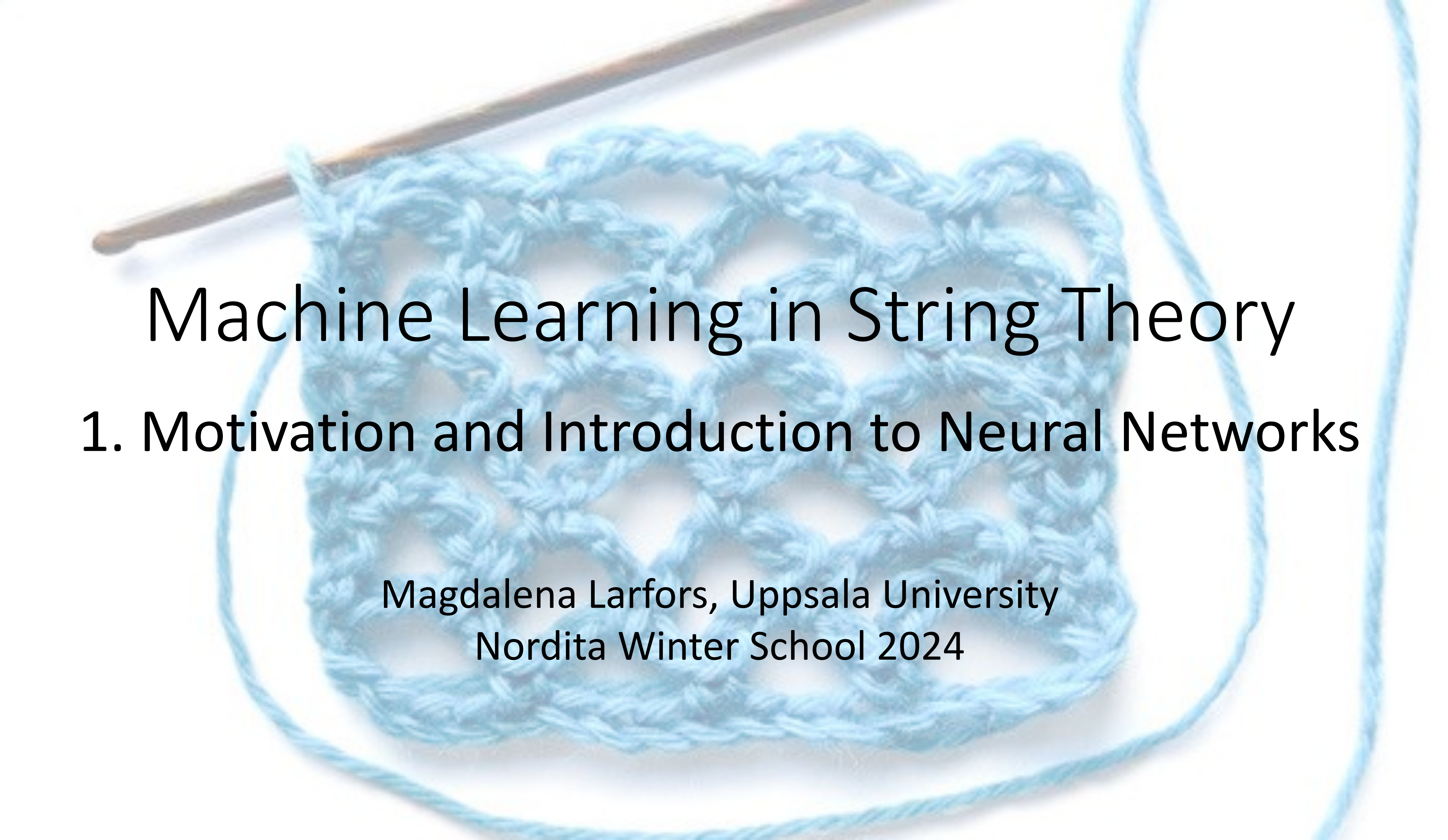


Machine Learning in String Theory

1. Motivation and Introduction to Neural Networks

Magdalena Larfors, Uppsala University

Nordita Winter School 2024

Outline: Machine Learning in String Theory

- Why machine learning?
- Motivational problems: how can ML help string theorists?
- Getting started with ML
- ML problems and ML techniques
- Neural Networks
- Training NNs with backpropagation
- A first peek at ML libraries

Why machine learning?

It works!

- Automating tasks (label images, speech recognition, ...)
- Solve hard problems (play go, synthesize information (chat-GPT etc) …)

Recent successes driven by

- better network architectures and optimization methods
- better computational hardware (GPUs)
- more data (… and more money/energy for training)
- user-friendly libraries

Why ML in maths, physics, strings?

Physics:

- very large data volumes from experiments and observations
- image recognition and classification sometimes spot-on for analysis, e.g. jet tagging in collisions, finding exoplanets

  HEPML-LivingReview

  nasa/deep-learning-adds-301-planets-to-keplers-total-count
- Theoretical physics/math:
  study *examples* to find *patterns*, gain *intuition* and make *conjectures*
  ML can effectively explore such “pure” data sets

Some motivational problems

- Find {SM, MSSM, inflation, dS, scale separation, ..} in the string theory landscape
  - Often requires hard computations! May even be practically unfeasible.
  - Compute topology and geometry of extra dimensions
- Build/analyse effective field theories; what theories are allowed?
- Bootstrap for CFT
- Learn mathematical structures (perhaps of relevance for physics)
- Use physics-inspired models to explain how Machine Learning works

… *literature exists on all of these topics*

Getting started

- Having identified an interesting problem in string theory, how to get started with the ML implementation?

- Is data given? If not, can you create it (at least in simpler settings)?
- For your input data, do you know the answer (label)?
- Classification or regression problem?
- Are there additional constraints on the answer?
- Do you need to search through complex landscapes?

ML techniques

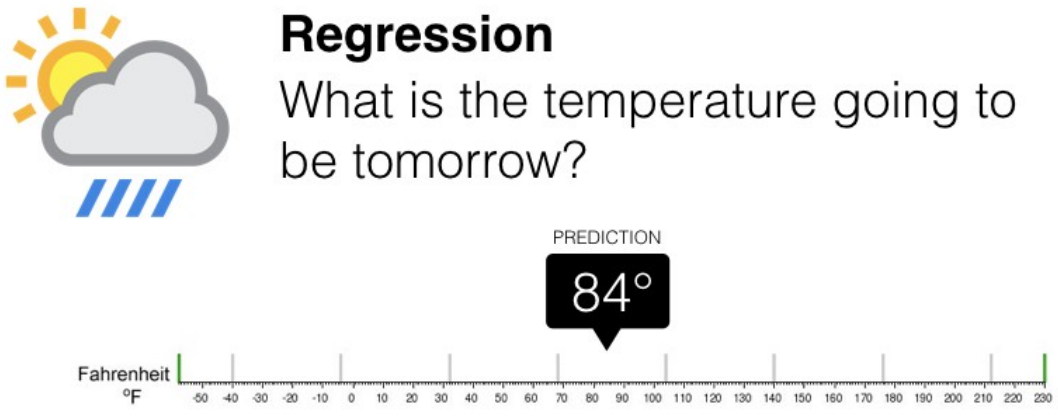
- Have labelled data (x, y), i.e. know true answer - today
  Use supervised learning techniques

- Have unlabeled data (x) → Unsupervised learning - Thursday
  Unlabeled data with constraints → Semi-supervised learning, PINNs

- Reinforcement learning (also genetic algorithms): - Friday
  “solve” complex environment with known rules and goal (no data)

ML problems: regression or classification

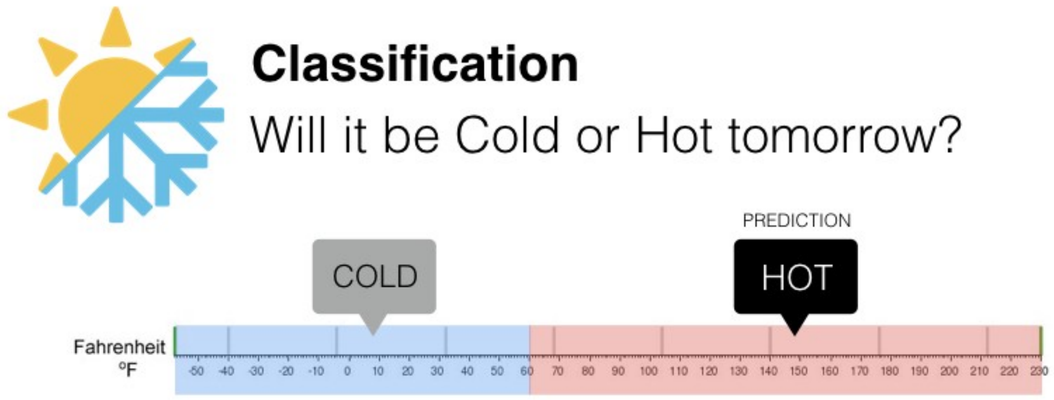
- Regression:
  predict output as function of data



Regression

What is the temperature going to be tomorrow?

