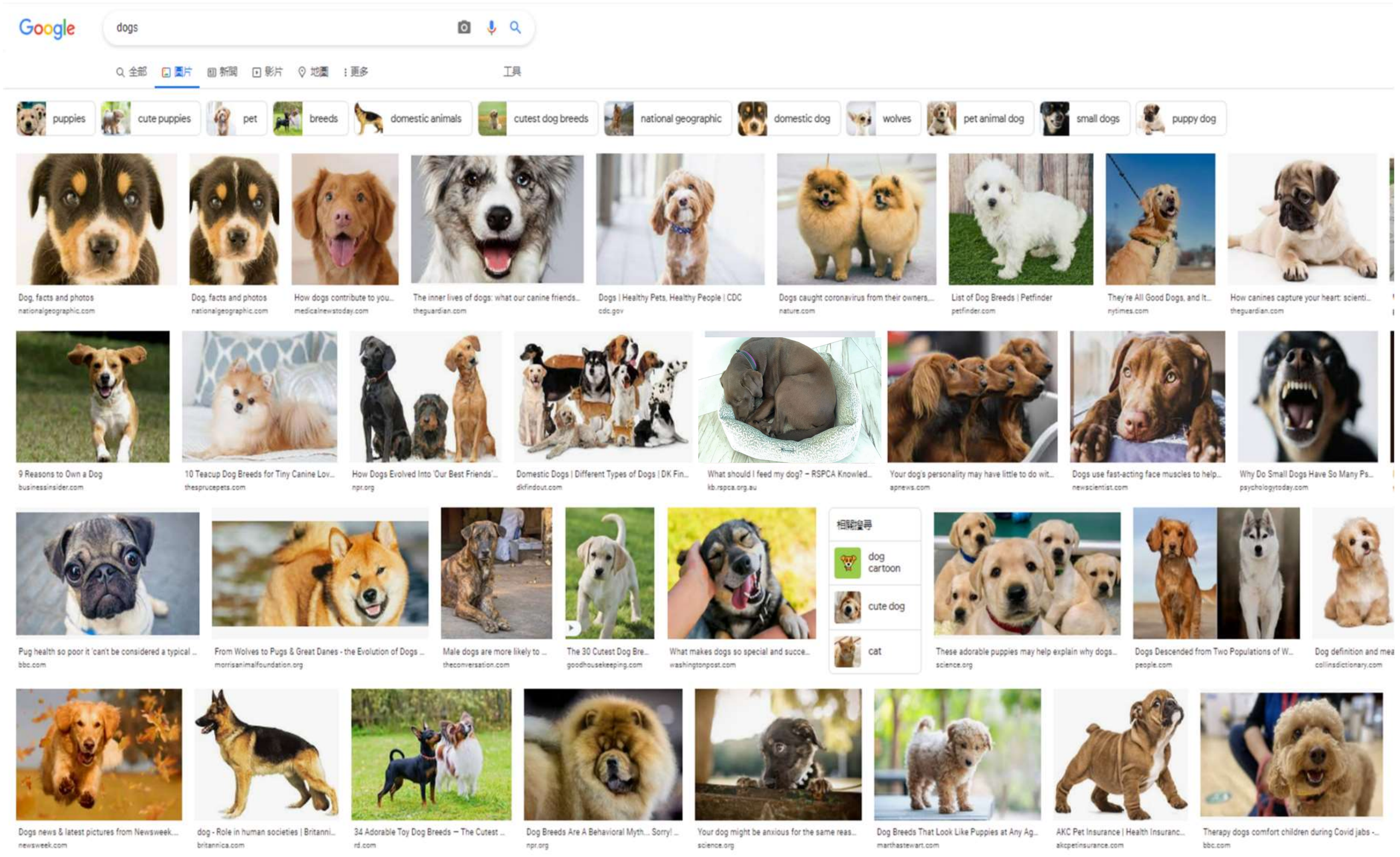
Activation functions and error functions

Last time

- Convolution operation
- Co-occurrence matrix
- Multi-class classification
- Optimizing the Neural Network
- Assignment last time

Image recognition is hard

- Different angles
- Different lighting conditions
- Deformation
- Occlusions
- Background



Linear classification

image → vectors → prediction

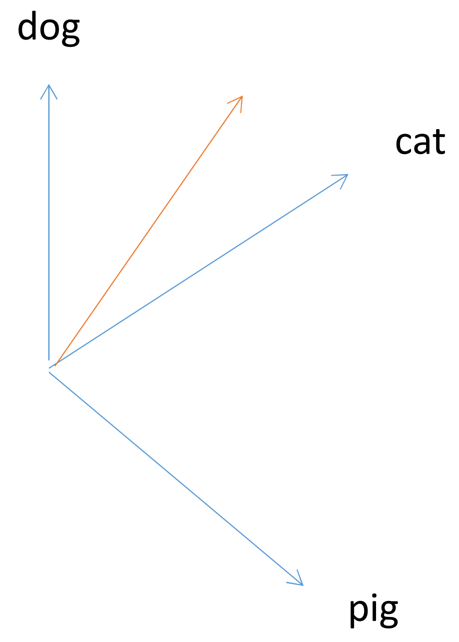
1	2
3	4

1
2
3
4

$f(\mathbf{x},\mathbf{W})=\mathbf{W}\mathbf{x}+\mathbf{b}$

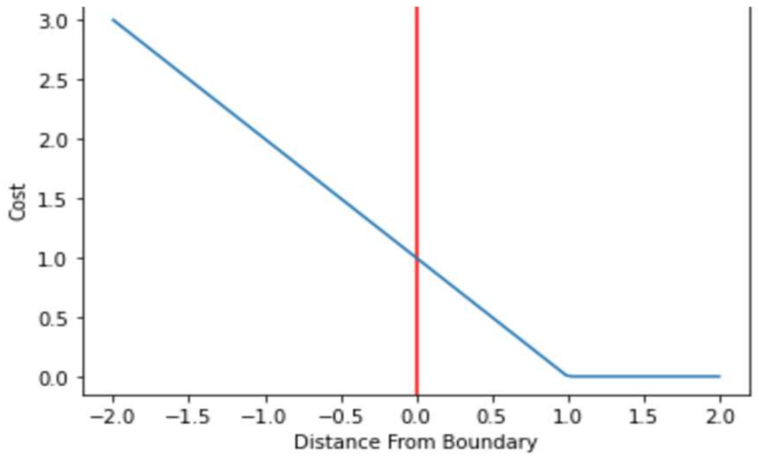
0.2	0.4	0.1	0.6
0.3	0.8	0.2	0.1
0.5	0.6	0.4	0.7

3.6	dog
2.6	cat
-3.0	pig



Loss functions

- Way to quantify the “badness” of a certain (set prediction(s)



maximize the distance.

Hinge loss: $l(y) = max(0, 1 - t \cdot y)$

The hinge loss is a specific type of cost function that incorporates a margin or distance from the classification boundary into the cost calculation. Even if new observations are classified correctly, they can incur a penalty if the margin from the decision boundary is not large enough. The hinge loss increases linearly.

regularization

Total loss: $$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \boxed{\lambda R(W)}$$

Cross entropy loss: $$L_i = -\log P(Y = k|X = x_i) = \boxed{-\log\left(\frac{e^{s_k}}{\sum_j e^{s_j}}\right)}$$ softmax

https://programmathically.com/understanding-hinge-loss-and-the-svm-cost-function/

Optimization

- Optimization

- Follow the slope to go downwards in the loss landscape

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

- Compute gradient information to find the slope

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

Use update function
$W \leftarrow W - \eta \nabla W L$
to update parameters

- Data sets can be large, we don’t want to wait until we’ve seen all the data

Mini batch gradient descent!

