MIDDLE EAST TECHNICAL UNIVERSITY

DI504 Foundations of Deep Learning

Loss Functions and Optimization

Welcome again!

This is DI504, Foundations of Deep Learning

- We previously talked about feature spaces and score functions, which are basically the fundamentals concept of machine learning.
- Now, we will talk about ways to create a desired score function.
- Then we will talk about a concept of loss function. Loss function will help us find the weights (which is basically “training the model”)

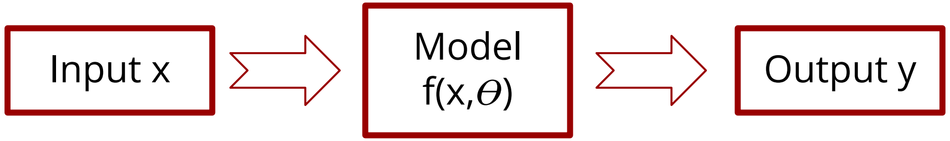
Score Function

Let’s remember what the score function was

- $f(x,\Theta)$ is basically a function of input x and model parameters Θ, that output a score value y.

$$f(x,\Theta) = y$$

-



METU

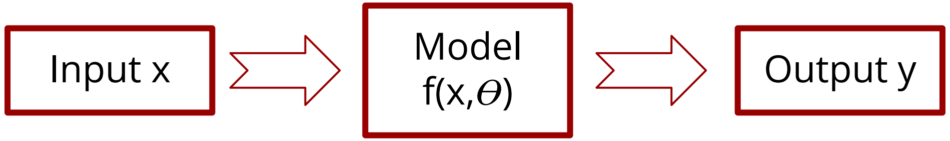
Score Function

Let's remember what the score function was

- $f(x,\Theta)$ is basically a function of input x and model parameters Θ, that output a score value y.

$$f(x,\Theta) = y$$

- So, how to find the correct Θ?



Supervised Learning

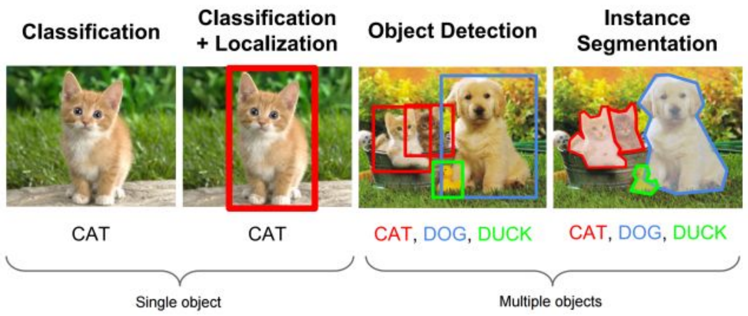
- Supervised learning (SL) is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.
- It infers a function from labeled training data consisting of a set of training examples.
- In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the ground truth).

Ground Truth

- A ground-truth dataset is a regular dataset, but with *annotations* added to it.
- *Annotations* can be boxes drawn over images, written text indicating samples, a new column of a spreadsheet or anything else the machine learning algorithm should learn to output.
-

Ground Truth

- A ground-truth dataset is a regular dataset, but with *annotations* added to it.
- *Annotations* can be boxes drawn over images, written text indicating samples, a new column of a spreadsheet or anything else the machine learning algorithm should learn to output.
- Depending on problem type, the character of annotations may differ:



Ground Truth

(how to find one)

- There are many freely available tools for annotating a dataset to make it a ground-truth dataset such as *Universal Data Tool*, *Label Studio* and *Labelimg*.
- You can see screenshots and interactive search of different machine learning tools using Compare Data Tools.
- Companies will also annotate your data for you.
- The biggest challenge for annotating data externally is tracking the quality of your labels. We recommend taking a sample slice of your data from the annotation company and measure the percentage of samples that are correctly annotated.

Loss Function

- Loss functions define if the score function $f(x,\Theta)$ is working fine.

  Let’s define a loss function: for a given input - ground truth pair (x_i,g_i), the loss is a function of the score value of that input and the ground truth.

  $L(x_i , g_i) = g [f(x_i,\Theta) , g_i]$

METU

Loss Function

- Loss functions define if the score function $f(x,\Theta)$ is working fine.

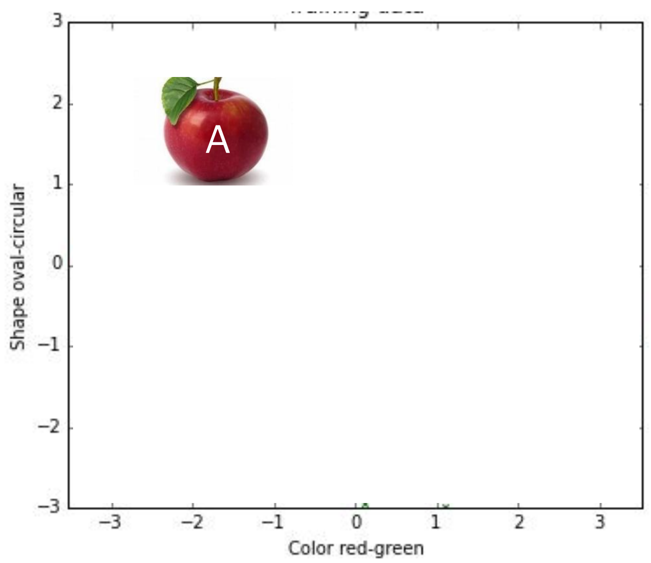
  Let’s define a loss function: for a given input - ground truth pair (x_i,g_i), the loss is a function of the score value of that input and the ground truth.

  $L(x_i, g_i) = g[f(x_i,\Theta), g_i]$

  For example: 2D input (colour and shape) for a model that predicts how juicy an apple is

  $f(x,\Theta) = 2\cdot x_2 - 3\cdot x_1$

  So Θ is simply [-3, +2] and $f(x,\Theta)$ is $f(x,\Theta) = x^T \cdot \Theta$



Loss Function

- Loss functions define if the score function $f(x,\Theta)$ is working fine.

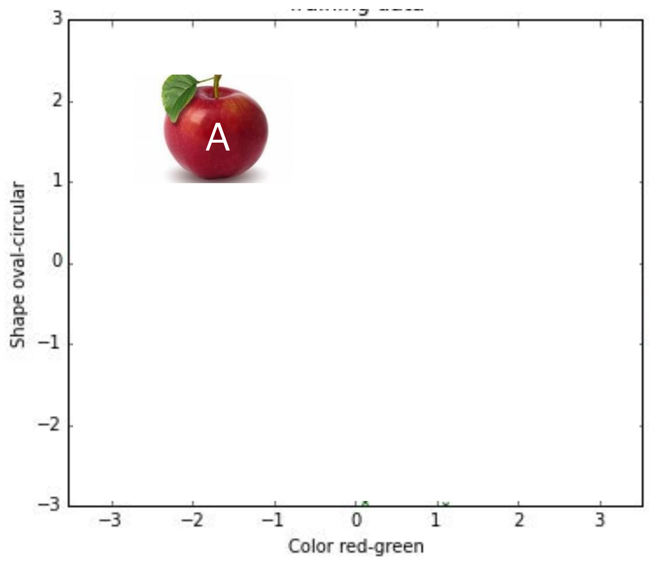
  Let's define a loss function: for a given input - ground truth pair (x_i,g_i), the loss is a function of the score value of that input and the ground truth.

  $L(x_i , g_i) = g [f(x_i,\Theta) , g_i]$

  For example: 2D input (colour and shape) for a model that predicts how juicy an apple is

  $f(x,\Theta) = 2 \cdot x_2 - 3 \cdot x_1$

  Say $x = [-1.5 +1.5] \rightarrow y = +7.5$
  (score function says it is "7.5 much" juicy)



Loss Function

- Loss functions define if the score function $f(x,\Theta)$ is working fine.

Let's define a loss function: for a given input - ground truth pair (x_i,g_i), the loss is a function of the score value of that input and the ground truth.

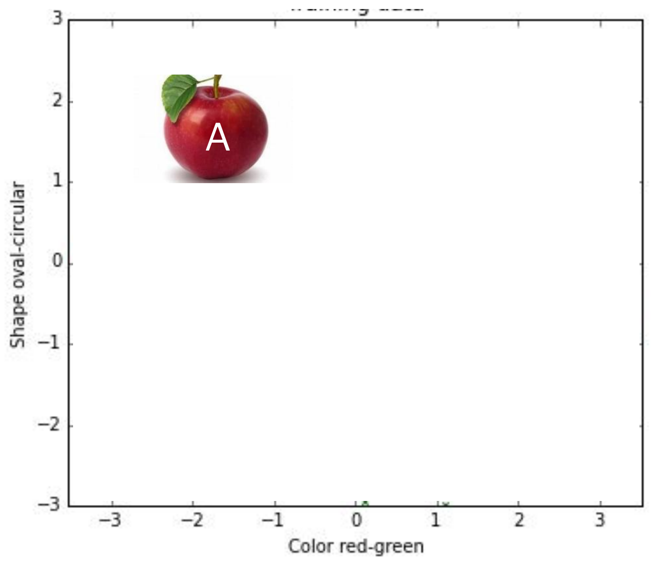
$L(x_i, g_i) = g[f(x_i,\Theta), g_i]$

For example: 2D input (colour and shape) for a model that predicts how juicy an apple is

$f(x,\Theta) = 2 \cdot x_2 - 3 \cdot x_1$

Say $x = [-1.5 \ +1.5] \rightarrow y = +7.5$
(but the ground truth says it is 5.0 juicy.)

Gourmets from all around the world decided on this value (5.0) for this apple.



Loss Function

- So for this apple, how good is this score function

We define a loss function for

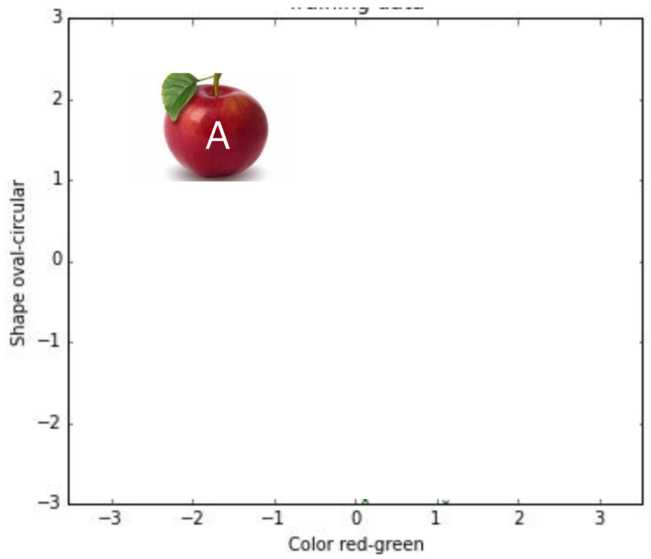
$L(x_i, g_i) = g[f(x_i,\Theta), g_i]$
$L(x_i, g_i, \Theta) = |f(x_i,\Theta) - g_i| = +2.5.$

For example: 2D input (colour and shape) for a model that predicts how juicy an apple is

$f(x,\Theta) = 2 \cdot x_2 - 3 \cdot x_1$

Say $x = [-1.5\ +1.5] \rightarrow y = +7.5$
(but the ground truth says it is 5.0 juicy.)

Gourmets from all around the world decided on this value (5.0) for this apple.



METU

Loss Function

- So for this apple, how good is this score function

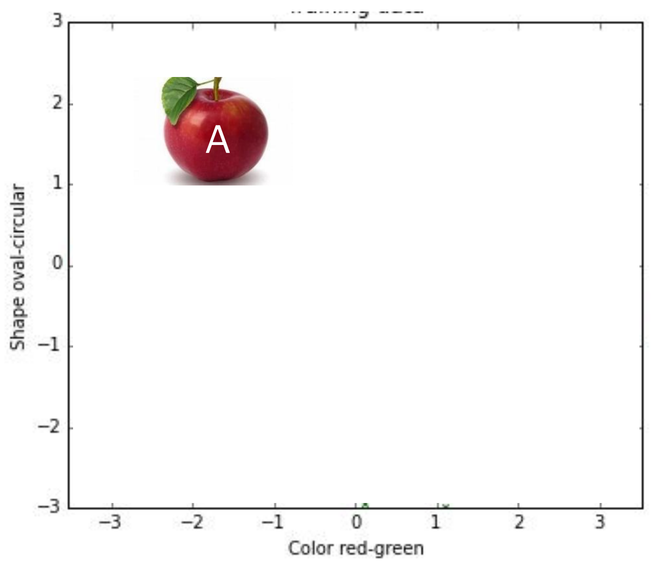
  We define a loss function for

  $L(x_i, g_i) = g[f(x_i,\Theta), g_i]$
  $L(x_i, g_i, \Theta) = |f(x_i,\Theta) - g_i| = +2.5.$

  So the score function is "+2.5 much" bad.

