Deep Learning

Neural Networks – Learning the network

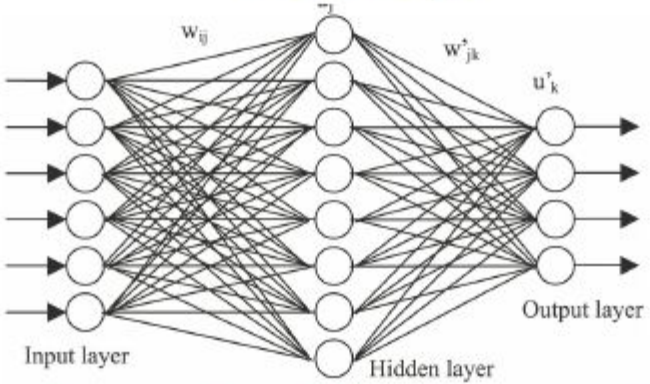
Slide Credit:

1. Bikash Raj, 11-785, Fall 2017
2. Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Topics for the session

- The problem of learning
- The perceptron rule for perceptrons
  - And its inapplicability to multi-layer perceptrons
- Learning through Empirical Risk Minimization
- Intro to function optimization and gradient descent
- Computational Graphs
- Back Propagation Based Deep Networks

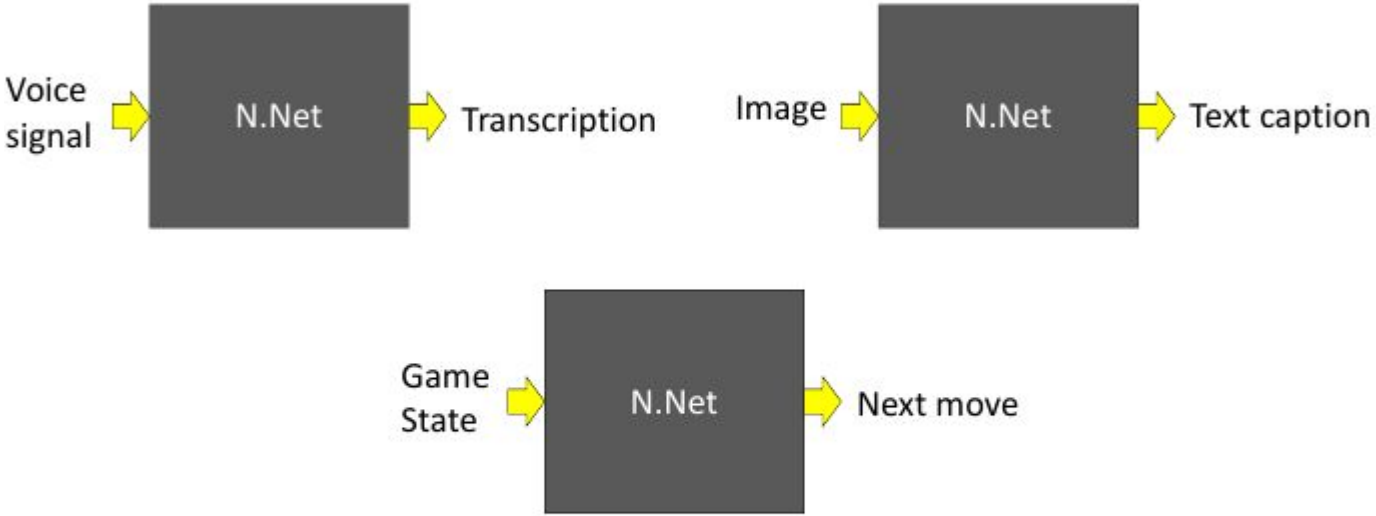
Recap



- **Neural networks are universal function approximators**
  - Can model any Boolean function
  - Can model any classification boundary
  - Can model any continuous valued function

- *Provided the network satisfies minimal architecture constraints*
  - Networks with fewer than required parameters can be very poor approximators

These boxes are functions



- Take an input
- Produce an output
- Can be modeled by a neural network!

Questions

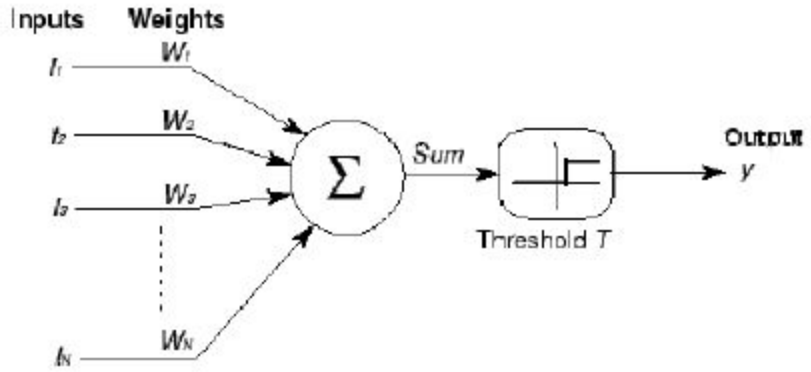
- Preliminaries:
  - How do we represent the input?
  - How do we represent the output?
- How do we compose the network that performs the requisite function?

Questions

- Preliminaries:
  - How do we re[illegible]nt the input?
  - How do [illegible]epresent the output?
- *How do we compose the network that performs the requisite function?* ⟵

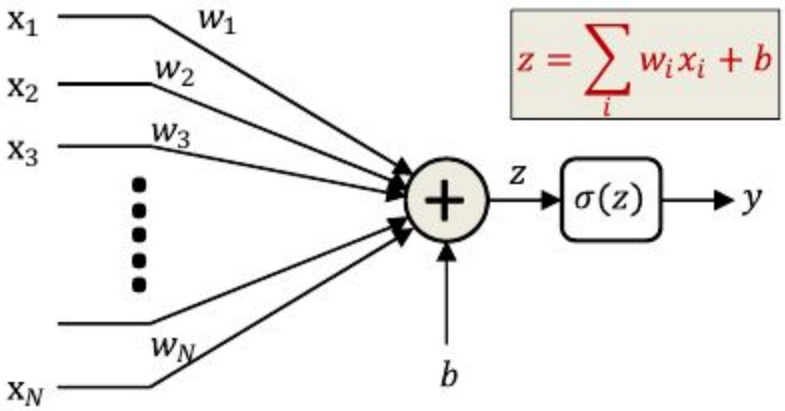
A bit later in the program

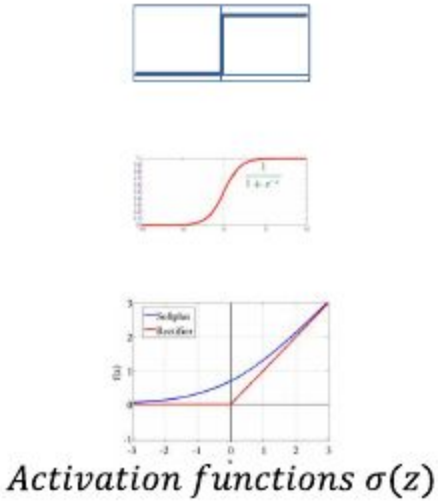
The original perceptron



- Simple threshold unit
  - Unit comprises a set of weights and a threshold

Preliminaries: The units in the network

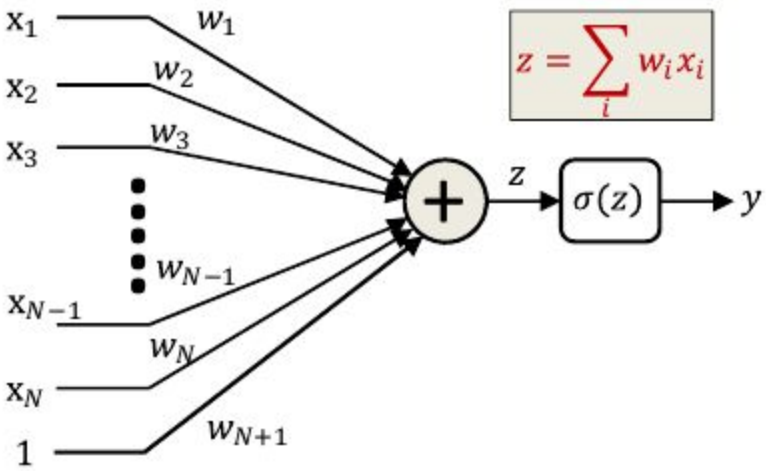


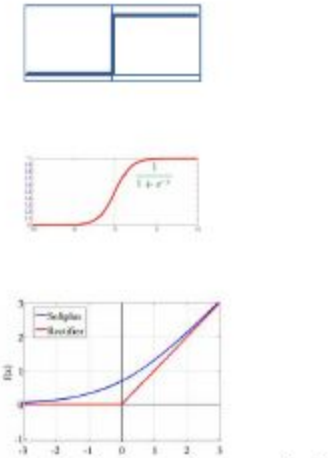


Activation functions $\sigma(z)$

- Perceptron
  - General setting, inputs are real valued
  - Activation functions are not necessarily threshold functions
  - A *bias* b representing a threshold to trigger the perceptron

Preliminaries: Redrawing the neuron

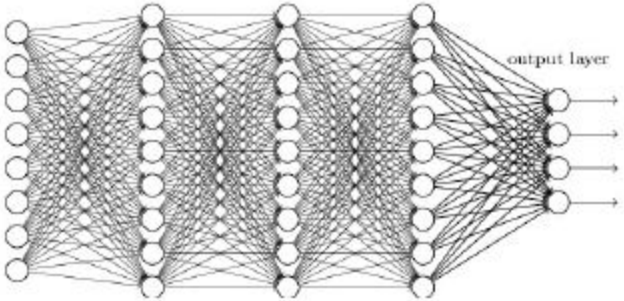


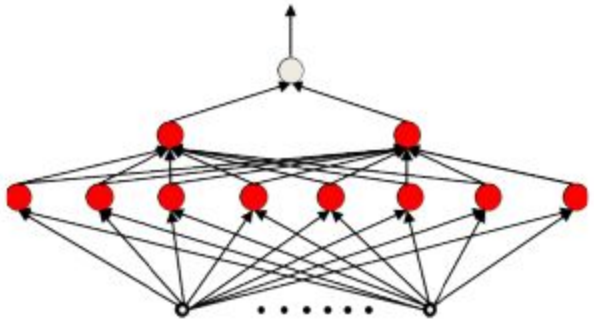


Activation functions $\sigma(z)$

- The bias can also be viewed as the weight of another input component that is always set to 1
  - If the bias is not explicitly mentioned, we will implicitly be assuming that every perceptron has an additional input that is always fixed at 1

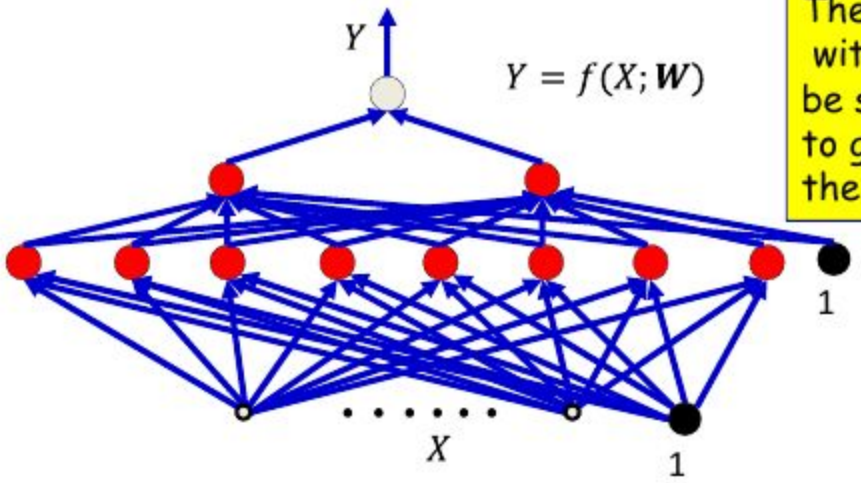
First: the structure of the network





- We will assume a *feed-forward* network
  - No loops: Neuron outputs do not go feed back to their inputs directly or indirectly
  - Loopy networks are a future topic
- **Part of the design of a network: The architecture**
  - How many layers/neurons, which neuron connects to which and how, etc.
- For now, assume the architecture of the network is capable of representing the needed function

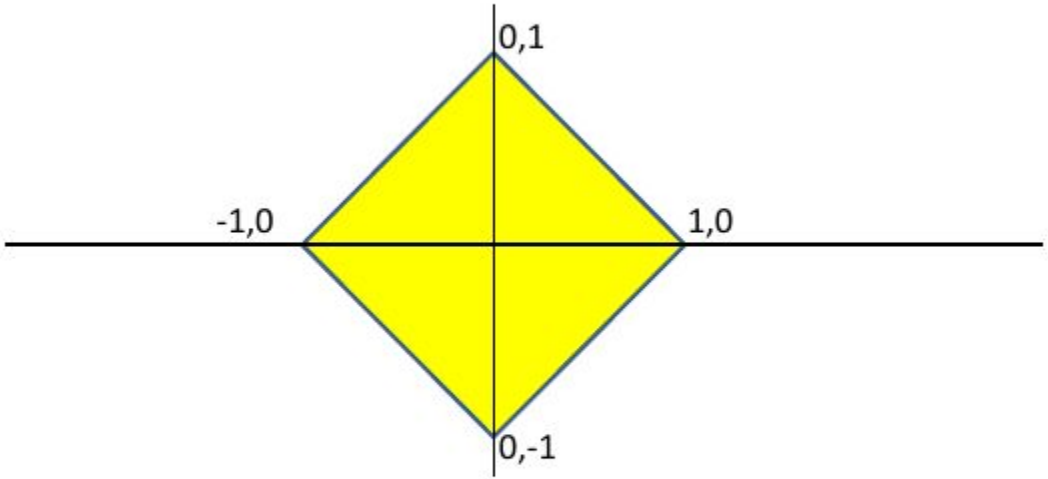
What we learn: The parameters of the network



The network is a function f() with parameters W which must be set to the appropriate values to get the desired behavior from the net

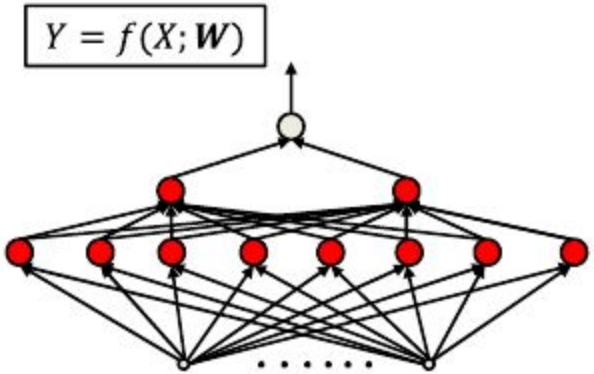
- **Given:** the architecture of the network
- The parameters of the network: The weights and biases
  - The weights associated with the blue arrows in the picture
- *Learning the network :* Determining the values of these parameters such that the network computes the desired function

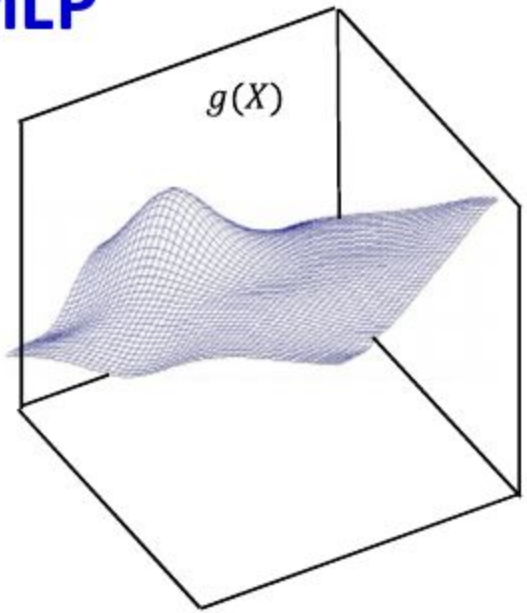
Option 1: Construct by hand



- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary
- Not possible for all but the simplest problems..

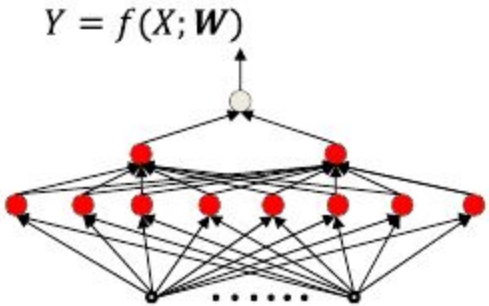
Option 2: Automatic estimation of an MLP

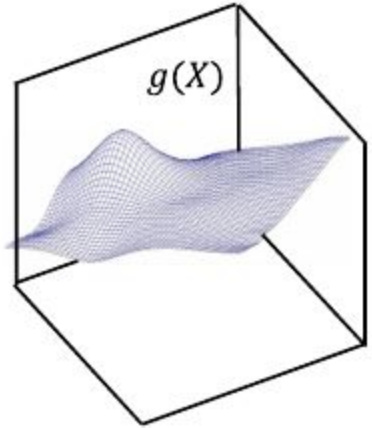




- More generally, *given* the function $g(X)$ to model, we can *derive* the parameters of the network to model it, through computation

How to learn a network?