Neural networks

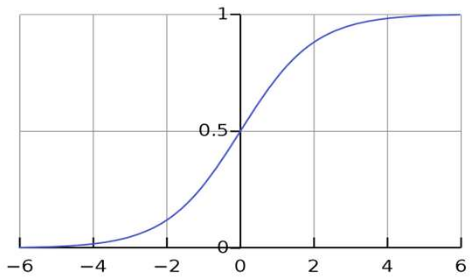
recall linear classification

$$f\,(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$M \times N \times 3$

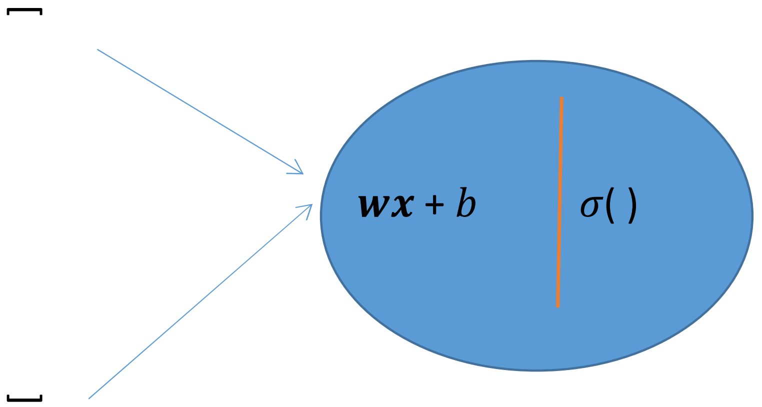
Logistic function

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



Produces a score
$-\infty < z < \infty$

maps the score
to a range 0,1



$3MN \times 1$

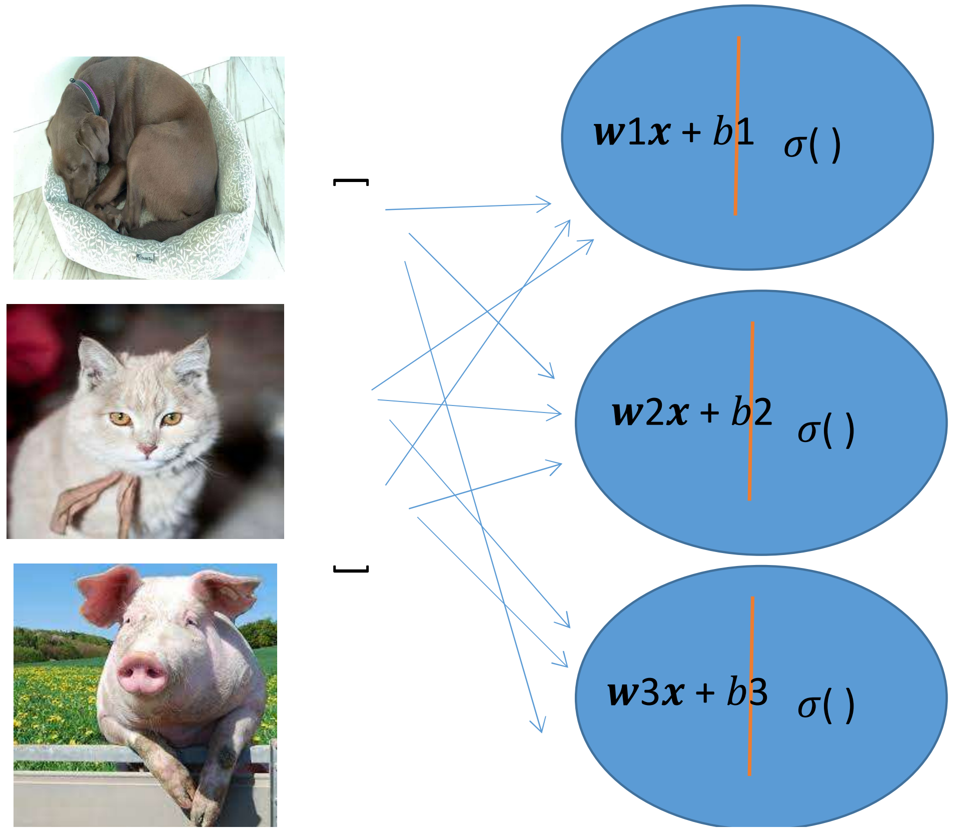
$\hat{y} = \sigma(\boldsymbol{wx} + b)$

Binary Cross-Entropy (BCE) loss

$$\mathcal{L} = y \log\hat{y} + (1 - y) \log\ (1 - \hat{y}\ )$$

Neural networks

- Can we do the same thing for k=3classes?



$\boldsymbol{w}1\boldsymbol{x} + b1$ $\sigma(\ )$
$\boldsymbol{w}2\boldsymbol{x} + b2$ $\sigma(\ )$
$\boldsymbol{w}3\boldsymbol{x} + b3$ $\sigma(\ )$

$\hat{y}_1$=$\sigma(\boldsymbol{w}_1\boldsymbol{x}$+$\boldsymbol{b_1})$

$\hat{y}_2$=$\sigma(\boldsymbol{w}_2\boldsymbol{x}$+$\boldsymbol{b_2})$

$\hat{y}_2$3=$\sigma(\boldsymbol{w}_3\boldsymbol{x}$+$\boldsymbol{b_3}$）

What is the problem with this setup?

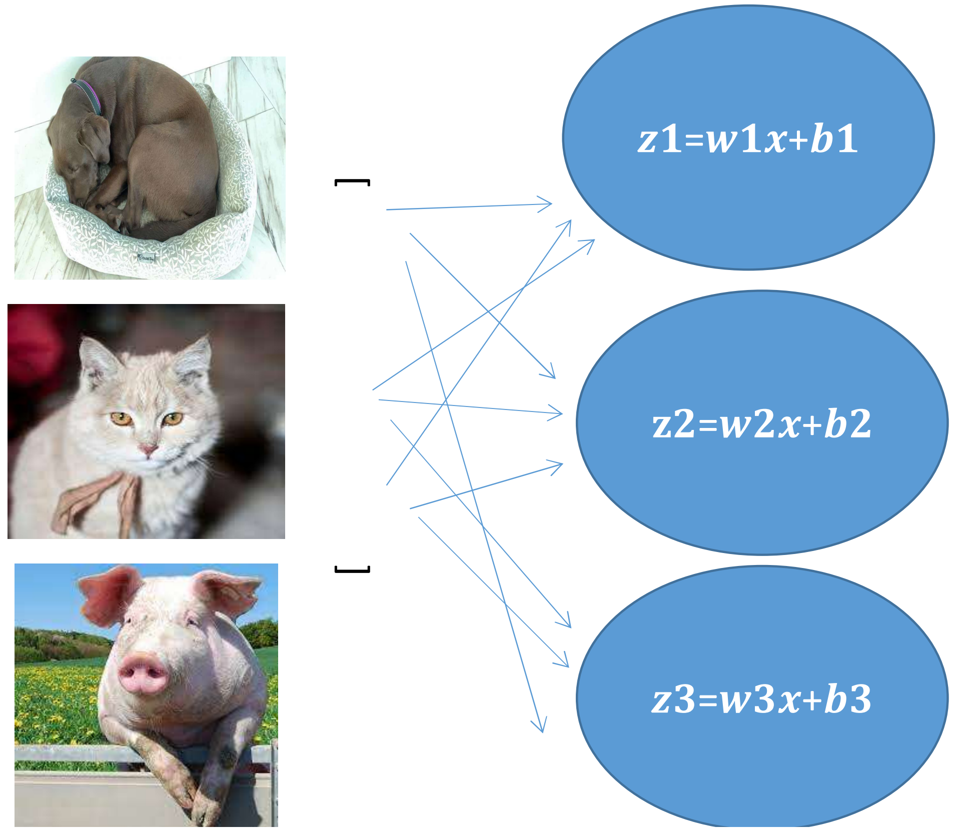
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$