- If this score function was perfect, it would have output a 5.0 value.
- What can we change to obtain a 5.0 value?

  $f(x,\Theta) = 2 \cdot x_2 - 3 \cdot x_1$



METU

Loss Function

- So for this apple, how good is this score function

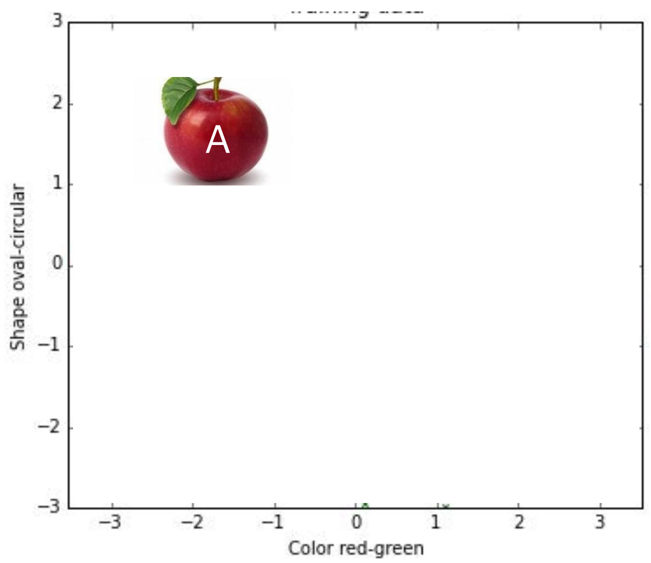
  We define a loss function for

  $L(x_i, g_i) = g[f(x_i, \Theta), g_i]$
  $L(x_i, g_i, \Theta) = |f(x_i, \Theta) - g_i| = +2.5.$

  So the score function is “+2.5 much” bad.

- If this score function was perfect, it would have output a 5.0 value.
- What can we change to obtain a 5.0 value?

  $f(x, \boldsymbol{\theta}) = 2 \cdot x_2 - 3 \cdot x_1$



Loss Function

- So for this apple, how good is this score function

  We define a loss function for

  $L(x_i, g_i) = g[f(x_i,\theta), g_i]$
  $L(x_i, g_i, \theta) = |f(x_i,\theta) - g_i| = +2.5.$

  So the score function is “+2.5 much” bad.

- If this score function was perfect, it would have output a 5.0 value.
- What can we change to obtain a 5.0 value?

  $f(x,\theta) = 2 \cdot x_2 - 3 \cdot x_1$

Loss Function

- Loss function quantitatively tell us how good/bad our score function is.
- Usually (at start) score functions are bad. Because in ML, they are "initialized" randomly.
- By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters.
    - We don't do it. We write an algorithm to do it for us: which is called "optimization".

Score/Loss Function for classification

- Sometimes the predicted value is not a real number, but category out of N classes. How to decide the penalty then.
- Then we use a score and loss function suitable for classification. e.g.
    - Score funcion: soft-max
    - Loss function: cross-entropy

Soft-max

- Soft-Max is a score function normalizer for categorical scores.
- It converts real value scores to probabilities (that sum upto 1).

Want to interpret raw classifier scores as **probabilities**

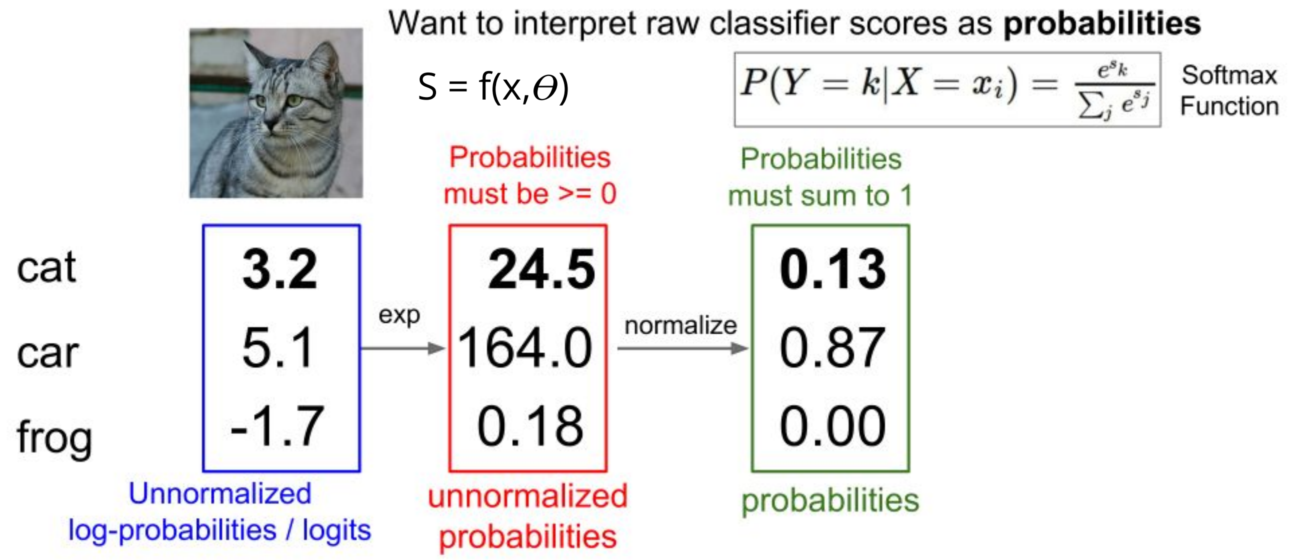
$S = f(x,\Theta)$

cat	**3.2**
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

Soft-max

- Soft-Max is a score function normalizer for categorical scores.
- It converts real value scores to probabilities (that sum upto 1).

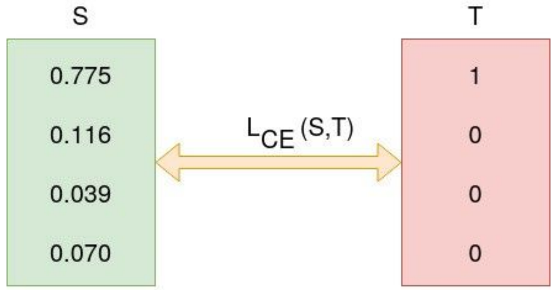


Want to interpret raw classifier scores as **probabilities**
S = f(x,Θ)
$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$
Softmax Function
Probabilities must be >= 0
Probabilities must sum to 1
cat
car
frog
3.2
5.1
-1.7
exp
24.5
164.0
0.18
normalize
0.13
0.87
0.00
Unnormalized log-probabilities / logits
unnormalized probabilities
probabilities

METU

Cross-Entropy Distance (i.e. loss)

- Cross-entropy is commonly used in machine learning as a loss function.
- Cross-entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions.
- It is closely related to but is different from KL divergence that calculates the relative entropy between two probability distributions, whereas cross-entropy can be thought to calculate the total entropy between the distributions.



METU

Optimization

"By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."

- Ok, how? Random search?

Optimization

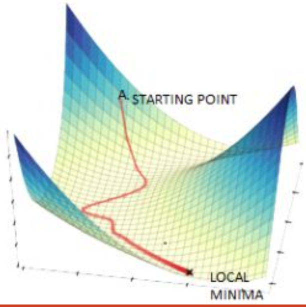
"By changing the model parameters (Θ*) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."*

- Ok, how? Random search?
  - Very bad idea.
  - Takes to long.
  - No strategy.
  - Experience shows that it does not work, when the parameter space is vast.

Optimization

"By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."

- Ok, how? Follow the slope (Gradient Descent)
  - For each iterative result, we may follow a direction, if the loss has a decreasing trend in that direction.
  -

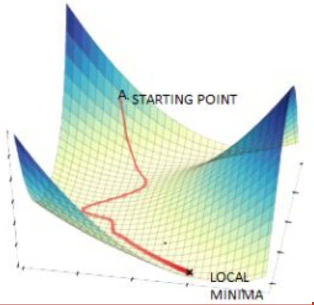


METU

Optimization

"By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."

- Ok, how? Follow the slope (Gradient Descent)
  - For each iterative result, we may follow a direction, if the loss has a decreasing trend in that direction.
  - It is like you are a blind man, walking around on an N-dimension field. Feeling the slope and following it.



Optimization

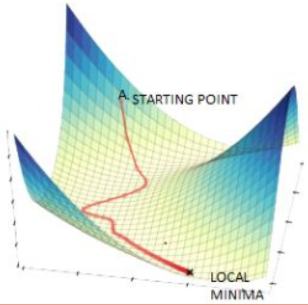
"By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."

In 1-dimension, the derivative of a function:

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

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**



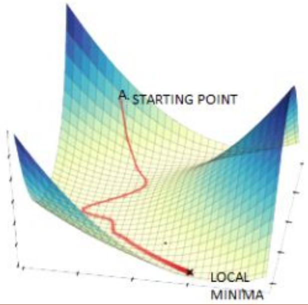
Gradient Descent

“By changing the model parameters (Θ) and “iteratively” checking if the score function is doing well (using the loss function), we find the correct parameters.”

- Repeat until convergence: *i.e. until you reach a plateau*

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \lambda \frac{\delta L(x_i, g_i, \Theta_{\text{old}})}{\delta \Theta_{\text{old}}}$$

- λ is the step size of the blind man,
  also called “the learning rate”



METU

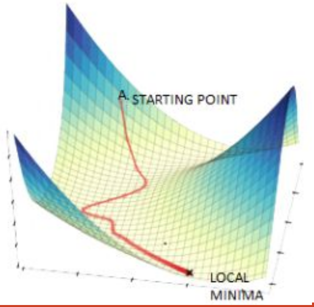
Gradient Descent

"By changing the model parameters (Θ) and "iteratively" checking if the score function is doing well (using the loss function), we find the correct parameters."

- So how to find the partial derivative of the loss function w.r.t the parameters.

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \lambda \frac{\delta L(x_i, g_i, \Theta_{\text{old}})}{\delta \Theta_{\text{old}}}$$

We will use a method called "backpropagation".



So far, so Good

Up to know we talked about

● Score function	:	$f(x_i,\Theta) = y_i$
● Ground Truth	:	g_i
● Loss function	:	$L(x_i , g_i) = g [f(x_i,\Theta) , g_i]$
● Gradient Descent	:	$\Theta_{\text{new}} = \Theta_{\text{old}} - \lambda \frac{\delta L (x_i, g_i, \Theta_{\text{old}})}{\delta\Theta_{\text{old}}}$

So far, so Good

Up to know we talked about

- Score function : $f(x_i,\Theta) = y_i$
- Ground Truth : g_i
- Loss function : $L(x_i, g_i) = g[f(x_i,\Theta), g_i]$
- Gradient Descent :

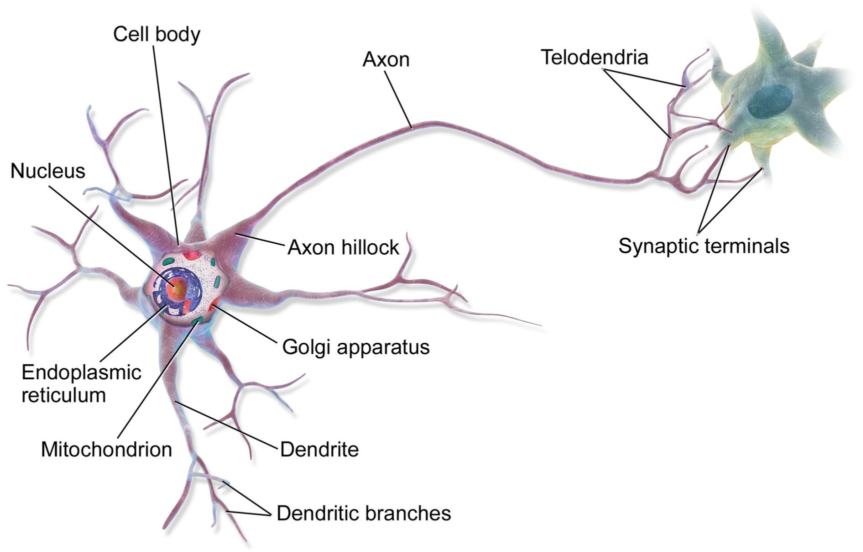
$$\Theta_{new} = \Theta_{old} - \lambda \frac{\delta L(x_i, g_i, \Theta_{old})}{\delta \Theta_{old}}$$

Now let's take a deep breath and start talking about Neural Networks, and focus on how backpropagation works, later.