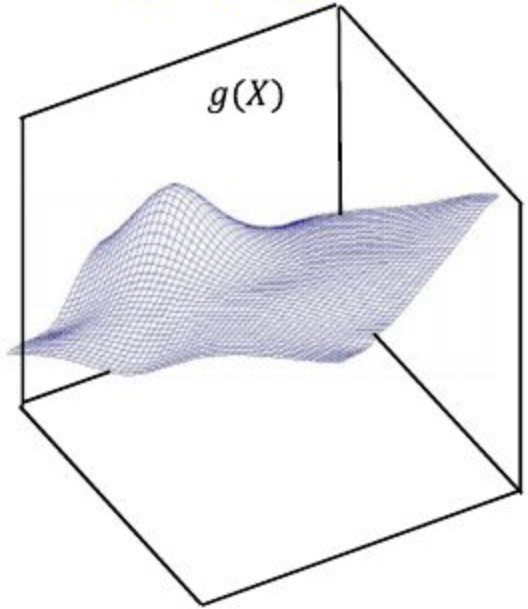


- When $f(X; \boldsymbol{W})$ has the capacity to exactly represent $g(X)$

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}} \int_X div\big(f(X; W), g(X)\big) dX$$

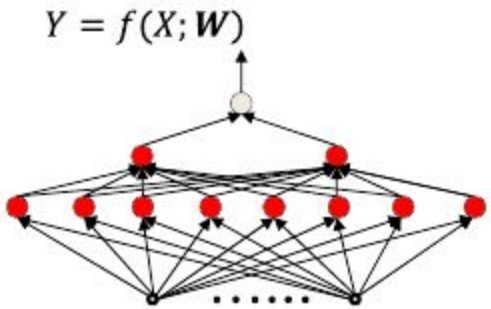
- div() is a *divergence* function that goes to zero when $f(X; W) = g(X)$

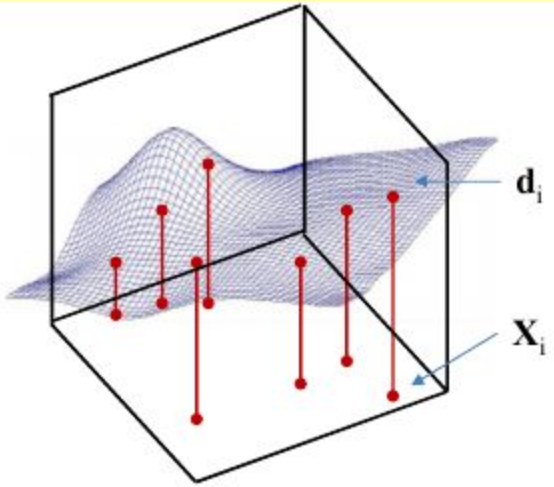
Problem $g(X)$ is unknown



- Function $g(X)$ must be fully specified
  - Known *everywhere,* i.e. for *every* input X
- **In practice we will not have such specification**

Learning the function



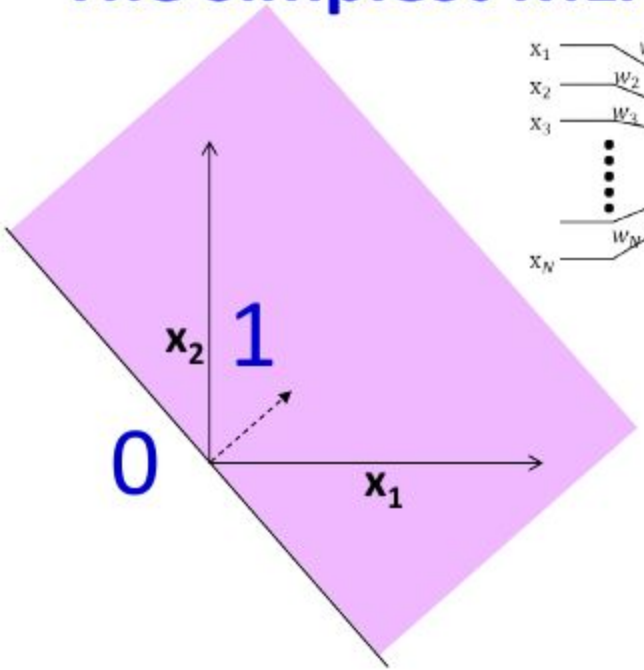


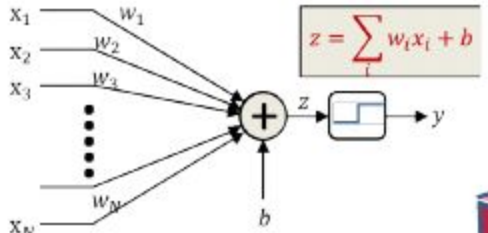
- Estimate the network parameters to "fit" the training points exactly
  - Assuming network architecture is sufficient for such a fit
  - Assuming unique output d at any $\mathbf{X}$
    - And hopefully the function is also correct where we *don't* have training samples

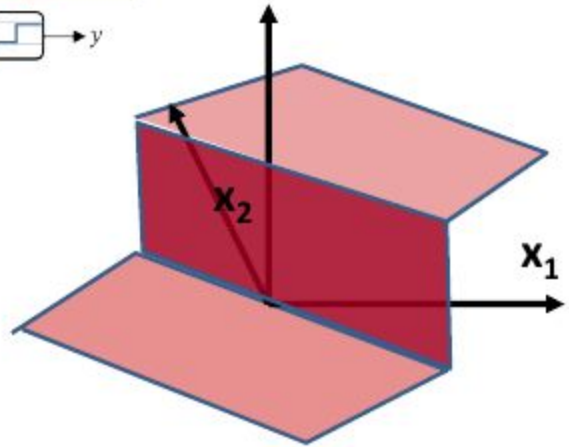
Lets begin with a simple task

- Learning a *classifier*
  - Simpler than regressions
- This was among the earliest problems addressed using MLPs
- Specifically, consider *binary* classification
  - Generalizes to multi-class

The simplest MLP: a single perceptron

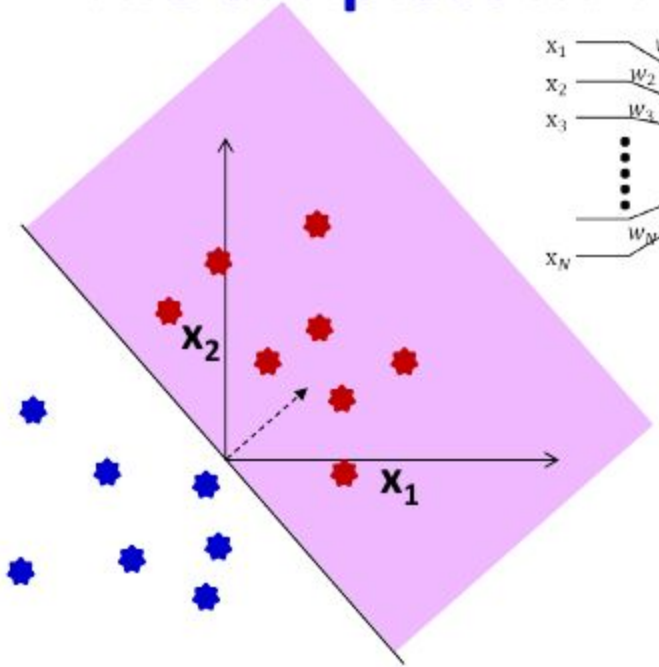


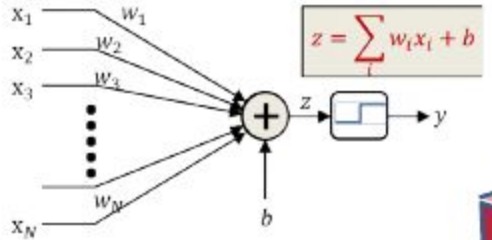


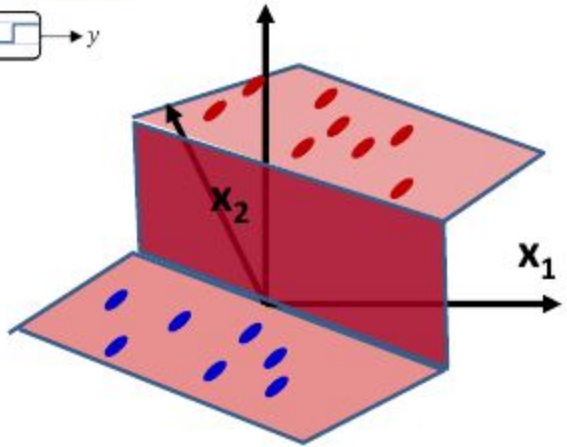


- Learn this function
  - **A step function across a hyperplane**

The simplest MLP: a single perceptron

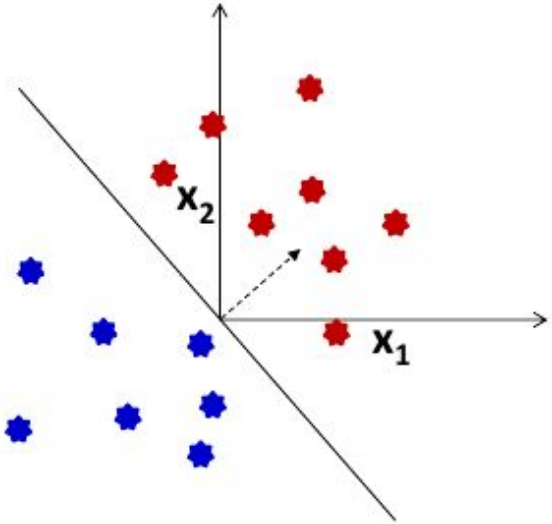


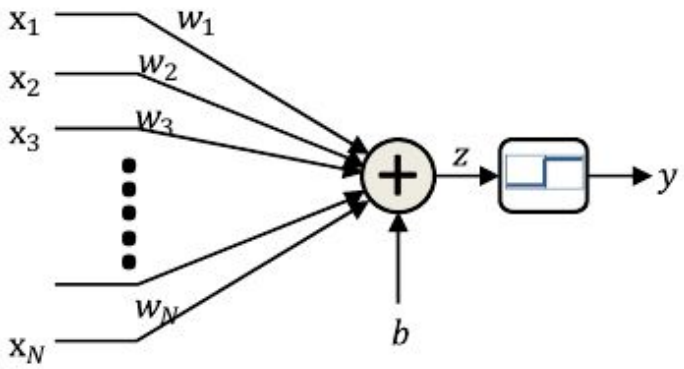




- Learn this function
  - **A step function across a hyperplane**
  - **Given only samples form it**

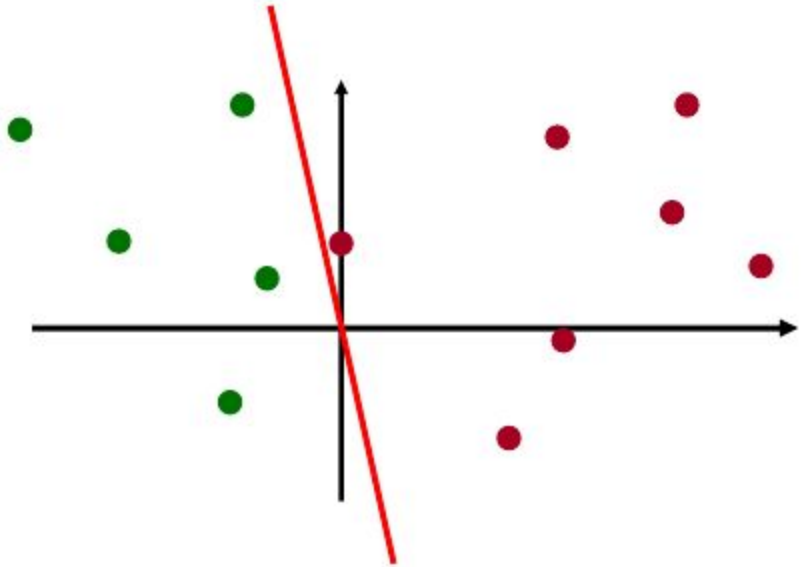
Learning the perceptron





- Given a number of input output pairs, learn the weights and bias
  - $y = \max(0, \sum_i w_i x_i + b)$
  - Learn $W = [w_1 .. w_N]$ and b, given several (X, y) pairs

The Perceptron Problem



- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points

Perceptron Learning Algorithm

- Given N training instances $(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)$
  - $Y_i = +1$ or -1

- Initialize W
- Cycle through the training instances:
- While more classification errors
  - For $i = 1 \ldots N_{train}$

$$O(X_i) = sign(W^T X_i)$$

    - If $O(X_i) \neq Y_i$

$$W = W + Y_i X_i$$

Perceptron Algorithm: Summary

- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class

$$W = W + X_i$$

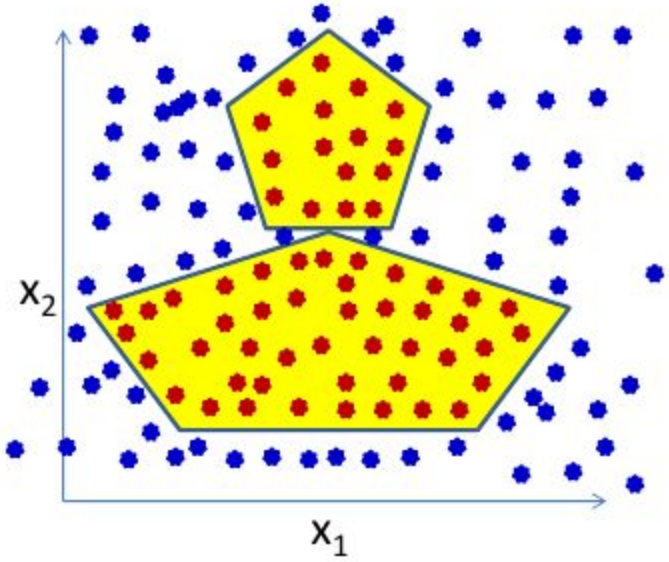
  - If instance is negative class

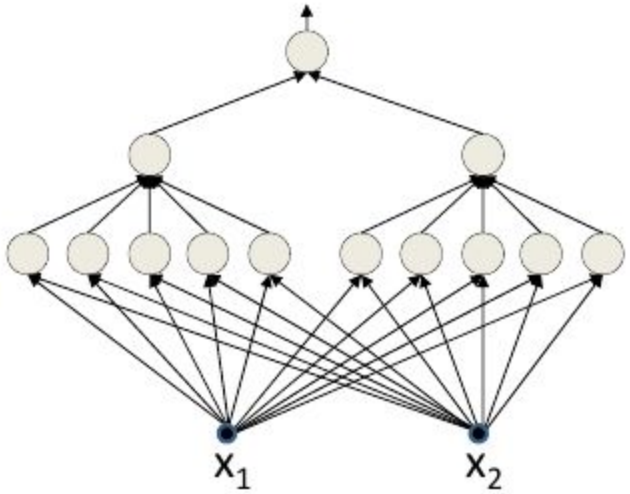
$$W = W - X_i$$

Convergence of Perceptron Algorithm

- Guaranteed to converge if classes are linearly separable
  - After no more than $\left(\frac{R}{\gamma}\right)^2$ misclassifications
    - Specifically when W is initialized to 0
  - R is length of longest training point
  - $\gamma$is the *best case* closest distance of a training point from the classifier
    - Same as the margin in an SVM
  - Intuitively – takes many increments of size γ to undo an error resulting from a step of size R

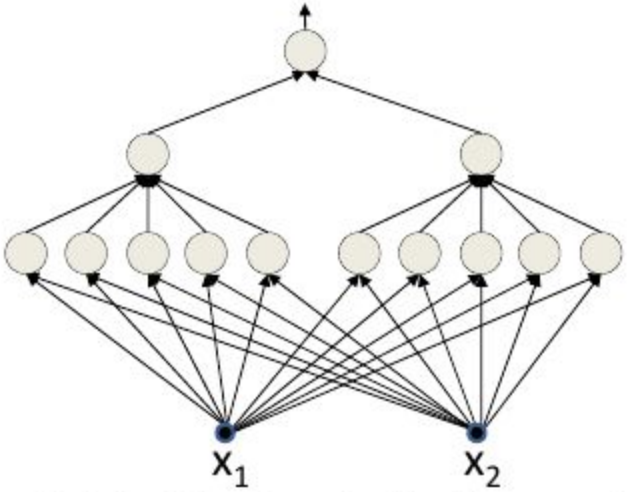
More complex decision boundaries





- Even using the perfect architecture
- Can we use the perceptron algorithm?

Learning a *multilayer* perceptron

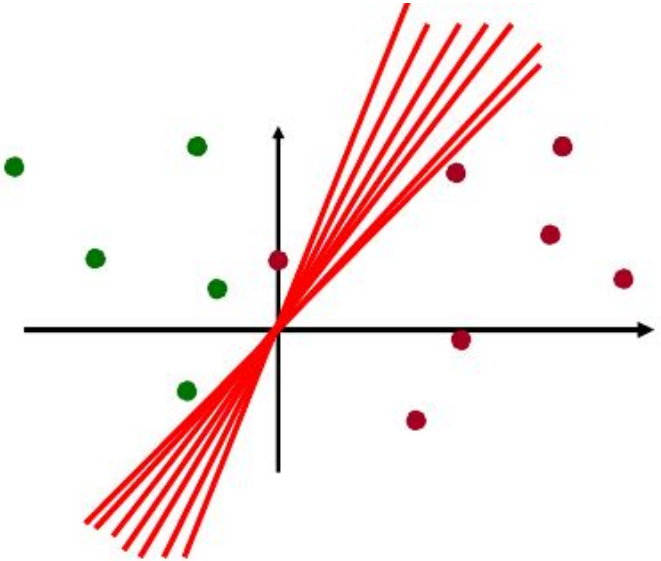


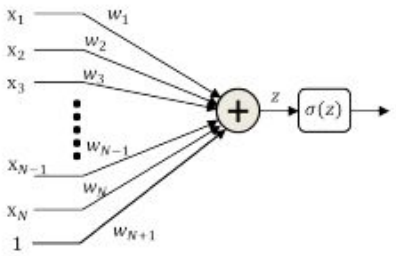
Training data only specifies input and output of network

Intermediate outputs (outputs of individual neurons) are not specified

- Training this network using the perceptron rule is a combinatorial optimization problems
- We don't know the outputs of the individual intermediate neurons in the network for any training input
- **Must also determine the correct output for *each* neuron for *every* training instance**
- **NP! Exponential complexity**

Issue 1

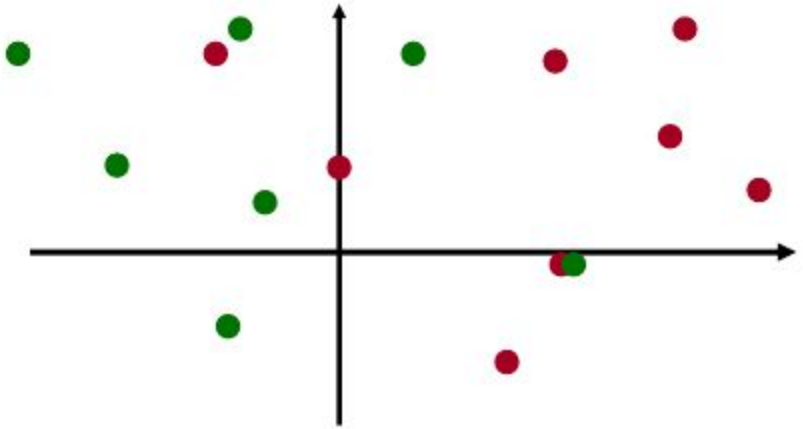




$\sigma(z) =$ 

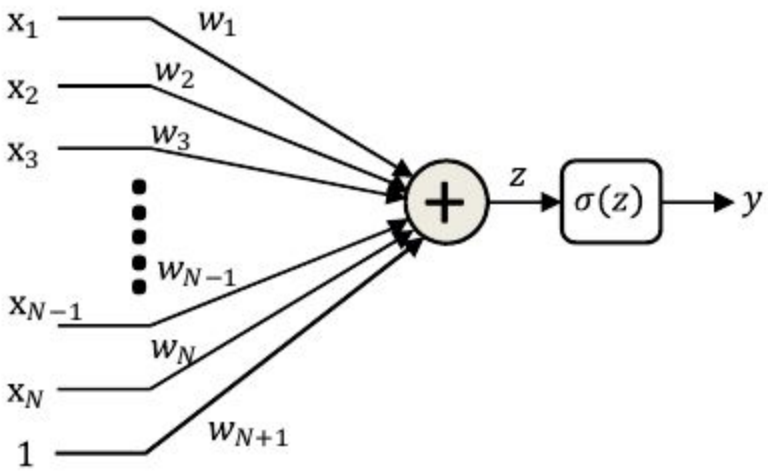
- The perceptron is a flat function with zero derivative everywhere, except at 0 where it is non-differentiable
  - You can vary the weights a *lot* without changing the error
  - There is no indication of which direction to change the weights to reduce error

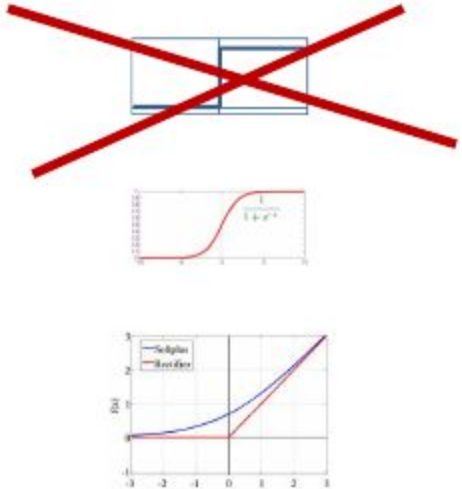
A second problem: What we *actually* model



