Introduction to
Artificial Neural Networks
Part II

Dr. Andrea Santamaria Garcia, Chenran Xu

Institute of Beam Physics and Technology (KIT)

Recap from previous lecture

Neural networks:

- are **powerful function approximators** that are **computationally efficient** for big data.
- rely of **large quantities of training data**, and their performance is affected by the **quality** and **variety** of the data.
- are part of **narrow AI**, which means they are specialized to solve particular tasks and do not generalize to different tasks.
- are used in **supervised** and **unsupervised** learning.
- are parametrized by their **weights and biases**, which are iteratively optimized by minimizing the error between the predicted value and target value, called **loss function.**
- are widely used with **first-order gradient methods** to minimize the loss function.
- update their weights through **backpropagation**, which is based on the multivariate chain rule.

Important limitation of gradient descent

- Number of points = n
- Number of parameters = p

How much computational time does it take to calculate the gradients?

$$\sum_{i=1}^{n} \underbrace{(h_{\mathrm{w}}(x_i) - y_i)^2}_{n \text{ terms}} = J(W)$$

$$W \leftarrow W - \underbrace{\alpha\, \nabla J(W)}_{\text{Calculate gradient } p \text{ times}}$$

Let's consider a small example:

- 2000 data points
- 5-64-64-2 network (578 parameters)
  - ➢ ~1.1 million computations for one model update

Evaluates the loss over the entire dataset

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 until approx. minimum

Randomly shuffle samples in the data set

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

$$w \leftarrow w - \alpha \, \nabla J(\mathrm{w}_k)$$

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.

Why is this a good idea, computational efficiency aside?

- In SGD the loss is approximated over a subset of the data, which greatly improves computational efficiency.
- This approximation results in a noisy loss function, different from the "all data" loss function.
- The stochasticity might help in some cases to avoid local minima.

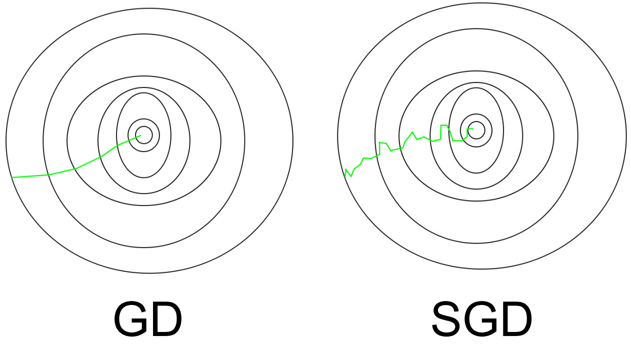
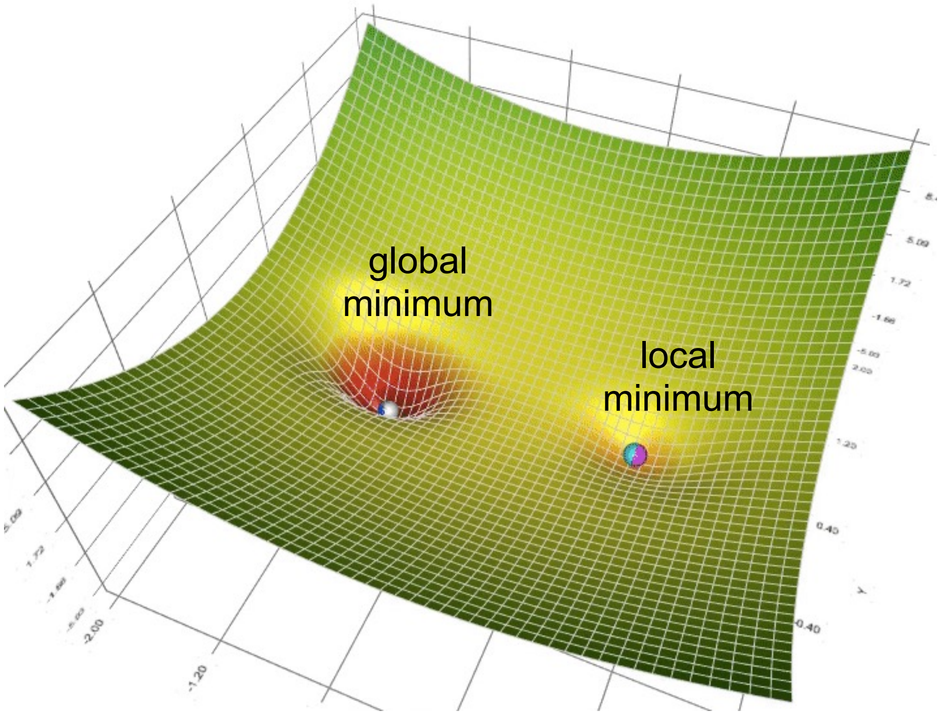


Image from https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

Other optimizers 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 optimization.
- "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:

- All 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 differentiable (first-order derivative can be calculated for a given input value).

