Cockcroft Institute Lectures – Register Attendance

QR code will take you to a Google form.

Record attendance for each day of lectures.

UNIVERSITY OF LIVERPOOL

Department of Physics

INTRODUCTION TO NEURAL NETWORKS II

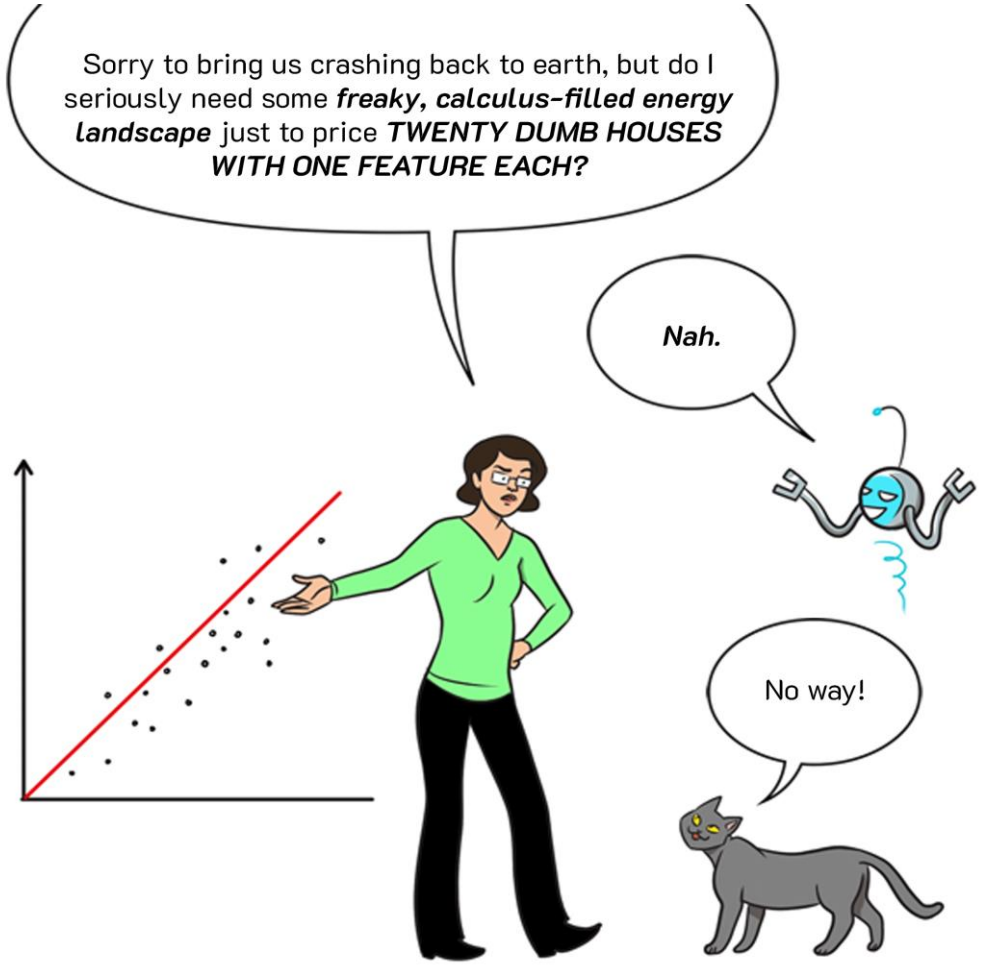
Dr. Andrea Santamaria Garcia
Lecturer

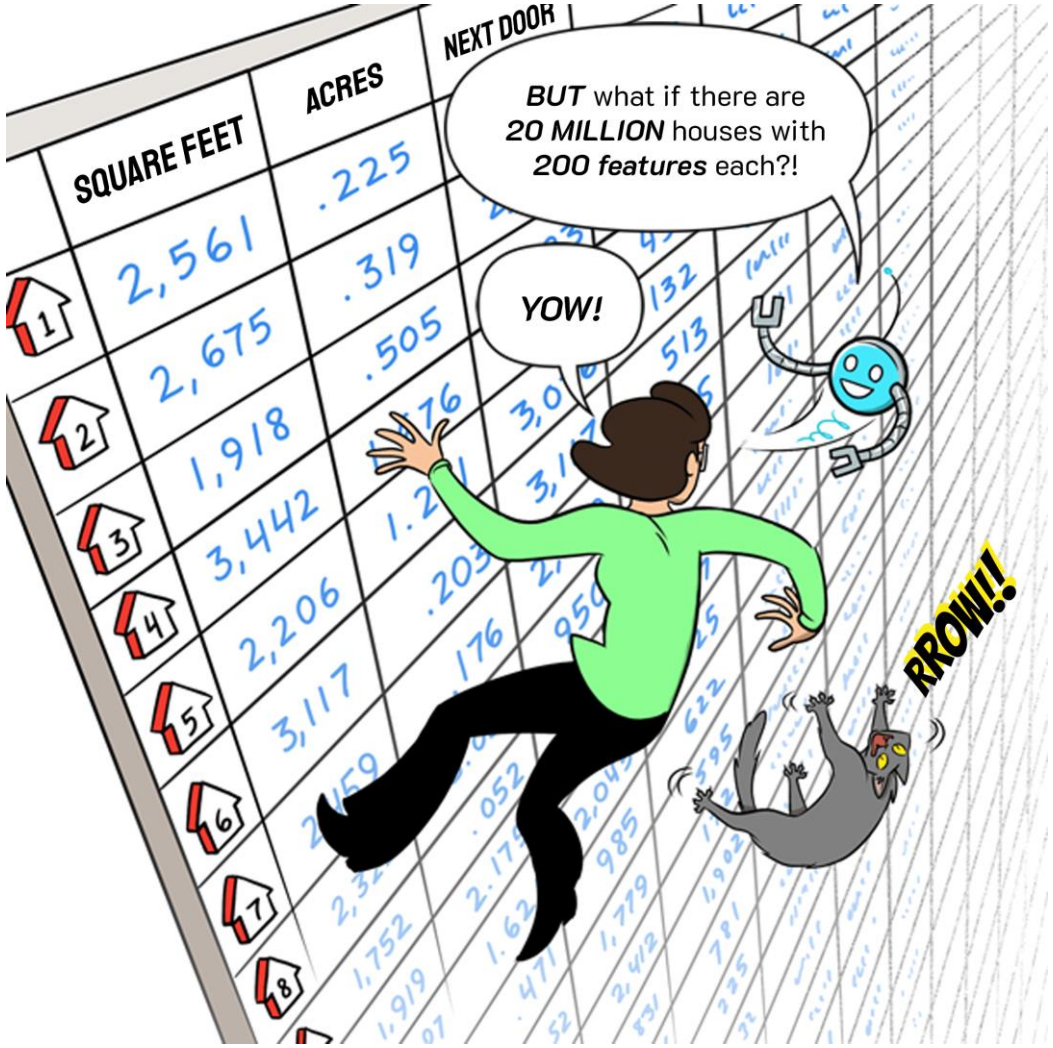
CI lectures CI-ACC-226
Lecture on 23/02/2026

OUTLINE

- **Stochastic gradient descent** (3 slides)
- **Activation functions** (4 slides)
- **Overfitting** (1 slide)
- **Regularisation** (1 slide)
- **Training stages in supervised learning** (5 slides)
- **Deep neural networks** (6 slides)
- **Practice exercise**

NNs: computationally efficient for big data!