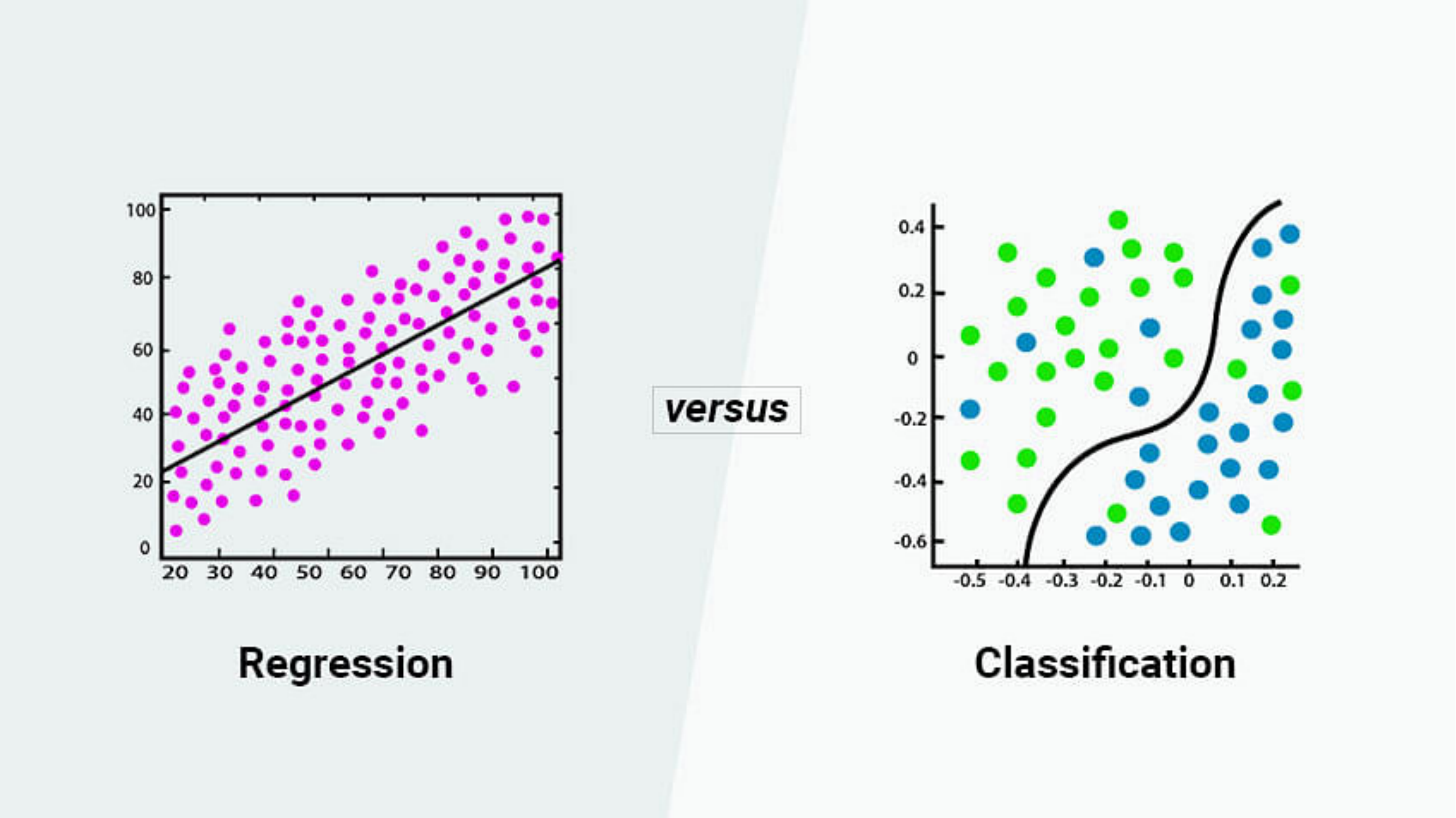
- Classification:
  is data point of type A, B, C, ...?
  (really what is the probability ?)



Classification

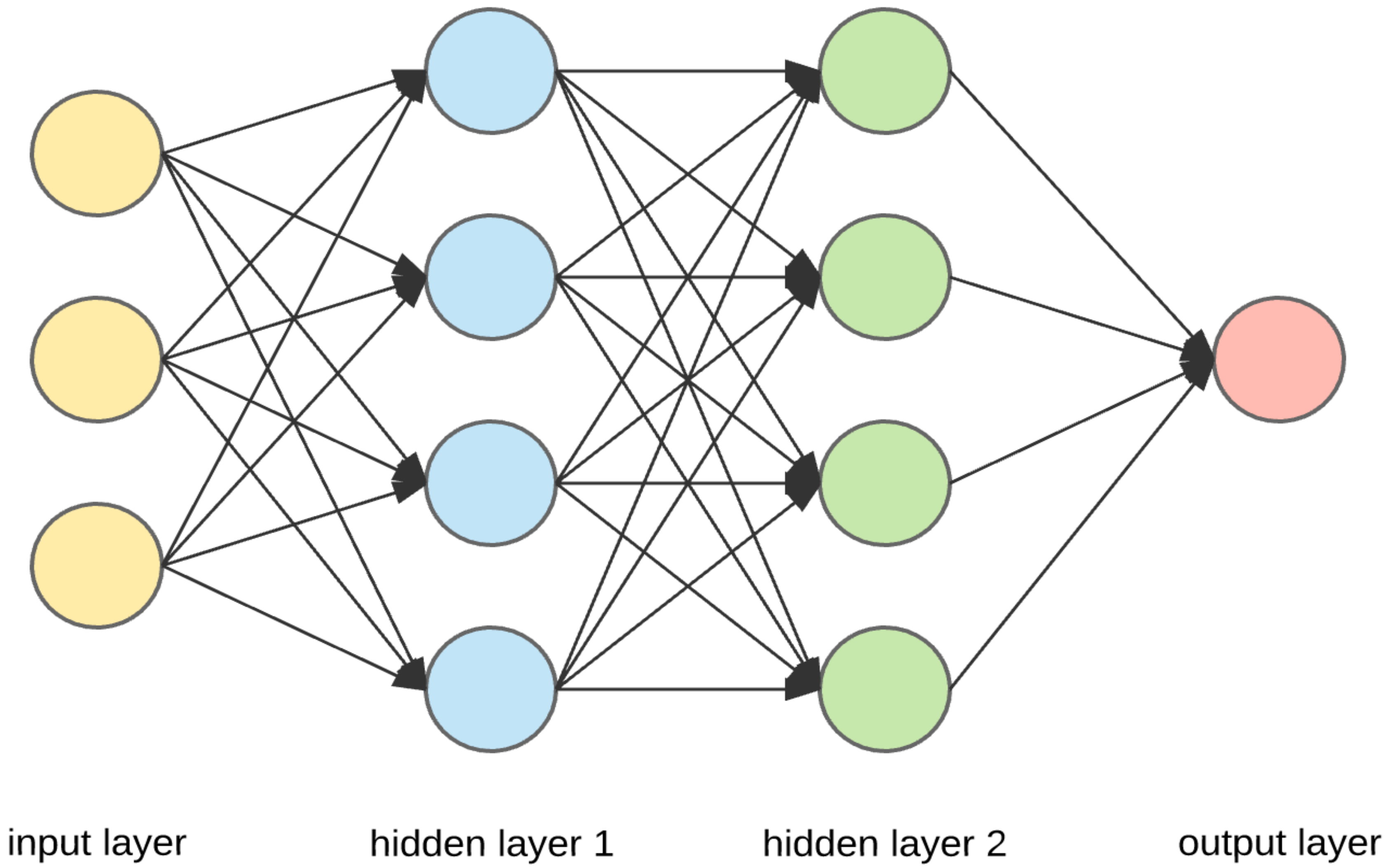
Will it be Cold or Hot tomorrow?

