Machine Learning Course 3
Neural Networks

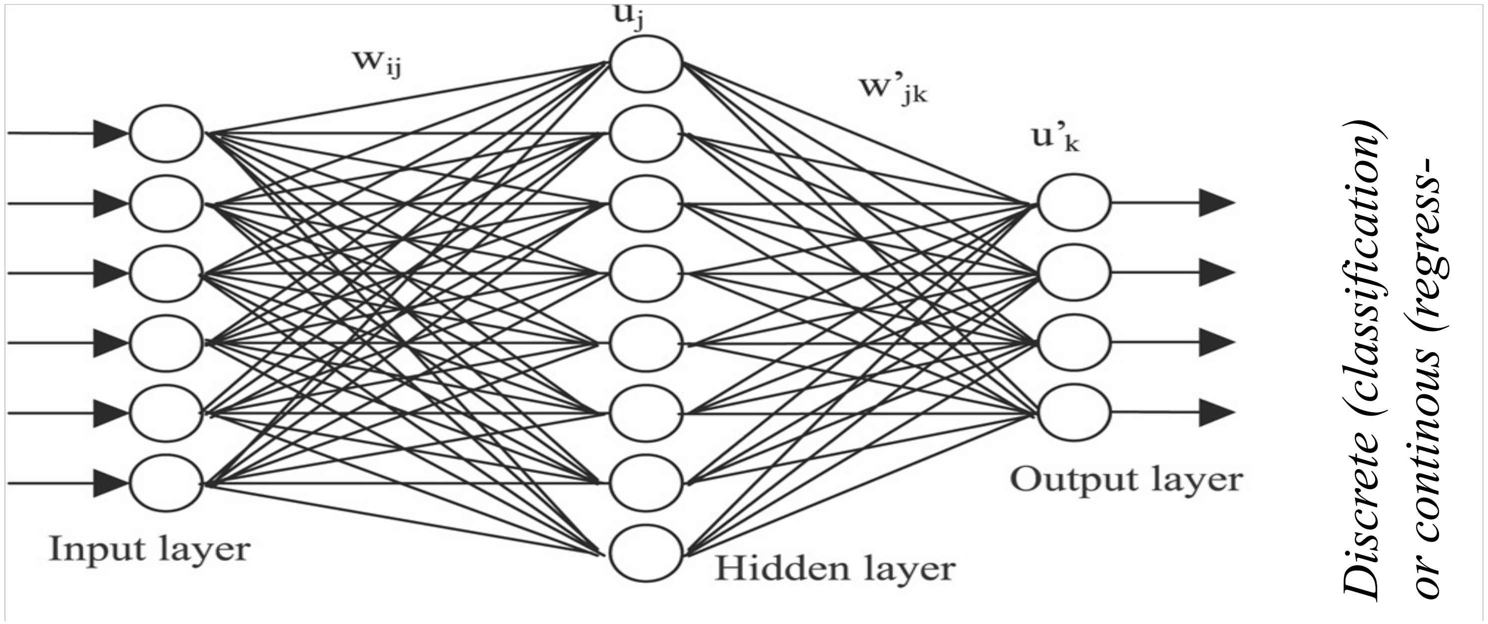
David Rousseau

IJCLab-Orsay

david.rousseau@in2p3.fr @dhpmrou

CHACAL, Johannesburg, Jan 2024

Neural Net in a nutshell

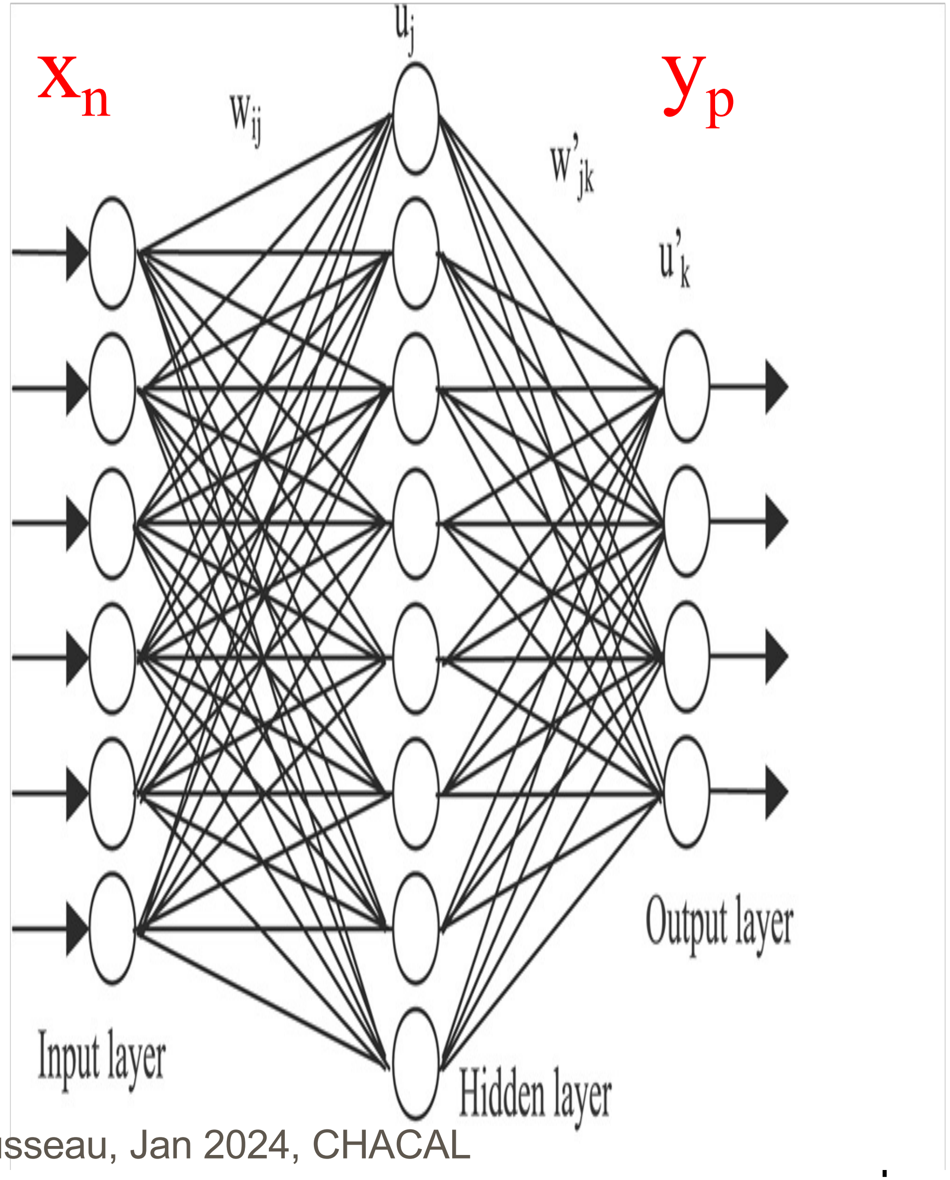


- Neural Net ~1950!
- But many many new tricks for learning
- “Deep Neural Net” up to 100 layers and more
- Computing power (DNN training can take days, GPT months on thousands of GPU…)

Universal Theorem

Universal Approximation theorem

- Mathematical theorem 1991 https://en.wikipedia.org/wiki/Universal_approximation_theorem
- Any continuous, bounded function $R^n \rightarrow R^p$
- … can be approximately sufficiently well (better than a given ε)
- … with a sufficiently large single hidden layer neural net

