Knowledge and solutions for a changing world

Be boundless

Advancing data-intensive discovery in all fields

NEURAL NETWORKS

UW DIRECT

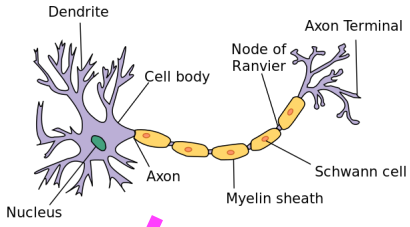
(Data Intensive Research Enabling Cutting-edge Tech)

https://uwdirect.github.io

Stéphanie Valleau

Chemical Engineering

Biological neural networks



Biological neurons

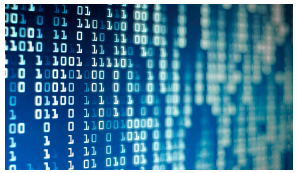
Y

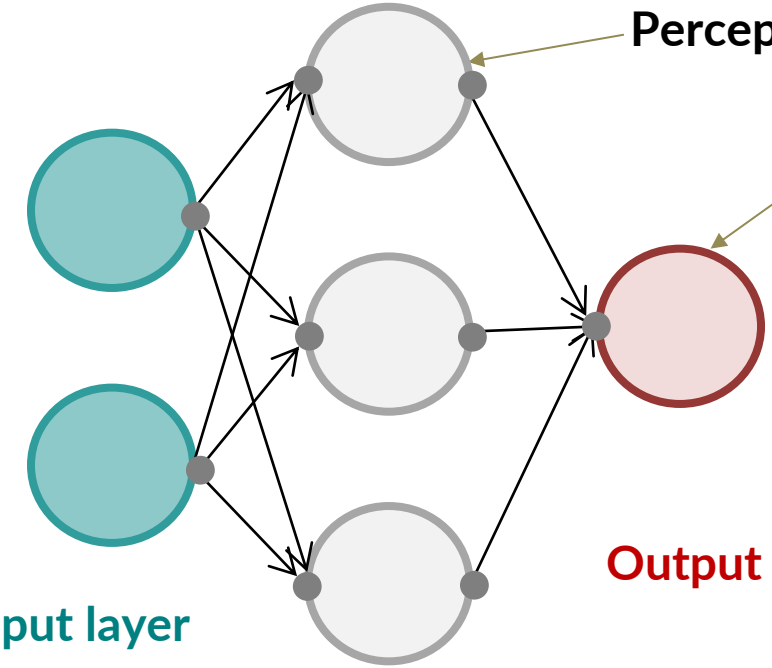
It's a Puppy!

