CS 1678: Intro to Deep Learning

Neural Network Basics

Prof. Adriana Kovashka
University of Pittsburgh
January 22, 2024

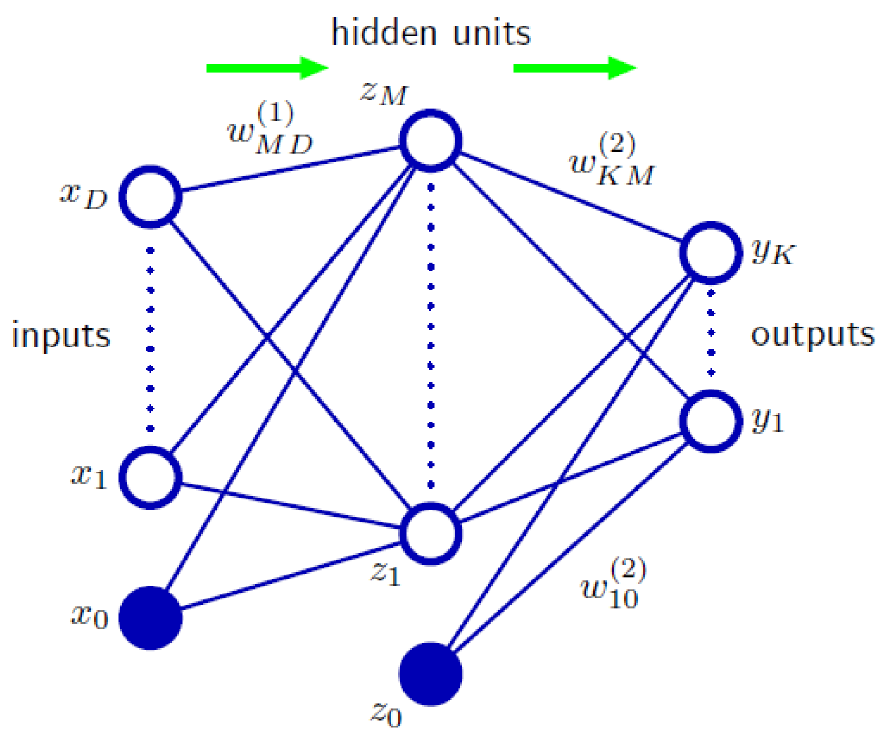
Plan for this lecture (next few classes)

- Definition
  - Architecture
  - Basic operations
  - Biological inspiration
- Goals
  - Loss functions
- Training
  - Gradient descent
  - Backpropagation
- Hands-on exercise

Definition

Neural network definition

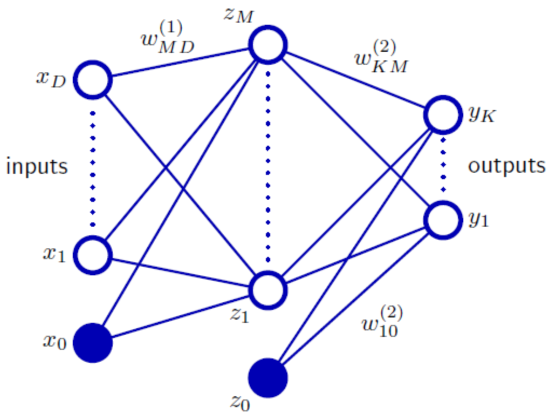
Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.



- Raw activations: $a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Nonlinear activation function *h* (e.g. sigmoid, tanh, RELU): $z_j = h(a_j)$ e.g. z = RELU(a) = max(0, a)

Figure from Christopher Bishop

Neural network definition



- Layer 2

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- Layer 3 (final)

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

- Outputs

(binary) $y_k = \sigma(a_k) = \dfrac{1}{1 + \exp(-a_k)}$

(multiclass) $y_k = \dfrac{\exp(a_k)}{\sum_j \exp(a_j)}$

- Finally:

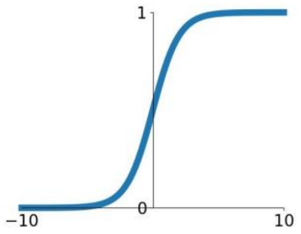
(binary)

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Activation functions

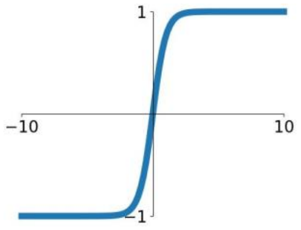
Sigmoid

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



tanh

$\tanh(x)$

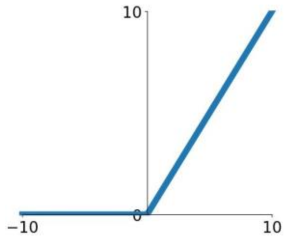


ReLU

$\max(0, x)$

f(x) = x if x >= 0

f(x) = 0 if x <= 0

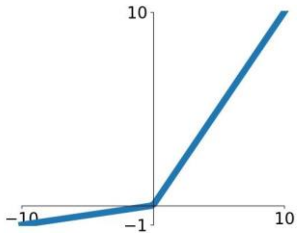


Leaky ReLU

$\max(0.1x, x)$

f(x) = x   if x >= 0

f(x) = 0.1x if x <= 0



PReLU

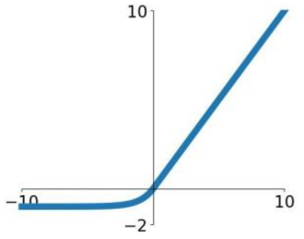
$\max(\text{a}\ x, x)$

Maxout

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

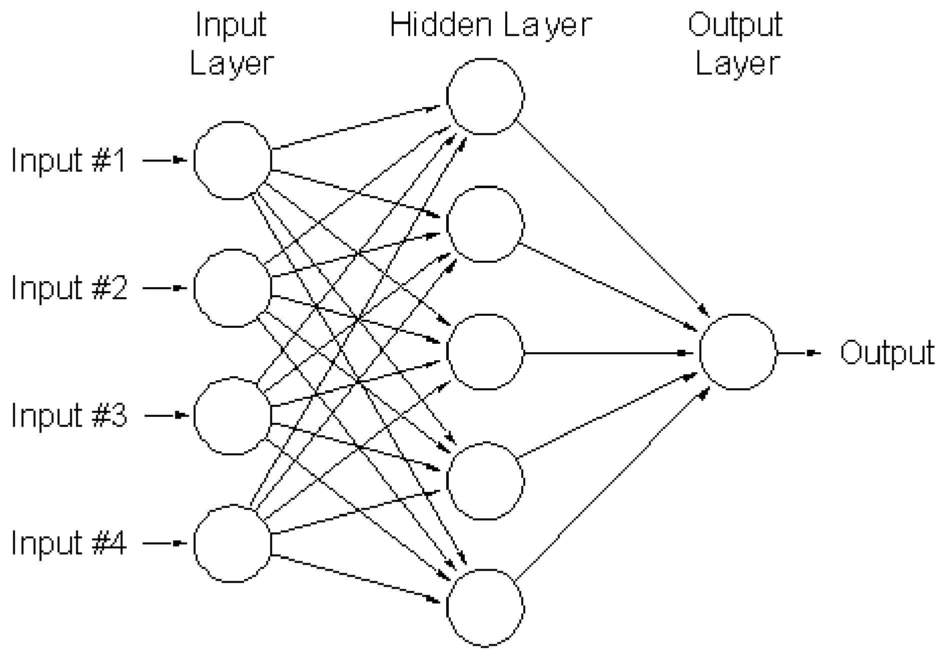
ELU

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



Adapted from Fei-Fei Li, Andrej Karpathy, Justin Johnson, Serena Yeung

A multi-layer neural network…

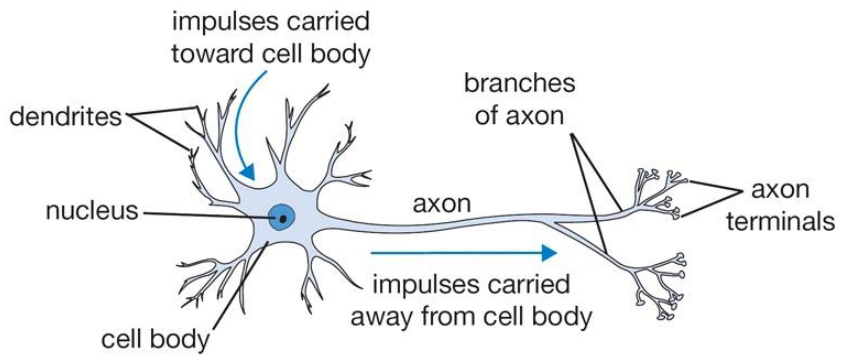


- Is a non-linear classifier
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units

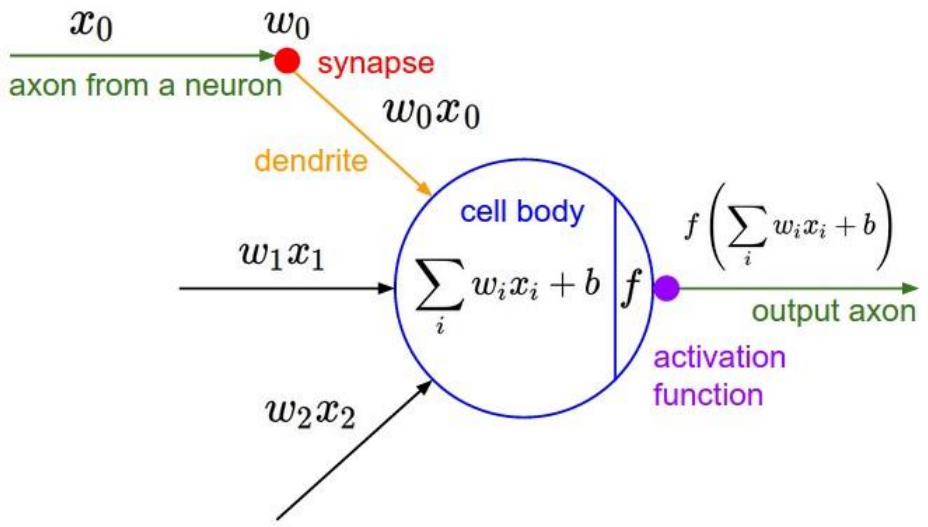
Lana Lazebnik

Inspiration: Neuron cells

- Neurons
  - accept information from multiple inputs
  - transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron "fires"