- Both for regression and classification, want to predict some function



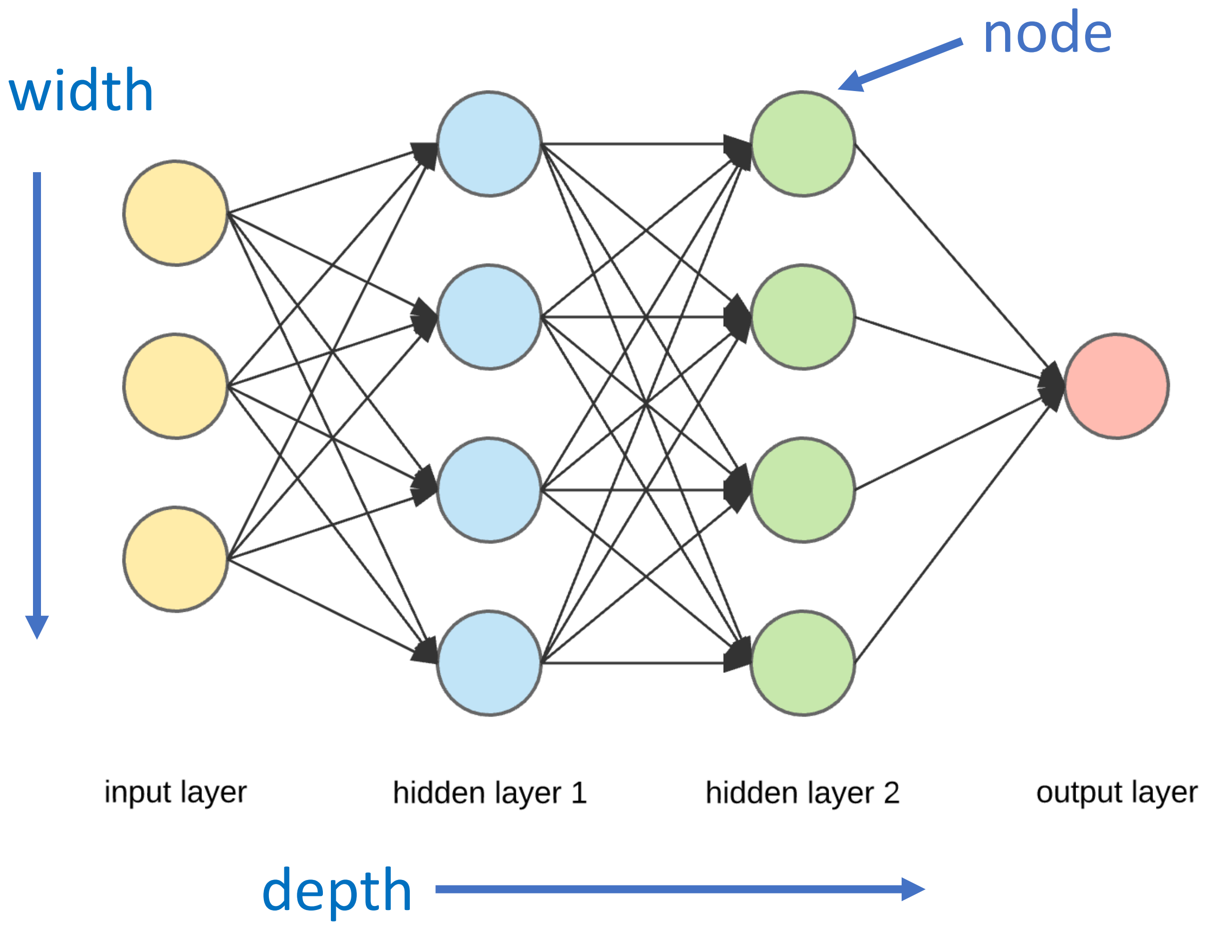
- Use a Neural Network = universal function approximator

Cybenko’89, Hornik et al’89, Hornik’91,…



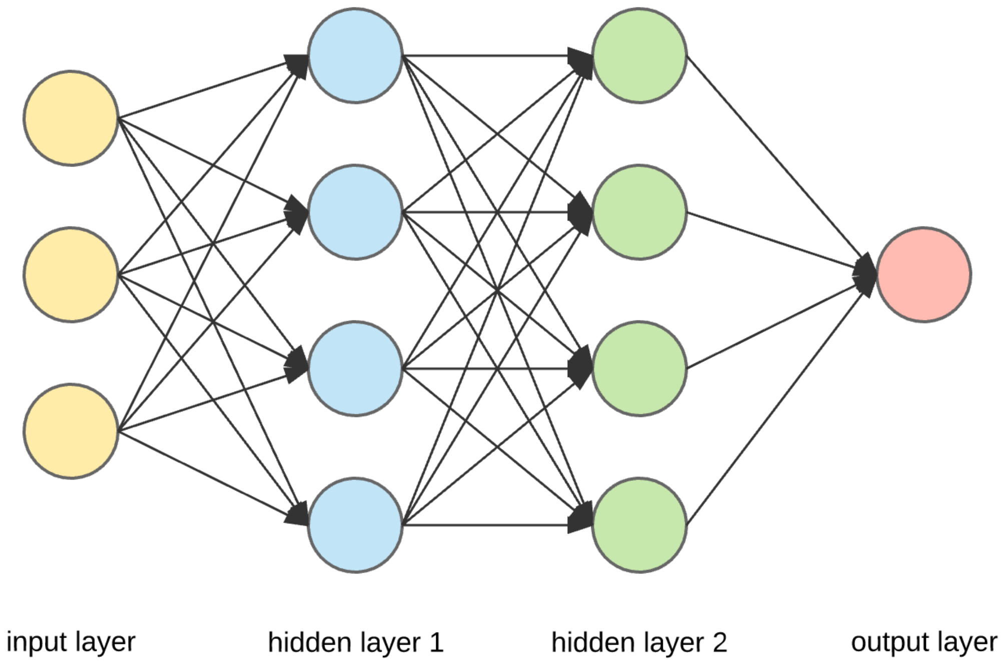
Neural networks

Introducing neural nets



- Input layer = data x
- Output layer = prediction $f_\theta(x)$
- Hidden layers
- Each layers has nodes (neurons)

Introducing neural nets



Hidden layers

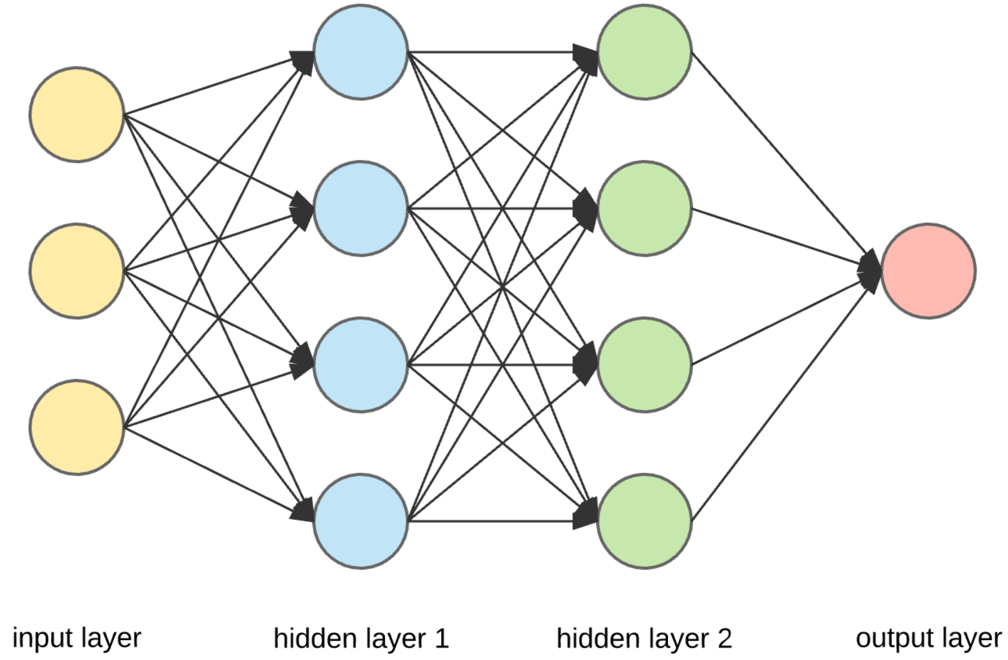
- affine/linear transformation

$$v_k = \underset{\uparrow}{a_k}\left(\overbrace{W_k v_{k-1} + b_k}\right)$$

- nonlinear activation function

The NN is a parametrized map $f_\theta\colon \mathbb{R}^n \to \mathbb{R}^m$ where $\theta = \{W_k, b_k\}$