- Real-life data are rarely clean
  - Not linearly separable

Solution

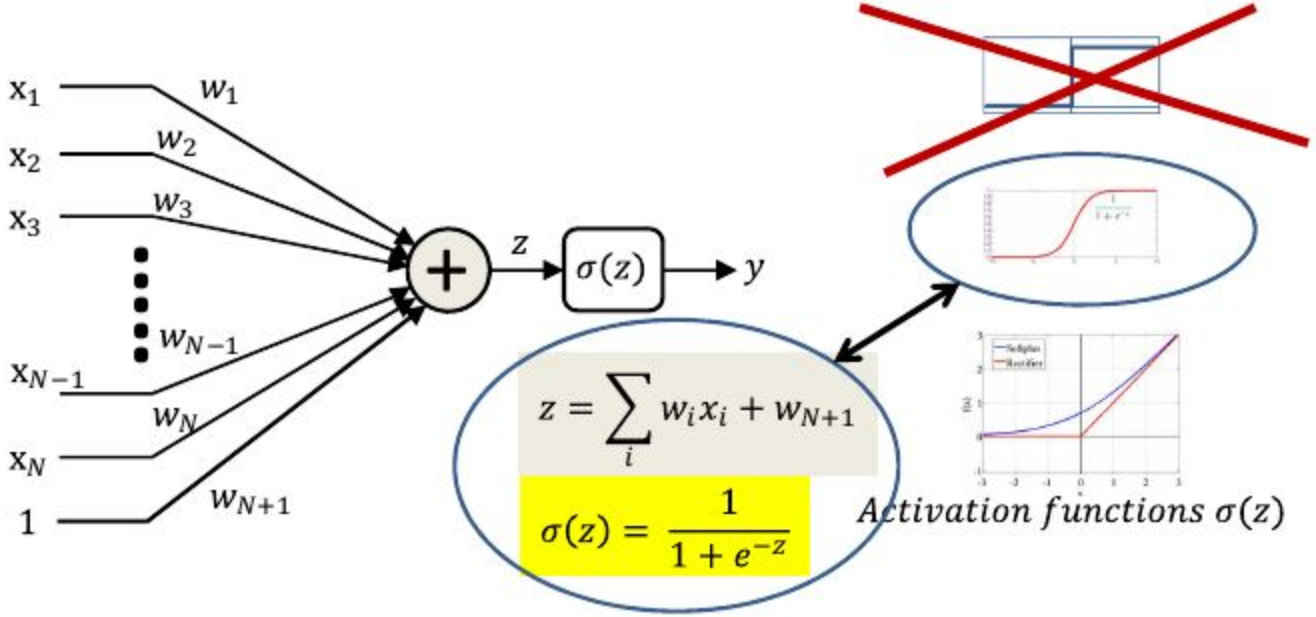




Activation functions $\sigma(z)$

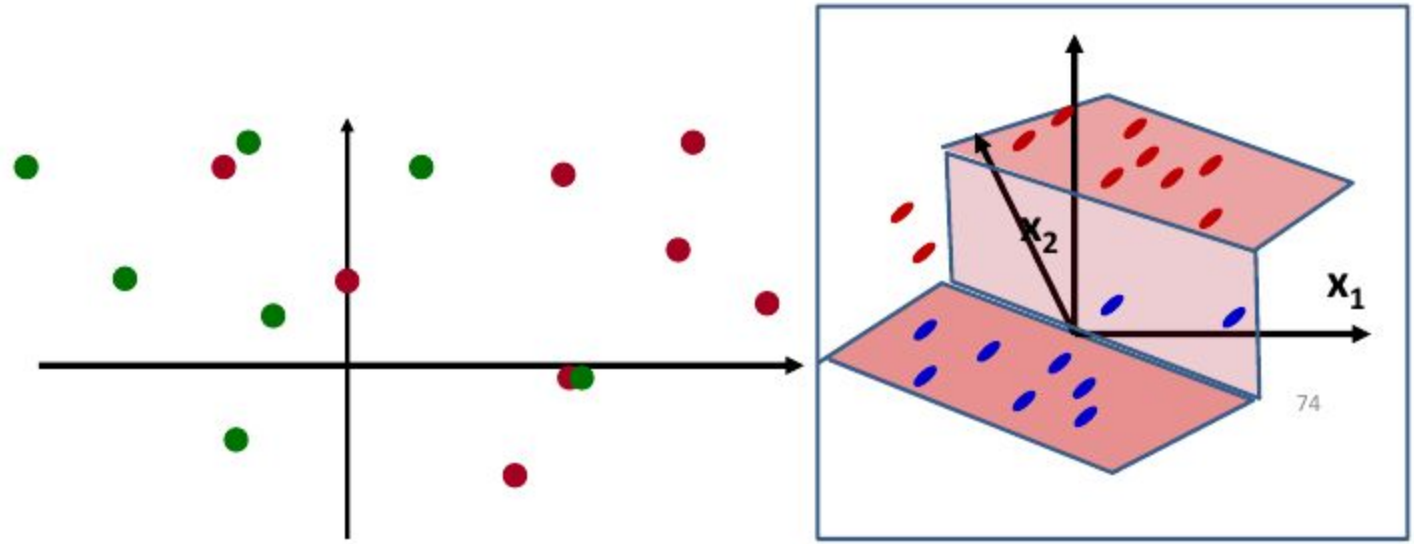
- Lets make the neuron differentiable
  - Small changes in weight can result in non-negligible changes in output
  - This enables us to estimate the parameters using gradient descent techniques..

Differentiable Activations: An aside



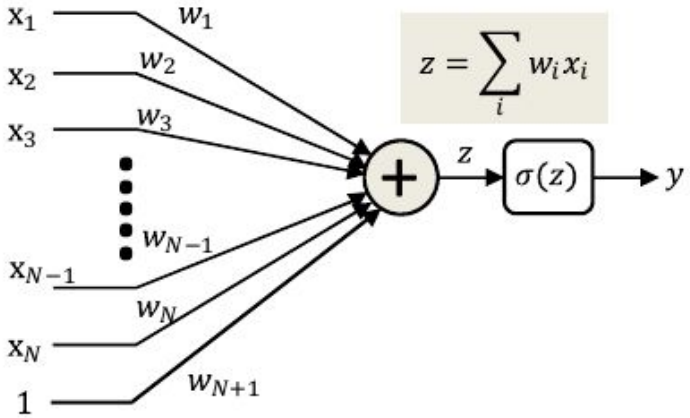
- This particular one has a nice interpretation

Non-linearly separable data



- Two-dimensional example
  - Blue dots (on the floor) on the “red” side
  - Red dots (suspended at Y=1) on the “blue” side
  - No line will cleanly separate the two colors

Perceptrons with differentiable activation functions



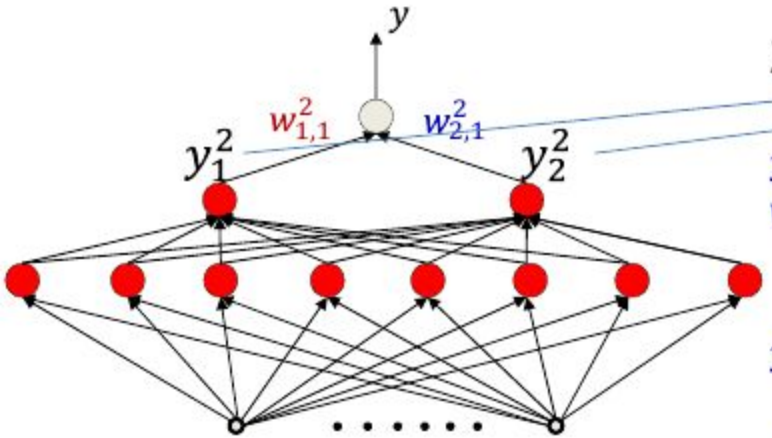
$$z = \sum_i w_i x_i$$

$$\frac{dy}{dw_i} = \frac{dy}{dz}\frac{dz}{dw_i} = \sigma'(z)x_i$$

$$\frac{dy}{dx_i} = \frac{dy}{dz}\frac{dz}{dx_i} = \sigma'(z)w_i$$

- $\sigma(z)$ is a differentiable function of z
  - $\frac{d\sigma(z)}{dz}$ is well-defined and finite for all z
- Using the chain rule, y is a differentiable function of both inputs x_i and weights w_i
- This means that we can compute the change in the output for *small* changes in either the input or the weights

Overall network is differentiable



$$y = \sigma(w^2_{1,1}y^2_1 + w^2_{2,1}y^2_2 + w^2_{3,1})$$

y = output of overall network

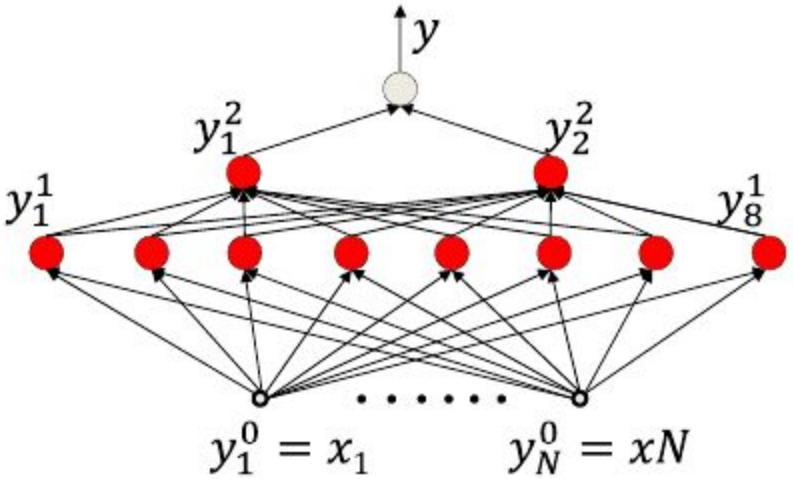
$w^k_{i,j}$ = weight connecting the ith unit of the kth layer to the jth unit of the k+1-th layer

y^k_i = output of the ith unit of the kth layer

$\sigma()$ is differentiable w.r.t both w and y^k_i

- Every individual perceptron is differentiable w.r.t its inputs and its weights (including "bias" weight)
- By the chain rule, the overall function is differentiable w.r.t every parameter (weight or bias)
  - Small changes in the parameters result in measurable changes in output

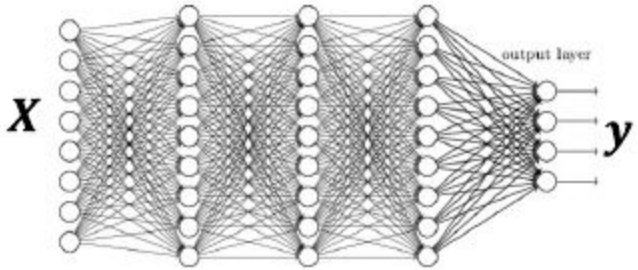
Overall function is differentiable

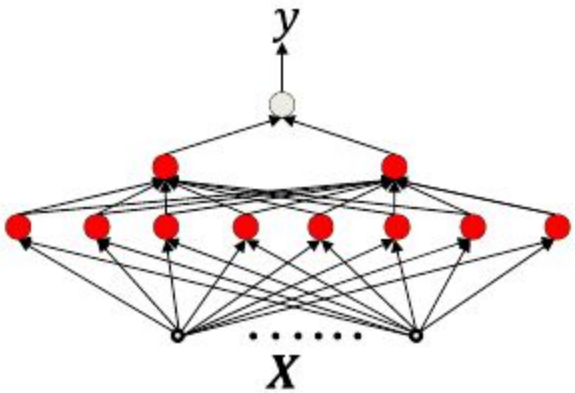


$$y_j^k = \sigma\left(\sum_i w_{i,j}^{k-1} y_i^{k-1}\right)$$

- The overall function is differentiable w.r.t every parameter
  - Small changes in the parameters result in measurable changes in the output
  - We will derive the actual derivatives using the chain rule later

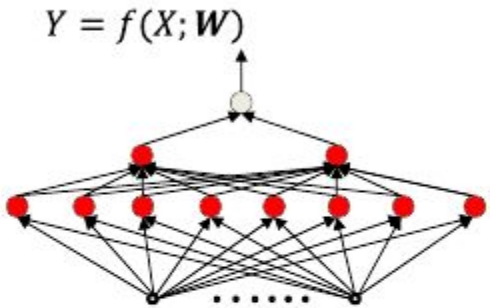
Overall setting for "Learning" the MLP

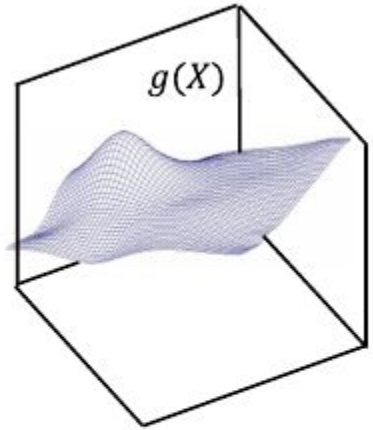




- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_N, \boldsymbol{d}_N)$ …
  - $\boldsymbol{d}$ is the *desired output* of the network in response to $\boldsymbol{X}$
  - $\boldsymbol{X}$ and $\boldsymbol{d}$ may both be vectors
- …we must find the network parameters such that the network produces the desired output for each training input
  - Or a close approximation of it
  - **The *architecture* of the network must be specified by us**

Recap: Learning the function



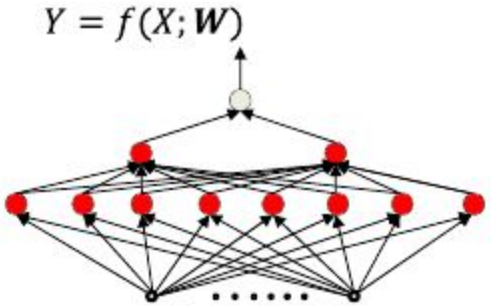


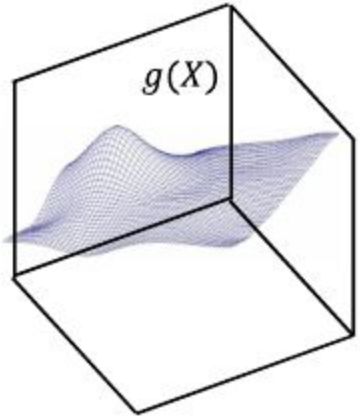
- When $f(X; \boldsymbol{W})$ has the capacity to exactly represent $g(X)$

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}} \int_X div\big(f(X; W), g(X)\big) dX$$

- div() is a divergence function that goes to zero when $f(X; W) = g(X)$

Minimizing *expected* error



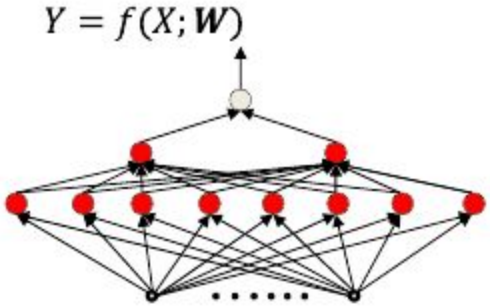


- More generally, assuming X is a random variable

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}} \int_X div\big(f(X;W), g(X)\big) P(X) dX$$

$$= \underset{W}{\operatorname{argmin}}\, E\big[div\big(f(X;W), g(X)\big)\big]$$

Empirical Risk Minimization



- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_N, \boldsymbol{d}_N)$
  - Error on the ith instance: $div(f(X_i; W), d_i)$
  - Empirical average error on all training data:

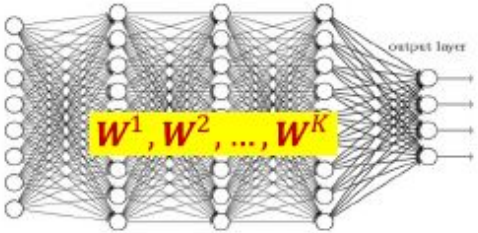
$$Err(W) = \frac{1}{N} \sum_i div(f(X_i; W), d_i)$$

- Estimate the parameters to minimize the empirical estimate of expected error

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}}\, Err(W)$$

  - I.e. minimize the *empirical error* over the drawn samples

ERM for neural networks



Actual output of network:

$$Y_i = net\left(X_i; \{w_{i,j}^k \forall i,j,k\}\right)$$
$$= net(X_i; W^1, W^2, \ldots, W^K)$$

Desired output of network: d_i

Error on i-th training input: $Div(Y_i, d_i; W^1, W^2, \ldots, W^K)$

Total training error:

$$Err(W^1, W^2, \ldots, W^K) = \frac{1}{N}\sum_{i=1}^{N} Div(Y_i, d_i; W^1, W^2, \ldots, W^K)$$

- What is the exact form of Div()? More on this later

- Optimize network parameters to minimize the total error over all training inputs

Problem Statement

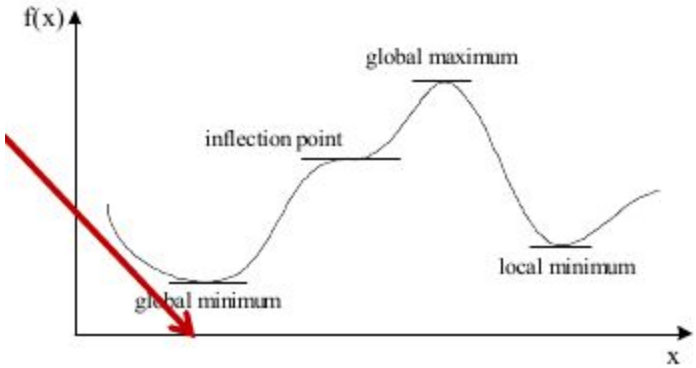
- Given a training set of input-output pairs $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_N, \boldsymbol{d}_N)$
- Minimize the following function

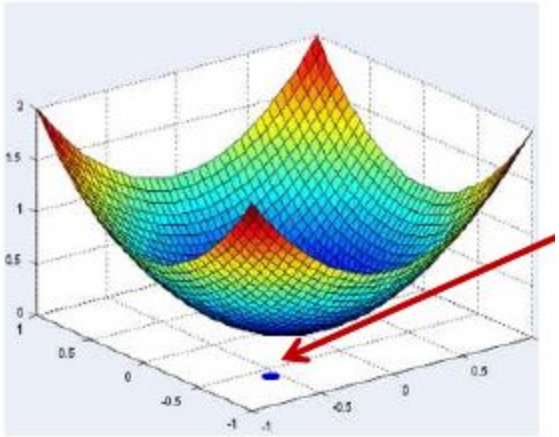
$$Err(W) = \frac{1}{N} \sum_i div(f(X_i; W), d_i)$$

w.r.t W

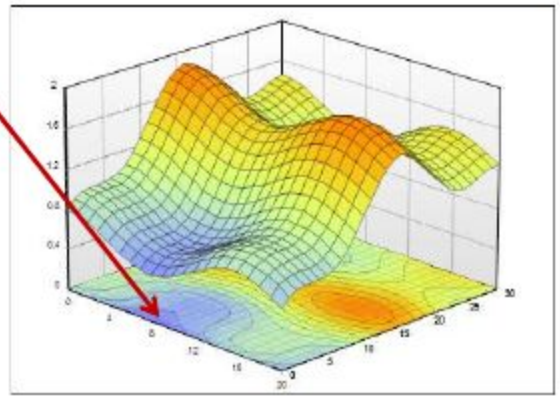
- This is problem of function minimization
  - An instance of optimization

The problem of optimization

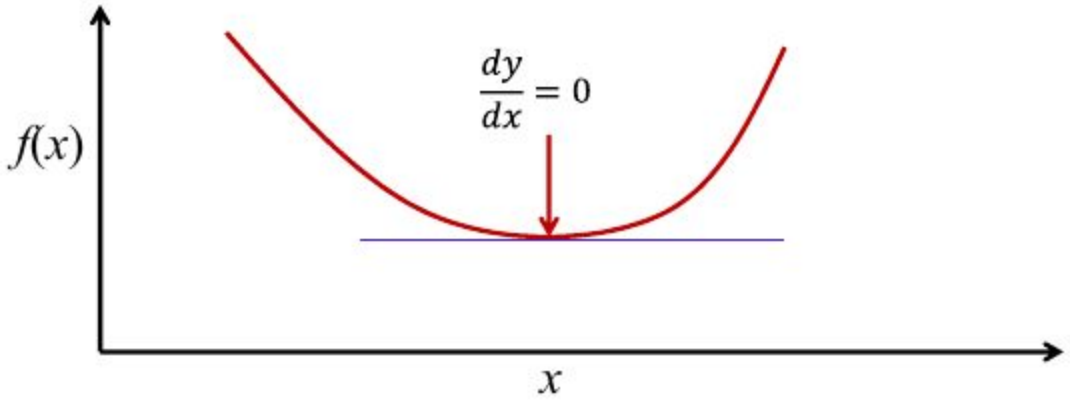




- General problem of optimization: find the value of x where $\mathbf{f}(x)$ is minimum



