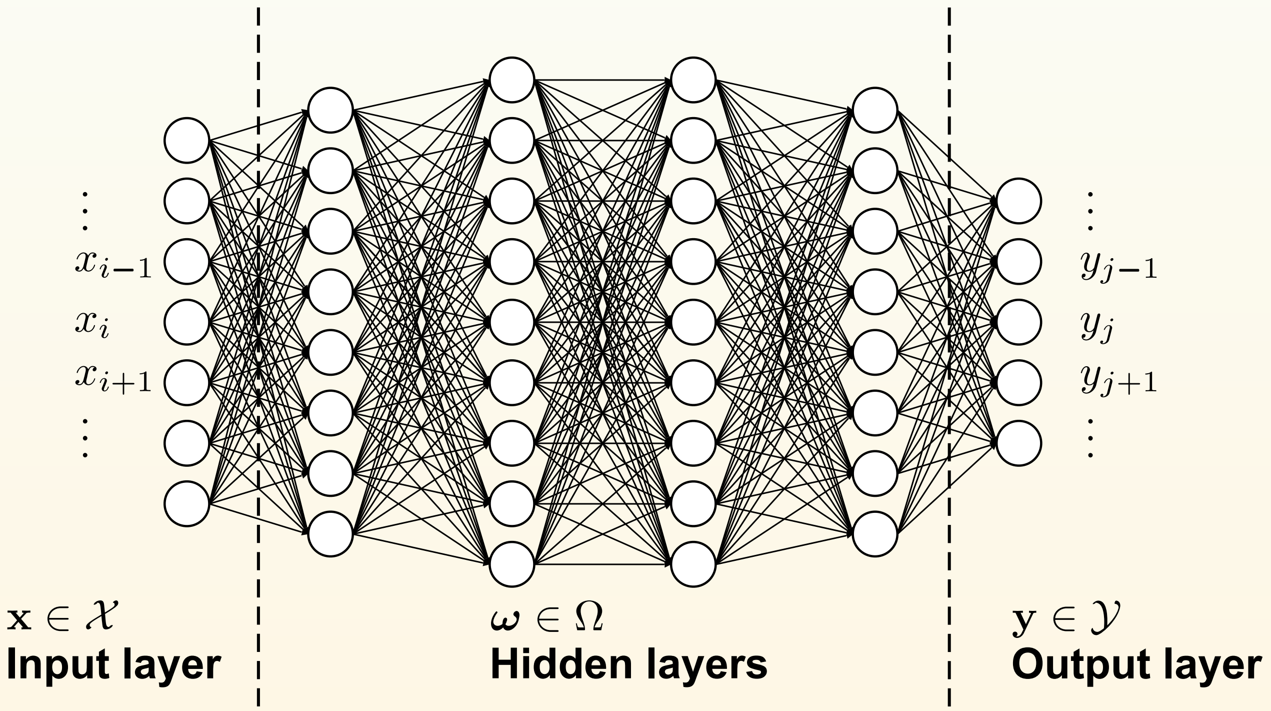
Calorimeters meet ***<u>MLPs</u>***

Jan Kieseler

Recap: DNNs and their parameters

- All nodes of consecutive layers are connected with each other
- Typically an ANN is called “deep” if it has >4 hidden layers
- Referred to as Multi-Layer Perceptron, Feed-Forward NN