Introducing neural nets



The NN is a parametrized map

Example: $f_\theta\colon\ \mathbb{R}^3 \to \mathbb{R}^1$

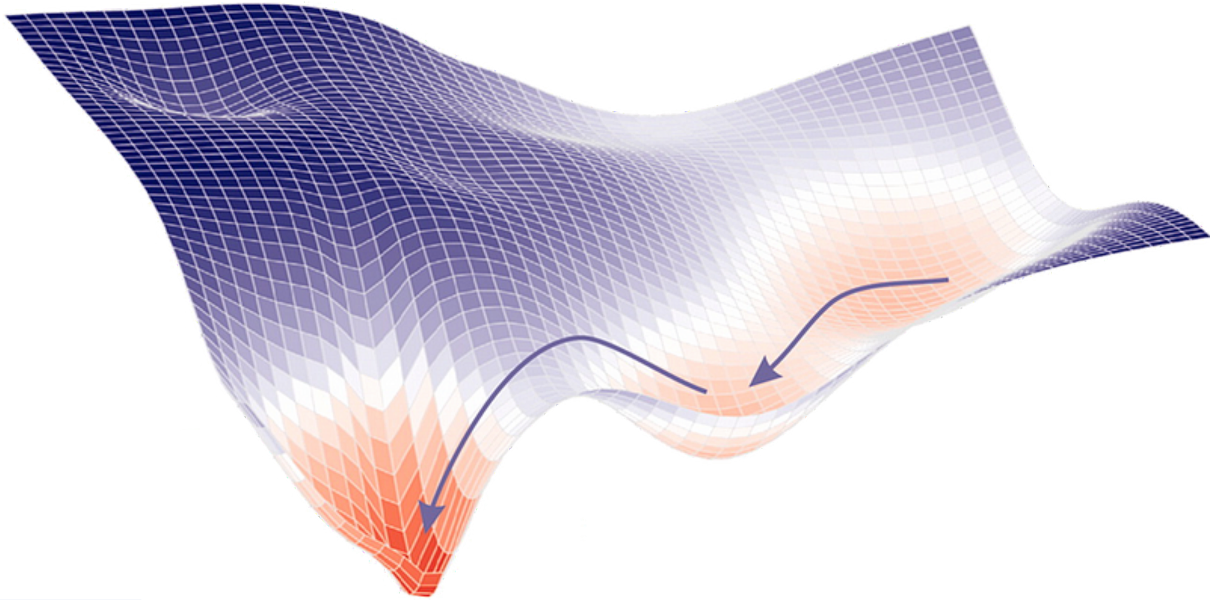
- $v_0 = (x_0, x_1, x_2)^T$
- $v_1 = \begin{bmatrix} a^1(w^1_{00}x_0 + w^1_{01}x_1 + w^1_{02}x_2 + b^1_0) \\ a^1(w^1_{10}x_0 + w^1_{11}x_1 + w^1_{12}x_2 + b^1_1) \\ a^1(w^1_{20}x_0 + w^1_{21}x_1 + w^1_{22}x_2 + b^1_2) \\ a^1(w^1_{30}x_0 + w^1_{31}x_1 + w^1_{32}x_2 + b^1_3) \end{bmatrix}$

Training the network

- Layers with parameters

$$v_k = a_k(W_k v_{k-1} + b_k)$$

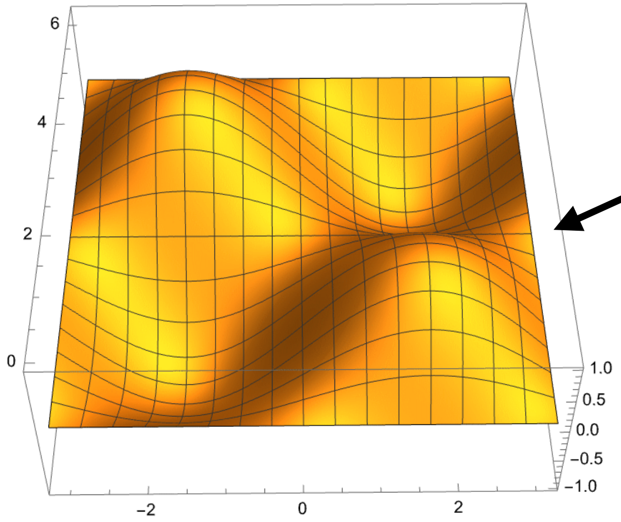
Parameters = weights + bias



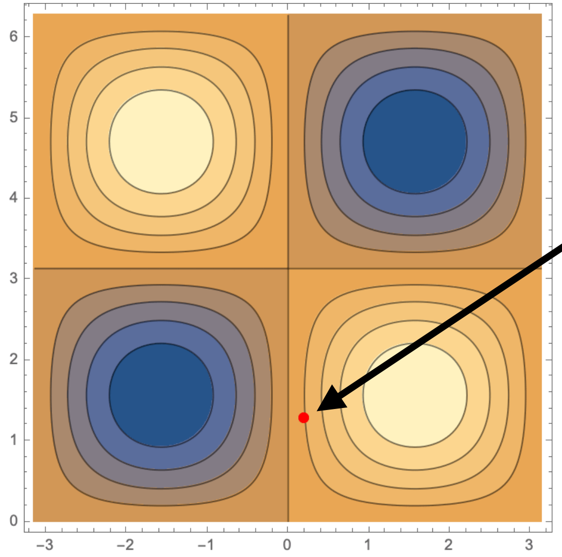
- **Loss function** says how far prediction is from true answer
  Choose so that $L \geq 0$, with = iff $f_\theta(x) = y$

- Aim: (good/global) minimum of loss function (in million-dim'l parameter space)
- Method: tune parameters so loss decreases → **Gradient descent**

Visualization - Gradient descent



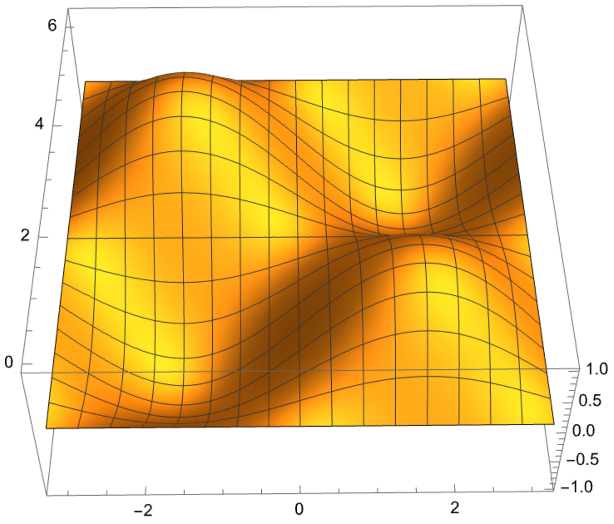
Loss landscape as a function of the NN parameters, and thus as a functional (function of a function) of the NN