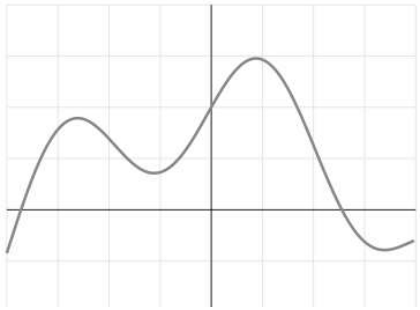


Addendum ResNet 1 neuron sufficient depth

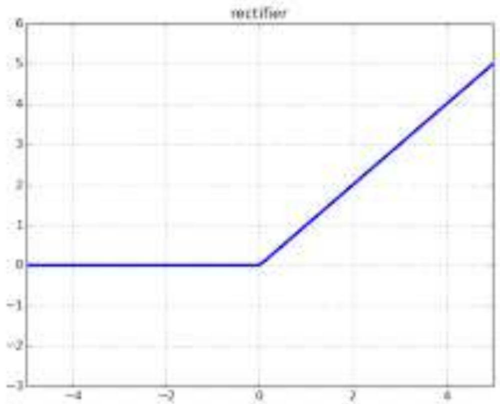
Universal Theorem at work

Universal approximation

We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



relu(x) = x if x>0 & 0 otherwise



$$\mathrm{Relu}(\mathrm{ax} + \mathrm{b})$$

Universal Theorem at work

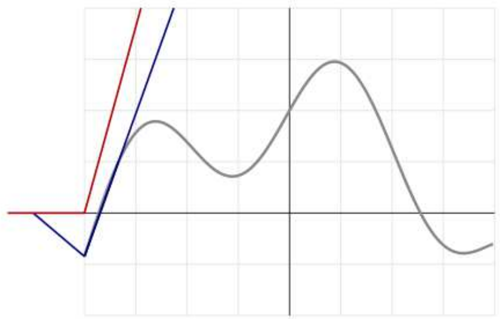
Universal approximation

We can approximate any $f \in \mathscr{C}([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions

Universal Theorem at work

Universal approximation

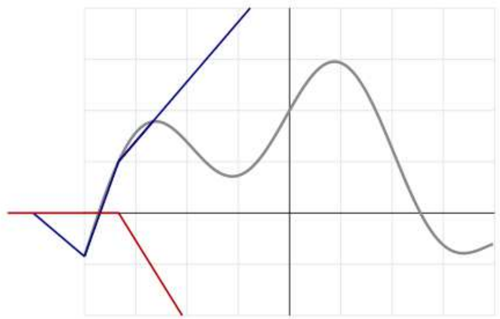
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

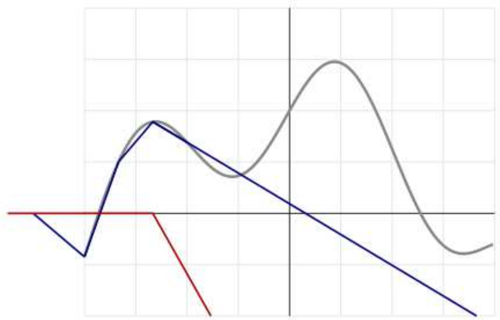
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

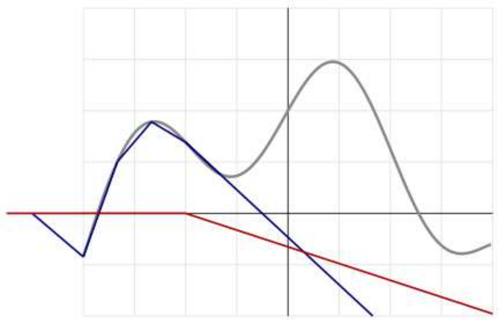
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

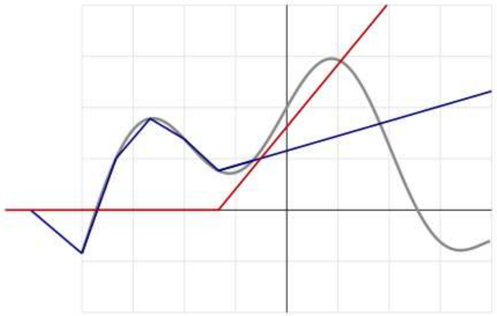
We can approximate any $f \in \mathscr{C}([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

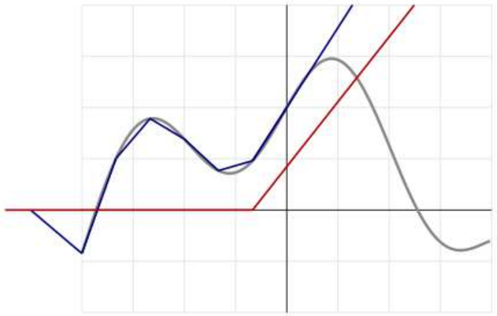
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

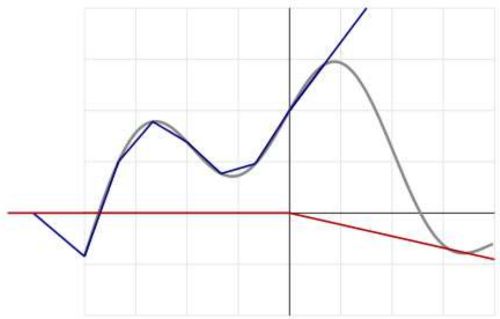
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

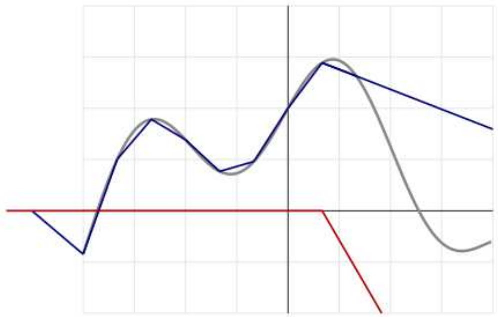
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

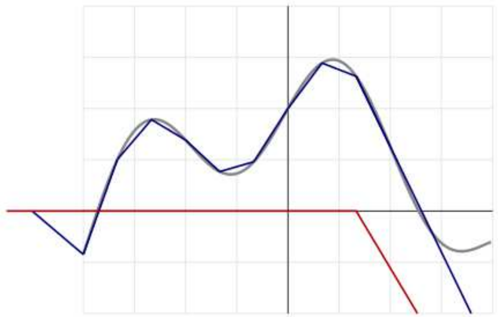
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

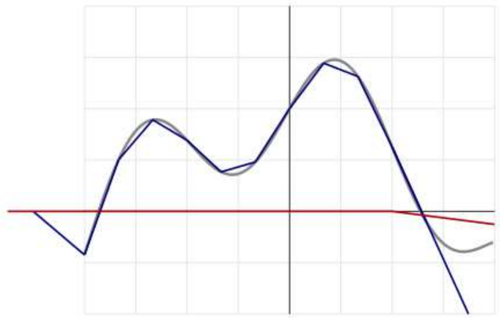
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

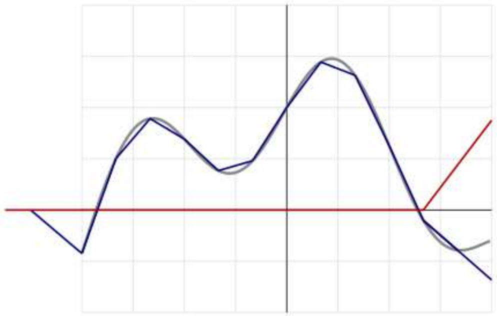
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

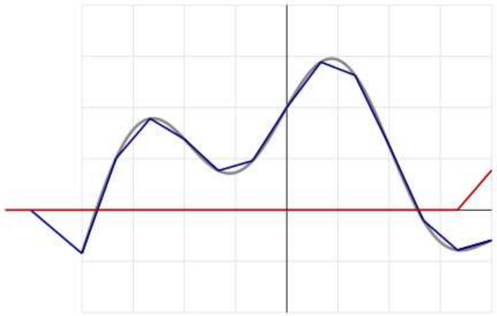
We can approximate any $f \in \mathscr{C}([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions



Universal Theorem at work

Universal approximation

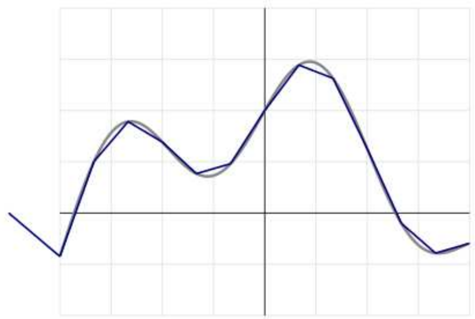
We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions

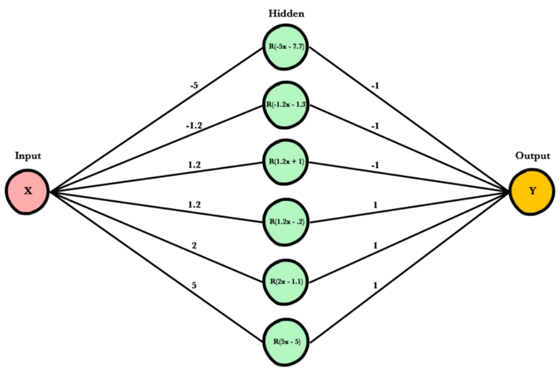


Universal Theorem at work

Universal approximation

We can approximate any $f \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions

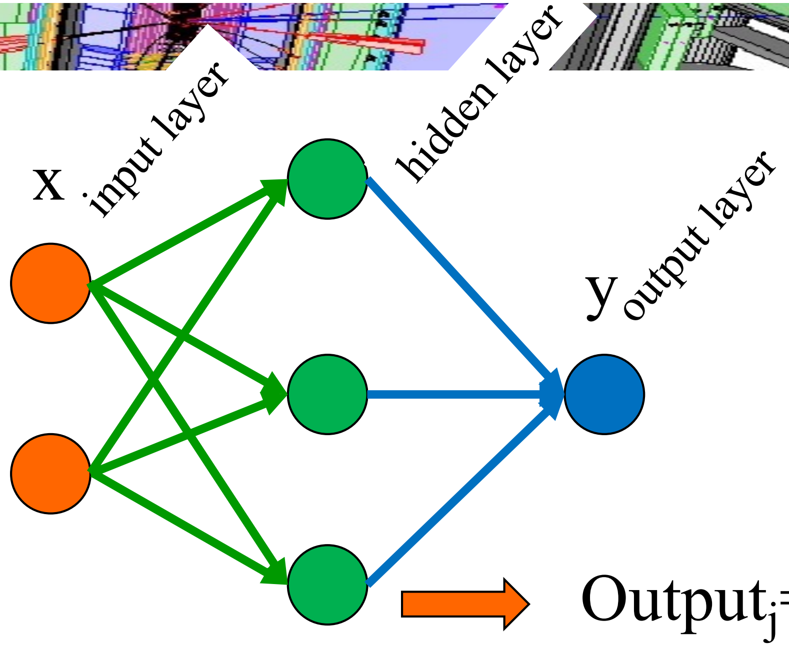




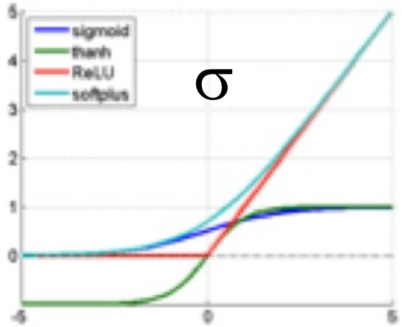
$$y = \sum_i \mathrm{Relu}(\mathrm{a_i} \times \mathrm{x} + \mathrm{b_i})$$

$$\mathbb{R} \to \mathbb{R} \text{ generalised to } \mathbb{R}^n \to \mathbb{R}^p$$

Simple NN



activation function



Output$_j$= $\sigma(b_j + \sum w_{ij}x_i)$

$$h(x) = \sigma(b^2 + W^2\sigma(b^1 + W^1 x))$$

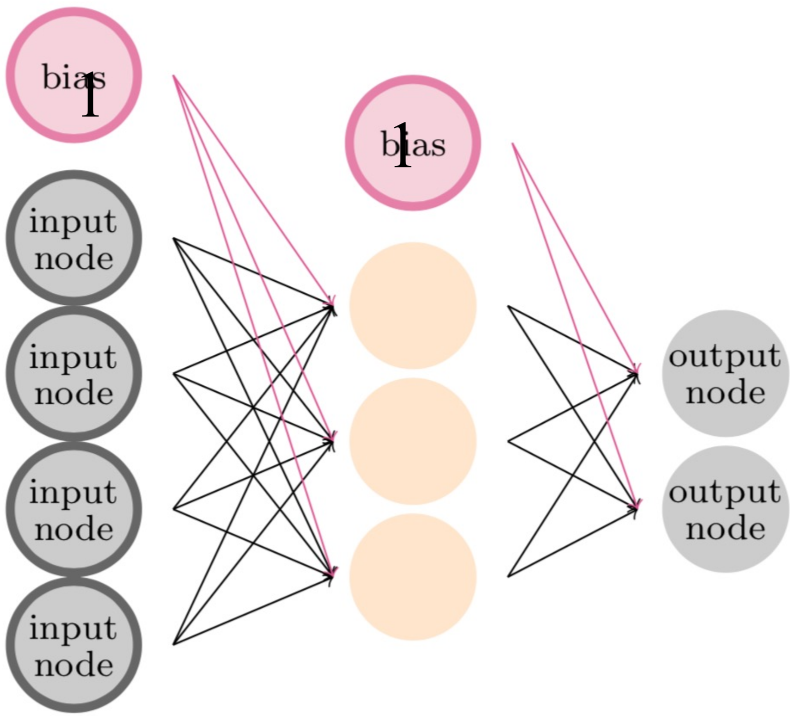
Beware: superscript are layer indices!

Now with dimensions

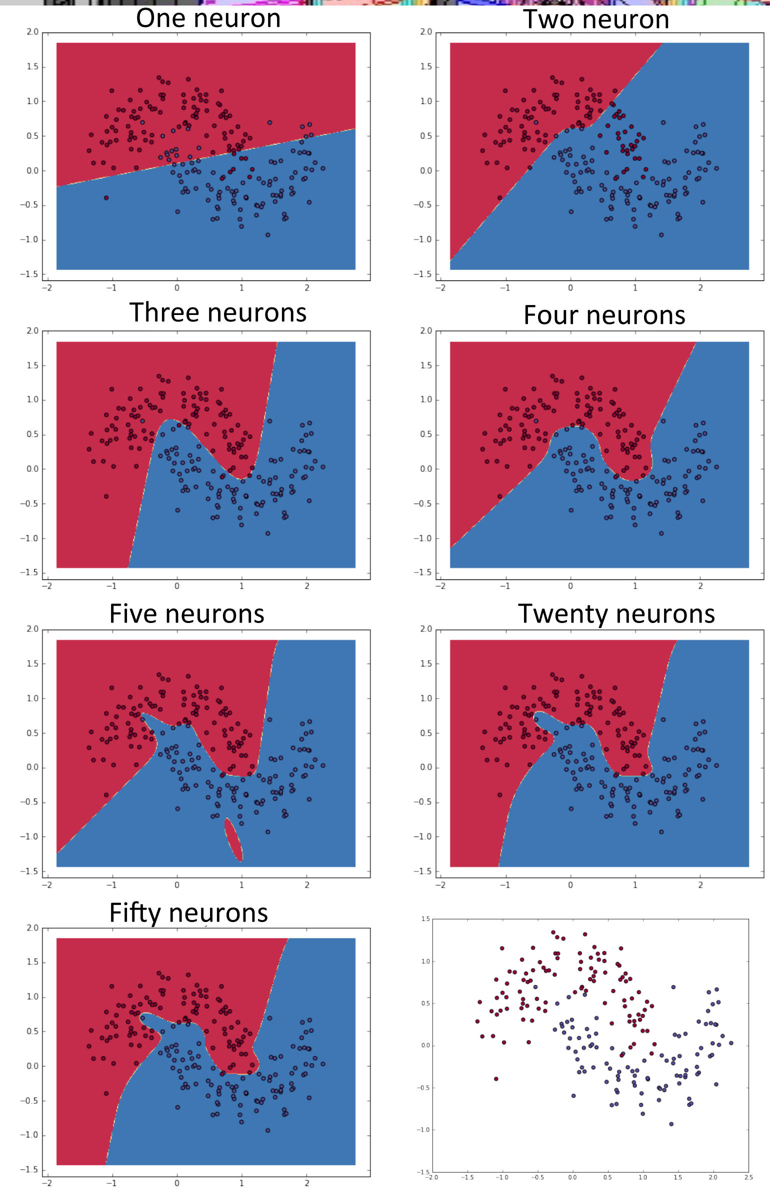
$$h(x_{(2)}) = \sigma(b^2_{(1)} + W^2_{(1,3)}\sigma(b^1_{(3)} + W^1_{(3,2)}x_{(2)}))$$