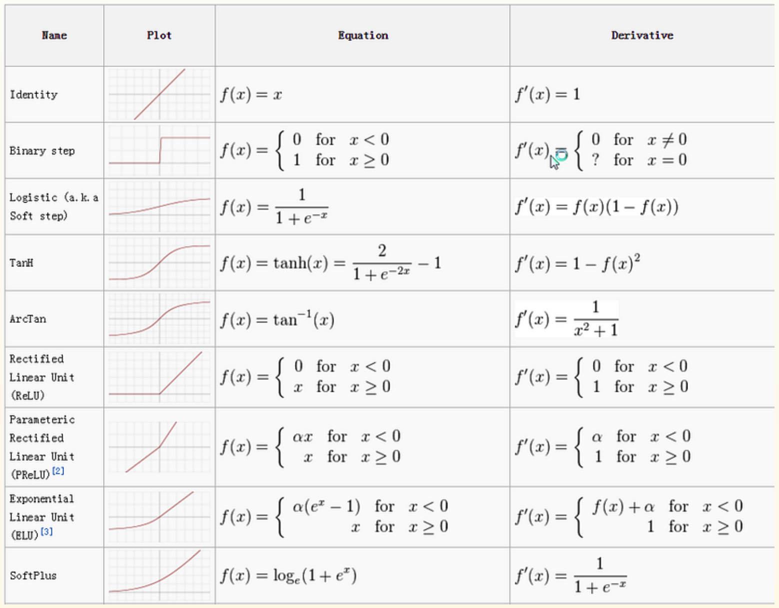


Activation functions: adding non-linearities

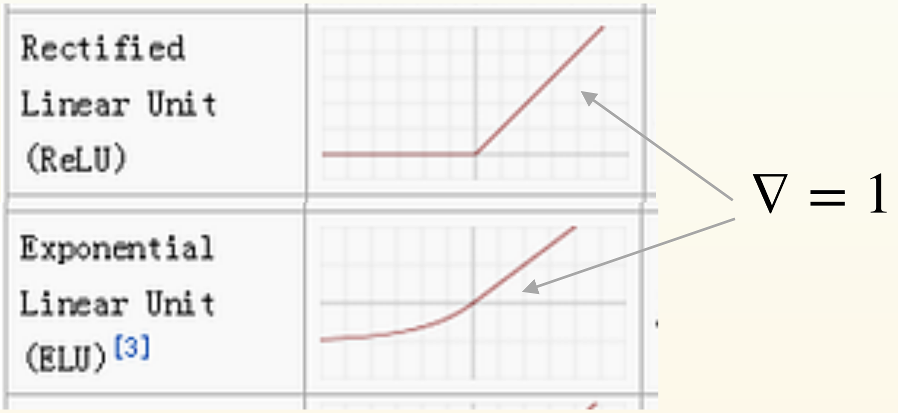
- One layer: $h^{(k+1)}(h^{(k)}) = \theta(\omega_k h^{(k)} + b_k)$

- Without non-linear activation:
  $y(x) = h^{(4)}(h^{(3)}(h^{(2)}(h^{(1)}(x)))) = \tilde{\omega}x + \tilde{b}$