Artificial neural networks

Simple feed forward neural network

Input features *X*



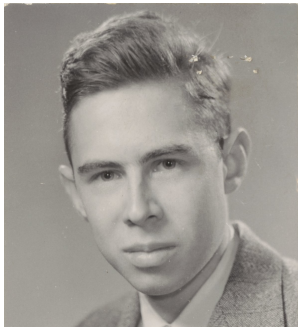


Target output *Y*

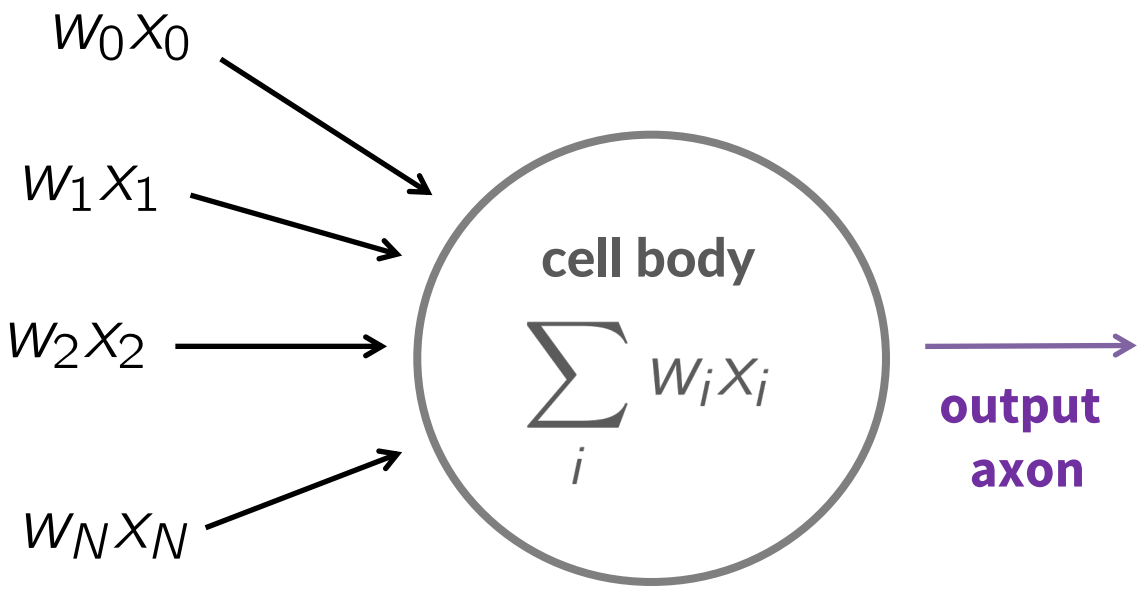
- Numerical
- Categorical

Historically … perceptrons

Proposed by
Frank Rosenblatt
in early 1960s



Input *X*
(e.g. through dendrites)



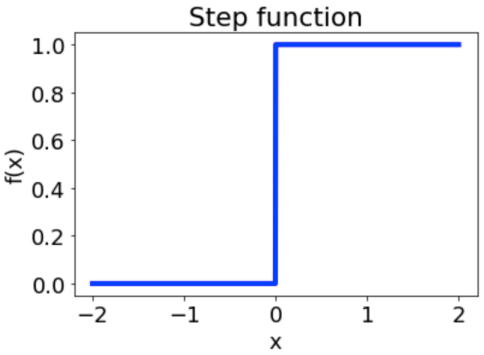
Binary output

$$\begin{cases} 0 & \text{if } \mathbf{x} \cdot \mathbf{w} \leq \text{threshold} \\ 1 & \text{if } \mathbf{x} \cdot \mathbf{w} > \text{threshold} \end{cases}$$

Vector dot product

$$\mathbf{x} \cdot \mathbf{w} = \sum_{i=1}^{N} x_i w_i$$

Artificial neurons



Input
(e.g. through dendrites)

$w_0 x_0$

$w_1 x_1$

$w_2 x_2$

$w_N x_N$

***f* - activation function**

cell body

$\sum_i w_i x_i + b$

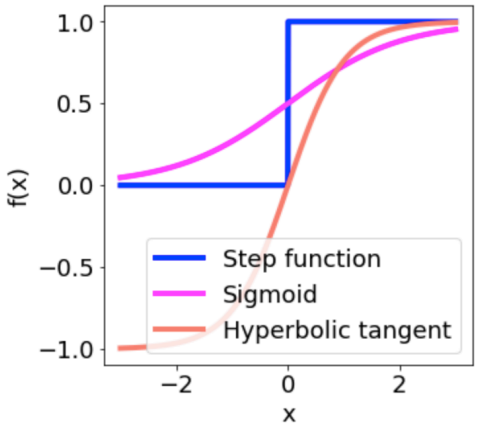
f

$$f\left(\sum_{i=1}^{N} x_i w_i + b\right)$$

output axon

Bias – alters the position of the decision boundary

Artificial neurons



Input
(e.g. through dendrites)

$w_0 x_0$

$w_1 x_1$

$w_2 x_2$

$w_N x_N$

***f* - activation function**

cell body

$\sum_i w_i x_i + b$

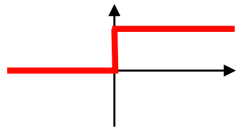
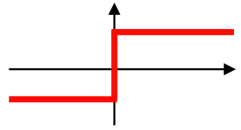
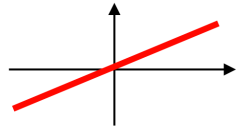
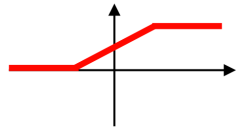
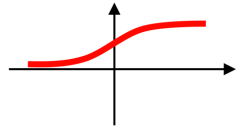
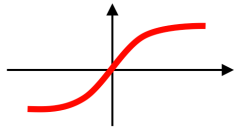
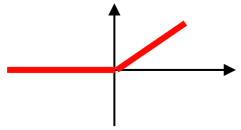
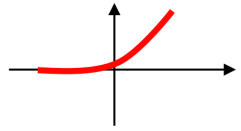
f

$$f\left(\sum_{i=1}^{N} x_i w_i + b\right)$$

output axon

Bias -

Many types of activation functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