Neural networks

$$\mathcal{L}_{CE} = -\sum_k y_k \log \hat{y}_k$$

Remove individual sigmoid functions

Introduce inter class dependencies



$$\hat{y}_1 = \frac{e^{z_1}}{\sum_k e^{z_k}}$$

$$\hat{y}_2 = \frac{e^{z_2}}{\sum_k e^{z_k}}$$

$$\hat{y}_3 = \frac{e^{z_3}}{\sum_k e^{z_k}}$$

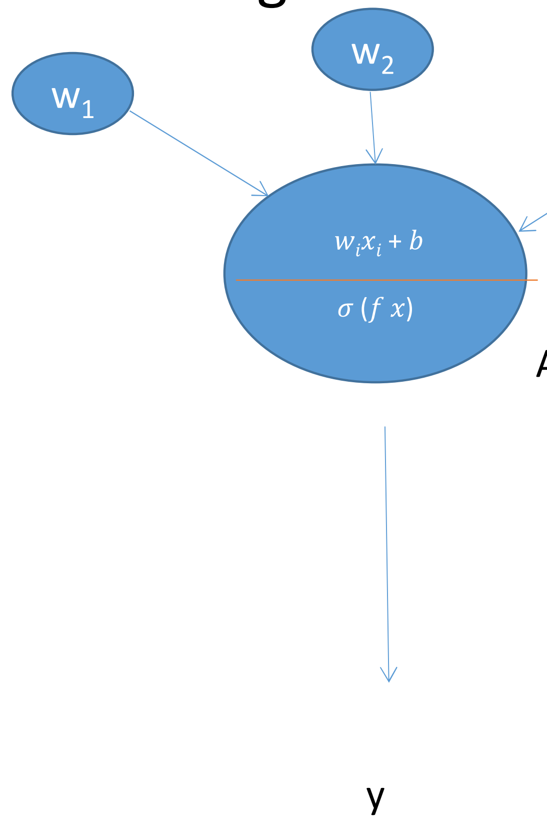
Softmax function

Note: this is a probability distribution over the classes!

Neural networks

- The biological link

A synapse is the connection between nodes, or neurons, in an artificial neural network (ANN). Similar to biological brains, the connection is controlled by the strength or amplitude of a connection between both nodes, also called the synaptic weight.

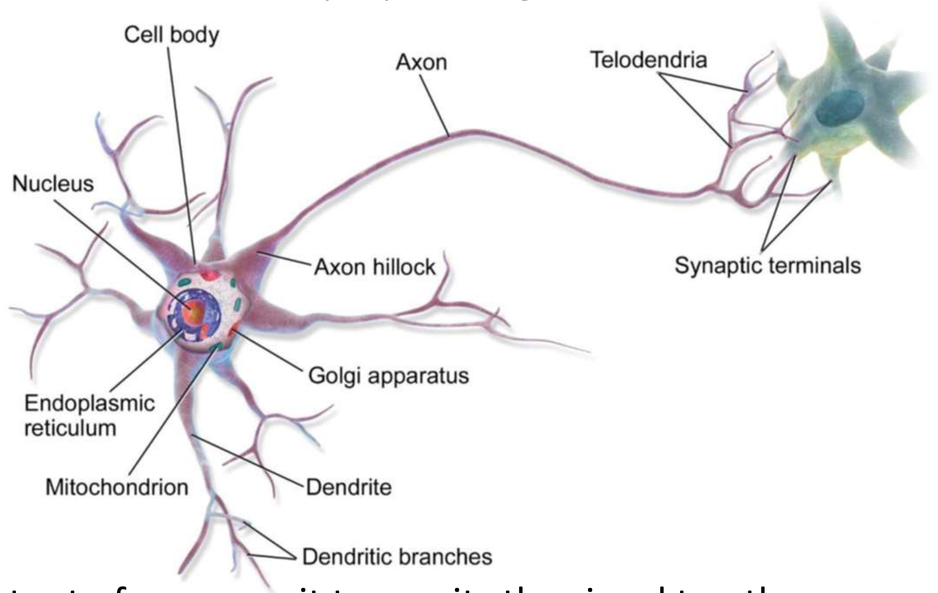


Synapses

Activation function

Dendrites

Axon

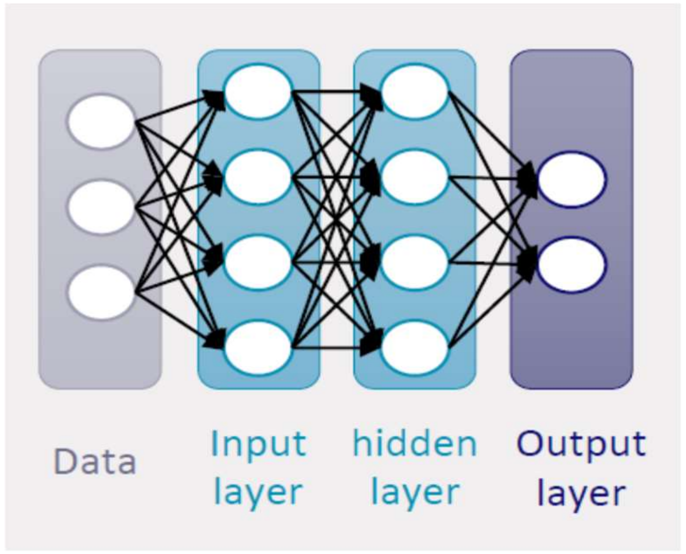


The axon is the output of a neuron it transmits the signal to other neurons. The cell body contains a nucleus and genetic material, which controls the cell's activities

In the nervous system, dendrites, branches of neurons that transmit signals between synapses and soma, play a critical role in processing functions, such as nonlinear integration of postsynaptic signals.

Neural networks

- What have we seen so far?
- We can combine several artificial neurons (perceptrons) to do multiclass prediction
- By stacking multiple layers of these perceptrons, we can capture non linear relations!



Multi
layer perceptron
(Artificial Neural Network)

Hidden layer(s) figure(s) out how to combine the clues from the previous output to learn to make useful predictions in the output

Neural networks

- But how do we find the optimal weights and biases for a particular problem?
- In other words: how can we train the network ?

- Gradient descent!
- From last lecture we know that the update rule is $\theta \leftarrow \theta - \lambda \nabla \theta \mathcal{L}(x, \theta)$

- Let's start by computing the gradient of the loss function $\nabla \theta \mathcal{L}(x, \theta)$

Note that in the case of MLPs $\boldsymbol{\theta}$includes both weights $\boldsymbol{w}$and biases $\boldsymbol{b}$

Backpropagation

Can we find the gradient of the loss with respect to the parameters?