Back-of-the envelope exercise



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

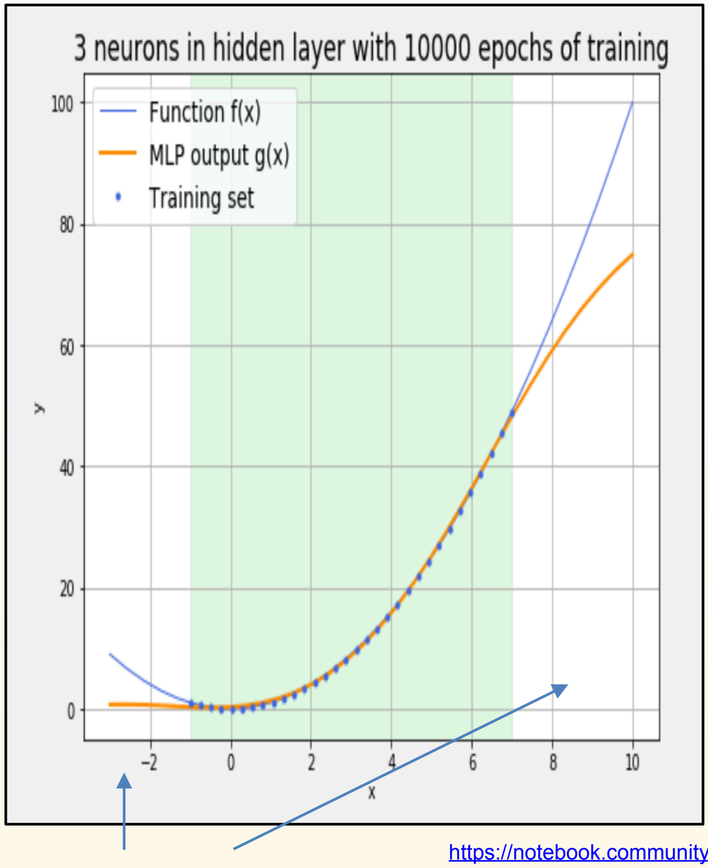


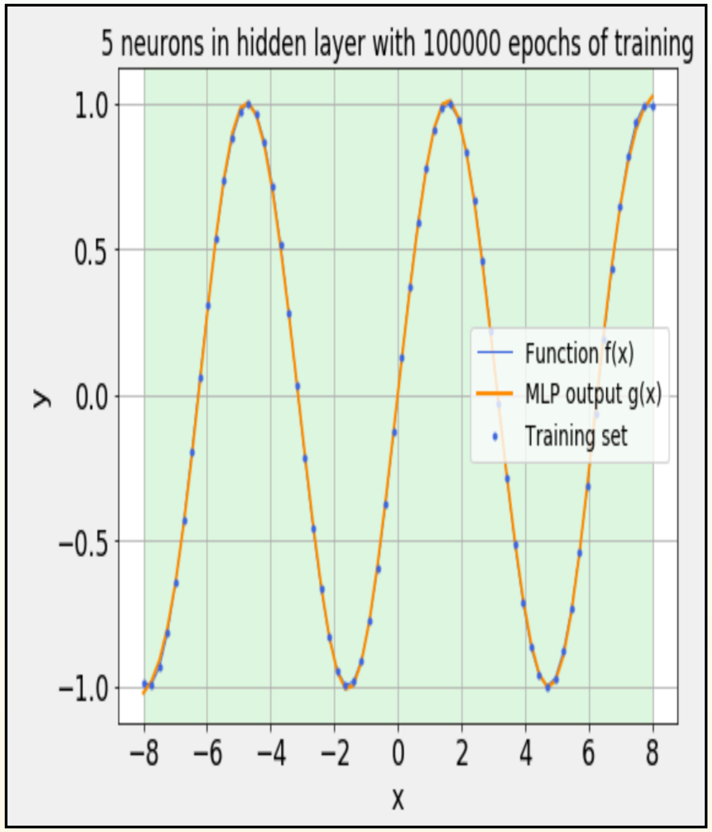
- There is a whole zoo: theoretically, the choice does not matter for hidden layers
  - For the output it **does** matter as it restricts / shapes the output distribution
- In practice: not vanishing/exploding output (and gradients)
  - Suggestion: (s/r)elu

https://machinelearninggeek.com/activation-functions/

DNNs as universal function approximators

- Very simple NN: one hidden layer, one input, one output, tanh activation





https://notebook.community/kit-cel/lecture-examples/mloc/ch3_Deep_Learning/pytorch/function_approximation_with_MLP

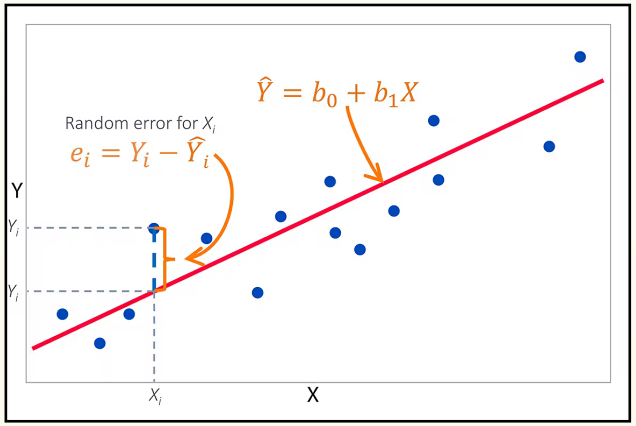
"Out-of-distribution"

- DNNs are universal function approximators, already with very few parameters
    - But beware of extrapolation / out-of-distribution effects

Loss (cost) function

- The loss function quantifies how well a model performs
- Here we compare sample-by-sample the NN output with a truth label