Finding the minimum of a function

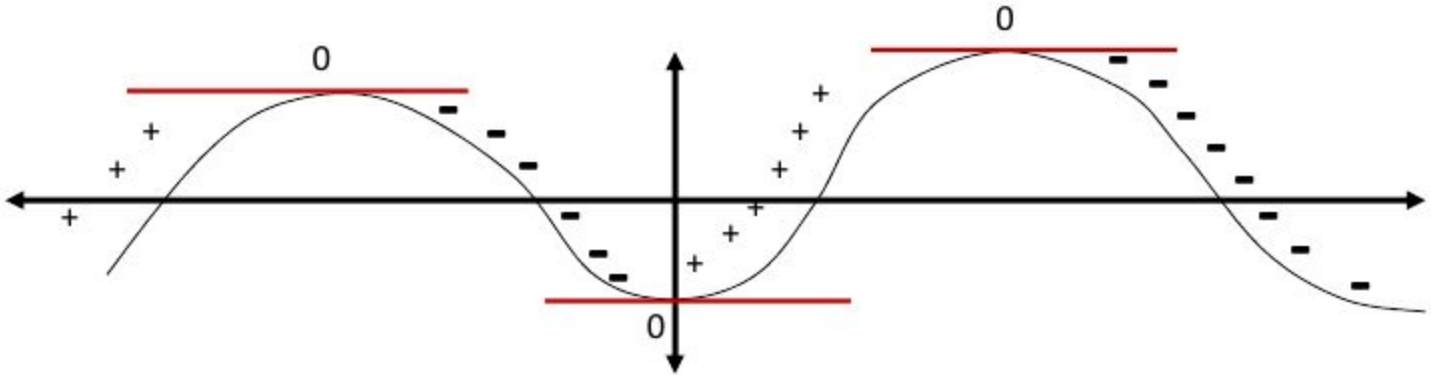


- Find the value x at which $f'(x) = 0$
  - Solve

$$\frac{df(x)}{dx} = 0$$

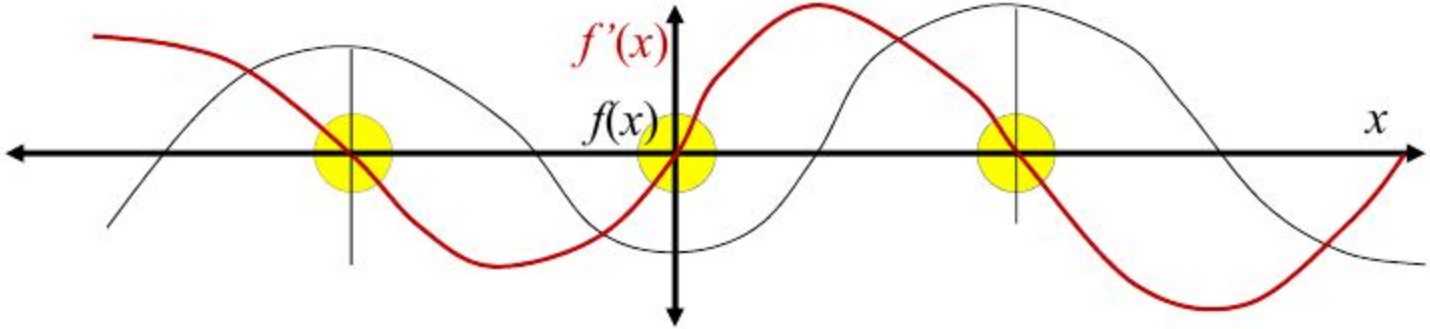
- The solution is a "turning point"
  - Derivatives go from positive to negative or vice versa at this point
- But is it a minimum?

Turning Points



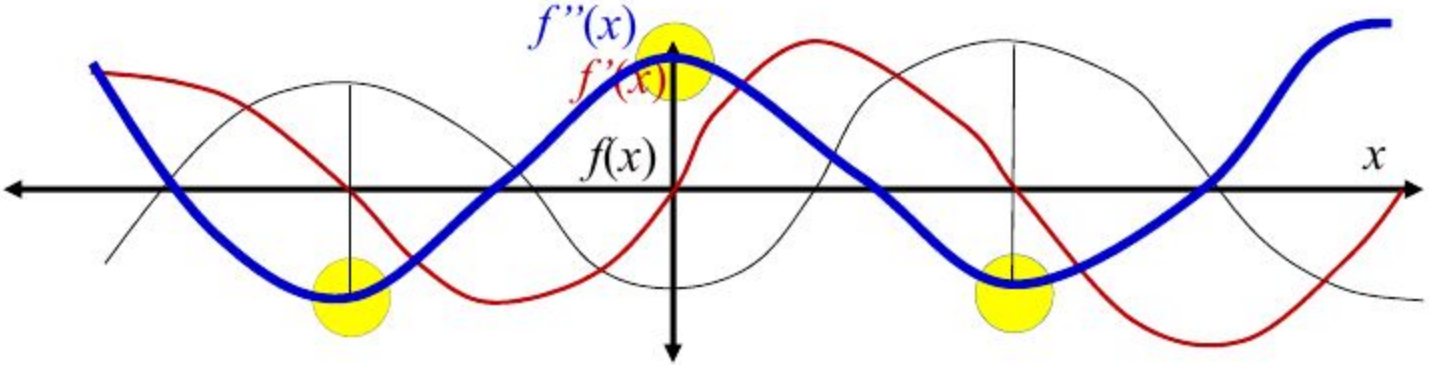
- Both *maxima* and *minima* have zero derivative
- Both are turning points

Derivatives of a curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

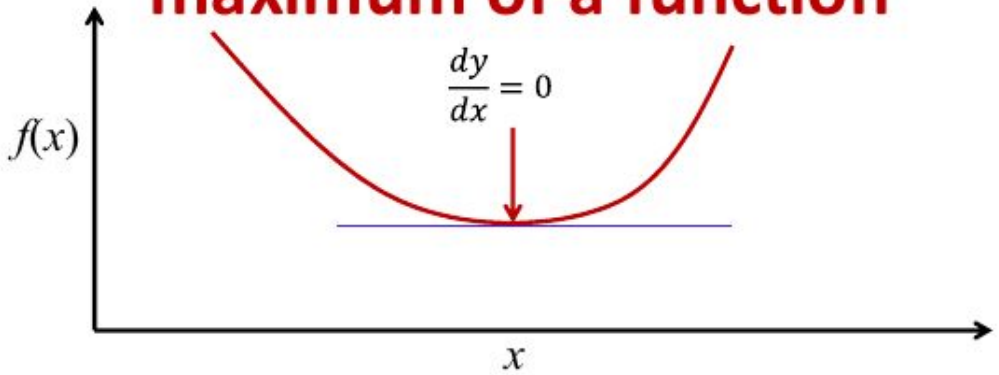
Derivative of the derivative of the curve



- Both *maxima* and *minima* are turning points
- Both *maxima* and *minima* have zero derivative

- The *second derivative f''*(*x*) is –ve at maxima and +ve at minima!

Soln: Finding the minimum or maximum of a function



- Find the value x at which $f'(x) = 0$: Solve

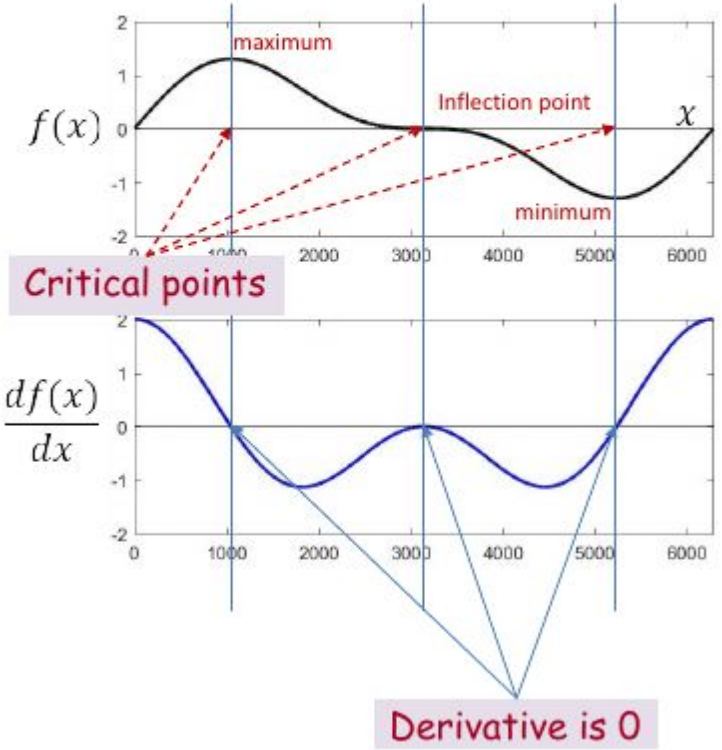
$$\frac{df(x)}{dx} = 0$$

- The solution x_{soln} is a turning point
- Check the double derivative at x_{soln} : compute

$$f''(x_{soln}) = \frac{df'(x_{soln})}{dx}$$

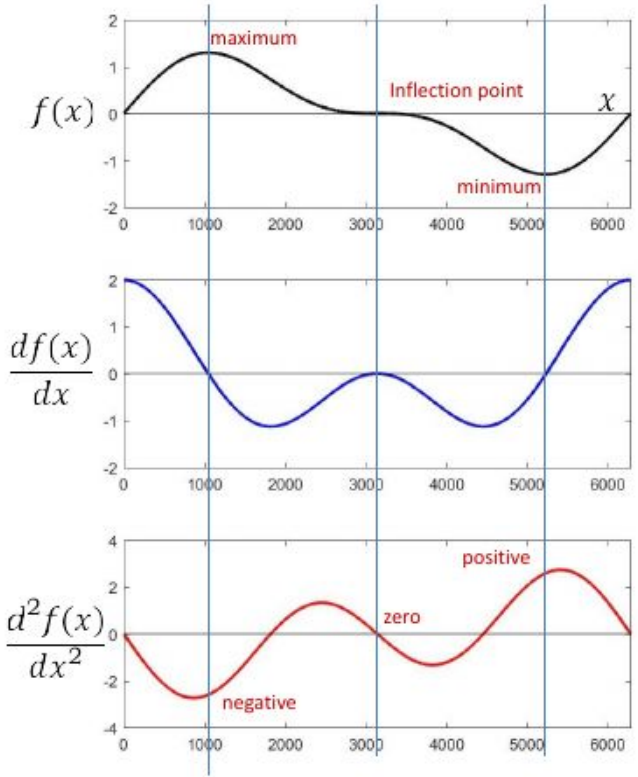
- If $f''(x_{soln})$ is positive x_{soln} is a minimum, otherwise it is a maximum

A note on derivatives of functions of single variable



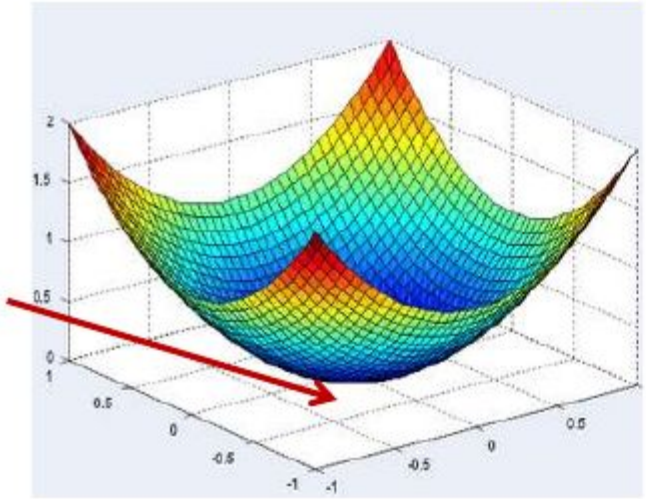
- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points

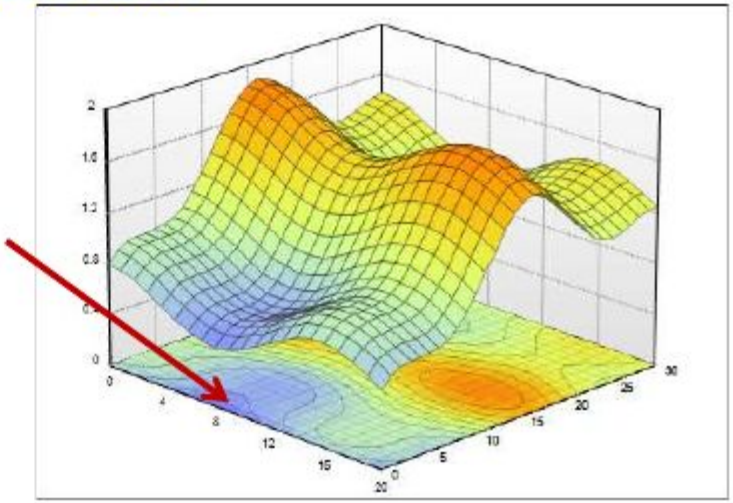
A note on derivatives of functions of single variable



- All locations with zero derivative are *critical* points
  - These can be local maxima, local minima, or inflection points
- The *second* derivative is
  - ≥ 0 at minima
  - ≤ 0 at maxima
  - Zero at inflection points
- It's a little more complicated for functions of multiple variables..

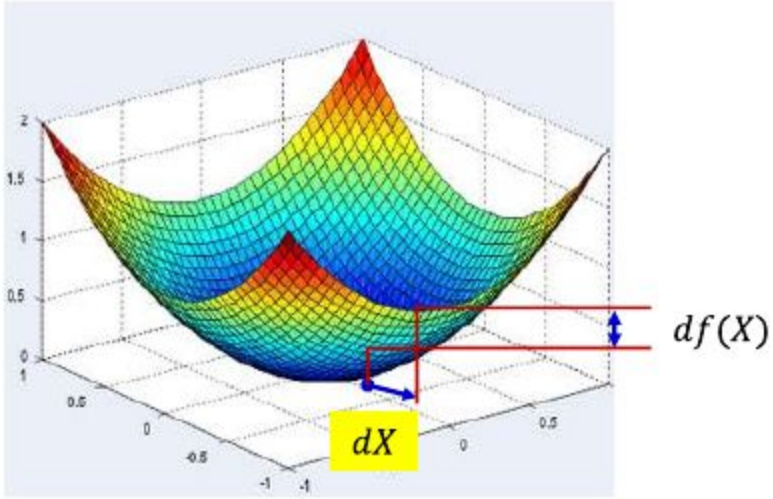
What about functions of multiple variables?





- The optimum point is still "turning" point
  - Shifting in any direction will increase the value
  - For smooth functions, miniscule shifts will not result in any change at all
- We must find a point where shifting in any direction by a microscopic amount will not change the value of the function

The *Gradient* of a scalar function

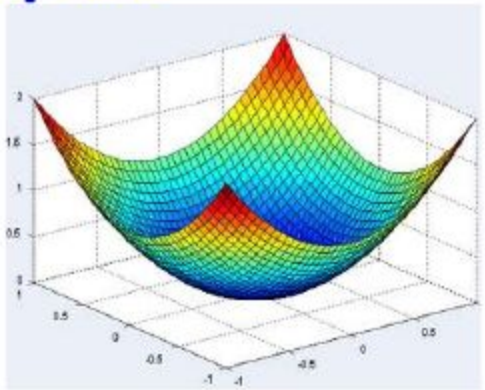


- The *Gradient* $\nabla f(X)$ of a scalar function $f(X)$ of a multi-variate input X is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in X

$$df(X) = \nabla f(X) dX$$

Gradients of scalar functions with multi-variate inputs

- Consider $f(X) = f(x_1, x_2, \ldots, x_n)$



$$\nabla f(X) = \left[\frac{\partial f(X)}{\partial x_1} \quad \frac{\partial f(X)}{\partial x_2} \quad \cdots \quad \frac{\partial f(X)}{\partial x_n}\right]$$

- Check:

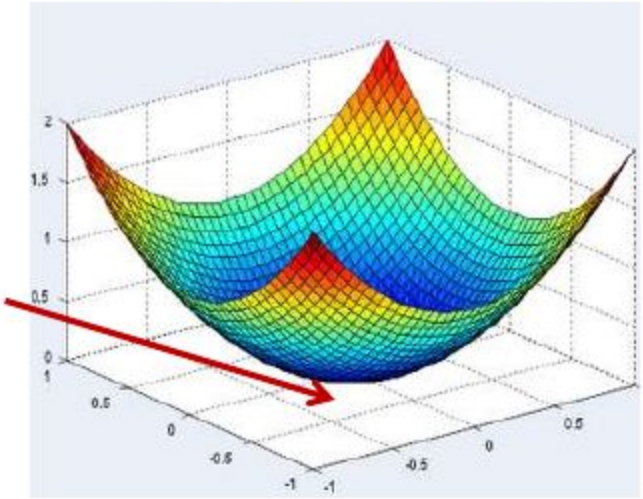
$$df(X) = \nabla f(X) dX$$
$$= \frac{\partial f(X)}{\partial x_1} dx_1 + \frac{\partial f(X)}{\partial x_2} dx_2 + \cdots + \frac{\partial f(X)}{\partial x_n} dx_n$$

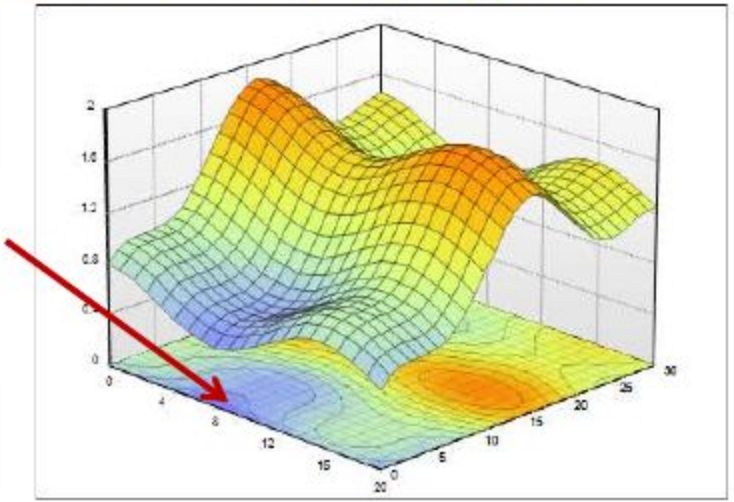
The Hessian

- The Hessian of a function $f(x_1, x_2, \ldots, x_n)$ is given by the second derivative

$$\nabla^2 f(x_1,\ldots,x_n) := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Finding the minimum of a scalar function of a multi-variate input





- The optimum point is a turning point – the gradient will be 0

Unconstrained Minimization of function (Multivariate)

1. Solve for the X where the gradient equation equals to zero

$$\nabla f(X) = 0$$

2. Compute the Hessian Matrix $\nabla^2 f(X)$ at the candidate solution and verify that
   - Hessian is positive definite (eigenvalues positive) -> to identify local minima
   - Hessian is negative definite (eigenvalues negative) -> to identify local maxima

Unconstrained Minimization of function (Example)

- Minimize

$$f(x_1, x_2, x_3) = (x_1)^2 + x_1(1 - x_2) - (x_2)^2 - x_2 x_3 + (x_3)^2 + x_3$$

- Gradient

$$\nabla f = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}^T$$

Unconstrained Minimization of function (Example)

- Set the gradient to null

$$\nabla f = 0 \Rightarrow \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Solving the 3 equations system with 3 unknowns

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

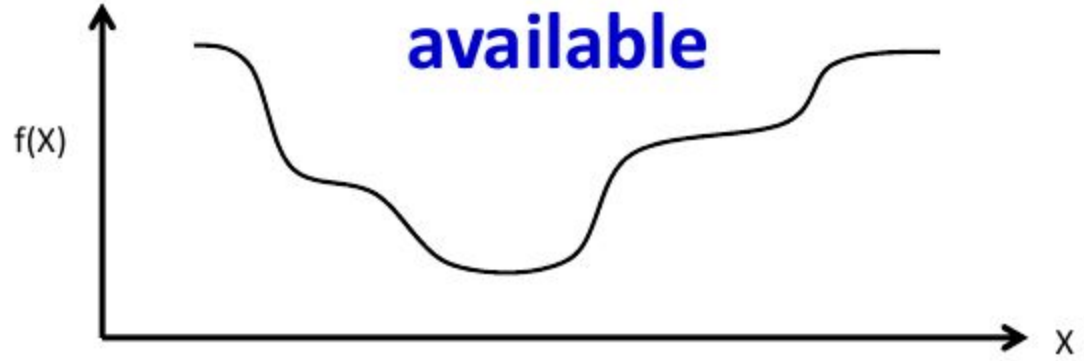
Unconstrained Minimization of function (Example)

