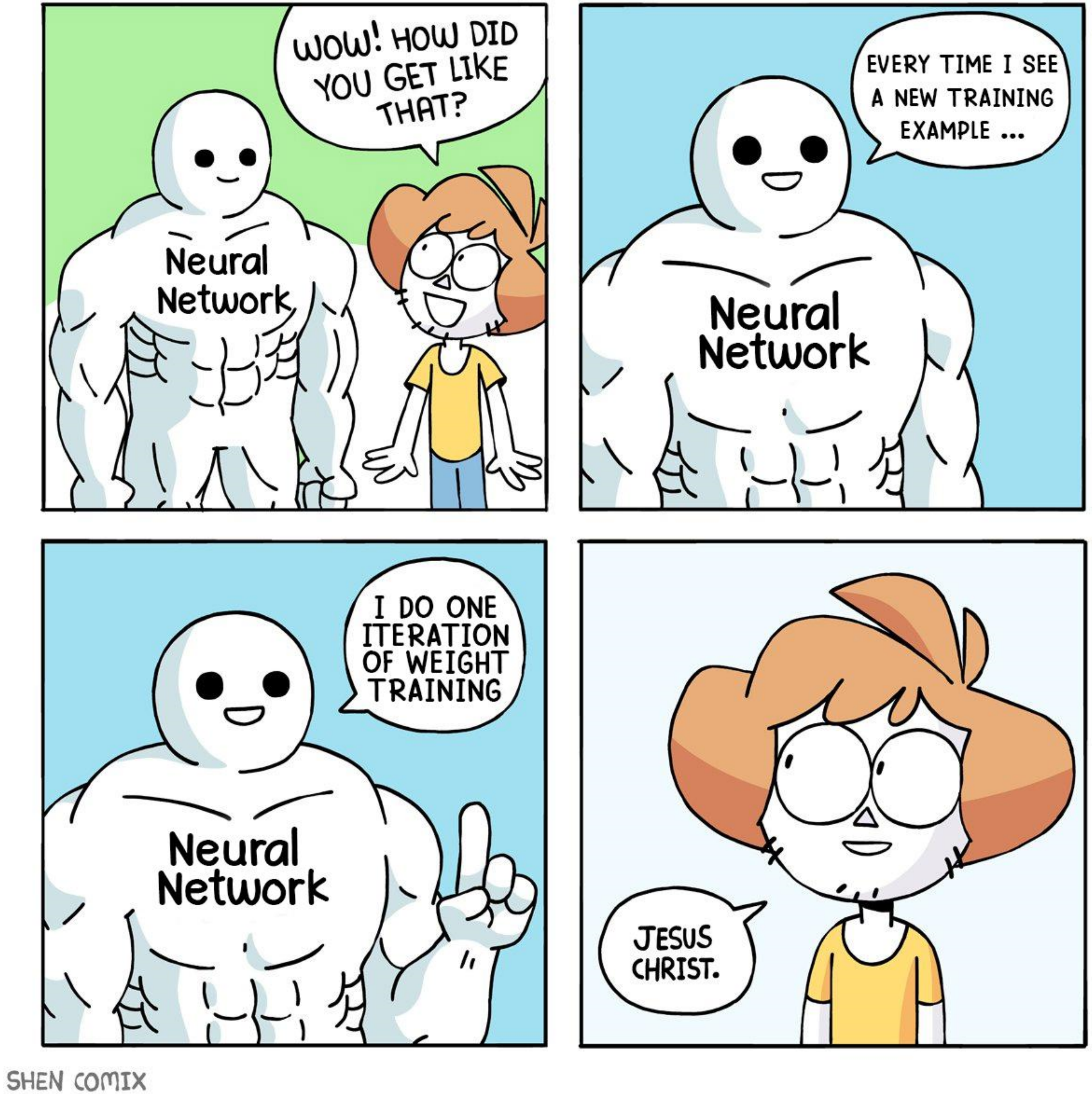
$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

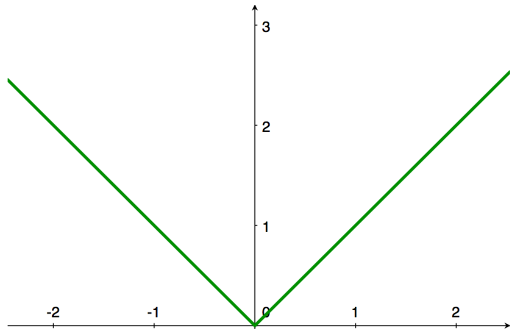
Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter **perceptron output** **true label**


WOW! HOW DID YOU GET LIKE THAT?
Neural Network
EVERY TIME I SEE A NEW TRAINING EXAMPLE ...
Neural Network
I DO ONE ITERATION OF WEIGHT TRAINING
Neural Network
JESUS CHRIST.
SHEN COMIX

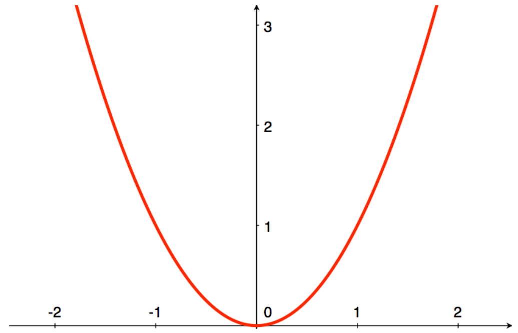
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



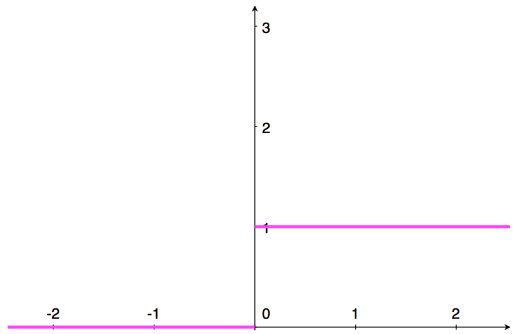
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



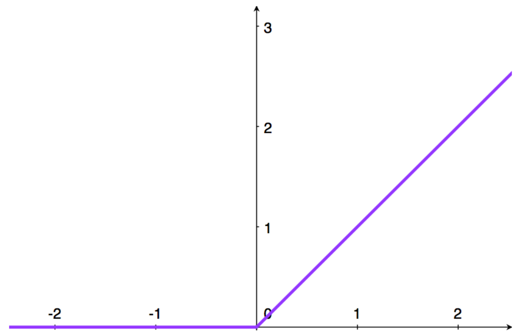
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$

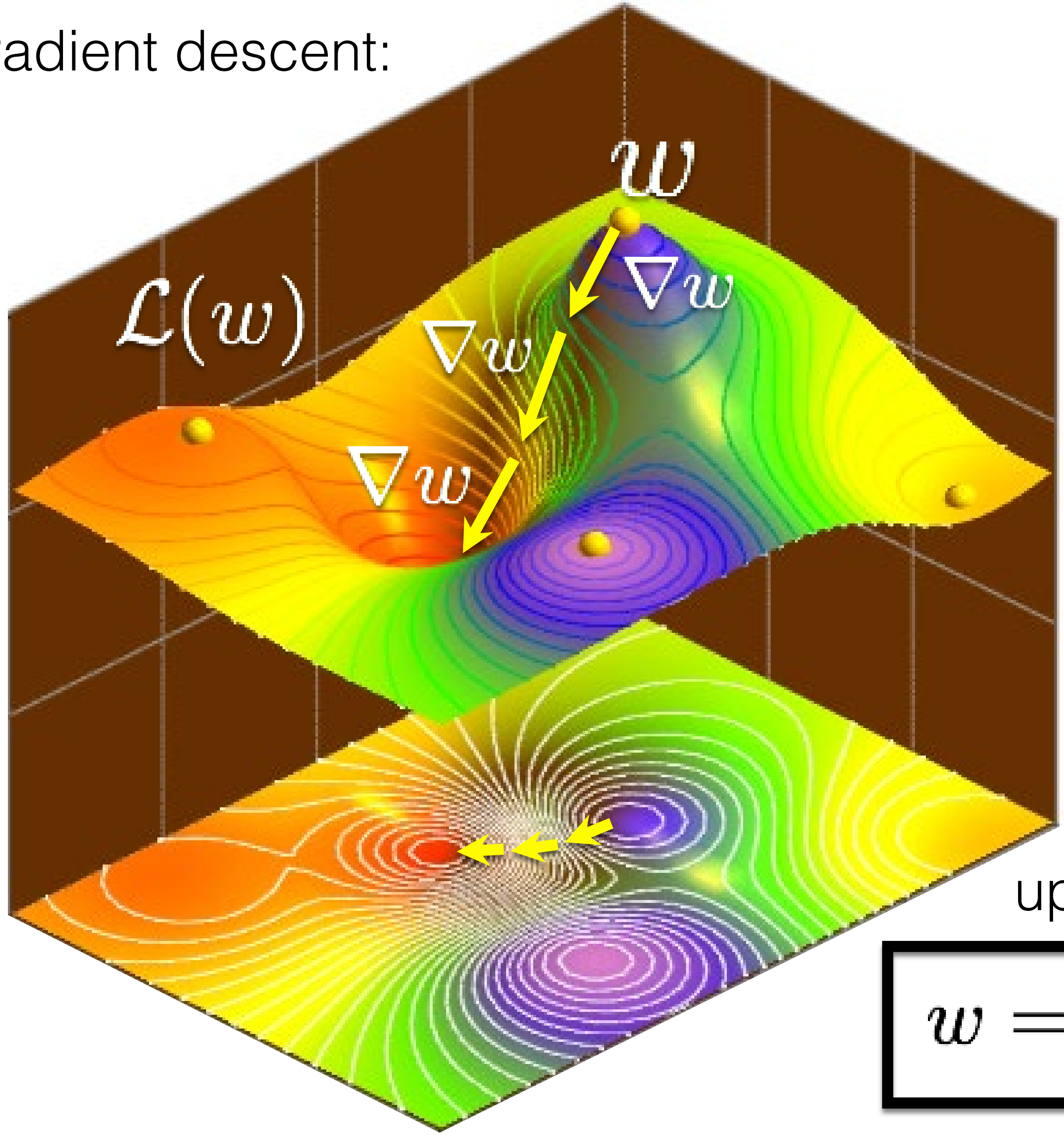


Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



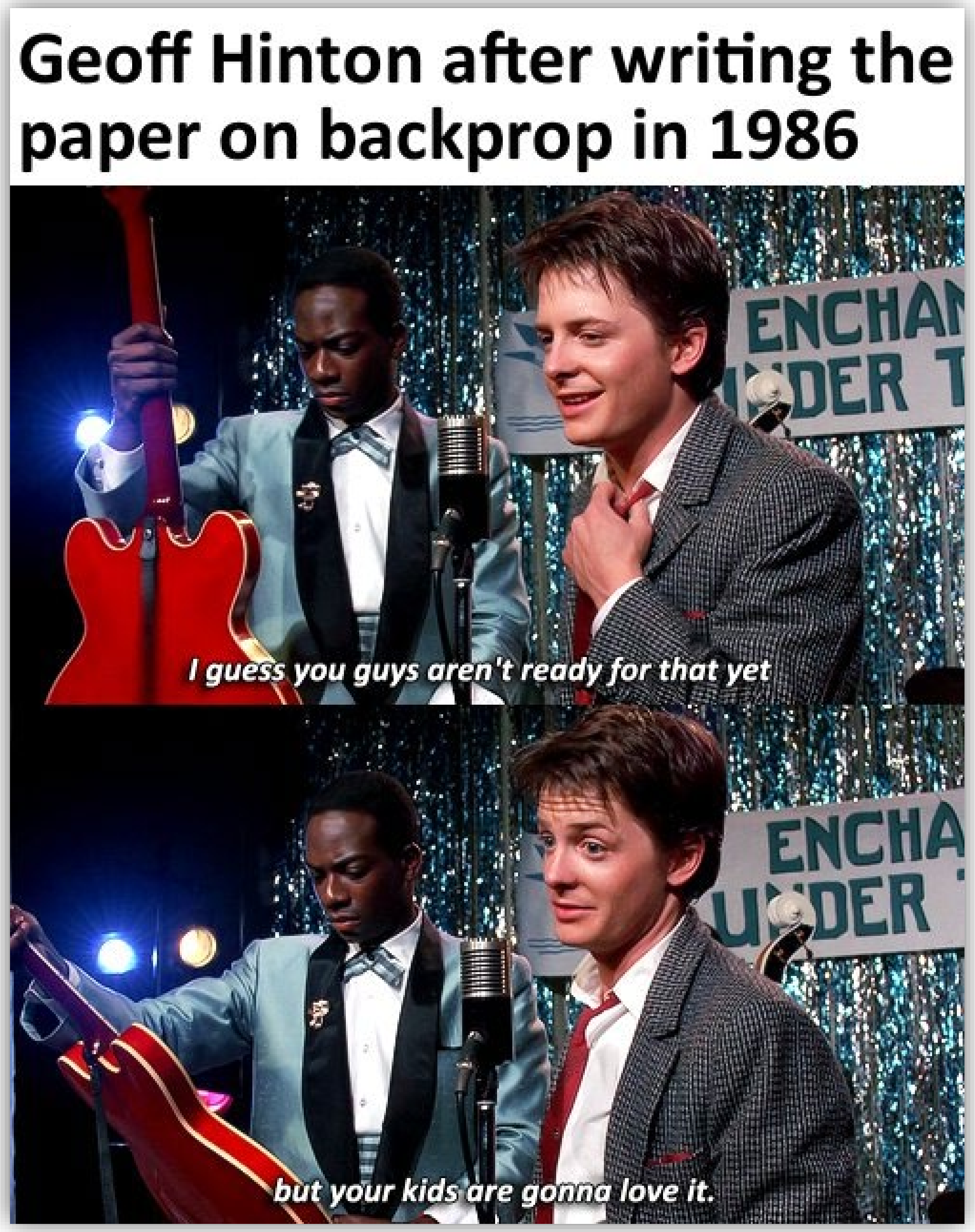
Gradient descent:



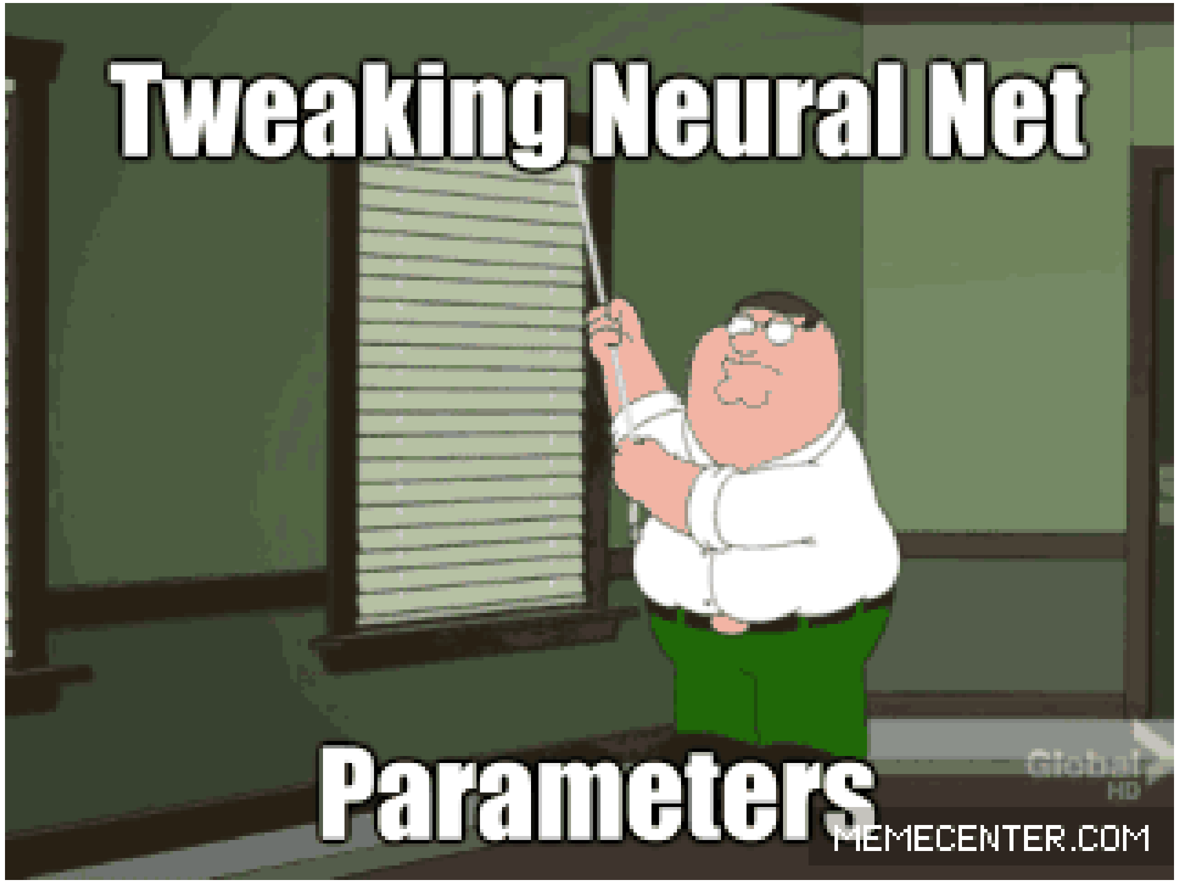
update rule:

$$\boxed{w = w - \nabla w}$$

Backpropagation



Backpropagation



$$\frac{d\mathcal{L}}{dw}$$ ...is the rate at which **this** will change...

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

the loss function

... per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative...