- E.g. text book linear regression: we know the 'truth'
    - Model: $\Phi(\omega, x) = \omega_a x + \omega_b$
    - Least-square method:



$$\min 1/N \sum_i^N \left((\Phi(\omega, x_i) - y_i)^2 \right) = \min \text{MSE}(\Phi(\omega, x), y)$$

Mean squared error loss

- The mean squared error loss is a standard loss for regression tasks

- It assumes a Gaussian distribution of the NN estimates (log(L))
- We want to map to the whole output range: linear output activation

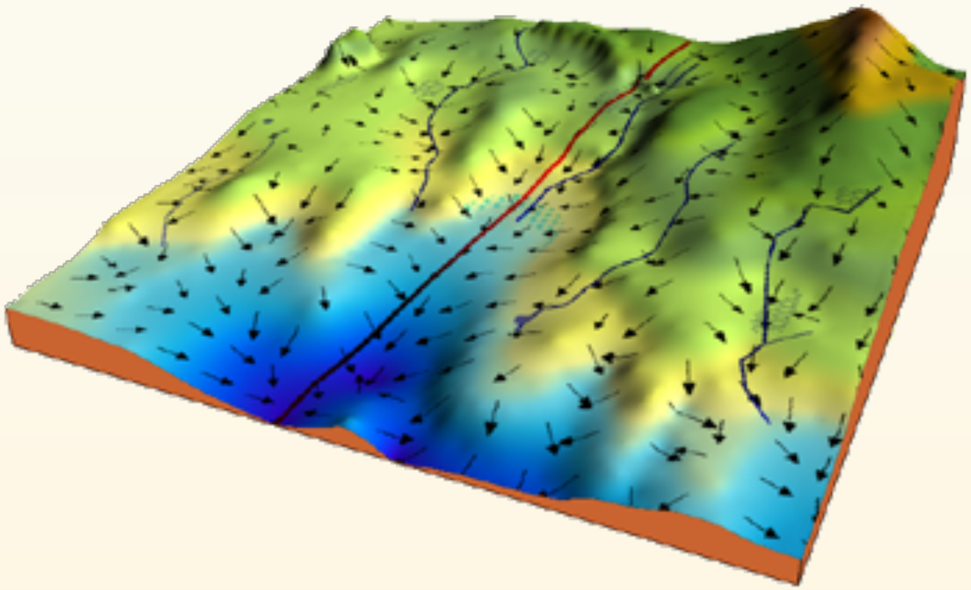
What does that mean for calorimeters??

How do we train: gradient descent

- Well established, robust numerical minimisation procedure:

$$\omega^{(k+1)} = \omega^{(k)} - \underbrace{\eta}_{\text{Learning rate}} \nabla_{\omega^{(k)}} L\left(\Phi(\omega, x), y\right)$$

- Update ω until $L\left(\Phi(\omega^{(k)}, x), y\right) - L\left(\Phi(\omega^{(k+1)}, x), y\right) < \epsilon$



https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

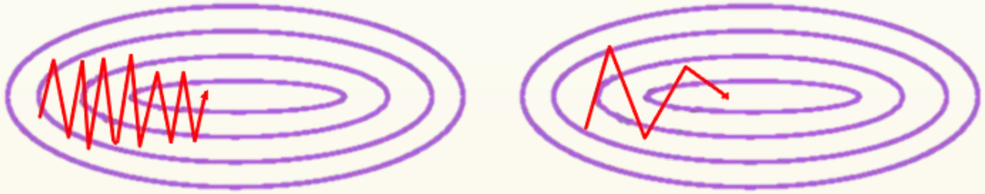
Stochastic gradient descent and momentum

- Stochastic gradient descent is gradient descent on (mini) batches instead of the full data set

$$\omega^{(k+1)} = \omega^{(k)} - \eta \nabla_{\omega^{(k)}} L\left(\Phi(\omega, x), y\right) \rightarrow \omega^{(k+1)} = \omega^{(k)} - \eta \nabla_{\omega^{(k)}} L\left(\Phi(\omega, \{x\}_k), \{y\}_k\right)$$