- Compute the Hessian matrix $\nabla^2 f = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$
- Evaluate the eigenvalues of the Hessian matrix

$$\lambda_1 = 3.414, \quad \lambda_2 = 0.586, \quad \lambda_3 = 2$$

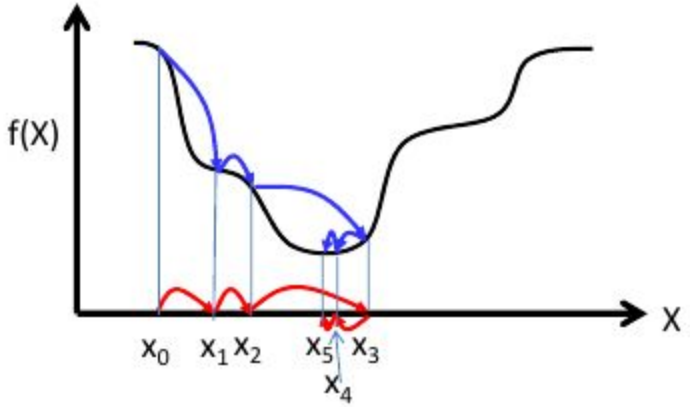
- All the eigenvalues are positives => the Hessian matrix is positive definite
- The point $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$ is a minimum

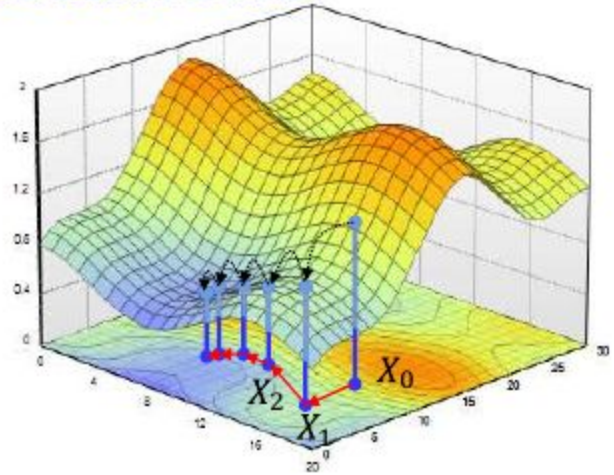
Closed Form Solutions are not always available



- Often it is not possible to simply solve $\nabla f(X) = 0$
  - The function to minimize/maximize may have an intractable form
- In these situations, iterative solutions are used
  - Begin with a "guess" for the optimal X and refine it iteratively until the correct value is obtained

Iterative solutions



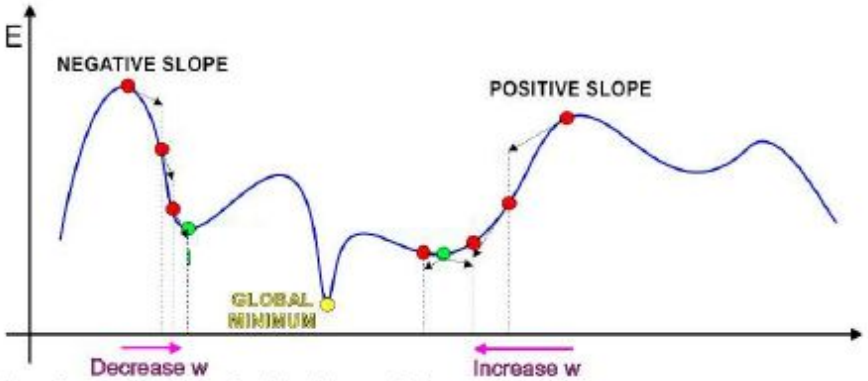


- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases
- Problems:
  - Which direction to step in
  - How big must the steps be

The Approach of Gradient Descent

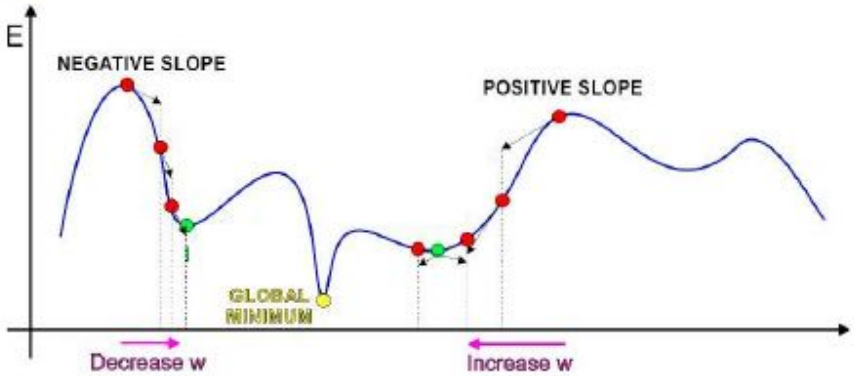
- Iterative solution:
  - Start at some point
  - Find direction in which to shift this point to decrease error
    - This can be found from the derivative of the function
      - A positive derivative → moving left decreases error
      - A negative derivative → moving right decreases error
  - Shift point in this direction

The Approach of Gradient Descent



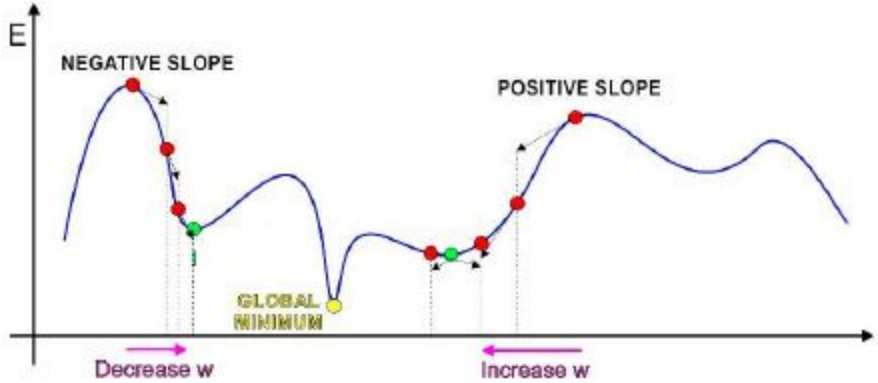
- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'(x^k) \neq 0$
    - If $sign\left(f'(x^k)\right)$ is positive:
      - $x^{k+1} = x^k - step$
    - Else
      - $x^{k+1} = x^k + step$
  - What must step be to ensure we actually get to the optimum?

The Approach of Gradient Descent



- Iterative solution: Trivial algorithm
  - Initialize x^0
  - While $f'(x^k) \neq 0$
    - $x^{k+1} = x^k - sign\left(f'(x^k)\right).step$
  - Identical to previous algorithm

The Approach of Gradient Descent



- Iterative solution: Trivial algorithm
  - Initialize x_0
  - While $f'(x^k) \neq 0$
    - $x^{k+1} = x^k - \eta^k f'(x^k)$
  - η^k is the "step size"

Gradient descent/ascent (multivariate)

- The gradient descent/ascent method to find the minimum or maximum of a function f iteratively
  - To find a *maximum* move *in the direction of the gradient*

$$x^{k+1} = x^k + \eta^k \nabla f\left(x^k\right)^T$$

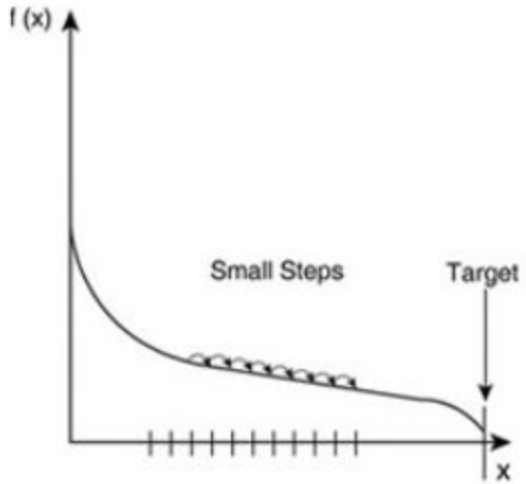
  - To find a *minimum* move *exactly opposite the direction of the gradient*

$$x^{k+1} = x^k - \eta^k \nabla f\left(x^k\right)^T$$

- Many solutions to choosing step size η^k

1. Fixed step size

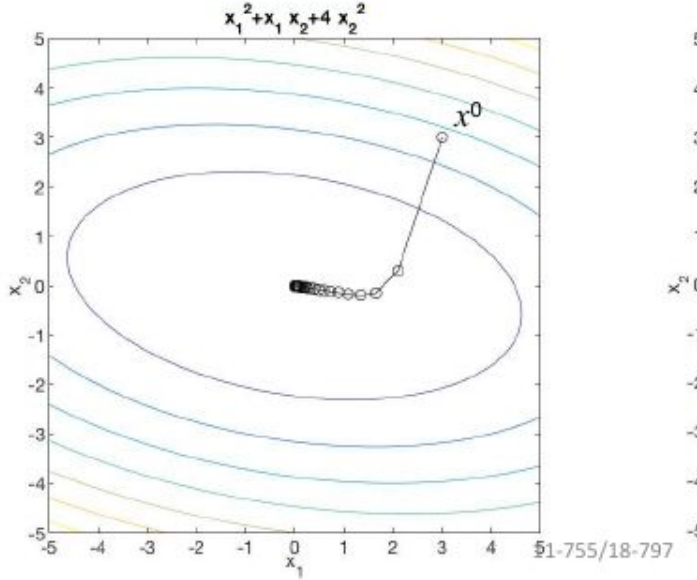
- Fixed step size
  - Use fixed value for η^k



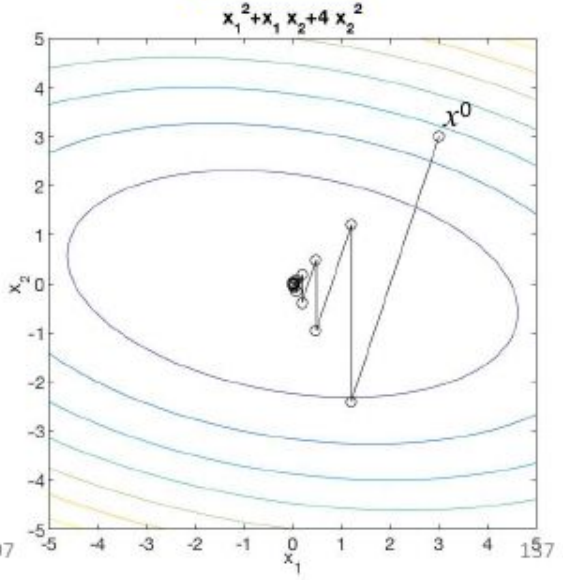
Influence of step size example (constant step size)

$$f(x_1, x_2) = (x_1)^2 + x_1 x_2 + 4(x_2)^2 \qquad x^{initial} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$\eta = 0.1$



$\eta = 0.2$

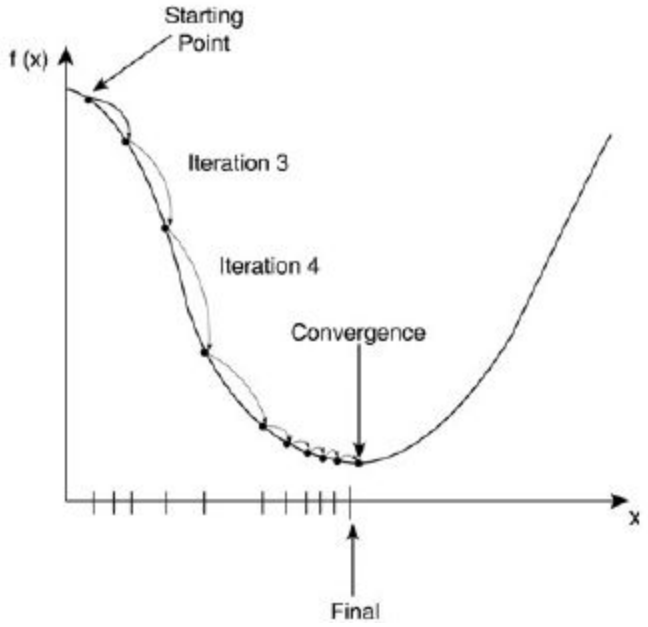


Gradient descent convergence criteria

- The gradient descent algorithm converges when one of the following criteria is satisfied

$$\left| f(x^{k+1}) - f(x^k) \right| < \varepsilon_1$$

- Or $\left\| \nabla f(x^k) \right\| < \varepsilon_2$



Overall Gradient Descent Algorithm

- Initialize:
  - x^0
  - $k = 0$
- While $\left|f\left(x^{k+1}\right) - f\left(x^k\right)\right| > \varepsilon$
  - $x^{k+1} = x^k - \eta^k \nabla f\left(x^k\right)^T$
  - $k = k + 1$

How do we compute gradients?

- Analytic or “Manual” Differentiation
  - Applying formulas from calculus
- Numerical Differentiation
- Automatic Differentiation
  - Computation Graph

Numerical vs Analytic Gradients

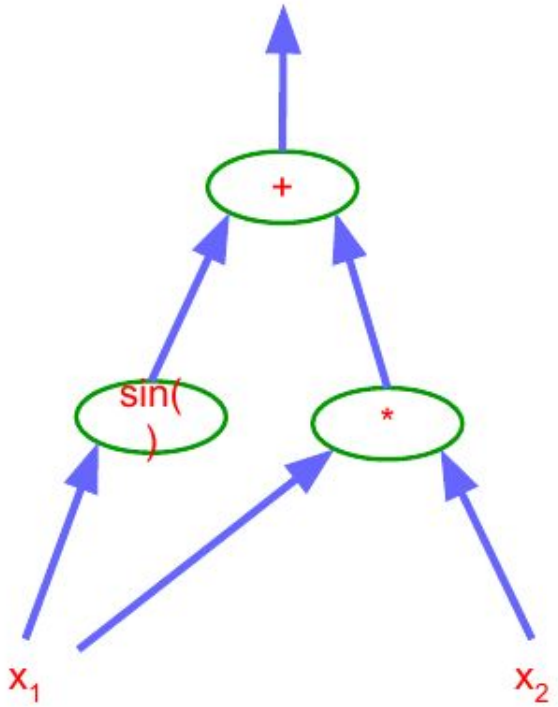
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.
This is called a **gradient check.**

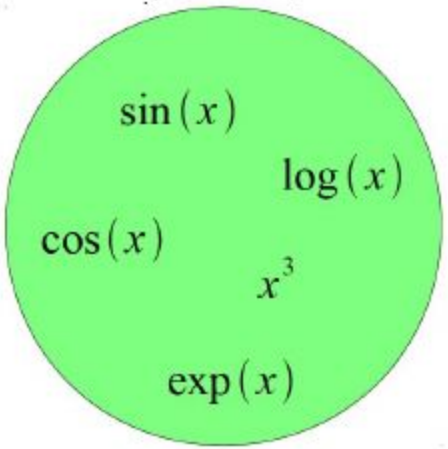
Computational Graph: Example

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Logistic Regression as a Cascade

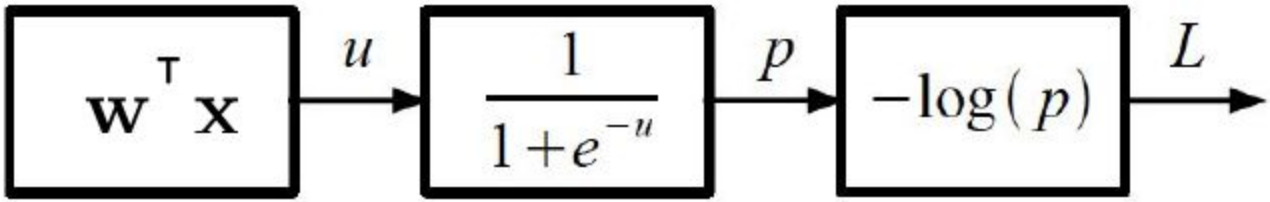
Given a library of simple functions



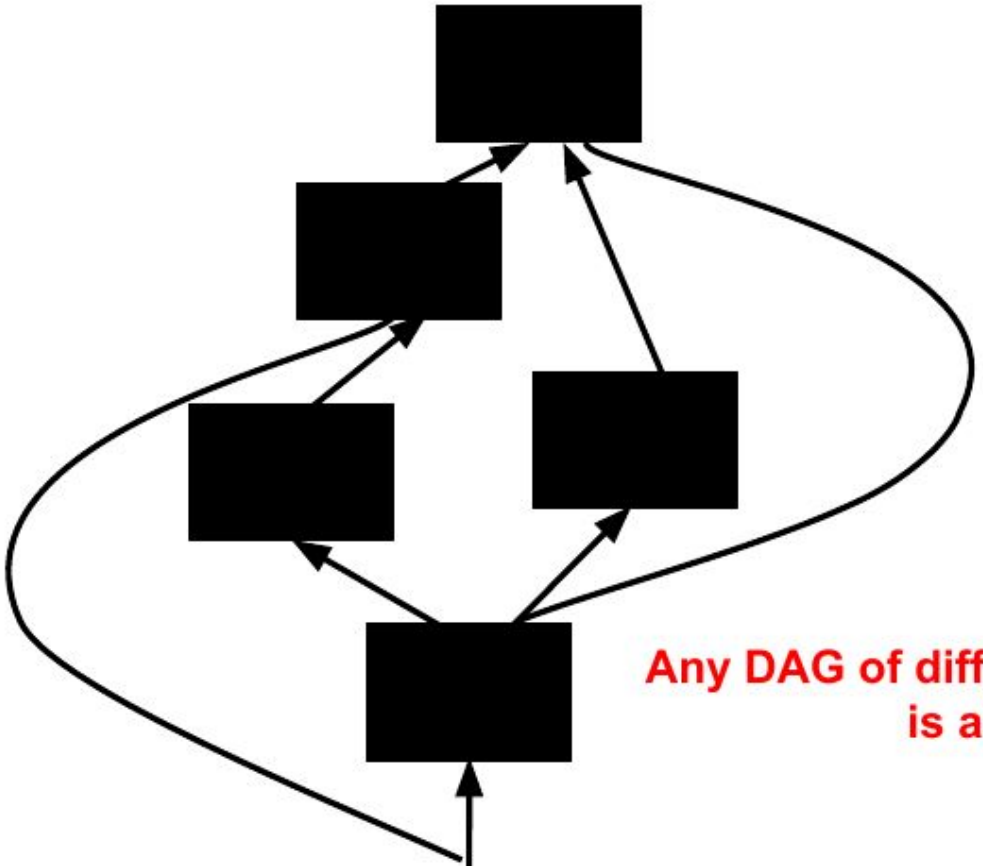
Compose into a

complicate function

$$-\log\left(\frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}\right)$$



Computational Graph



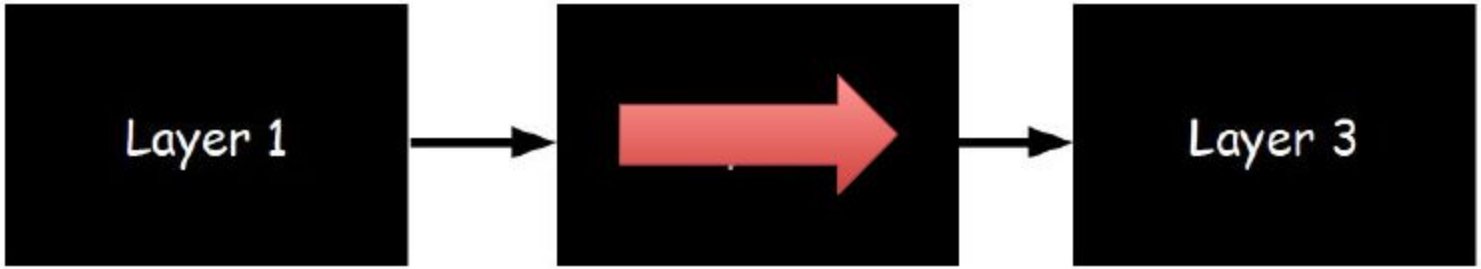
Any DAG of differentiable modules is allowed!