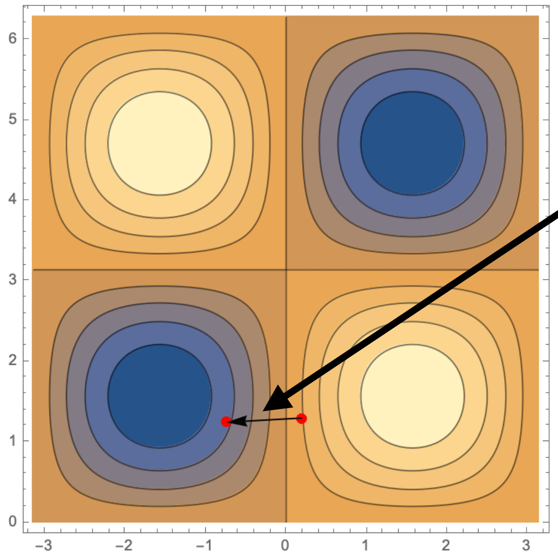


Randomly initialized NN
= Random function
= Random point in loss landscape

Slide credit: Fabian Ruehle, MLMTP school 2023

Visualization - Gradient descent

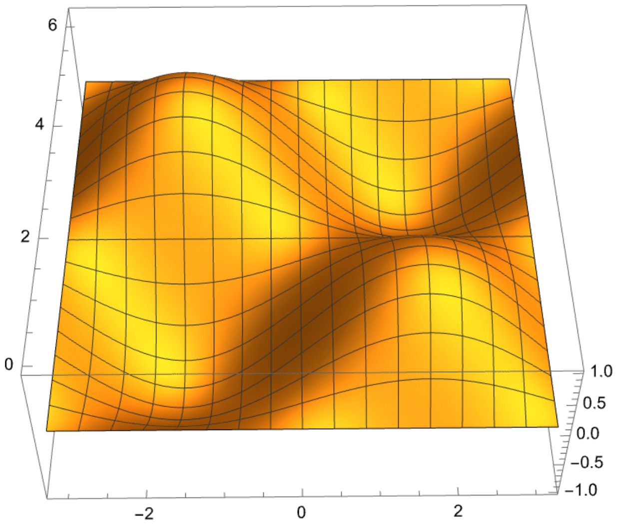


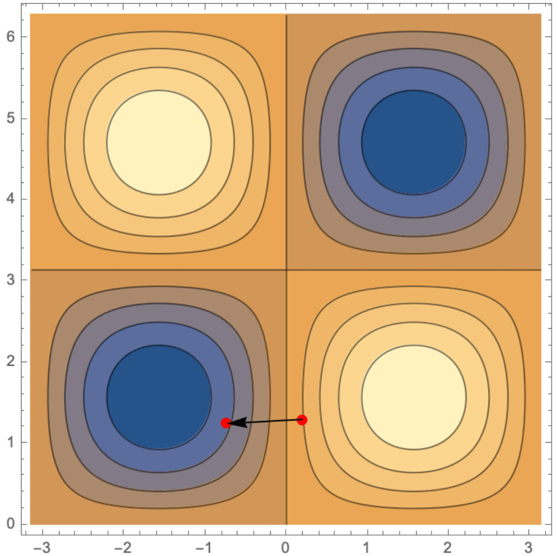


step size = learning rate

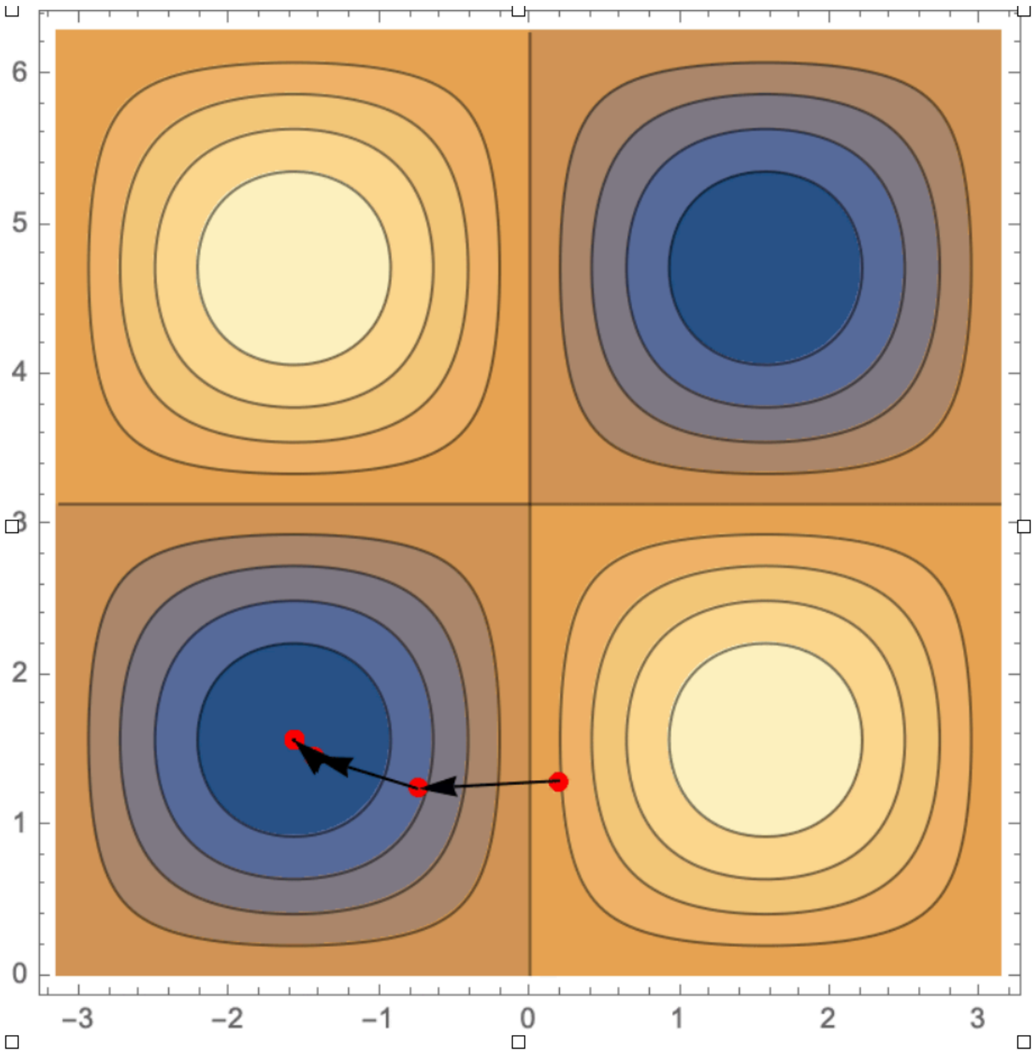
Slide credit: Fabian Ruehle, MLMTP school 2023

Visualization - Gradient descent



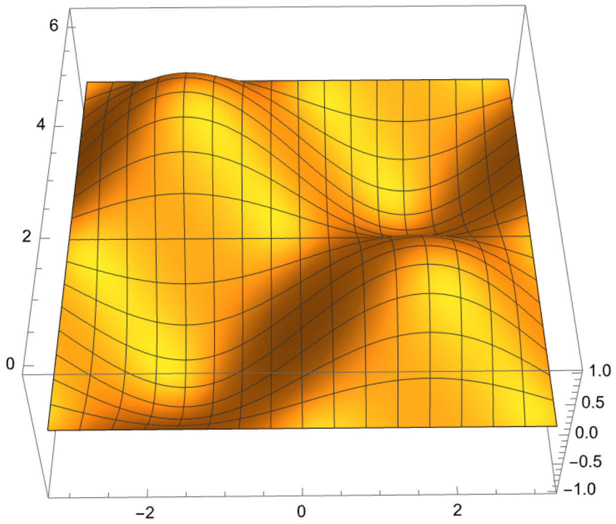


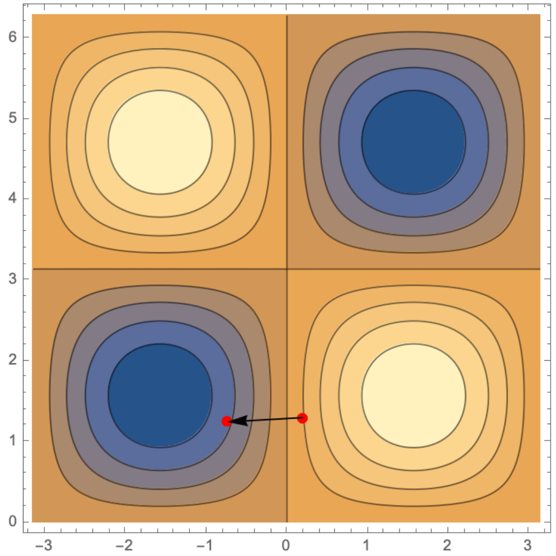
good learning rate



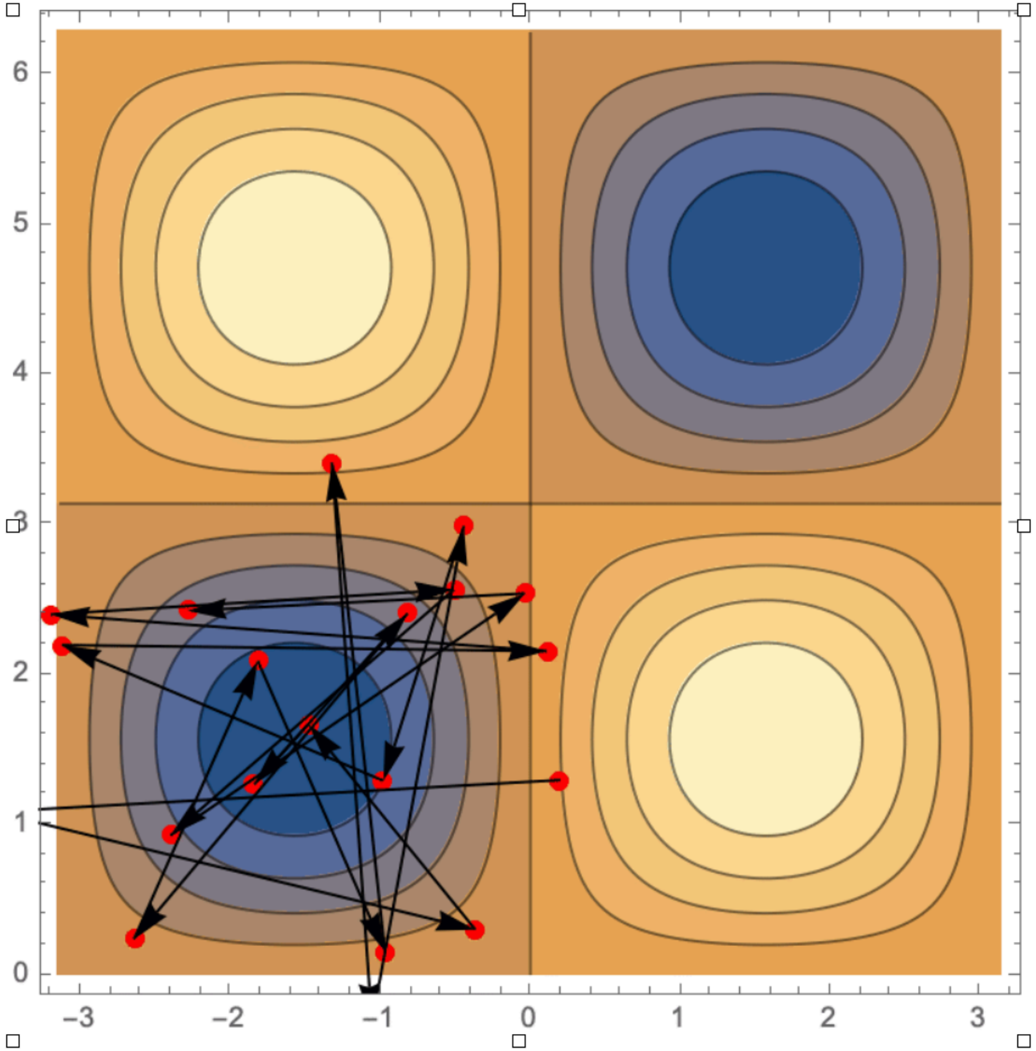
Slide credit: Fabian Ruehle, MLMTP school 2023