GD SGD

- Reduces computational burden: makes training feasible
- Introduces extra noise that can actually **help**



Goodfellow et al. (2016)

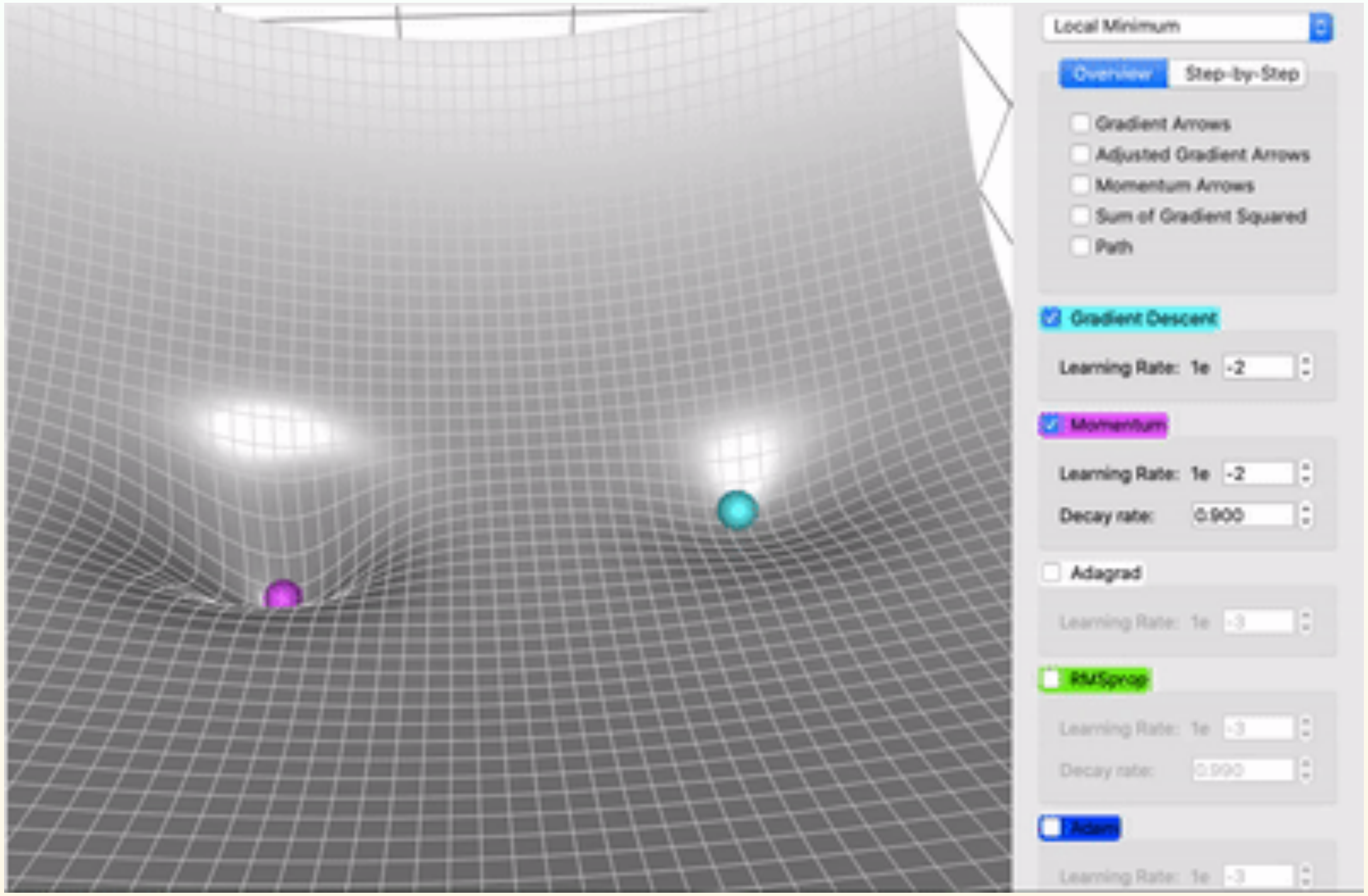
- Add a momentum/velocity that averages the general directions in parameter space

$$v^{(k)} = \alpha v^{(k-1)} - \eta \nabla_{\omega^{(k)}} L$$
$$\omega^{(k+1)} = \omega^{(k)} + v^{(k)}$$