METU

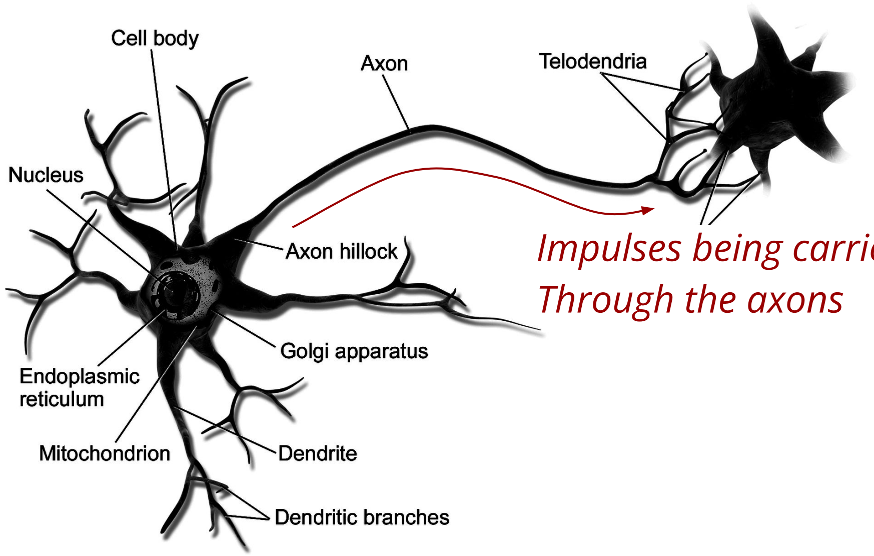
Neurons

Neurons are the main component of nervous tissue in all animals (except sponges and placozoa).



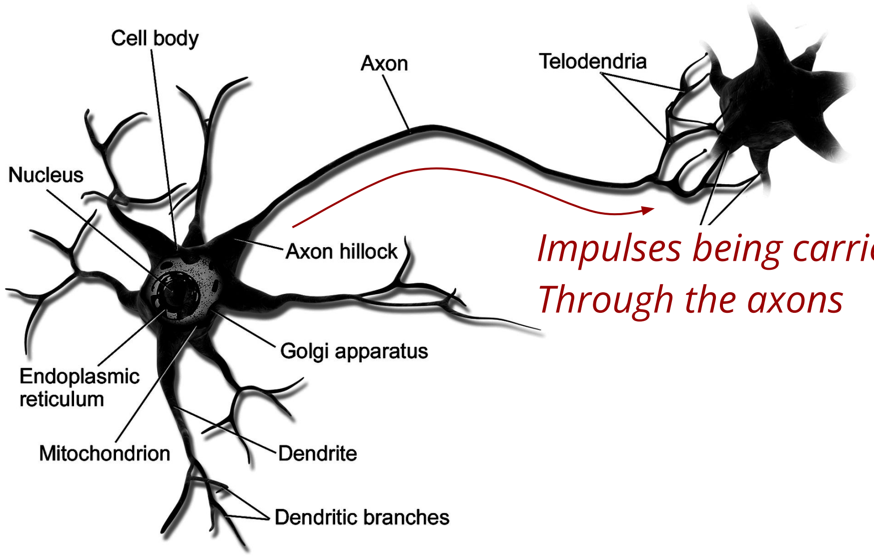
Neurons

Neurons are the main component of nervous tissue in all animals (except sponges and placozoa).



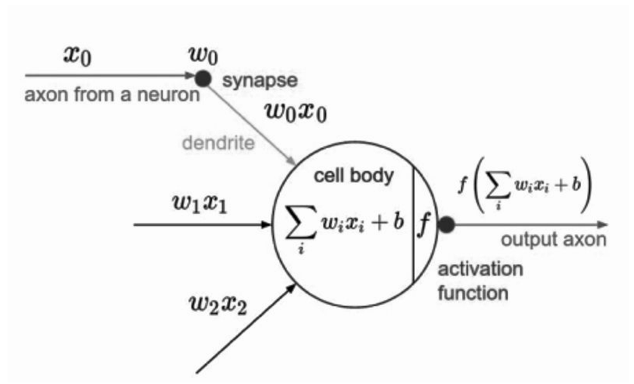
Impulses being carried away
Through the axons

Artificial Neurons



Impulses being carried away Through the axons

An artificial neuron is a mathematical function conceived as a model of biological neurons.



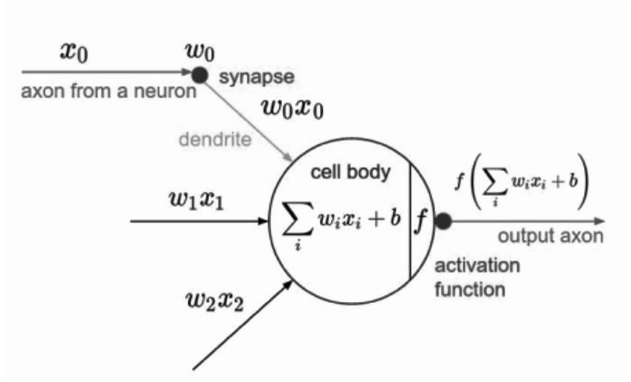
METU

Artificial Neurons

Artificial Neuron is a (simple) *score function* in a Neural Network.

- $\boldsymbol{f}\,(x_{Nx1}, \mathcal{W}_{NxN}, \boldsymbol{b}_{Nx1}) = g\,[\,\Sigma_N\,(x^T \cdot \mathcal{W} + \boldsymbol{b}\,)\,]$

  where $g\,(\cdot)$ is a nonlinear "activation function"

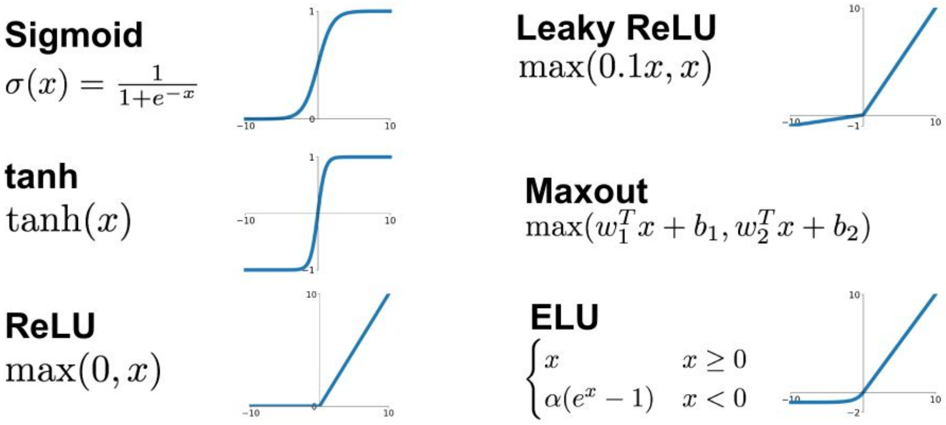


Artificial Neurons

Artificial Neuron is a (simple) *score function* in a Neural Network.

- $\boldsymbol{f}\,(\,x_{\mathrm{Nx1}},\, \mathcal{W}_{\mathrm{NxN}}\,,\, \boldsymbol{b}_{\mathrm{Nx1}}) = g\,[\,\Sigma_{\mathrm{N}}\,(x^{\mathrm{T}}\cdot\mathcal{W} + \boldsymbol{b}\,)\,]$

  where $g\,(\cdot)$ is a nonlinear "activation function"



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

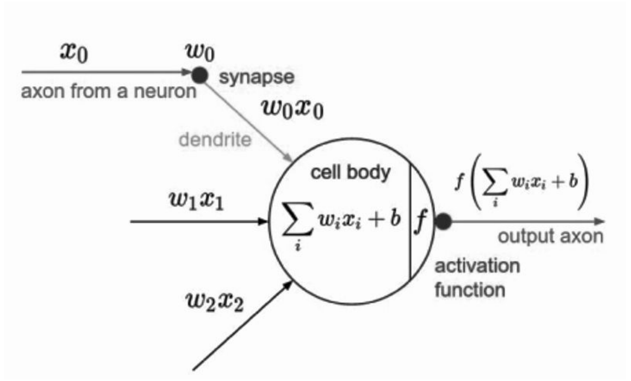
tanh
$\tanh(x)$

ReLU
$\max(0, x)$

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

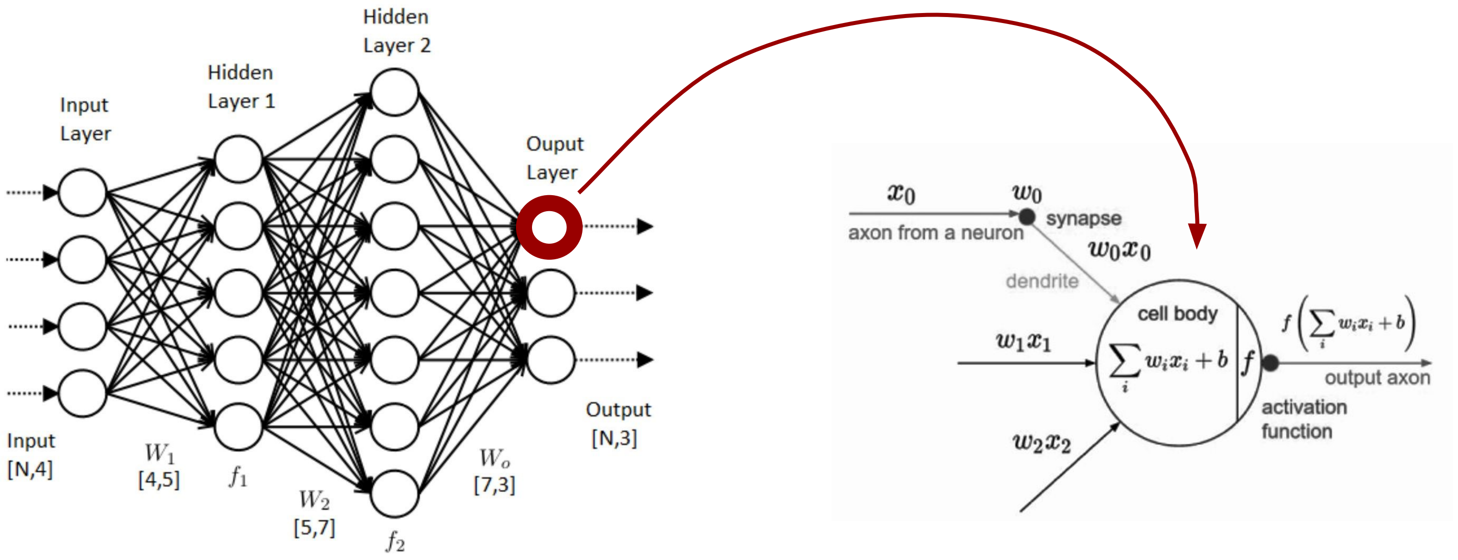
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

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



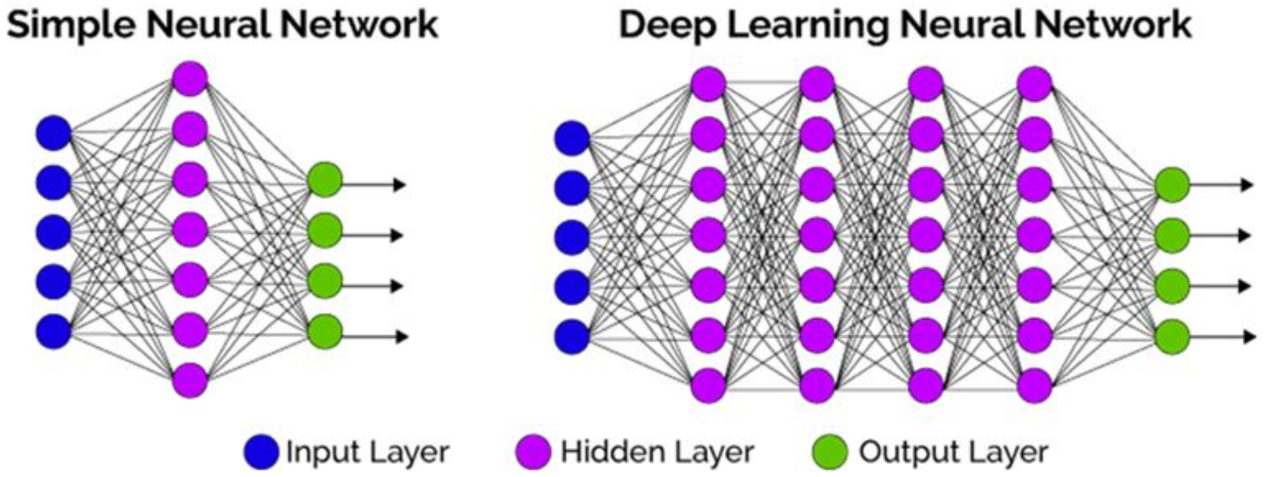
Artificial Neural Networks (ANNs)

ANNs (in short) are based on a collection of connected artificial neurons, (which loosely model the neurons in a biological brain).