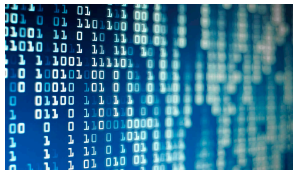
https://medium.com/datathings/dense-layers-explained-in-a-simple-way-62fe1db0ed75

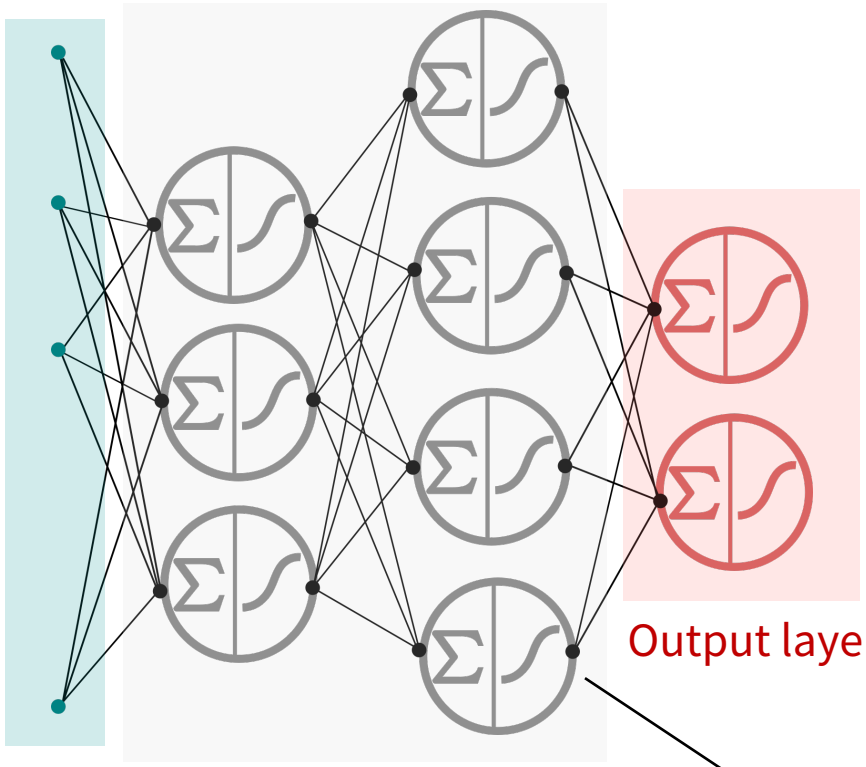
What are neural networks?

Simple decisions

More complex decisions

Input features *X*





Input layer

N Hidden layers

N > 1

Output layer

Target output *Y*

- Numerical
- Categorical

One output fed into multiple layers!

How do we find weights & bias?

We can evaluate our error using a **"Cost" function** i.e. the MSE

$$C(\mathbf{w}, \mathbf{b}) \equiv \text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

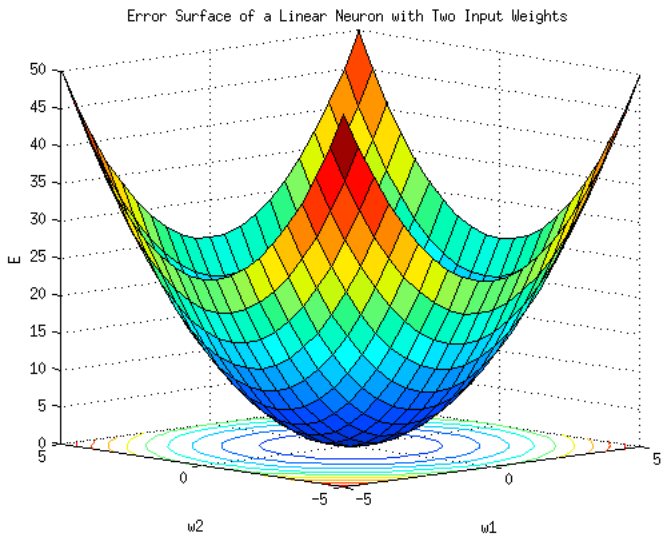
and minimize respect to **w** and **b**. Often it is called the **loss function**