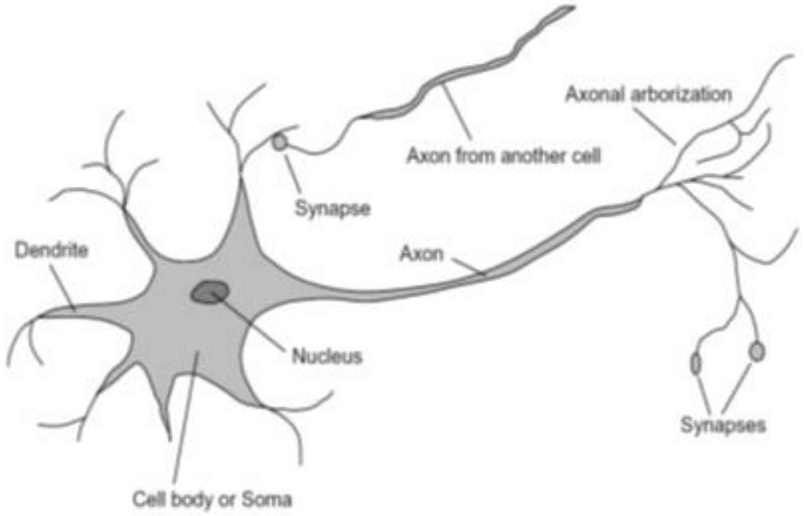


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

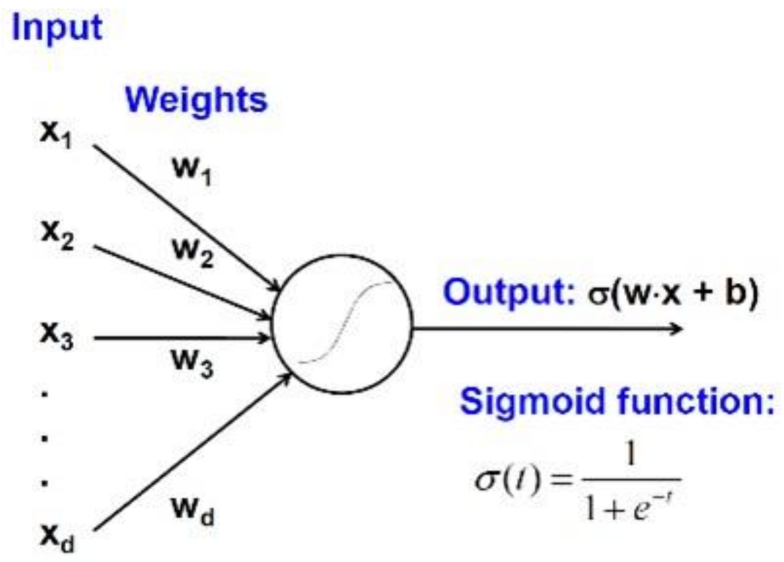


Text: HKUST, figures: Andrej Karpathy

Biological analog

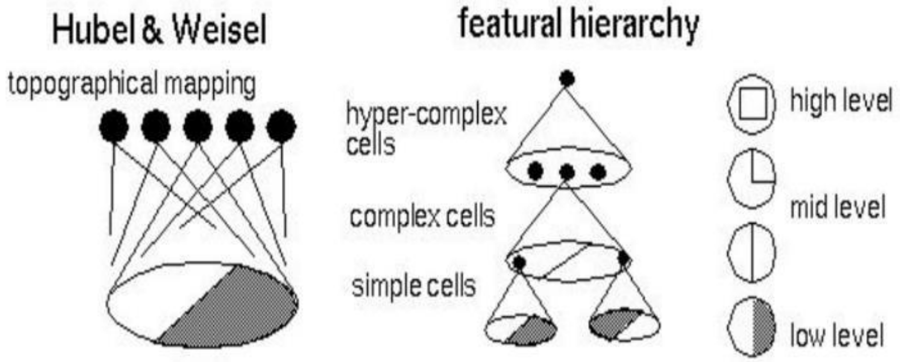


A biological neuron

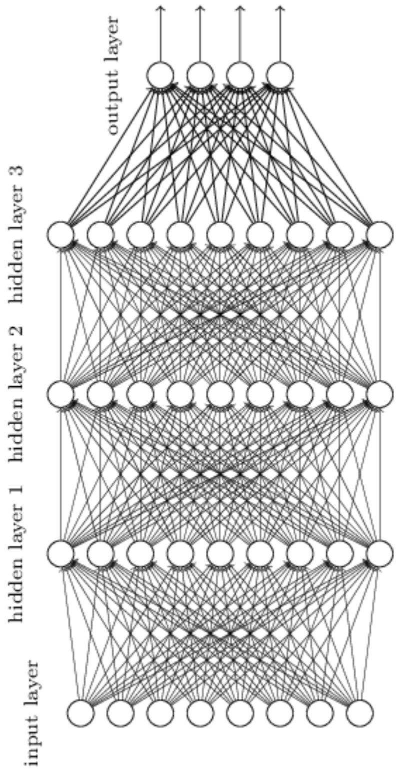


An artificial neuron

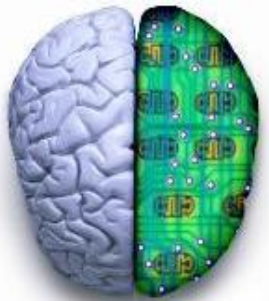
Biological analog



Hubel and Weisel's architecture



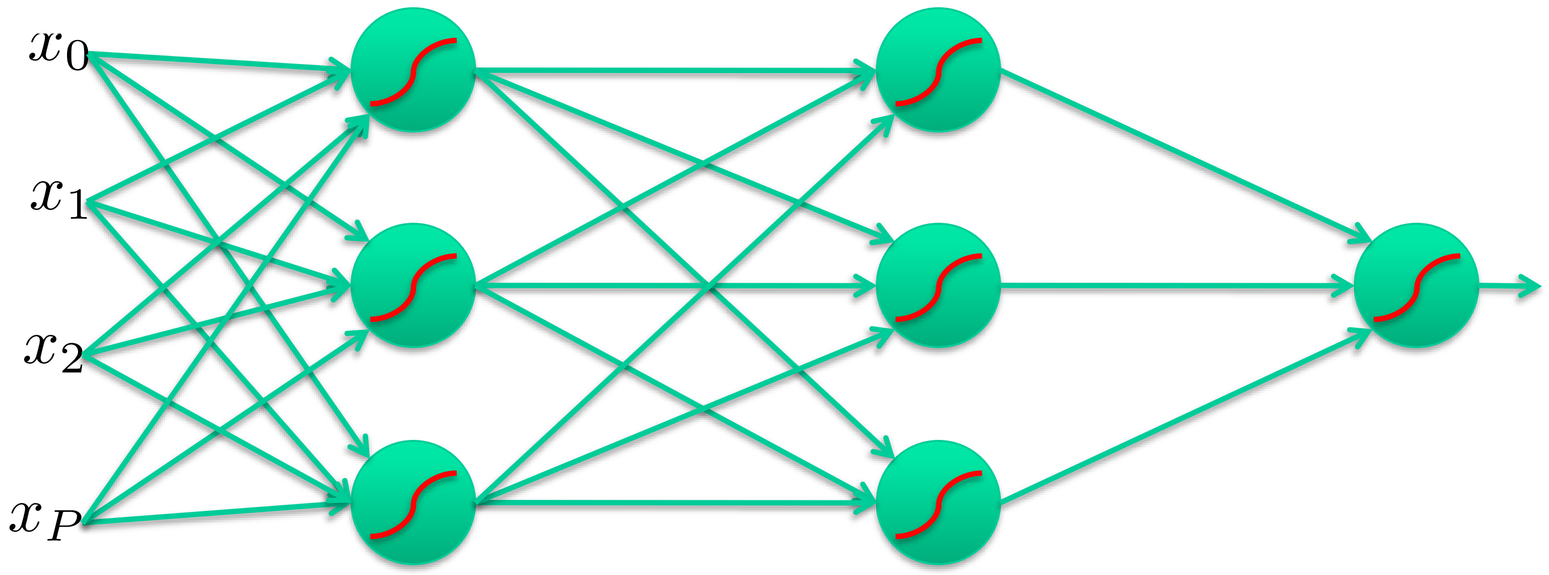
Multi-layer neural network



Adapted from Jia-bin Huang

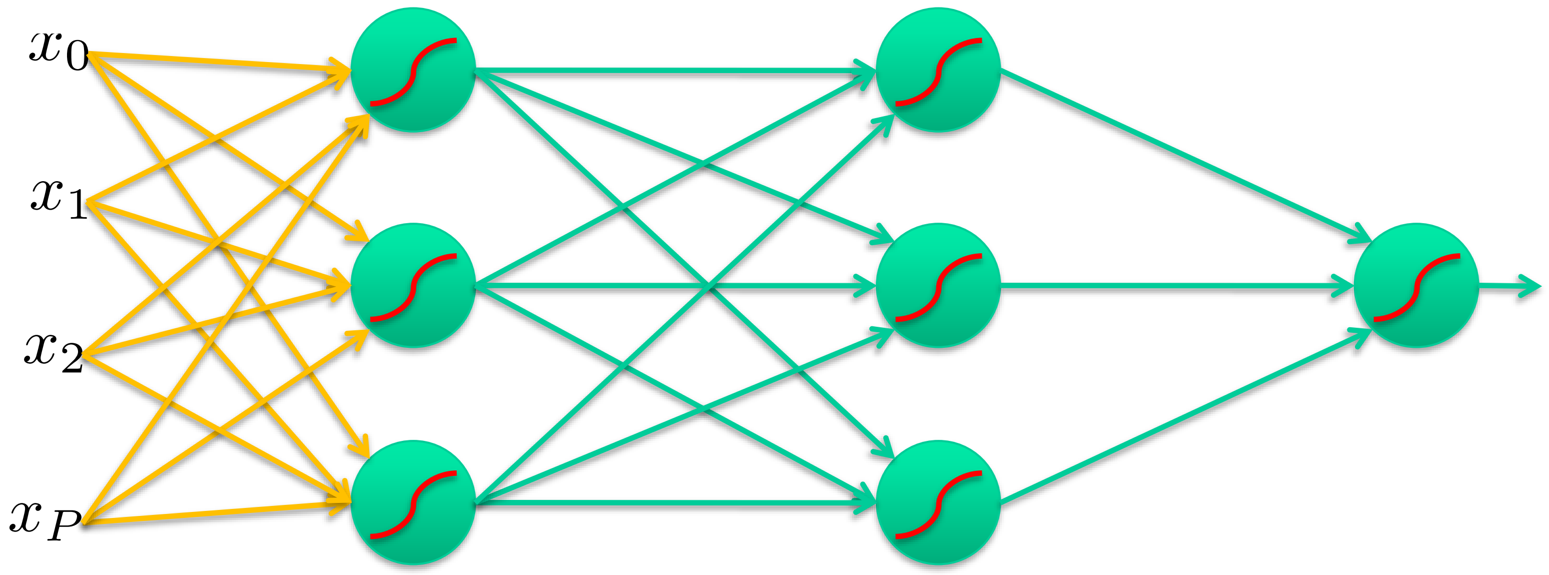
Feed-forward networks

- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights



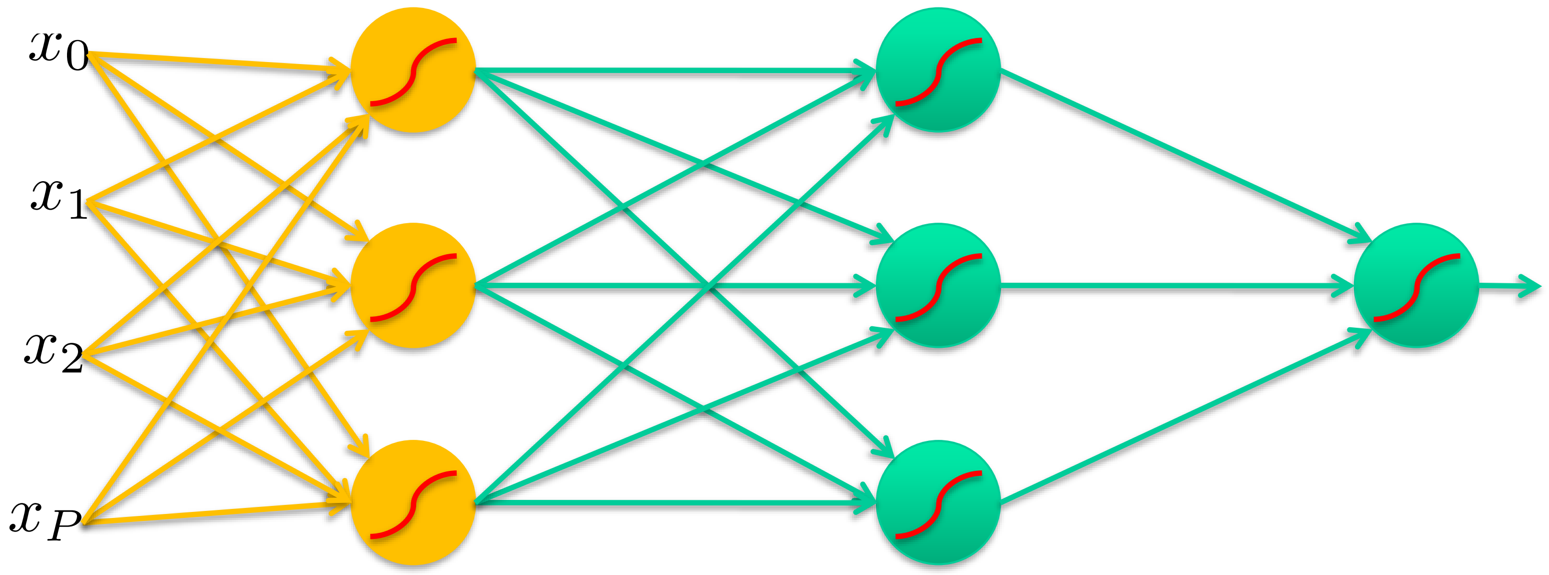
Feed-forward networks

- Inputs multiplied by initial set of weights



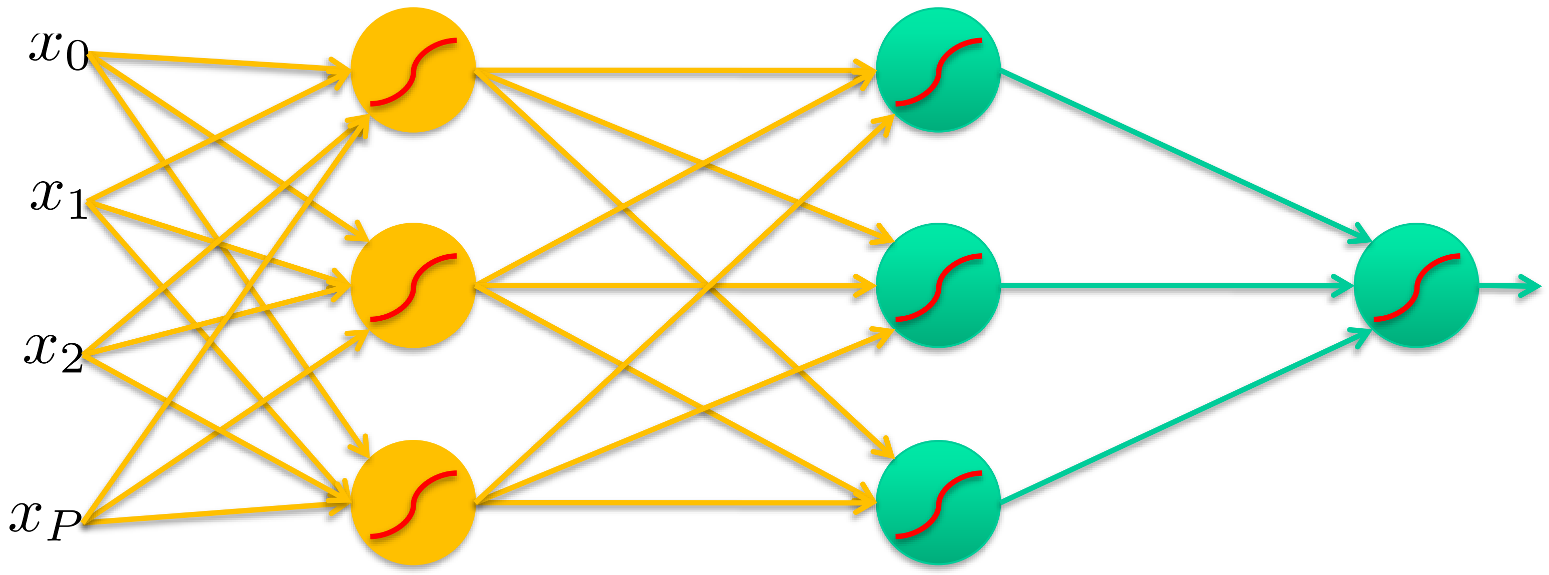
Feed-forward networks

- Intermediate “predictions” computed at first hidden layer



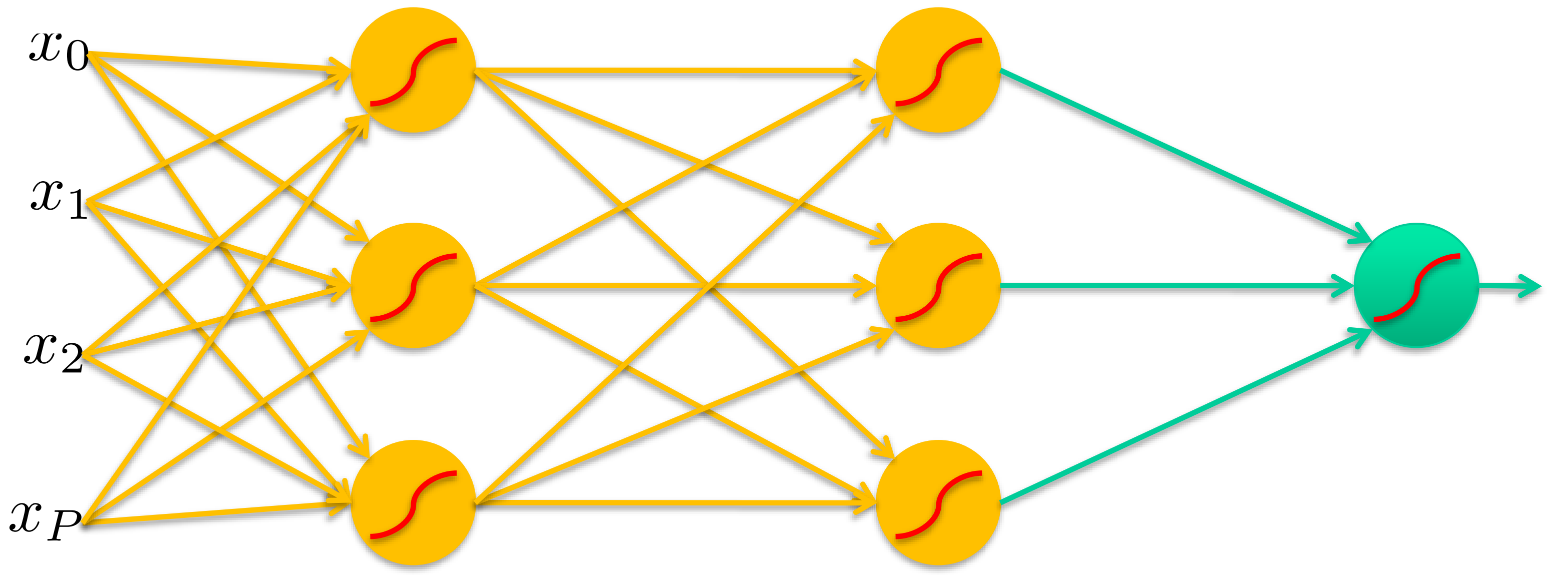
Feed-forward networks

- Intermediate predictions multiplied by second layer of weights
- Predictions are fed forward through the network to classify