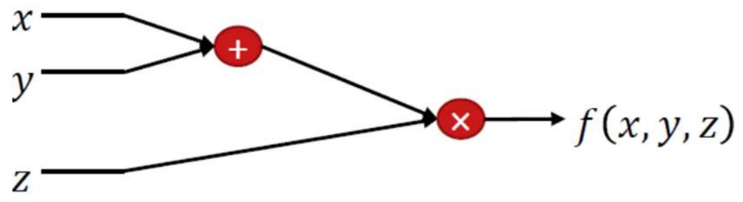
- If we could, then we can use gradient descent to train the weights/biases of the network

So, can we?

- Yes we can! Using backpropagation

Find out how each parameter in a (machine learning) model affects the loss

$f(x,y,z)$ = （$x+y$）z



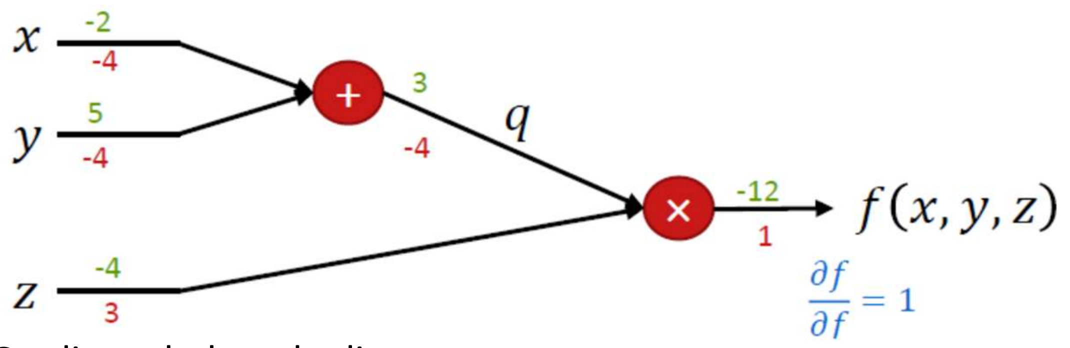
Rumelhart , David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back propagating errors". Nature. 323 (6088): 533 536

Backpropagation

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

We want: $\partial f/\partial x,\ \partial f/\partial y, \partial f/\partial z$

Inputs above the lines



Gradients below the lines

For x,y,z=−2,5,−4

q=x+y $\qquad$ $\partial q/\partial x$=1, $\qquad$ $\partial q/\partial y$=1

f=qz $\qquad$ $\partial f/\partial q$=z $\qquad$ $\partial f/\partial z$=q

chain rule

$\partial f/\partial y$=$\partial f/\partial q\ \partial q/\partial y$

$\partial f/\ \partial q$= −4

$\partial f/\ \partial z$ = 3

$\nabla f(x,y,z)\ |$ −2,5,−4=

$\partial f/\ \partial$y = -4

$\partial\ f/\partial$x = -4

Backpropagation

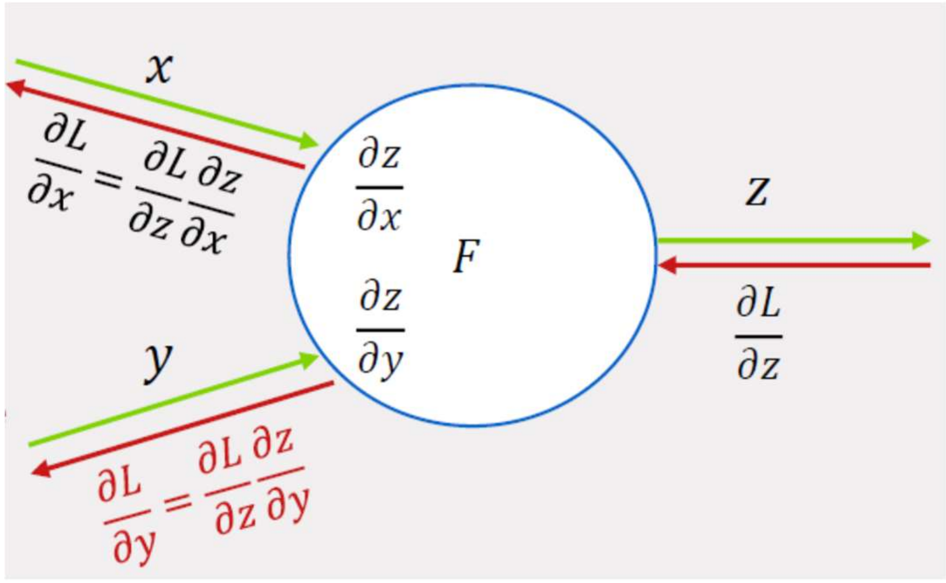
- Forward pass   Compute network prediction and loss
- Backward pass   Backpropage the error (loss) to find the influence of all the weights

Local gradient information

- Each node only considers the inputs and output
- Two things a gate can compute on forward pass

  (1) Output z, (2) gradient w.r.t. all the inputs

- Receive influence on final loss during backward pass
- Compute effect of inputs on final loss by means of the chain rule
- Gates communicate the influence on the final loss



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
F
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

Backpropagation

Another complex example

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

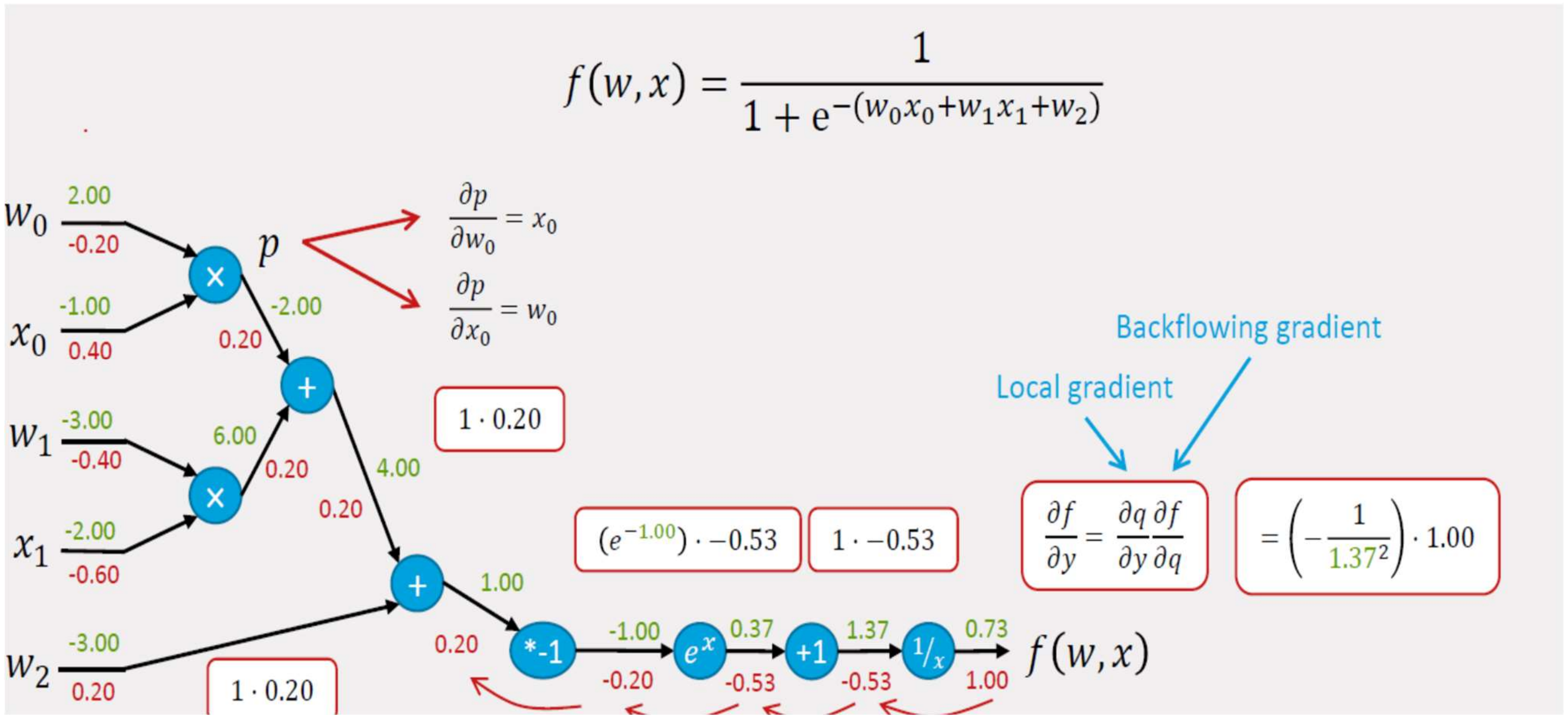
$$\frac{d\sigma}{dx} = \frac{e^{-x}}{(1+e^{x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