Visualization - Gradient descent



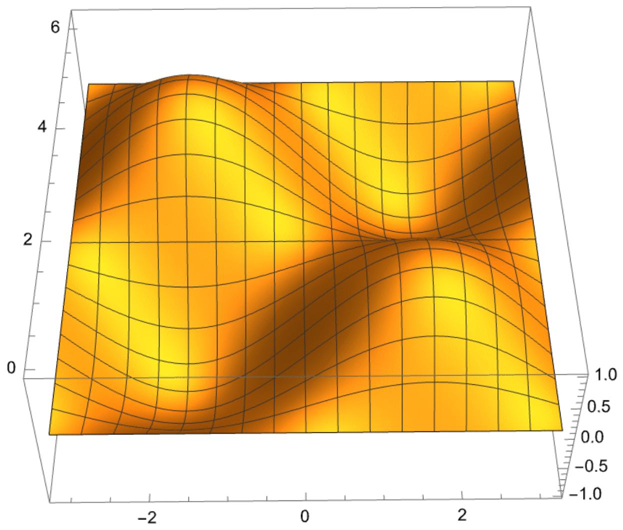


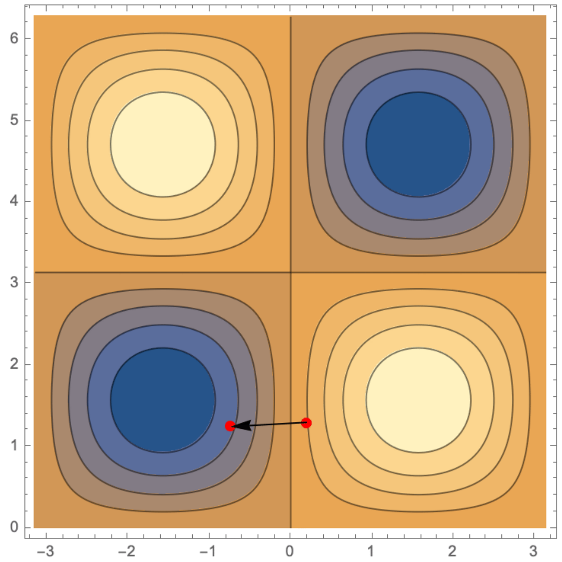
learning rate too big



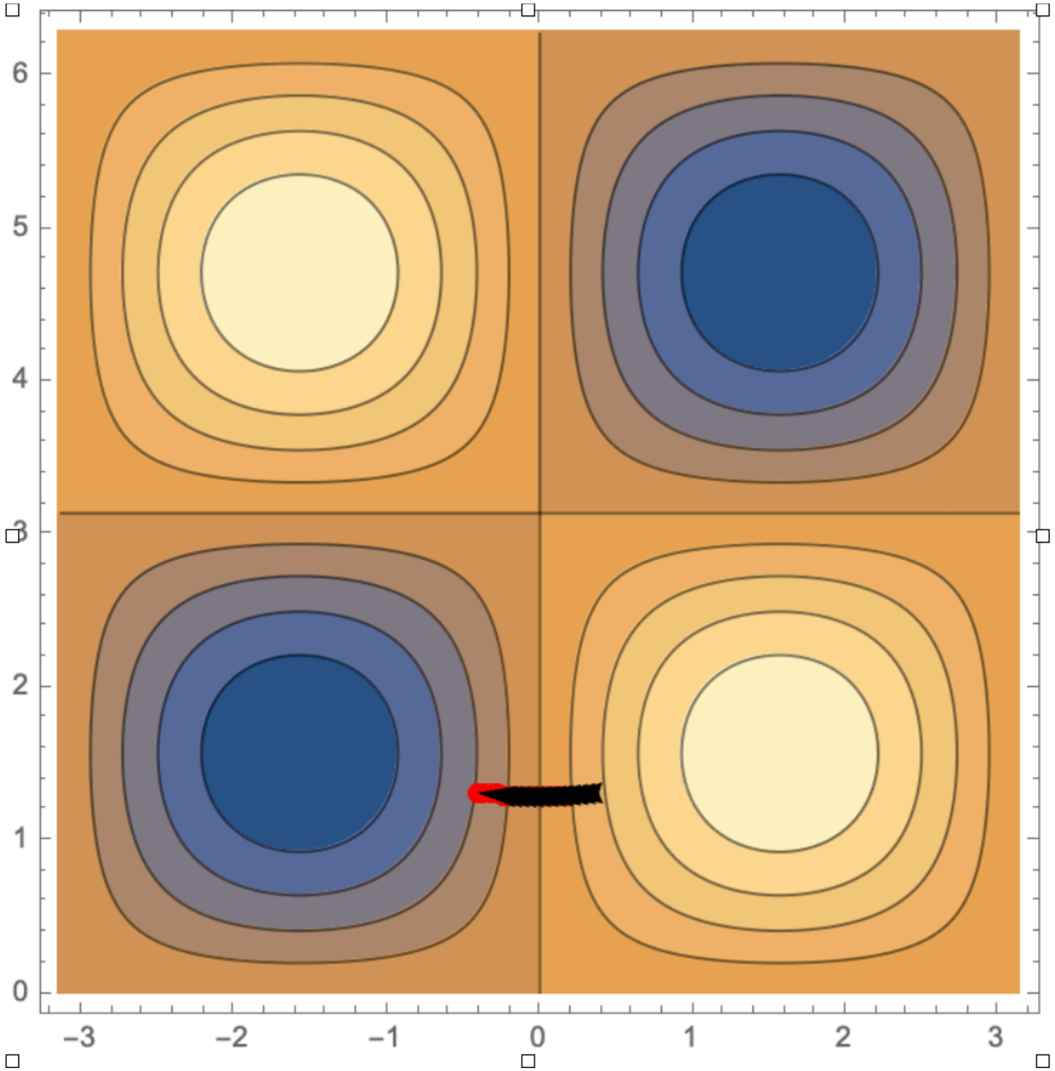
Slide credit: Fabian Ruehle, MLMTP school 2023

Visualization - Gradient descent





learning rate too small



Slide credit: Fabian Ruehle, MLMTP school 2023

Loss functions for supervised learning

Have: labelled data (x,y).
Loss function says how far prediction is from true answer
Choose so that $L \geq 0$, with = iff $f_\theta(x) = y$

Regression:

• MSE (Mean squared error)	• $L_{MSE} = \frac{1}{N}\sum\left(y(x_i) - f_\theta(x_i)\right)^2$
• MAE (Mean absolute error)	• $L_{MAE} = \frac{1}{N}\sum \lvert y(x_i) - f_\theta(x_i)\rvert$
• MAPE (M. a. percentage e.)	• $L_{MAPE} = \frac{1}{N}\sum \lvert \frac{y(x_i) - f_\theta(x_i)}{y(x_i)} \rvert$

Loss functions for supervised learning

Regression:

- MSE (Mean squared error)
- MAE (Mean absolute error)
- MAPE (M. a. percentage e.)