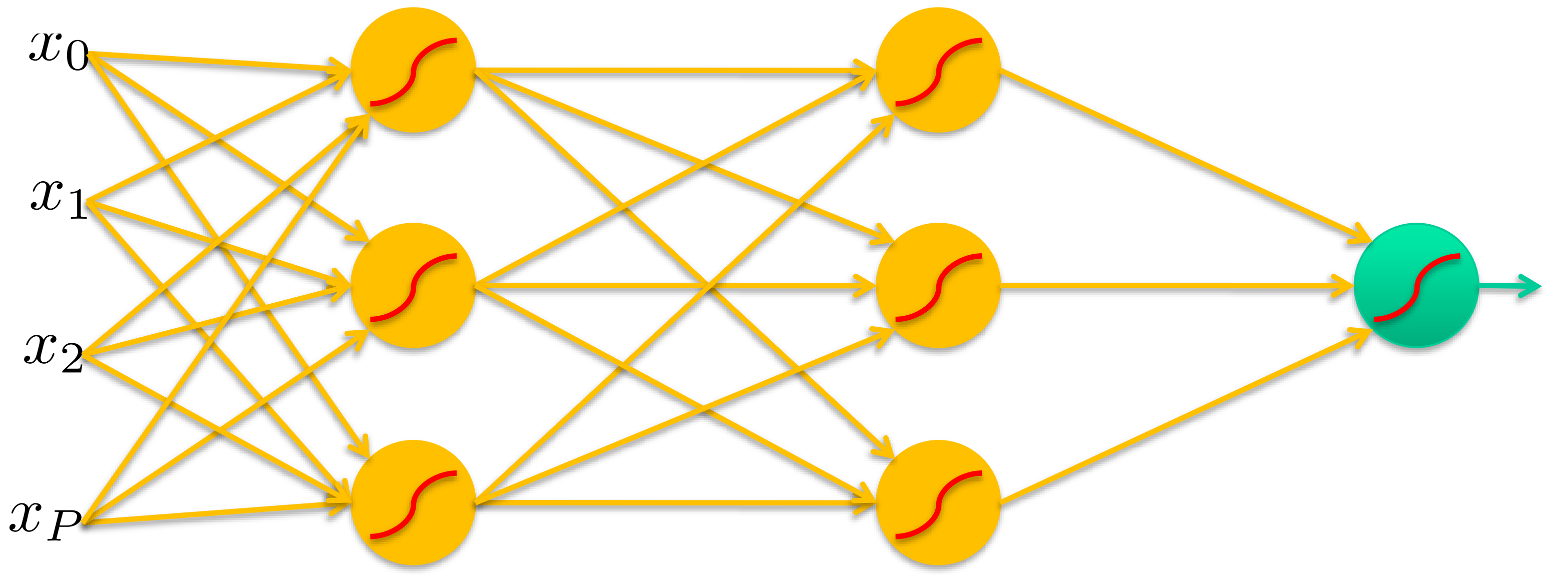
Feed-forward networks

- Compute second set of intermediate predictions



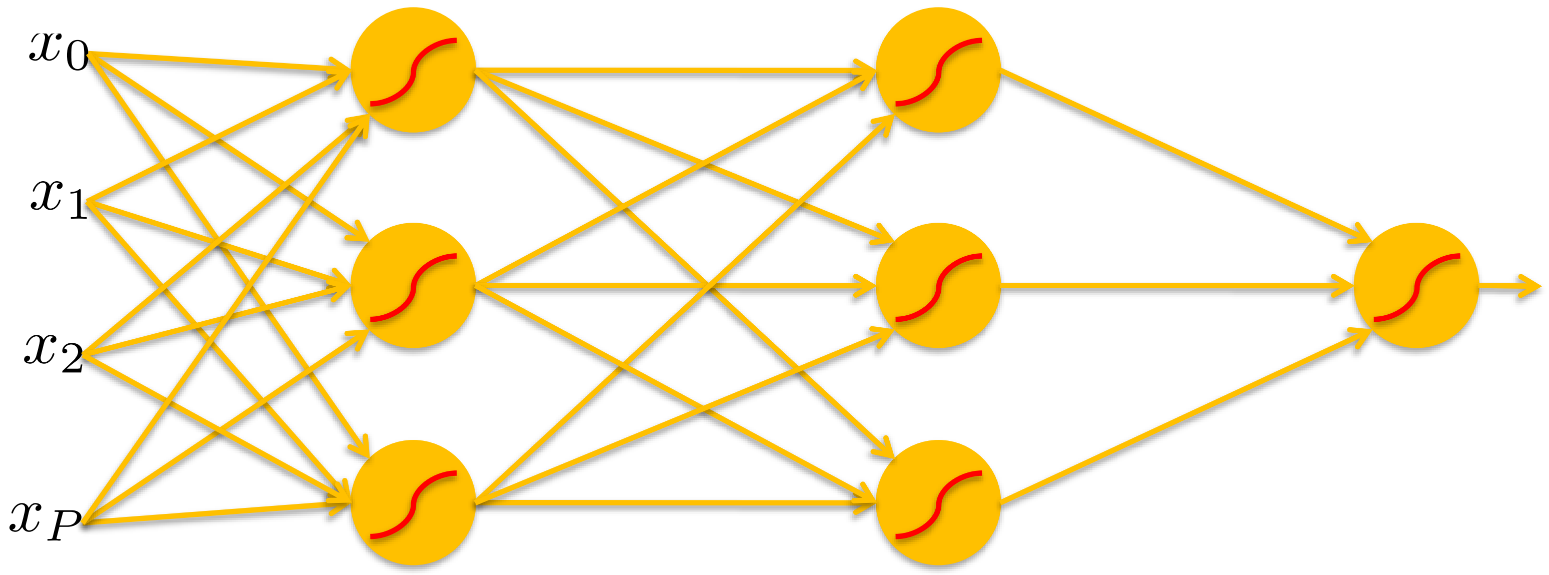
Feed-forward networks

- Multiply by final set of weights



Feed-forward networks

- Compute output (e.g. probability of a particular class being present in the sample)



Deep neural networks

- Lots of hidden layers
- Depth = power (usually)

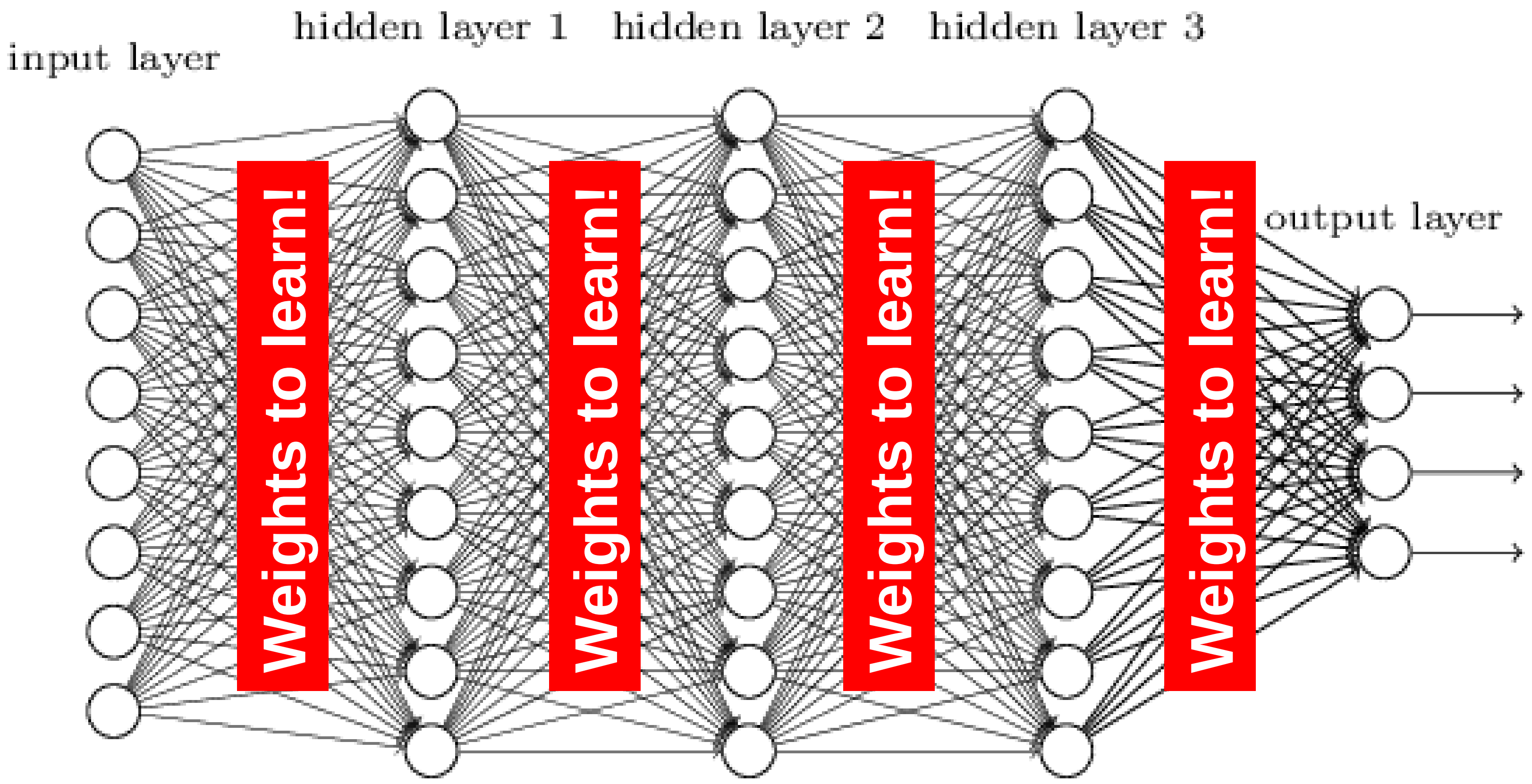


Figure from http://neuralnetworksanddeeplearning.com/chap5.html

Goals

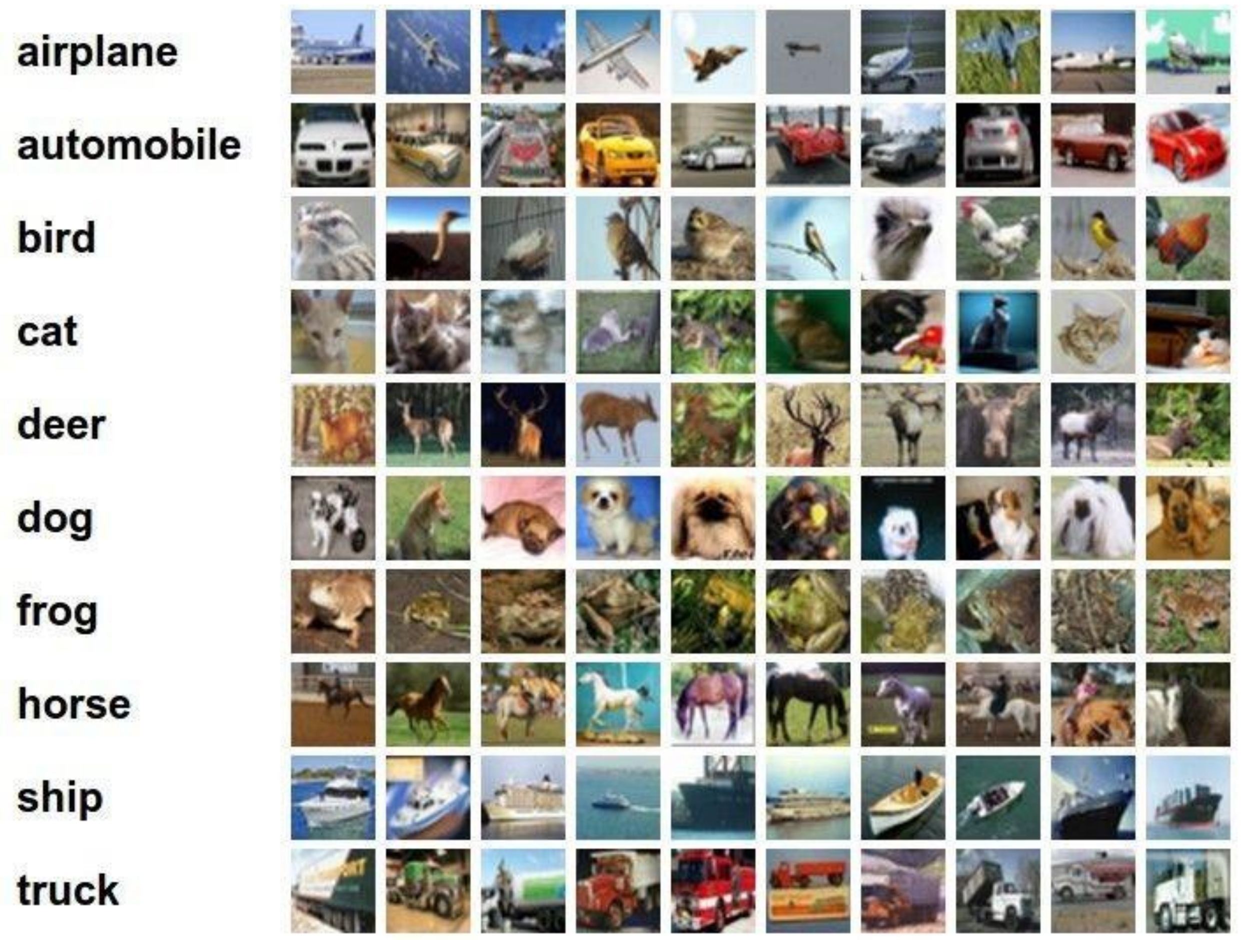
How do we train deep networks?

- No closed-form solution for the weights (can’t set up a system **A*w = b**, solve for **w**)
- We will iteratively find such a set of weights that allow the outputs to match the desired outputs
- We want to minimize a loss function (a function of the weights in the network)
- For now, let’s simplify and assume there’s a single layer of weights in the network, and no activation function (i.e., output is a linear combination of the inputs)

Finding the optimal **w**: Example

- Suppose **w** is just a scalar, w, that can only take values 1, 2, 3
- Suppose L(w=1) = 2, L(w=2) = 5, L(w=3) = 0.5
- Find the optimal w as argmin_w L(w)
- What is the optimal w?
- Now suppose L^(w) = L(w) + ||w||
- L^(1) = 2+1 = 3
- L^(2) = ? L^(3) = ?
- Now what is the optimal w?

Classification goal



Example dataset: **CIFAR-10**
10 labels
50,000 training images
each image is **32x32x3**
10,000 test images.

Classification scores

$$f(x, W) = Wx$$

$$\xrightarrow{f(x,W)}$$

10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Linear classifier

$$\underbrace{\boxed{f(x, W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\,\overset{3072\text{x}1}{\boxed{x}} \quad \underset{10\text{x}1}{\boxed{(+b)}}$$

[32x32x3]
array of numbers 0...1

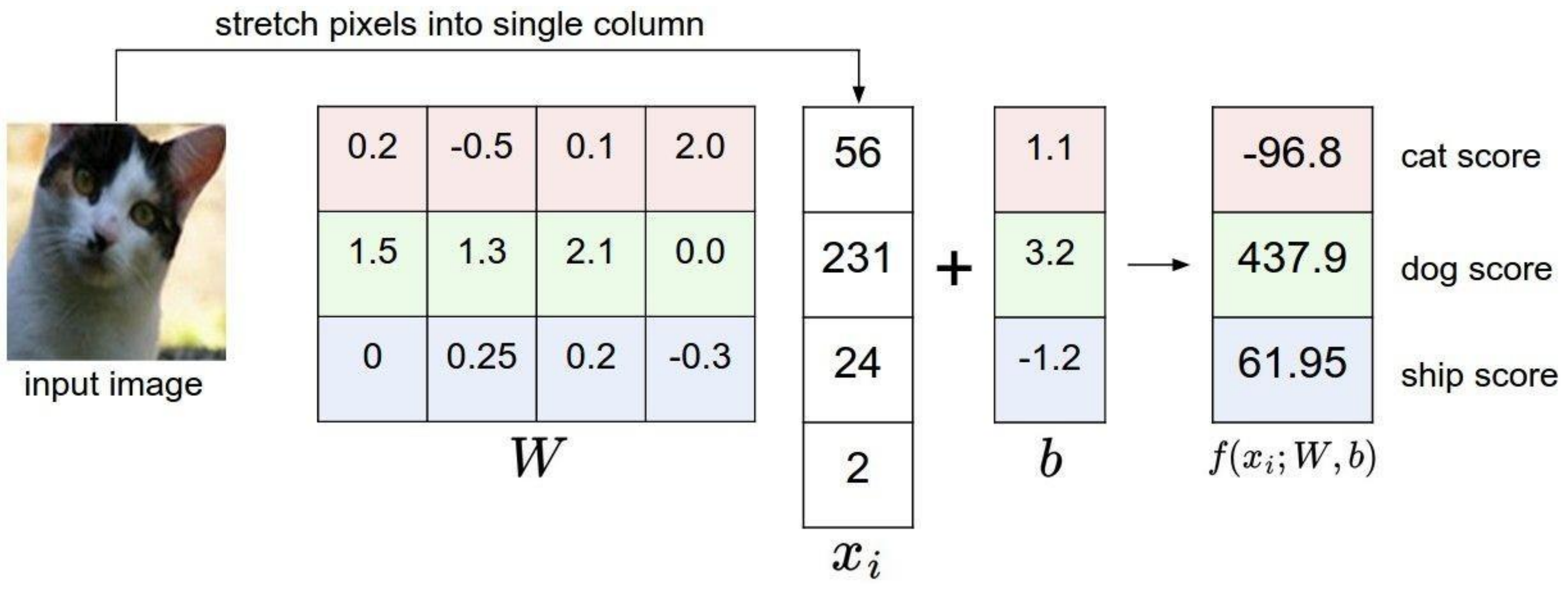
$\longrightarrow$

10 numbers, indicating class scores

parameters, or "weights"

Linear classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Linear classifier

Going forward: Loss function/Optimization

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Adapted from Andrej Karpathy

Linear classifier

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

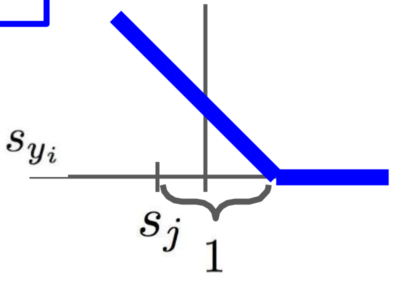
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**



Hinge loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Want: $s_{y_i} >= s_j + 1$
i.e. $s_j - s_{y_i} + 1 <= 0$

If true, loss is 0
If false, loss is magnitude of violation

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Hinge loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Hinge loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the
scores vector: $s = f(x_i, W)$

the loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

$= \max(0, 1.3 - 4.9 + 1)$
$\quad + \max(0, 2.0 - 4.9 + 1)$
$= \max(0, -2.6) + \max(0, -1.9)$
$= 0 + 0$
$= 0$

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Hinge loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the
scores vector: $s = f(x_i, W)$

the loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3 + 1)
+ max(0, 5.6 + 1)
= 6.3 + 6.6
= 12.9

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Hinge loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 12.9)/3
= 15.8 / 3 = **5.3**

Adapted from Andrej Karpathy

Linear classifier: Hinge loss

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0.
Is this W unique?

No! 2W also has L = 0!
How do we choose between W and 2W?