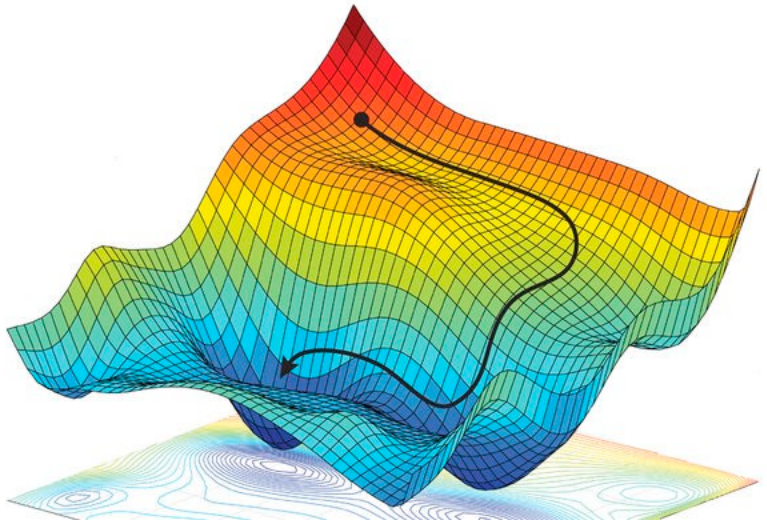
Note

- ***w*** and ***b*** are the **vectors** of weights and biases collectively
- ***y*** can be vectors (take the norm difference)
- cost is often defined as **MSE/2** (choice to remove factor of 2 when taking derivative)
- cost can be defined using e.g. MAE (mean absolute error), cross-entropy etc…
- ***C*** is often indicated as ***J***

Learning in neural networks

Goal: learn to minimize the ***C*** function





We have analytical, differential forms for all components
Find an analytical solution!

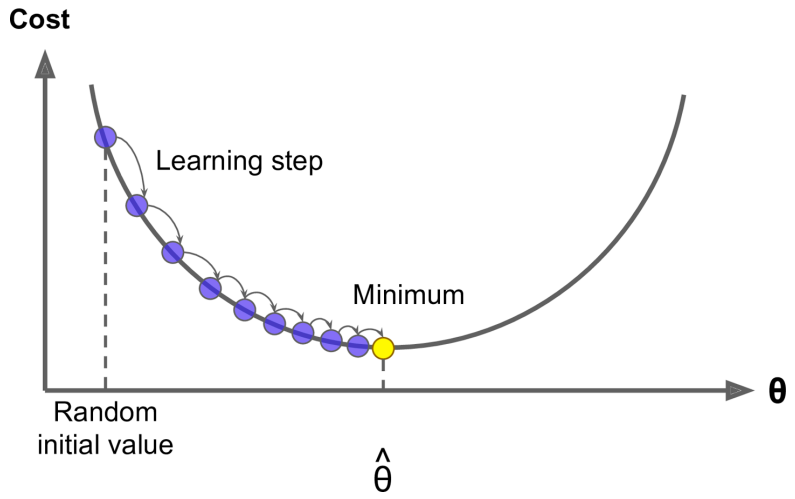
Can be very complicated in general

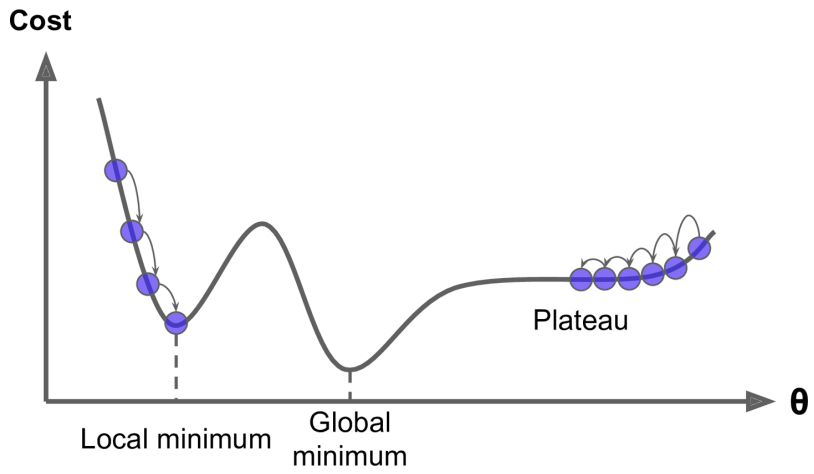
https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-gradient-descent-fcd5e0fc077d

Gradient descent

Update parameters $\theta = (w, b)$ based on the **slope of the cost** function!