Bias

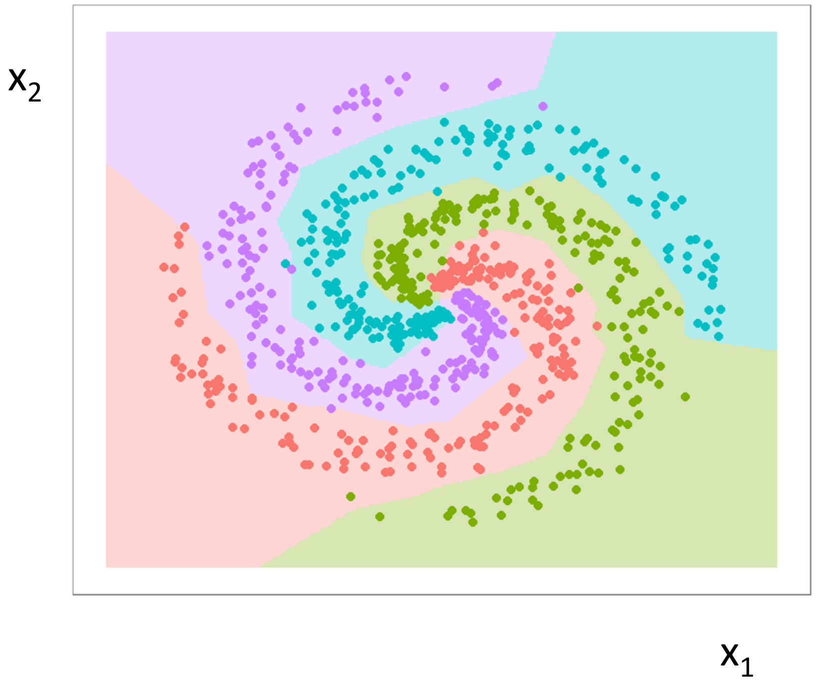
- Biases sometimes indicated as an additional node of value 1, and then can integrate the bias in the matrix of weight



NN at work



4-class classification
2-hidden layer NN
ReLU activations
L2 norm regularization



2-class classification
1-hidden layer NN
L2 norm regularization

http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

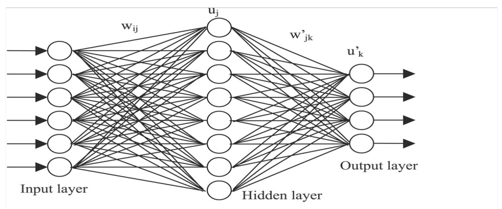
http://junma5.weebly.com/data-blog/build-your-own-neural-network-classifier-in-r

Loss function

Loss : regression

- Neural Network Model : h(w,x)
- Need to optimise the w (weights), so that h does what we want
- ➔define a « loss » function
- For a regression, typically quadratic loss function ~ $\frac{1}{2}\chi^2$

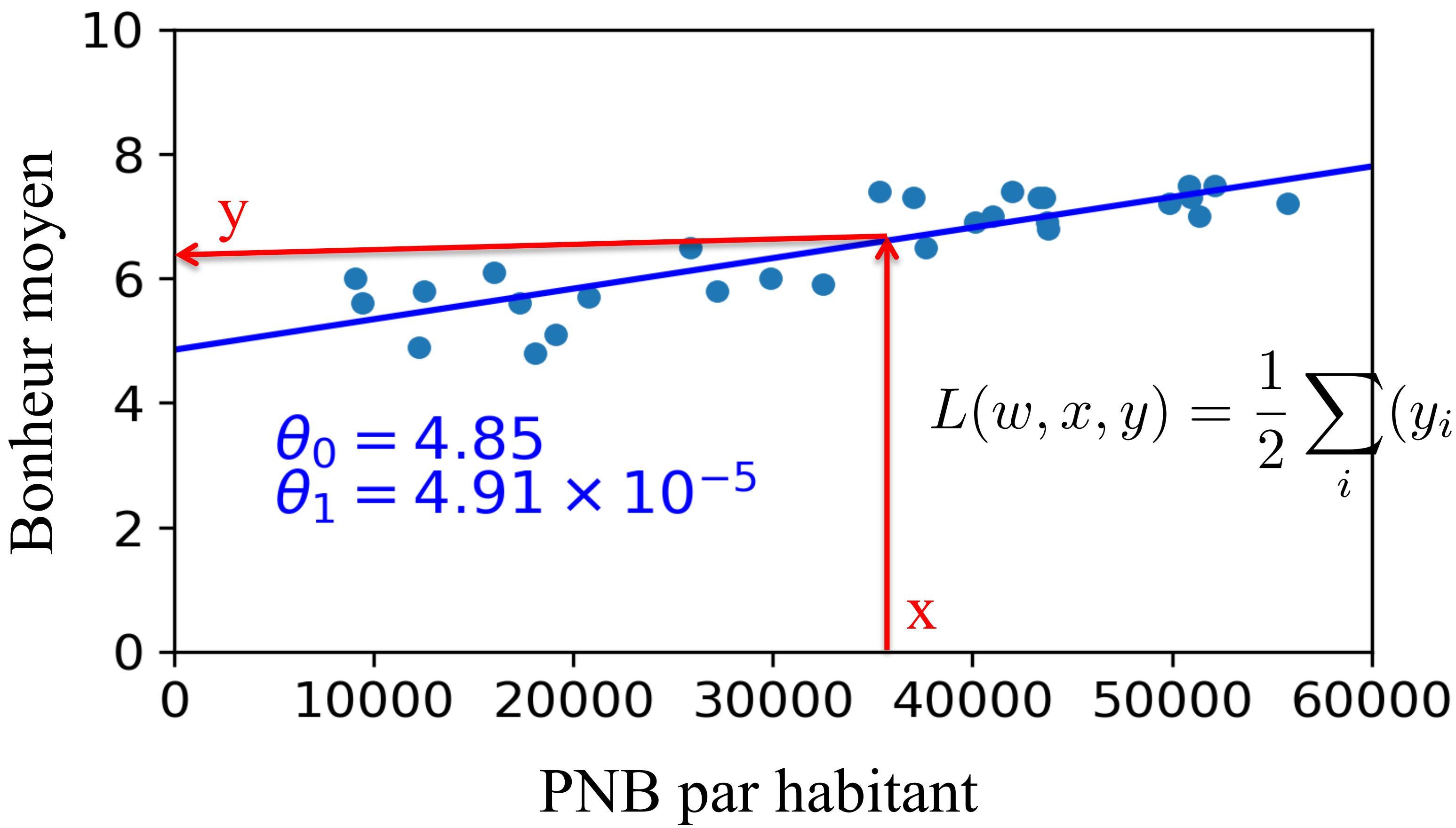
$$L(w,x) = \frac{1}{2}\sum_i (y_i - h(w,x_i))^2$$



- Beware notation :
  - o sum on all examples of the training dataset
  - o each y_i is a vector which length is the number of output variables to be regressed
  - o each x_i is a vector which length is the number of features

Recall : Linear Regression

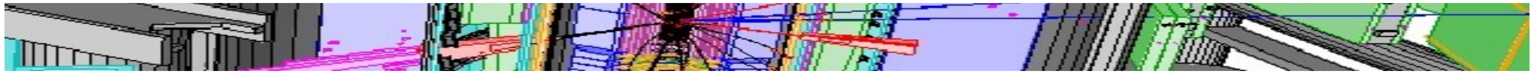
Boskovic, Legendre, Laplace, Gauss

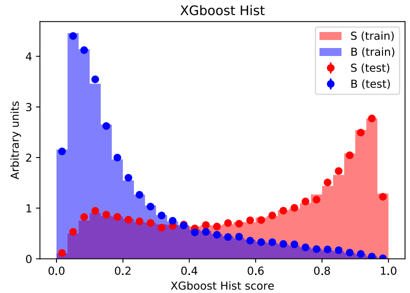


$$L(w, x, y) = \frac{1}{2} \sum_i (y_i - h(w, x_i))^2$$

y=f(x)