Weight Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth / pooling, etc

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data

Adapted from Fei-Fei, Johnson, Yeung

Weight Regularization

Expressing preferences

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

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

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

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

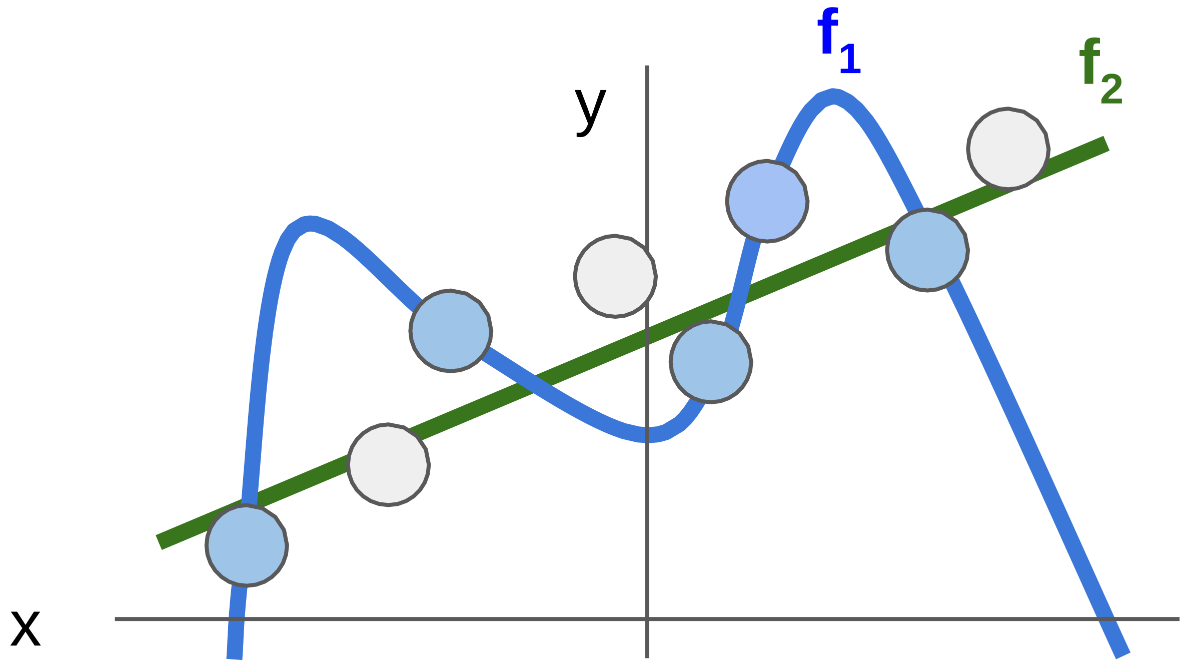
L2 Regularization

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

L2 regularization likes to "spread out" the weights

Weight Regularization

Preferring simple models



Regularization pushes against fitting the data *too* well so we don’t fit noise in the data

Another loss: Cross-entropy

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}}$$

where $\boxed{s=f(x_i;W)}$

Want to maximize the log likelihood, or (for a loss function) to **minimize** the *negative log likelihood of the correct class*:

$$L_i = \boxed{-}\boxed{\log}\boxed{P(Y=y_i|X=x_i)}$$

Adapted from Andrej Karpathy

Another loss: Cross-entropy

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

Probabilities must be >= 0

Probabilities must sum to 1

	unnormalized log probabilities		unnormalized probabilities		probabilities
cat	**3.2**	$\xrightarrow{\text{exp}}$	**24.5**	$\xrightarrow{\text{normalize}}$	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$\rightarrow$ L_i = -log(0.13)
= **0.89**

Aside:
- This is multinomial logistic regression
- Choose weights to maximize the likelihood of the observed x/y data (Maximum Likelihood Estimation; more discussion in CS 1675)

Adapted from Fei-Fei, Johnson, Yeung

Another loss: Cross-entropy

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

	unnormalized log-probabilities / logits		unnormalized probabilities		probabilities		correct probs
cat	**3.2**	exp	**24.5**	normalize	**0.13**	→ compare ←	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Kullback–Leibler divergence

$$D_{KL}(\underline{P}\|\underline{Q}) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Adapted from Fei-Fei, Johnson, Yeung

Other losses

- Triplet loss (Schroff, FaceNet, CVPR 2015)

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

a denotes anchor
p denotes positive
n denotes negative

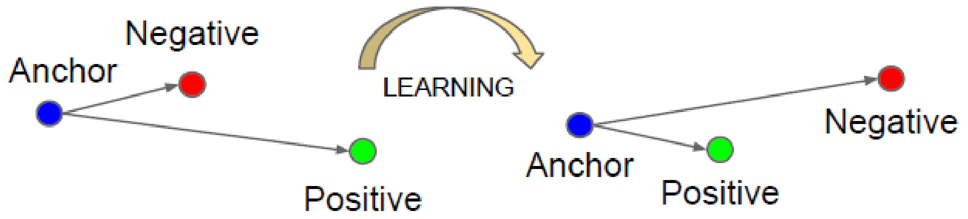


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

- Anything you want! (almost)

Training

To minimize loss, use gradient descent

How to minimize the loss function?

In 1-dimension, the derivative of a function is:

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

In multiple dimensions, the **gradient** is the vector of (partial derivatives):

That is, for $f\colon \mathbf{R}^n \to \mathbf{R}$, its gradient $\nabla f\colon \mathbf{R}^n \to \mathbf{R}^n$ is defined at the point $\mathrm{p} = (x_1, \ldots, x_n)$ in *n*-dimensional space as the vector:[b]

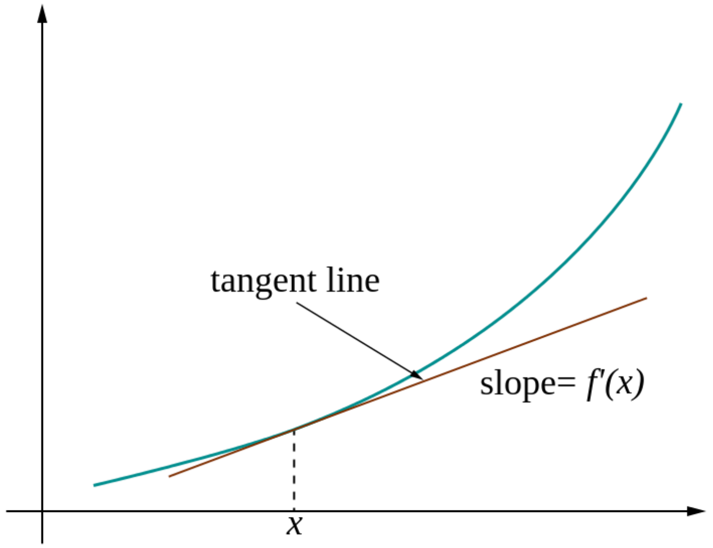
$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}.$$

The nabla symbol ∇, written as an upside-down triangle and pronounced "del", denotes the vector differential operator.

Adapted from Andrej Karpathy, definition/equation from https://en.wikipedia.org/wiki/Gradient

Loss gradients

- Denoted as (diff notations): $\frac{\partial E}{\partial w_{ji}^{(1)}}$ $\nabla_W L$
- i.e. how does the loss change as a function of the weights
- We want to change the weights in such a way that makes the loss decrease as fast as possible



current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

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

?,
?,…]

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

Andrej Karpathy

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

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

?,…]

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

This is silly. The loss is just a function of W:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Calculus

$\nabla_W L$ = ...

Andrej Karpathy

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

dW = ...
(some function
data and W)

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

Example of gradient calculation

- f(**x, w**) = dot(**w, x**) = w1*x1 + w2*x2 + … + wD*xD
- d f(**x, w**) / d w1 = ?
- d f(**x, w**) / d w1 = x1
- d f(**x, w**) / d w2 = x2
- …
- Gradient of f(**x, w**) wrt **w** is [x1 x2 … xD] i.e. **x**

Gradient descent

- We'll update weights
- Move in direction opposite to gradient:

$$\underset{\text{Time}\uparrow}{\mathbf{w}^{(\tau+1)}} = \mathbf{w}^{(\tau)} - \underset{\uparrow\text{Learning rate}}{\eta}\overbrace{\nabla E(\mathbf{w}^{(\tau)})}$$

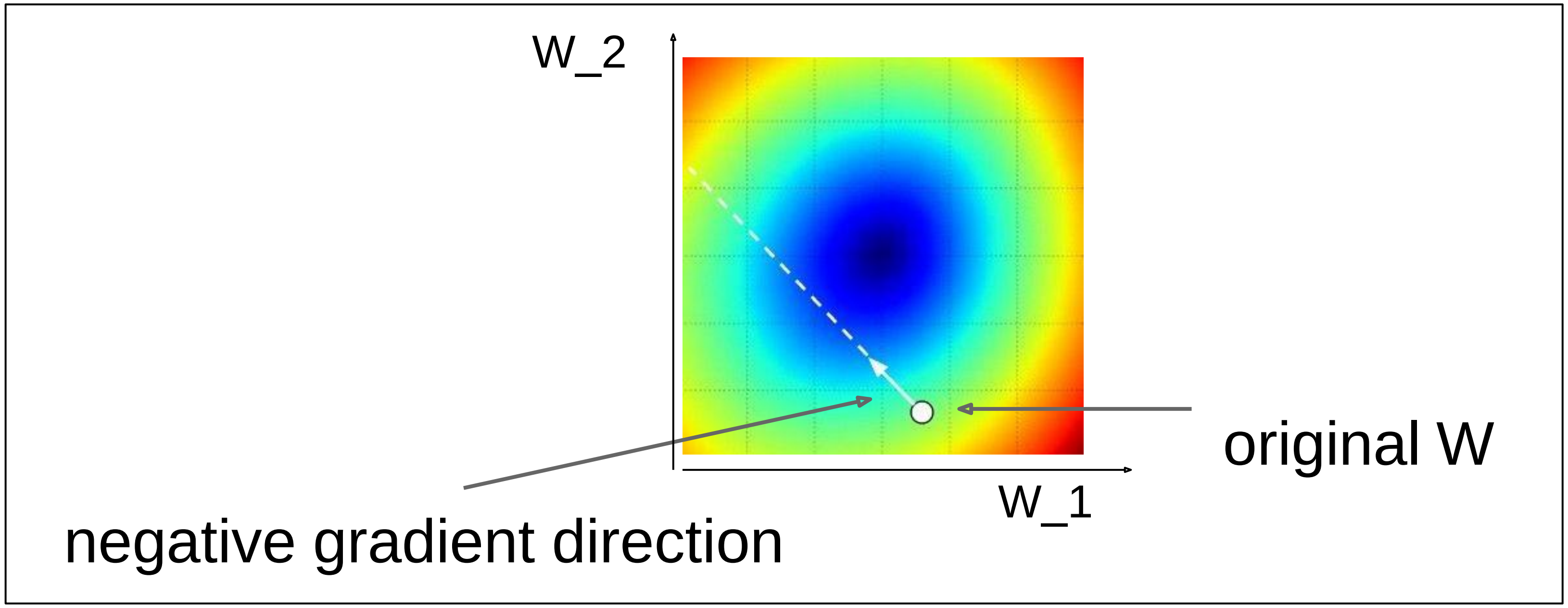
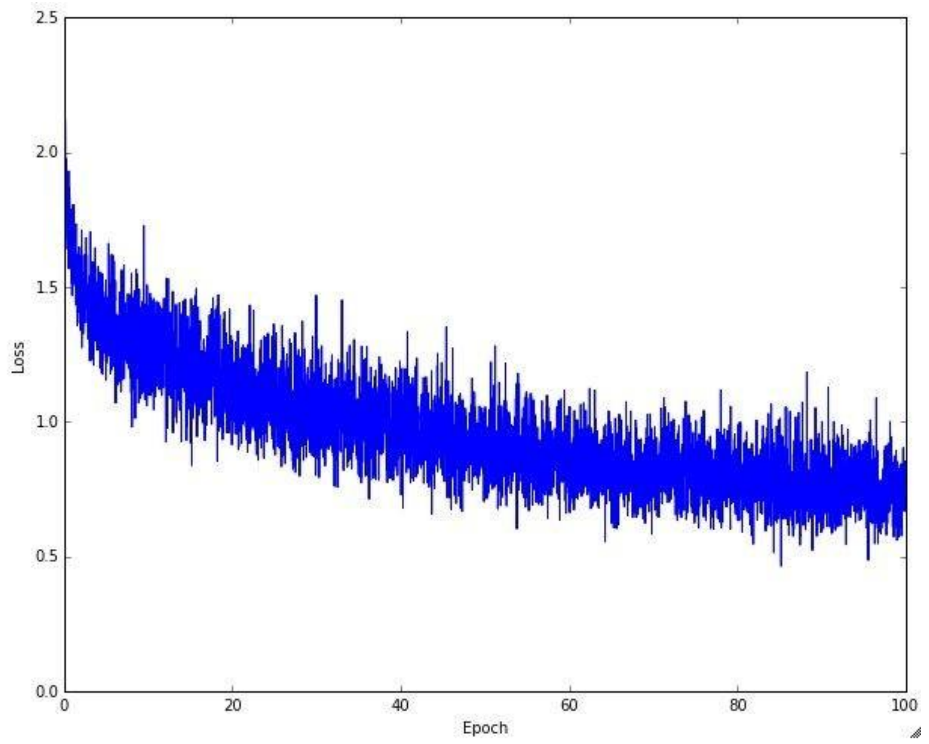


Figure from Andrej Karpathy

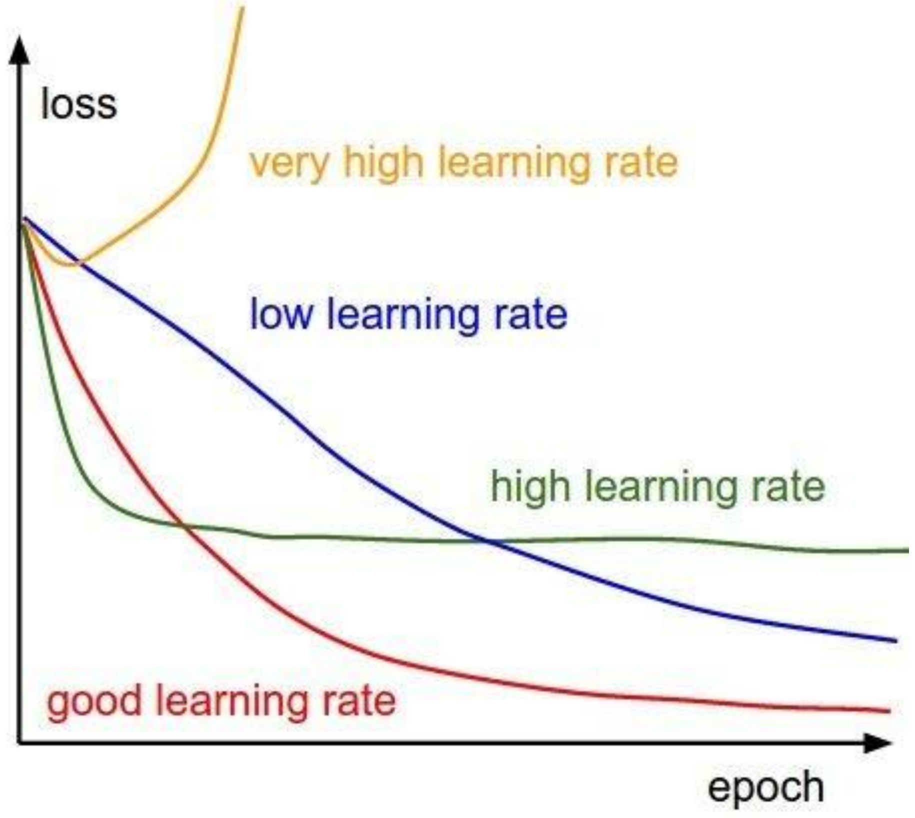
Gradient descent

- Iteratively *subtract* the gradient with respect to the model parameters (w)
- I.e., we're moving in a direction opposite to the gradient of the loss
- I.e., we're moving towards *smaller* loss

Learning rate selection



The effects of step size (or “learning rate”)



Andrej Karpathy

Comments on training algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Local minima – not a huge problem in practice for deep networks.
- Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
- May be hard to set learning rate and to select number of hidden units and layers.
- When in doubt, use validation set to decide on design/hyperparameters.
- Neural networks had fallen out of fashion in 90s, early 2000s; now significantly improved performance (deep networks trained with dropout and lots of data).