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

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

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

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



$f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$
w_0
2.00
-0.20
x_0
-1.00
0.40
w_1
-3.00
-0.40
x_1
-2.00
-0.60
w_2
-3.00
0.20
p
$\frac{\partial p}{\partial w_0} = x_0$
$\frac{\partial p}{\partial x_0} = w_0$
-2.00
0.20
6.00
0.20
4.00
0.20
$1 \cdot 0.20$
1.00
0.20
$1 \cdot 0.20$
*-1
e^x
+1
$1/x$
-1.00
-0.20
0.37
-0.53
1.37
-0.53
0.73
1.00
$f(w,x)$
$(e^{-1.00}) \cdot -0.53$
$1 \cdot -0.53$
Backflowing gradient
Local gradient
$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$
$= \left(-\frac{1}{1.37^2}\right) \cdot 1.00$

Backpropagation

The backpropagation of errors

- Simple and efficient procedure to evaluate the local gradient of the loss function
- Basically : find out how each parameter in a (machine learning) model affects the loss

Basic idea:

On any computational graph, do until convergence:

1. A forward pass to compute the output (prediction) and compute the loss

2. A backward pass to backpropagate the error/loss through the graph

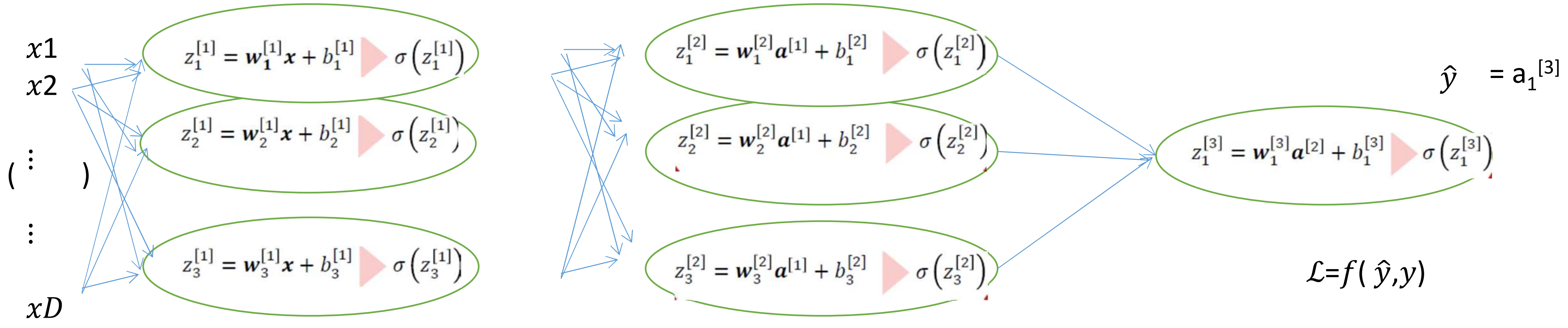
to find the influence of all the parameters θon the loss Gradient $\delta\mathcal{L}/\delta\theta$at point θi

3. Given the gradient $\delta\mathcal{L}/\delta\theta$, update the weights using e.g. SGD and go to step 1

Backpropagation

- Lets try this on a neural network!
- we want to know: $\frac{d\mathcal{L}}{d\boldsymbol{w}}$

$\boldsymbol{w}_m^{[n]}$	Weigths neuron m in layer n
$b_m^{[n]}$	bias neuron m in layer n
$z_m^{[n]}$	Linear output of neuron m in layer n
$a_m^{[n]}$	Activation of neuron m in layer n



$x1$
$x2$
xD
$z_1^{[1]} = \boldsymbol{w}_1^{[1]}\boldsymbol{x} + b_1^{[1]}$ $\sigma(z_1^{[1]})$
$z_2^{[1]} = \boldsymbol{w}_2^{[1]}\boldsymbol{x} + b_2^{[1]}$ $\sigma(z_2^{[1]})$
$z_3^{[1]} = \boldsymbol{w}_3^{[1]}\boldsymbol{x} + b_3^{[1]}$ $\sigma(z_3^{[1]})$
$z_1^{[2]} = \boldsymbol{w}_1^{[2]}\boldsymbol{a}^{[1]} + b_1^{[2]}$ $\sigma(z_1^{[2]})$
$z_2^{[2]} = \boldsymbol{w}_2^{[2]}\boldsymbol{a}^{[1]} + b_2^{[2]}$ $\sigma(z_2^{[2]})$
$z_3^{[2]} = \boldsymbol{w}_3^{[2]}\boldsymbol{a}^{[1]} + b_3^{[2]}$ $\sigma(z_3^{[2]})$
$z_1^{[3]} = \boldsymbol{w}_1^{[3]}\boldsymbol{a}^{[2]} + b_1^{[3]}$ $\sigma(z_1^{[3]})$
$\hat{y} = a_1^{[3]}$

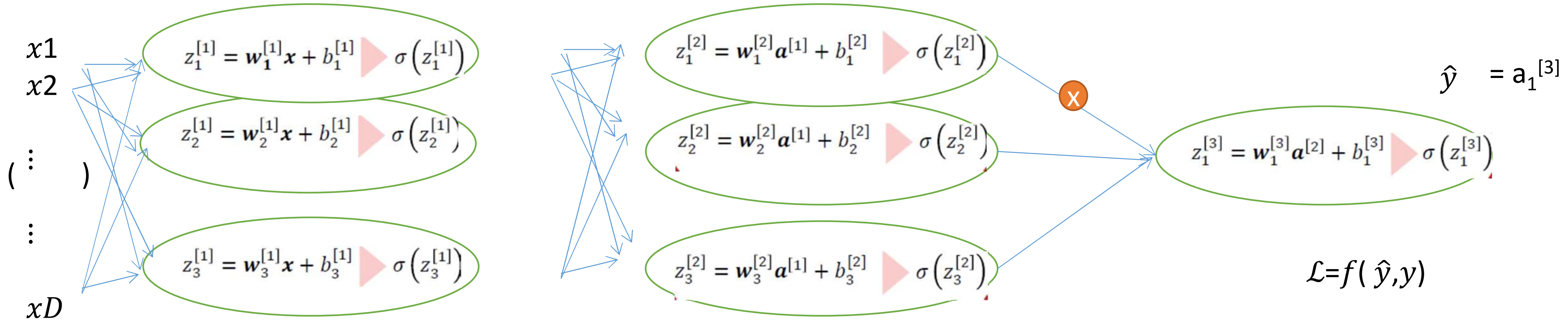
$\mathcal{L}=f(\hat{y},y)$

Number of parameters?