Loss : classification





- Desired answer (binary classification) : h(w,x) real close to y=1 for signal, close to y=0 for background
- Defining : $p_i = h(w, x_i) \in [0, 1]$
- The *cross-entropy* loss is:

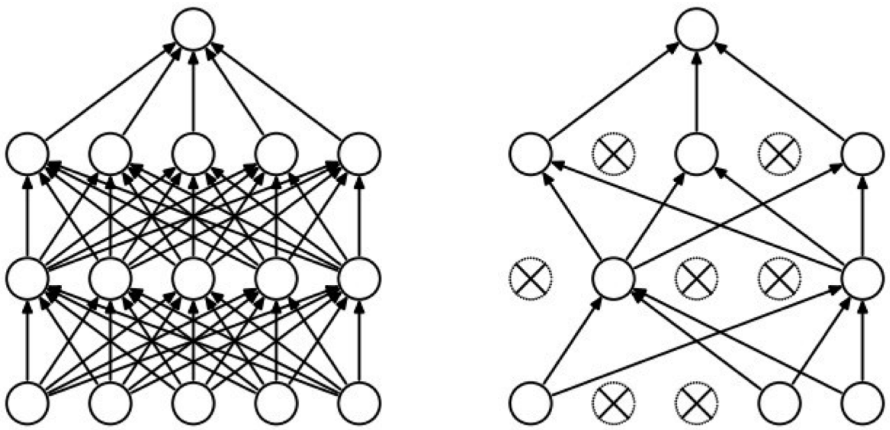
$$L(w, x, y) = -\sum_i y_i \ln p_i + (1 - y_i) \ln(1 - p_i)$$

$$L(w, x, y) = -\left(\sum_{\text{signal}} \ln p_i + \sum_{\text{background}} \ln(1 - p_i) \right)$$

Regularization

- L2 regularization: add $\Omega(\mathbf{w}) = ||\mathbf{w}||^2$ to loss
  - Also called "weight decay"
  - Gaussian prior on weights, keep weights from getting too large and saturating activation function

- Regularization inside network, example: **Dropout**
  - Randomly remove nodes during training
  - Avoid co-adaptation of nodes
  - Essentially a large model averaging procedure



(a) Standard Neural Net (b) After applying dropout.

arXiv:1207.0580

At this point

- We want the neural network model to do what we want on x $h(w, x)$
- We « just » need to find the correct values for w
- To do this, we'll minimise the loss function according to w $L(w, x, y)$
- …using x,y from the training sample, e.g:

```
df = pd.read_csv('assets/train.csv')
df.head()
```

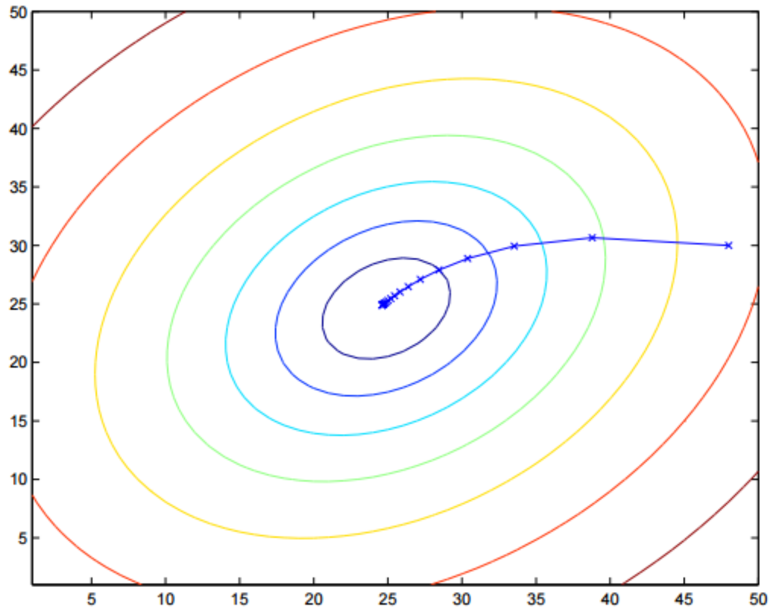
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Minimisation

BB

- Minimize loss by repeated gradient steps
  - Compute gradient w.r.t. parameters: $\dfrac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
  - Update parameters: $\mathbf{w}' \leftarrow \mathbf{w} - \eta \dfrac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$

Computing Hessian not practical!



Stochastic Gradient Descent

Minimising the cost function by gradients descent

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} - \gamma \nabla R(\vec{\theta}^{t})$$

If γ small enough, converge to a (possible local) minima

Stochastic (or « mini-batch") gradient descent

Compute the gradient by averaging the derivative of the loss in a **mini-batch**

1) Divide the training set into P batch of size B

2) For each batch, do

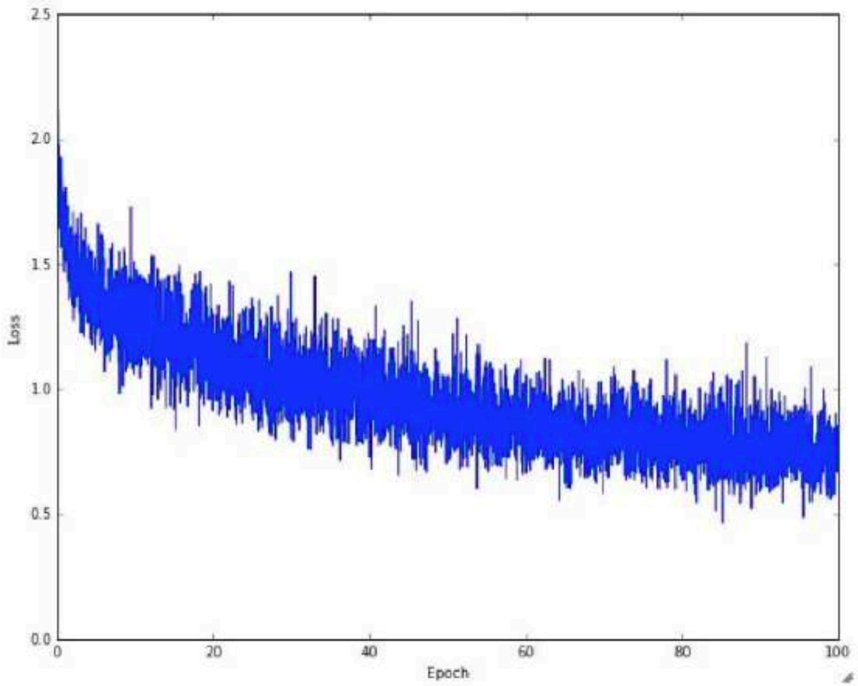
$$\vec{\theta}^{t+\frac{1}{P}} = \vec{\theta}^{t} - \gamma \sum_{i \text{ in mini batch}} \frac{1}{B} \nabla l(\vec{\theta}^{t}; \vec{x}_i, y_i)$$

3) One « epoch » (t->t+1) means running the algorithm through all mini-batches

SGD (2)