Compute the derivative

$$
\begin{aligned}
\frac{d\mathcal{L}}{dw} &= \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\
&= -(y-\hat{y})\frac{dwx}{dw} \\
&= -(y-\hat{y})x = \nabla w
\end{aligned}
$$

That means the weight update for **gradient descent** is:

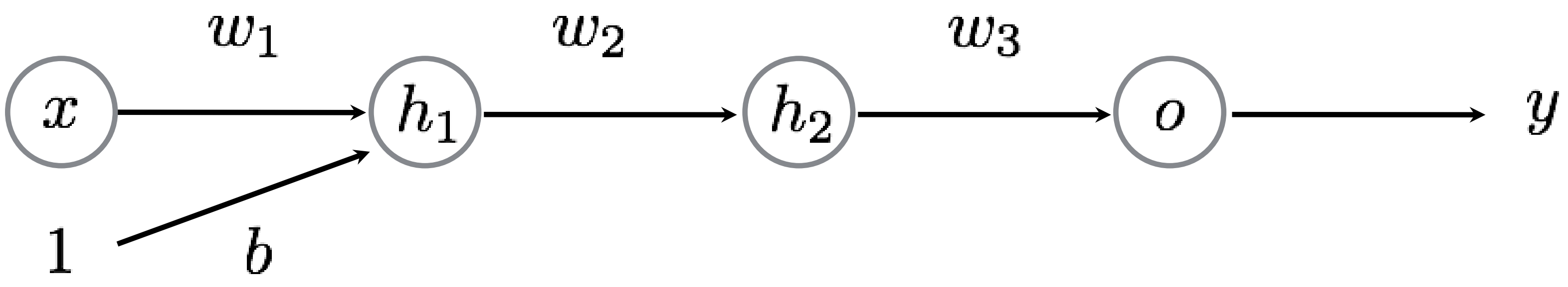
$$
\begin{aligned}
w &= w - \nabla w \quad \text{move in direction of negative gradient} \\
&= w + (y-\hat{y})x
\end{aligned}
$$

Gradient Descent (world's smallest perceptron)

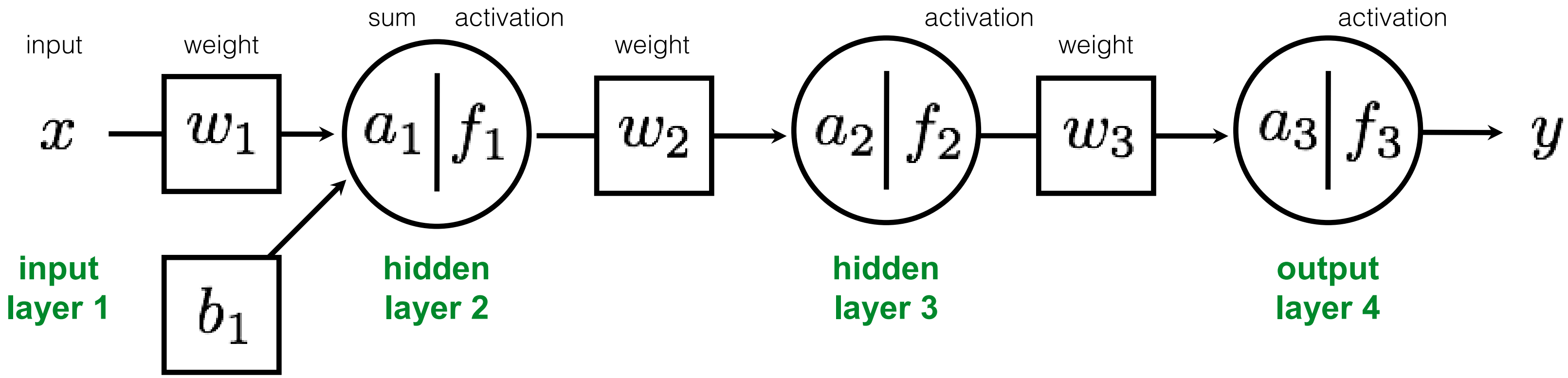
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

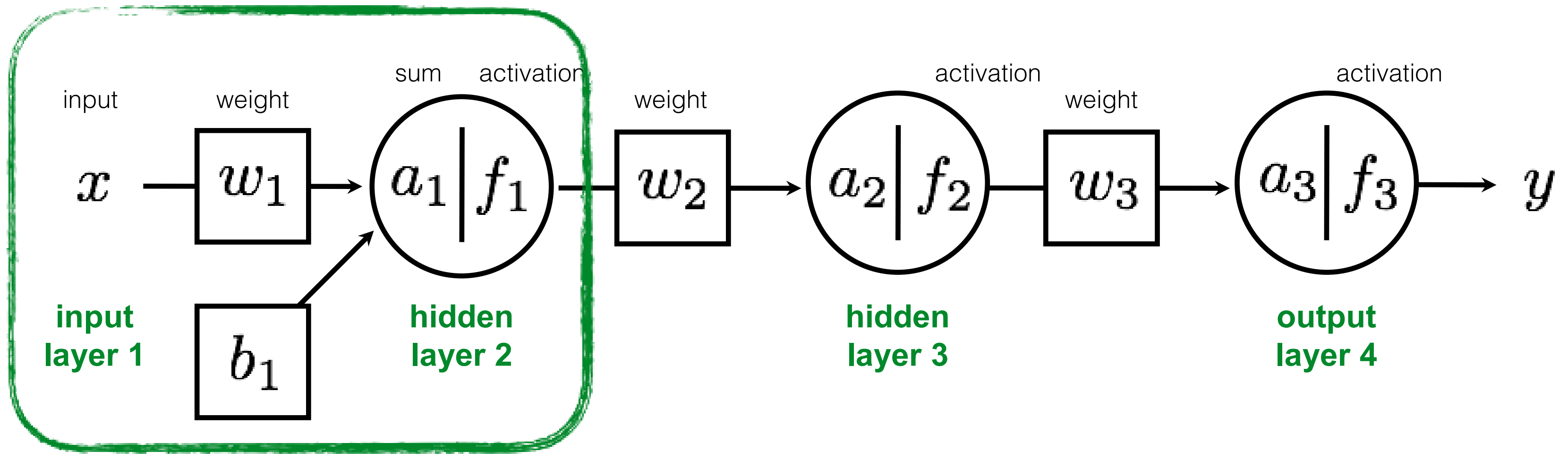
multi-layer perceptron



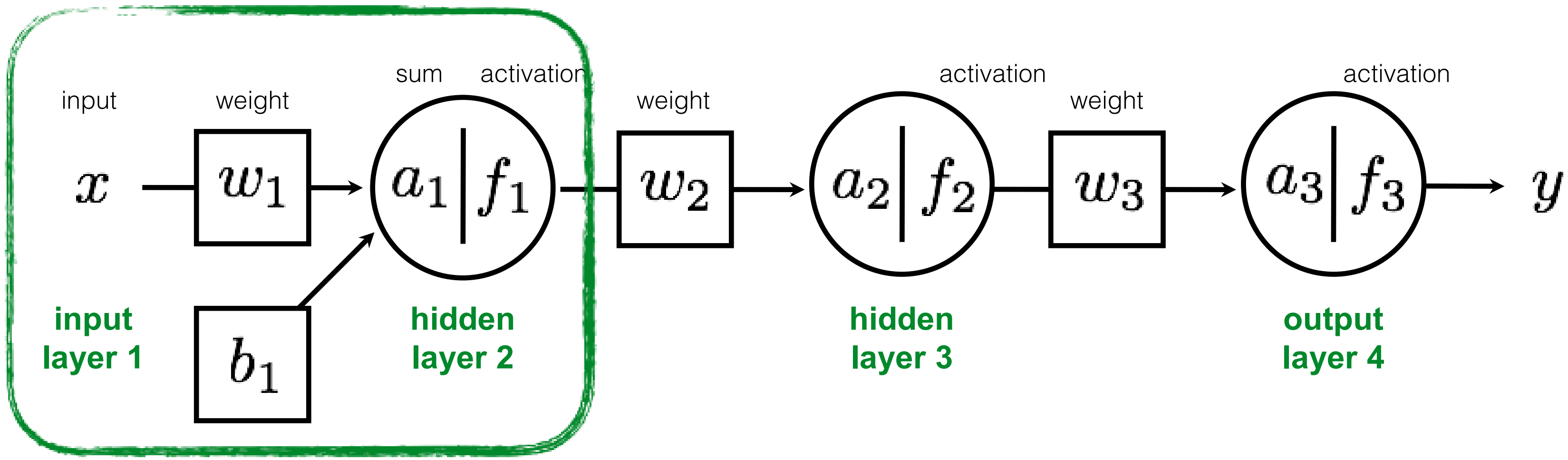
function of **FOUR** parameters and **FOUR** layers!



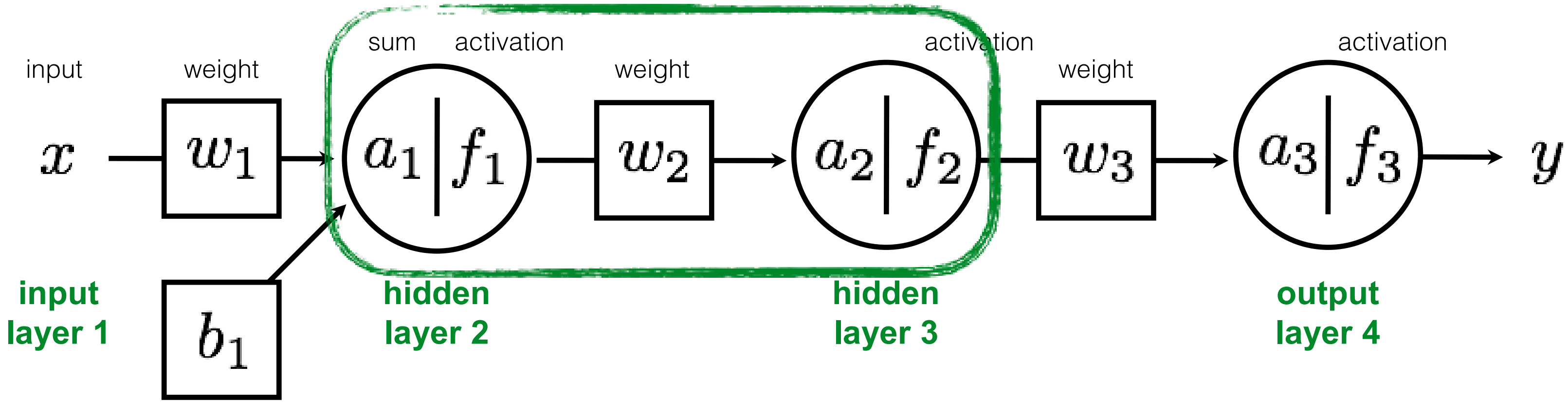
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



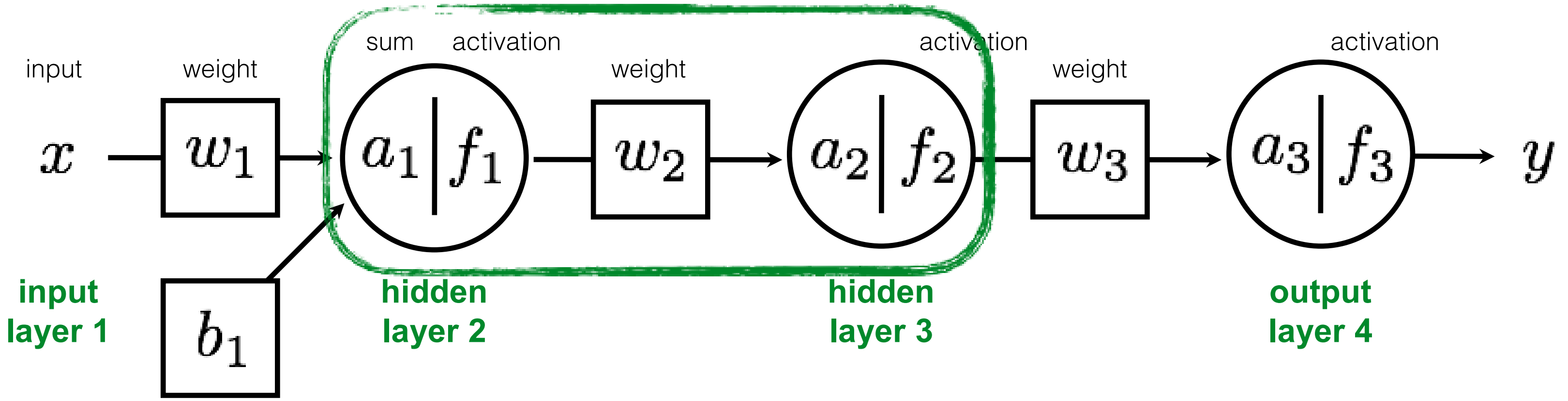
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4



$$a_1 = w_1 \cdot x + b_1$$

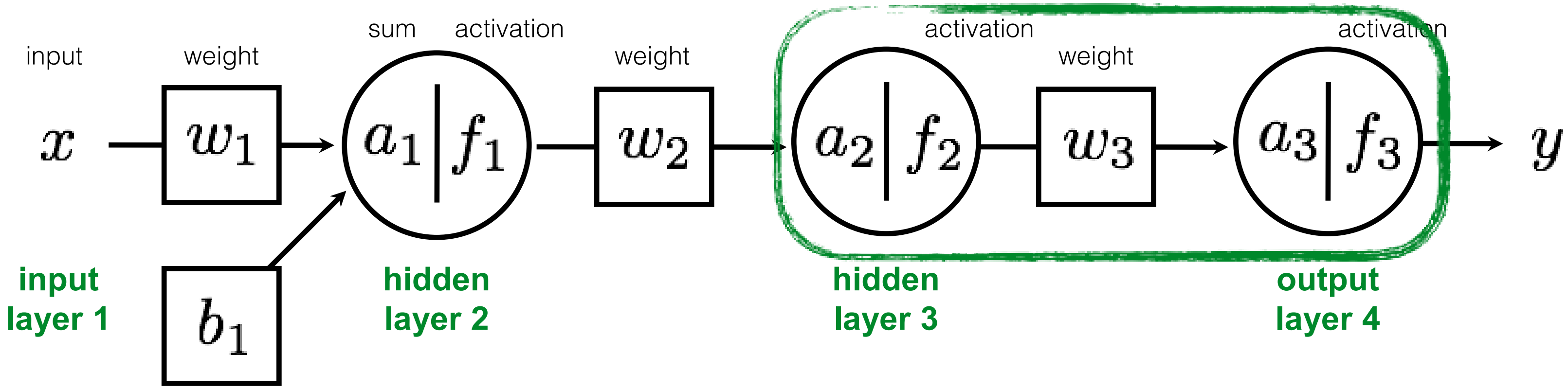


$$a_1 = w_1 \cdot x + b_1$$



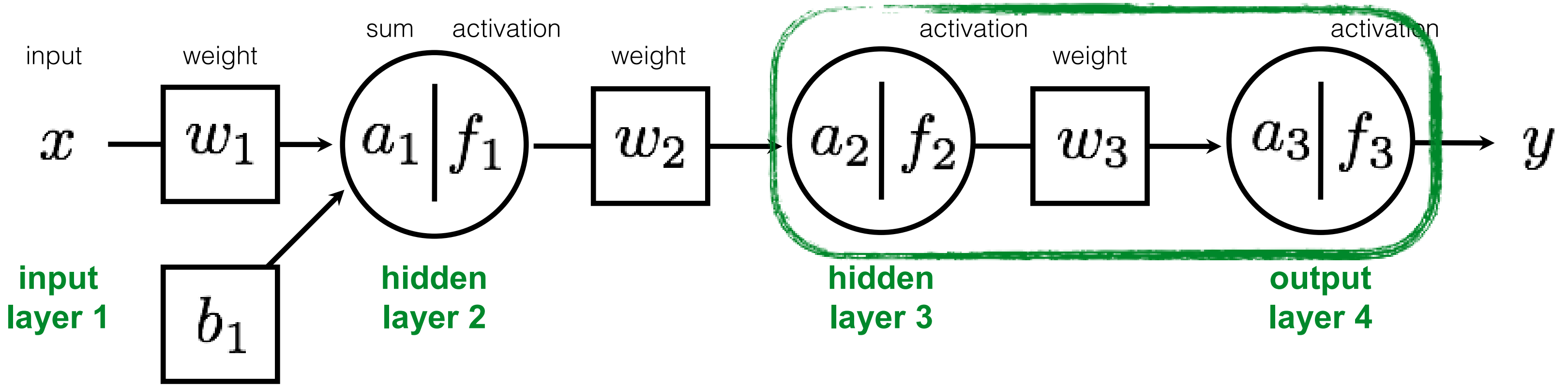
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

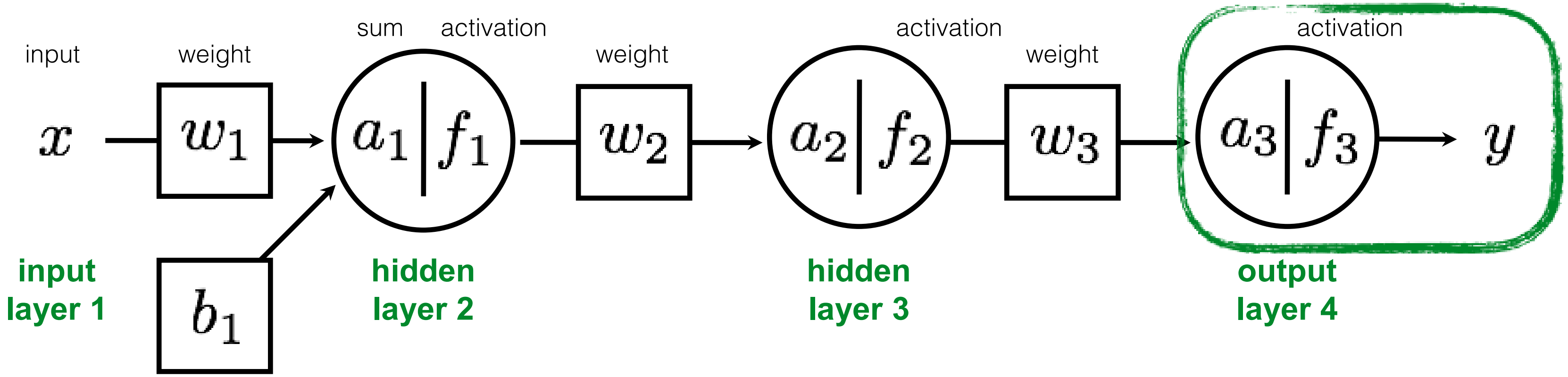
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

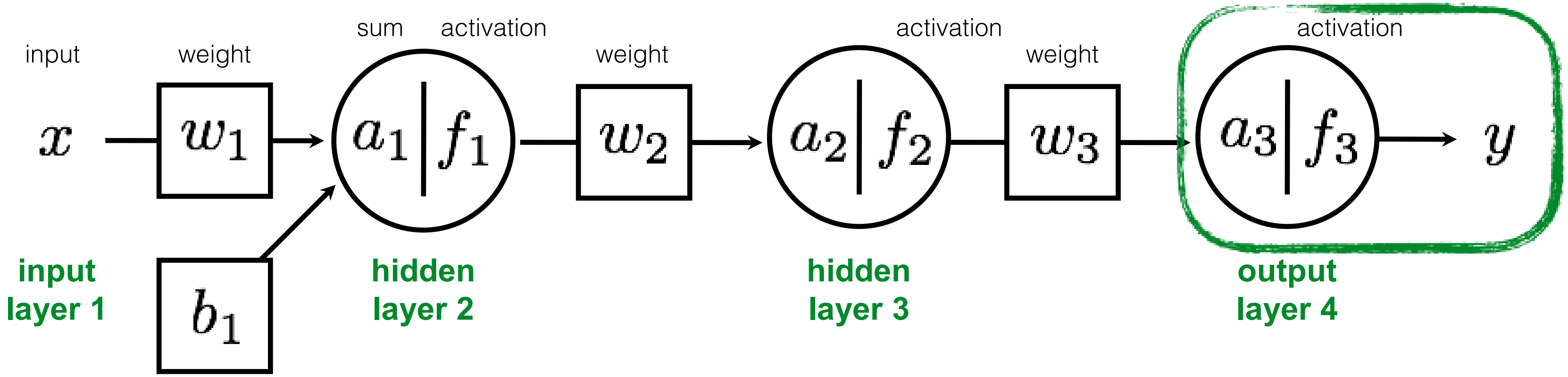
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has
parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\mathrm{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