Comic source: https://cloud.google.com/products/ai/ml-comic-1

Recap from previous lecture

Neural networks:

- are **powerful function approximators** that scale efficiently to large datasets
- typically require **large amounts of high-quality and diverse training data** for strong performance
- are part of **narrow AI**: they are trained for specific objectives rather than possessing general intelligence
- are used in **supervised** and **unsupervised learning**
- are parameterized by **weights and biases**, optimised by minimising a **loss function**
- are trained using **first-order gradient methods** (e.g., gradient descent)
- compute gradients via **backpropagation**, which applies the multivariate chain rule to a composition of functions

Important limitation of gradient descent

Computational cost per model update

$$\sum_{i=1}^{n} (h_w(x_i) - y_i)^2 = J(W)$$

n terms (data points)

$$W \leftarrow W - \alpha \nabla J(W)$$

Cost per data point scales with number of parameters p

In full-batch gradient descent, we **aggregate gradient contributions from all data points** and then perform a **single weight update** (one update step uses information from every training example)

Full batch cost per weight update scales as $\mathcal{O}(n\,p)$
Full batch requires holding full dataset in memory

Let's consider a small example:

- **2000 data points**
- 5-64-64-2 network (**4674 parameters**)

~9.3 million computations for one model update

calculate the gradient using just a random small part of the observations instead of all of them

Stochastic gradient descent

Stochastic gradient descent

In stochastic gradient descent (SGD) the gradient is approximated by a gradient at a single sample:

repeat for several epochs until convergence

Randomly shuffle samples in the data set

for $i = 1, \ldots, n$ do:

$$W \leftarrow W - \alpha \nabla \ell_i(W)$$

Why is this a good idea, computational efficiency aside?

- In SGD we **approximate the full gradient** using a single-sample gradient $\nabla \ell_i(W)$
- This produces a **noisy but unbiased estimate** of the **true gradient**
- The **stochasticity introduces variability**, which can help escape saddle points and sometimes improve generalisation

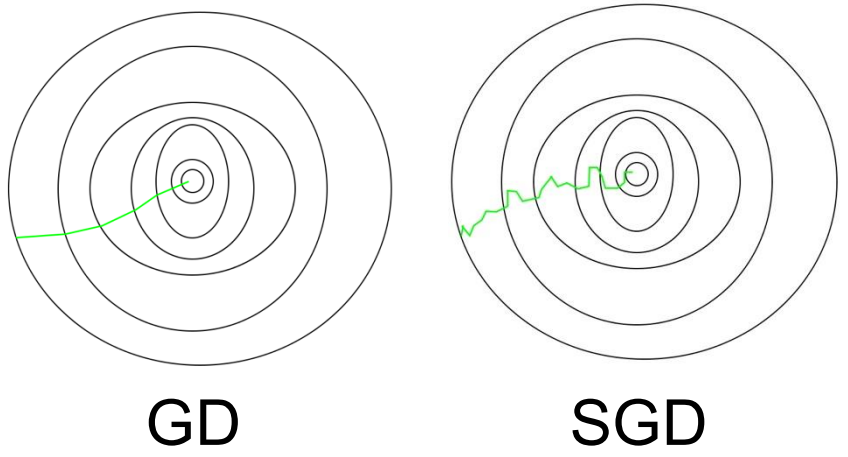


Image from https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

Mini-batch SGD

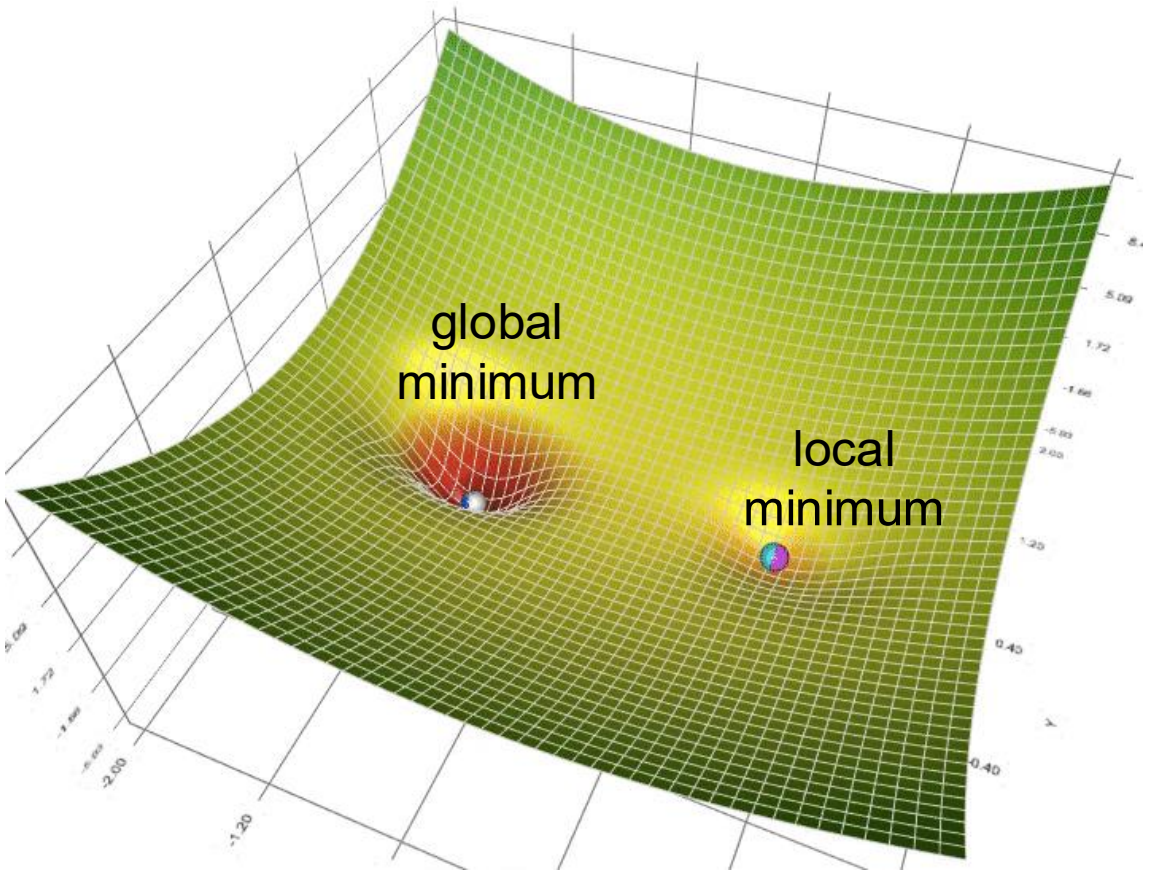
a compromise between GD and SGD

The dataset can be sliced in random mini-batches that are mutually independent.

Several passes can be made over the training set until the algorithm converges.

➢ A new hyperparameter to tune appears: the **minibatch size**.

Other optimisers vs gradient descent



Animation from Lili Jiang

Gradient based methods (first order):

- Gradient descent
- momentum
- AdaGrad
- RMSProp
- Adam

} SGD with adaptive learning rates

Second order gradient methods:

- They provide information about the curvature of the loss function (e.g. Newton's method)
- Complex and difficult to implement
- Expensive in iteration cost and memory occupation
- Active area of research

Gradient-free methods:

- Genetic algorithms, particle swarm optimisation
- "Neuroevolution"

Activation functions

The choice of activation function has a large impact on the capability and performance of the neural network!

They manage the flow of data through the network by activating or deactivating neurons based on their output

Typically:

- **Hidden layers** use the same activation function.
- The **output layer** will use a **different activation function** since is dependent upon the type of prediction required by the model.
- **Activation functions** are (almost everywhere) **differentiable** allowing gradient-based optimisation

Hidden layers

- **ReLU** (most feedforward and deep networks)
- **Tanh** (common in RNNs)
- **Sigmoid** (gated architectures)

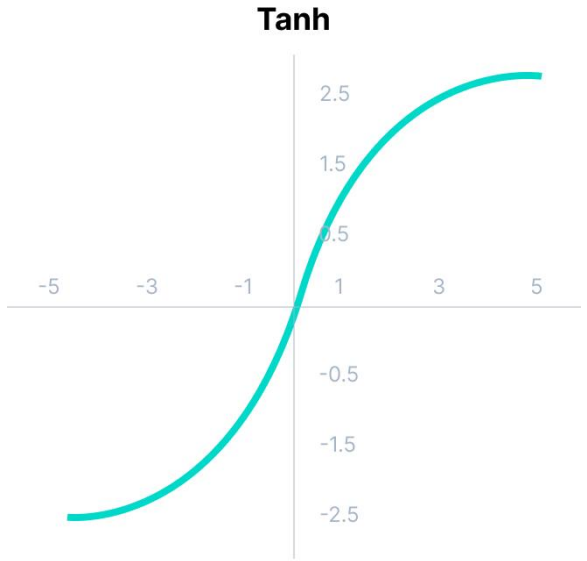
Output layers

- **Linear** (regression)
- **Sigmoid** (binary classification)
- **Softmax** (multiclass classification)