Mini-batch gradient descent

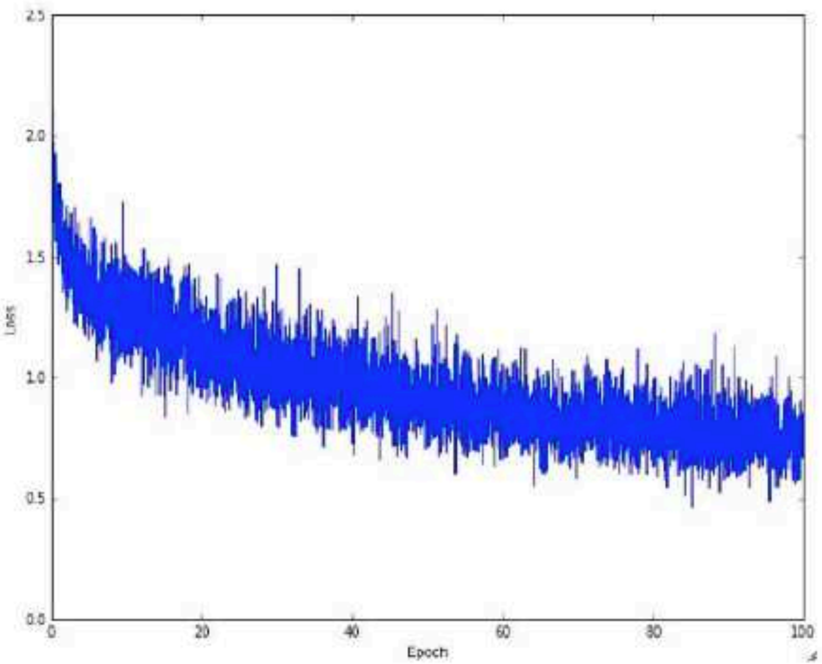
- Example of optimization progress while training a neural network
- Showing loss over mini-batches as it goes down over time

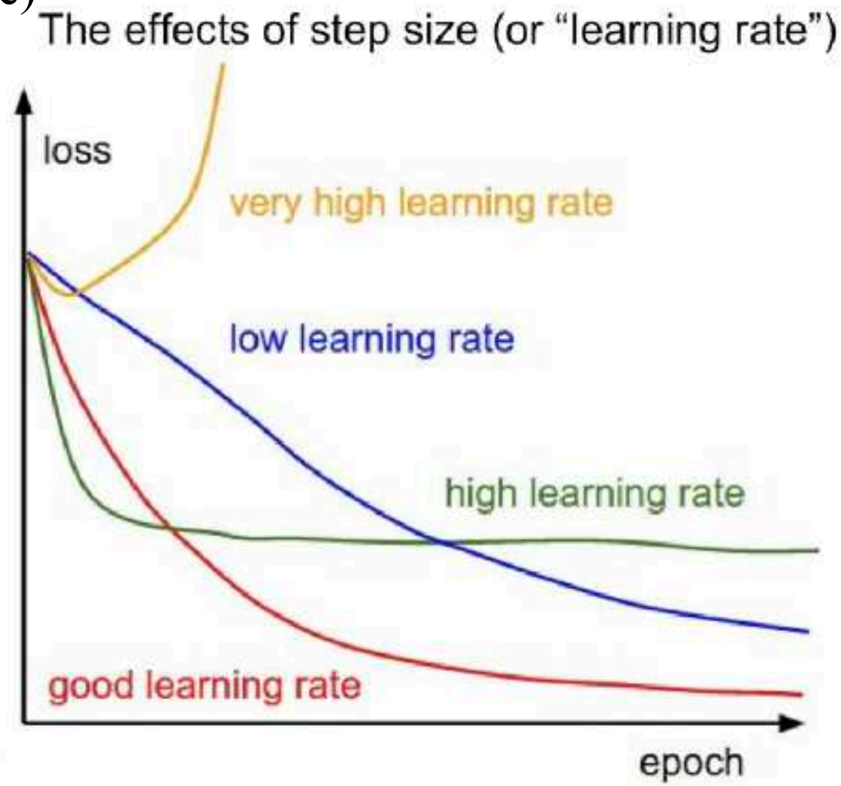


SGD(3)

- Example of optimization progress while training a neural network
- Epoch = one full pass of the training dataset through the network

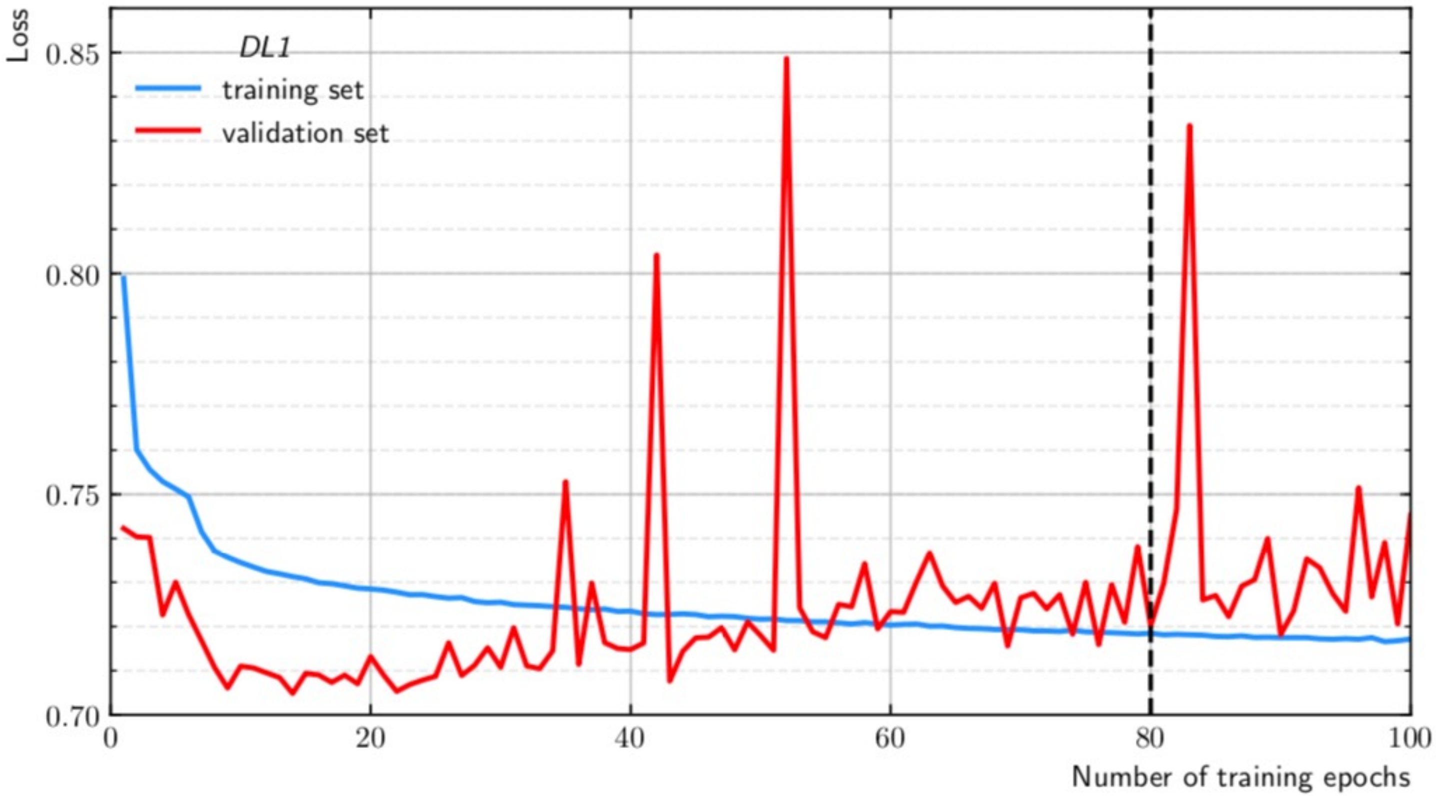
(Advanced algorithms have adaptive learning rate)





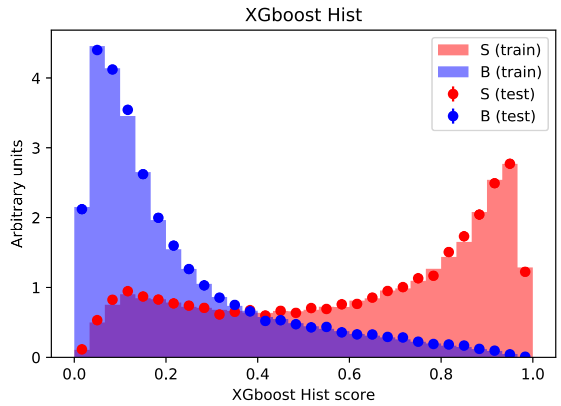
Tricky Loss

- Loss is minimised but can be tricky
- One need to look at performance plot (AUC) to chose the epoch