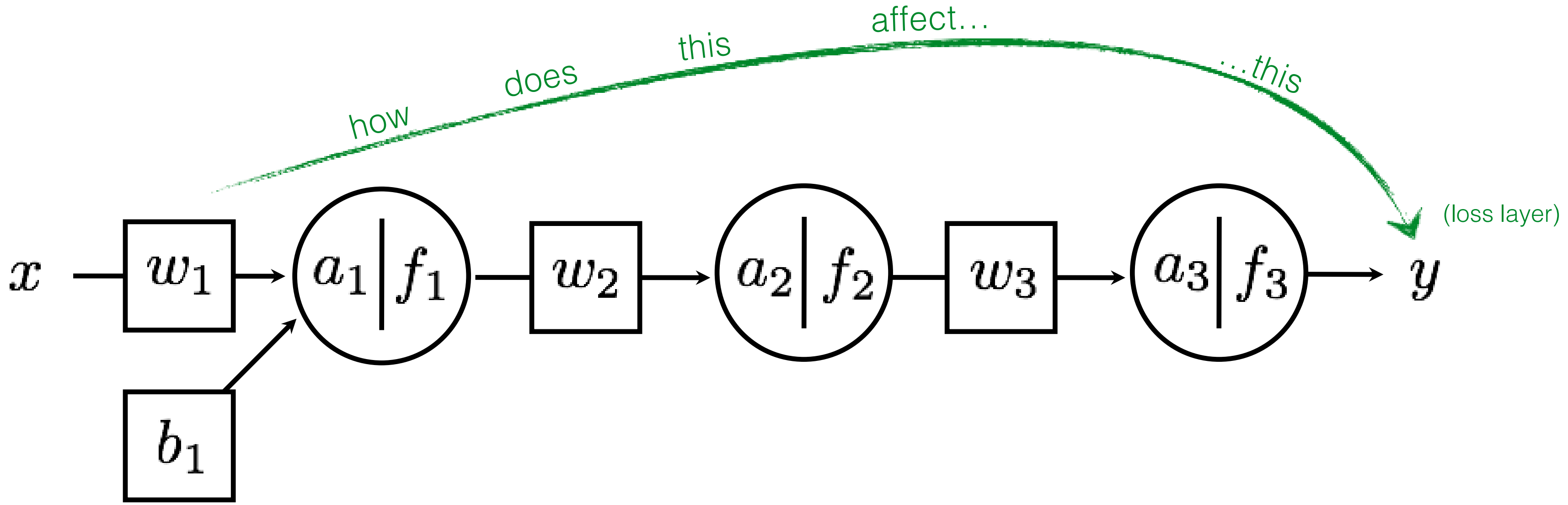
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember,

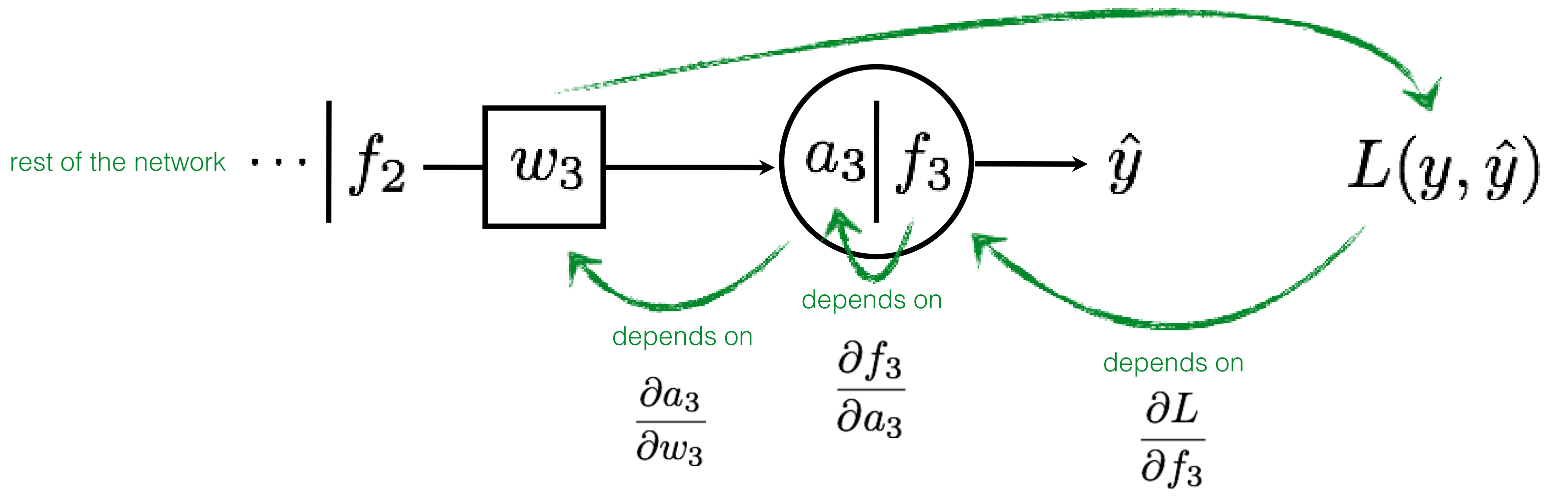
Partial derivative $\dfrac{\partial L}{\partial w_1}$ describes…

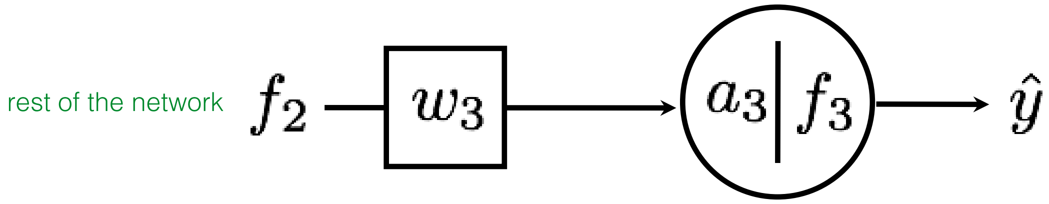


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$

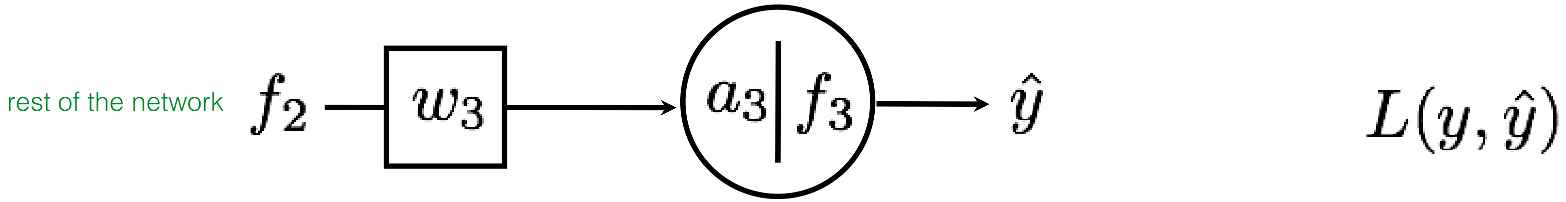




$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

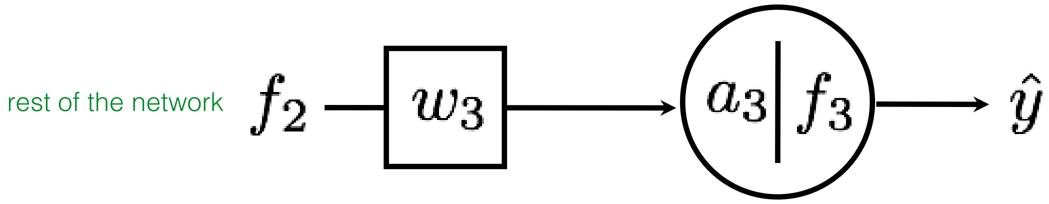
Chain Rule!



$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss



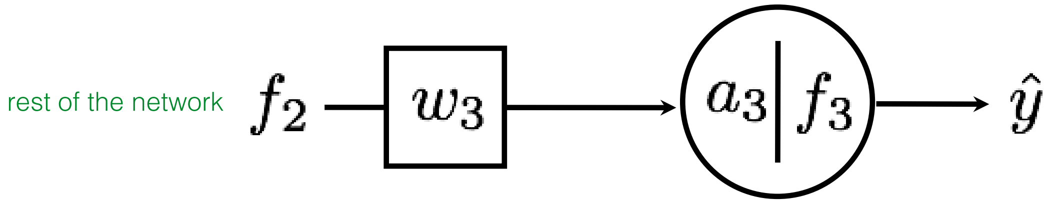
$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

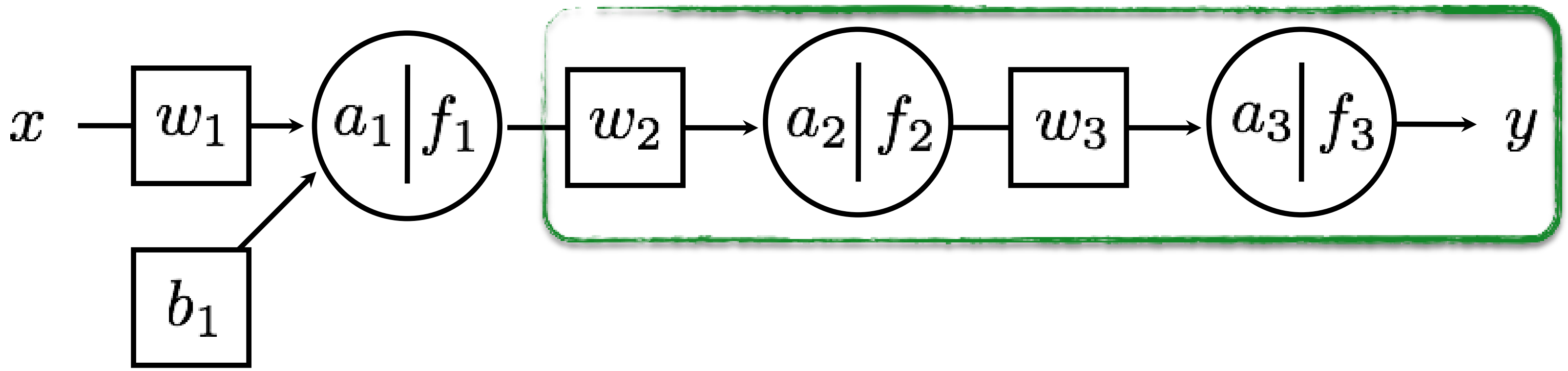
Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

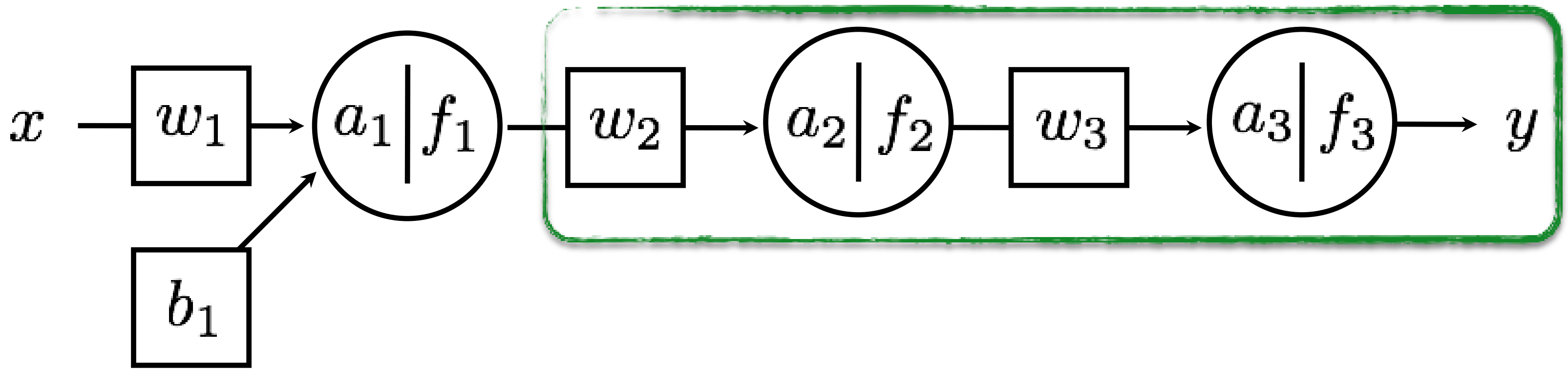


$$L(y, \hat{y})$$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)f_2
\end{aligned}
$$



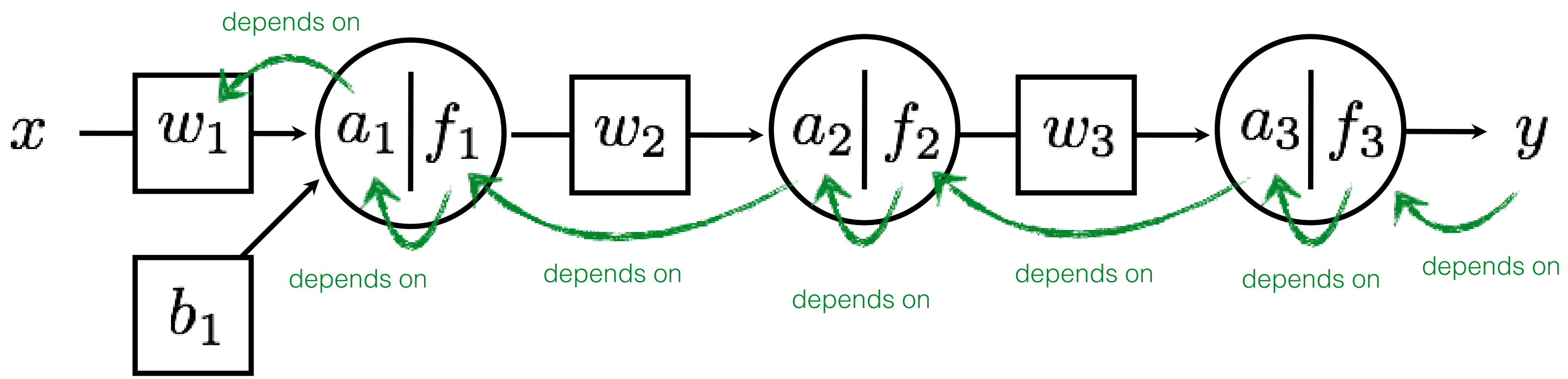
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

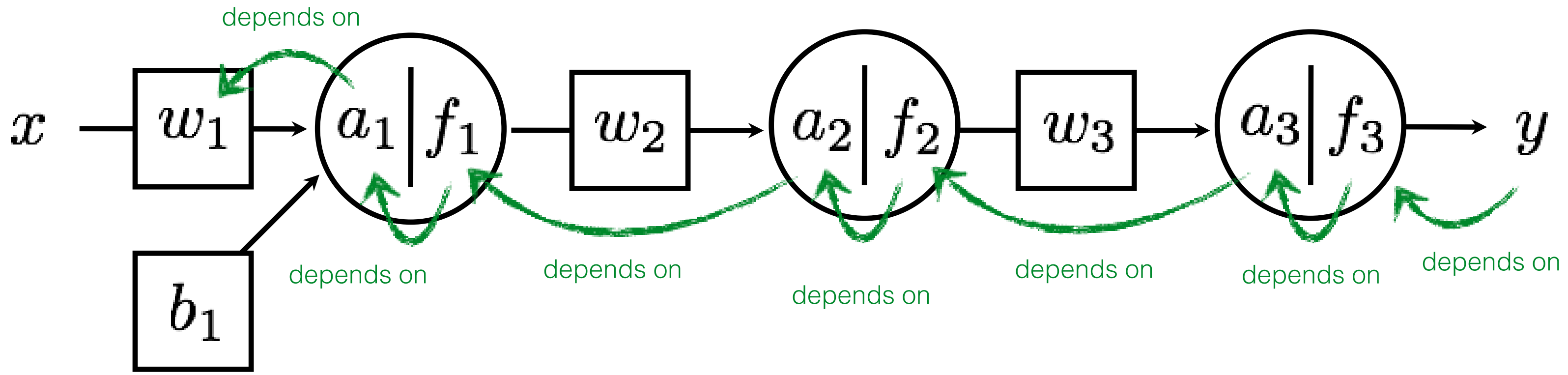
already computed.
re-use (propagate)!

The chain rule says…



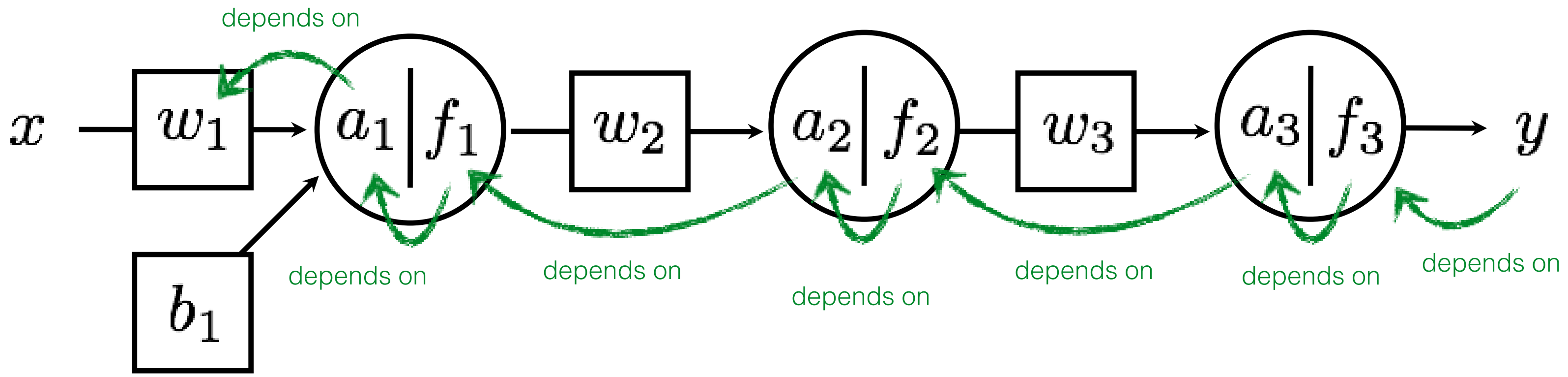
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

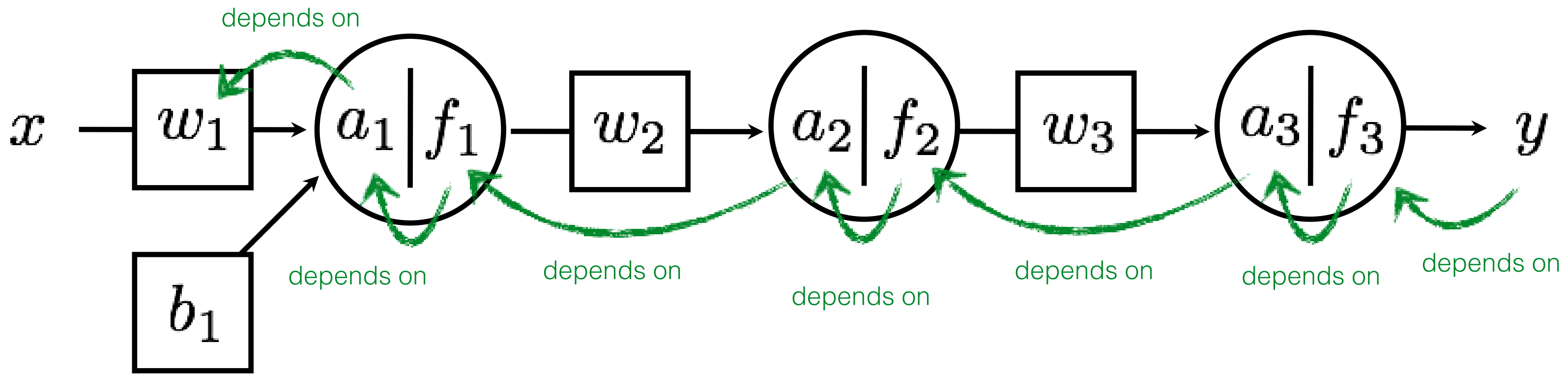


$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3}} \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3}} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}$$