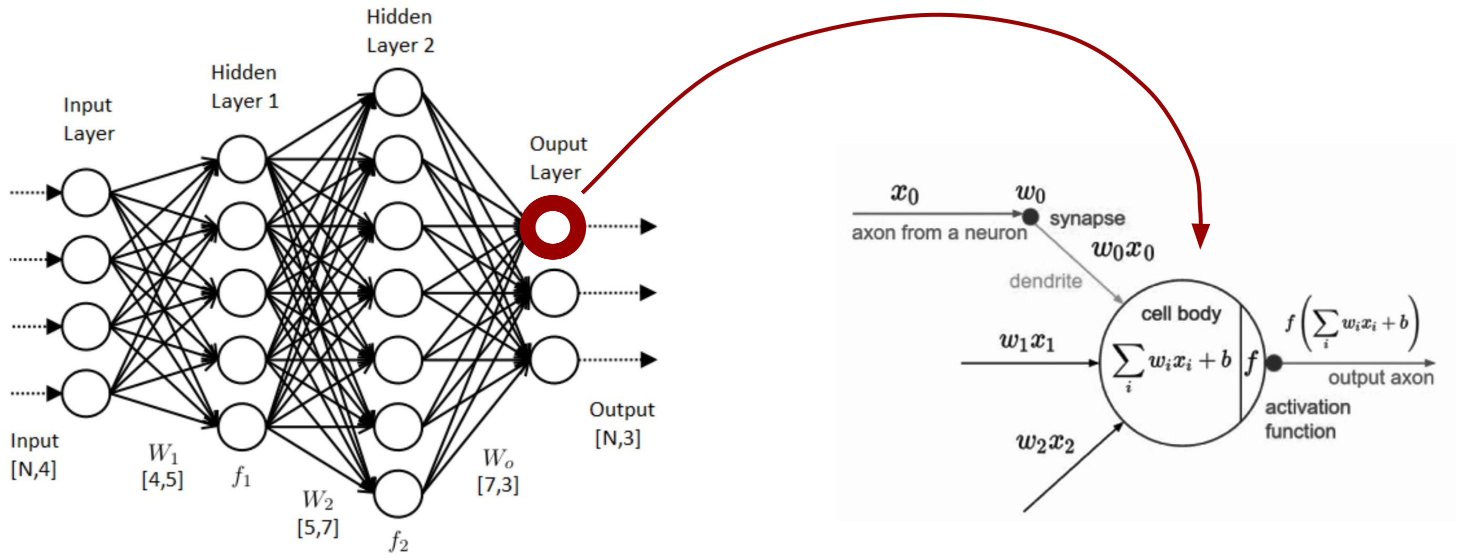
Artificial Neural Networks (ANNs)

When there are a lot of layers, we simple call these architectures “deep”.



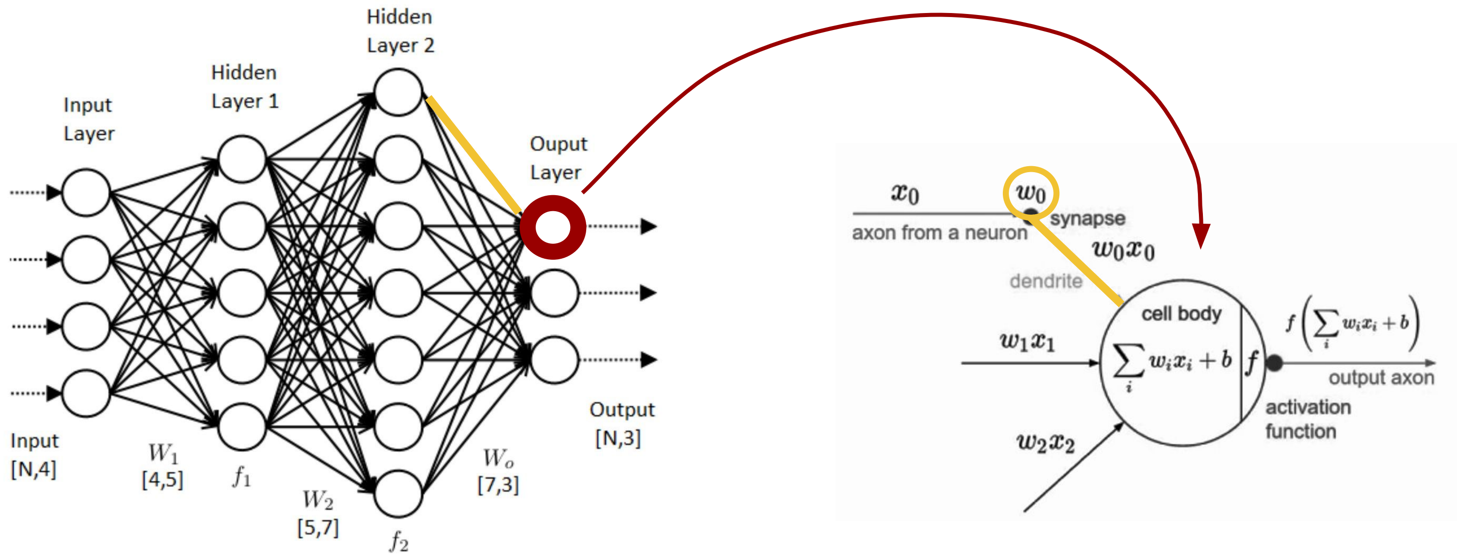
Artificial Neural Networks (ANNs)

What defines an ANN is the total set of **"Weights"** and each connection in an ANN represents a weight.



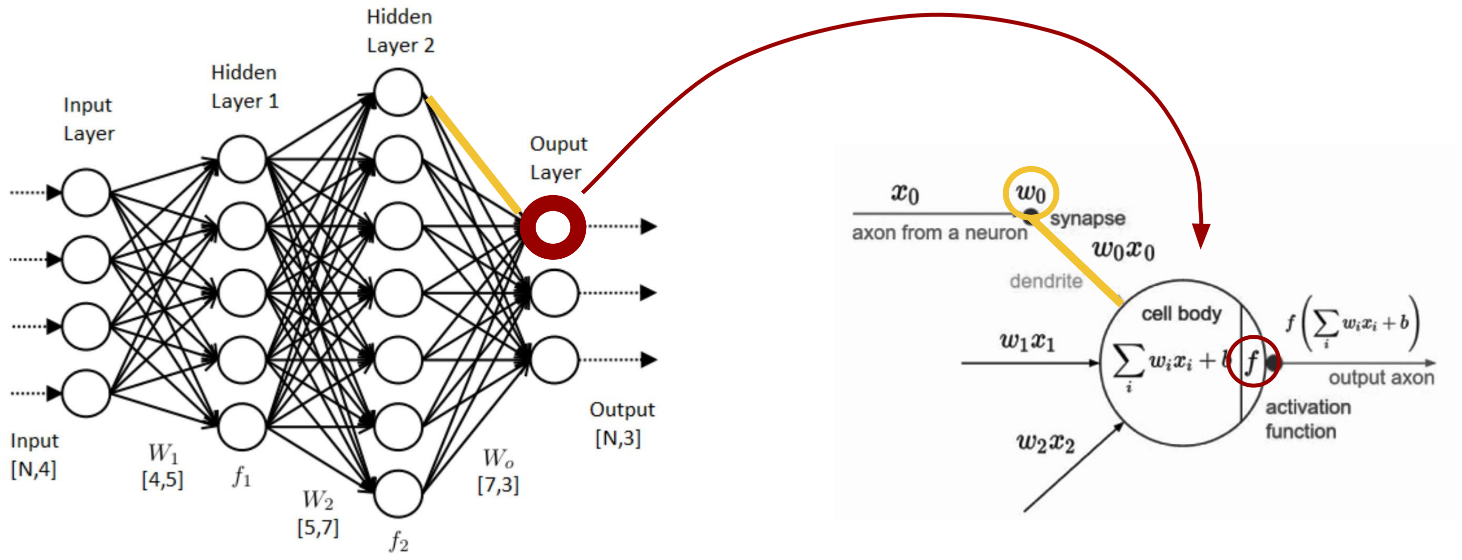
Artificial Neural Networks (ANNs)

What defines an ANN is the total set of **"Weights"** and each connection in an ANN represents a weight.



Artificial Neural Networks (ANNs)

The sum is applied into an "activation function" *f* which is usually non linear

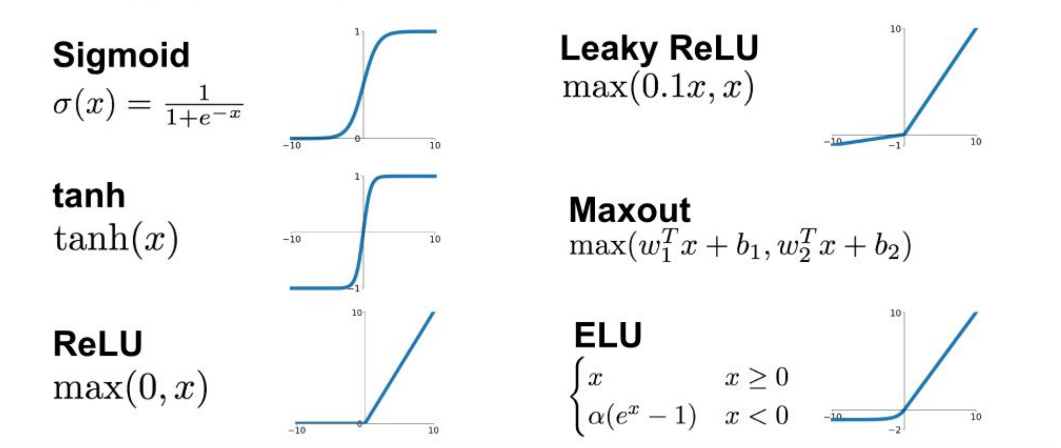


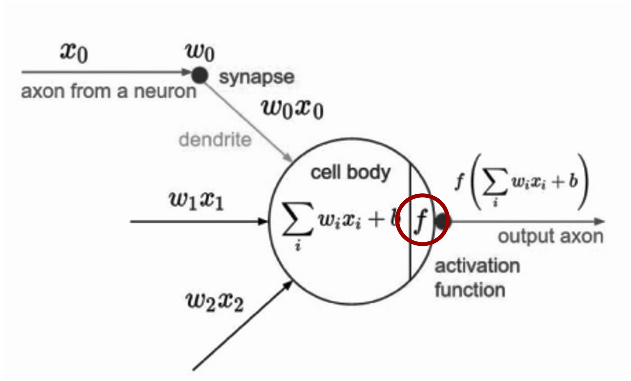
METU

Artificial Neural Networks (ANNs)

The sum is applied into an "activation function" *f* which is usually non linear

" *f* is always non-linear "