Hidden layers

- ReLU (all NNs)
- Tanh (RNNs)
- Sigmoid (RNNs)

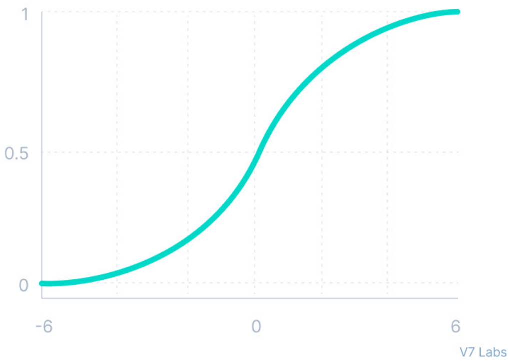
Output layers

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

Activation functions

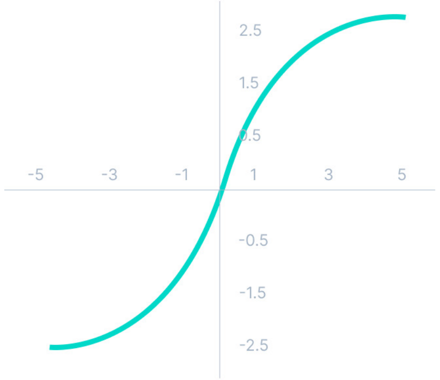
The choice of activation function has a large impact on the capability and performance of the neural network!

Sigmoid / Logistic



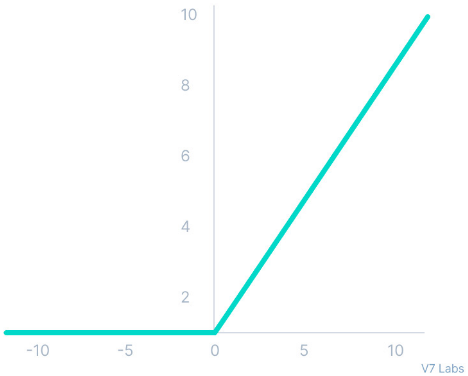
- Maps the input to a [0,1] range, useful to predict probabilities.
- The smaller the input, the closer it will be to 0.
- Saturates at the tail of 0 and 1.
- Output is not zero centered and will always be the same sign and in a small range, which makes training more difficult and unstable.

Tanh



- Maps the input to a [-1,1].
- The smaller the input, the closer it will be to -1.
- The output is zero centered, so output values can be easily mapped to strongly positive, negative or neutral.
- Neurons saturate for large negative and positive values.

ReLU

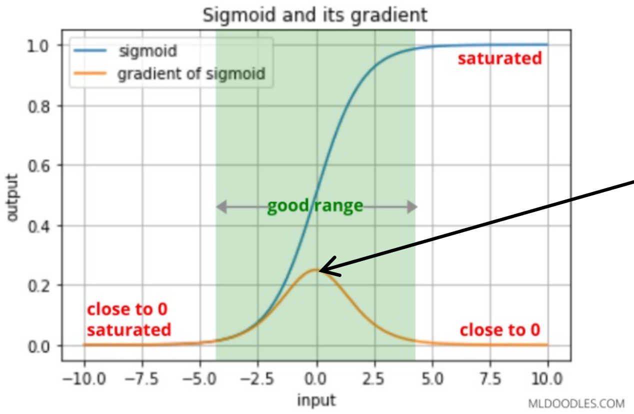


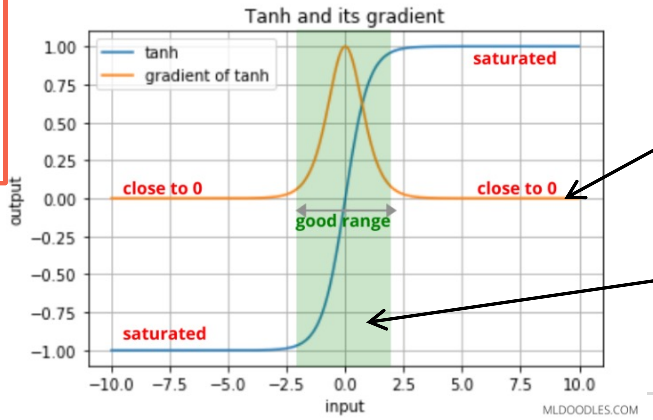
- Only positive values pass through (neuron deactivated if output is negative).
- Very computationally efficient.
- Non saturating property accelerates GD convergence.
- The gradient is also zero for negative values, which can create dead neurons that never get activated (leaky ReLU).