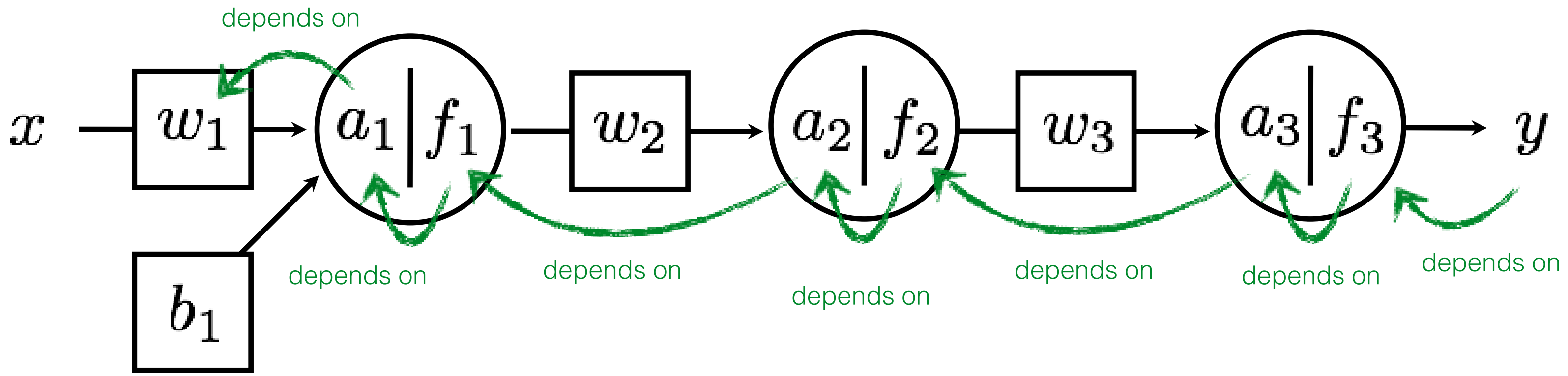


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\dfrac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta - \eta \dfrac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

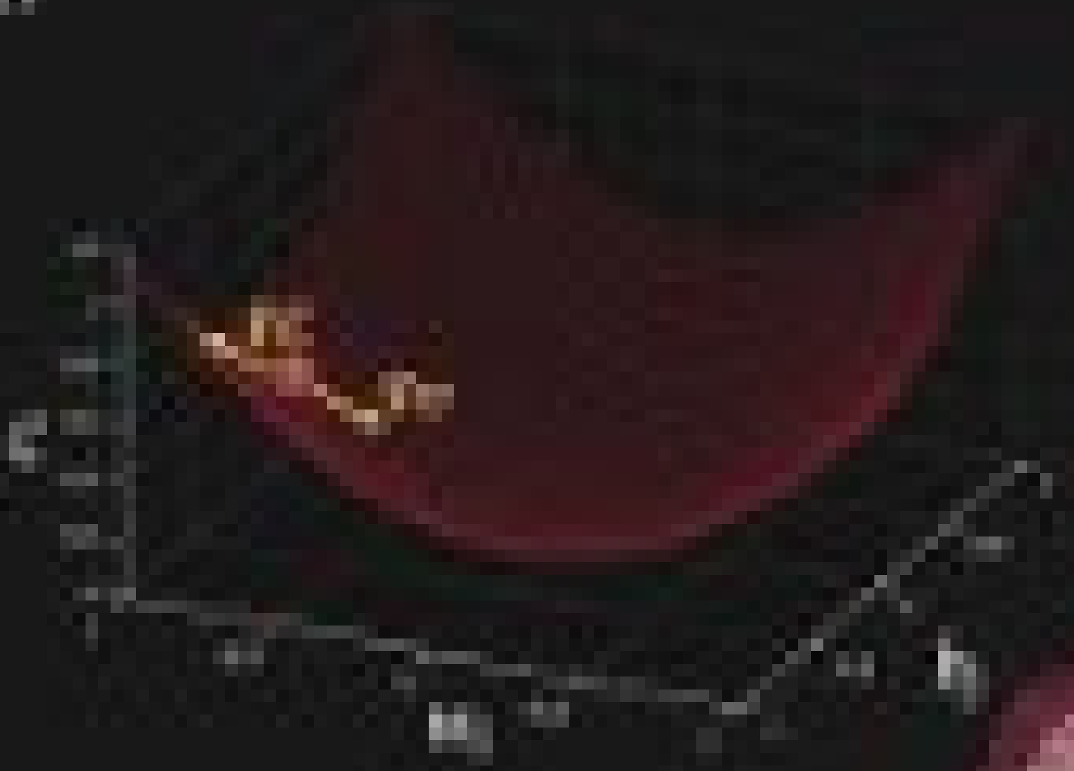
Gradient Descent

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial v_1} & \frac{\partial C}{\partial v_2} \end{bmatrix}^T \qquad \Delta v = \begin{bmatrix} \Delta v_1 \\ \Delta v_2 \end{bmatrix}$$

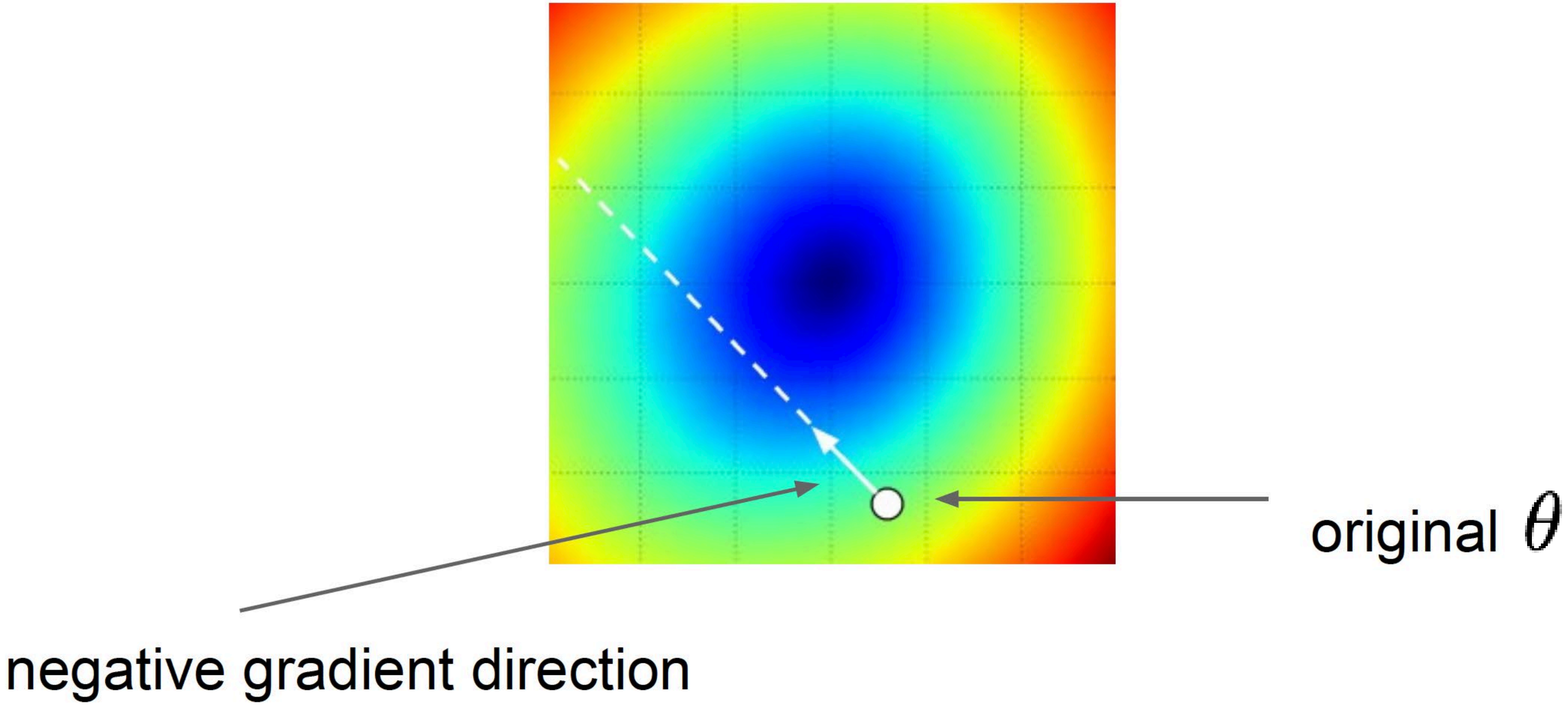
$$\Delta C \approx \nabla C \cdot \Delta v$$

Let $\Delta v = -\eta \nabla C$

$$\Delta C = -\eta \|\nabla C\|^2$$



Learning rates



$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate
Too big: will miss the minimum
Too small: slow convergence

Learning rate scheduling

- Use different learning rate at each iteration.
- Most common choice:

$$\eta_t = \frac{\eta_0}{\sqrt{t}}$$

Need to select initial learning rate η_0

More modern choice: Adaptive learning rates.

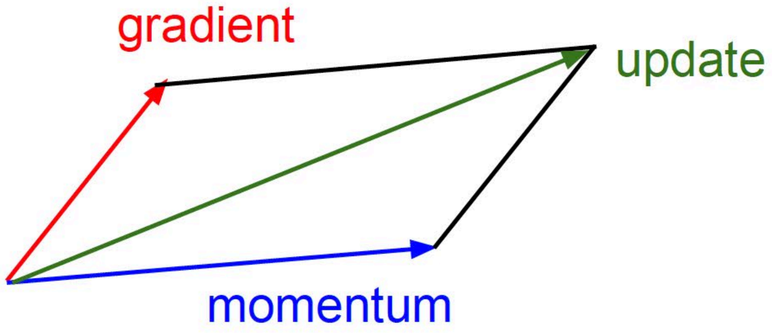
$$\eta_t = G\left(\left\{\frac{\partial L}{\partial \theta}\right\}_{i=0}^{t}\right)$$

Many choices for G (Adam, Adagrad, Adadelta).

Momentum Update

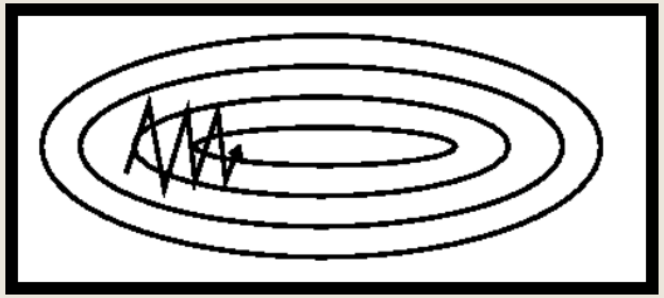
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

$$\Delta\theta \leftarrow w \frac{\partial L}{\partial \theta} + (1 - w)\Delta\theta$$

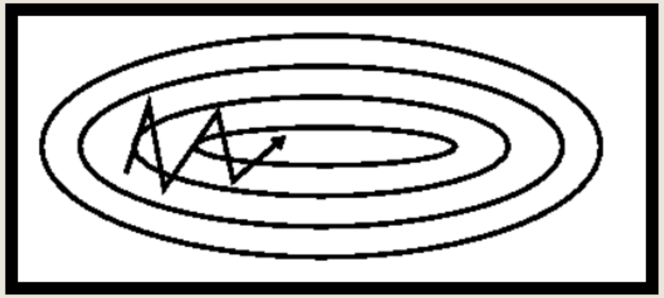


Take direction history into account!

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



(Fig. 2a)



(Fig. 2b)

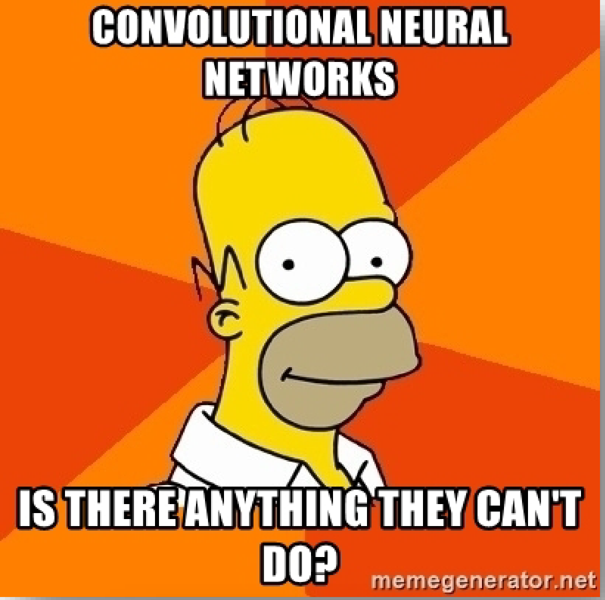
Many other ways to perform optimization…

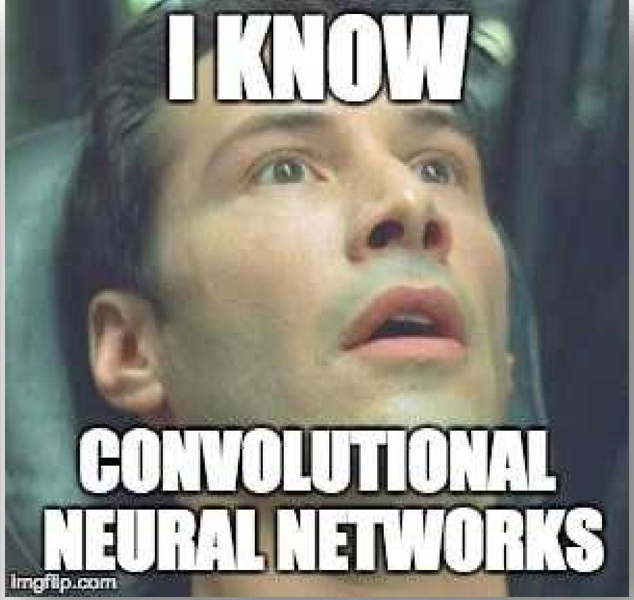
- Second order methods that use the Hessian (or its approximation): BFGS, **LBFGS**, etc.

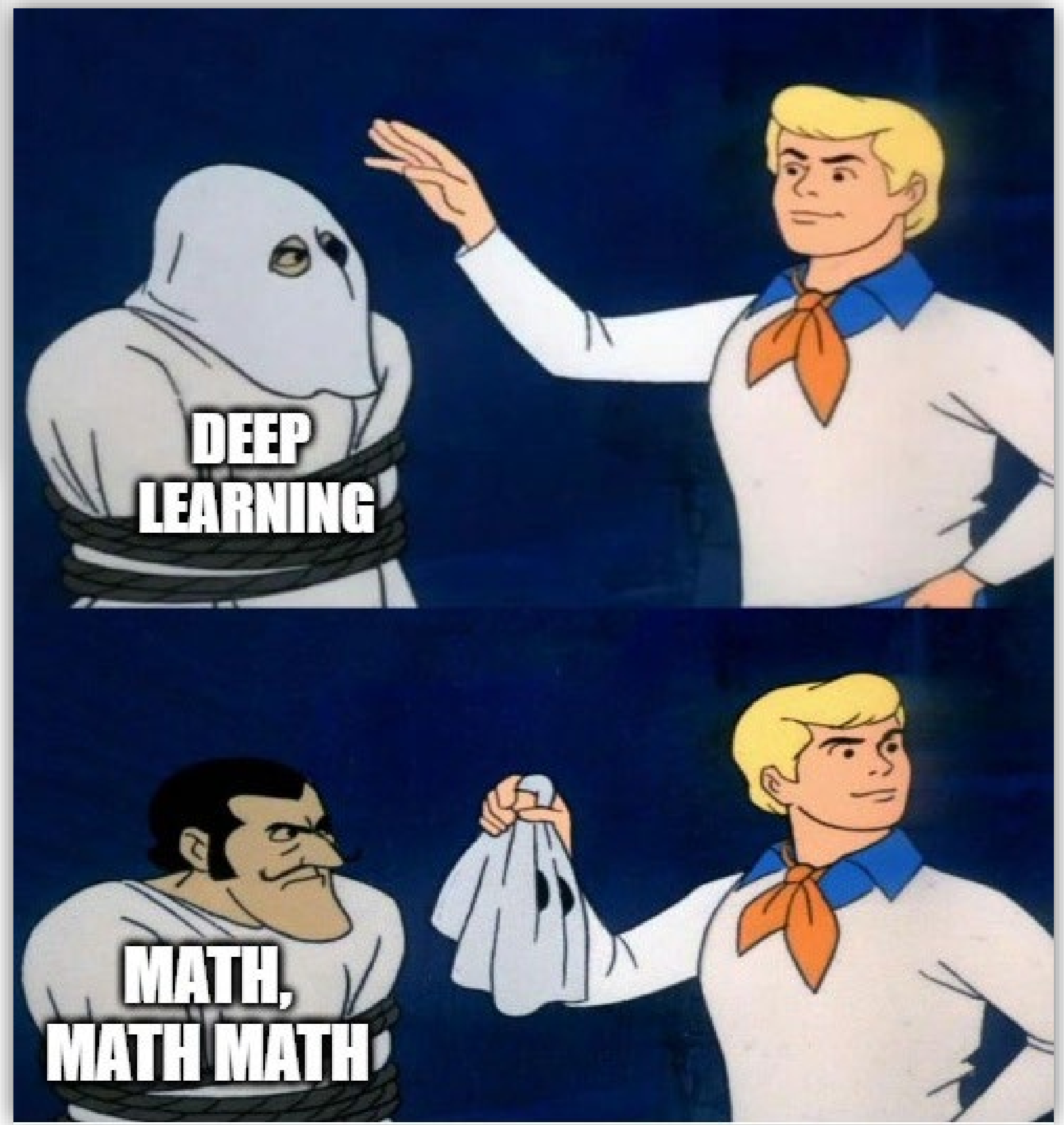
- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for CNNs.

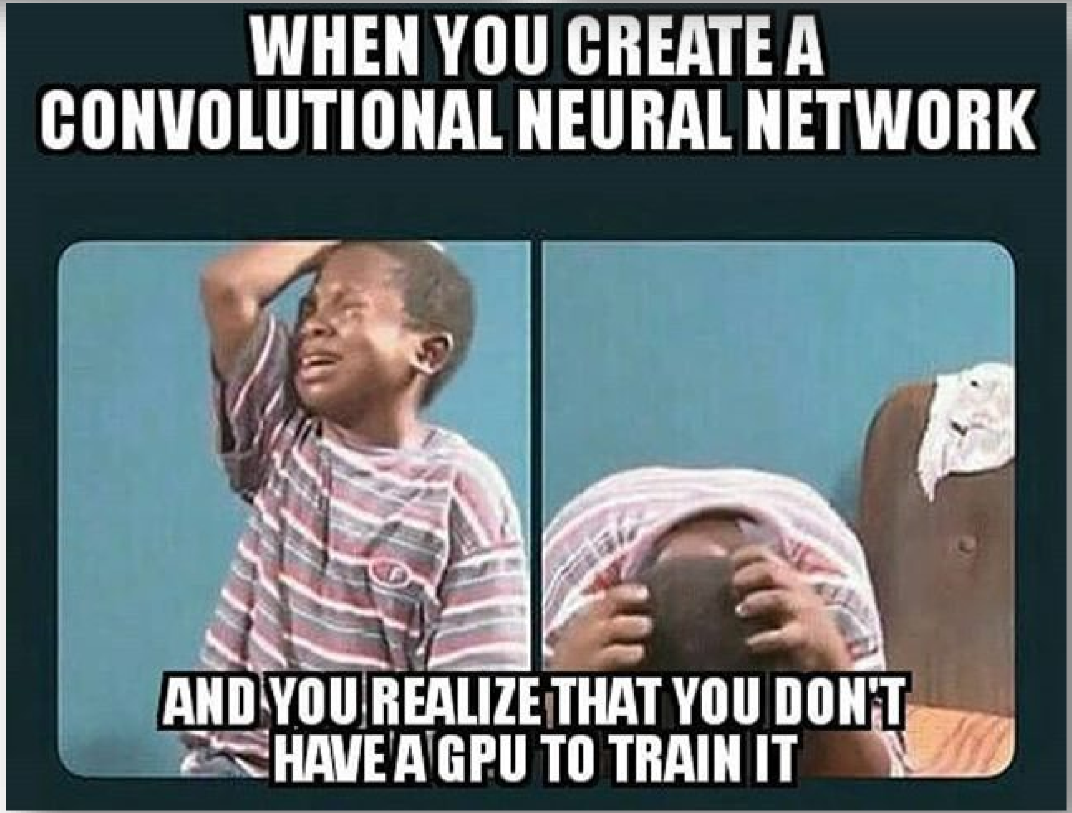
- No consensus on Adam etc.: Seem to give faster performance to worse local minima.