Backpropagation

$$\frac{d\mathcal{L}}{dw_1^{[3]}}$$

$$\frac{d\mathcal{L}}{dw_1^{[3]}} = \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}}{\partial w_1^{[3]}} = \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}}{\partial z_1^{[3]}}\frac{\partial z_1^{[3]}}{\partial w_1^{[3]}}$$



$x1$
$x2$
$\vdots$
xD
$z_1^{[1]} = w_1^{[1]}x + b_1^{[1]}$ $\sigma(z_1^{[1]})$
$z_2^{[1]} = w_2^{[1]}x + b_2^{[1]}$ $\sigma(z_2^{[1]})$
$z_3^{[1]} = w_3^{[1]}x + b_3^{[1]}$ $\sigma(z_3^{[1]})$
$z_1^{[2]} = w_1^{[2]}a^{[1]} + b_1^{[2]}$ $\sigma(z_1^{[2]})$
$z_2^{[2]} = w_2^{[2]}a^{[1]} + b_2^{[2]}$ $\sigma(z_2^{[2]})$
$z_3^{[2]} = w_3^{[2]}a^{[1]} + b_3^{[2]}$ $\sigma(z_3^{[2]})$
x
$z_1^{[3]} = w_1^{[3]}a^{[2]} + b_1^{[3]}$ $\sigma(z_1^{[3]})$
$\hat{y} = a_1^{[3]}$
$\mathcal{L}=f(\hat{y},y)$

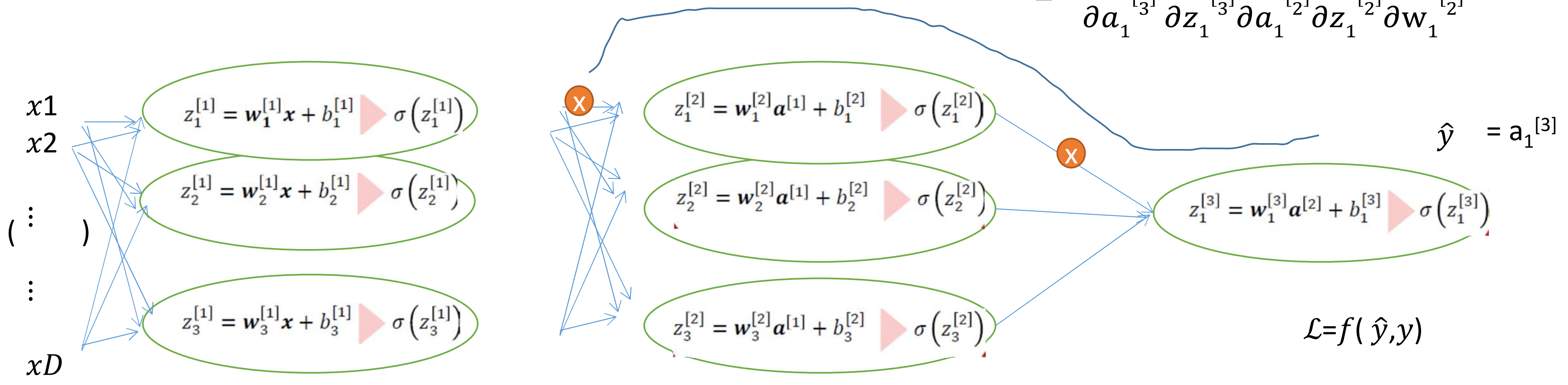
Backpropagation

$$\frac{d\mathcal{L}}{dw_1^{[2]}} = \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}}{\partial w_1^{[2]}} = \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}\partial z_1^{[3]}}{\partial z_1^{[3]}\partial w_1^{[2]}}$$

try $\frac{d\mathcal{L}}{dw_1^{[2]}}$

$$= \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}\partial z_1^{[3]}\partial a_1^{[2]}}{\partial z_1^{[3]}\partial a_1^{[2]}\partial w_1^{[2]}}$$

$$= \frac{\partial\mathcal{L}}{\partial a_1^{[3]}}\frac{\partial a_1^{[3]}\partial z_1^{[3]}\partial a_1^{[2]}\partial z_1^{[2]}}{\partial z_1^{[3]}\partial a_1^{[2]}\partial z_1^{[2]}\partial w_1^{[2]}}$$



$x1$
$x2$
xD
$z_1^{[1]} = w_1^{[1]}x + b_1^{[1]}$ $\sigma(z_1^{[1]})$
$z_2^{[1]} = w_2^{[1]}x + b_2^{[1]}$ $\sigma(z_2^{[1]})$
$z_3^{[1]} = w_3^{[1]}x + b_3^{[1]}$ $\sigma(z_3^{[1]})$
$z_1^{[2]} = w_1^{[2]}a^{[1]} + b_1^{[2]}$ $\sigma(z_1^{[2]})$
$z_2^{[2]} = w_2^{[2]}a^{[1]} + b_2^{[2]}$ $\sigma(z_2^{[2]})$
$z_3^{[2]} = w_3^{[2]}a^{[1]} + b_3^{[2]}$ $\sigma(z_3^{[2]})$
$z_1^{[3]} = w_1^{[3]}a^{[2]} + b_1^{[3]}$ $\sigma(z_1^{[3]})$
$\hat{y} = a_1^{[3]}$
$\mathcal{L}=f(\hat{y},y)$

We could repeat this for every weight…

Backpropagation

- Basically, neurons only have to communicate their impact on the loss to their neighbors;
- Neurons can update by applying the chain rule

$$\frac{d\mathcal{L}}{dw} = \frac{\partial \mathcal{L}}{\partial f}\frac{\partial f}{\partial w}$$