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

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

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

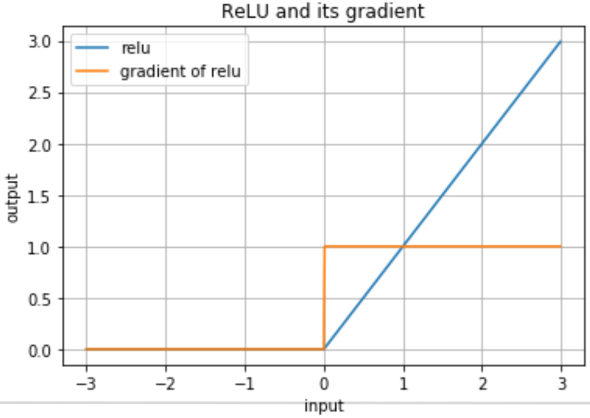
No response to changes in input

Very narrow range, small values

When the partial derivative vanishes the weights are not updated anymore

$$w \leftarrow w - \alpha \nabla J(\mathrm{w}_k)$$

good for hidden layers



Images from mldoodles

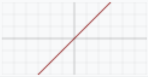
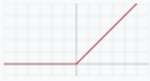
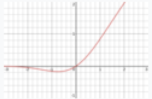
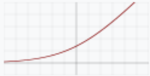
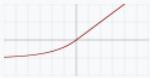
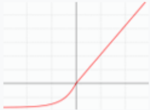
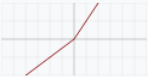
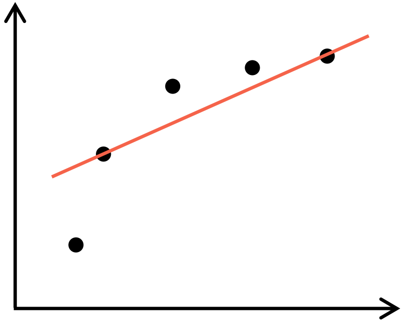
Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$
Identity		x	1
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$
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]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$	$\Phi(x) + x\phi(x)$
Softplus[9]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ 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

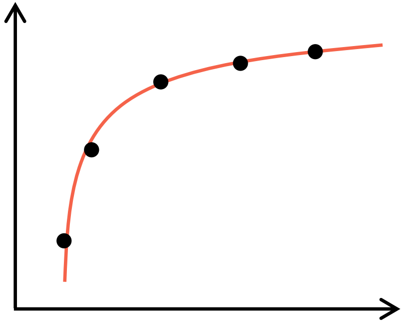
- Overfitting happens when the NN learns too many details about the training data and will fail to generalize to unseen data.
- This happens due to:
  - Overly complex models with too many parameters (e.g. deep neural networks).
  - Training for too long, capturing noise instead of genuine patterns.

Underfit



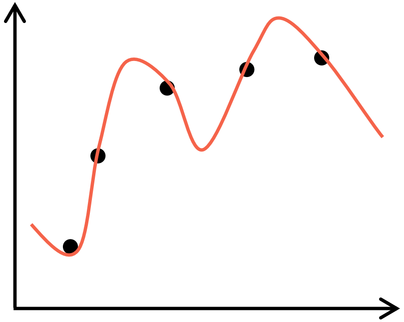
$\hat{y} = w_0 + w_1 x$

Good fit



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

Overfit



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

Regularization

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

Early stopping:

- Stop training before overfitting.
- Criteria: when the loss does not improve beyond a certain threshold.
- The "patience" parameter allows you to specify how much to wait after the threshold before stopping training.

Dropout:

- Randomly deactivate neurons from the during training in each iteration (equivalent to training different NNs).
- Reduces number of parameters.
- Avoids relying on certain nodes that only learn certain patterns.

Weight regularization

- Adds a weight penalty term to the loss function, penalizing large weights (too sensitive, large changes in the output).
- The term increases the error, forcing the network to minimize the weights contributing more to the loss.

Batch normalization:

- Normalizes the inputs to layers across each mini-batch to reduce internal covariate shift (the distribution of each layer's inputs changes during training, as the parameters of the previous layers change, slowing down training).

Stages in supervised learning

Training Phase:

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

- The algorithm is trained using a labeled dataset consisting of training examples (x, y).
- The model makes predictions on the training data based on the current state of its parameters.
- A **loss function** is used to measure the difference between the model's predictions and the actual target values for the training data.
- The objective of the training is to minimize this loss function. This is done using **optimization algorithms** like **gradient descent**.