Adapted from Ray Mooney, Carlos Guestrin, Dhruv Batra

Gradient descent in multi-layer nets

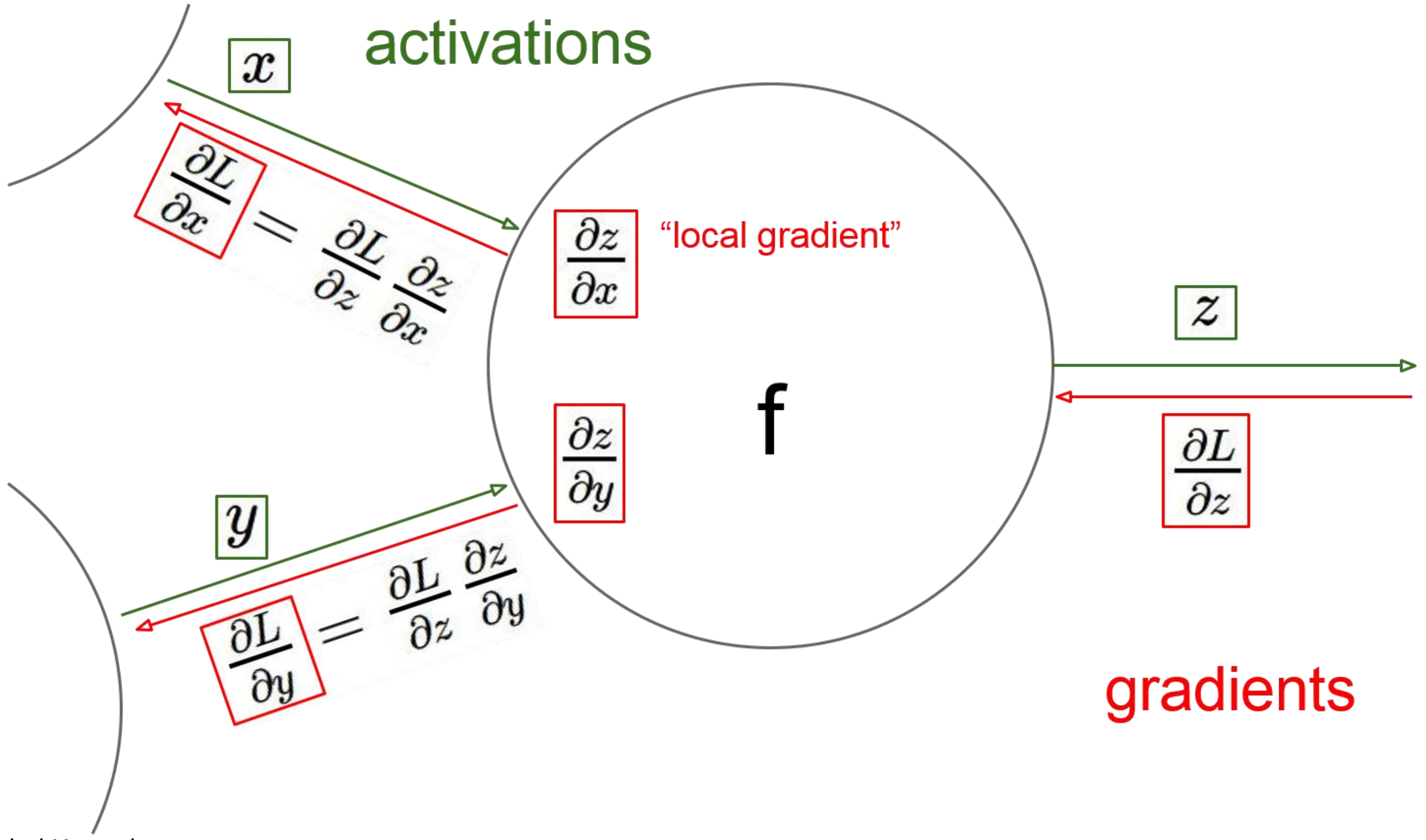
- We’ll update weights
- Move in direction opposite to gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

- How to update the weights at all layers?
- Answer: backpropagation of error from higher layers to lower layers

Gradient descent in multi-layer nets

- **How to update the weights at all layers?**
- Answer: backpropagation of error from higher layers to lower layers



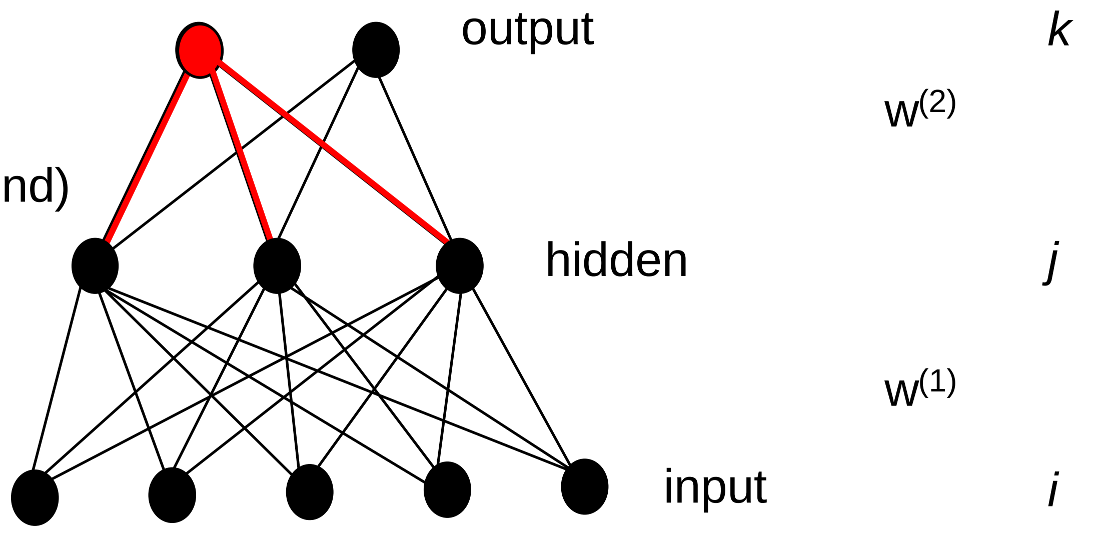
activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
"local gradient"
z
f
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure from Andrej Karpathy

Backpropagation: Graphic example

First calculate error of output units and use this to change the top layer of weights.

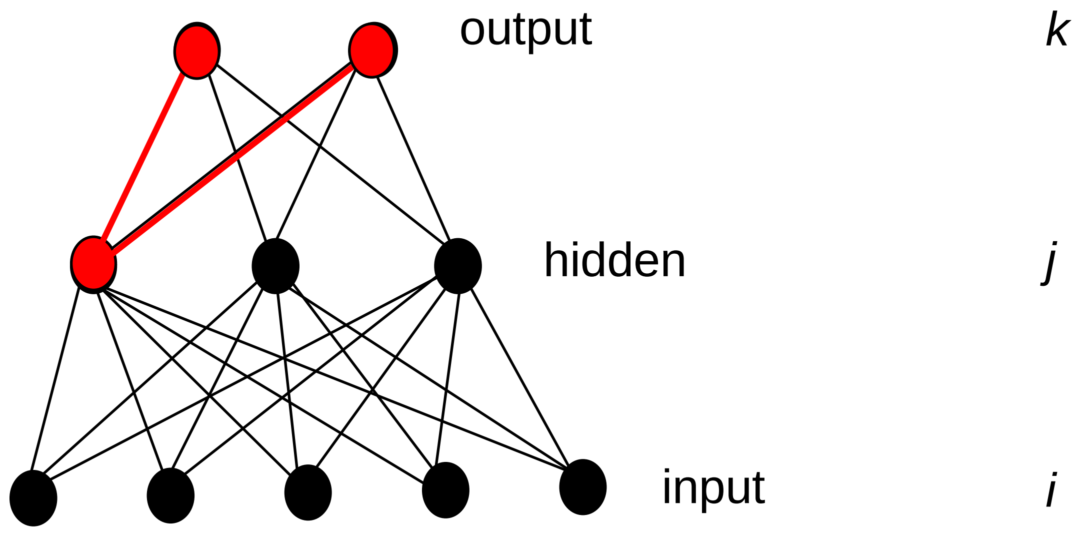
Update weights into *j*
(store diff, update @end)



Adapted from Ray Mooney, equations from Chris Bishop

Backpropagation: Graphic example

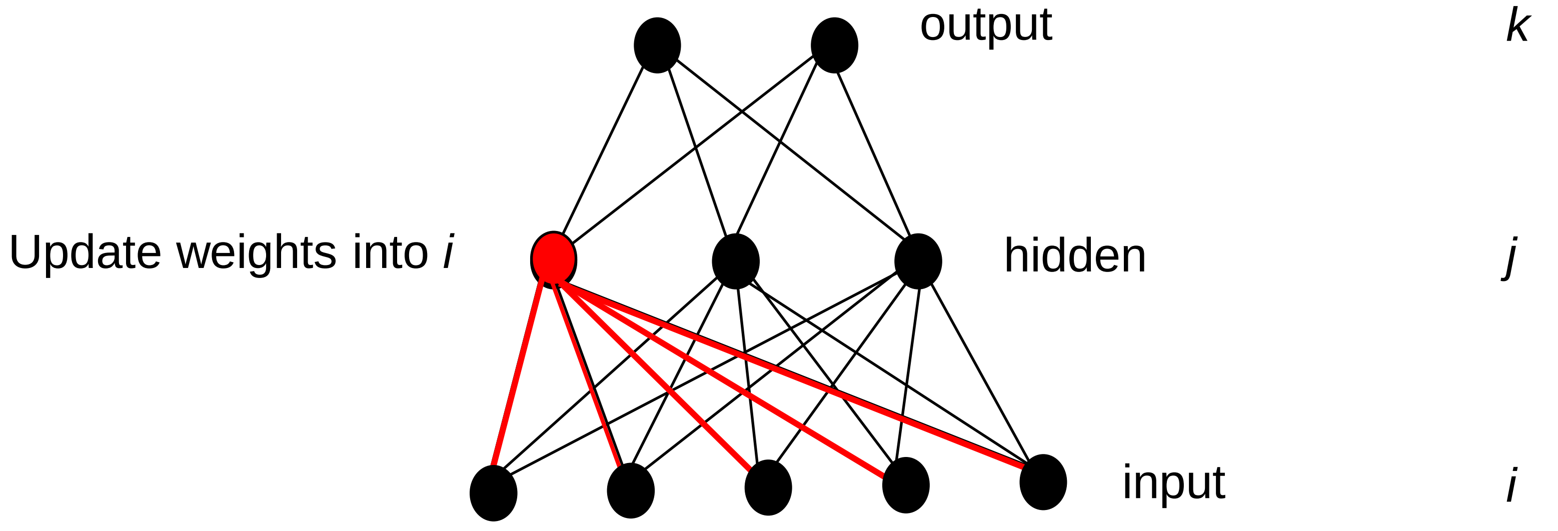
Next calculate error for hidden units based on errors on the output units it feeds into.



Adapted from Ray Mooney, equations from Chris Bishop

Backpropagation: Graphic example

Finally update bottom layer of weights based on errors calculated for hidden units.



Adapted from Ray Mooney, equations from Chris Bishop

Take a deep breath – we’re diving in

Computing gradient for each weight

- We need to move weights in direction opposite to gradient of loss wrt that weight:

  $w_{kj} = w_{kj} - \eta\ dL/dw_{kj}$ (output layer)

  $w_{ji} = w_{ji} - \eta\ dL/dw_{ji}$ (hidden layer)

- Loss depends on weights in an indirect way, so we'll use the chain rule and compute:

  $dL/dw_{kj} = dL/dy_k\ dy_k/da_k\ da_k/dw_{kj}$

  $dL/dw_{ji} = dL/dz_j\ dz_j/da_j\ da_j/dw_{ji}$

$$\boxed{L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)}$$

$$\boxed{y_k = \sigma(a_k) = \frac{1}{1+\exp(-a_k)}}$$

$$\boxed{a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}}$$

Gradient for **output** layer weights

- Loss depends on weights in an indirect way, so we'll use the chain rule and compute:

$$dL/dw_{kj} = dL/dy_k \; dy_k/da_k \; da_k/dw_{kj}$$

- How to compute each of these?
- dL/dy_k : need to know form of error function
  - Example: if $L = (y_k - y_k')^2$, where y_k' is the ground-truth label, then $dL/dy_k = \cancel{2}(y_k - y_k')$
- dy_k/da_k : need to know output layer activation
  - If $h(a_k)=\sigma(a_k)$, then $d\,h(a_k)/d\,a_k = \sigma(a_k)(1-\sigma(a_k))$
- da_k/dw_{kj} :
  - z_j since a_k is a linear combination
  - $a_k = w_{k:}^T z = \Sigma_j \, w_{kj} \, z_j$

Gradient for **hidden** layer weights

- We’ll use the chain rule again and compute:

$$dL/dw_{ji} = dL/dz_j \; dz_j/da_j \; da_j/dw_{ji}$$

- Unlike the previous case (weights for output layer), the error (dL/dz_j) is hard to compute (indirect, need chain rule again)
- We’ll simplify the computation by doing it step by step via *backpropagation* of error
- You could directly compute this term– you will get the same result as with backprop (do as an exercise!)

Backprop – rough notation

- *The following is a framework, slightly imprecise*
- Let us denote the inputs at a layer *i* by in_i, the linear combination of inputs computed at that layer as raw_i, the activation as act_i
- We define a new quantity that will roughly correspond to accumulated error, err_i :

$$err_i = d\ L\ /\ d\ act_i\ *\ d\ act_i\ /\ d\ raw_i$$

- Then we can write the updates as

$$w = w - \eta * err_i * in_i$$

Backprop – formulation

- We'll write the weight updates as follows
  - $w_{kj} = w_{kj} - \eta \, \delta_k \, z_j$ for output units
  - $w_{ji} = w_{ji} - \eta \, \delta_j \, x_i$ for hidden units
- What are δ_k, δ_j?
  - They store error, gradient wrt raw activations (i.e. dL/da)
  - They're of the form $dL/dz_j \; dz_j/da_j$
  - The latter is easy to compute – just use derivative of activation function
  - The former is easy for output – e.g. $(y_k - y_k')$
  - It is harder to compute for hidden layers
  - $dL/dz_j = \sum_k w_{kj} \, \delta_k$ (where did this come from?)