Backflowing gradient　　Local gradient

Backpropagation

- Matrix notation

$w^{[1]} = [\ w_1^{[1]T}\ \ w_2^{[1]\,T}\ \ w_3^{[1]\,T}]^T$

$b^{[1]} = [\ b_1^{[1]}\ \ b_2^{[1]}\ \ b_3^{[1]}\ ]^T$

3 x1 3XD Dx1 3X1

$z^{[1]} = w^{[1]} x + b^{[1]}$

$a^{[1]} = \sigma(\ z^{[1]})$

3 x1 3X3 3x1 3X1

$z^{[2]} = w^{[2]}]\ a^{[1]} + b^{[2]}$

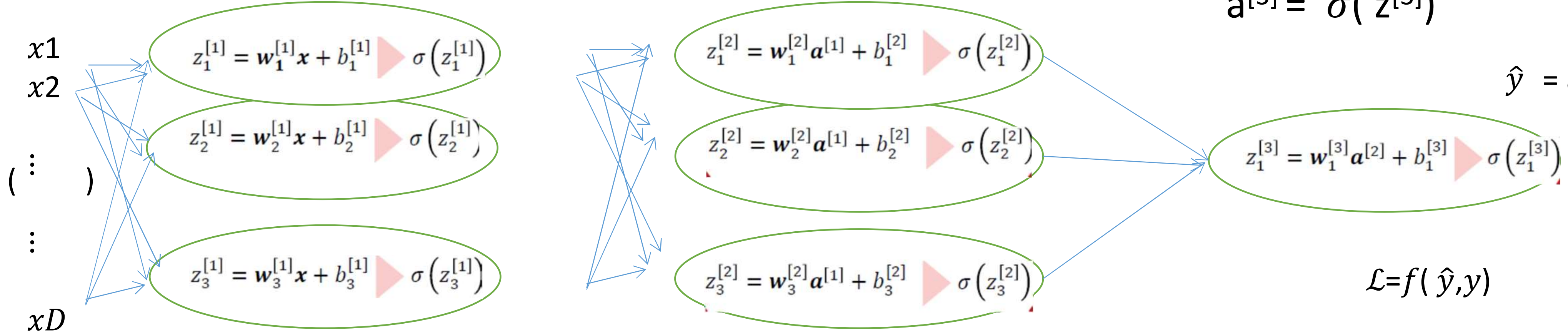
$a^{[2]} = \sigma(\ z^{[2]})$

1 1X3 3x1

$z^{[3]} = w^{[3]}]\ a^{[2]} + b^{[3]}$

$a^{[3]} = \sigma(\ z^{[3]})$

$\hat{y} = a_1^{[3]}$

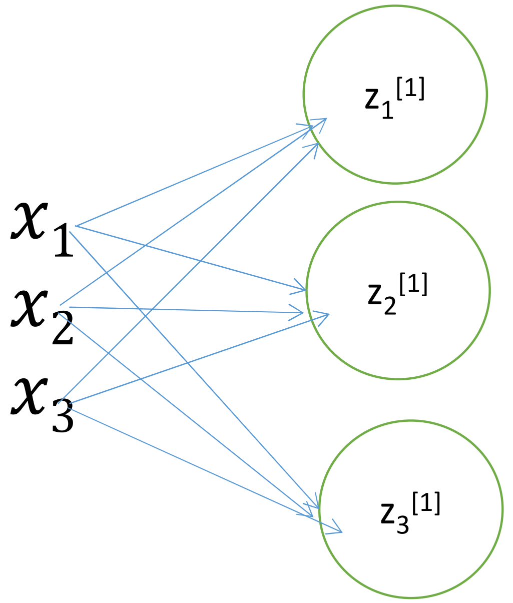


$x1$
$x2$
$\vdots$
$\vdots$
xD
$z_1^{[1]} = \boldsymbol{w}_1^{[1]}\boldsymbol{x} + b_1^{[1]}$ $\sigma\left(z_1^{[1]}\right)$
$z_2^{[1]} = \boldsymbol{w}_2^{[1]}\boldsymbol{x} + b_2^{[1]}$ $\sigma\left(z_2^{[1]}\right)$
$z_3^{[1]} = \boldsymbol{w}_3^{[1]}\boldsymbol{x} + b_3^{[1]}$ $\sigma\left(z_3^{[1]}\right)$
$z_1^{[2]} = \boldsymbol{w}_1^{[2]}\boldsymbol{a}^{[1]} + b_1^{[2]}$ $\sigma\left(z_1^{[2]}\right)$
$z_2^{[2]} = \boldsymbol{w}_2^{[2]}\boldsymbol{a}^{[1]} + b_2^{[2]}$ $\sigma\left(z_2^{[2]}\right)$
$z_3^{[2]} = \boldsymbol{w}_3^{[2]}\boldsymbol{a}^{[1]} + b_3^{[2]}$ $\sigma\left(z_3^{[2]}\right)$
$z_1^{[3]} = \boldsymbol{w}_1^{[3]}\boldsymbol{a}^{[2]} + b_1^{[3]}$ $\sigma\left(z_1^{[3]}\right)$

$\mathcal{L} = f(\ \hat{y}, y)$

Backpropagation

Matrix notation



$$z^{[1]} = \begin{bmatrix} w_1^{[1]} \\ w_2^{[1]} \\ w_3^{[1]} \end{bmatrix} x + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

$$z^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} \end{bmatrix} x + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

Backpropagation

- Let’s find the update for the third layer

$$\boxed{z^{[3]} = w^{[3]} a^{[2]} + b^{[3]} \Rightarrow \sigma(z^{[3]})} \qquad \hat{y} = a_1^{[3]}$$

For binary classes: Binary Cross Entropy (BCE)

$$\mathcal{L}^{(i)} = -\left(y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right)$$

Update rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$$

Need to find the gradient!

$$\frac{d\mathcal{L}}{dw^{[3]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]} z^{[3]}}{\partial z^{[3]} \partial w^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\partial a^{[3]}} = \frac{\partial}{\partial a^{[3]}} \left(- y^{(i)} \log a^{[3]} - (1-y^{(i)}) \log(1-a^{[3]}\right)$$

$$= -y \frac{1}{a^{[3]}} - (1-y) \frac{1}{1-a^{[3]}}$$

$$\frac{\partial a^{[3]}}{\partial z^{[3]}} = \frac{\partial}{\partial z^{[3]}} \sigma(z^{[3]}) = \sigma(z^{[3]}) (1 - \sigma(z^{[3]}))$$ see slide 15

$$\frac{\partial z^{[3]}}{\partial w^{[3]}} = \frac{\partial}{\partial w^{[3]}} w^{[3]} a^{[2]} + b^{[3]}$$

$$= a^{[2]T}$$

the output of derivative should be the same as w, w is of the size 3x1, so the derivative result should also be 3x1, which is the Transpose of a

Backpropagation

- Put everything together

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{[3]}} = (a_1{}^{[3]} - y)\boldsymbol{a}^{[2]}T$$

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \left(-y\frac{1}{a_1^{[3]}} + (1-y)\frac{1}{1-a_1^{[3]}}\right)\left(a_1^{[3]}\left(1-a_1^{[3]}\right)\right)\boldsymbol{a}^{[2]^T}$$

$$= \left(-y\frac{a_1^{[3]}\left(1-a_1^{[3]}\right)}{a_1^{[3]}} + (1-y)\frac{a_1^{[3]}\left(1-a_1^{[3]}\right)}{1-a_1^{[3]}}\right)\boldsymbol{a}^{[2]^T}$$

$$= \left(-y\left(1-a_1^{[3]}\right) + (1-y)a_1^{[3]}\right)\boldsymbol{a}^{[2]^T}$$

$$= \left(-y + \cancel{ya_1^{[3]}} + a_1^{[3]} \cancel{-ya_1^{[3]}}\right)\boldsymbol{a}^{[2]^T}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{[3]}} = \left(a_1^{[3]} - y\right)\boldsymbol{a}^{[2]^T}$$

Effect on the loss of the three weights of the single neuron in the last layer

Backpropagation

Why do we want to find the gradient of the loss w.r.t. to the network parameters?

- Neural networks will be large
- Impractical to write down an explicit formulation of the gradient by hand

- Backpropagation recursively applies the chain rule to find the influence of all the parameters and inputs on the final loss

- Forward pass:
  - Compute the outputs of all the gates to finally compute the network output and the resulting loss.
  - Also compute the local gradients and store them for use during backward pass.