50% of data
Curated
Representative
“Gold standard”

Stages in supervised learning

Validation Phase:

Select best performing model or approach

- The model's performance is evaluated on a separate dataset not seen by the model during training (validation dataset).
- This phase helps in tuning the model's hyperparameters and provides an estimate of how well the model has generalized to unseen data.

25% of data
Real-world data
Data of your study

Testing Phase:

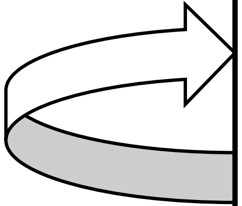
Evaluation of final model performance

- Once the model is trained and validated, its performance is tested on another set of unseen data (test dataset).
- This phase provides an unbiased evaluation of the final model fit on the training dataset.

25% of data
Real-world data
Data of your study

Not used in training for unbiased performance estimation

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 & convergence criteria
5. Choose an optimizer & set its hyperparameters
6. Choose number of epochs to train
7. Decide on regularization techniques
8. Perform forward propagation
9. Compute gradients with backwards propagation
10. Update weights & keep track of loss
11. Evaluate the model's efficiency

Supervised learning training loop

Pseudo code for training a NN with SGD with mini-batches

```
randomly initialize the NN weights w
for i = 1, ..., n_epoch do:
        randomly shuffle samples in the data set
        for batch in mini-batches do:  # loop over the data set in batches
                perform forward pass ŷ = f(x)
                calculate loss  J (ŷ, y)
                update weights w ← w − α ∇_w J  # one step SGD update
```

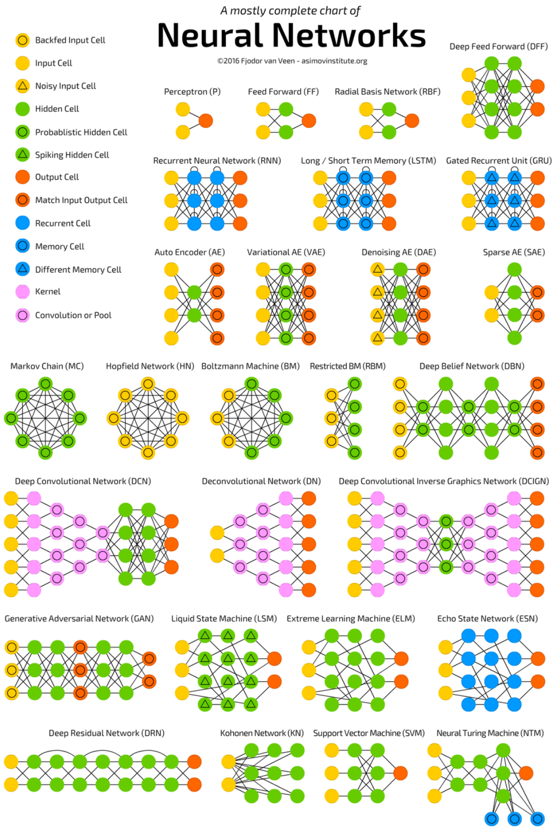
- n_{epoch} is the number of times the NN goes over the entire data set
- The data set is randomly shuffled and split into small chunks (mini-batches) each with $\mathrm{batch\_size}$ samples
- One batch is fed through the the NN (in parallel) and weights are updated once.

Further considerations

- **Error analysis**: under which conditions does the model perform poorly?
- **Statistical testing**: are the performance metrics between models statistically significant?
- **Robustness and generalization**: how well does the model perform on variations of the test data it was not explicitly trained on?
- **Real-world performance**: evaluate the model in a real-world setting or a simulation that closely mimics the production environment.
- **Resource utilization**: 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

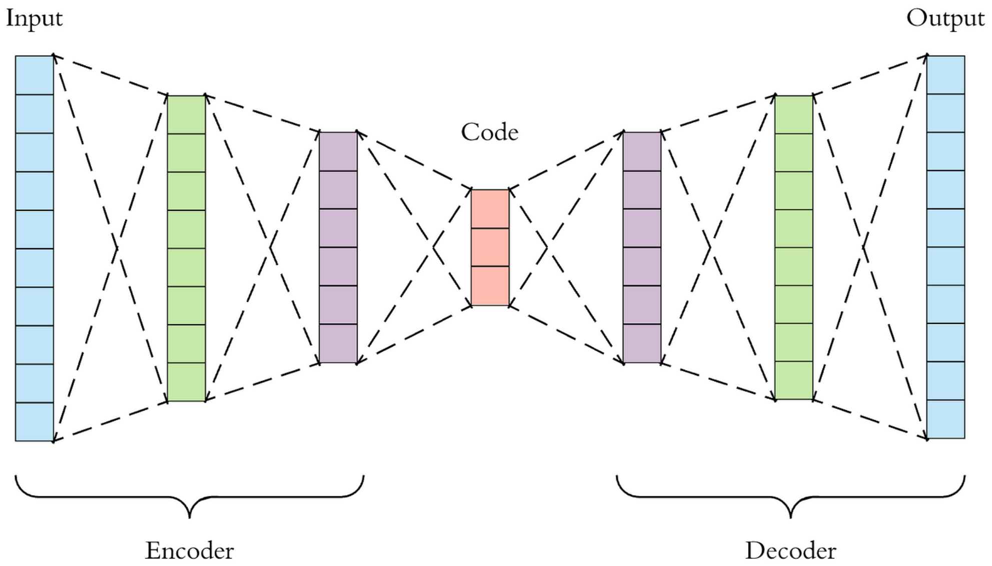
Autoencoder (AE)