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]



Neural Network Training

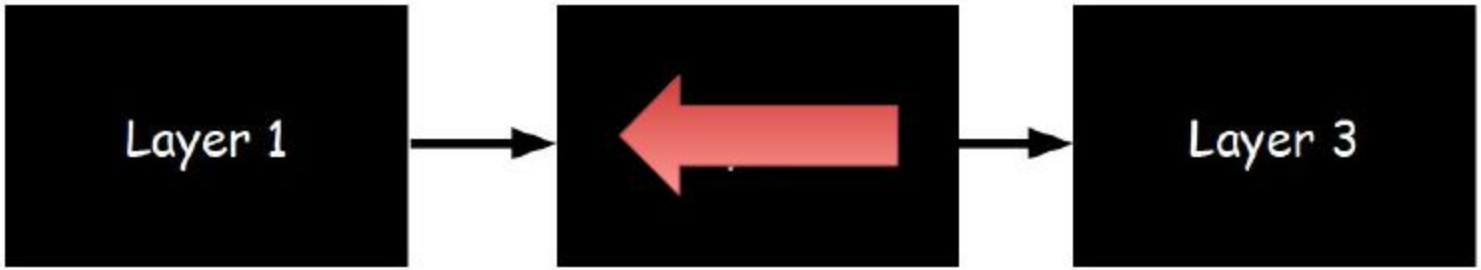
- Step 1: Compute Loss on mini-batch [F-Pass]

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients w.r.t. parameters [B-Pass]

Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients w.r.t. parameters [B-Pass]



Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients w.r.t. parameters [B-Pass]

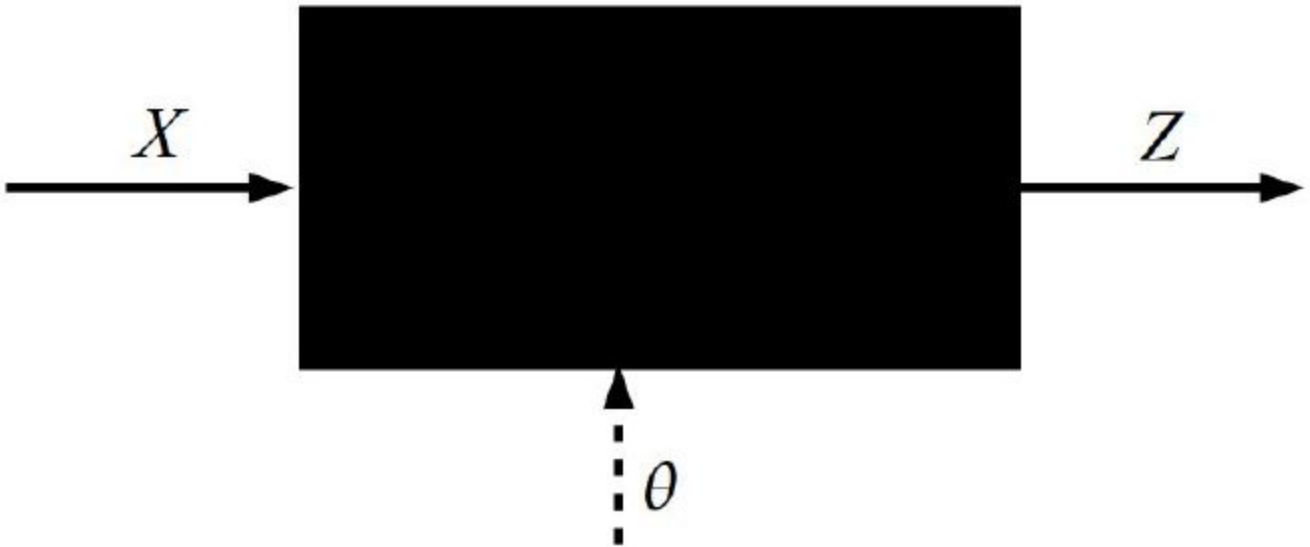
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients w.r.t. parameters [B-Pass]
- Step 3: Use gradient to update parameters

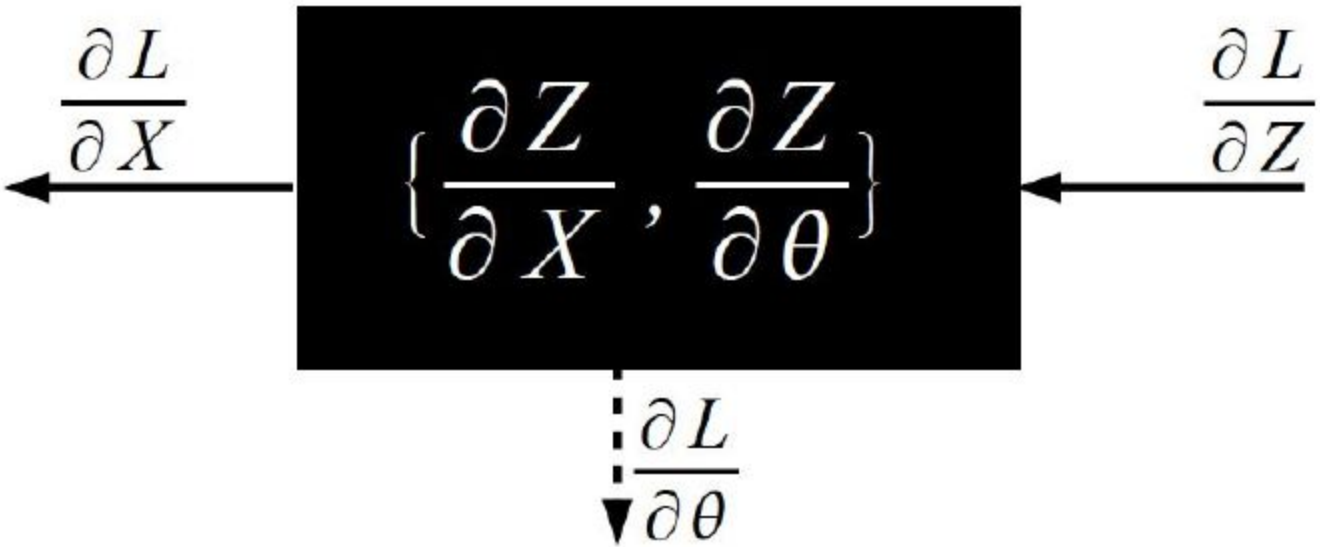
$$\theta \leftarrow \theta - \eta \frac{dL}{d\theta}$$

*** ɵ same as w

Key Computation: Forward-Prop



Key Computation: Back-Prop

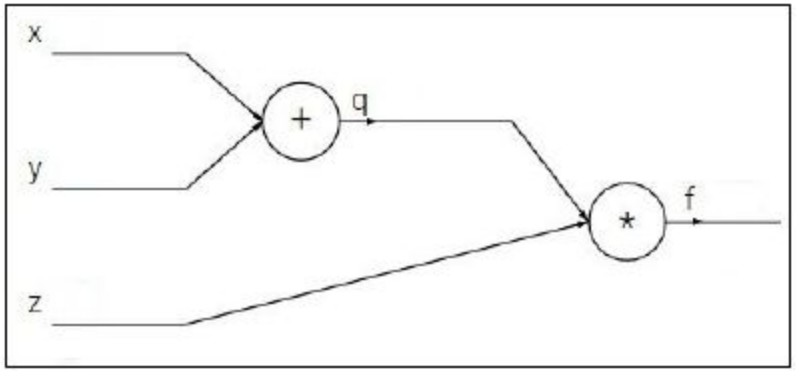


Backpropagation: a simple example

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

Backpropagation: a simple example

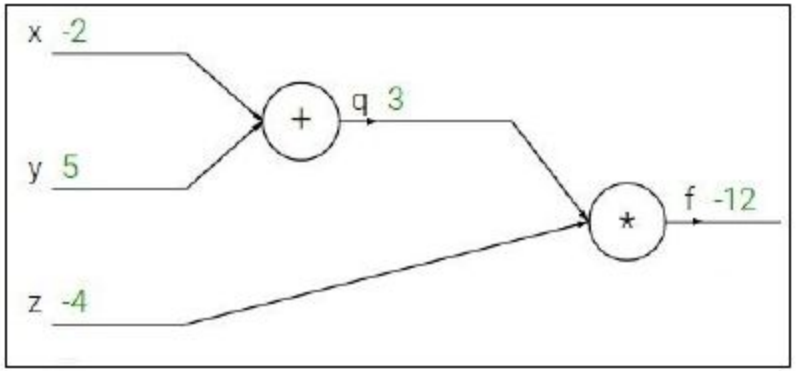
$$f(x, y, z) = (x + y)z$$



Backpropagation: a simple example

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

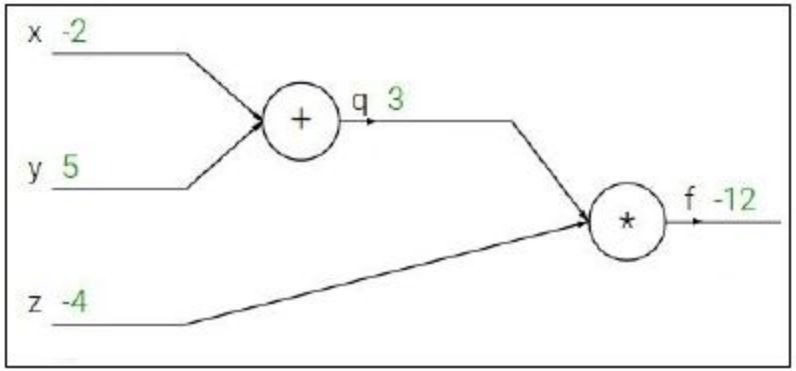
e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

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

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

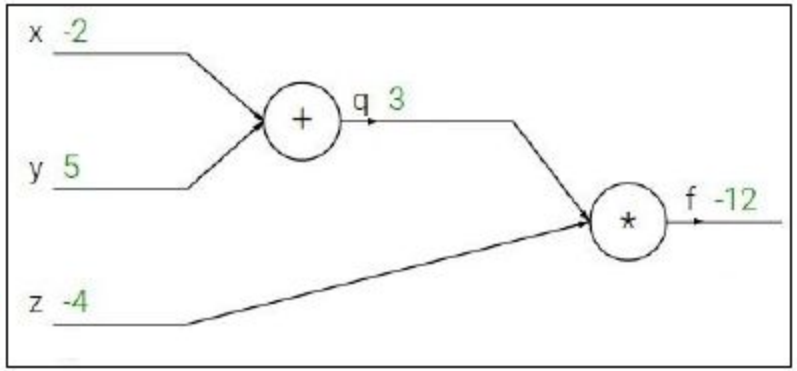


Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: a simple example

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

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



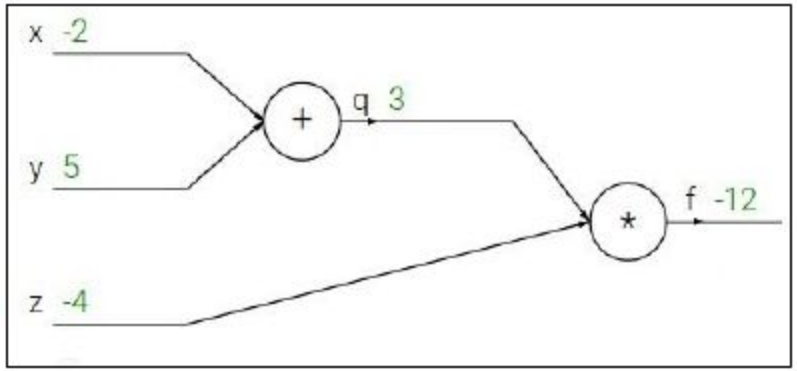
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

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

Backpropagation: a simple example

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

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



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

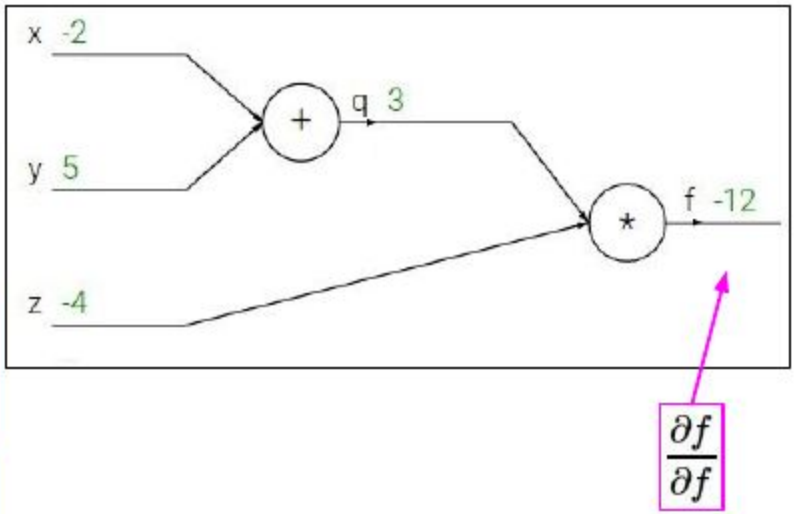
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

Backpropagation: a simple example

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

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



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

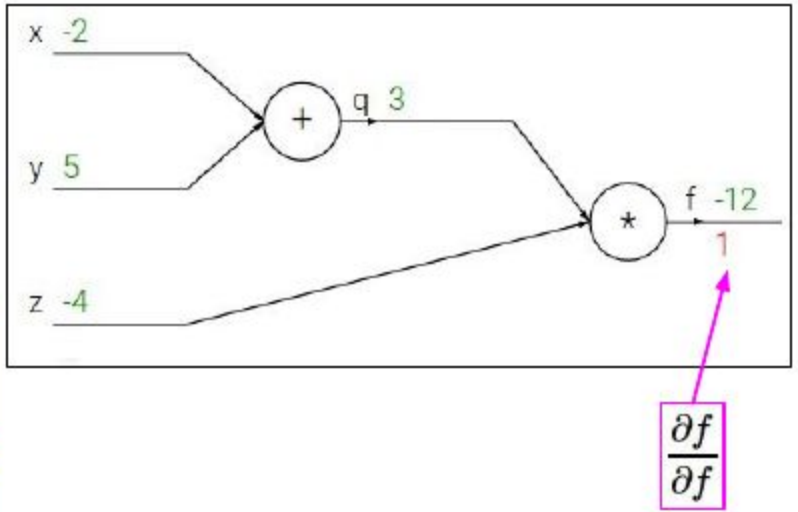
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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

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

Backpropagation: a simple example

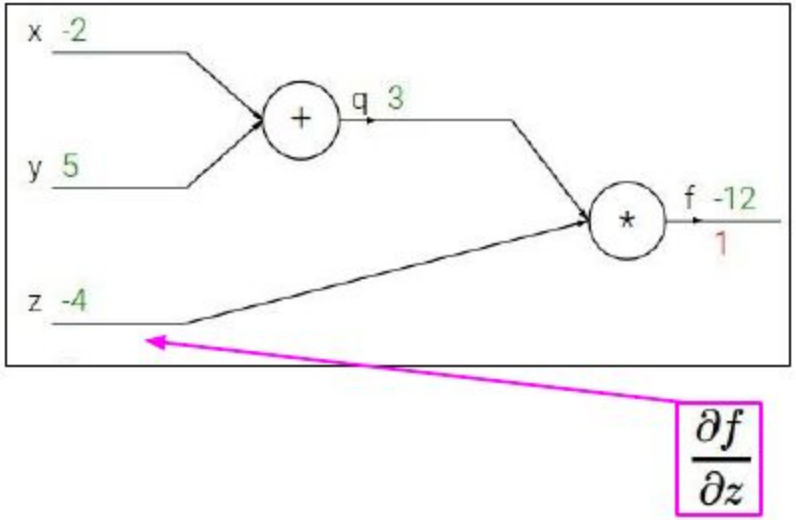
$$f(x, y, z) = (x + y)z$$

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



Backpropagation: a simple example

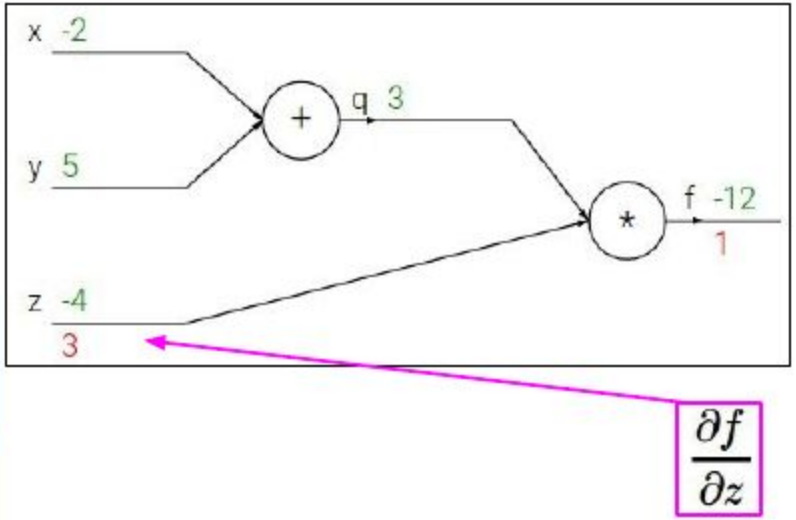
$$f(x, y, z) = (x + y)z$$

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



Backpropagation: a simple example

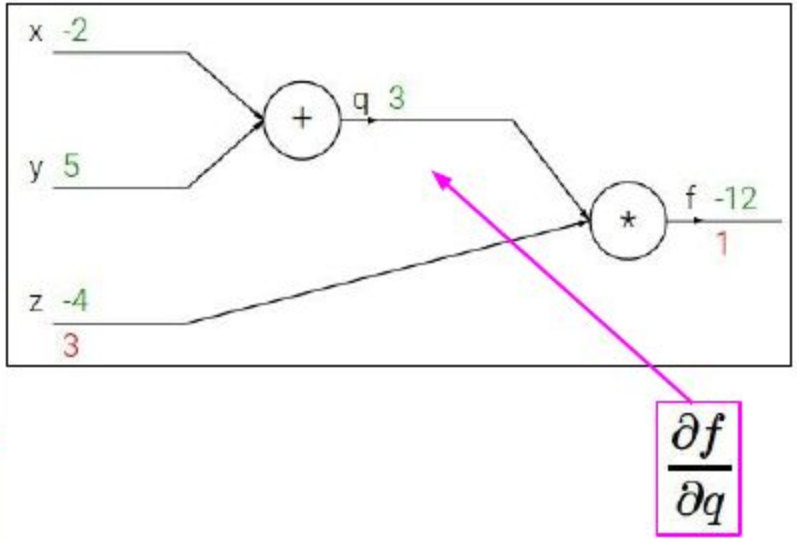
$$f(x, y, z) = (x + y)z$$

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



Backpropagation: a simple example

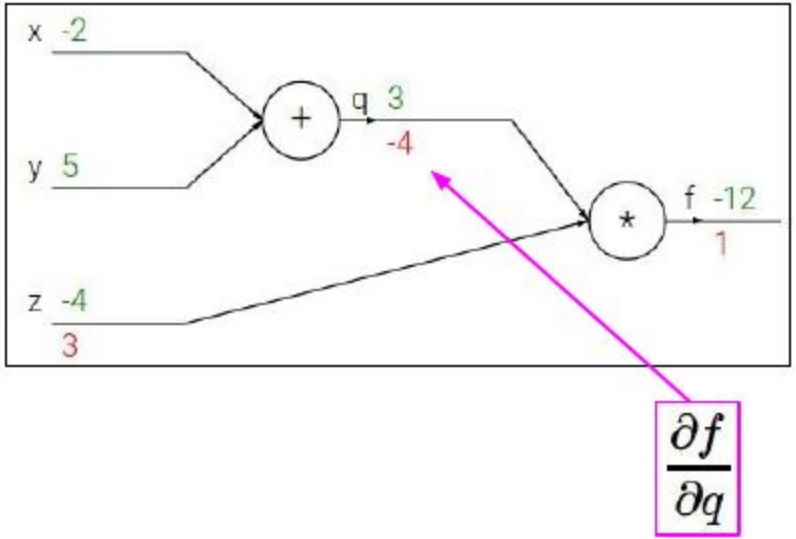
$$f(x, y, z) = (x + y)z$$

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

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

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

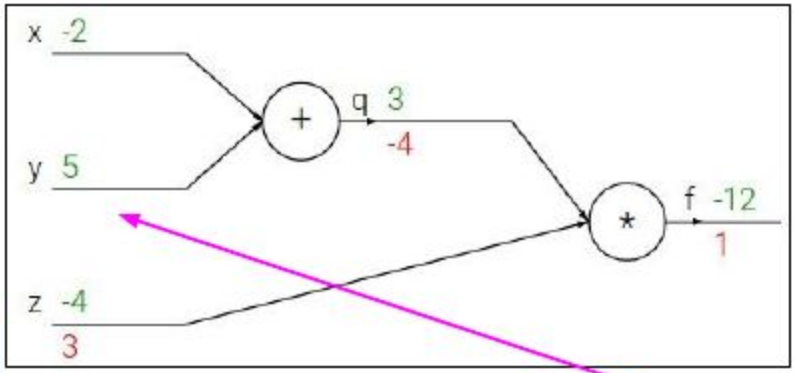
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