- Process is iterative
- Starts with a random guess of weights / slopes
- Stops when you reach a local minimum

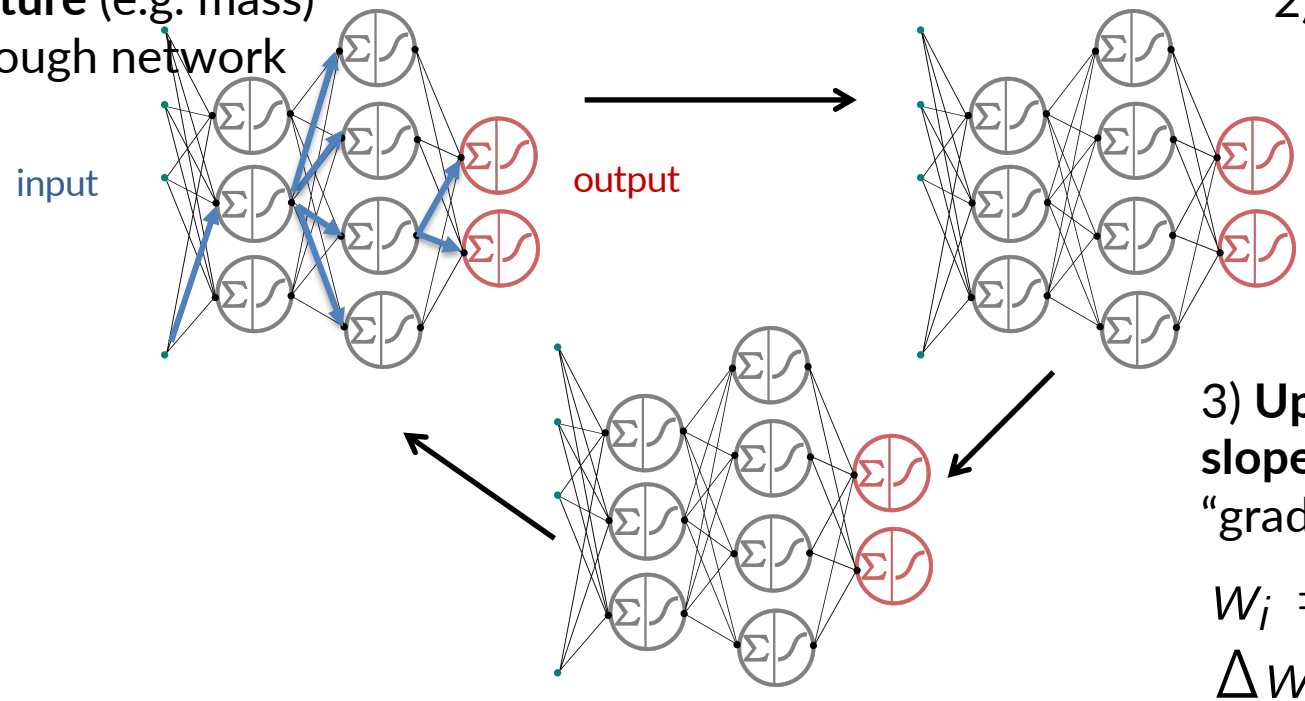




https://learning.oreilly.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Backpropagation + gradient descent