- 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

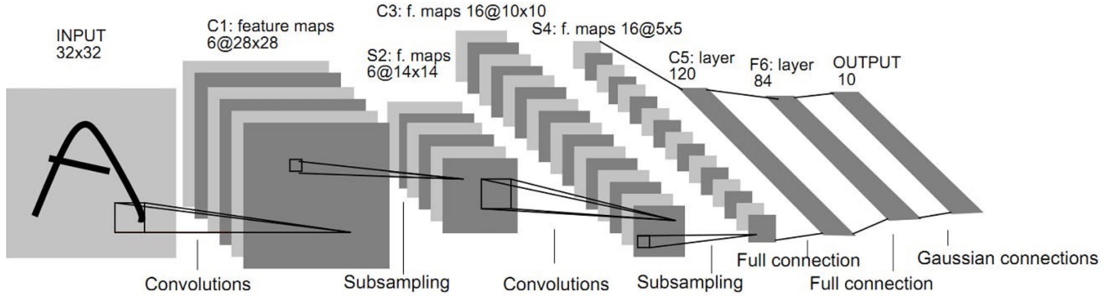
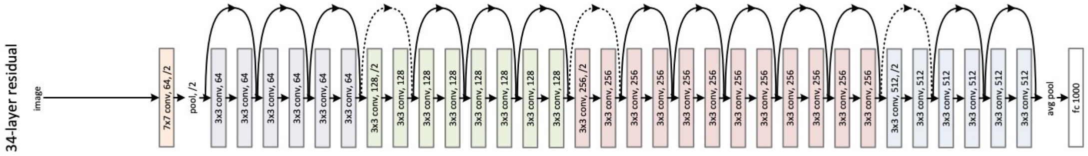


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

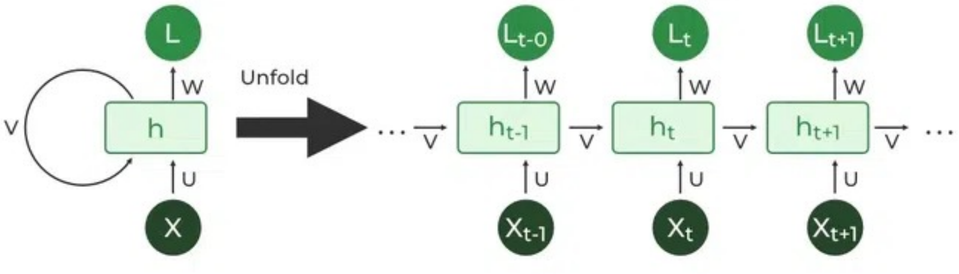


Recurrent Neural Network (RNN)

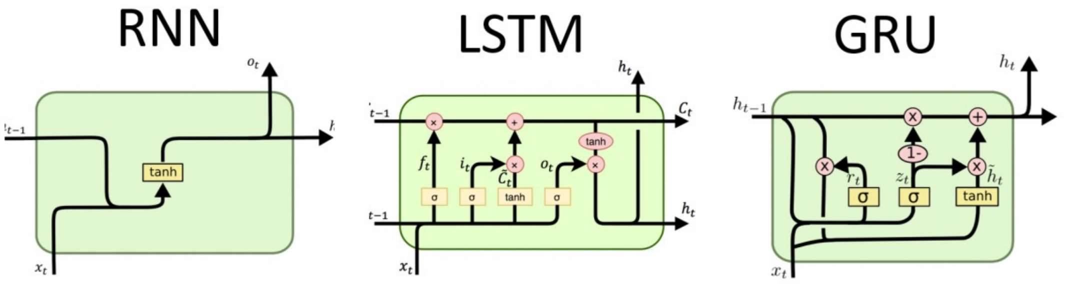
- Used in **time-series** data.
  e.g. Natural language processing (NLP), forecasting, ...

- Contains hidden state variables (memory, context).

- Allows variational lengths of inputs and outputs.