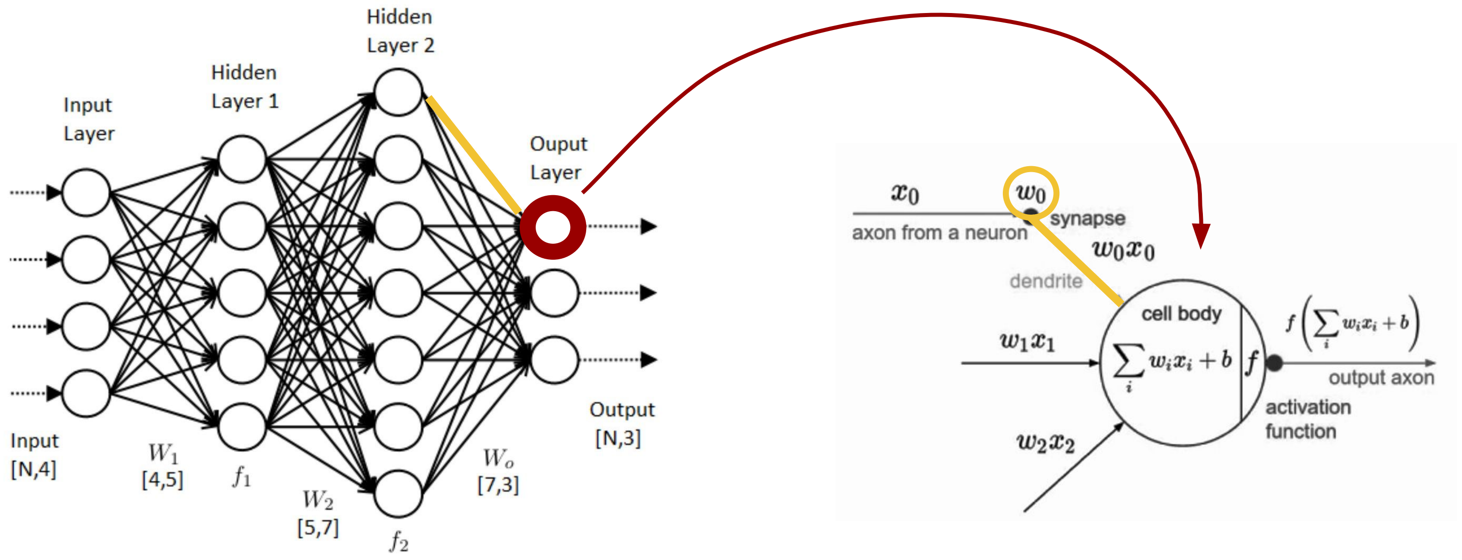




METU

Artificial Neural Networks (ANNs)

??? B U T - H O W - T O - F I N D - T H O S E - W E I G H T S ???



METU

Computational Graphs

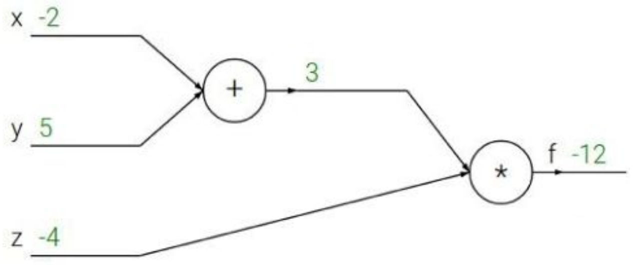
- A computational graph is a way to represent a mathematical function in the language of graph theory.
  - Graph Theory in a nutshell: *nodes are connected by edges, and everything in the graph is either a node or an edge.*

Computational Graphs

- A computational graph is a way to represent a mathematical function in the language of graph theory.
  - Graph Theory in a nutshell: *nodes are connected by edges, and everything in the graph is either a node or an edge.*
  - Simple Example:

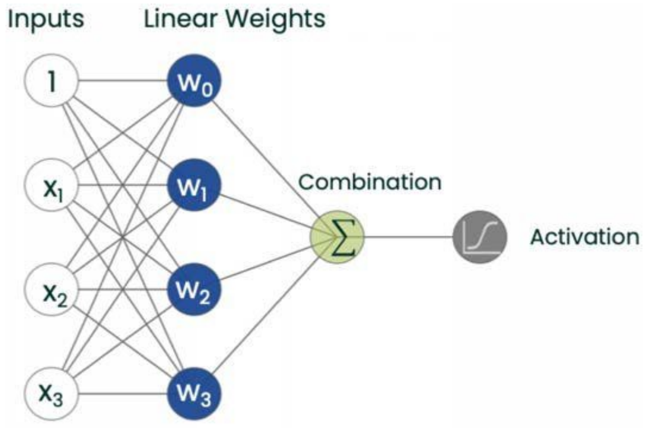
$$f(x, y, z) = (x + y)z$$

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



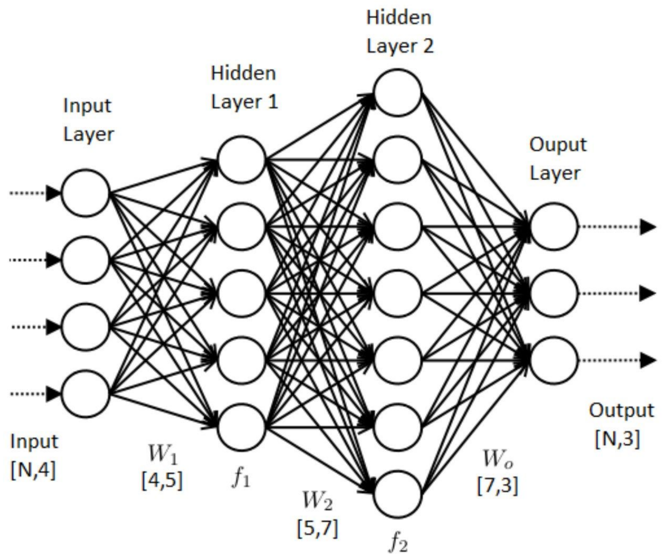
Computational Graphs

- An ANN is a text-book computational graphs:
- Below is a computational graph for an example neural network with three inputs and one hidden layer.



Computational Graphs

- An ANN is a text-book computational graphs:
- ANN may get more complex, but it never loses its property of being a computational graph:
    - *nodes are connected by edges, and everything in the graph is either a node or an edge.*



Backpropagation

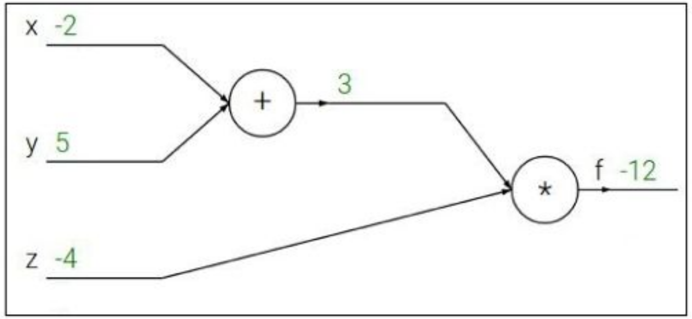
- Backpropagation (BP) is short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent.
- Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the ANN’s weights.

Backpropagation

- Applying BP to computational graphs is simple:

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

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



Backpropagation

- Applying BP to computational graphs is simple:

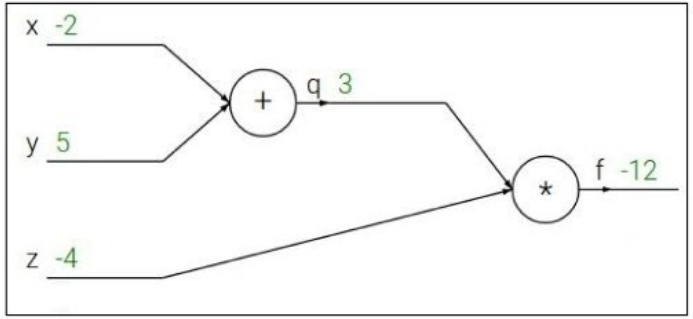
$$f(x,y,z) = (x+y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

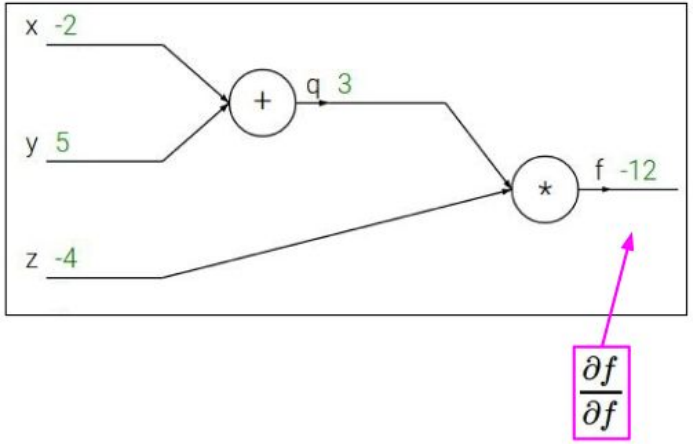
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

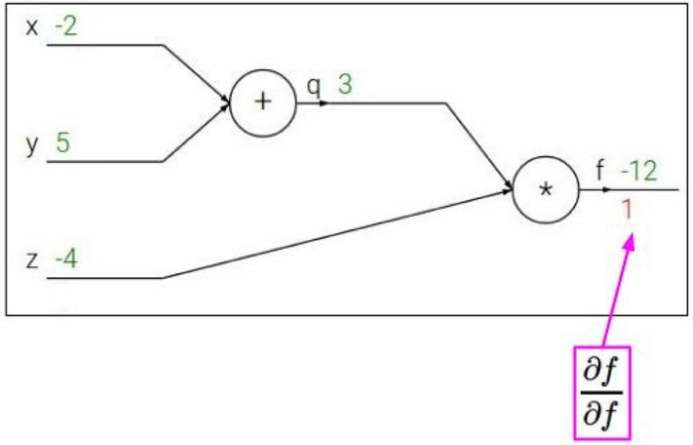
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

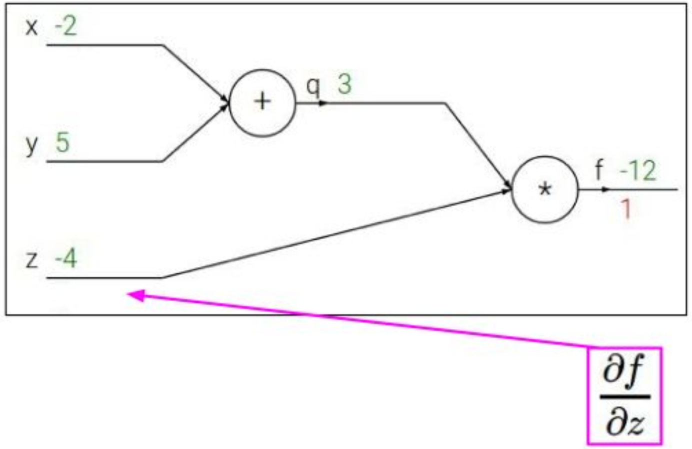
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

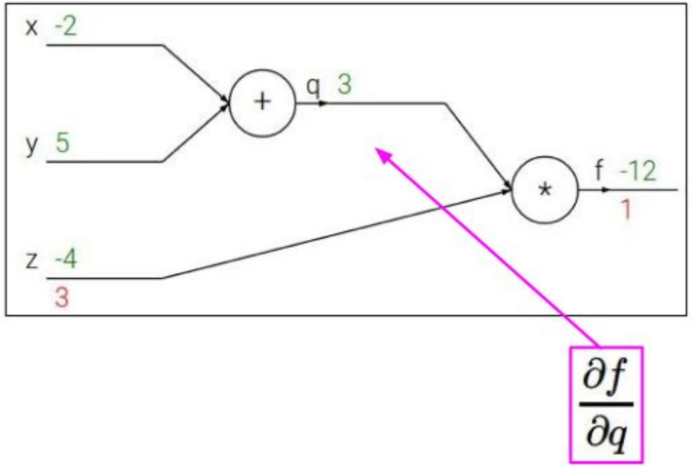
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