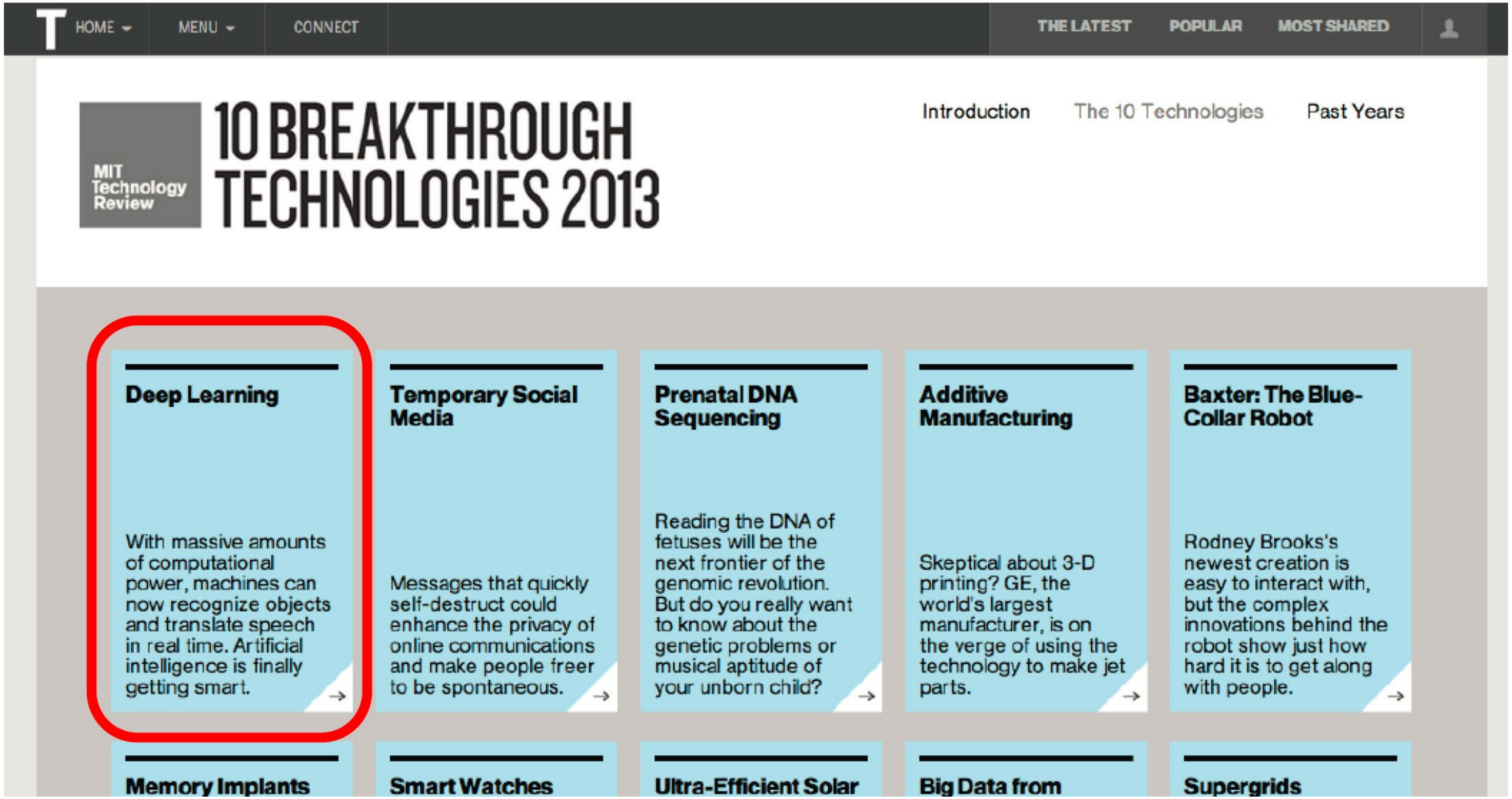
Convolutional Neural Networks


CONVOLUTIONAL NEURAL NETWORKS
IS THERE ANYTHING THEY CAN'T DO?
memegenerator.net


I KNOW
CONVOLUTIONAL NEURAL NETWORKS
imgflip.com


DEEP LEARNING
MATH, MATH MATH


WHEN YOU CREATE A CONVOLUTIONAL NEURAL NETWORK
AND YOU REALIZE THAT YOU DON'T HAVE A GPU TO TRAIN IT


HOME
MENU
CONNECT
THE LATEST
POPULAR
MOST SHARED
MIT Technology Review
10 BREAKTHROUGH TECHNOLOGIES 2013
Introduction
The 10 Technologies
Past Years
Deep Learning
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Temporary Social Media
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.
Prenatal DNA Sequencing
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?
Additive Manufacturing
Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts.
Baxter: The Blue-Collar Robot
Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people.
Memory Implants
Smart Watches
Ultra-Efficient Solar
Big Data from
Supergrids

(Unrelated) Dog vs Food

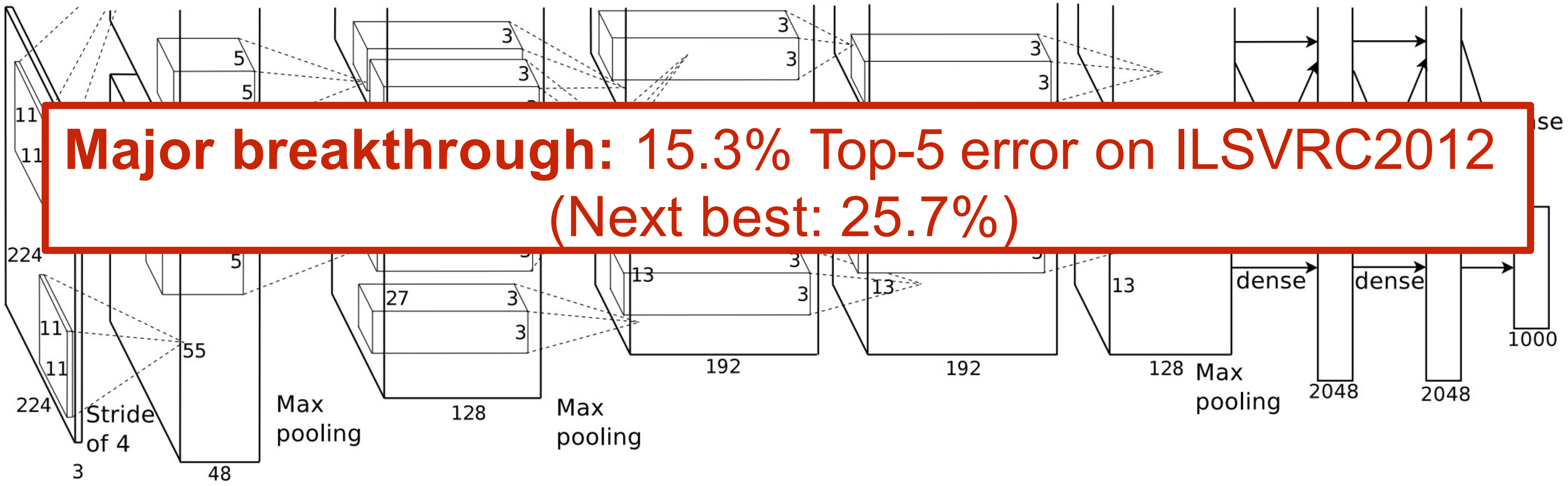
[Karen Zack, @teenybiscuit]

(Unrelated) Dog vs Food

[Karen Zack, @teenybiscuit]

CNNs in 2012: “SuperVision” (aka “AlexNet”)

“AlexNet” — Won the ILSVRC2012 Challenge

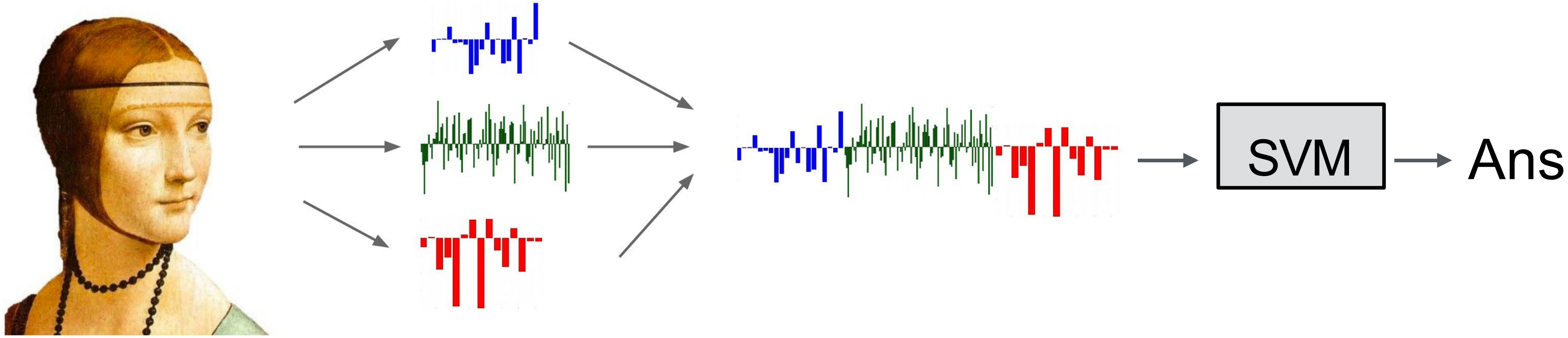


Major breakthrough: 15.3% Top-5 error on ILSVRC2012
(Next best: 25.7%)

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]

Recap: Before Deep Learning



Input Pixels

Extract Features

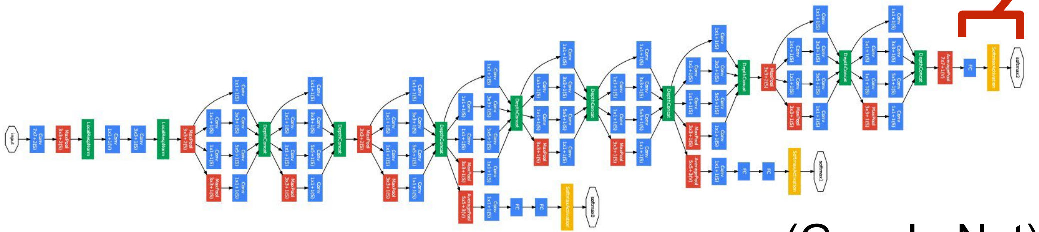
*Concatenate into a vector **x***

Linear Classifier

Figure: Karpathy 2016

The last layer of (most) CNNs are linear classifiers

This piece is just a linear classifier



(GoogLeNet)

Ans

Input Pixels

Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

ConvNets

They’re just neural networks with
3D activations and weight sharing

What shape should the activations have?

$$x \rightarrow \boxed{\text{Layer}} \rightarrow h^{(1)} \rightarrow \boxed{\text{Layer}} \rightarrow h^{(2)} \rightarrow \cdots \rightarrow f$$

- The input is an image, which is 3D (RGB channel, height, width)

What shape should the activations have?

$$x \to \boxed{\text{Layer}} \to h^{(1)} \to \boxed{\text{Layer}} \to h^{(2)} \to \cdots \to f$$

- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

What shape should the activations have?

$$x \to \boxed{\text{Layer}} \to h^{(1)} \to \boxed{\text{Layer}} \to h^{(2)} \to \cdots \to f$$

- The input is an image, which is 3D (RGB channel, height, width)

- We could flatten it to a 1D vector, but then we lose structure

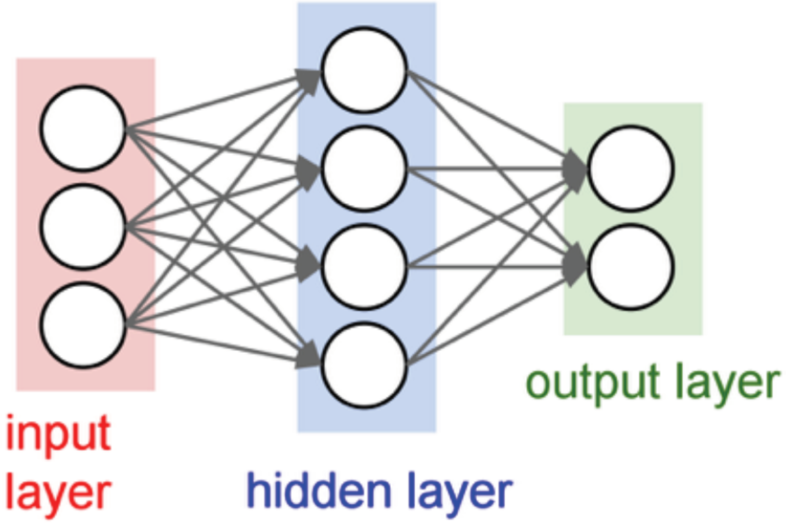
- What about keeping everything in 3D?

ConvNets

They’re just neural networks with
3D activations and weight sharing

3D Activations

before:

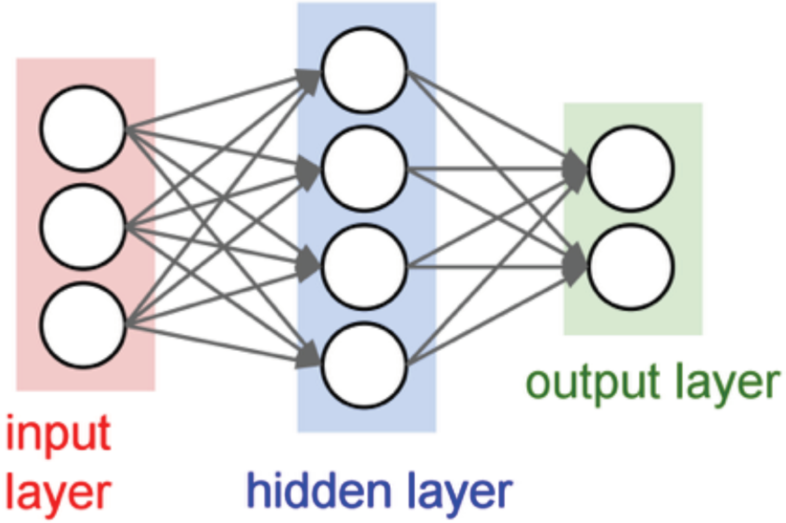


(1D vectors)

Figure: Andrej Karpathy

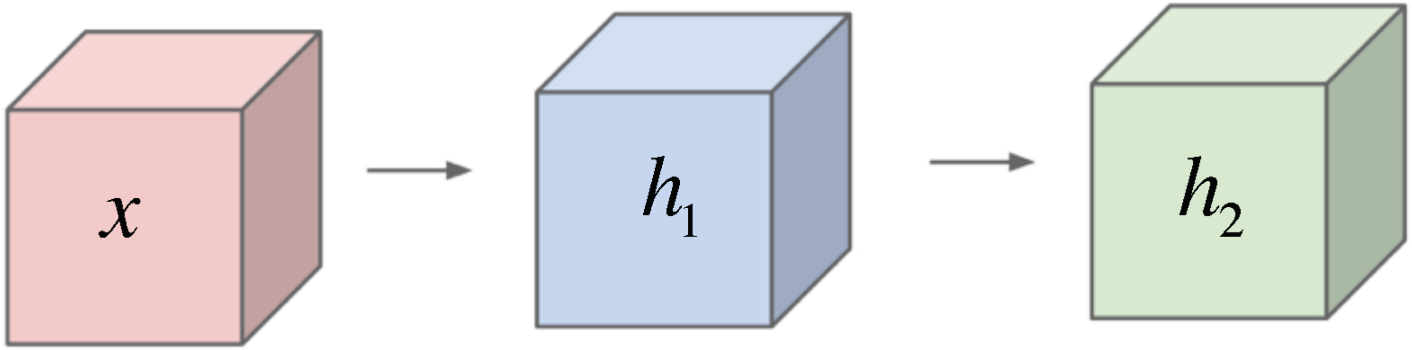
3D Activations

before:



(1D vectors)

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**

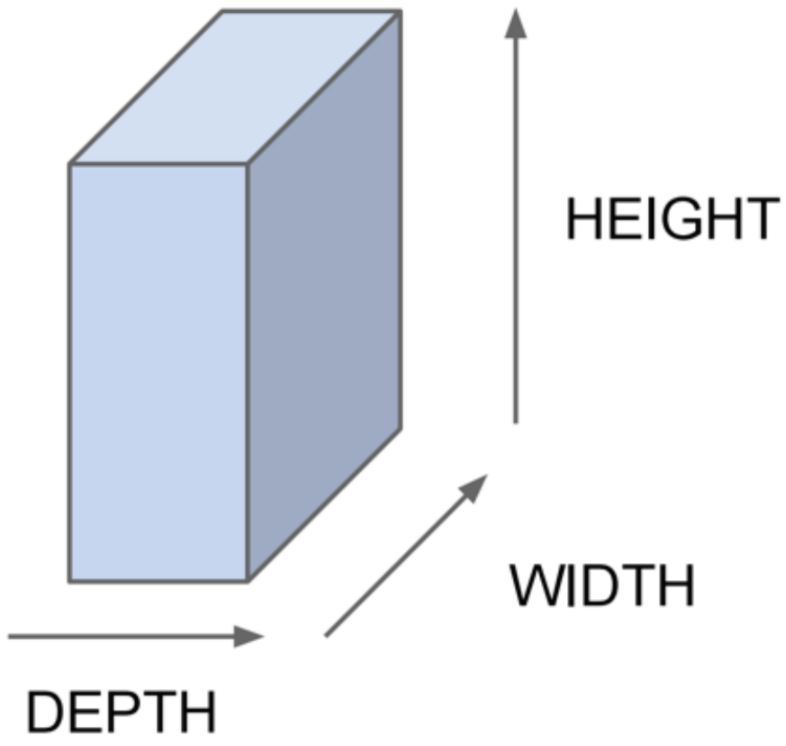
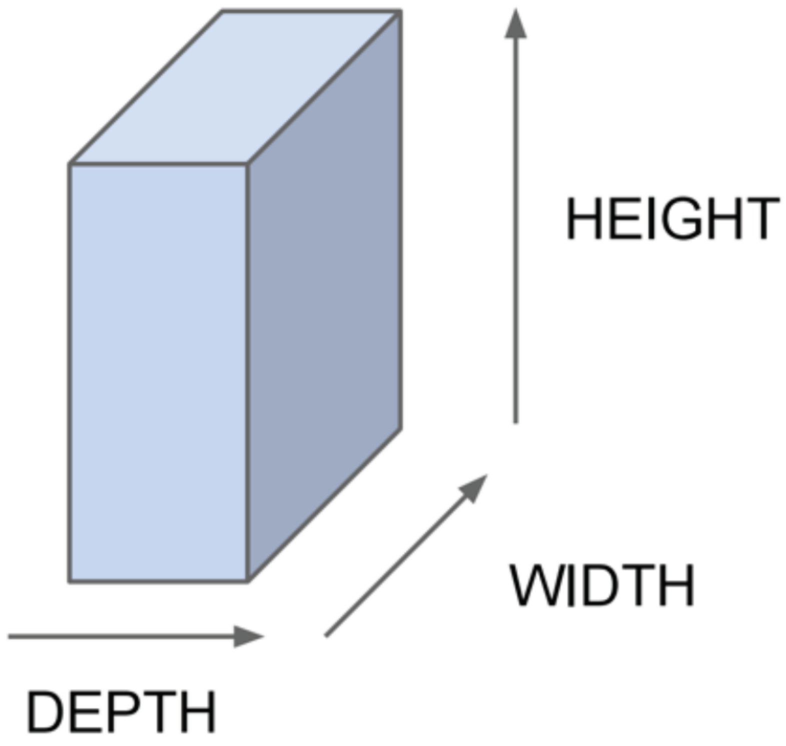


Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume
(3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:

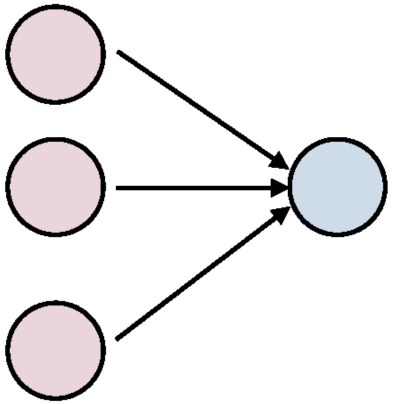
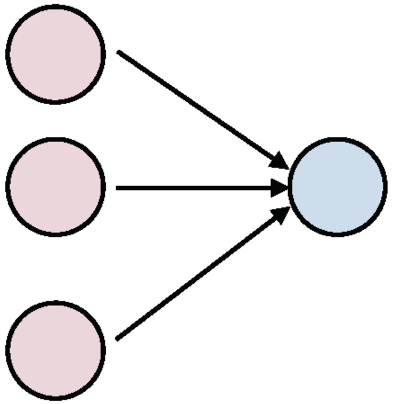


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

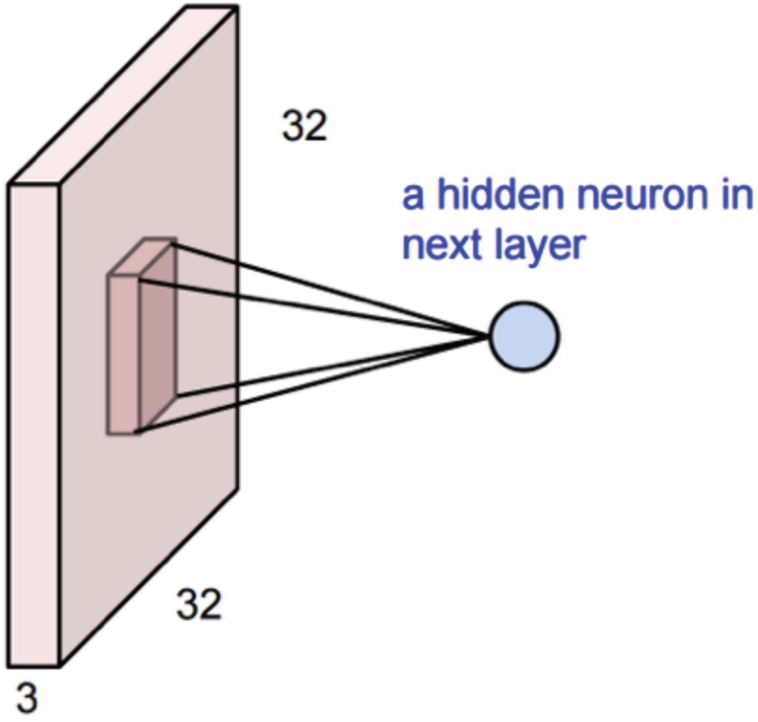


Figure: Andrej Karpathy

3D Activations

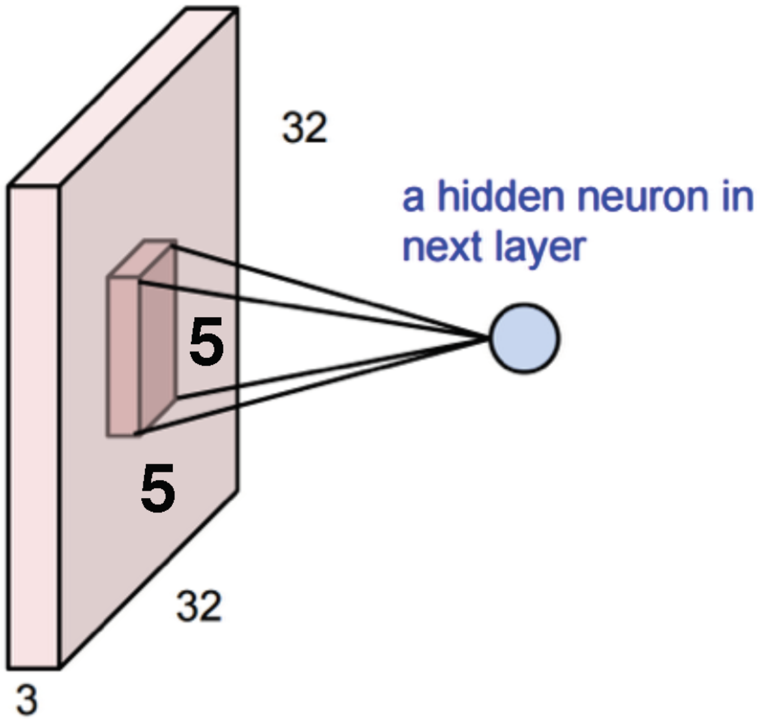


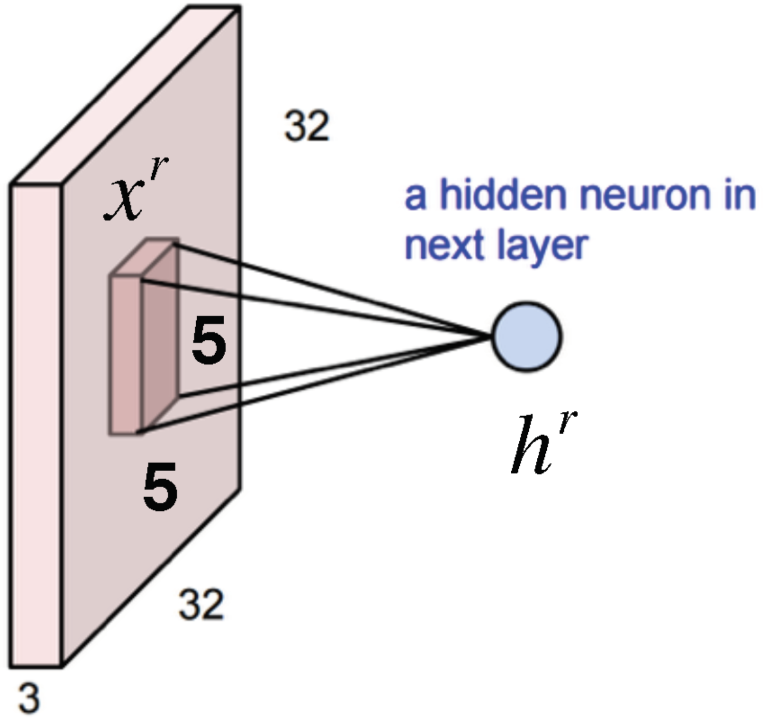
Figure: Andrej Karpathy

- The input is 3x32x32

- This neuron depends on a 3x5x5 chunk of the input

- The neuron also has a 3x5x5 set of weights and a bias (scalar)

3D Activations

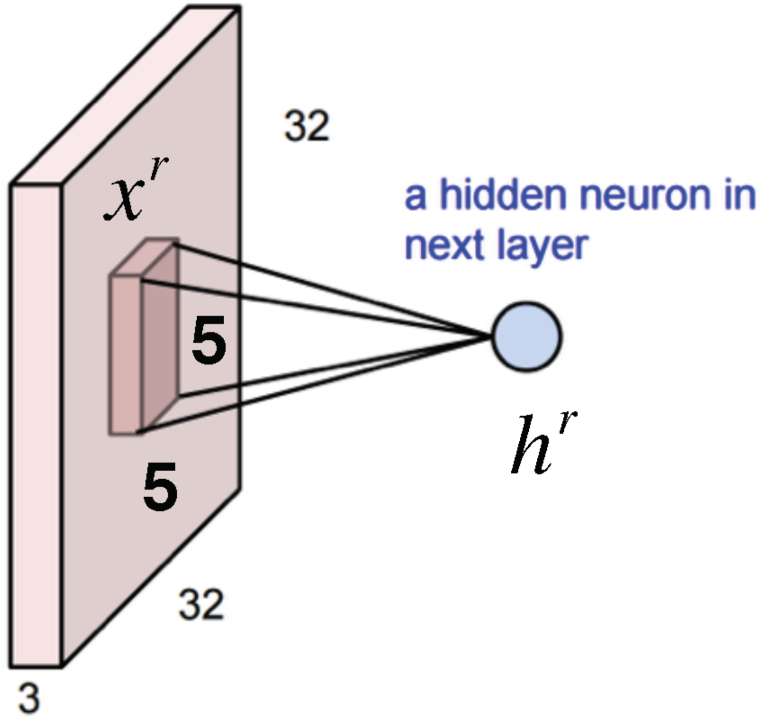


Example: consider the region of the input "x^r"

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Figure: Andrej Karpathy

3D Activations

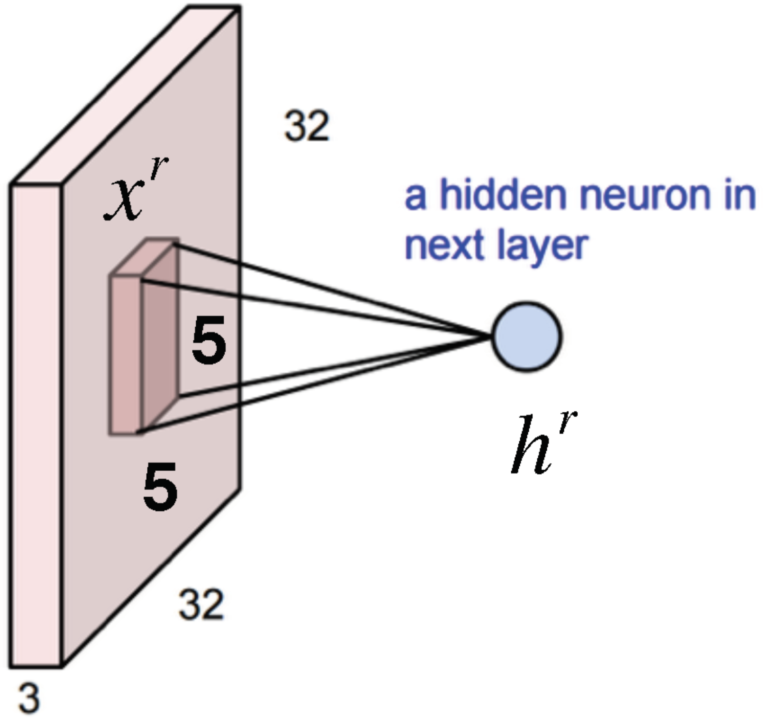


Figure: Andrej Karpathy

Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Sum over 3 axes

3D Activations

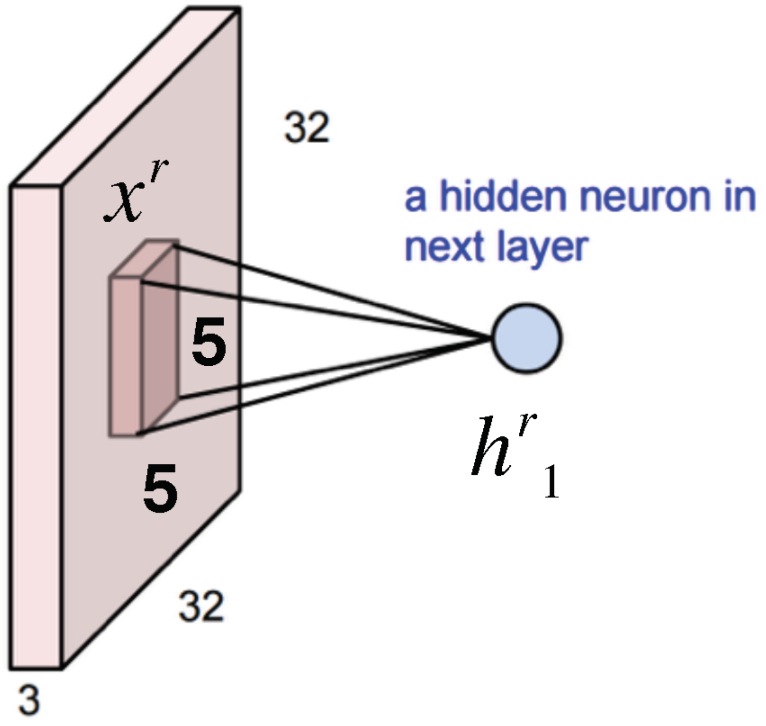


Figure: Andrej Karpathy

3D Activations

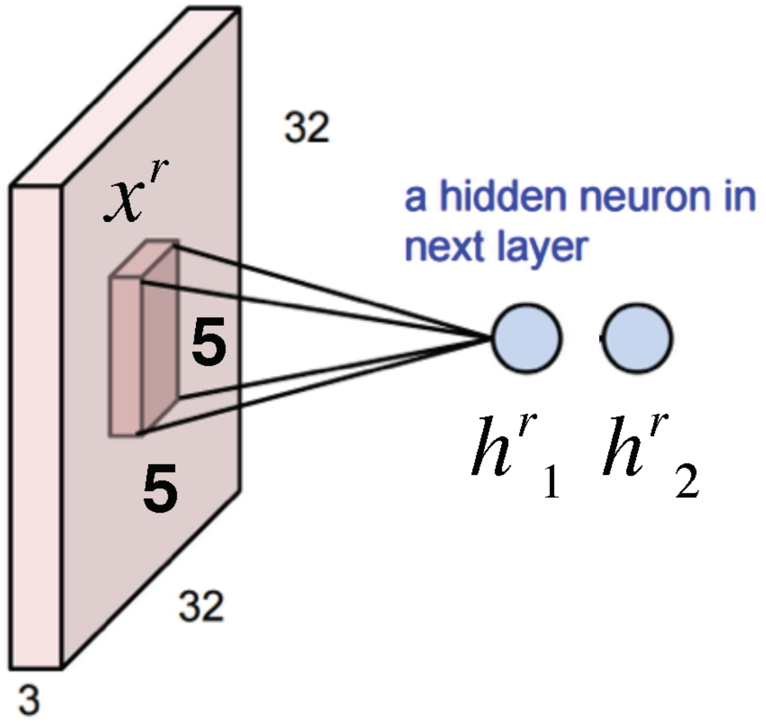
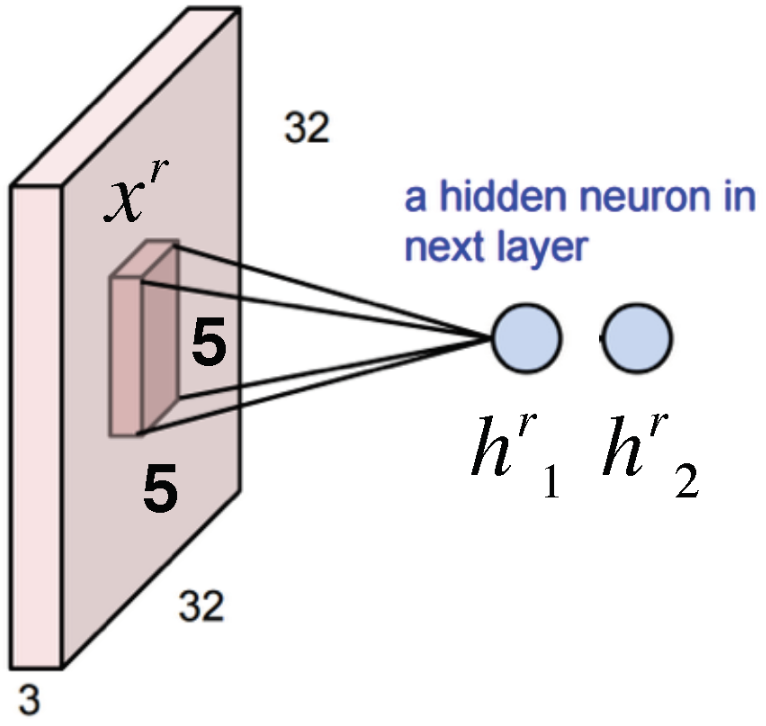


Figure: Andrej Karpathy

3D Activations



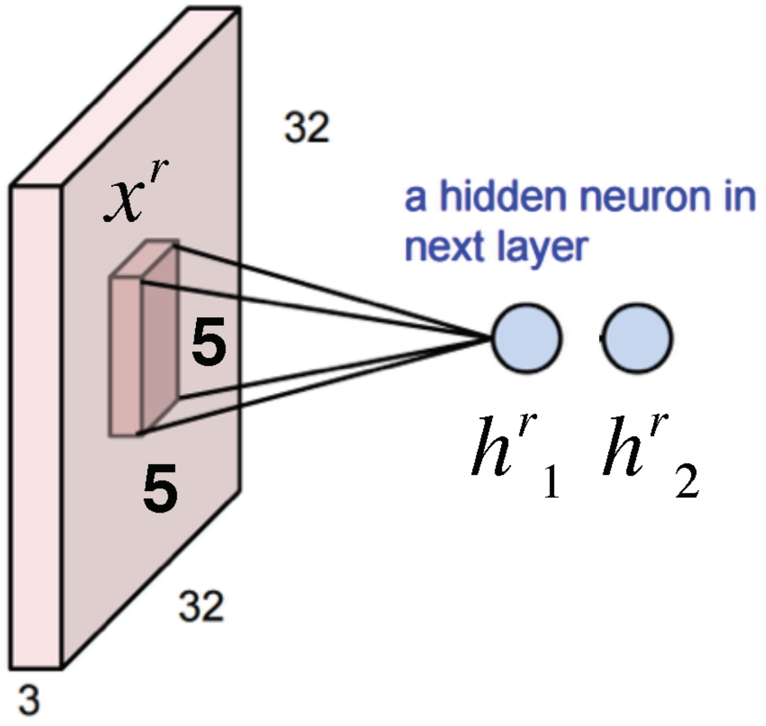
With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{1}ijk} + b_{\boxed{1}}$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{2}ijk} + b_{\boxed{2}}$$