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



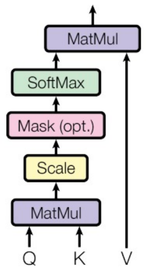
http://dprogrammer.org/rnn-lstm-gru

Transformer

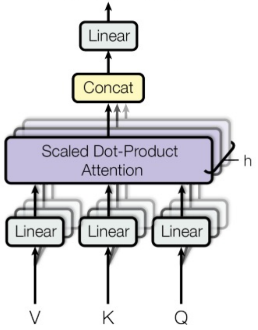
[Attention is all you need, 2017](#)

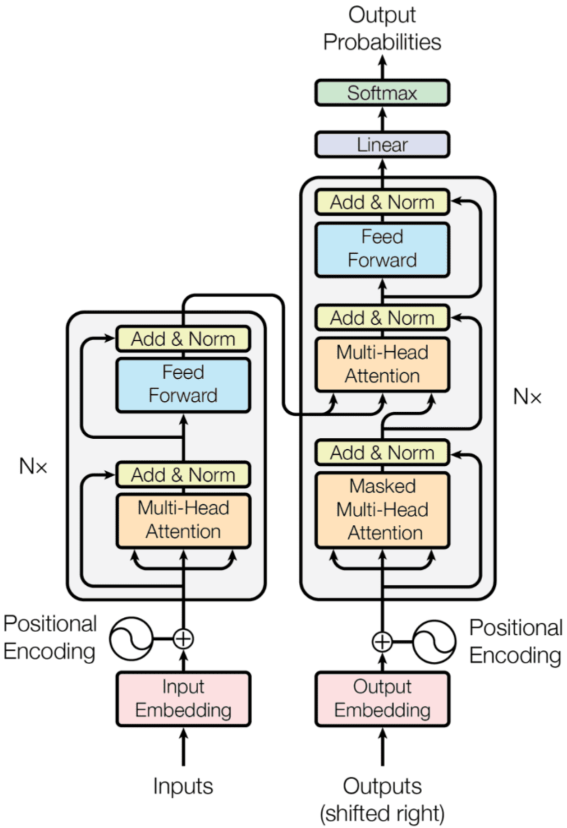
- Probably the most successful NN structure nowadays
- **Architecture behind the LLMs (ChatGPT,...)**
- Use only the *attention* mechanism without the recurrent structure
- Parallelizable -> faster training

Scaled Dot-Product Attention



Multi-Head Attention





There is not one library to rule them all

python

Neural networks/ Deep learning

JAX

Google DeepMind

TensorFlow

Google

PyTorch

facebook

(Meta-AI)

Keras

Tensorflow backend

mxnet

Caffe2

Microsoft Cognitive Toolkit

ML algorithms / optimization

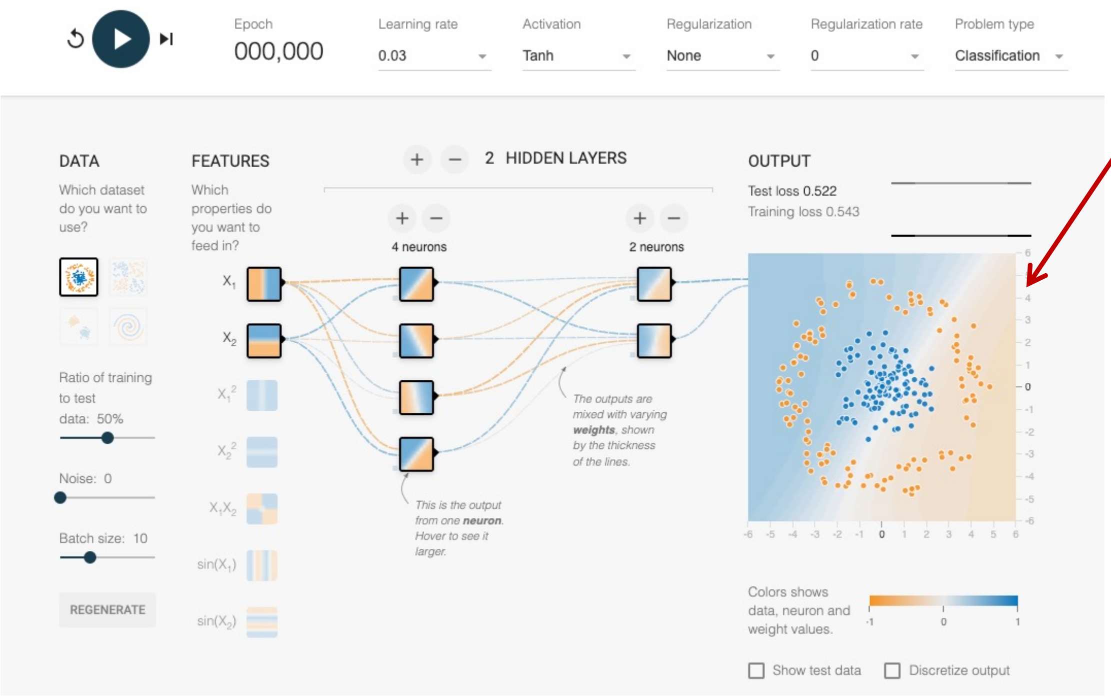
scikit learn + SciPy NumPy

Apache Spark

theano

Let’s put everything we have learned to practice!