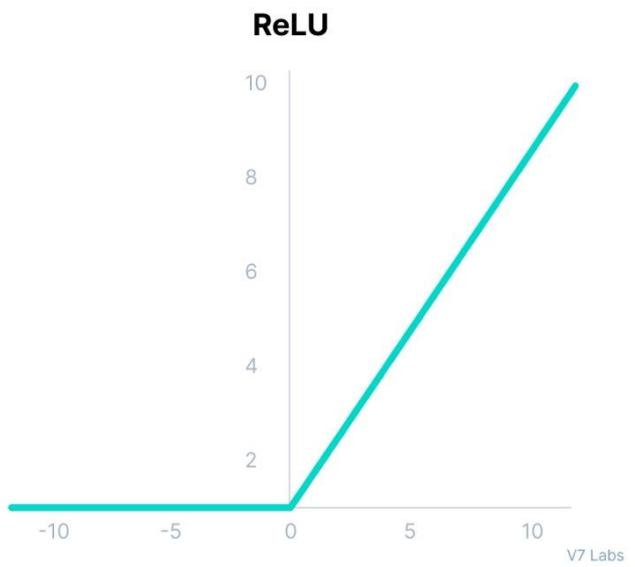
Activation functions

The choice of activation function has a large impact on the capability and performance of the neural network!

- Maps input to (0,1), useful for probabilities
- Saturates for large positive and negative inputs
- Not zero-centered (output always positive)



- Maps input to (-1,1)
- Saturates for large input
- Zero-centered, which helps gradient-based optimisation

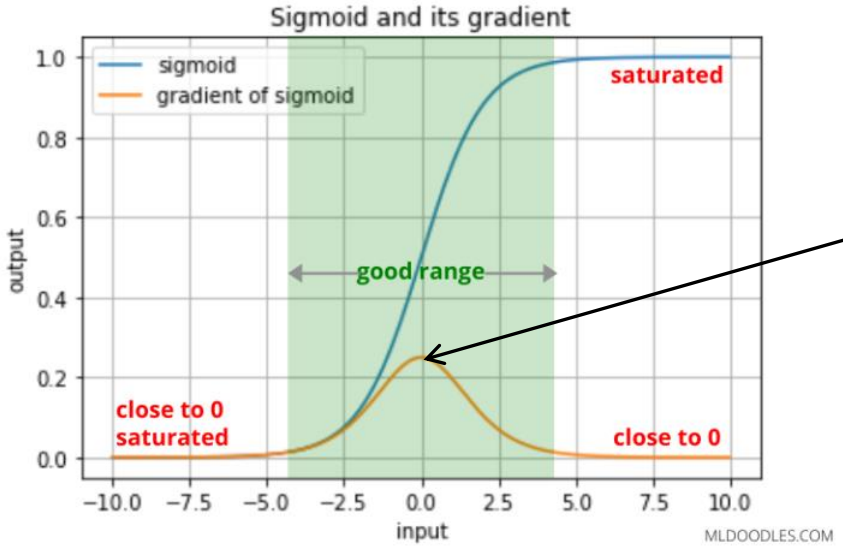


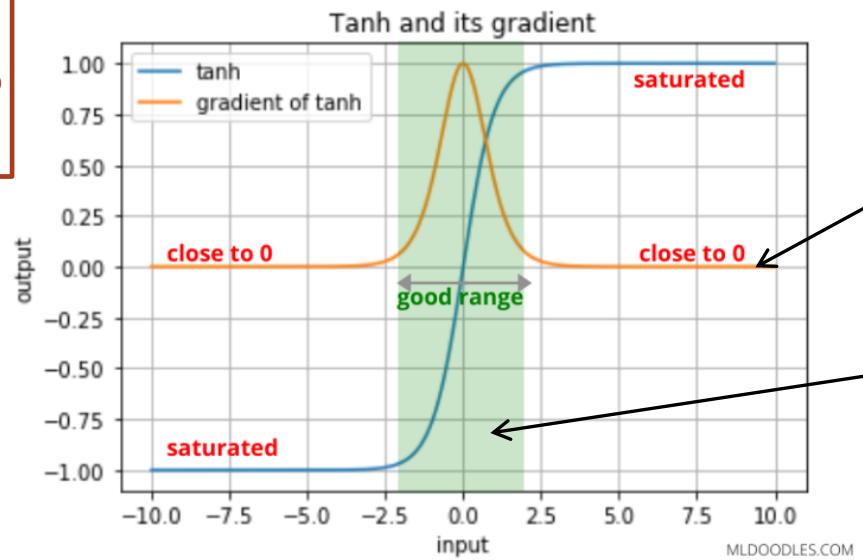
- Non-saturating for positive inputs
- Computationally efficient (simple max operation)
- Gradient is zero for negative inputs, which can create dead neurons that never get activated

Vanishing gradients

appear in backpropagation using gradient-based methods in deep networks

not goodin hidden layers





Maximum of gradient 0.25
With chain rule the gradient product can become very small

$$\frac{\partial J(\mathrm{w})}{\partial \mathrm{w}^{(0)}} = \frac{\partial J(\mathrm{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{(1)}} \cdots \frac{\partial a^{(l)}}{\partial \mathrm{w}^{(0)}}$$

$$0.2 \times 0.15 \times 0.22 \times 0.09 \ldots$$

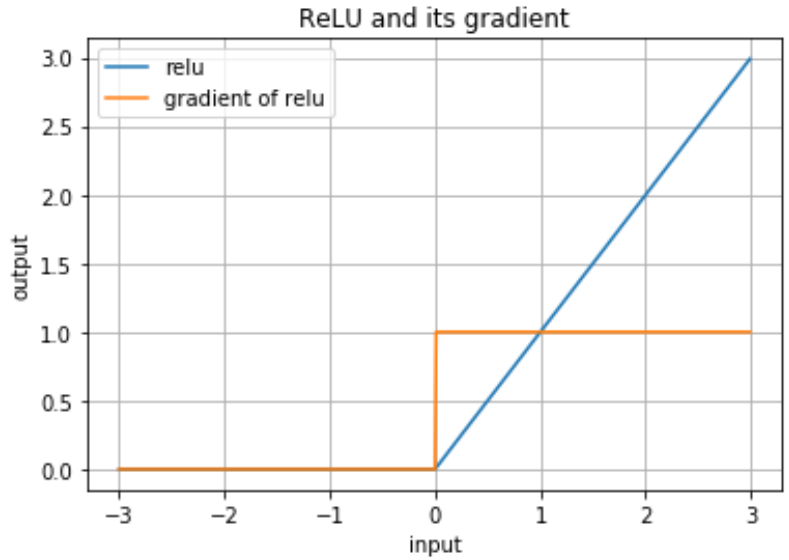
No response to changes in input, earlier layers receive almost no learning signal

Very narrow range, small values

When the partial derivative vanishes the weights are not updated anymore

$$W \leftarrow W - \alpha \nabla J(W)$$

good for hidden layers



Images from mldoodles

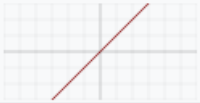
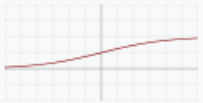
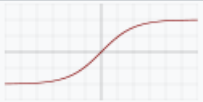
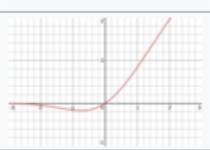
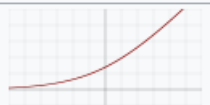
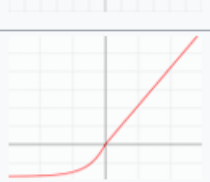
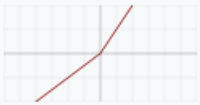
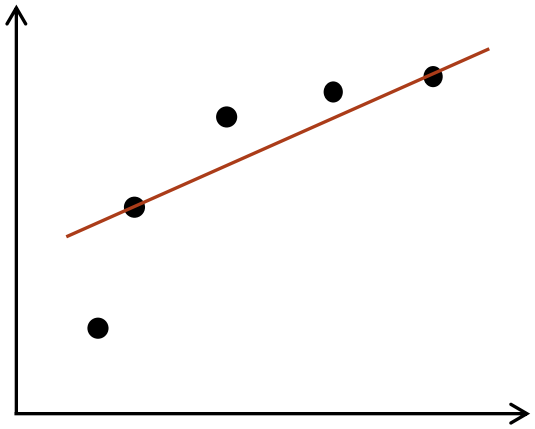
Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$
Identity		x	1
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1-g(x)^2$
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0,x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$
Softplus[9]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha\left(e^x-1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$
Scaled exponential linear unit (SELU)[11]		$\lambda\begin{cases} \alpha(e^x-1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[12]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$