1) Propagate **input feature** (e.g. mass) through network



2) Quantify network **error "cost/loss function"** e.g. MSE

3) **Update weights / slope** based on "gradient" of error

$$w_i = w_i + \Delta w_i$$

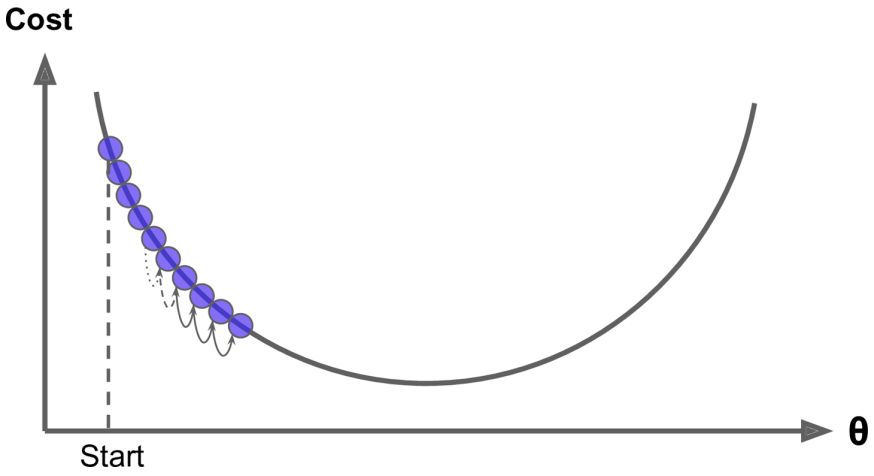
$$\Delta w_i = -\eta G(w)$$

Learning rate

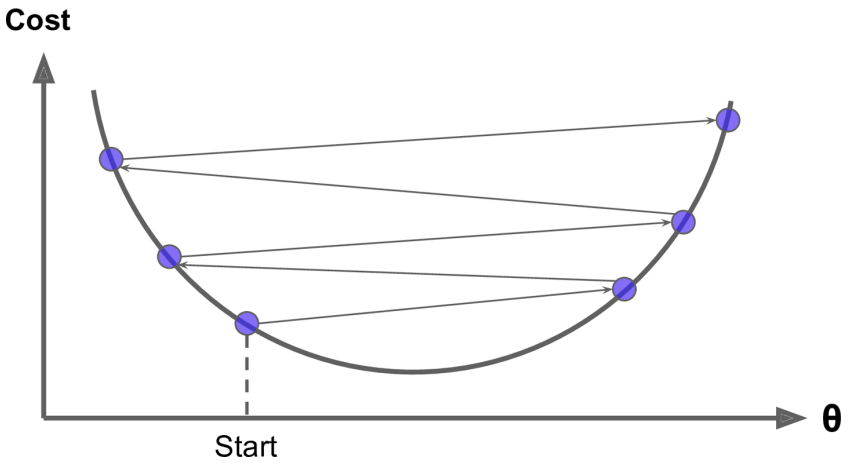
so long as change is smaller than max step size

The learning rate η

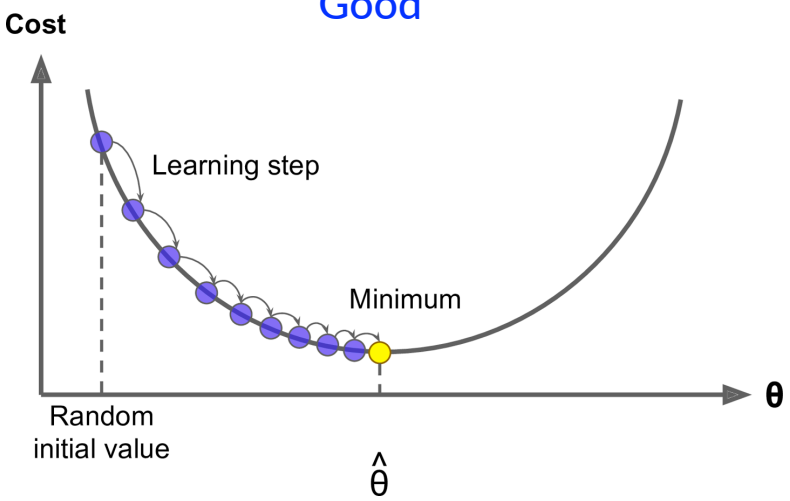
Too small



Too large



Good



Need to find a good value for the learning rate!

https://learning.oreilly.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Terminology

- **Cost** a.k.a. loss a.k.a. error function

$$C(\mathbf{w}, \mathbf{b}) \equiv \text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- **Maximum step size**, ϵ
- **Learning rate**, η (may change during training)
- One **epoch**: passing the entire data once
- **Maximum training epochs**
- **Convergence** of *C* and *w* and *b*
- **Hyperparameters** - those not “trained”, e.g. number of layers, number of neurons per layer, learning rate η