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

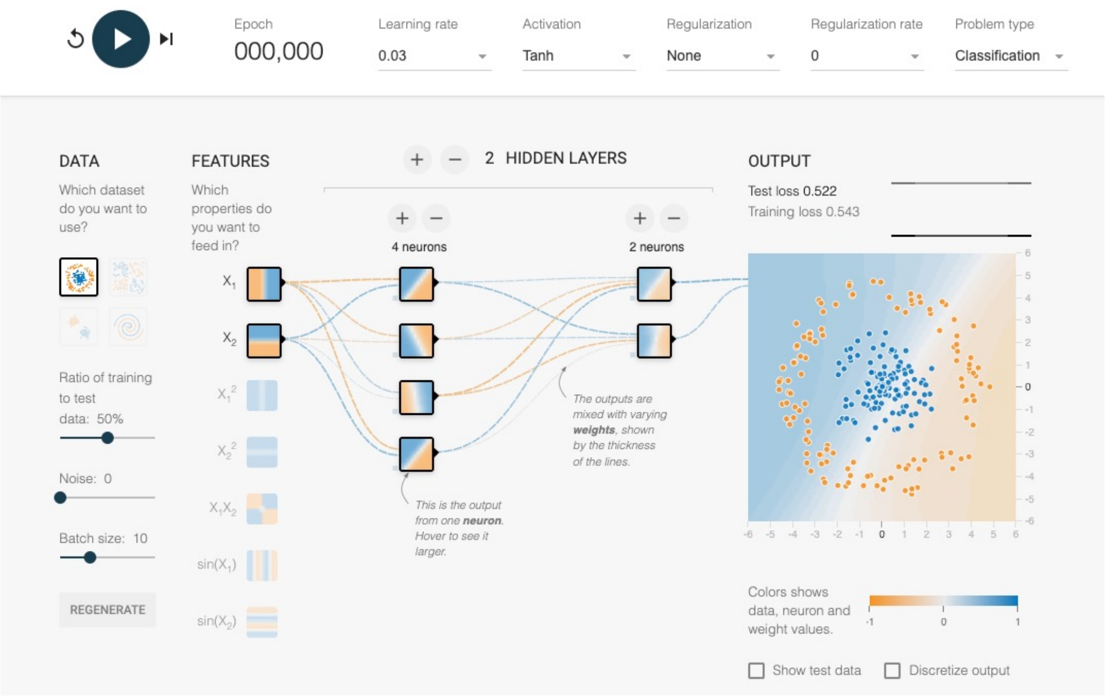


We have a classification task

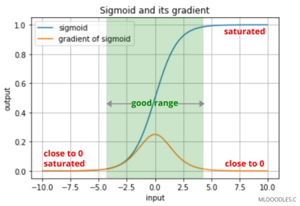
- How many targets/classes are there?
- What is the current input (features)?
- What is the current activation function?
- Is the problem nonlinear? Do we have nonlinearities in our network?
- Will the network correctly separate the classes with the current parameters?
- Run the example as given

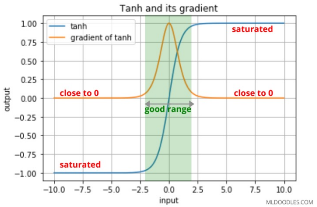
Let’s put everything we have learned to practice!