- $L_{MSE} = \frac{1}{N} \sum \left(y(x_i) - f_\theta(x_i) \right)^2$
- $L_{MAE} = \frac{1}{N} \sum |y(x_i) - f_\theta(x_i)|$
- $L_{MAPE} = \frac{1}{N} \sum | \frac{y(x_i) - f_\theta(x_i)}{y(x_i)} |$

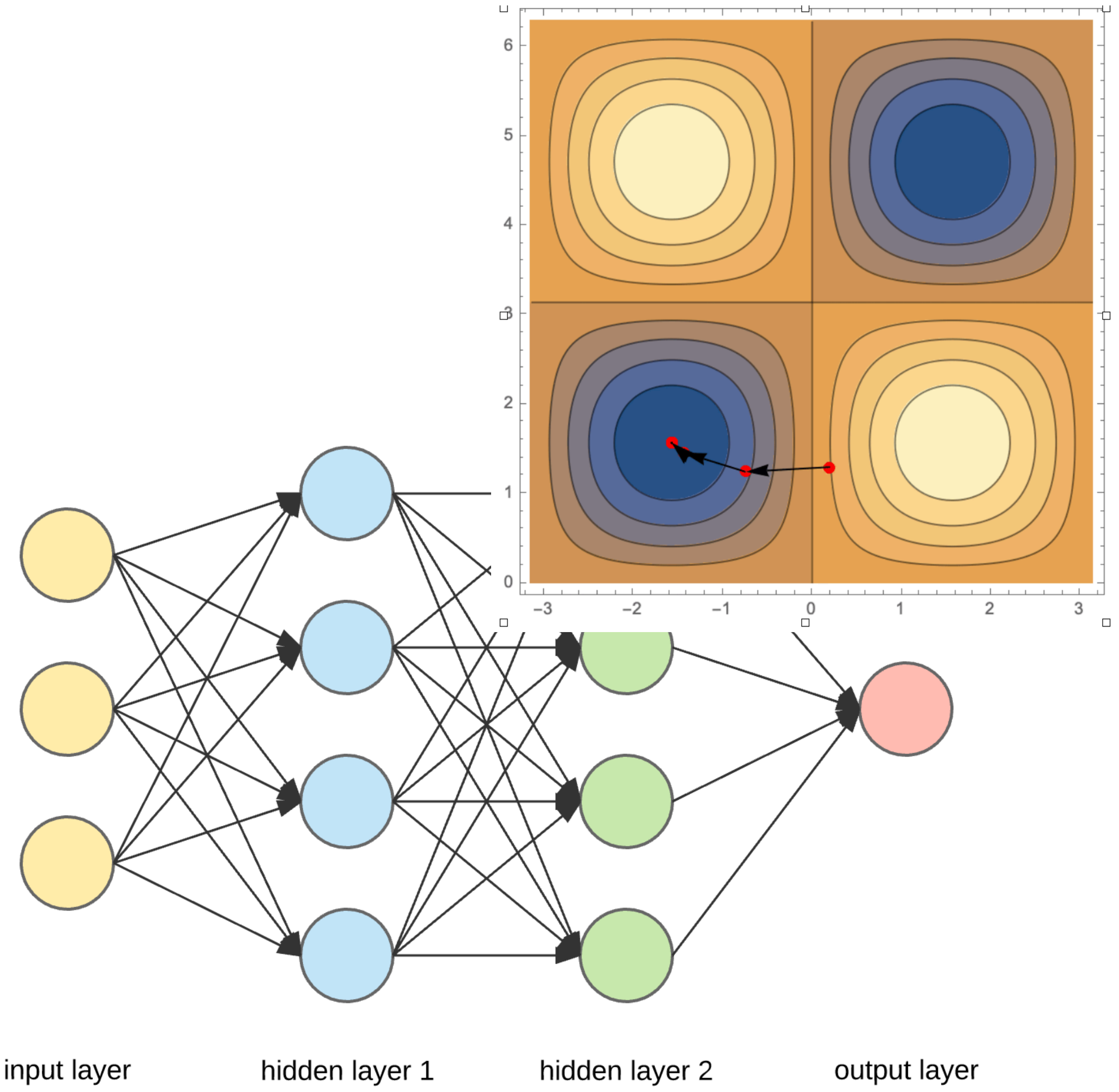
Classification

- Binary cross-entropy

- $L_{BXE} = \frac{1}{N} \sum \left(y(x_i) \log\left(f_\theta(x_i) \right) - [1 - y(x_i)] \log(1 - f_\theta(x_i)) \right)$

Training the network

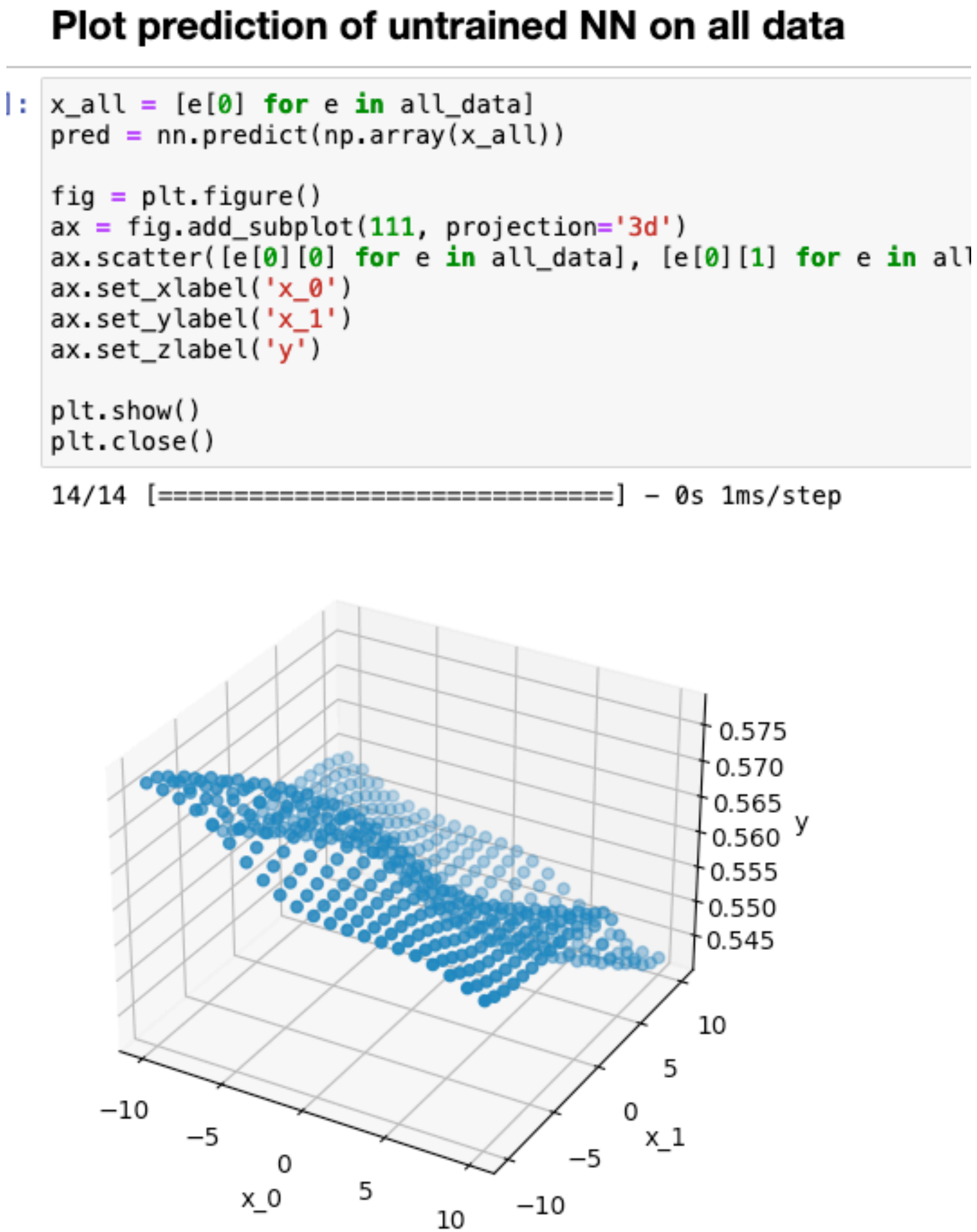
- Parameter updates w. gradient descent
- Hope to find good/global min → result that generalizes to new data
- Divide data into train and test sets; check result generalizes



Demo of simple NN

- Want to machine learn function

- $y(x_1, x_2) = \begin{cases} 1\,, & x_1 x_2 < 0 \\ 0, & x_1 x_2 > 0 \end{cases}$



Plot prediction of untrained NN on all data

```
]: x_all = [e[0] for e in all_data]
   pred = nn.predict(np.array(x_all))

   fig = plt.figure()
   ax = fig.add_subplot(111, projection='3d')
   ax.scatter([e[0][0] for e in all_data], [e[0][1] for e in al
   ax.set_xlabel('x_0')
   ax.set_ylabel('x_1')
   ax.set_zlabel('y')

   plt.show()
   plt.close()
```

```
14/14 [==============================] - 0s 1ms/step
```

Compare w ch 2 of https://github.com/ruehlef/Physics-Reports by Fabian Ruehle (physics context [1706.07024](1706.07024))

How can training be so quick?