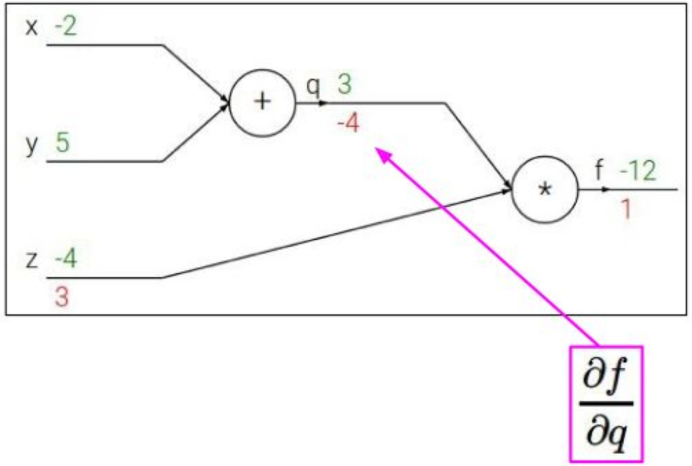
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

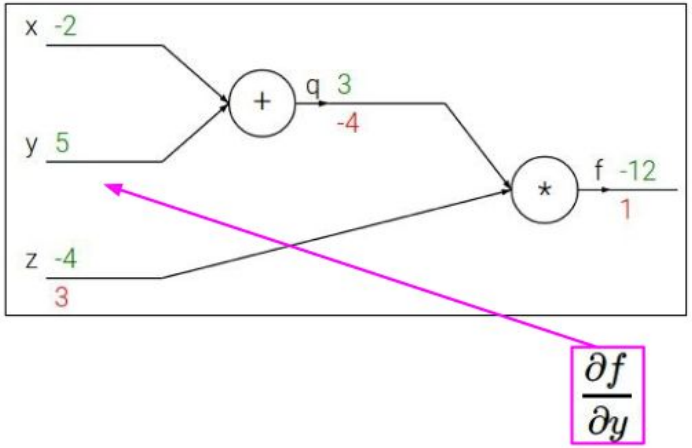
$$f(x,y,z) = (x+y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

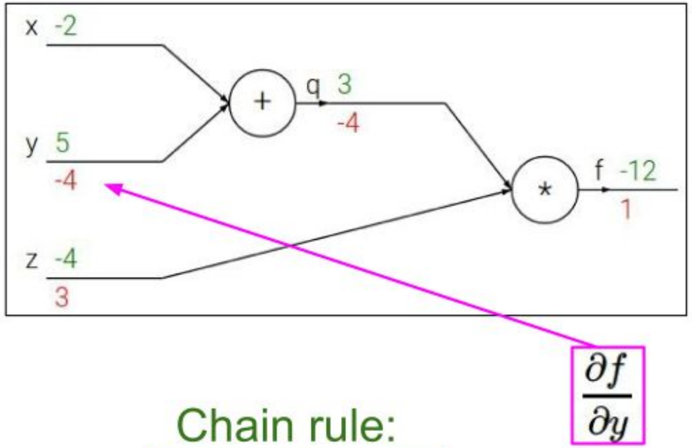
$$f(x, y, z) = (x + y)z$$

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



Chain rule:

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

Backpropagation

- Applying BP to computational graphs is simple:

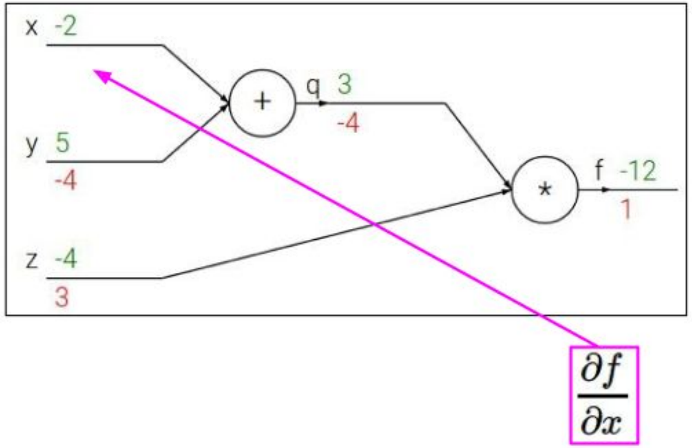
$$f(x,y,z) = (x+y)z$$

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

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

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

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



Backpropagation

- Applying BP to computational graphs is simple:

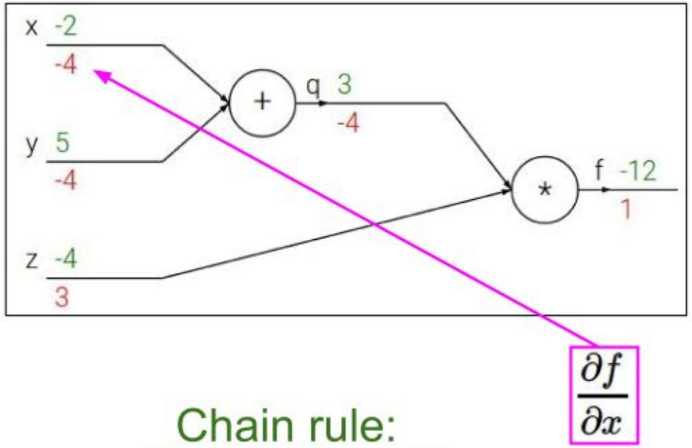
$$f(x,y,z) = (x+y)z$$

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:

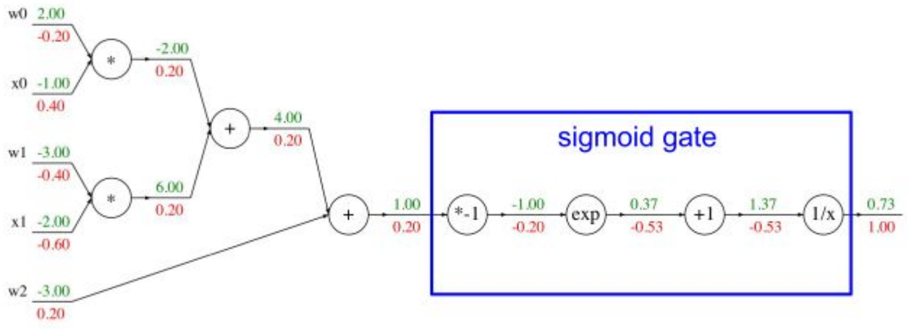
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}}$$

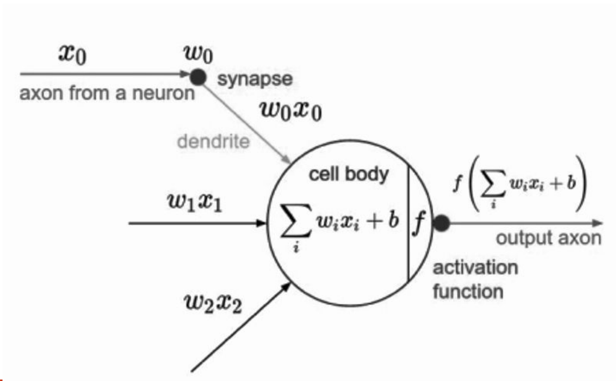
Backpropagation (of a single node)

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

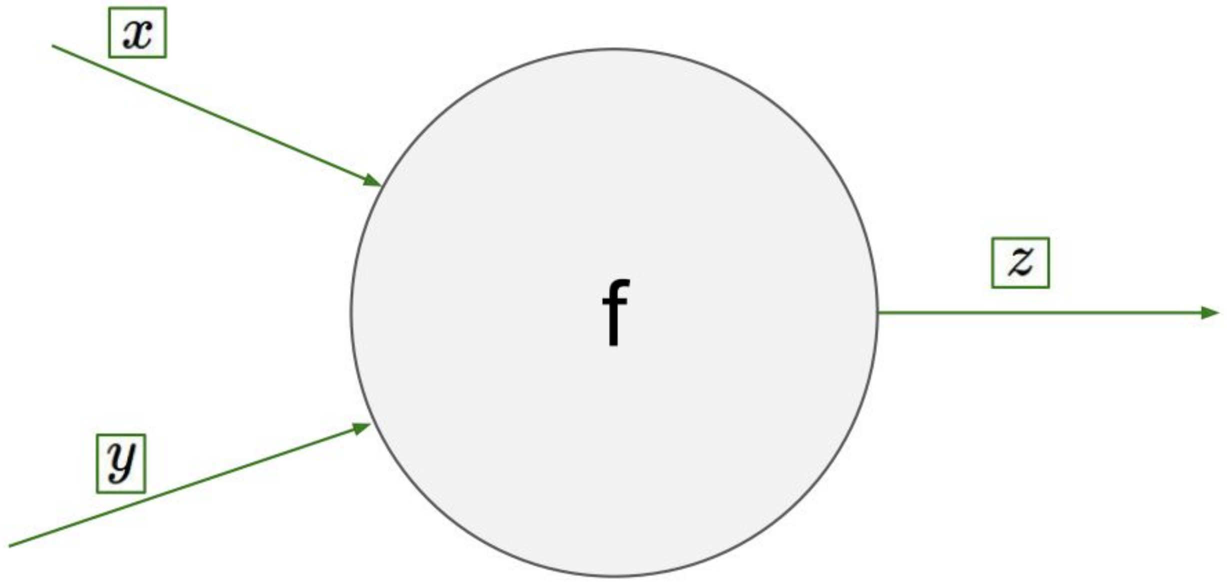
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$ sigmoid function

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

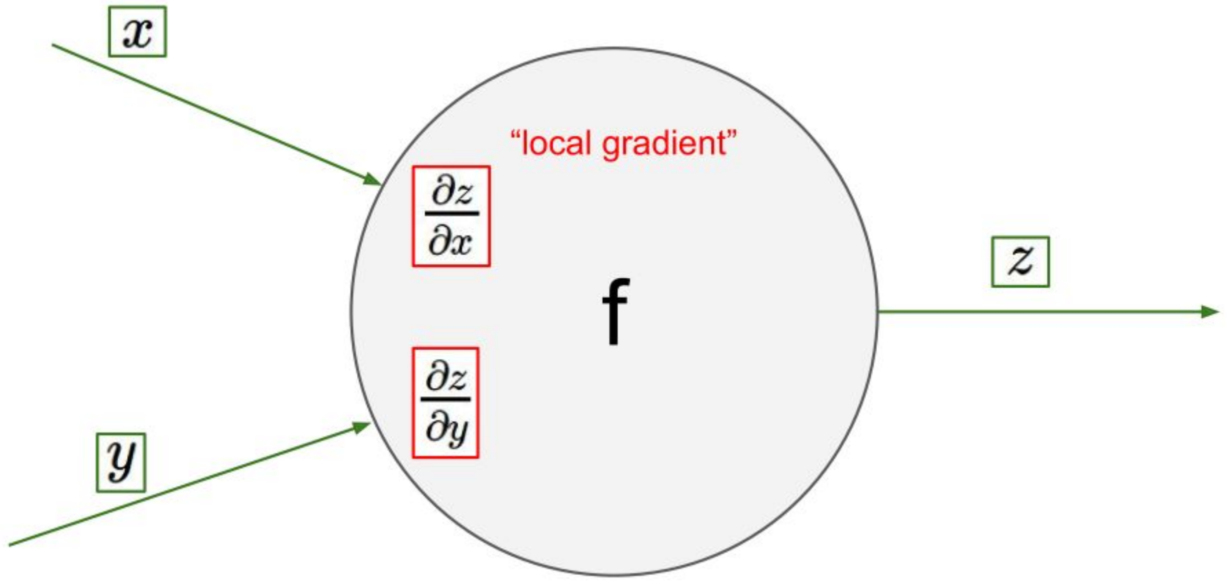




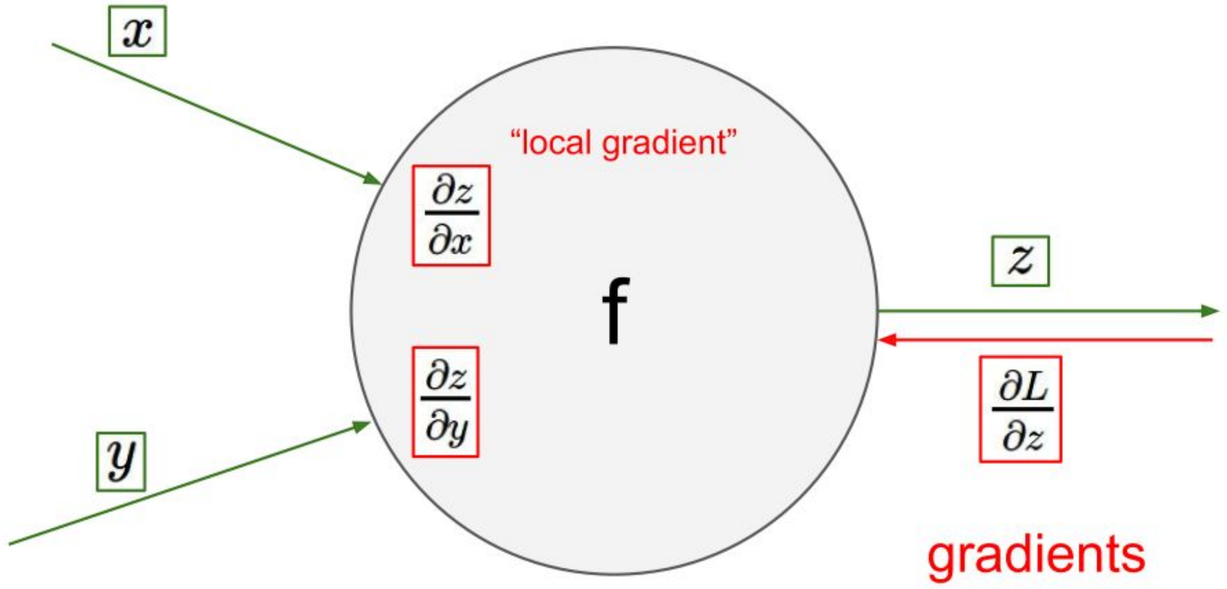
Backpropagation (in a block)