Image from wikipedia

Overfitting

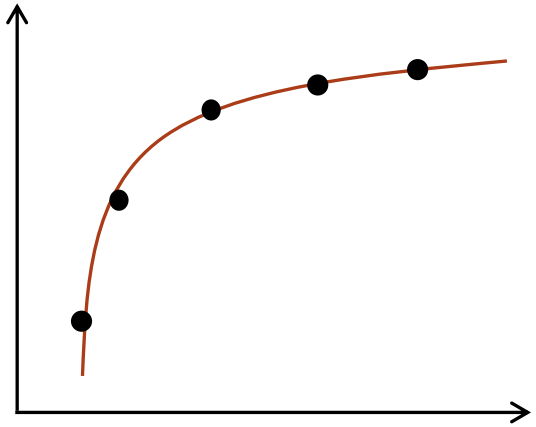
- Overfitting occurs when the model fits the training data too closely, **capturing noise rather than underlying structure**, leading to poor generalisation
- This happens due to:
  - Excessive **model capacity** relative to the **amount of data** (e.g. deep neural networks)
  - **Training for too long**, capturing noise instead of genuine patterns

Underfit



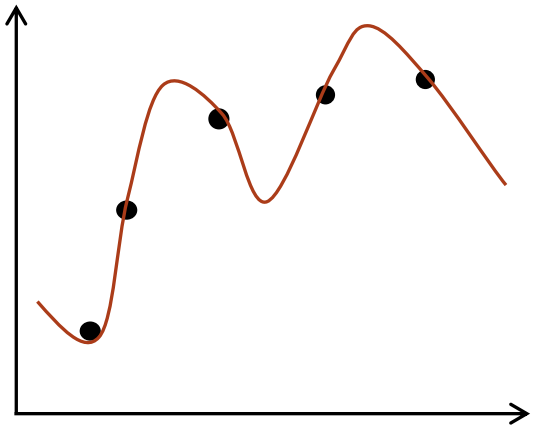
$\hat{y} = w_0 + w_1 x$

Good fit



$\hat{y} = w_0 + w_1 x + w_2 x^2$

Overfit



$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

Regularisation

Improves generalization on useen data by constraining the optimisation problem to discourage complex models

Early stopping

- Monitor validation loss
- Stop training when it stops improving
- Prevents overfitting by limiting effective model capacity

Weight regularisation

- Adds penalty term to the loss function
- Encourages smaller weights
- Reduces model complexity and sensitivity

Dropout

- Randomly deactivate neurons during training (ensemble of subnetworks)
- Forces the network to rely on distributed representations
- All neurons are used at test time with scaled weights

Stages in supervised learning

Training Phase

Learns the basic mapping between input and output
You can develop several models

- Use a labelled dataset (x, y) called the training set
- Compute predictions using current parameters
- Evaluate a loss function J(W) that measures prediction error
- Compute gradients via backpropagation
- Update parameters using an optimisation algorithm
- Repeat for multiple epochs

Training set

60-80% of data
Curated
Representative
“Gold standard”

Stages in supervised learning

Validation Phase

Select best performing model or approach

- Evaluate model on unseen validation data
- Used for model selection and hyperparameter tuning
- Used to decide when to stop training (early stopping)
- Not used to train parameters directly

Training set

10-20% of data
Real-world data
Data of your study

Testing Phase:

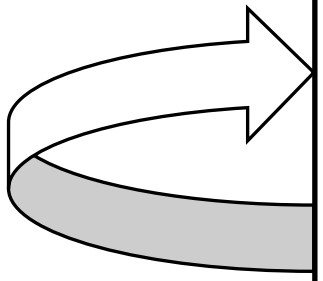
Evaluation of final model performance

- Final model performance on unseen data
- Not used for training or tuning
- Provides an unbiased estimate of generalisation performance

Training set

10-20% of data
Real-world data
Data of your study

Summary: how to train a neural network



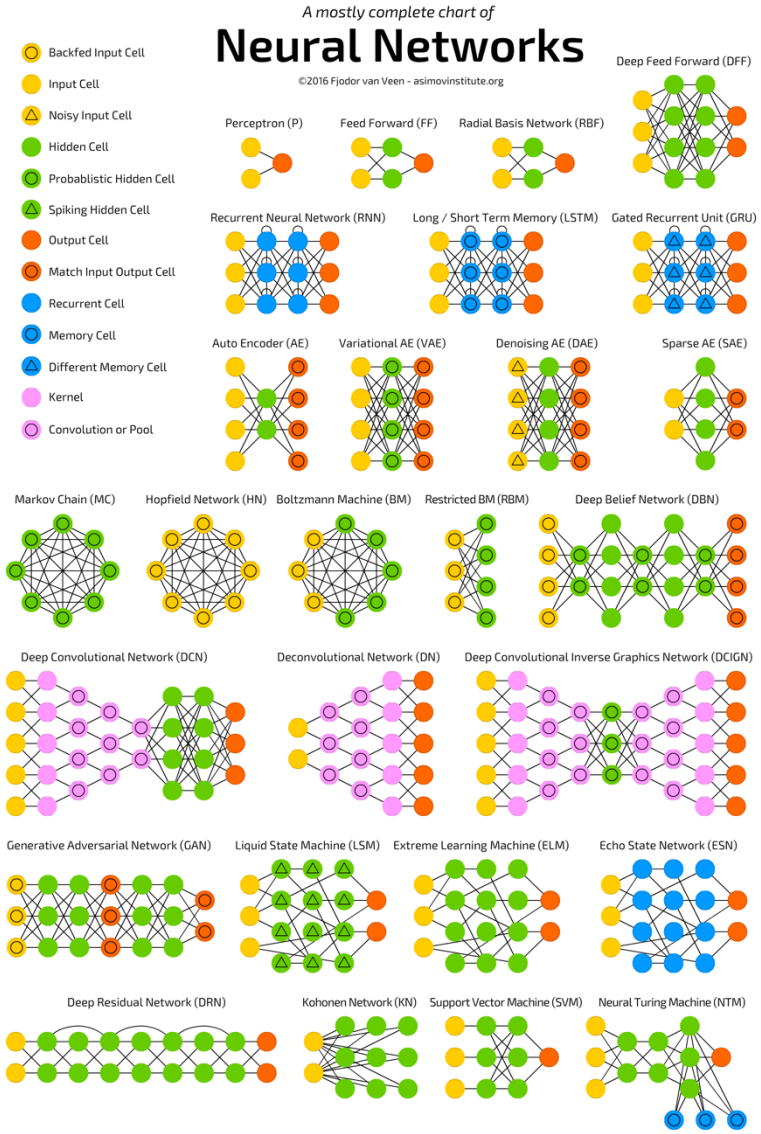
1. Select data features + perform data scaling
2. Choose network architecture
3. Choose activation functions
4. Choose loss function
5. Choose an optimiser & set its hyperparameters
6. Choose stopping criterion (epochs or early stopping)
7. Choose regularisation techniques
8. Forward pass
9. Backward pass (compute gradients)
10. Update weights & keep track of loss
11. Evaluate the model's performance on validation/test

Further considerations

- **Error analysis**: Identify systematic failure modes and data regimes where performance degrades
- **Statistical testing**: Are observed performance differences statistically significant?
- **Robustness and generalisation**: How does the model behave under noise, perturbations, or distribution shift?
- **Real-world performance**: Evaluate under distribution shift and real deployment conditions
- **Resource utilisation**: assess the model's resource usage, like inference time, memory footprint, and power consumption.