Chain rule:

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

Upstream gradient — Local gradient

Backpropagation: a simple example

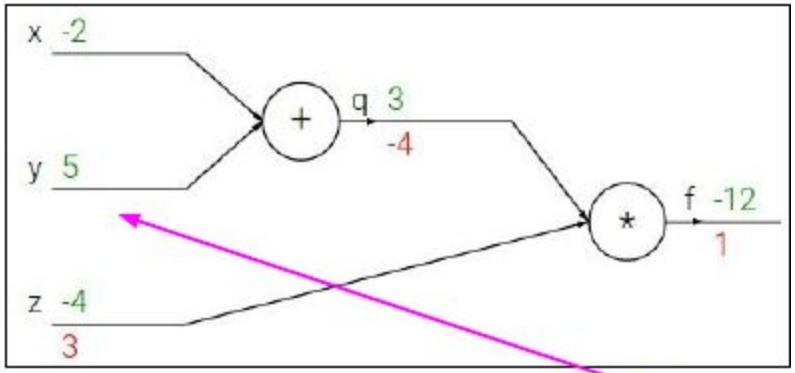
$$f(x, y, z) = (x + y)z$$

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



Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

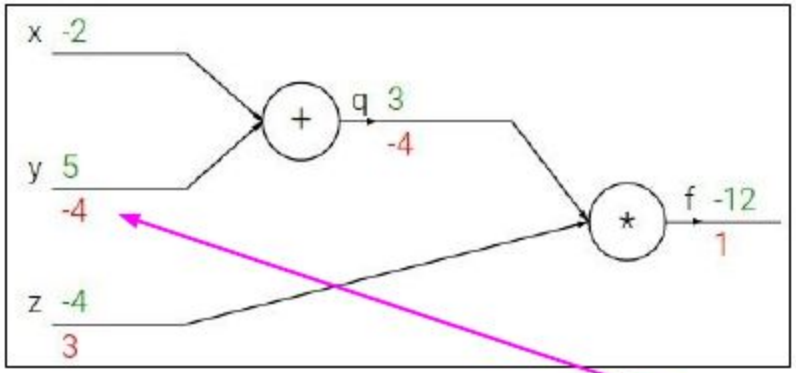
$$f(x, y, z) = (x + y)z$$

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



Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

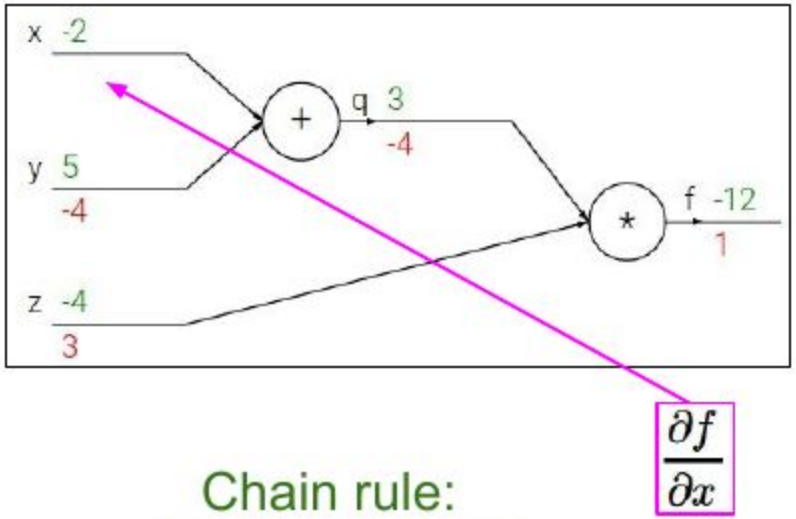
$$f(x, y, z) = (x + y)z$$

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



Chain rule:

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

Upstream gradient

Local gradient

Backpropagation: a simple example

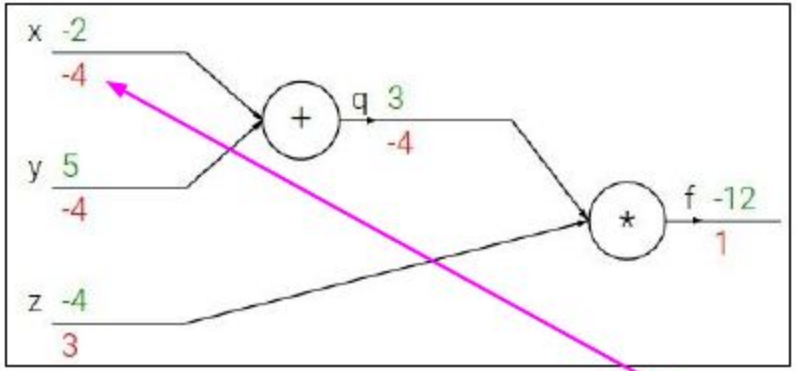
$$f(x, y, z) = (x + y)z$$

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



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

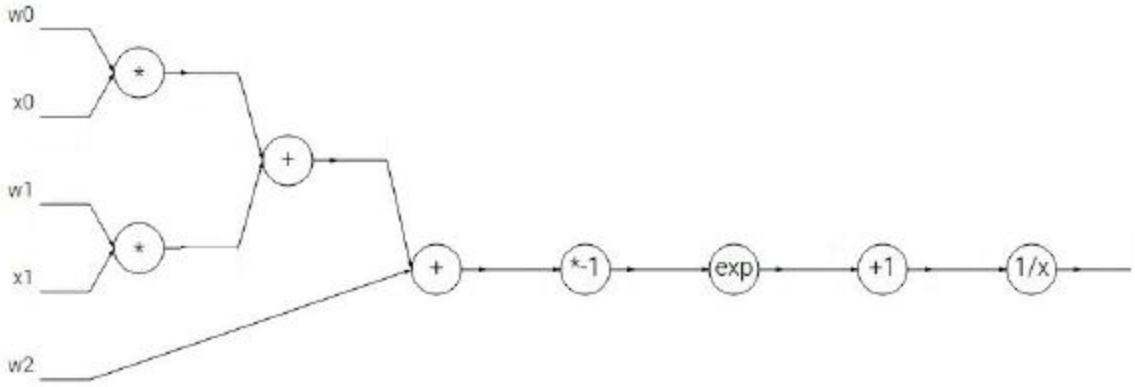
Chain rule:

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

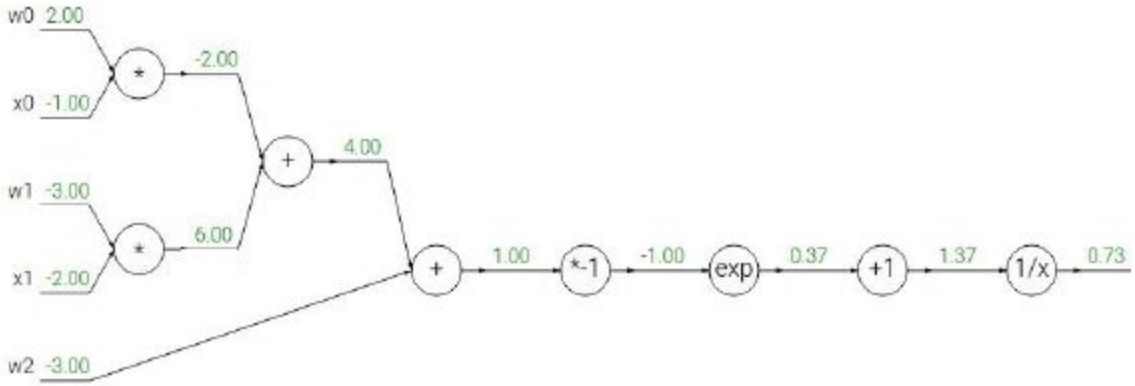
Upstream gradient

Local gradient

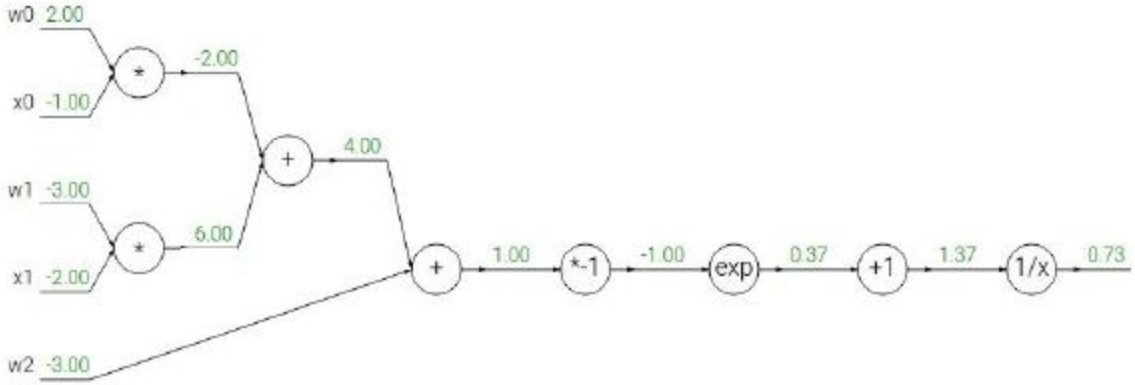
Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



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



Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



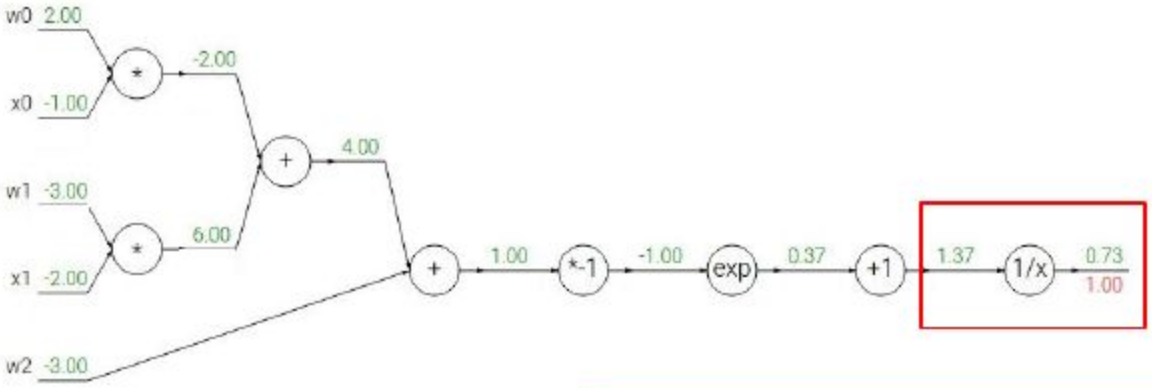
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

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

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

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

Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



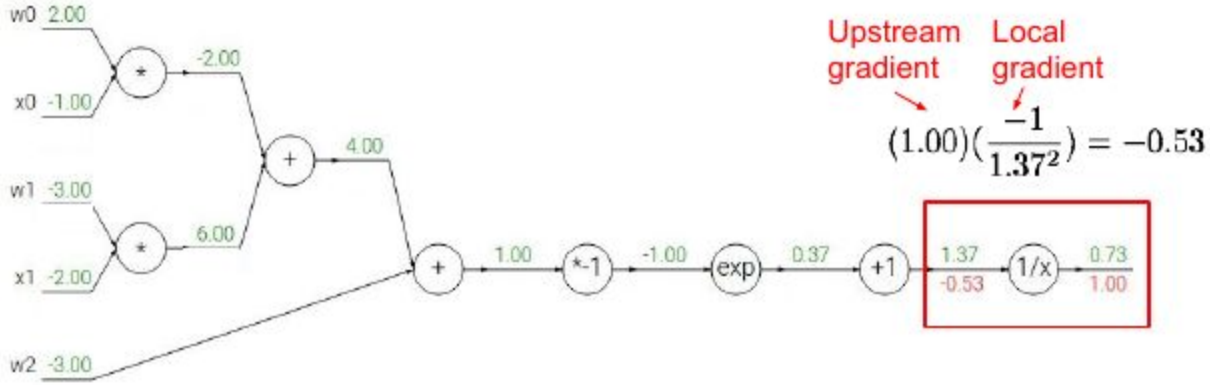
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



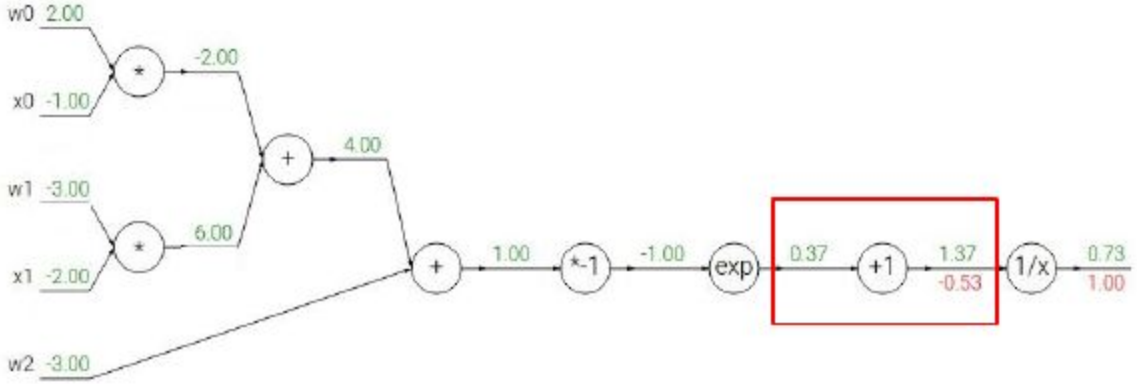
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



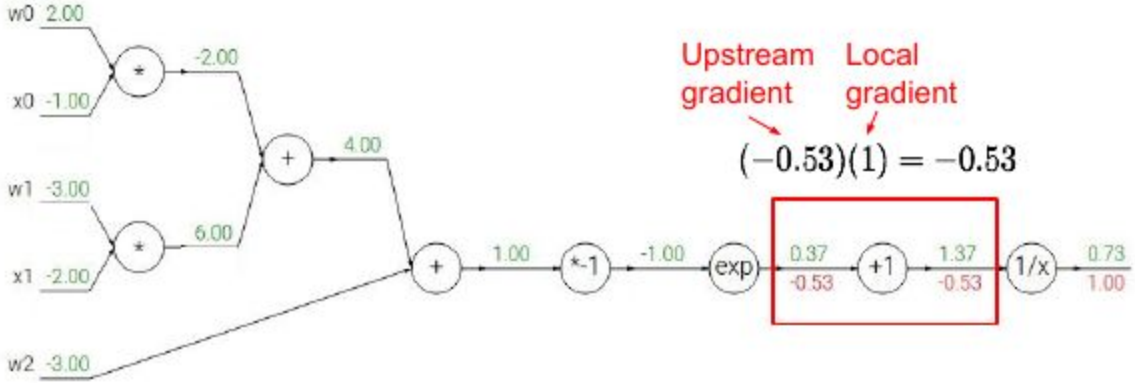
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



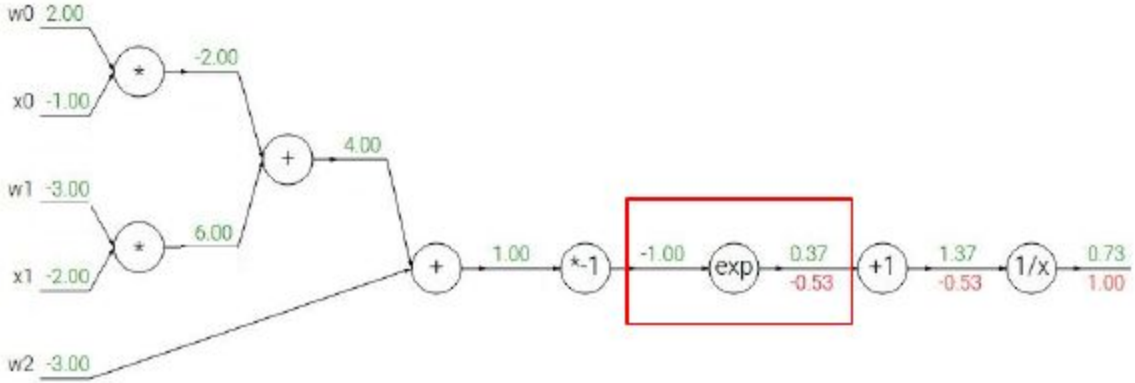
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



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

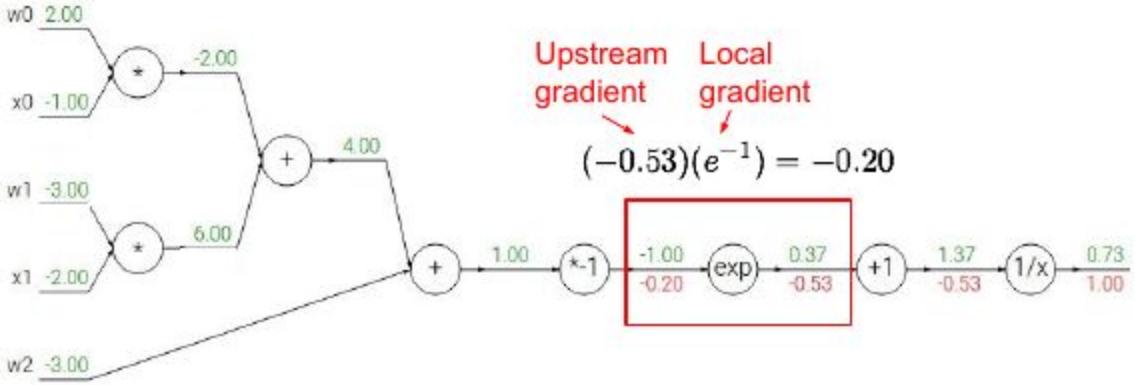
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