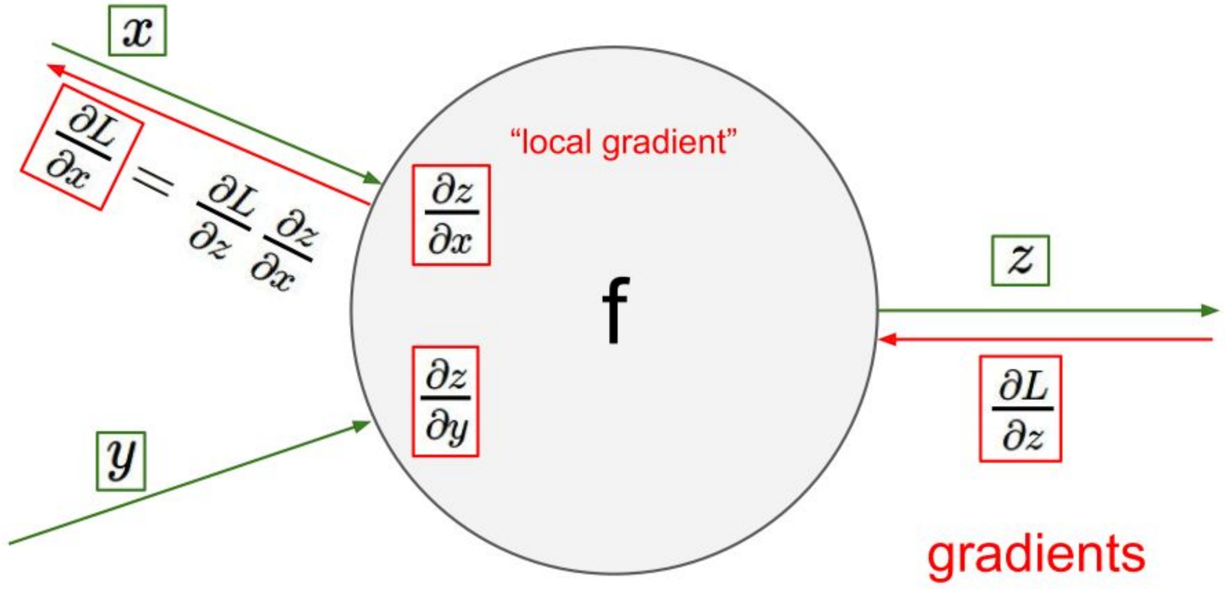
Backpropagation (in a block)


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

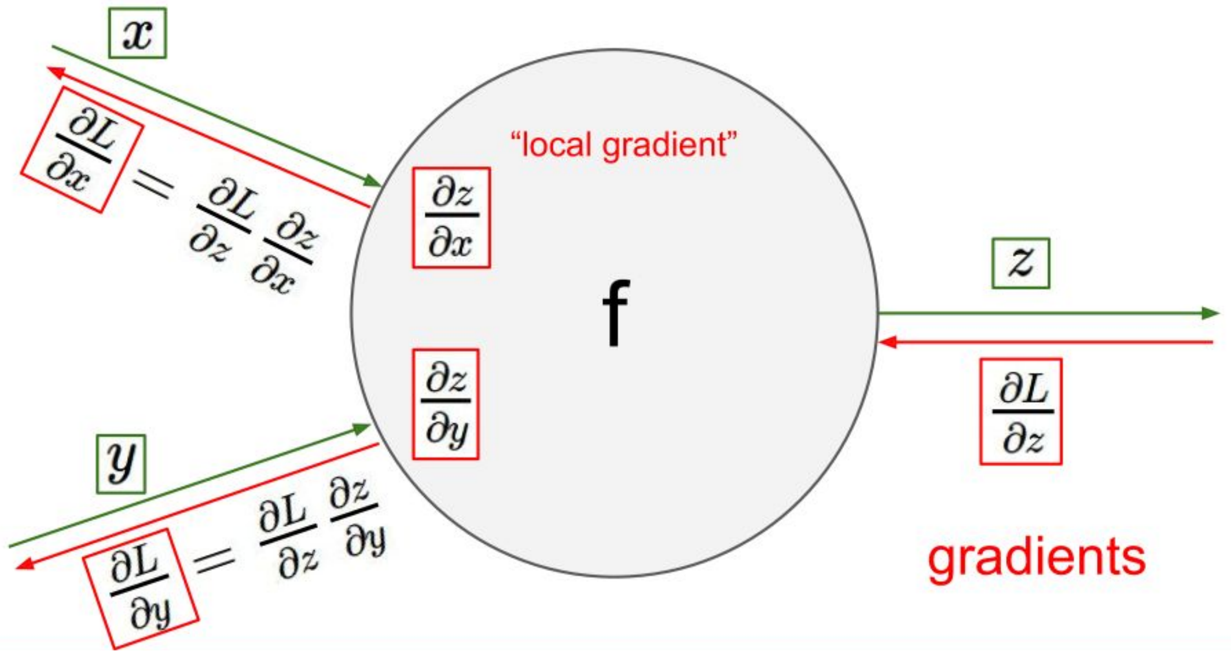
Backpropagation (in a block)


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

Backpropagation (in a block)

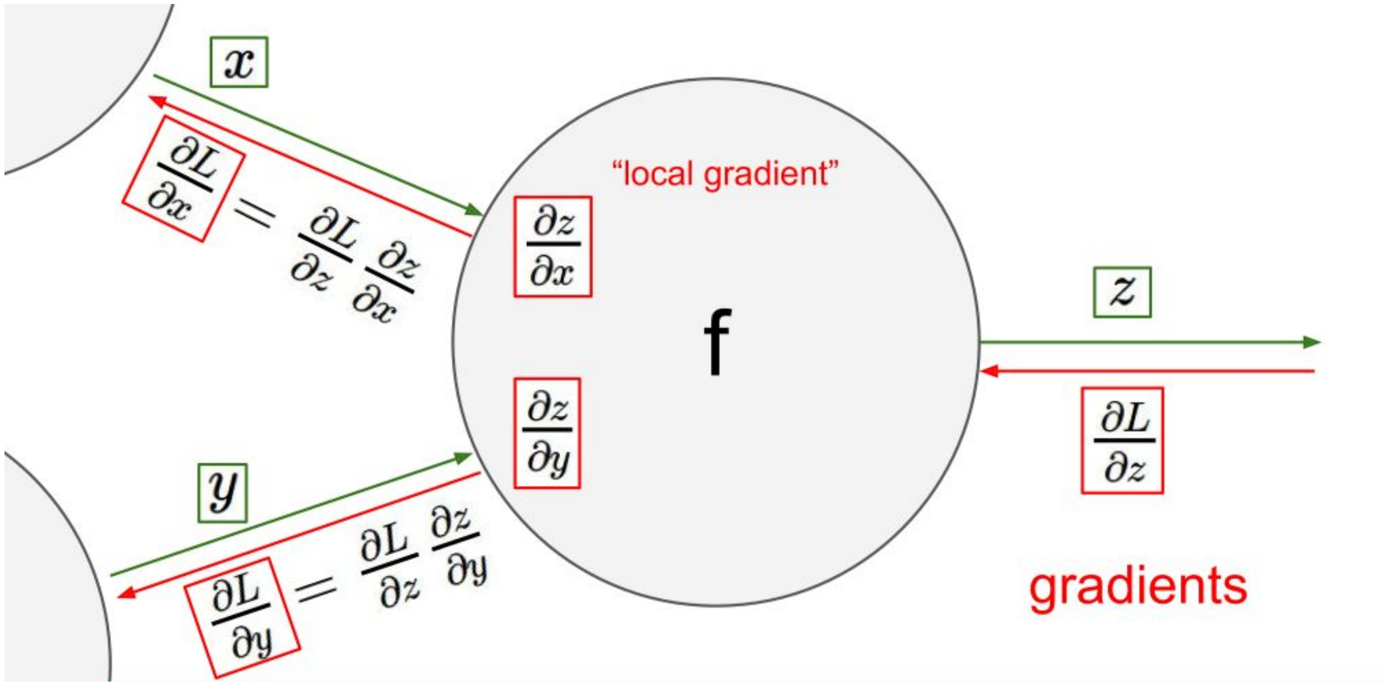

x
y
z
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
gradients

Backpropagation (in a block)



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

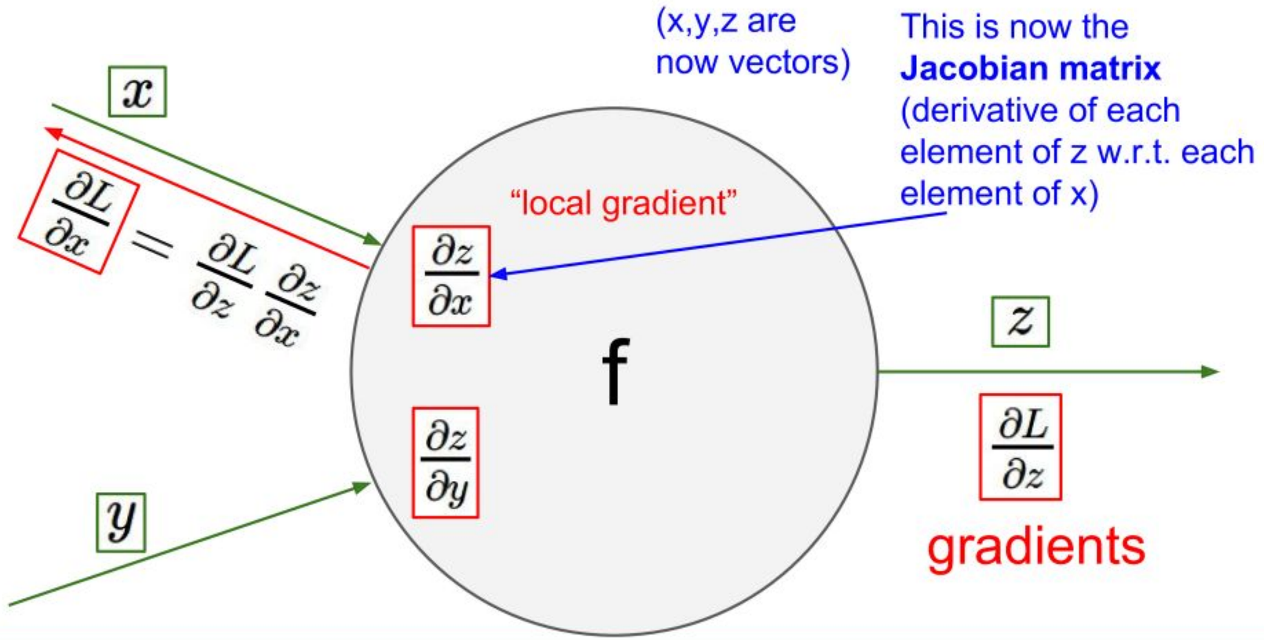
Backpropagation (in a block)