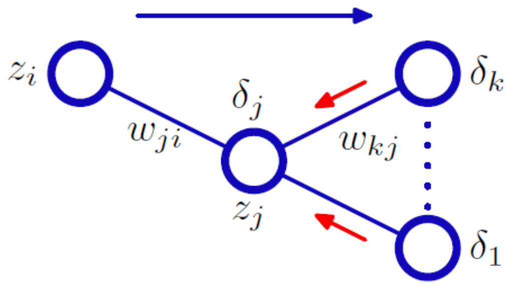


Figure from Chris Bishop

Deriving backprop (Bishop Eq. 5.56)

- In a neural network:

$$\boxed{a_j = \sum_i w_{ji} z_i \qquad z_i = h(a_i)}$$

- Gradient is (using chain rule):

$$\frac{\partial E_n}{\partial w_{ji}} = \boxed{\frac{\partial E_n}{\partial a_j}} \boxed{\frac{\partial a_j}{\partial w_{ji}}} = \boxed{\delta_j} \boxed{z_i}$$

- Denote the "errors" as:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

- Also:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Equations from Bishop

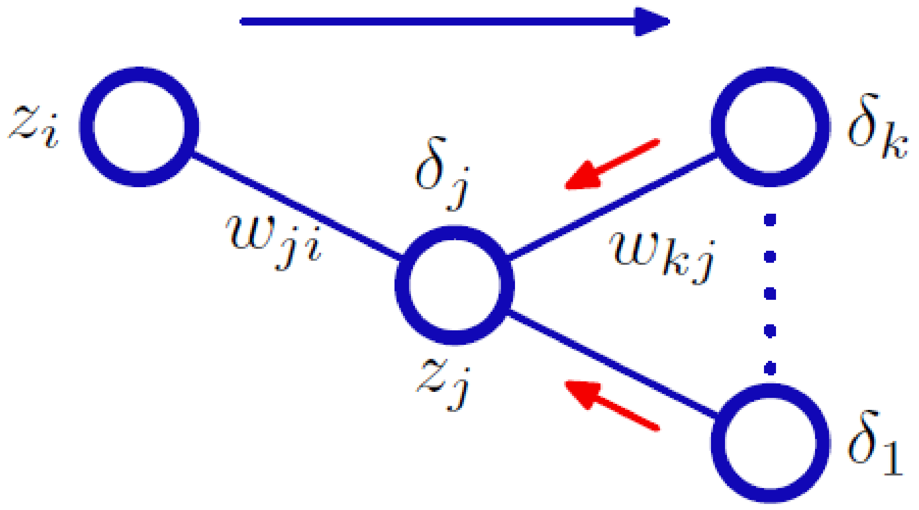
Deriving backprop (Bishop Eq. 5.56)

- For output (identity output, L2 loss): $\delta_k = y_k - t_k$
- For hidden units (using chain rule again):

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \boxed{\frac{\partial E_n}{\partial a_k}} \boxed{\frac{\partial a_k}{\partial a_j}}$$

- Backprop formula:

$$\delta_j = \boxed{h'(a_j)} \sum_k \boxed{w_{kj}} \boxed{\delta_k}$$



Equations from Bishop, also see https://en.wikipedia.org/wiki/Chain_rule#Example

Putting it all together

- Example: use sigmoid at hidden layer and output layer, loss is L2 between true/predicted labels

Example algorithm for sigmoid, L2 error

- Initialize all weights to small random values
- Until convergence (e.g. all training examples' error small, or error stops decreasing) repeat:
  - For each `(x, y'=class(x))` in training set:
    - Calculate network outputs: y_k
    - Compute errors (gradients wrt activations) for each unit:
      - » $\delta_k = y_k (1-y_k) (y_k - y_k')$ for output units
      - » $\delta_j = z_j (1-z_j) \sum_k w_{kj} \delta_k$ for hidden units
    - Update weights:
      - » $w_{kj} = w_{kj} - \eta \delta_k z_j$ for output units
      - » $w_{ji} = w_{ji} - \eta \delta_j x_i$ for hidden units

<u>Recall:</u> $w_{ji} = w_{ji} - \eta \, dE/dz_j \, dz_j/da_j \, da_j/dw_{ji}$

$$\boxed{\delta_j = h'(a_j) \sum_k w_{kj} \delta_k}$$

Adapted from R. Hwa, R. Mooney

Another example

- Two layer network w/ tanh at hidden layer:

$$h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

- Derivative: $h'(a) = 1 - h(a)^2$
- Minimize: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$
- Forward propagation:

$$\begin{aligned} a_j &= \sum_{i=0}^{D} w_{ji}^{(1)} x_i \\ z_j &= \tanh(a_j) \\ y_k &= \sum_{j=0}^{M} w_{kj}^{(2)} z_j \end{aligned}$$

Another example

- Errors at output (identity function at output):

$$\delta_k = y_k - t_k$$

$$\boxed{\delta_j = h'(a_j) \sum_k w_{kj} \delta_k}$$

- Errors at hidden units:

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$

- Derivatives wrt weights:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Same example with graphic and math

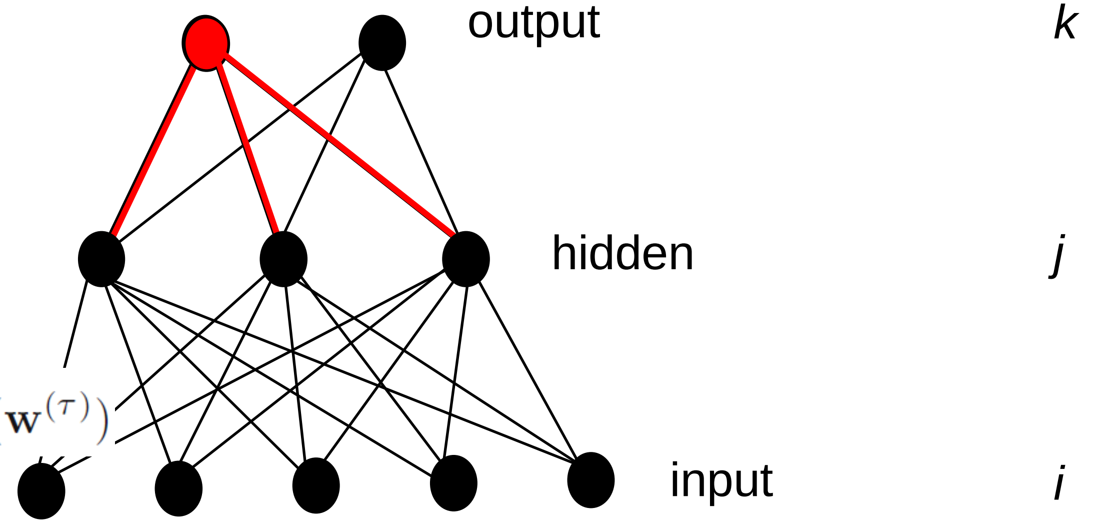
First calculate error of output units and use this to change the top layer of weights.

$$\delta_k = y_k - t_k$$

Update weights into *j*

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$



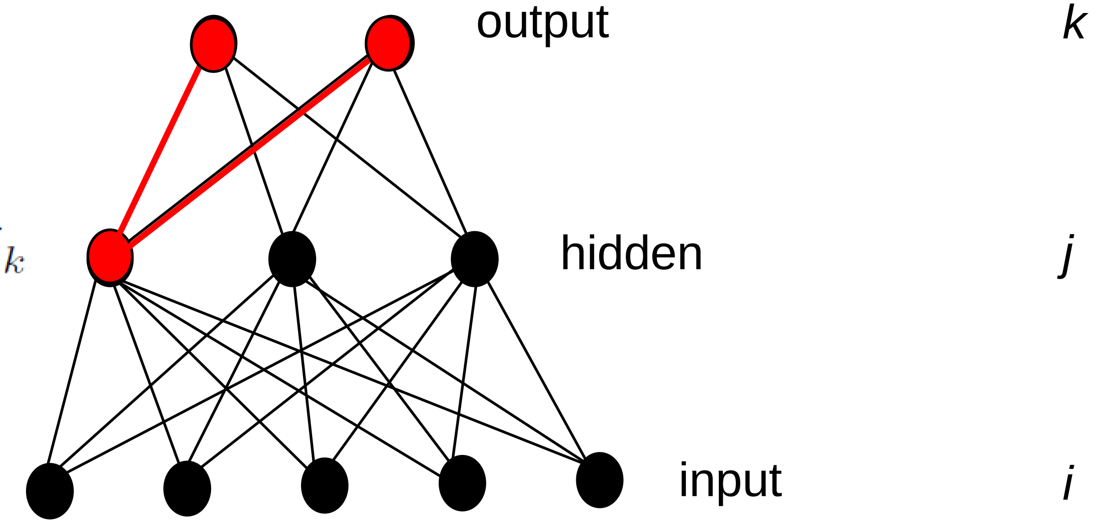
Adapted from Ray Mooney, equations from Chris Bishop

Same example with graphic and math

Next calculate error for hidden units based on errors on the output units it feeds into.

$$\delta_k = y_k - t_k$$

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$



Adapted from Ray Mooney, equations from Chris Bishop

Same example with graphic and math

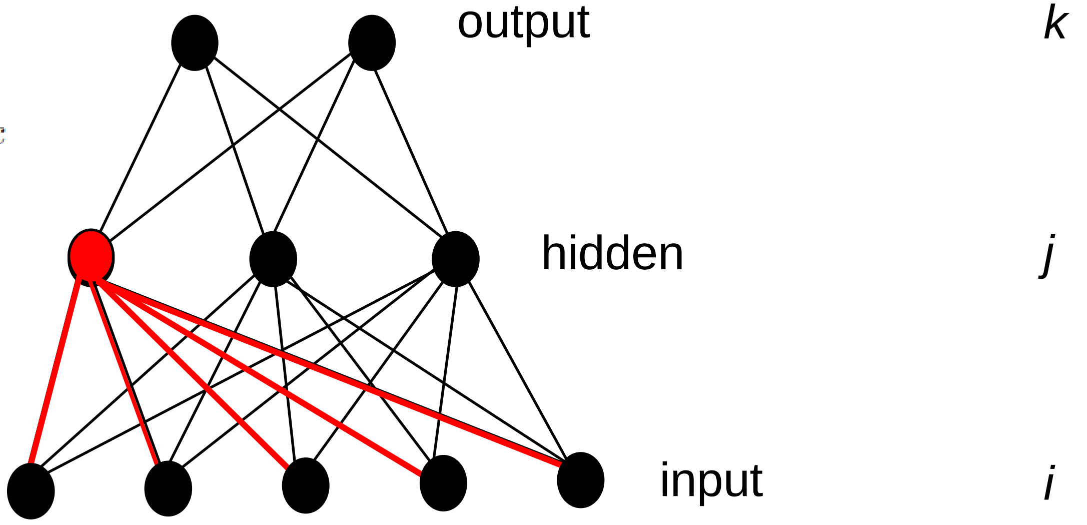
Finally update bottom layer of weights based on errors calculated for hidden units.

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$

Update weights into *i*

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$



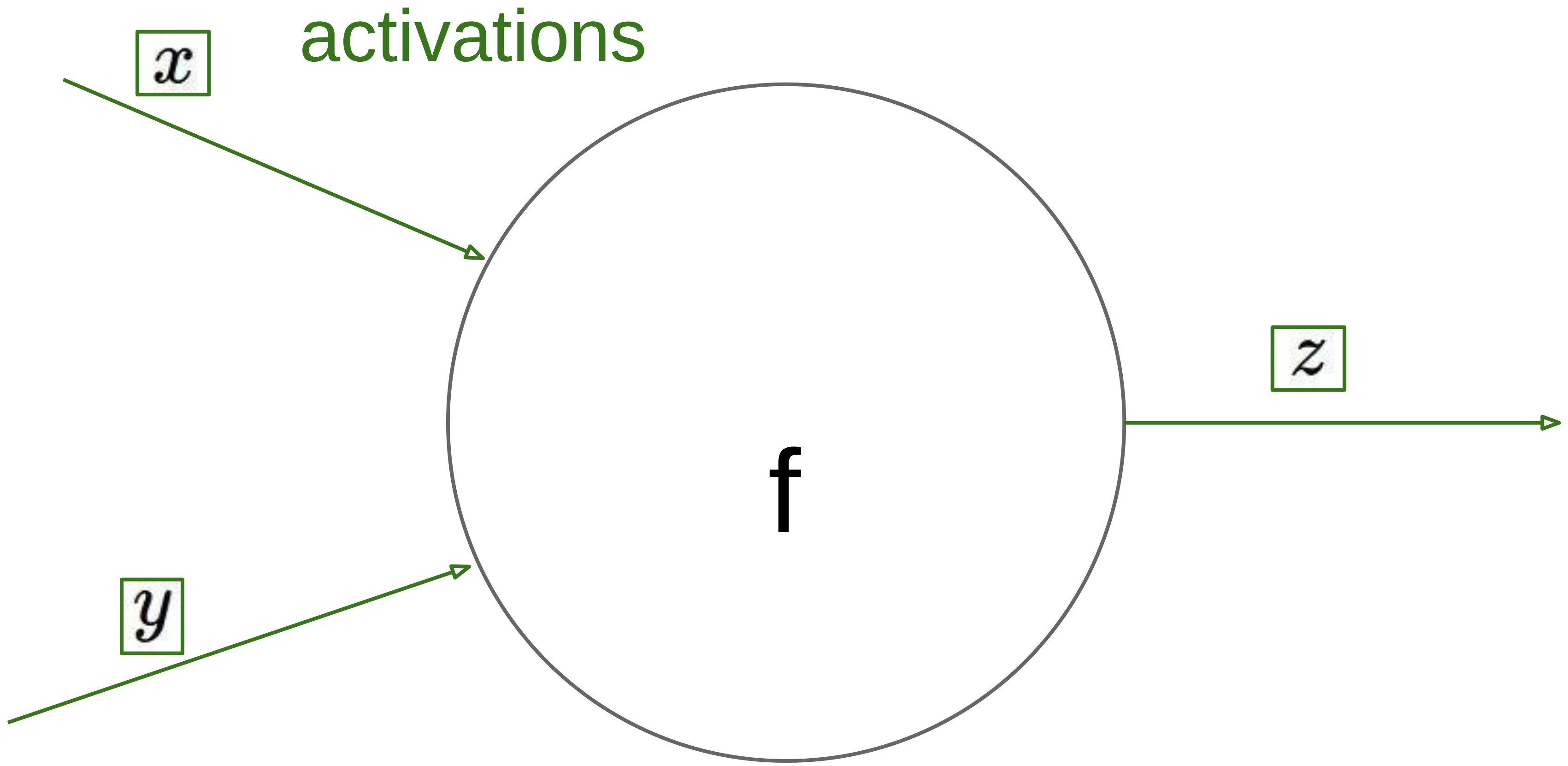
Adapted from Ray Mooney, equations from Chris Bishop

Another way of keeping track of error

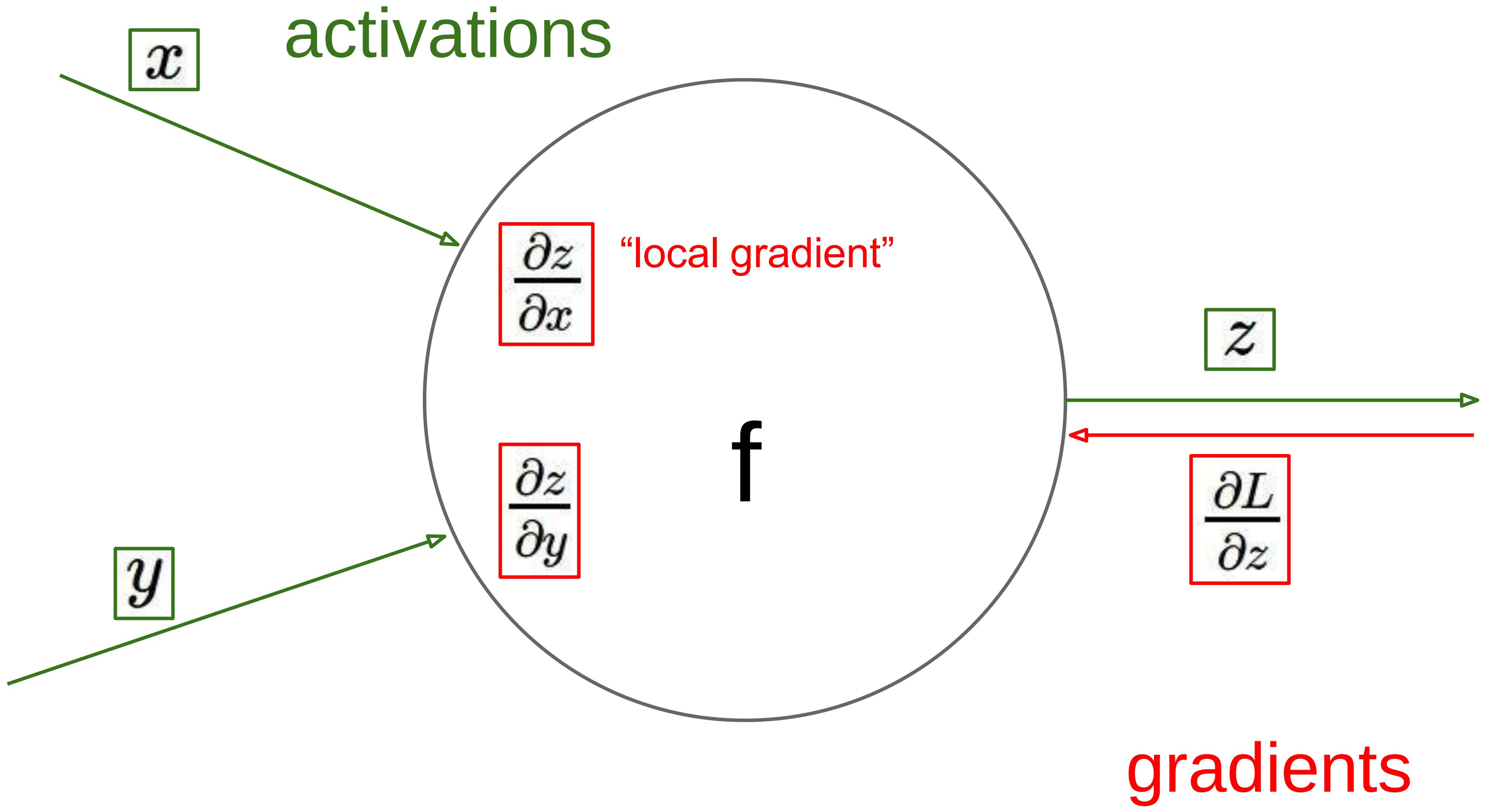
Computation graphs

- Accumulate upstream/downstream gradients at each node
- One set flows from inputs to outputs and can be computed without evaluating loss
- The other flows from outputs (loss) to inputs