A SNEAK PEAK OF MORE ADVANCED CONCEPTS: DEEP NEURAL NETWORKS

Neural network architectures



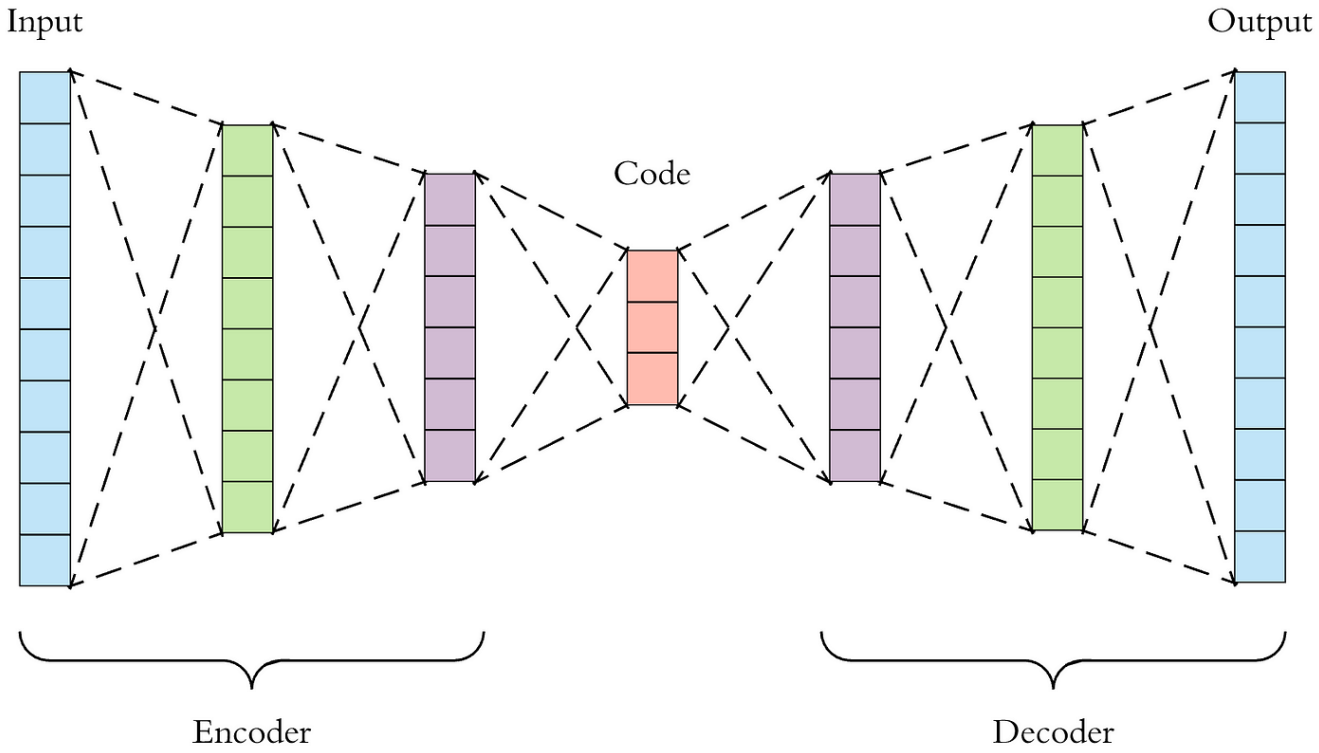
- Until now we only looked at the simple feed-forward, fully-connected case.
- Neural networks can have very different architectures.

Fjordo van Veen, 2016

Autoencoder (AE)

- Feed-forward structure, used for different purposes
- **Encodes** the input to low-dimensional latent space, and **decodes** to the original shape

Use cases
- Feature extraction
- Denoising input data
- Generative models



https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Convolutional Neural Network (CNN)

- Used for **image-related** tasks
- Applies spatial convolutions to the input

- Translation invariance
- **Hierarchical features**: deeper layers detect more complicated features
- Less parameters needed than fully-connected structure

LeNet-5 structure, Y. Lecun (1998). 61k parameters

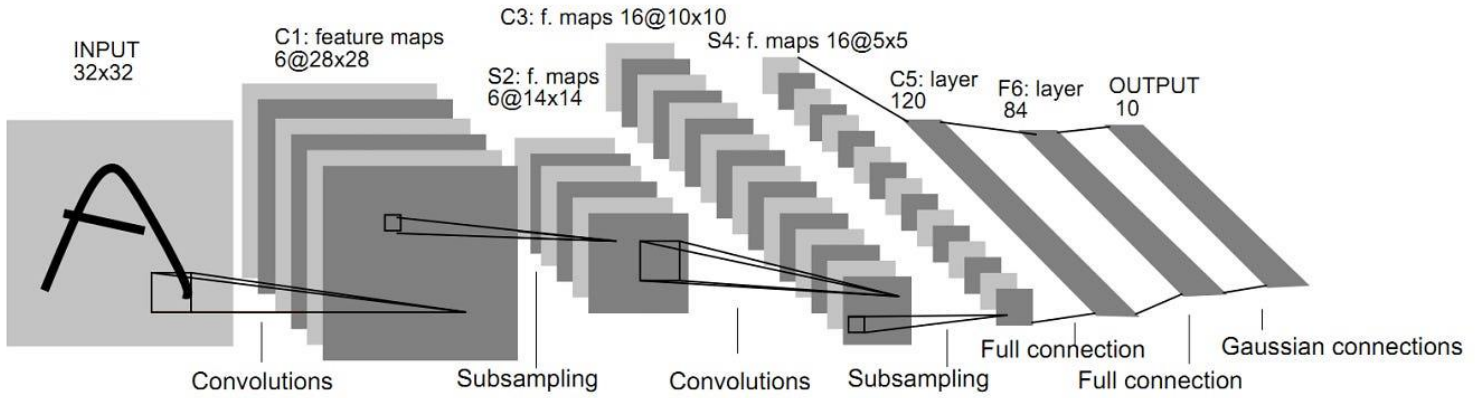
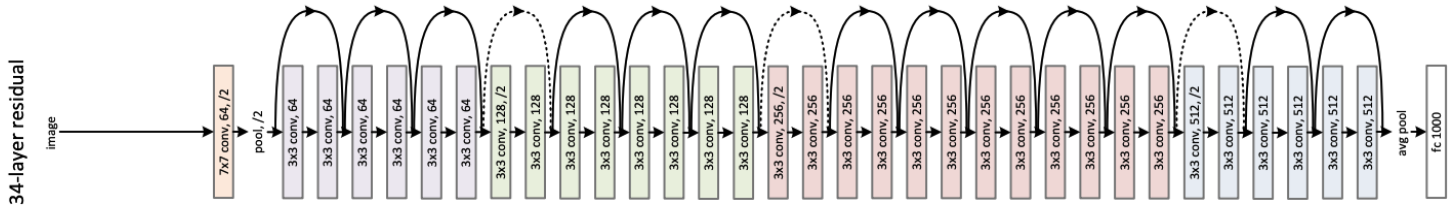


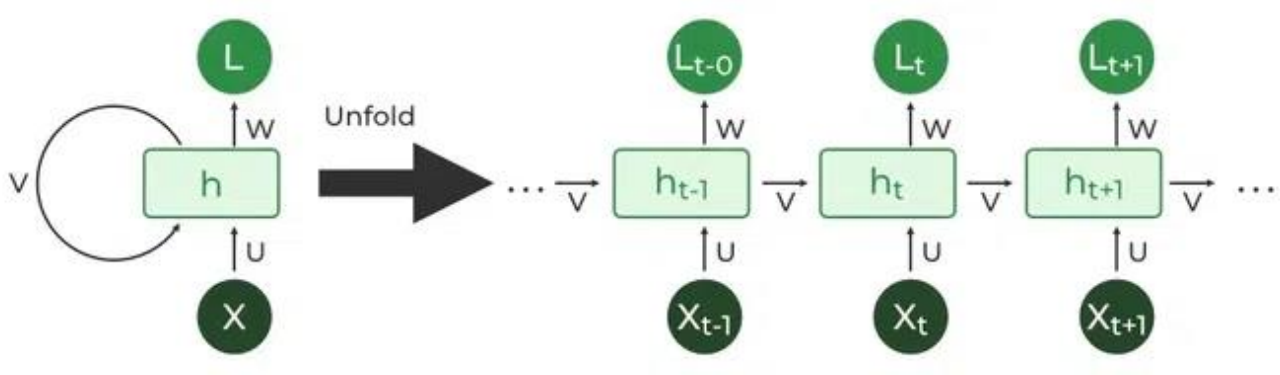
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