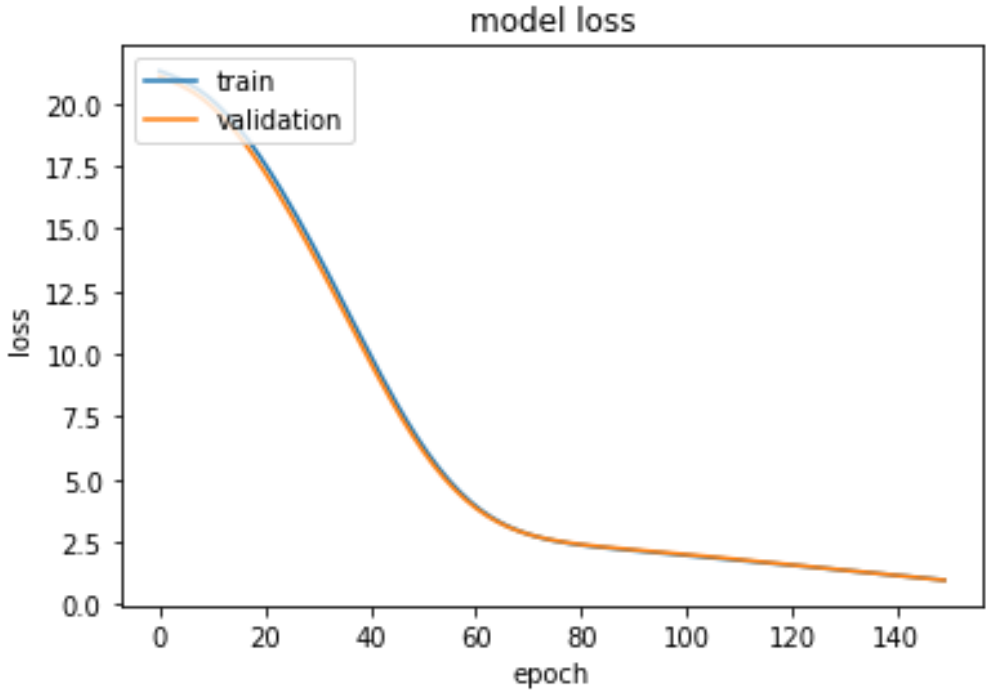
Monitoring training

With NN, we still do a test-train split

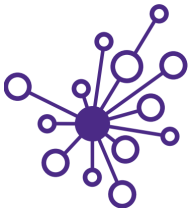
It is useful to plot the cost function or classification accuracy

- Over training epochs
- For train and validation

Data from keras on first 100k records in HCEPDB using only input and output layer.



Questions?



In your own time …

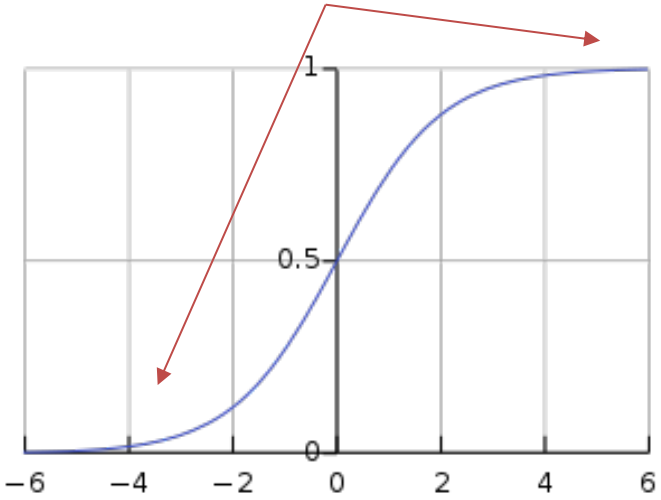
In the next few slides

- Types of **activation functions** (advantage / disadvantage)
- Types of **cost functions**