For our exercises, use "Adam"

➡The basis for most common optimisers that are in use

Momentum in action


Local Minimum

Overview | Step-by-Step

- ☐ Gradient Arrows
- ☐ Adjusted Gradient Arrows
- ☐ Momentum Arrows
- ☐ Sum of Gradient Squared
- ☐ Path

☑ Gradient Descent
Learning Rate: 1e -2

☑ Momentum
Learning Rate: 1e -2
Decay rate: 0.900

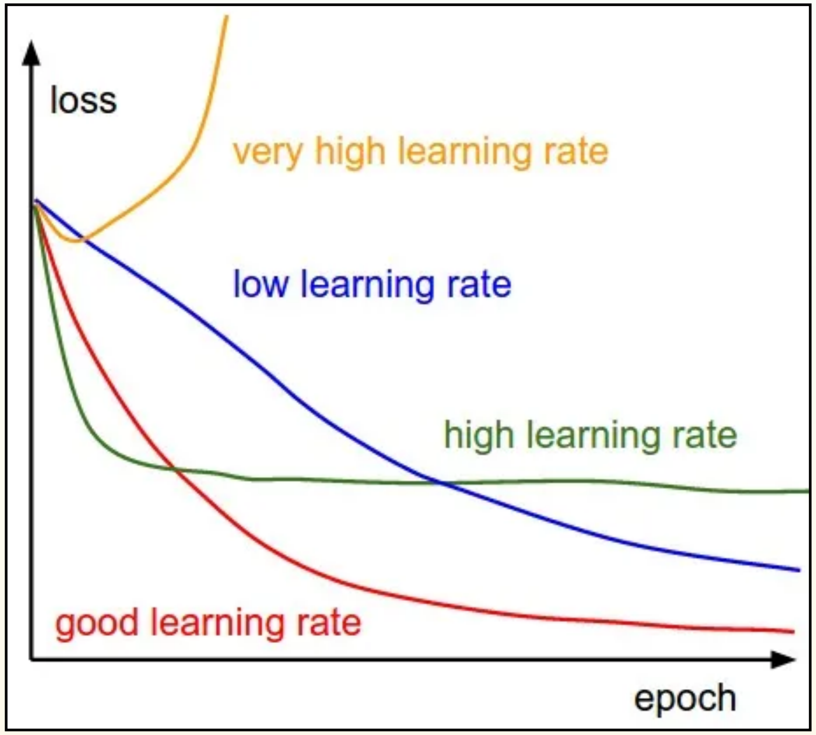
☐ Adagrad
Learning Rate: 1e -3

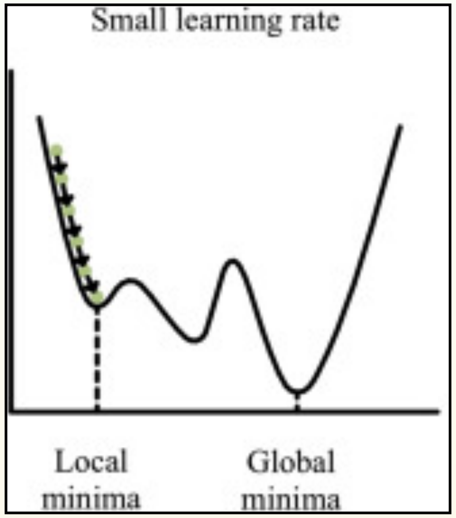
☐ RMSprop
Learning Rate: 1e -3
Decay rate: 0.990