ResNet-50, K. He (2015). 25M parameters

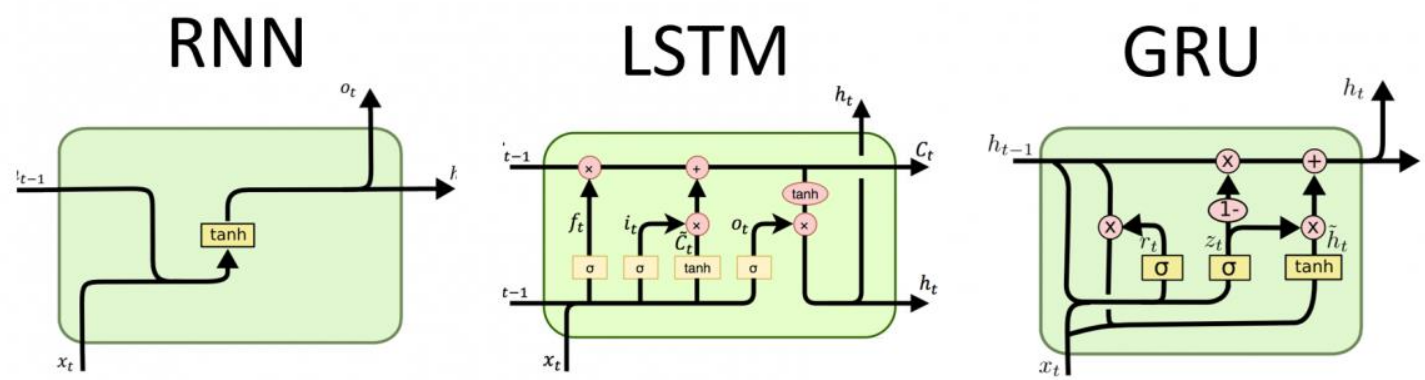


Recurrent Neural Network (RNN)

- Used in **time-series** data.
  e.g. Natural language processing (NLP), forecasting.
- Contains hidden state variables (memory, context).
- Allows variational lengths of inputs and outputs.



https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

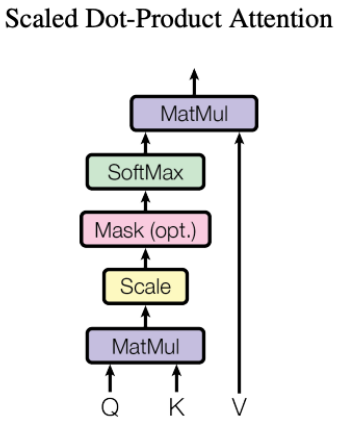


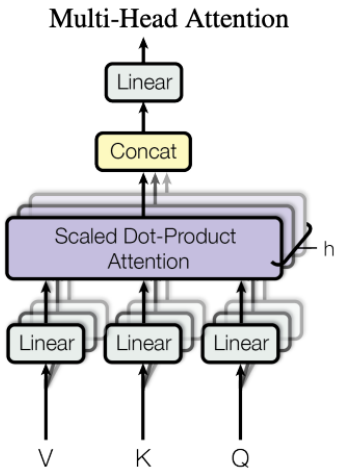
http://dprogrammer.org/rnn-lstm-gru

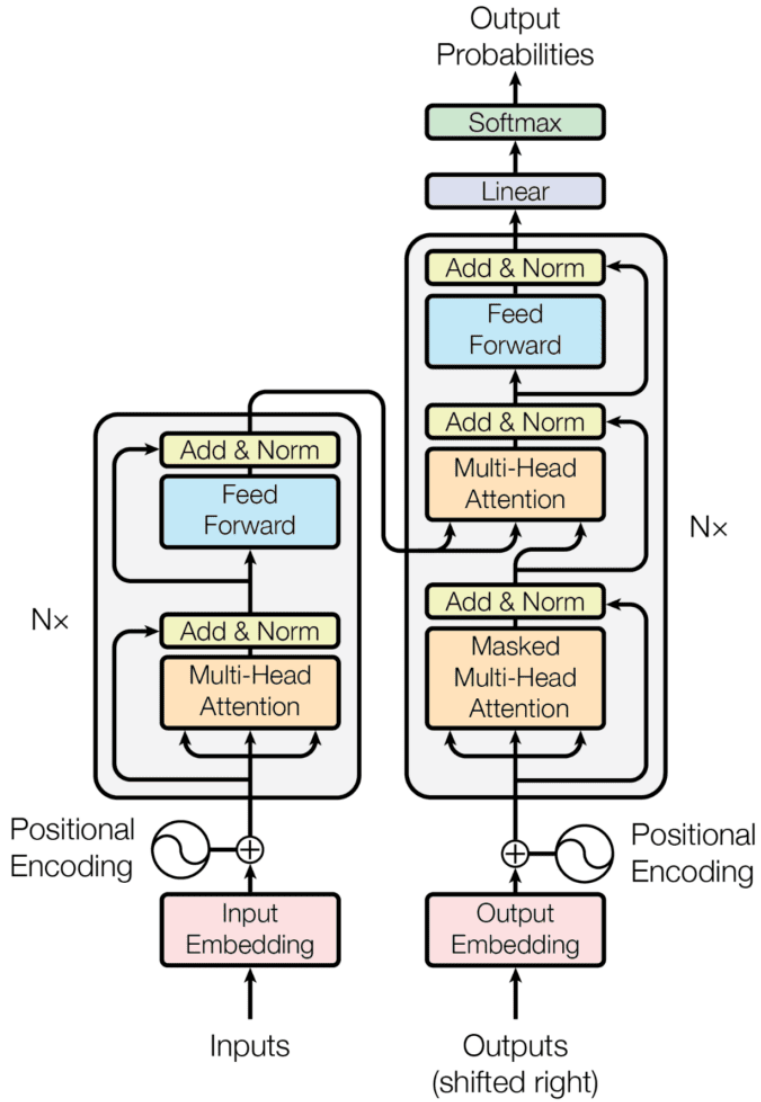
Transformer

Attention is all you need, 2017

- Probably the most successful NN structure nowadays
- **Architecture behind the LLMs (ChatGPT,...)**
- Use only the *attention* mechanism without the recurrent structure
- Parallelizable → faster training