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

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



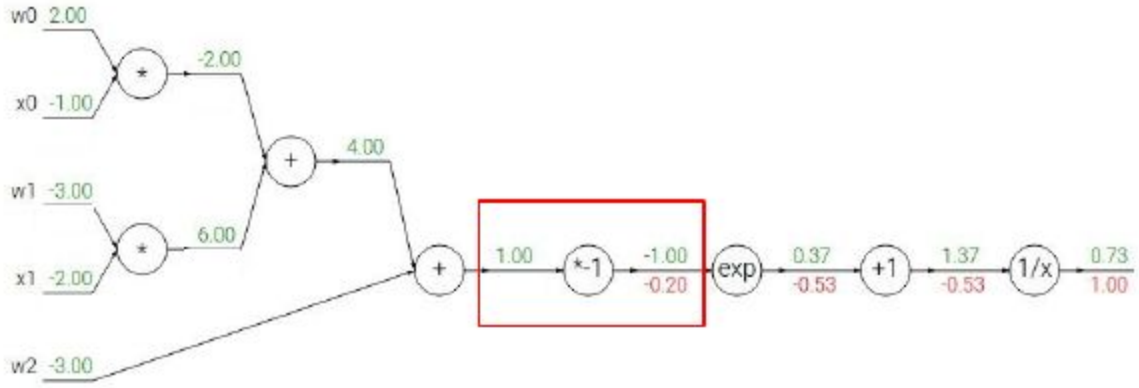
$f(x) = e^x$

$f_a(x) = a$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$

$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

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



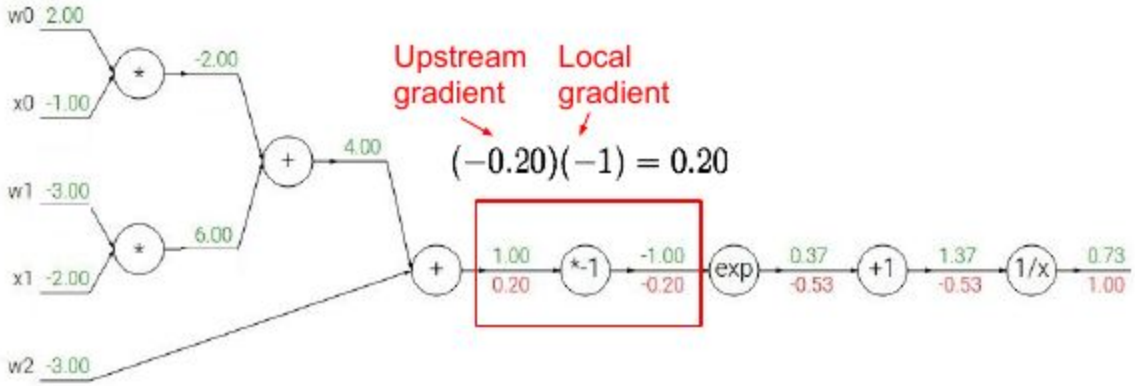
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



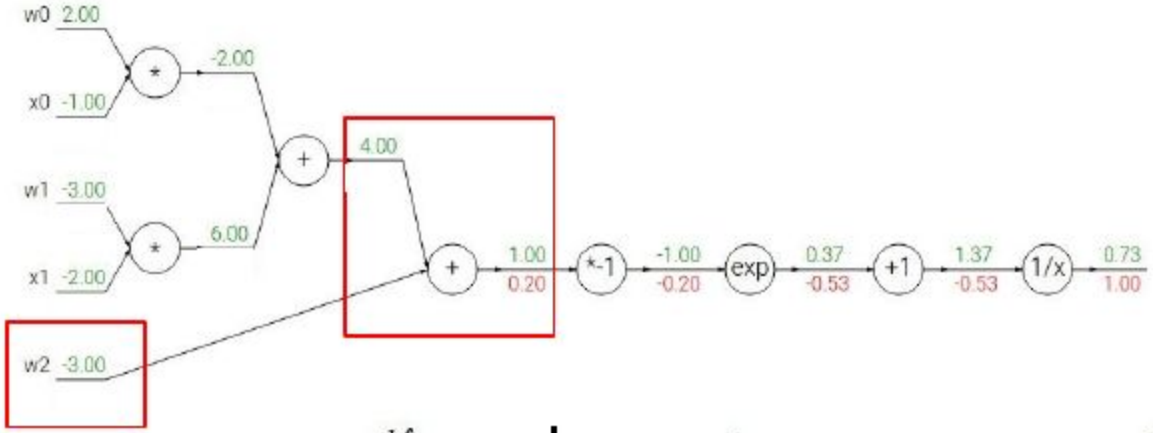
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



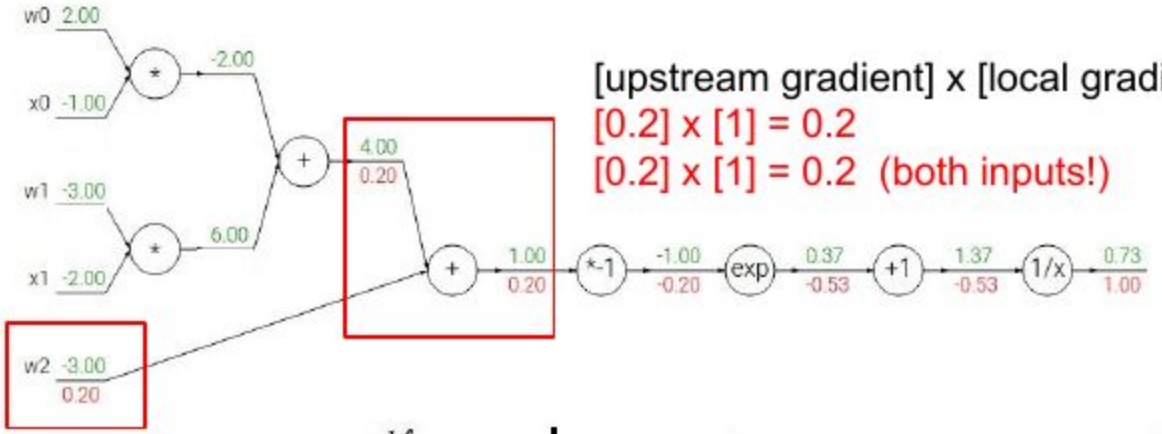
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

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



[upstream gradient] x [local gradient]
[0.2] x [1] = 0.2
[0.2] x [1] = 0.2 (both inputs!)

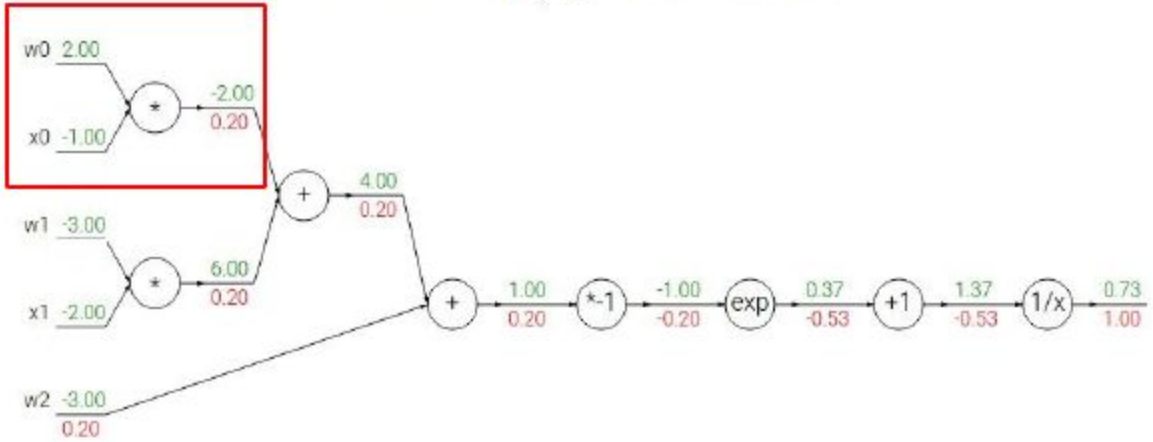
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

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

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

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

Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



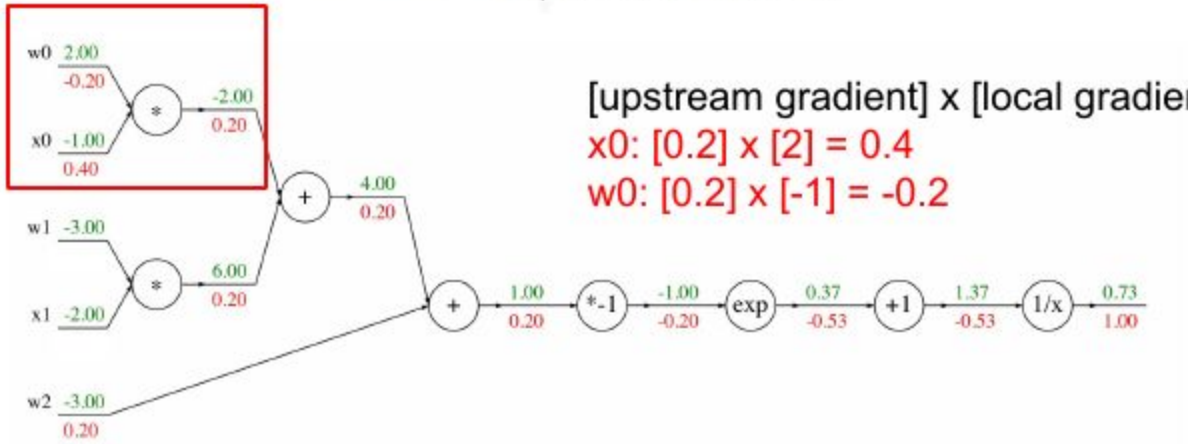
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

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

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

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

Another example: $$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



[upstream gradient] x [local gradient]

x0: [0.2] x [2] = 0.4

w0: [0.2] x [-1] = -0.2

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

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

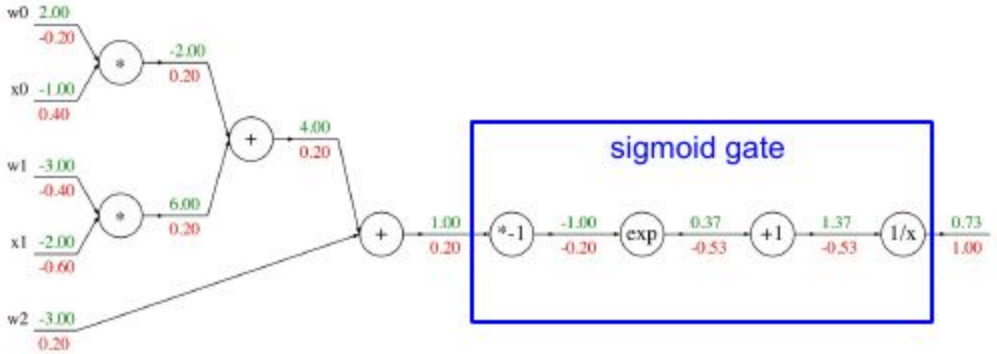
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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

sigmoid function

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



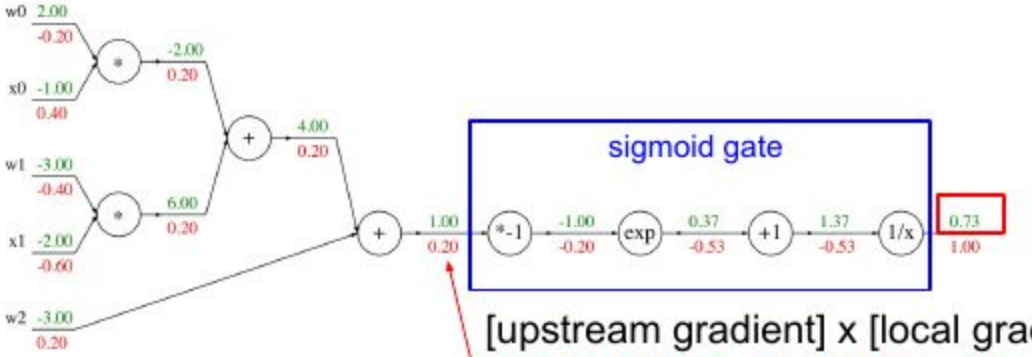
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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

sigmoid function

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



[upstream gradient] x [local gradient]

[1.00] x [(1 - 0.73) (0.73)]= 0.2

Problem Setup: Things to define

- Given a training set of input-output pairs
  $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Err(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

w.r.t W

- This is problem of function minimization
  - An instance of optimization

Problem Setup: Things to define

- Given a training set of input-output pairs
  $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

$$Err(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

w.r.t W

What is f() and what are its parameters?

- This is problem of functio
  - An instance of optimization

Problem Setup: Things to define

- Given a training set of input-output pairs
  $(\boldsymbol{X}_1, \boldsymbol{d}_1), (\boldsymbol{X}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{X}_T, \boldsymbol{d}_T)$
- What are these input-output pairs?

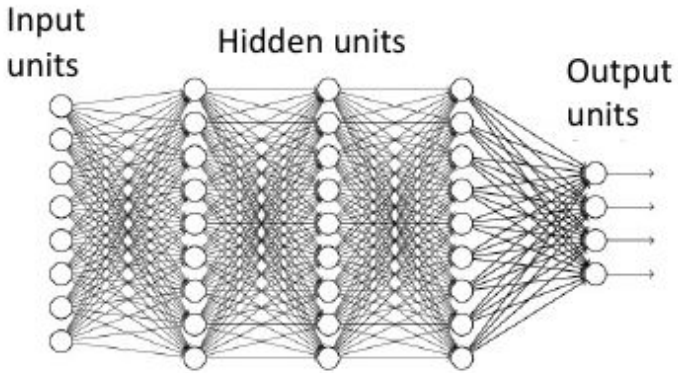
$$Err(W) = \frac{1}{T}\sum_i div(f(X_i; W), d_i)$$

What is the divergence div()?

What is f() and what are its parameters W?

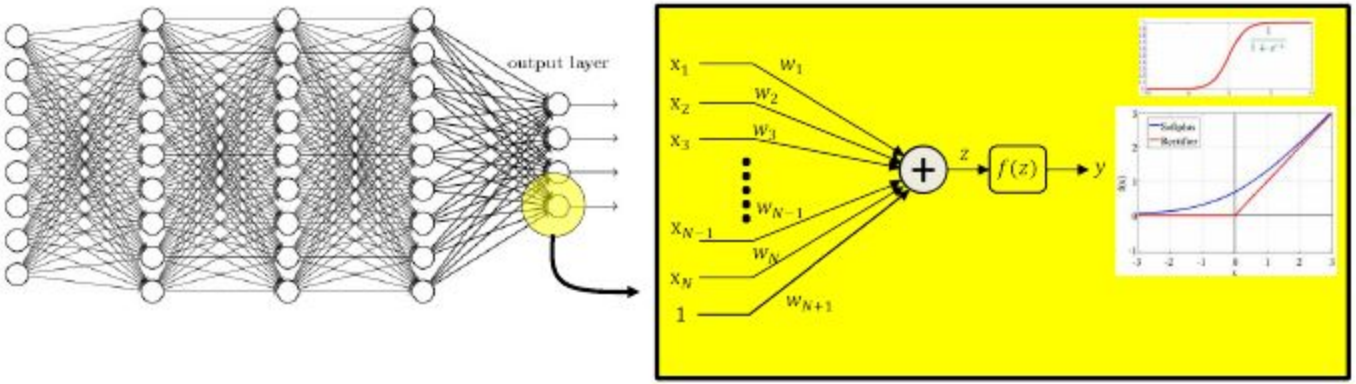
- This is problem of functio
  - An instance of optimization

What is f()? Typical network



- Multi-layer perceptron
- A *directed* network with a set of inputs and outputs
  - No loops
- Generic terminology
  - We will refer to the inputs as the *input units*
    - **No neurons here – the "input units" are just the inputs**
  - We refer to the outputs as the output units
  - Intermediate units are "hidden" units

The individual neurons



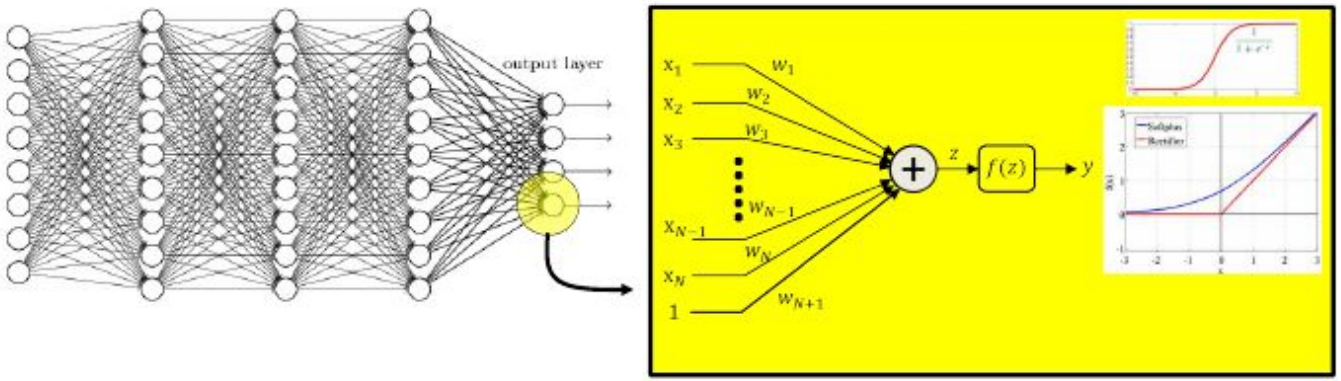
- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A differentiable activation function applied the sum of weighted inputs and a bias

$$y = f\left(\sum_i w_i\, x_i + b\right)$$

  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

The individual neurons



- Individual neurons operate on a set of inputs and produce a single output
  - **Standard setup:** A differentiable activation function applied the sum of weighted inputs and a bias

$$y = f\left(\sum_i w_i\, x_i + b\right)$$

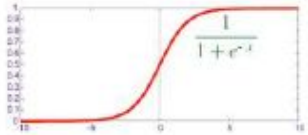
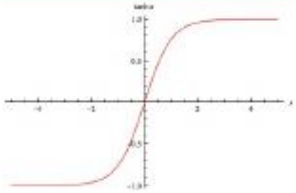
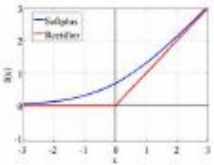
  - More generally: *any* differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

We will assume this unless otherwise specified

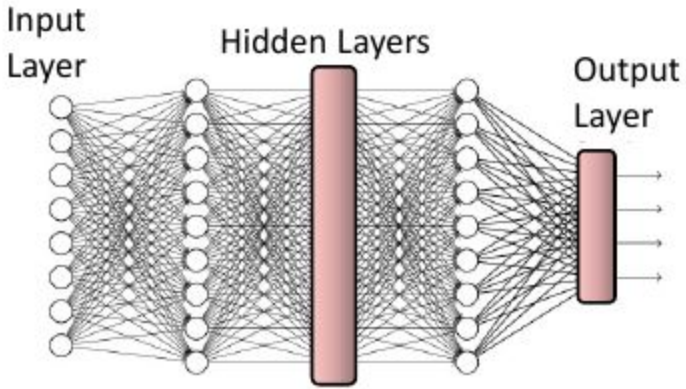
Parameters are weights w_i and bias b

Activations and their derivatives

	$f(z) = \frac{1}{1+\exp(-z)}$	$f'(z) = f(z)(1-f(z))$
	$f(z) = \tanh(z)$	$f'(z) = (1-f^2(z))$
	$f(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$	This space left intentionally (kind of) blank
	$f(z) = \log(1+\exp(z))$	$f'(z) = \frac{1}{1+\exp(-z)}$

- Some popular activation functions and their derivatives

Vector Activations

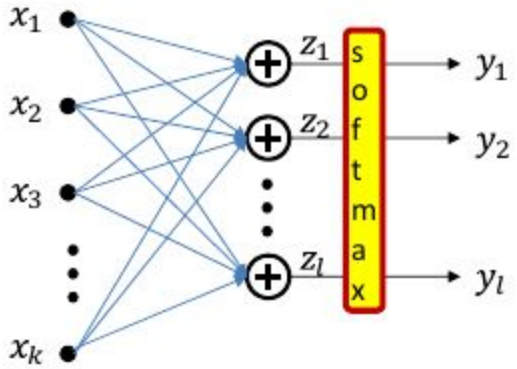


- We can also have neurons that have *multiple coupled* outputs

$$[y_1, y_2, \ldots, y_l] = f(x_1, x_2, \ldots, x_k; W)$$

  - Function $f()$ operates on set of inputs to produce set of outputs
  - Modifying the parameters W will affect *all* outputs

Vector activation example: Softmax



- Example: Softmax *vector* activation

$$z_i = \sum_j w_{ji} x_j + b_i$$

Parameters are weights w_{ji} and bias b_i

$$y = \frac{exp(z_i)}{\sum_j exp(z_j)}$$