- Backward pass:
  - Get the upstream gradient (from a specific gate to the output)
  - Multiply the upstream gradient with the local gradient, and pass it on to all parent nodes

Neural Network

- Linear score function: $f = Wx$

- 2-layer Neural Network： $f = W_2 \max(0, W_1 x)$

- 3-layer Neural Network $f = W_3 \max(0, W_2 \max(0, W_1 x))$

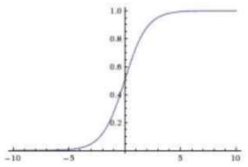
Implementing a two-layer neural network from scratch

- https://ljvmiranda921.github.io/notebook/2017/02/17/artificial-neural-networks/

Activation Functions

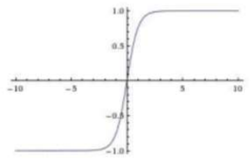
Sigmoid

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



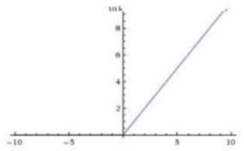
tanh

tanh(x)



ReLU

max(0,x)

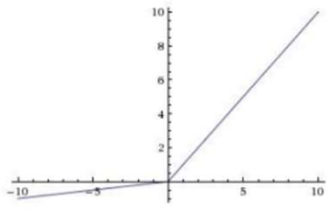


Maxout

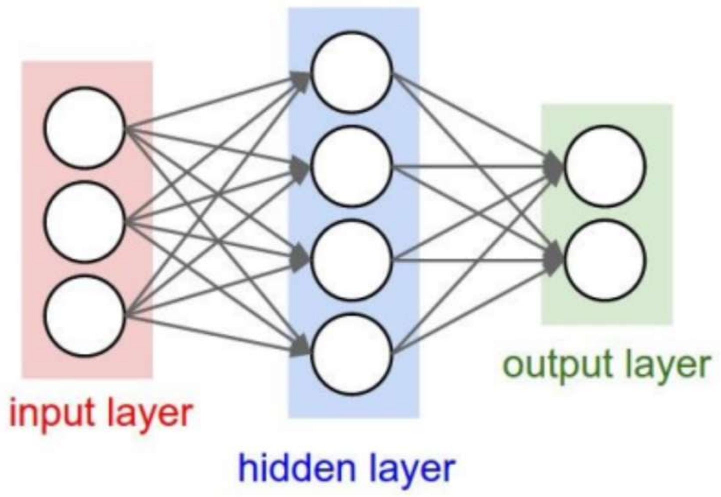
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

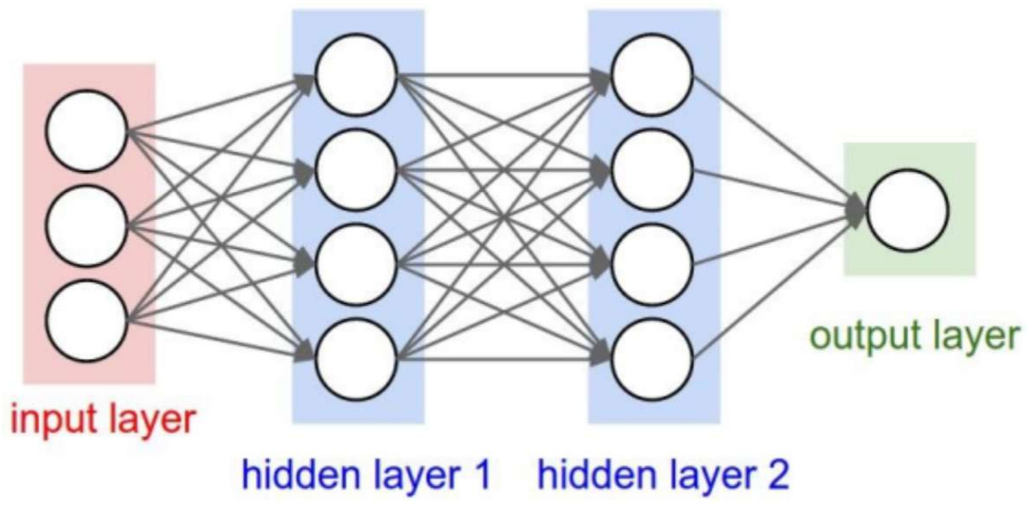
Leaky ReLU

max(0.1x, x)

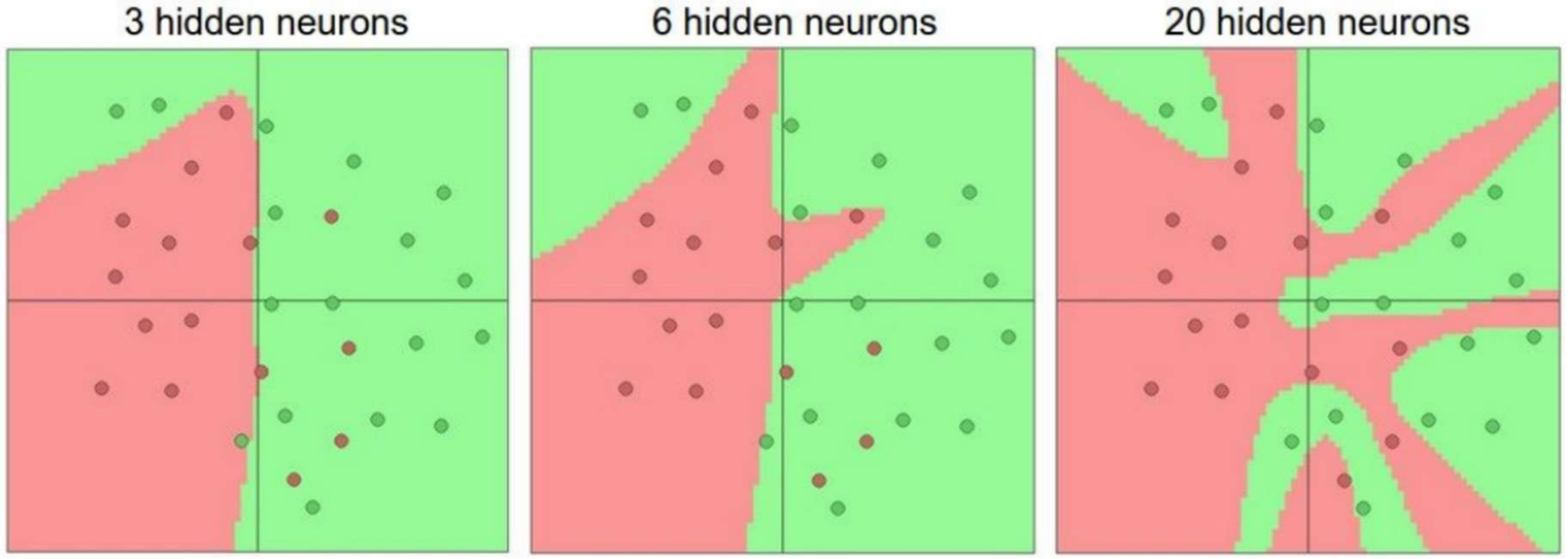


Neural Networks: Architectures





Setting the number of layers and their sizes



play the demo: https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Fei-Fei Li & Andrej Karpathy & Justin Johnson

Exercise

- https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/

- https://www.w3schools.com/ai/ai_perceptrons.asp

Summary

Neural networks

- Layer wise arrangement of artificial neurons (linear classifiers followed by a non linearity)
- Biological comparison sounds cool but has only limited validity
- Backpropagation efficiently finds the local gradient of the loss function
- Use computational graphs and the chain rule
- The local gradient of the loss function can be used to update network parameters
- Use gradient descent to find an optimal set of parameters for a given training set

Next time

- Organize a hierarchical filtering structure as a neural network
- We can learn our filter kernels using backprop

What can you do with CNNs?

What are they made of?