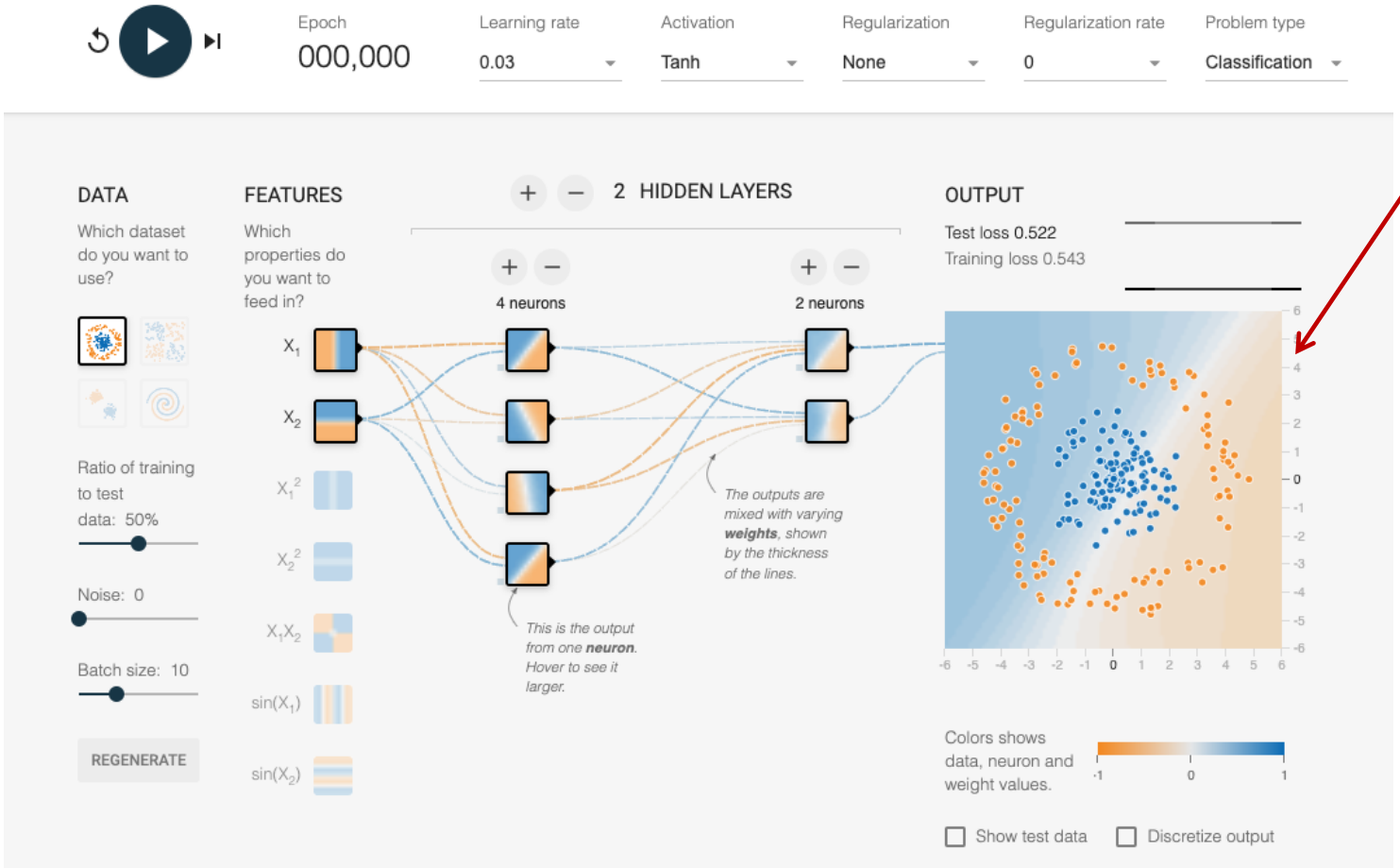




PRACTICE TIME!

Let's put everything we have learned to practice!

Go to: https://playground.tensorflow.org/

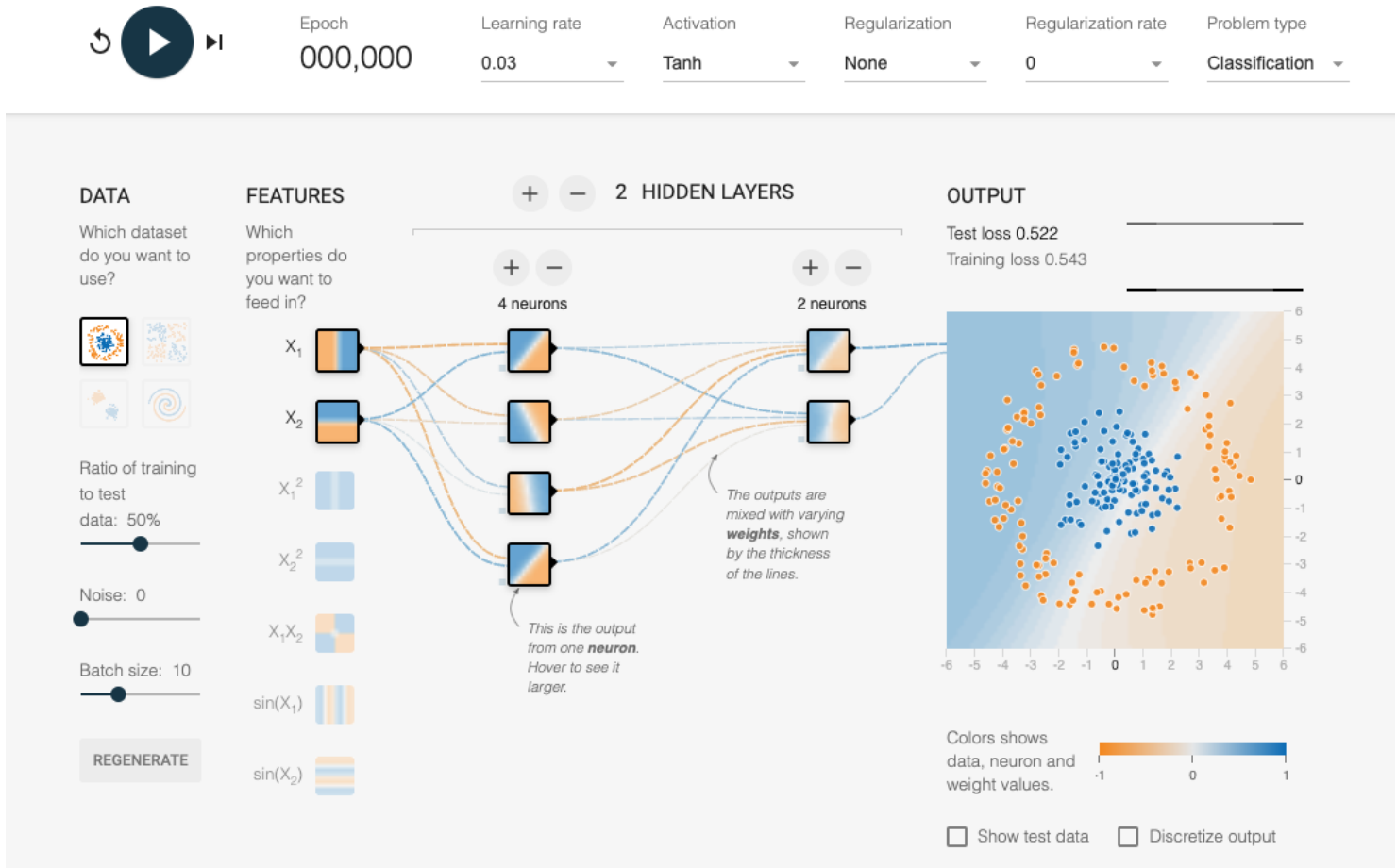


We have a classification task

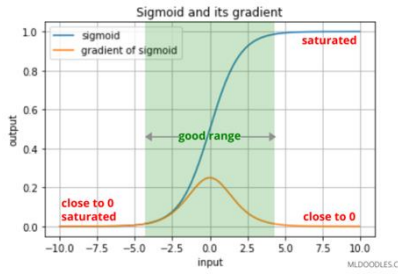
- How many targets/classes are there?
- What is the current input (features)?
- What is the current activation function?
- Is the problem nonlinear? Do we have nonlinearities in our network?
- Will the network correctly separate the classes with the current parameters?
- Run the example as given

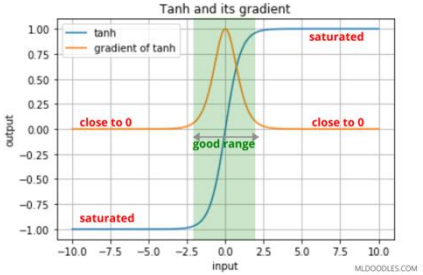
Let’s put everything we have learned to practice!