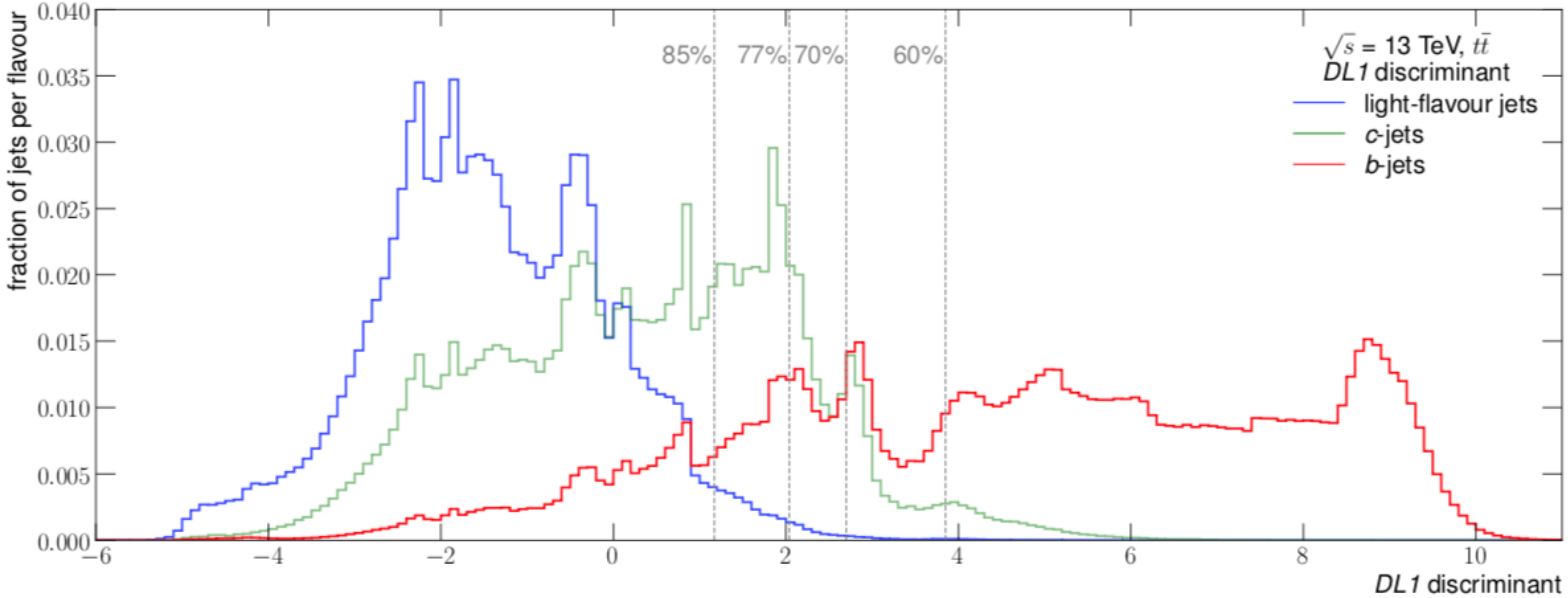


Tricky Score distributions

- Remember BDT score



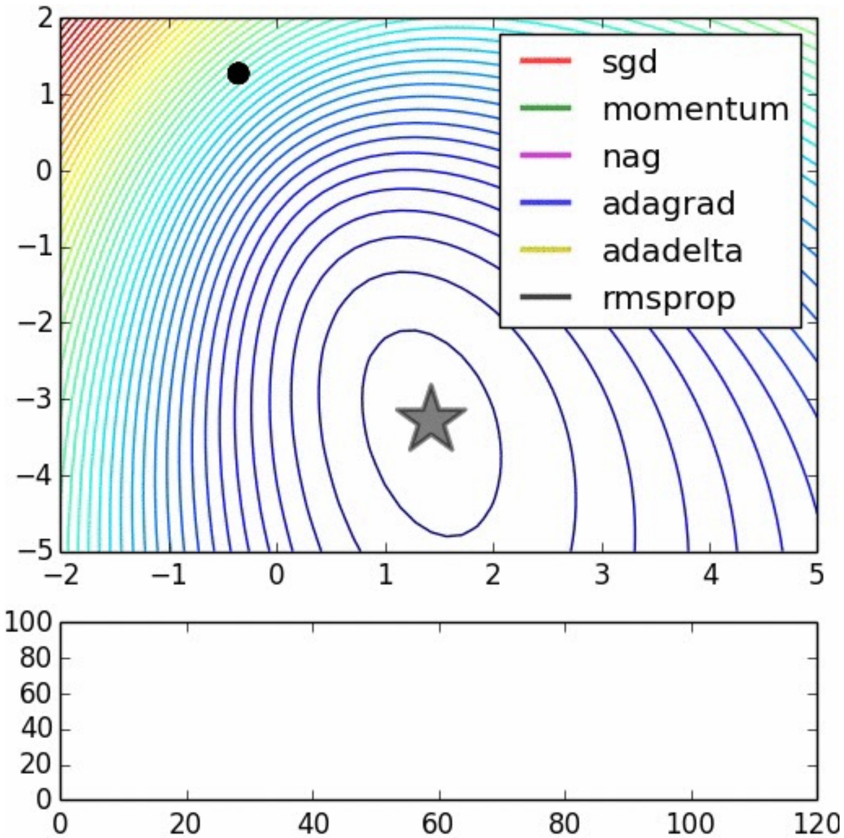
- Score distribution can be much more tricky:

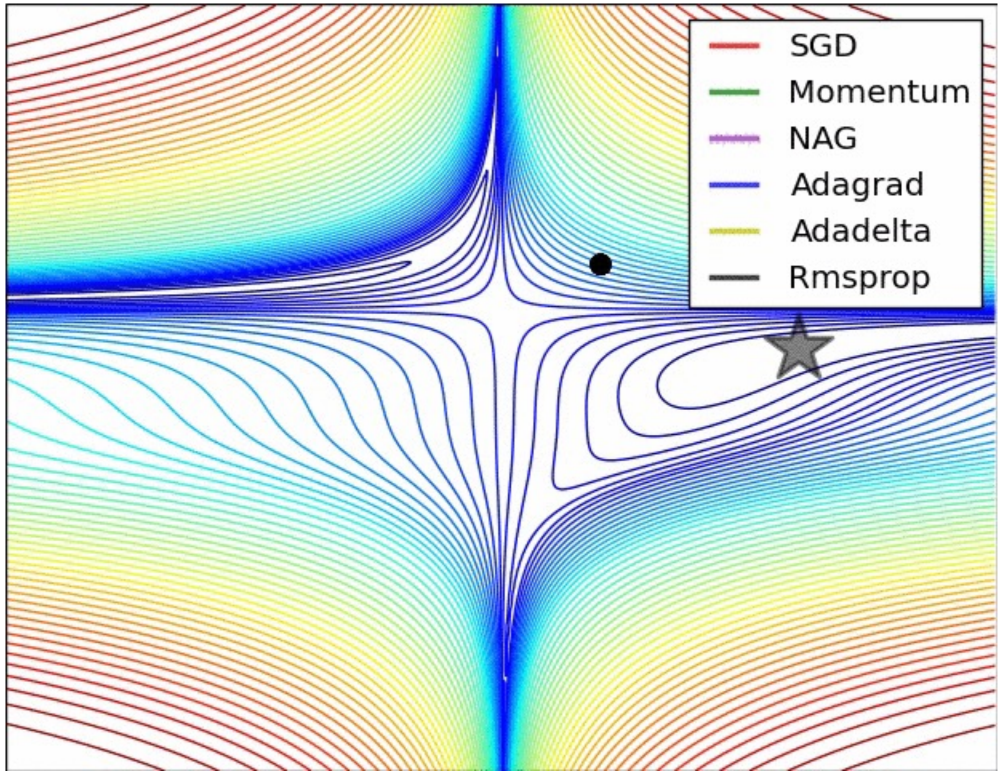


Optimisation

- Up to trillion of parameters to optimise….
- Wealth of newish algorithms in particular Stochastic Gradient Descent (SGD) and more