Figure: Andrej Karpathy

3D Activations

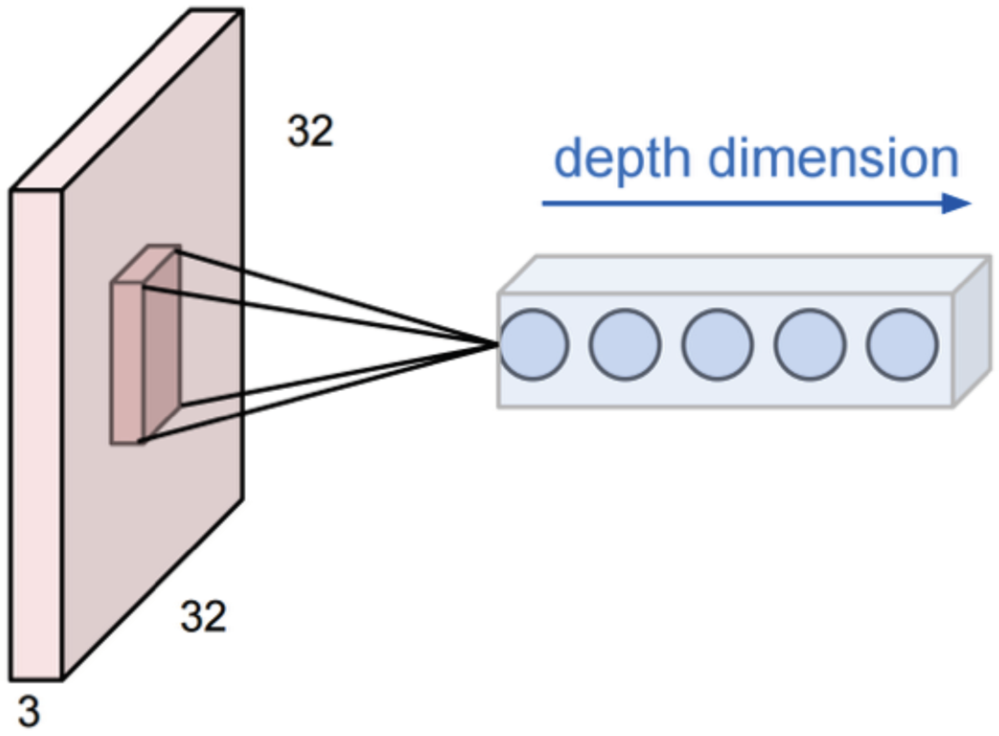
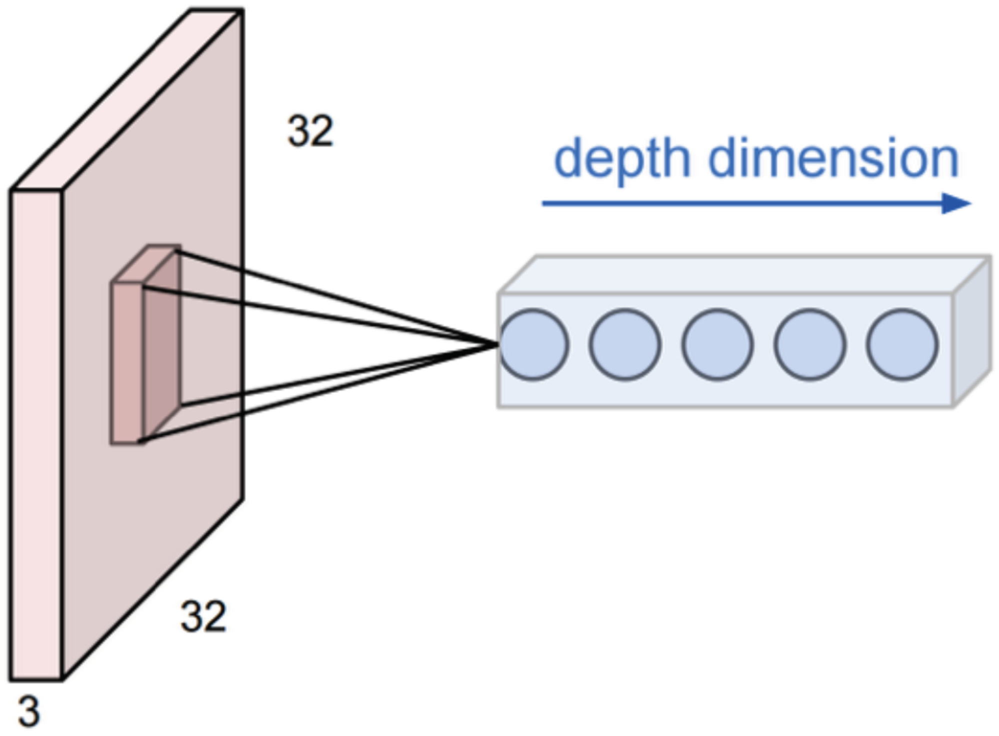


Figure: Andrej Karpathy

3D Activations

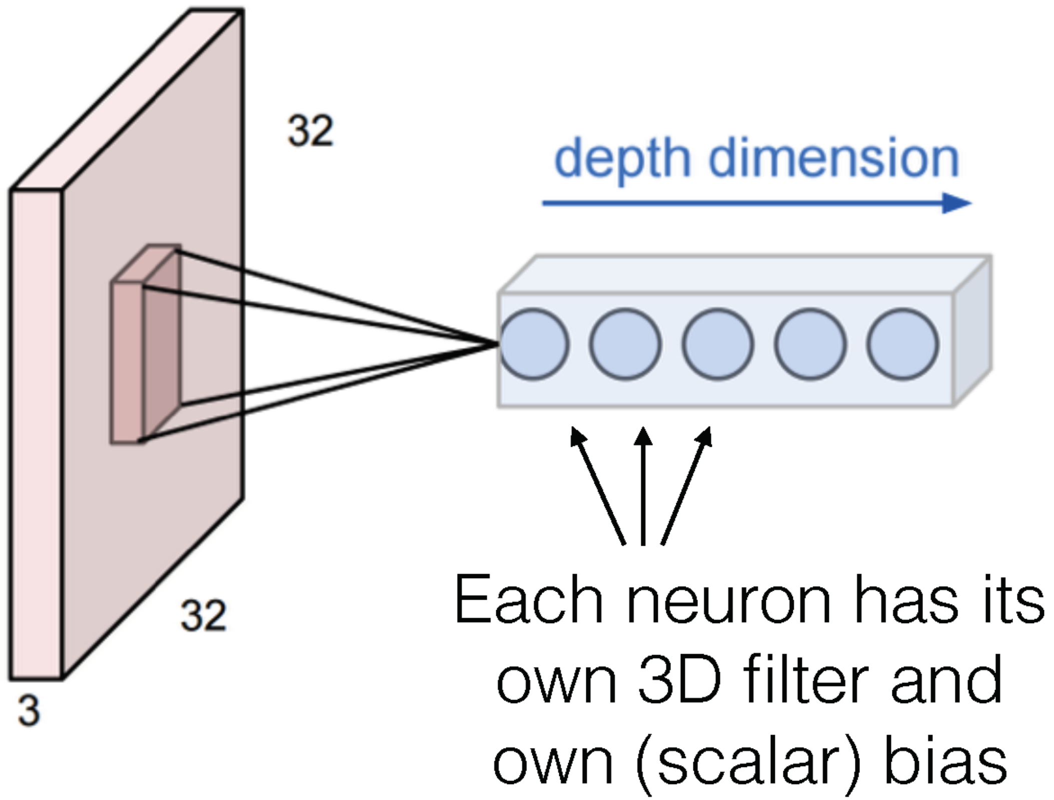


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations

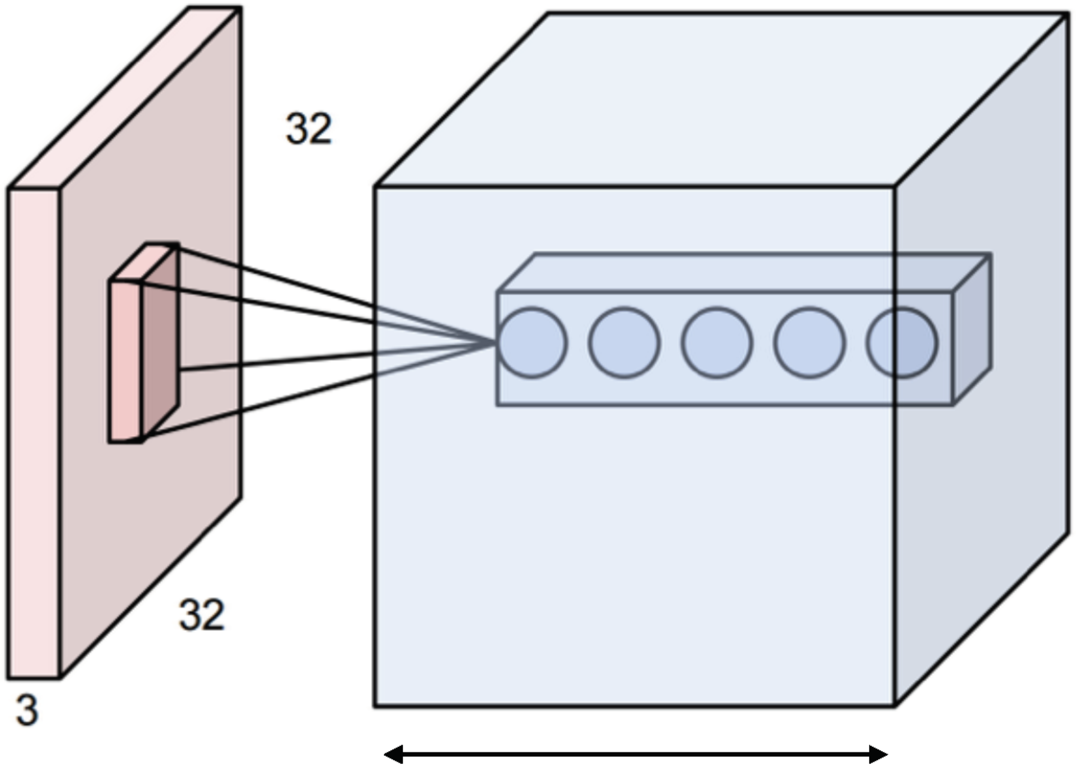


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations

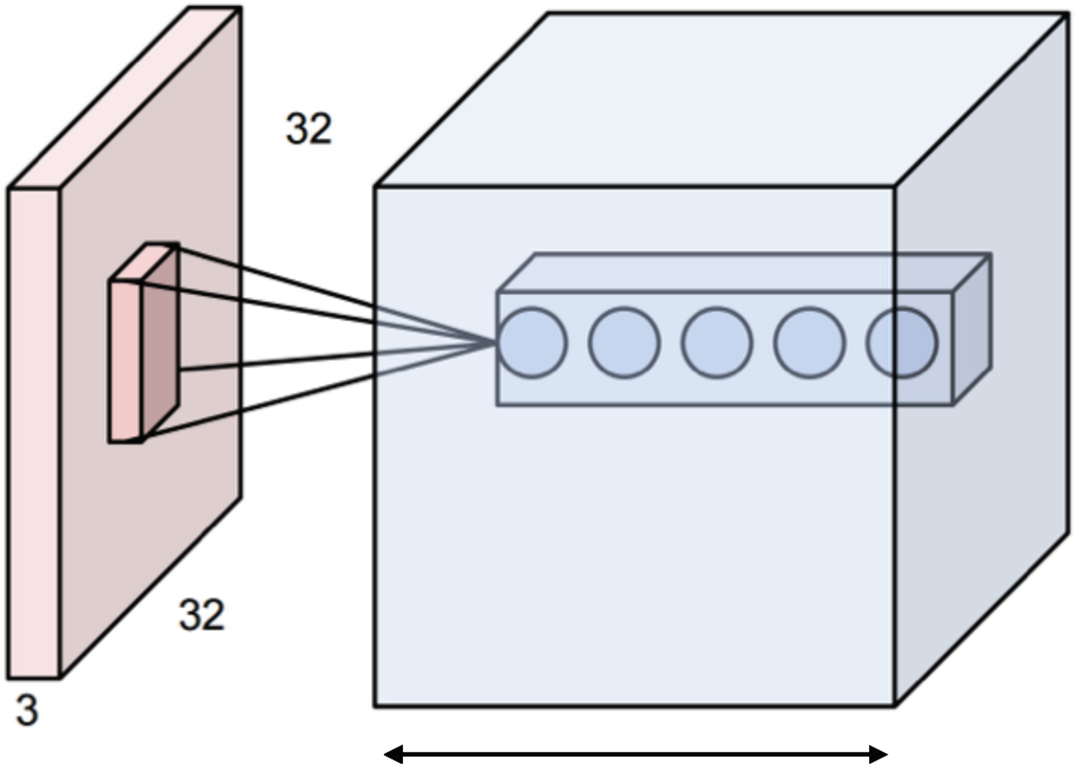


D sets of weights
(also called filters)

Now repeat this across the input

Figure: Andrej Karpathy

3D Activations



D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:
Each filter shares the same weights (but each depth index has its own set of weights)

Figure: Andrej Karpathy

3D Activations

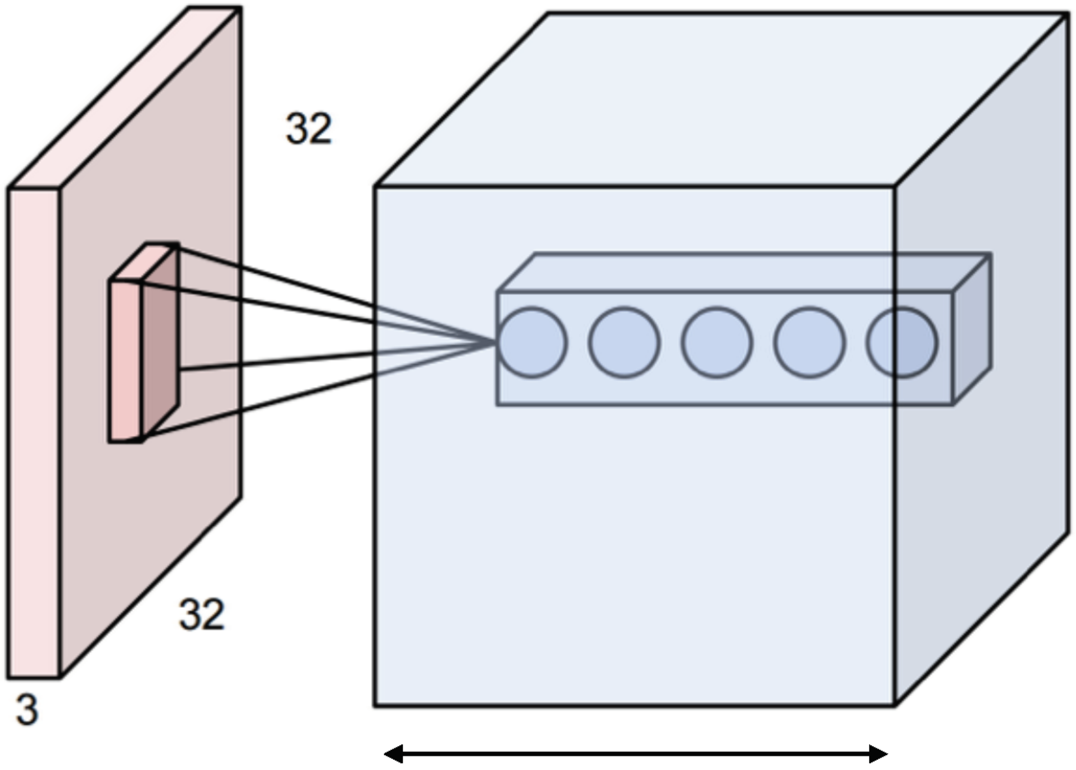
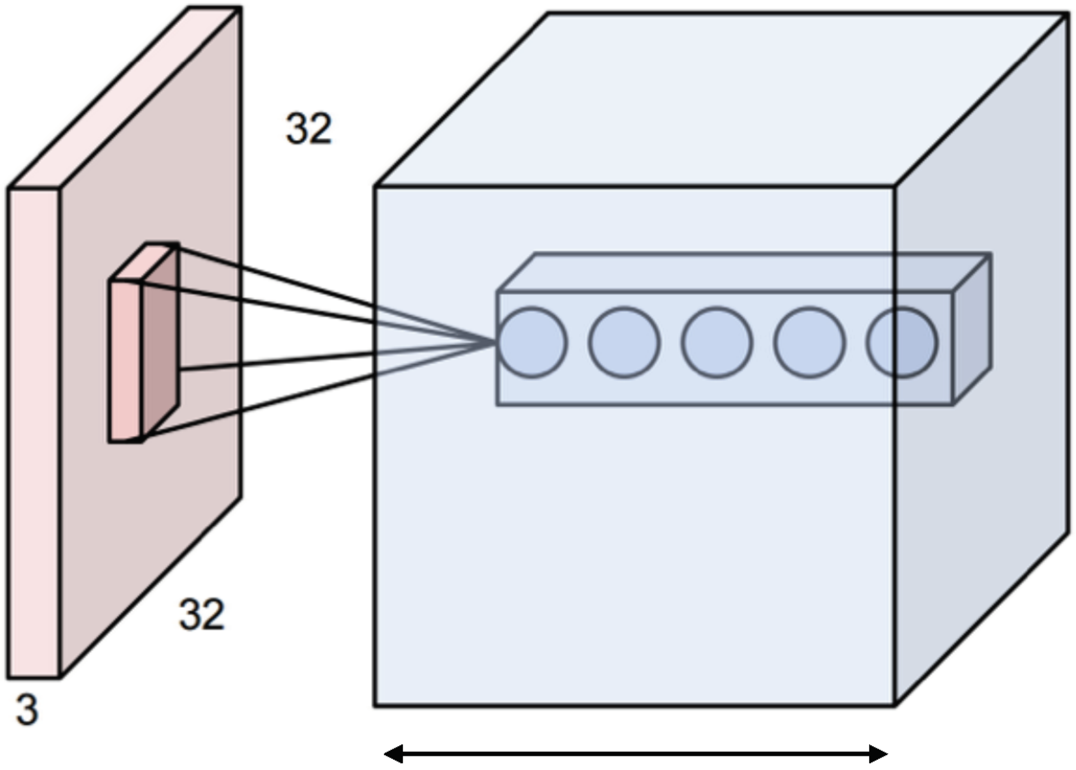


Figure: Andrej Karpathy

3D Activations

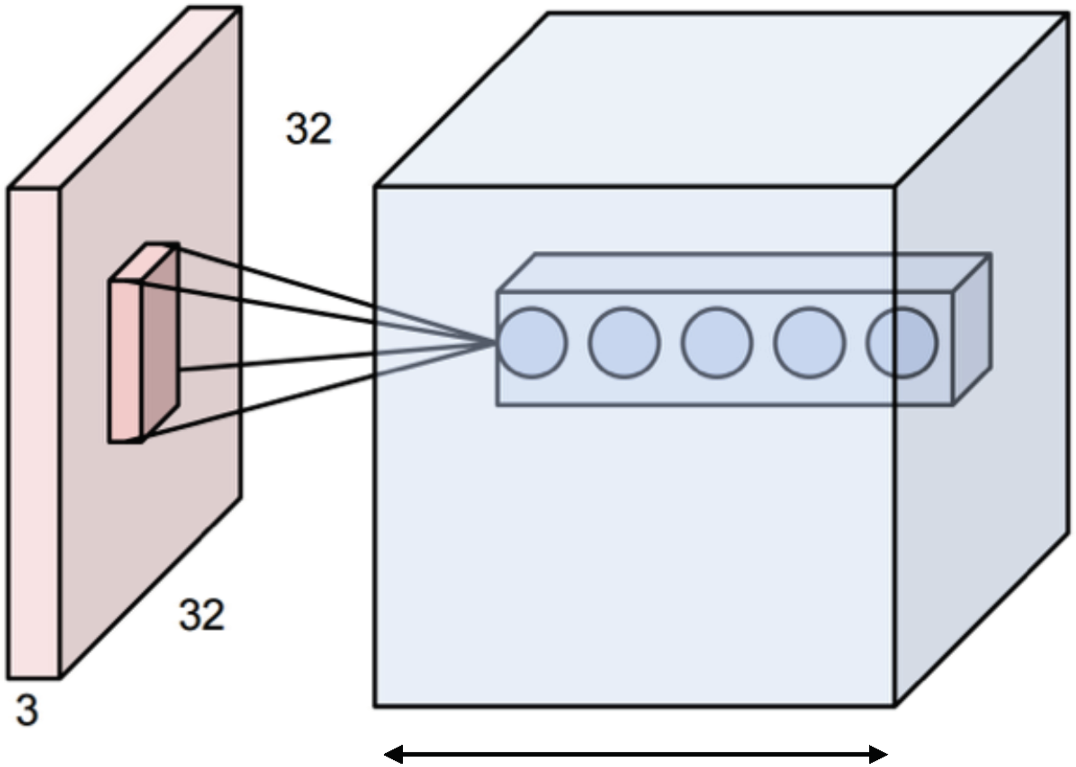


D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Figure: Andrej Karpathy

3D Activations



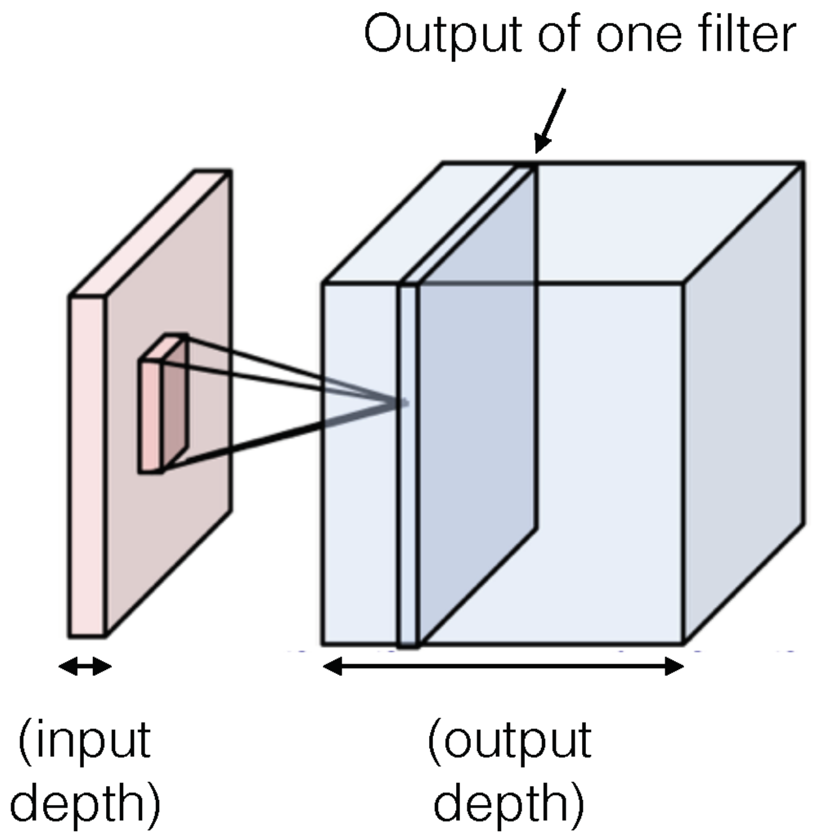
D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Without weight sharing, this is called a **locally connected layer**