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



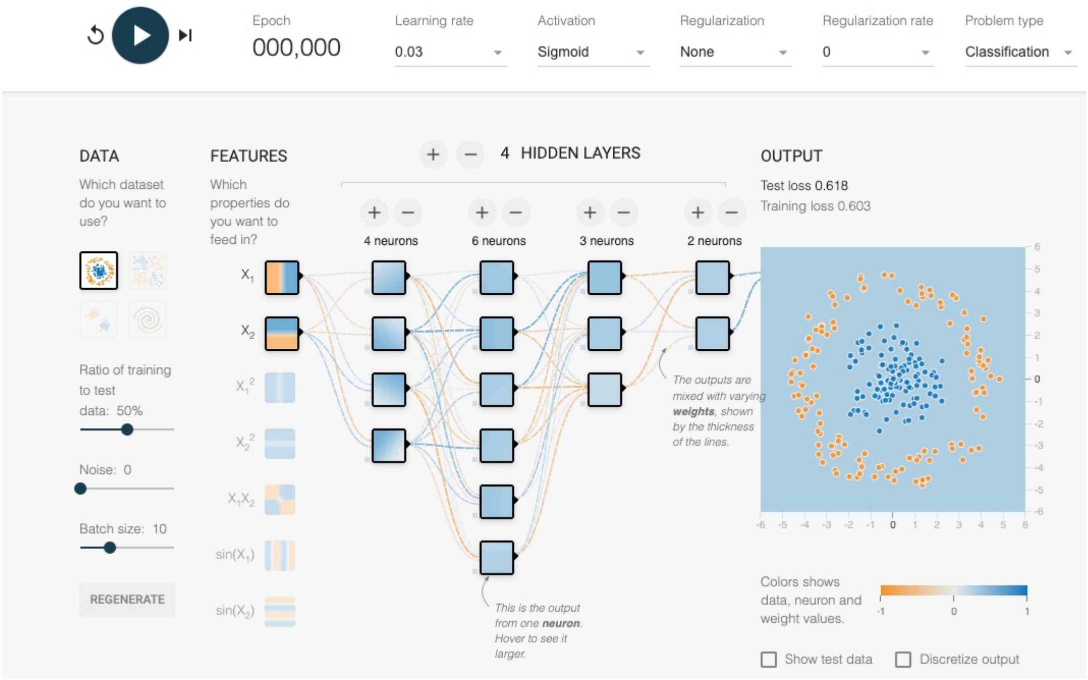
- Change the activation function to linear and run the example. What happens?
- Try adding some nonlinear combination of features. What happens? Which one works?
- Go back to only linear features. Compare the convergence speed of Tanh and sigmoid activation functions. Why is one faster than the other?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





Let’s put everything we have learned to practice!

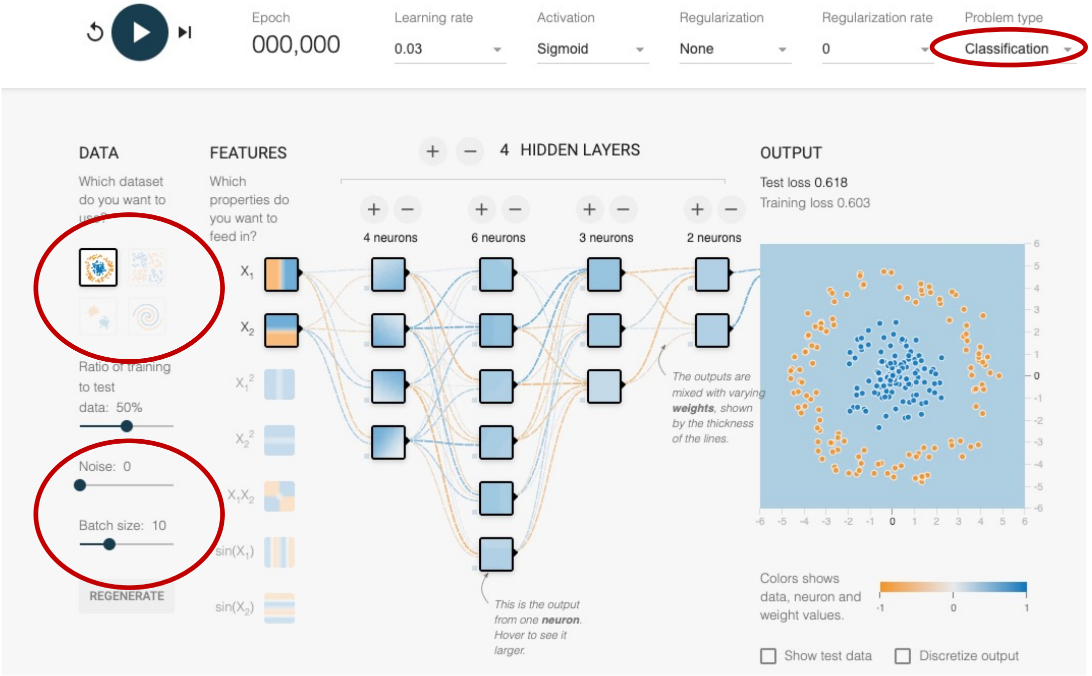
Go to: https://playground.tensorflow.org/



- 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.

Tutorials

https://github.com/machine-learning-tutorial/neural-networks

Thank you for your attention!

What questions do you have?

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/

All icons from this talk from TheNounProject