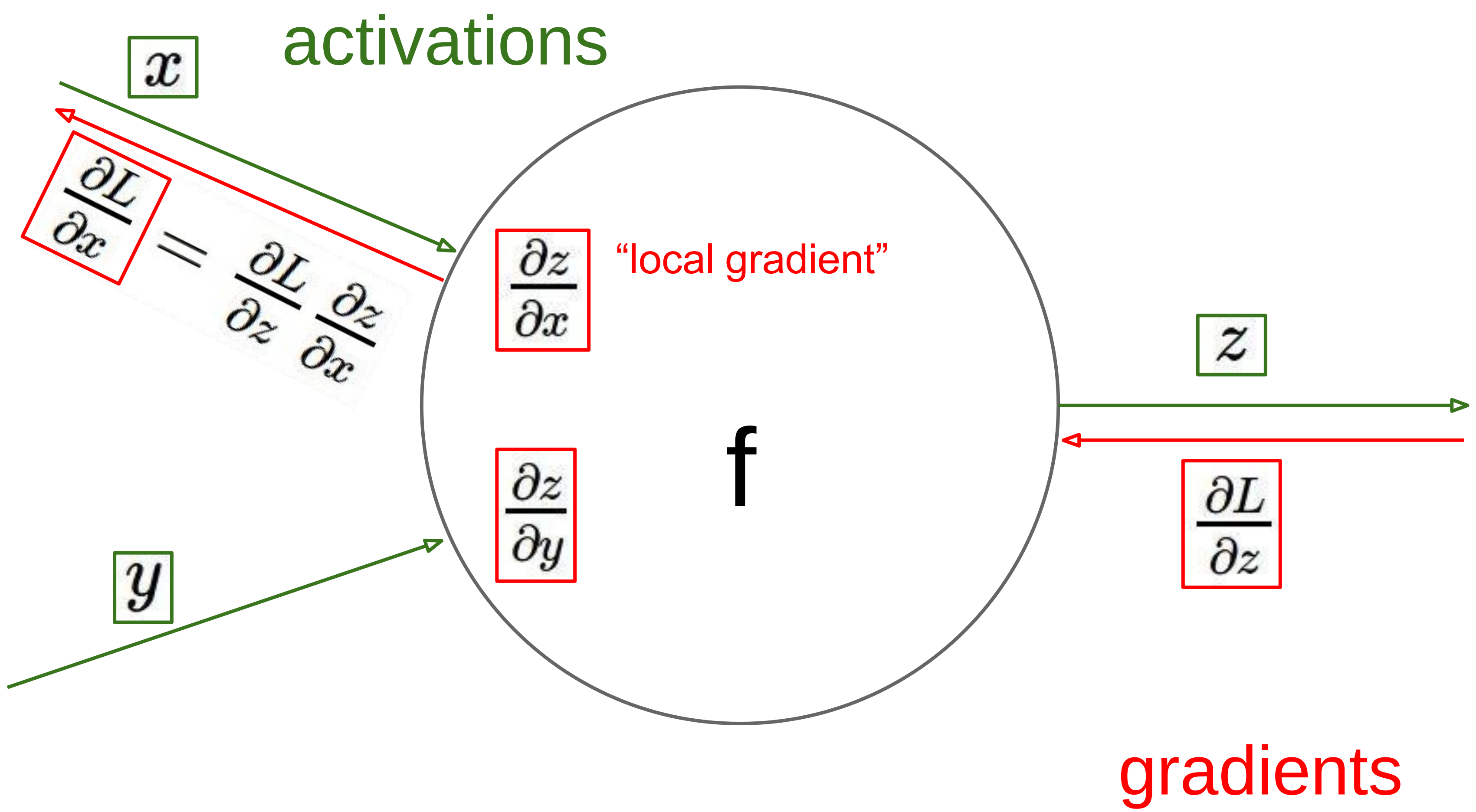
Generic example



Generic example

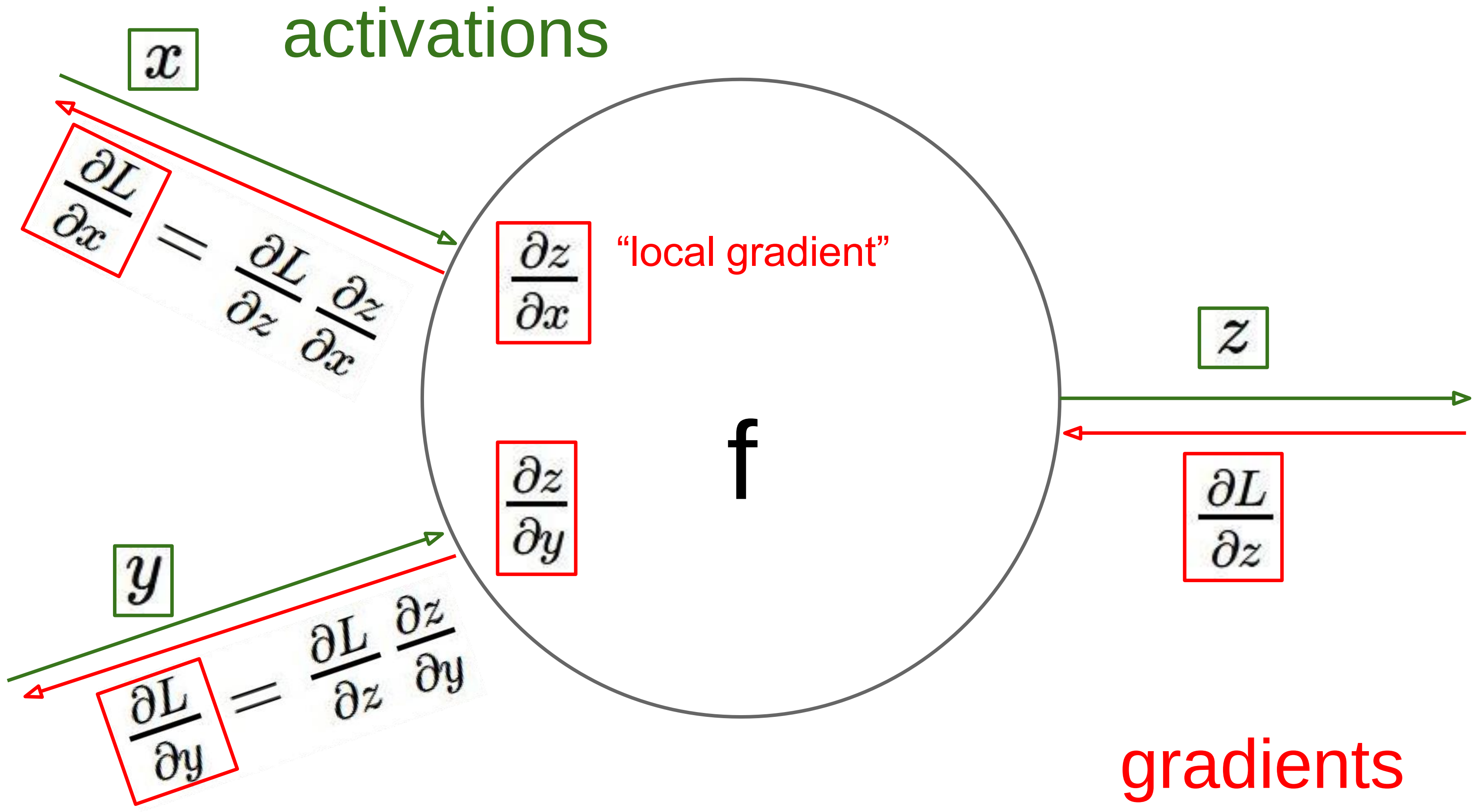

activations
x
y
$\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
gradients

Generic example


activations
x
y
z
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
gradients

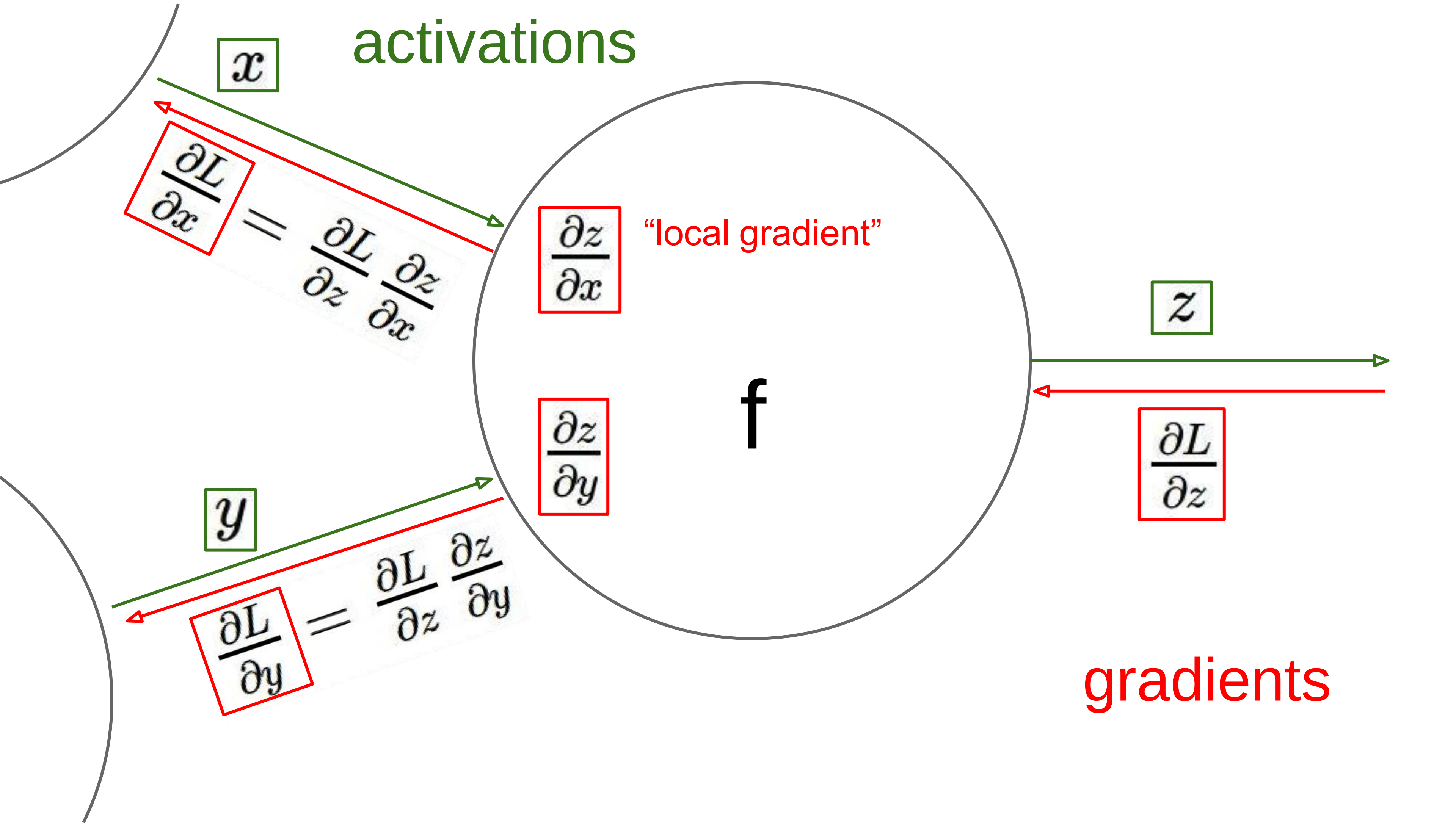
Andrej Karpathy

Generic example


activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
"local gradient"
z
f
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Andrej Karpathy

Generic example

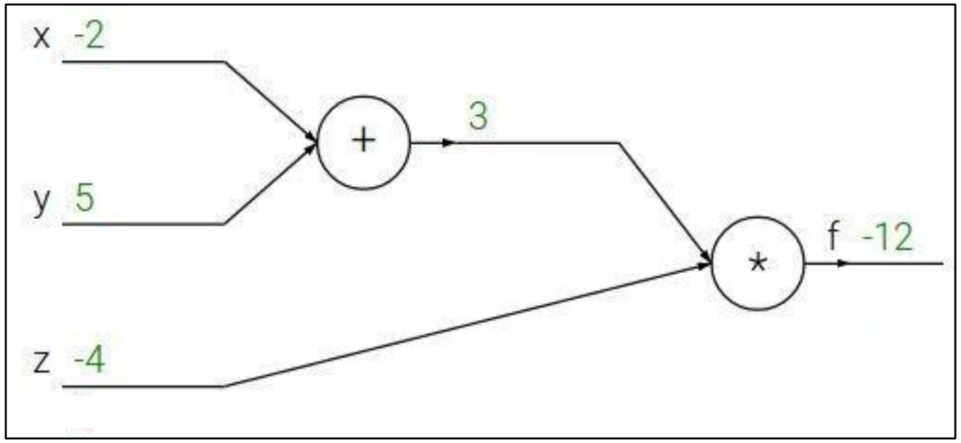

activations
x
y
z
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
$\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
gradients