Figure: Andrej Karpathy

3D Activations

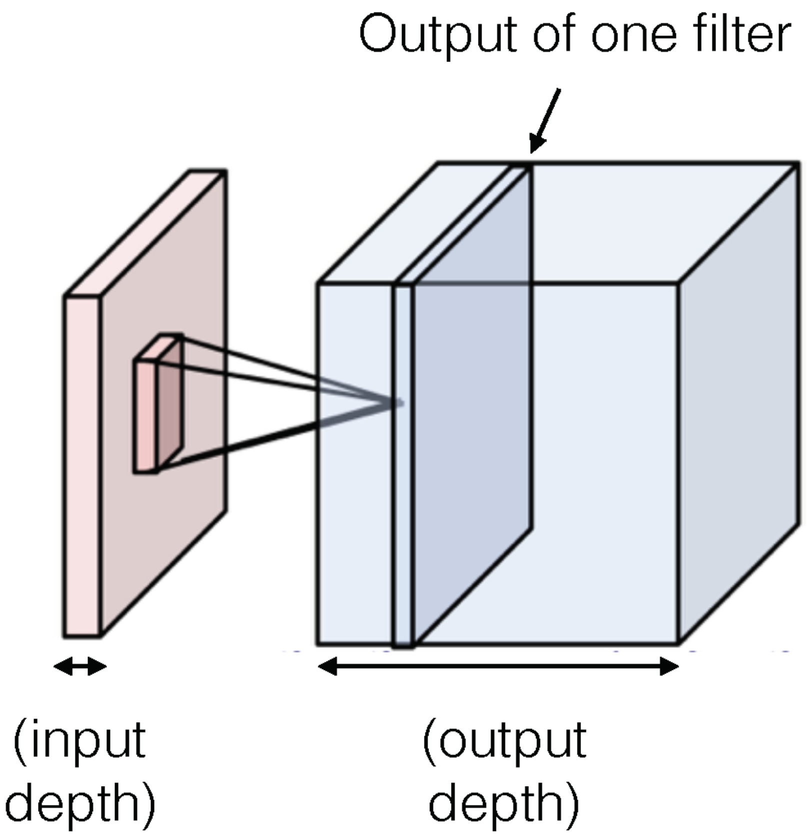


One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

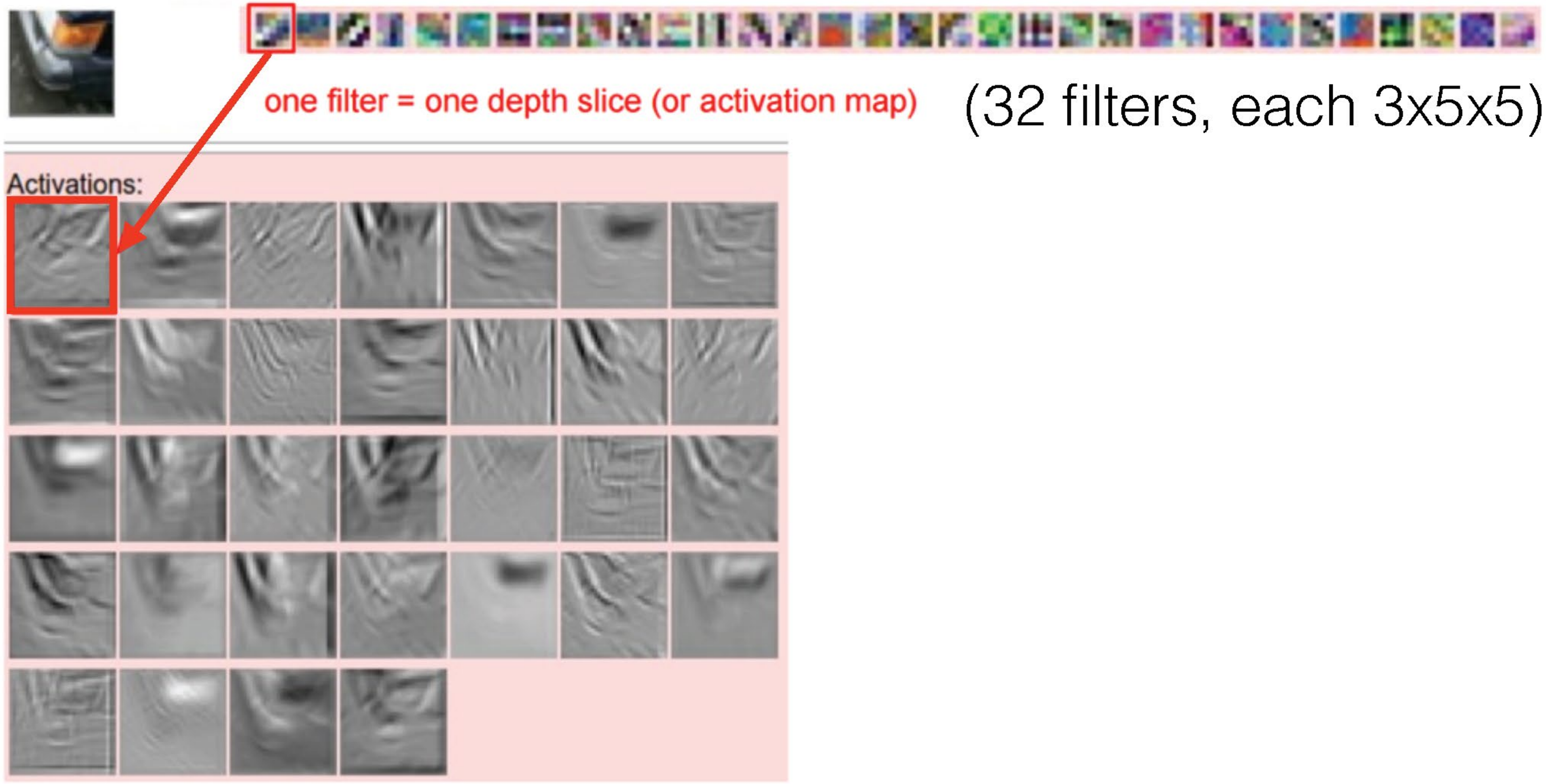
In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



(32 filters, each 3x5x5)

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)

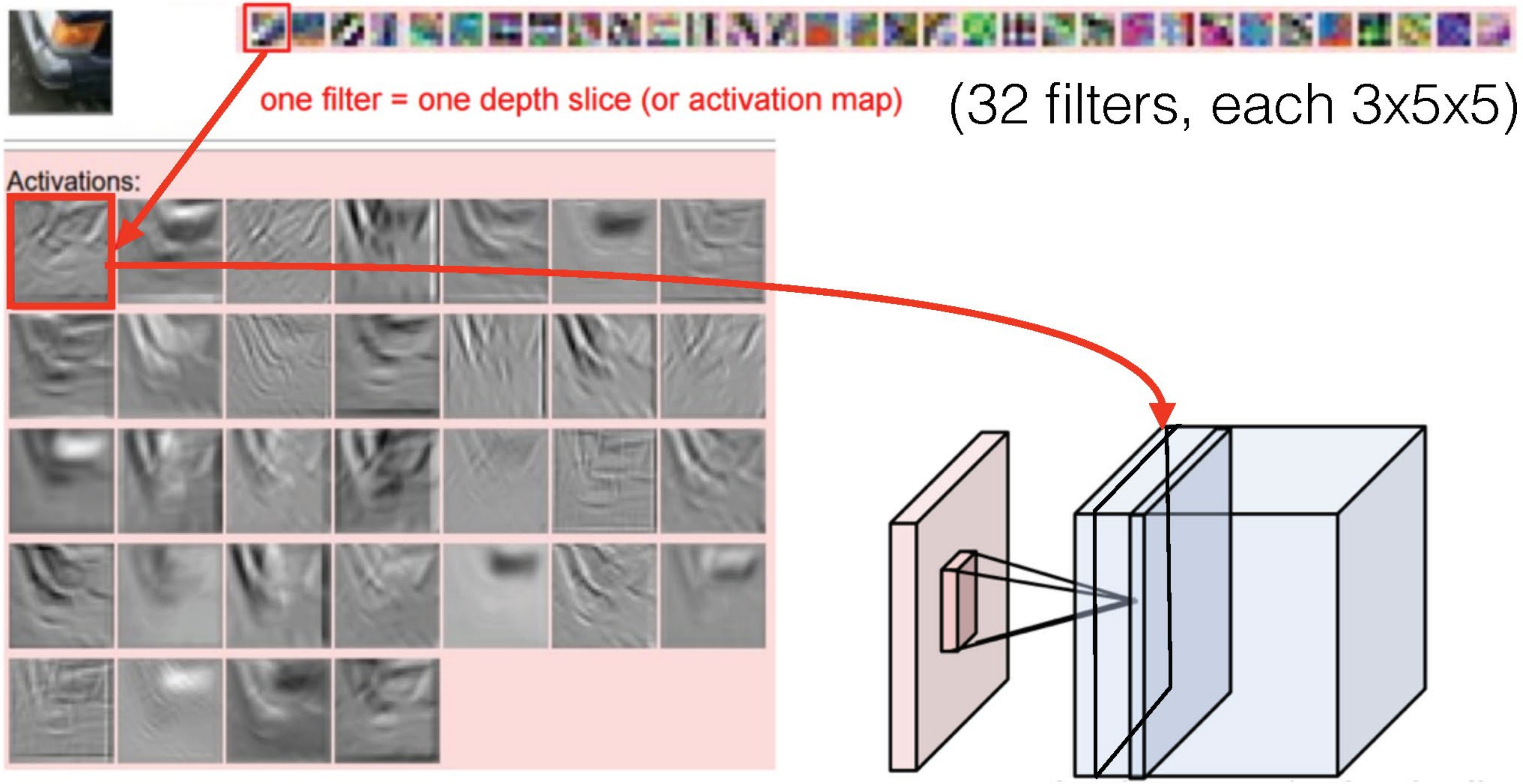


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

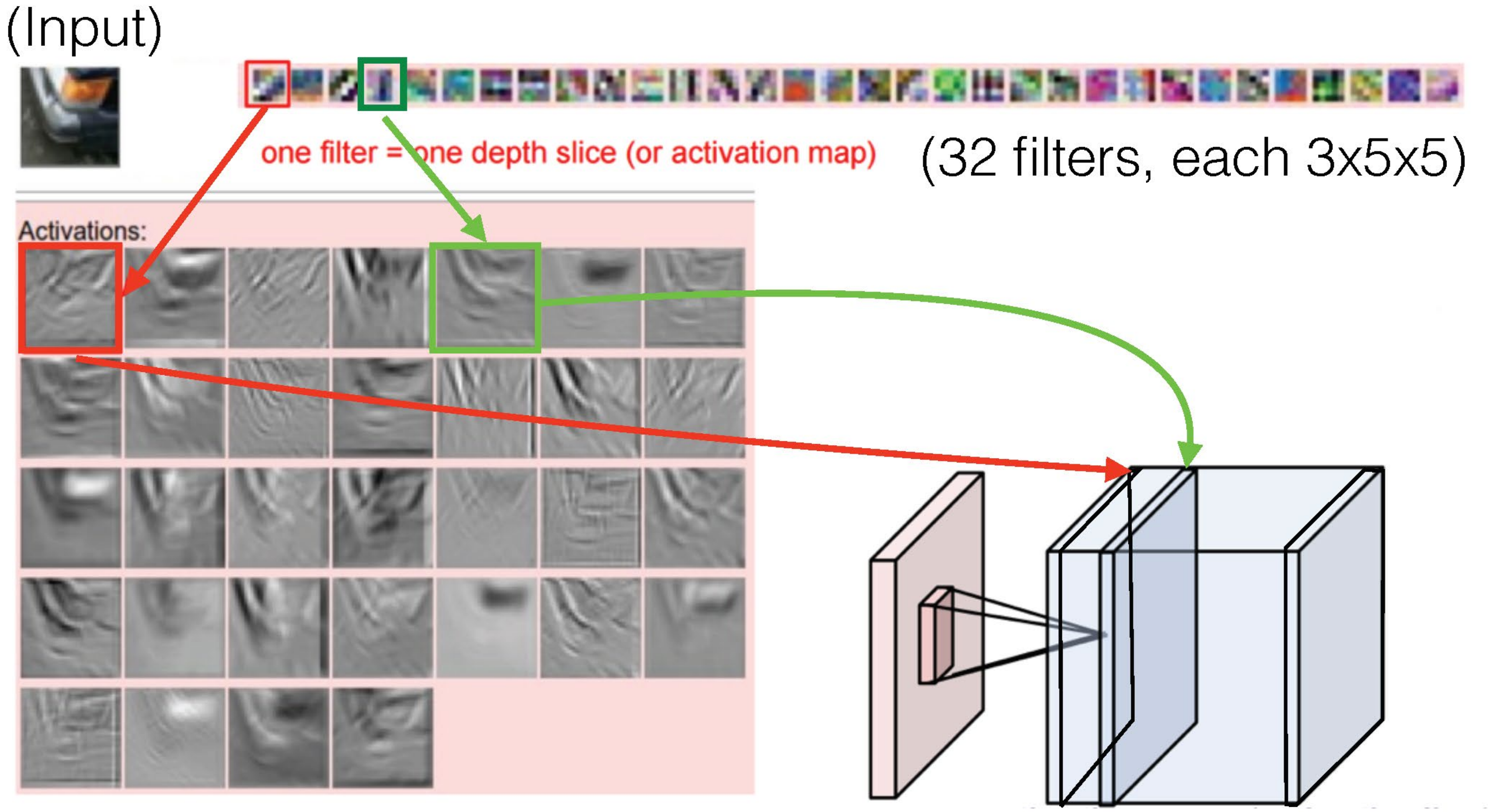


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

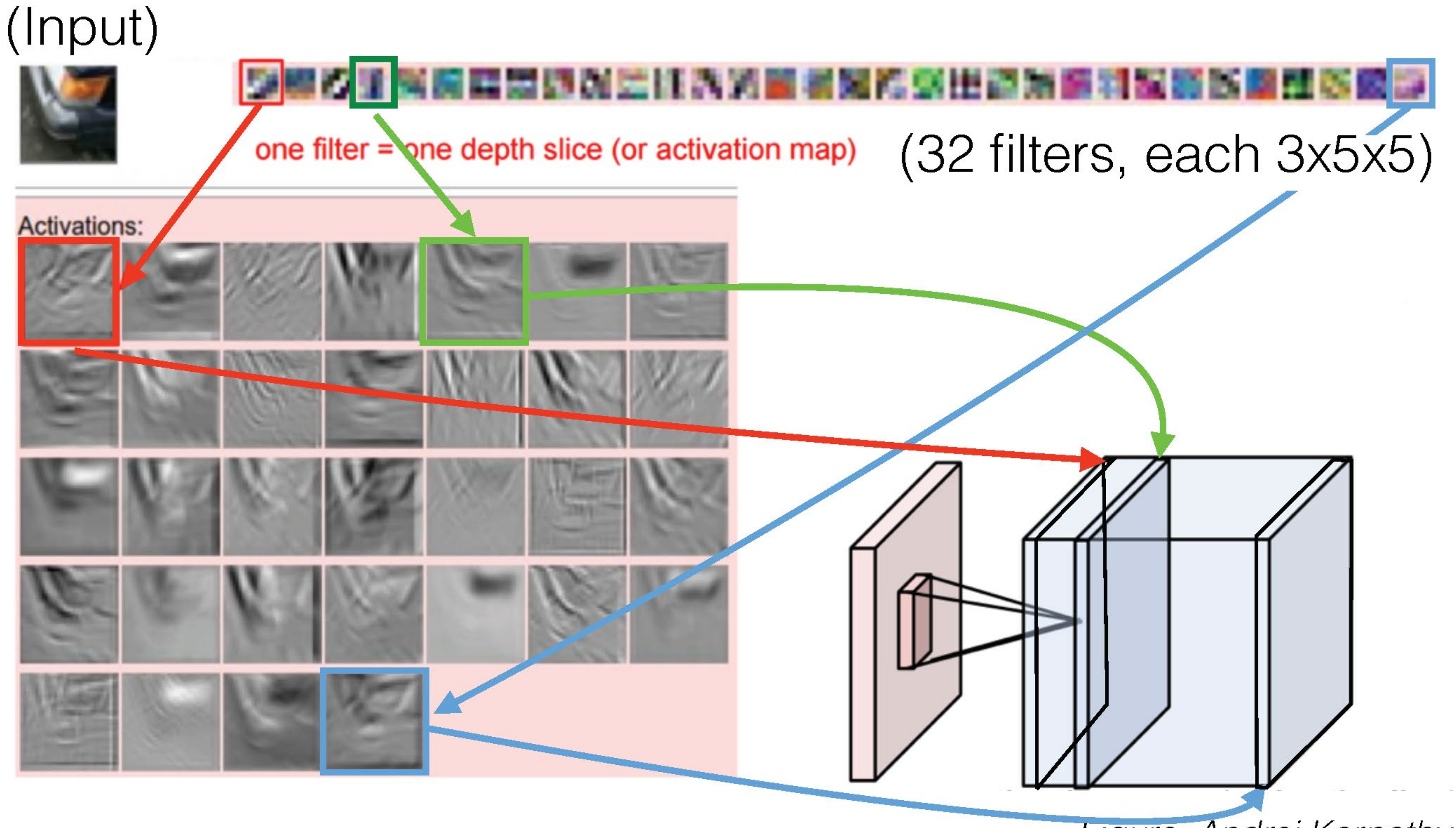


Figure: Andrej Karpathy