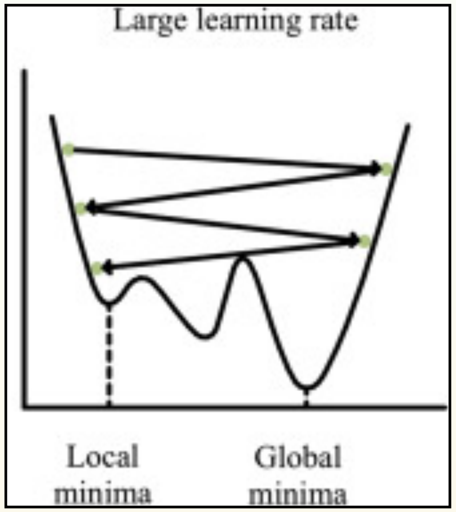
☐ Adam
Learning Rate: 1e -3

The above and many more details (great page)
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Learning rates

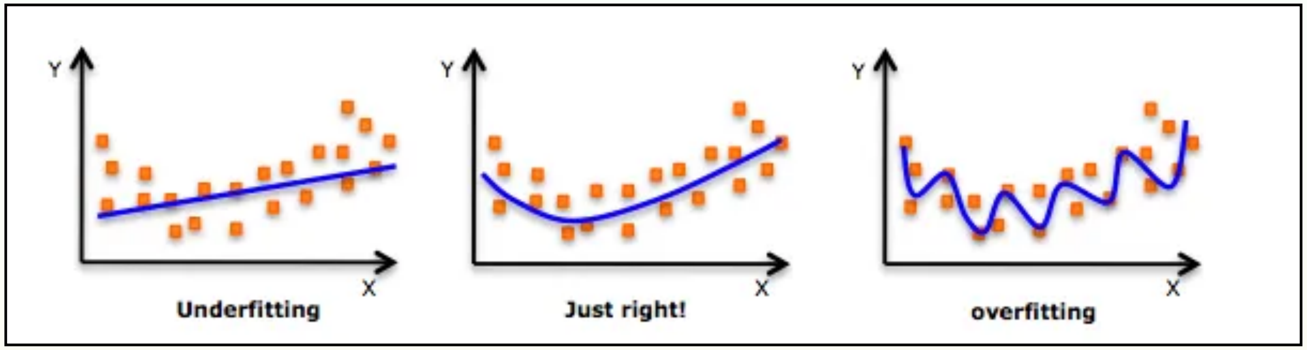


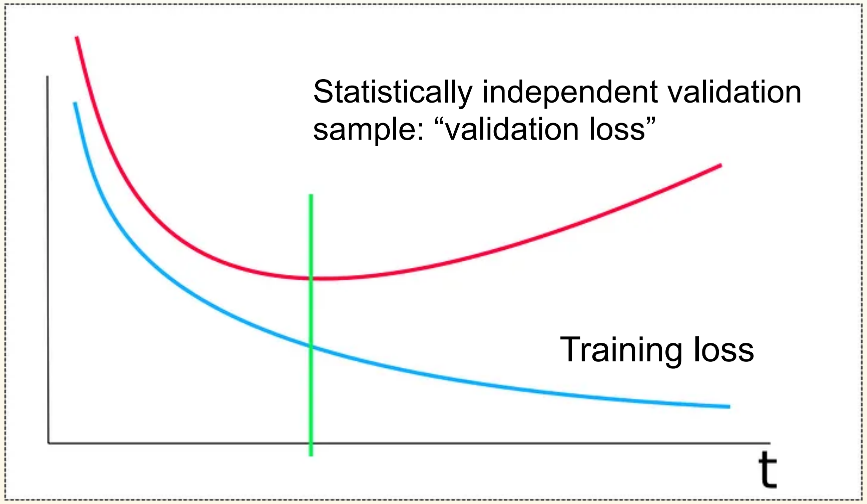




- There is no universally best learning rate - always needs to be adjusted
- Rule of thumb:
  - More parameters ↔ lower learning rate
  - Smaller batches ↔ lower learning rate

Overfitting / overtraining





- More data **per weight**:
  - Simpler network
  - More data
- Lower learning rate
- Regularisation (weight regularisation, Dropout) *

https://medium.com/analytics-vidhya/the-perfect-fit-for-a-dnn-596954c9ea39

Summary

- Interactive
- MLPs and activations: non-linearities
- Loss functions: how to chose them?
- Gradient descent and momentum
- How to chose learning rates