Limitations of sigmoids

Learning slows down at sigmoid asymptotes

$$\sigma(w \cdot x + b) = \frac{1}{1 + e^{-\sum_i x_i w_i - b}}$$



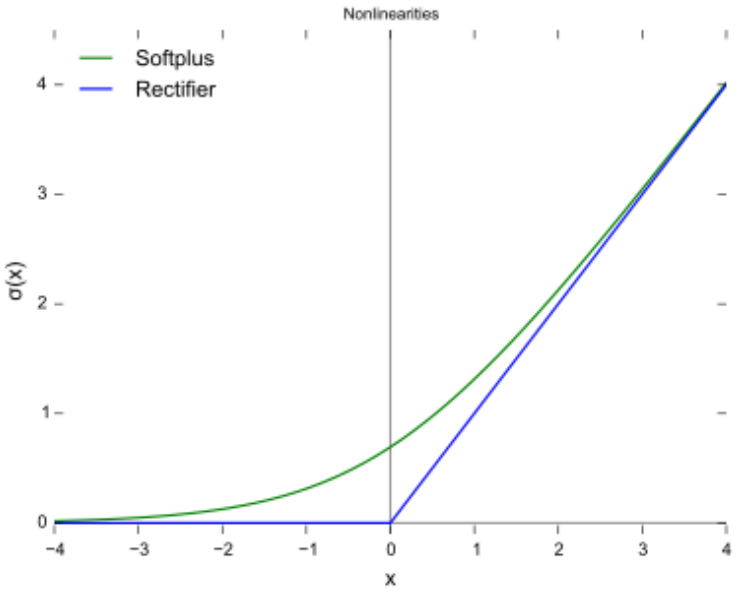
Partial derivative of *C* with respect to *w* & *b*
is very small near 0 and 1:
Large changes in w_i and or b_i have **little effect on C**

Solutions …. Change the activation or cost functions!

Activation alternatives

Recently (>2011) many have started using the **Rectified Linear Unit (RLU or relu)**

$$f(x) \coloneqq \max(0, x)$$



- Positive, finite
- Very fast (no exp)
- Doesn't saturate
- Creates sparse networks
- Not differentiable at 0
- No symmetry & unbounded
- Large η can kill neurons so that they never fire

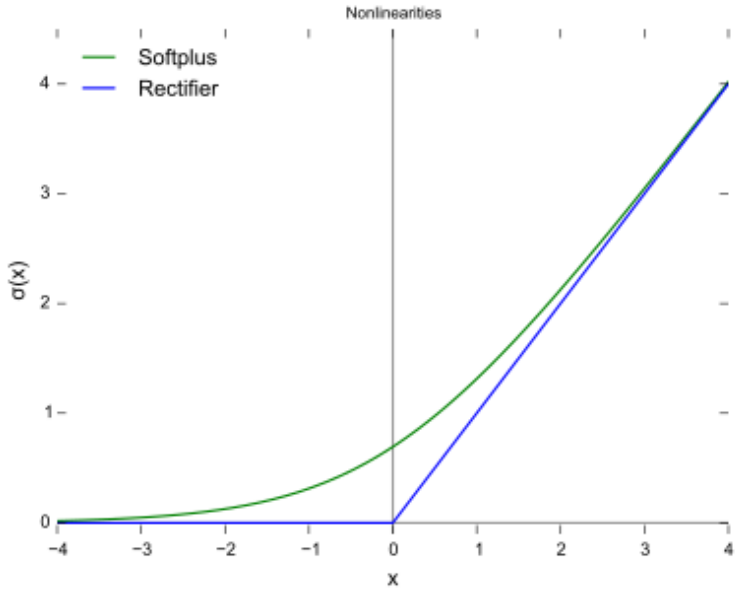
Wikipedia

Activation alternatives

Some of the problems are solved by **softplus**

$$f(x) := \log(1 + e^x)$$

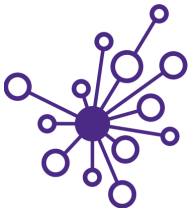
- Smoothed ReLU
- Doesn't saturate
- Differentiable everywhere
  - Derivative is sigmoid
- Slower than ReLU



Wikipedia

A better cost function

for classification



A new cost function, **cross-entropy**

$$C := -\frac{1}{n}\sum_{x}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})]$$

- n is the number of samples in the training set
- $\hat{y}$ is the prediction for the a given sample x
- y is the observed value for the sample x

- *Note*: this function is always positive