Andrej Karpathy

Another generic example

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

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



Another generic example

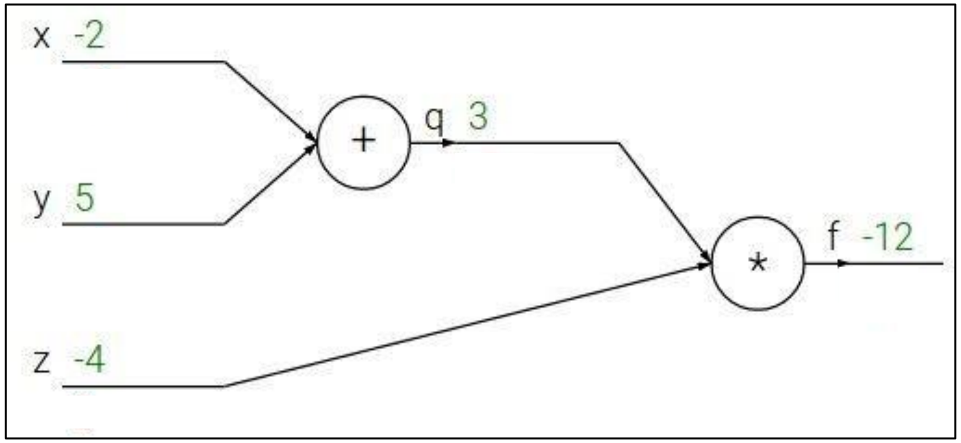
$$f(x, y, z) = (x + y)z$$

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



Another generic example

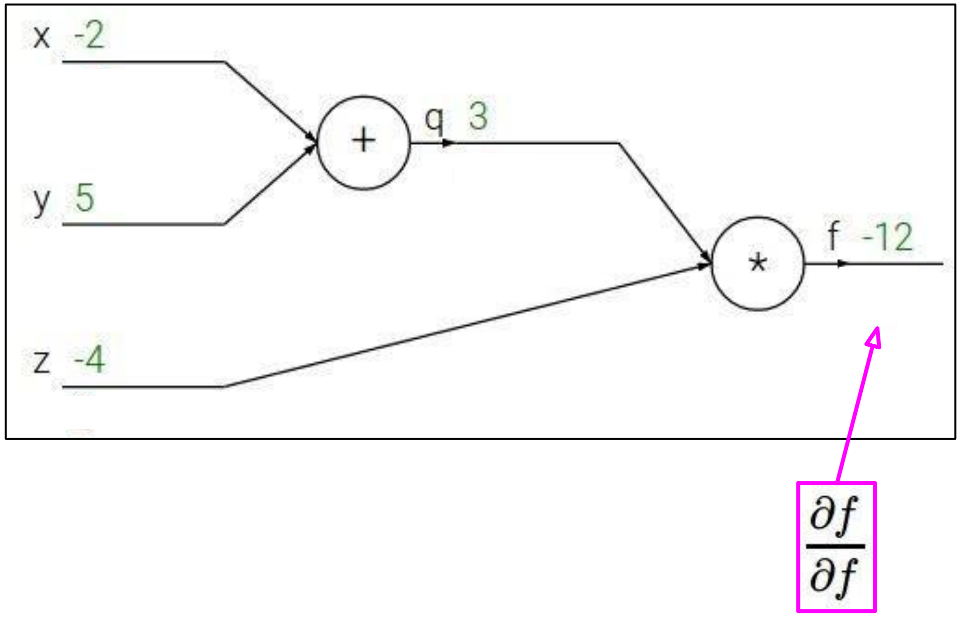
$$f(x,y,z) = (x+y)z$$

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



Another generic example

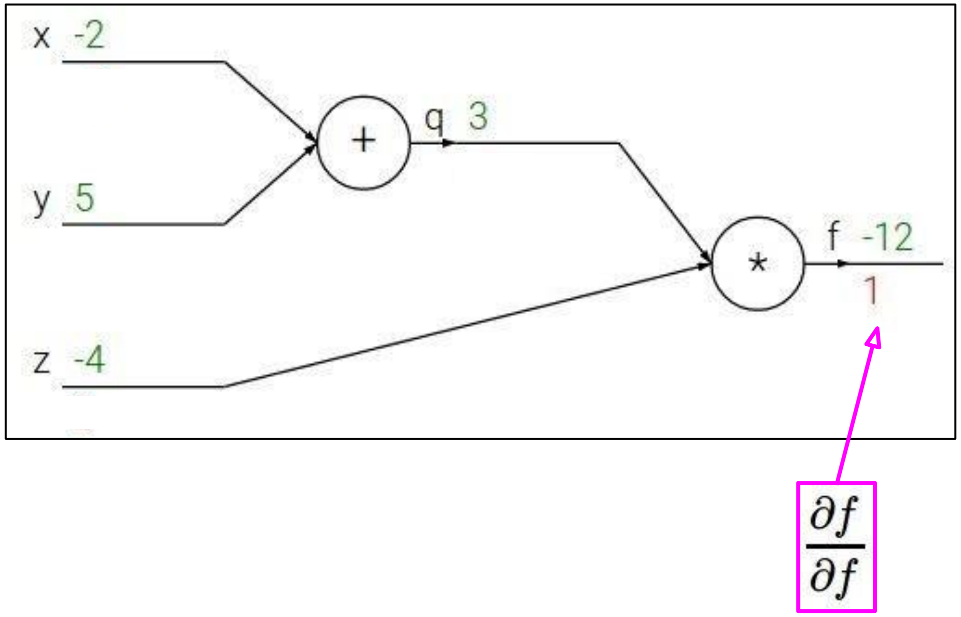
$$f(x, y, z) = (x + y)z$$

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

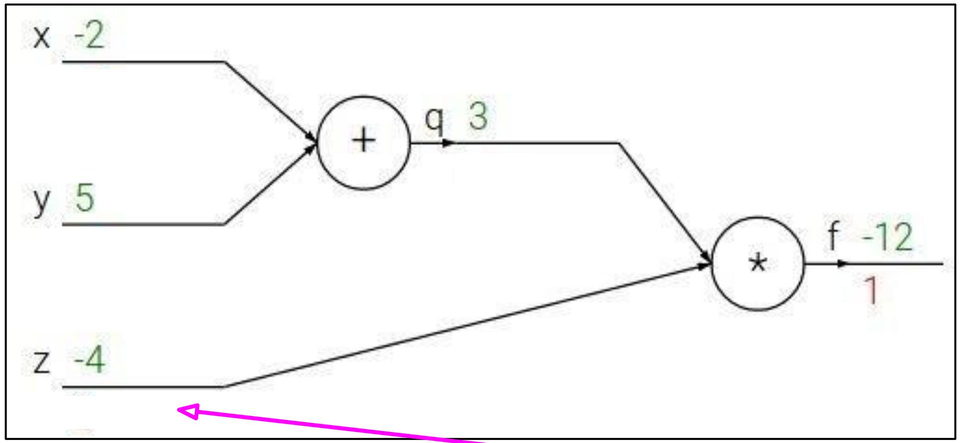


$$\boxed{\frac{\partial f}{\partial f}}$$

Another generic example

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

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



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

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

$$\boxed{\frac{\partial f}{\partial z}}$$

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

Andrej Karpathy

Another generic example

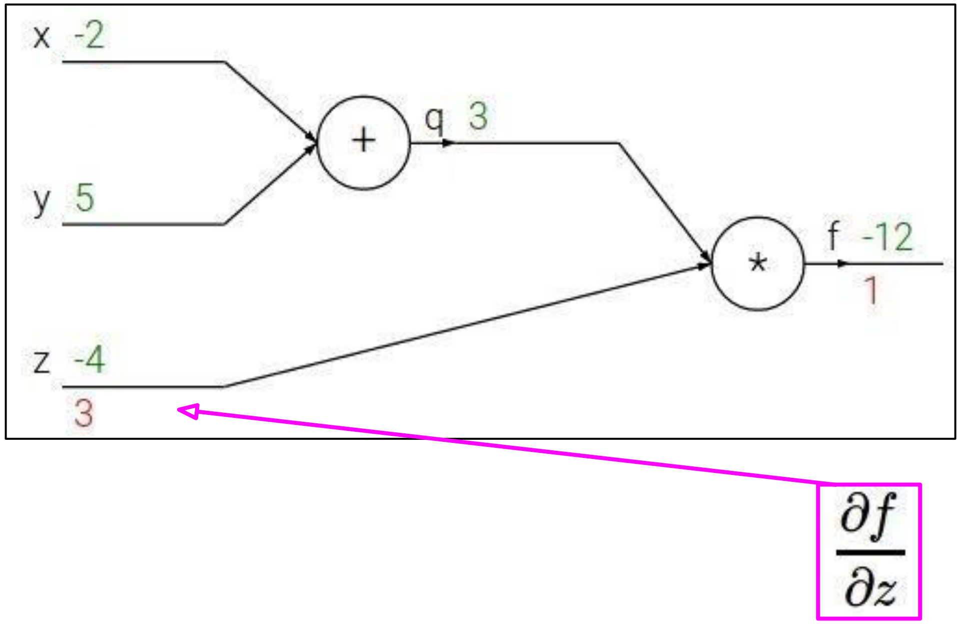
$$f(x,y,z) = (x+y)z$$

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



Another generic example

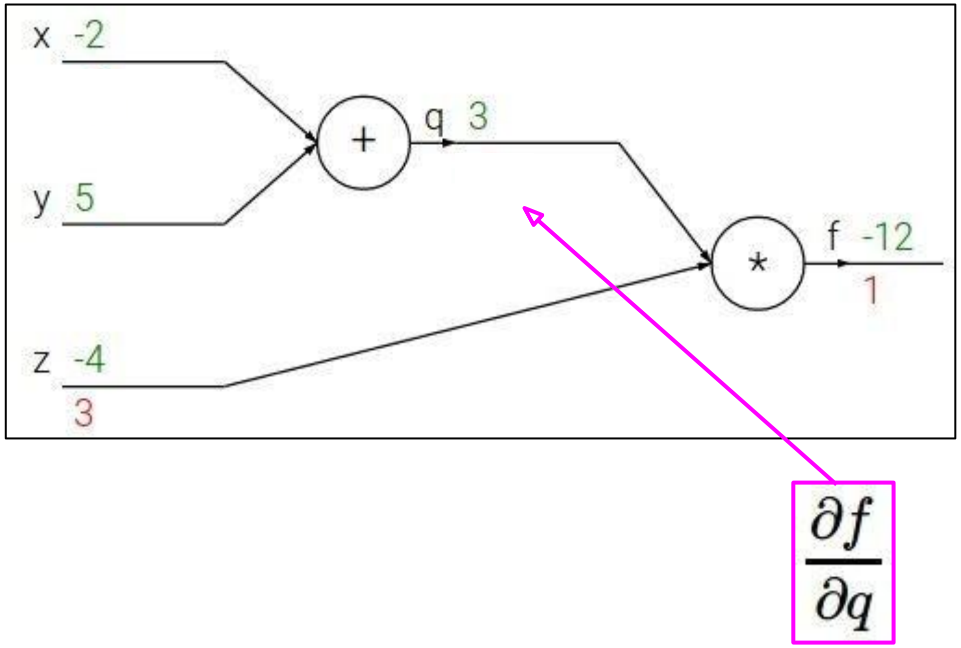
$$f(x,y,z) = (x+y)z$$

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

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

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

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



$\boxed{\frac{\partial f}{\partial q}}$

Andrej Karpathy

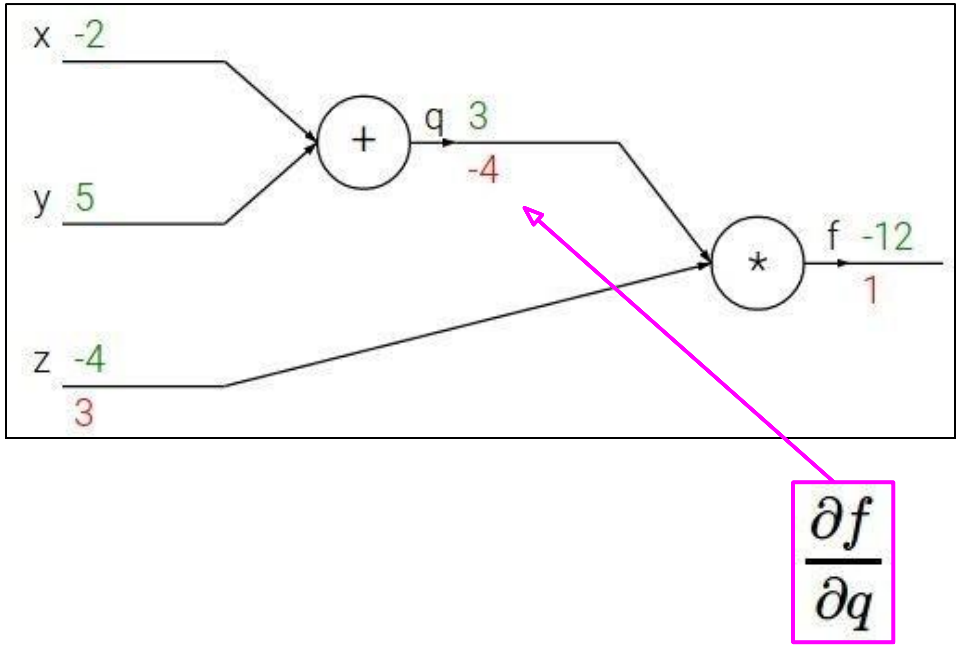
Another generic example

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

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

Another generic example

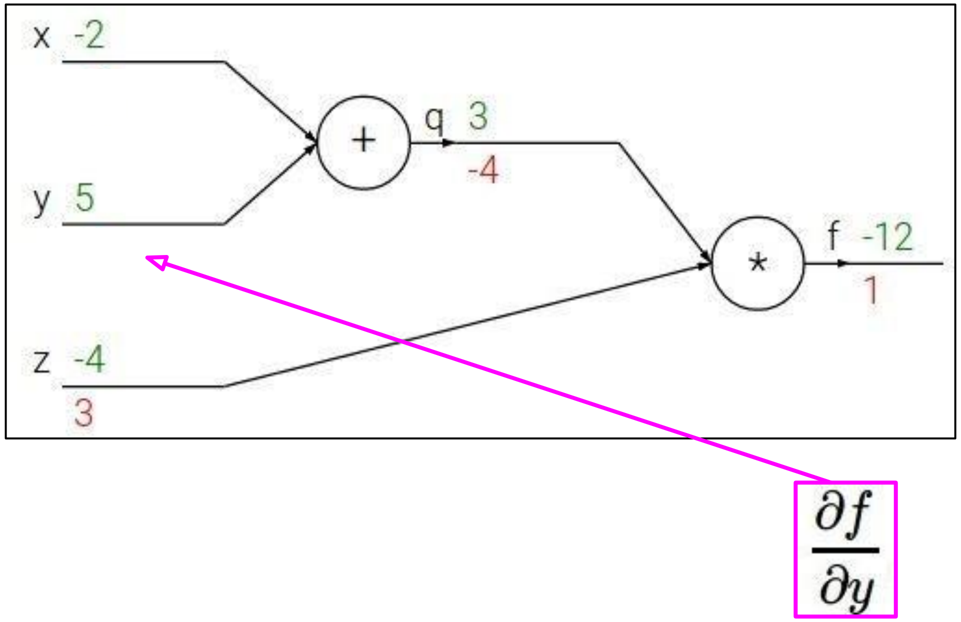
$$f(x, y, z) = (x + y)z$$

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



Another generic example

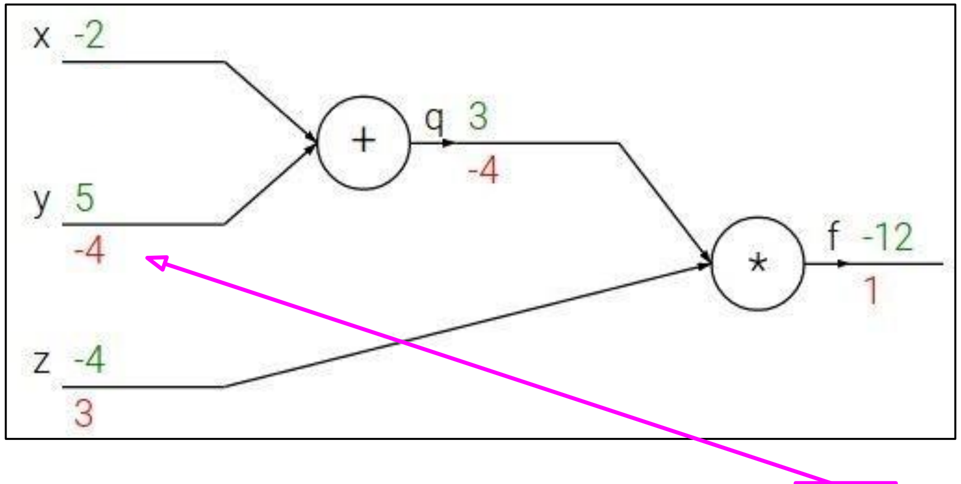
$$f(x, y, z) = (x + y)z$$

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



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

Chain rule:

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

Another generic example

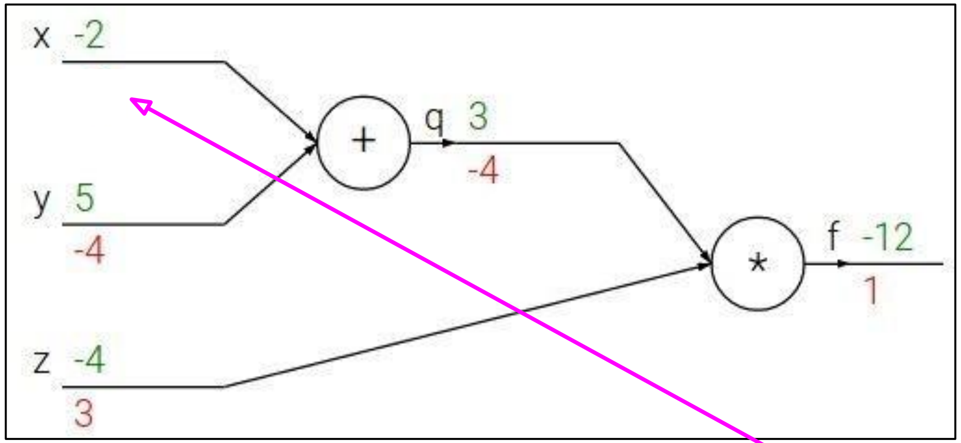
$$f(x, y, z) = (x + y)z$$

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



$\boxed{\frac{\partial f}{\partial x}}$

Another generic example

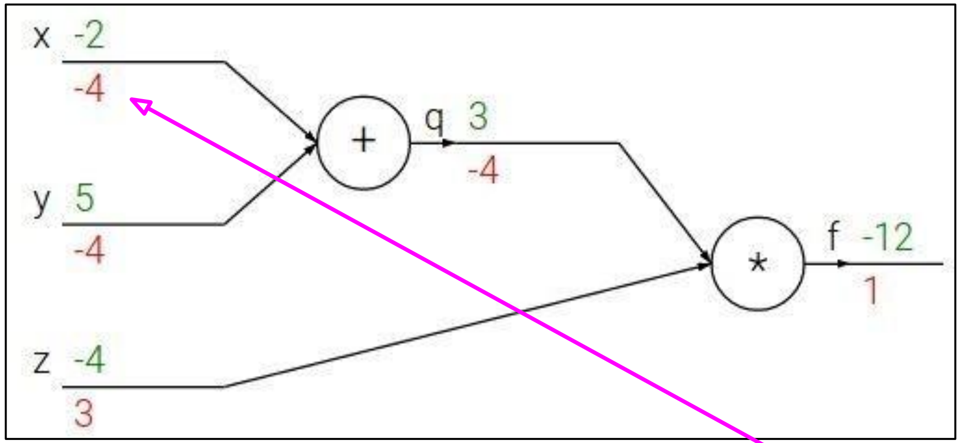
$$f(x,y,z) = (x+y)z$$

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

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

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

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



$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

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

Summary

- Feed-forward network architecture
- Training deep neural nets
    - We need an objective function that measures and guides us towards good performance
    - We need a way to minimize the loss function: gradient descent
    - We need backpropagation to propagate error towards all layers and change weights at those layers
- Next: Practices for preventing overfitting, training with little data, examining conditions for success, alternative optimization strategies