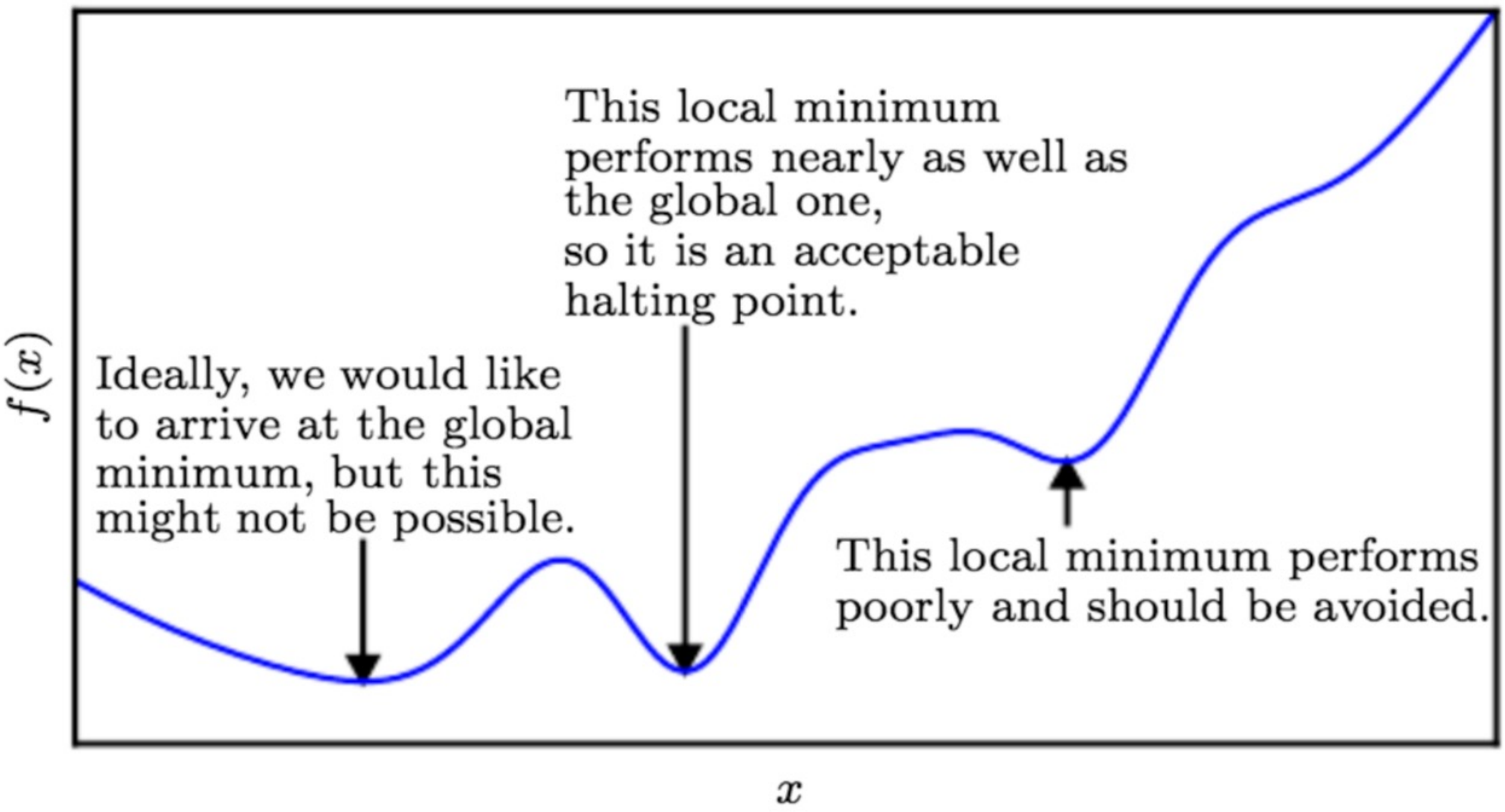
Alec Radford



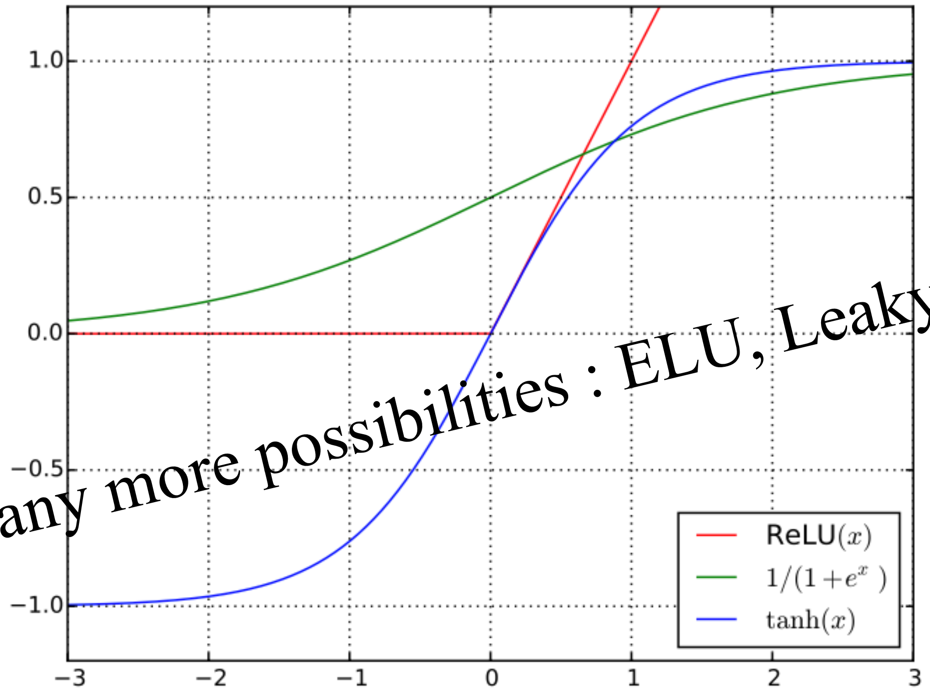


Addendum on optimisation

- Many minima but….



More on activation functions



Many more possibilities : ELU, LeakyRelU, Swish…

XXIth century

~classic XXth century

- **Vanishing gradient problem**
  - Derivative of sigmoid:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

  - Nearly 0 when x is far from 0!
  - Gradient descent difficult!

- **Rectified Linear Unit (ReLU)**
  - ReLU(x) = max{0, x}
  - Derivative is constant!

$$\frac{\partial \mathrm{Re}LU(x)}{\partial x} = \begin{cases} 1 & when\ x > 0 \\ 0 & otherwise \end{cases}$$

  - ReLU gradient doesn't vanish

Neural Network software

- Essentially TensorFlow (Google) and PyTorch (Facebook)
- Free Open Source software
- Python, interface nicely to numpy arrays and pandas dataframe
  - Heavy duty done in C
- Define and train a NN in a few lines
- Uses GPU if available (performance boost 5-20)
- Run on laptop as well as supercomputers
- In general, NN more complex and heavy to train than Boosted Decision Tree

NN Hyper-Parameter Optimisation

- NN optimising much more complex than BDT, plus much slower to train
  - ==>always start with BDT if not dealing with images
  - Access to computing resources
- Incomplete list of HPO (for dense NN)
  - Width (Start with anything between like 32 to 128)
  - Depth (start with 1)
  - Epochs : track "validation loss" to decide this but often the significance might improve even though the loss does not
  - Batch Size (default is 32)
  - Activation (Start with Relu/LeakyRelu)
  - Early Stopping (Start with it off)
  - Optimiser (Start with Adam (momentum + learning rate manipulation))
  - Learning Rate (Default already present in optimiser, try to lower it by orders of mag)
  - Drop Out (Start with it off)
  - Batch Norm (Start with Off)
  - Etc...etc....