Go to: https://playground.tensorflow.org/



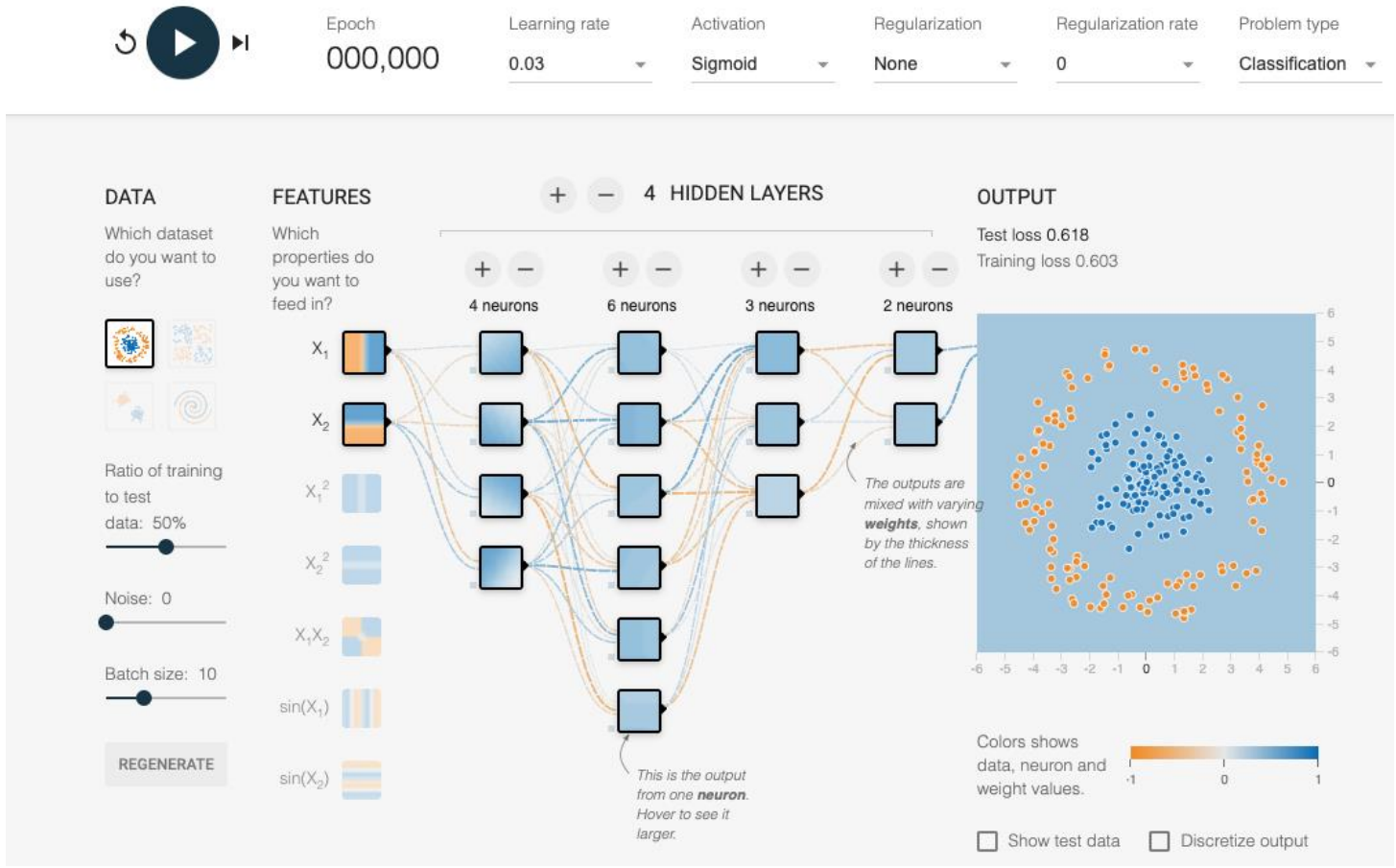
- Change the activation function to linear and run the example. What happens?
- Try adding some nonlinear combination of features. What happens? Which one works?
- Go back to only linear features. Compare the convergence speed of Tanh and sigmoid activation functions. Why is one faster than the other? Try then ReLU.





Let’s put everything we have learned to practice!

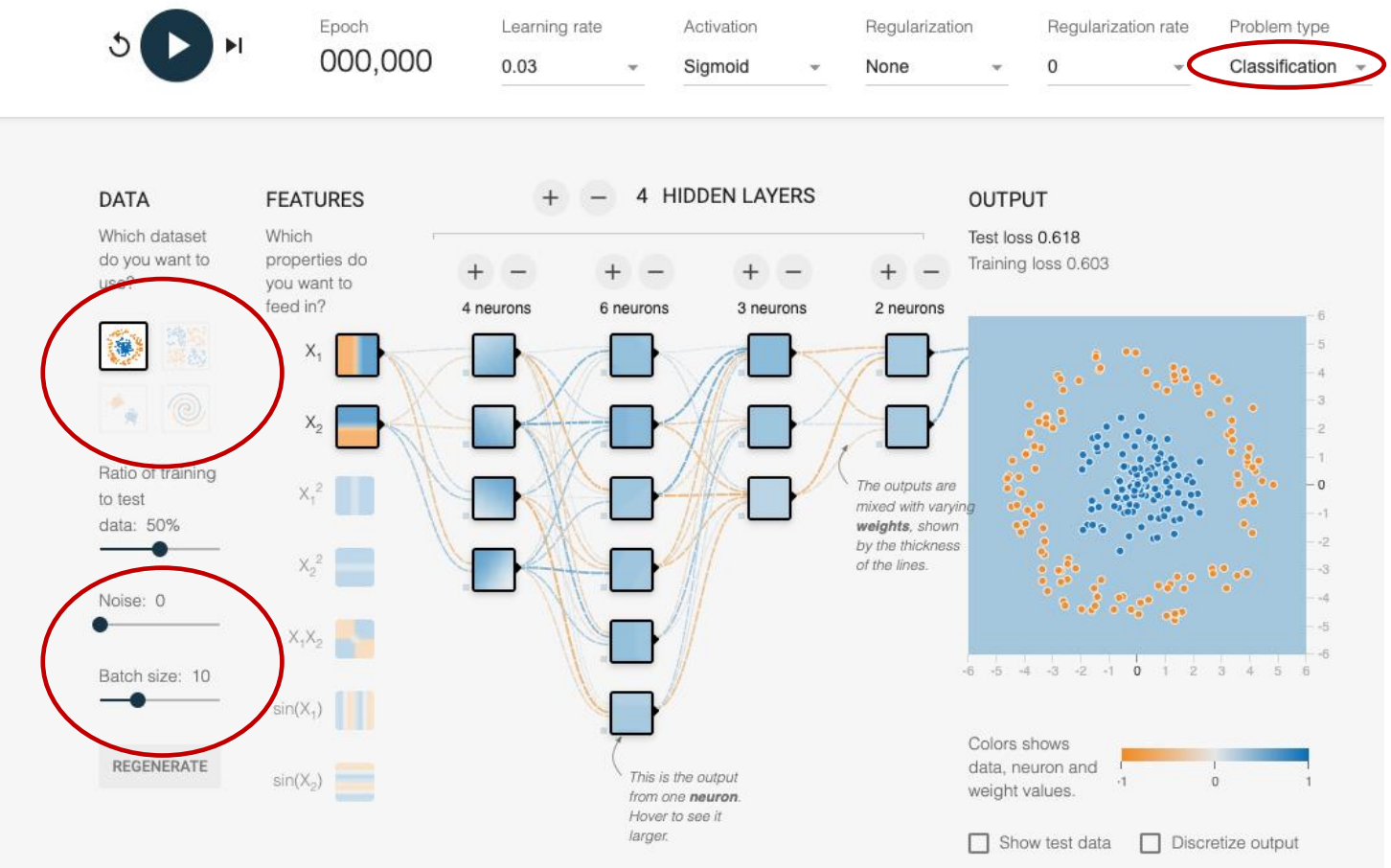
Go to: https://playground.tensorflow.org/



- Increase the size of the network by adding a pair of layers.
- What happens when you try to run it with the sigmoid activation function?
- Increase the number of layers to 5, with 5 or 6 neurons per layer. Run it with the Tanh activation function. What happens to the loss?
- How can we fix it?

Let's put everything we have learned to practice!

Go to: https://playground.tensorflow.org/



Explore by yourself! (~15 min)

- Try the different classification datasets. Which features fit each problem best?
- Play with the batch size and noise.
- Try the second dataset of the regression task.

UNIVERSITY OF LIVERPOOL

THANK YOU FOR YOUR ATTENTION!

What questions do you have for me?

RESOURCES

- https://ml-cheatsheet.readthedocs.io/
- https://notesonai.com/
- https://buildmedia.readthedocs.org/media/pdf/ml-cheatsheet/latest/ml-cheatsheet.pdf
- http://introtodeeplearning.com/
- https://www.offconvex.org/
- https://developers.google.com/machine-learning

Dr. Andrea Santamaria Garcia
Lecturer at University of Liverpool
Cockcroft Institute

ansantam@liverpol.ac.uk
https://www.linkedin.com/in/ansantam/
https://github.com/ansantam
https://instagram.com/ansantam
https://www.liverpool.ac.uk/people/andrea-santamaria-garcia