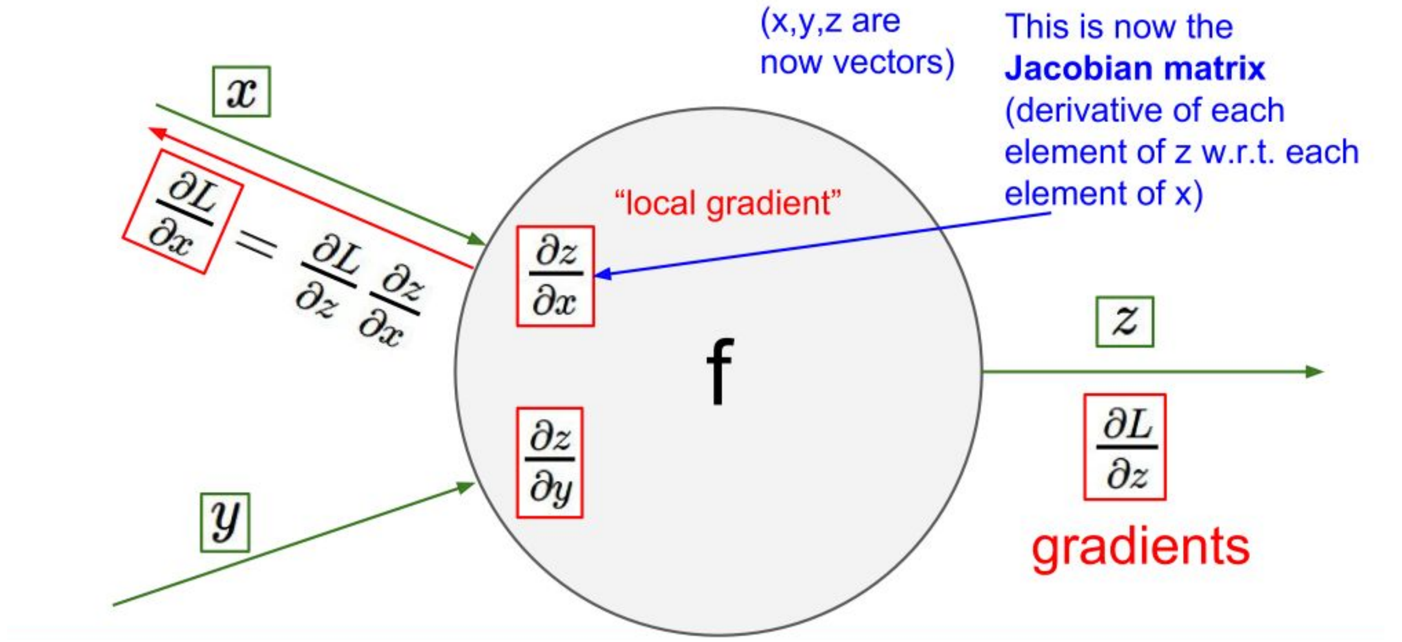
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial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

Figures and example from Stanford's CS231n 2017 slides

Backpropagation (of vector I/O)


(x,y,z are now vectors)
This is now the **Jacobian matrix** (derivative of each element of z w.r.t. each element of x)
x
y
z
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial z}$
gradients

Backpropagation (of vector I/O)

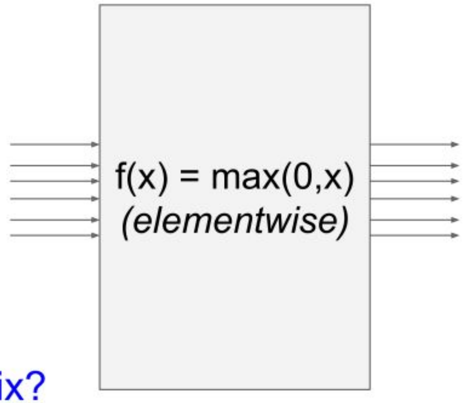

(x,y,z are now vectors)
This is now the **Jacobian matrix** (derivative of each element of z w.r.t. each element of x)
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
gradients
y

Backpropagation (of vector I/O)

Vectorized operations

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

Jacobian matrix



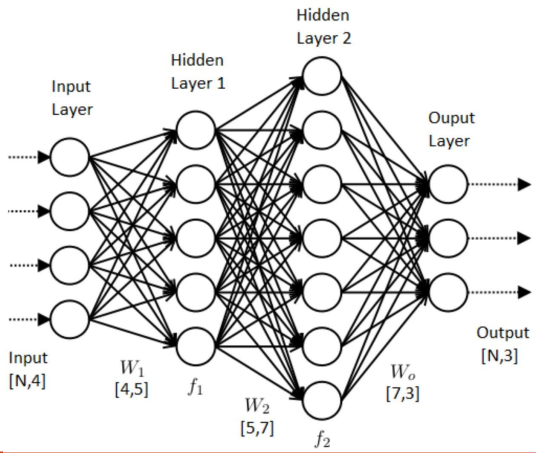
4096-d
input vector

4096-d
output vector

Q: what is the size of the Jacobian matrix?
[4096 x 4096!]

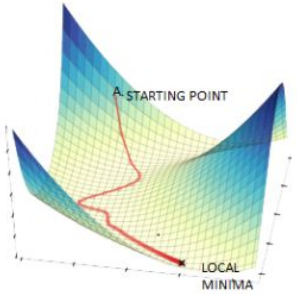
Training Neural Nets

- Training consists of two main steps:
  - Forward pass: Function Evaluation,
  - Backward bass: Backpropagation.



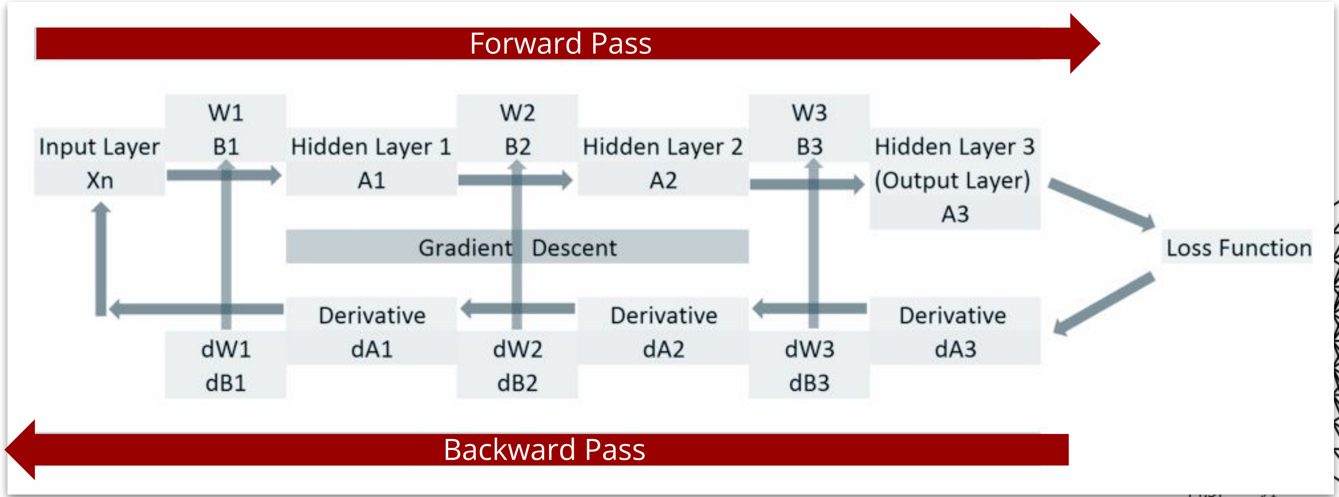
METU

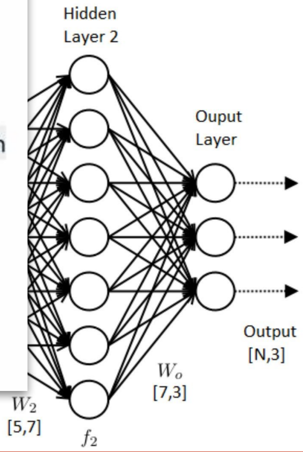
Figure from here.



Training Neural Nets

- Training consists of two main steps:
  - Forward pass: Function Evaluation,
  - Backward bass: Backpropagation.





Solvers

- The solver orchestrates model optimization by coordinating the network’s forward inference and backward gradients to form parameter updates that attempt to improve the loss.
- The responsibilities of learning are divided between the Solver for overseeing the optimization and generating parameter updates and the Net for yielding loss and gradients.

Solvers

- scaffolds the optimization bookkeeping and creates the training network for learning and test network(s) for evaluation,
- iteratively optimizes by calling forward / backward and updating parameters,
- (periodically) evaluates the test/validation networks,
- snapshots the model and solver state throughout the optimization.
-

Solvers

- scaffolds the optimization bookkeeping and creates the training network for learning and test network(s) for evaluation,
- iteratively optimizes by calling forward / backward and updating parameters,
- (periodically) evaluates the test/validation networks,
- snapshots the model and solver state throughout the optimization.
- is the implementation of the optimization algorithm in code.

METU

Solvers

- Various solvers are implemented in most DL libraries, such as:
    - Stochastic Gradient Descent (type: "SGD"),
    - AdaDelta (type: "AdaDelta"),
    - Adaptive Gradient (type: "AdaGrad"),
    - Adam (type: "Adam"),
    - Nesterov’s Accelerated Gradient (type: "Nesterov") and
    - RMSprop (type: "RMSProp")
- These are different implementations of gradient descent.

Stochastic Gradient Descent (SGD)

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

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

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Stochastic Gradient Descent (SGD)

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

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

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Regularization

- Regularization is the process of adding information in order to solve an (usually ill-posed) optimization problem and/or to prevent overfitting.

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data loss}\text{: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}\text{: Prevent the model from doing } \textit{too} \text{ well on training data}}$$

Regularization

- Regularization is the process of adding information in order to solve an (usually ill-posed) optimization problem and/or to prevent overfitting.

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\substack{\textbf{Data loss}\text{: Model predictions} \\ \text{should match training data}}} + \underbrace{\lambda R(W)}_{\substack{\textbf{Regularization}\text{: Prevent the model} \\ \text{from doing } \textit{too} \text{ well on training data}}}$$

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

Regularization

- Regularization is the process of adding information in order to solve an (usually ill-posed) optimization problem and/or to prevent overfitting.

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

Data loss: Model predictions should match training data

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

Simple examples

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

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

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

METU

Solvers, Optimization, etc...

- How will you know?
- Which solver, regularization, parameters?
- SGD is the slowest one and is hardly used.
- Most models implement either
  - **Adam**
    - *a parameter adaptive version of SGD version with momentum, or*
  - **RMSProp**
    - *Similar to Adam.*

Live Code!

- Let's see how this algorithm is implemented.
- Please, find the code here at https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

What to do until next week?

- You already (!) have a working DL environment.
- Think of a project idea. Preferably related to your thesis.
  - ➢ You don’t know how to do it yet, even if it can be done or not. Do some research. Did people do it before? What could they achieve? Maybe you will implement a modification on the problem? Start thinking about it.
  - ➢ Start preparing an abstract on your project idea and next time, ask for feedback.

METU

What will we do next week?

- Starting with next week...
  - ➢ Convolutional Neural Networks
- Following weeks
  - ➢ Deep Architectures, Introduction to NLP ...

Additional Reading & References

- https://medium.com/tebs-lab/deep-neural-networks-as-computational-graphs-867fcaa56c9
- https://morioh.com/p/7748f371ab45
- http://cs231n.stanford.edu/slides/2017/
- https://www.researchgate.net/publication/341061782_A_Digital_Twin_of_Drilling_Fluids_Rheology_for_Real-Time_Rig_Operations/figures?lo=1
- https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- https://medium.com/dair-ai/a-simple-neural-network-from-scratch-with-pytorch-and-google-colab-c7f3830618e0
- file:///C:/Users/Erdem%20Akag%C3%BCnd%C3%BCz/Downloads/symmetry-10-00648.pdf
- https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab
- https://ruder.io/optimizing-gradient-descent/
- https://machinelearningmastery.com/cross-entropy-for-machine-learning/