- Gradient descent needs computing derivatives
  – should take time, scaling problems (eg finite difference methods)

- ML libraries → efficient implementations of automatic differentiation
- Backpropagation: chain rule + info from ”forward pass”

Backpropagation

- Have computed/initialized NN $f_\theta\colon \mathbb{R}^n \to \mathbb{R}^m$ where $\theta = \{W_k, b_k\}$
- This means we know, for each layer
  post-activation $v^k = a^k(z^k)$ pre-activation $z^k_\mu = W^k_{\mu\nu}\, v^{k-1}_\nu + b^k_\mu$

- Compute gradients of last layer

$$\frac{\partial\, L}{\partial\, \theta^n} = \frac{\partial\, L}{\partial\, z^n} \frac{\partial\, z^n}{\partial\, \theta^n}$$

$$\frac{\partial z^k_\mu}{\partial\, \theta^n} = \begin{cases} \dfrac{\partial z^k_\mu}{\partial W^k_{\lambda\,\nu}} = \delta_{\mu\lambda} v^{k-1}_\nu \\[2ex] \dfrac{\partial z^k_\mu}{\partial b^k_\lambda} = \delta_{\mu\lambda} \end{cases}$$

e.g. $\dfrac{\partial\, L_{MSE}}{\partial\, z^n} = \dfrac{2}{N}[y - a^n(z^n)]a'^n(z^n)$

Backpropagation

- Have computed/initialized NN $f_\theta\colon\ \mathbb{R}^n \to \mathbb{R}^m$ with $\theta = \{W_k, b_k\}$
- This means we know, for each layer
  post-activation $v^k = a^k\left(z^k\right)$ pre-activation $z^k_\mu = W^k_{\mu\nu}\, v^{k-1}_\nu + \ b^k_\mu$

- Compute gradients of last layer

$$\frac{\partial\ L}{\partial\ \theta^n} = \frac{\partial\ L}{\partial\ z^n}\frac{\partial\ z^n}{\partial\ \theta^n}$$

$$\frac{\partial z^k_\mu}{\partial\ \theta^n} = \begin{cases} \dfrac{\partial z^k_\mu}{\partial W^k_{\lambda\,\nu}} = \delta_{\mu\lambda} v^{k-1}_\nu \\ \dfrac{\partial z^k_\mu}{\partial b^k_\lambda} = \delta_{\mu\lambda} \end{cases}$$

e.g. $\frac{\partial\ L_{MSE}}{\partial\ z^n} = \frac{2}{N}[y - a^n(z^n)]a'^n(z^n)$

All known from the “forward pass”

Backpropagation

- Have computed/initialized NN $f_\theta\colon\ \mathbb{R}^n \to \mathbb{R}^m$ with $\theta = \{W_k, b_k\}$
- This means we know, for each layer
  post-activation $v^k = a^k\left(z^k\right)$ pre-activation $z_\mu^k = W_{\mu\nu}^k\, v_\nu^{k-1} + \ b_\mu^k$

- Compute gradients of layer i

$$\frac{\partial\ L}{\partial\ \theta^i} = \frac{\partial\ L}{\partial\ z^n}\,\frac{\partial\ z^n}{\partial\ z^{n-1}}\,\frac{\partial\ z^{n-1}}{\partial\ z^{n-2}}\ldots\,\frac{\partial\ z^i}{\partial\ \theta^i}$$

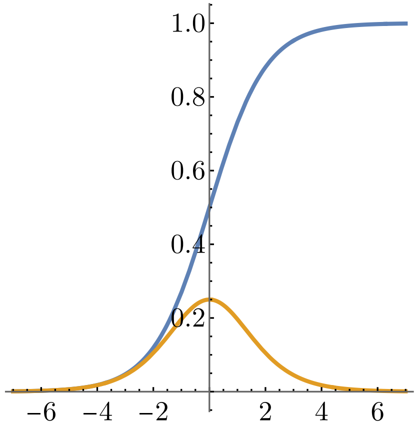
again known from forward pass

- Update parameters (step size α)

$$\theta^i \to \theta^i - \alpha\,\frac{\partial\ L}{\partial\ \theta^i}$$

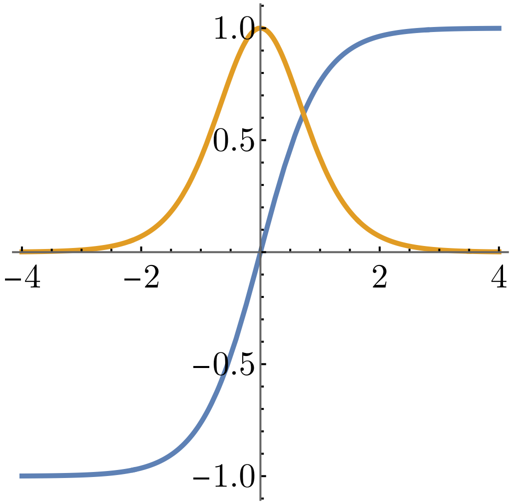
Nice activation functions

- Need the derivatives $a'(z)$
- Nice activation functions: derivative known from forward pass.
- Ex: sigmoid, tanh, ReLU:



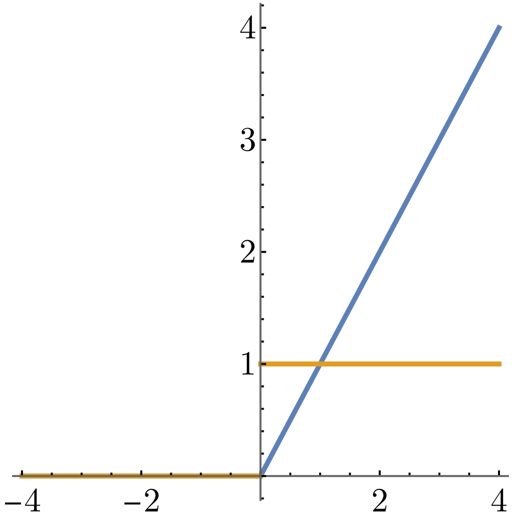
$$a(x) = \sigma(x) = 1/(1 + e^{-x})$$

$$a'(x) = a(x)[1 - a(x)]$$



$$a(x) = \tanh(x)$$

$$a'(x) = 1 - [a(x)]^2$$



$$a(x) = \mathrm{ReLU}(x) = \max(0, x)$$

$$a'(x) = \theta(x)$$

SGD and mini-batches

- Minimizing over all data has problems
    - Large data sets need too much memory
    - Get stuck in local minima/saddles

- Divide data into mini-batches and run GD updates batch by batch
- Stochastic update of parameters (see partial info in each batch)
- One epoch = one run over full data set. Train for several epochs.

Summary SGD and Backpropagation

- Goal: Update parameters to minimize loss function
- Chain rule -> gradients $g_t = \frac{\partial\, L}{\partial\, \theta^i} = \frac{\partial\, L}{\partial\, z^N} \frac{\partial\, z^N}{\partial\, z^{N-1}} \ldots \frac{\partial\, z^i}{\partial\, \theta^i}$
- Derivatives known using info from forward pass
- Update parameters $\theta^i_{t+1} = \theta^i_t - \alpha_t \frac{\partial\, L}{\partial\, \theta^i}$
- Updates done after run of one mini-batches
  One epoch = one run over full data set. Train for several epochs.

- Implemented in ML libraries s.a. TensorFlow/Keras, PyTorch, JAX

Alternative optimization methods

- Variants of SGD aim to get
  - Faster convergence
  - Less problems with local min and saddles
- **RMSprop**: Root mean square propagation

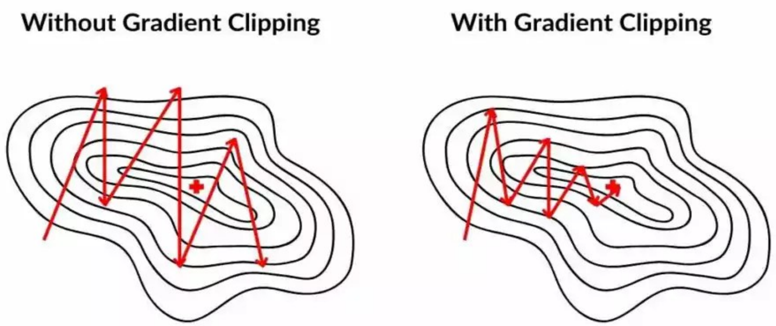
$$g_t = \frac{\partial\, L}{\partial\, \theta} \qquad s_t = \beta\, g_t + (1-\beta) g_t^2 \quad \Rightarrow\; \theta_{t+1} = \theta_t \quad -\alpha_t \frac{g_t}{\sqrt{s_t + \epsilon}}$$

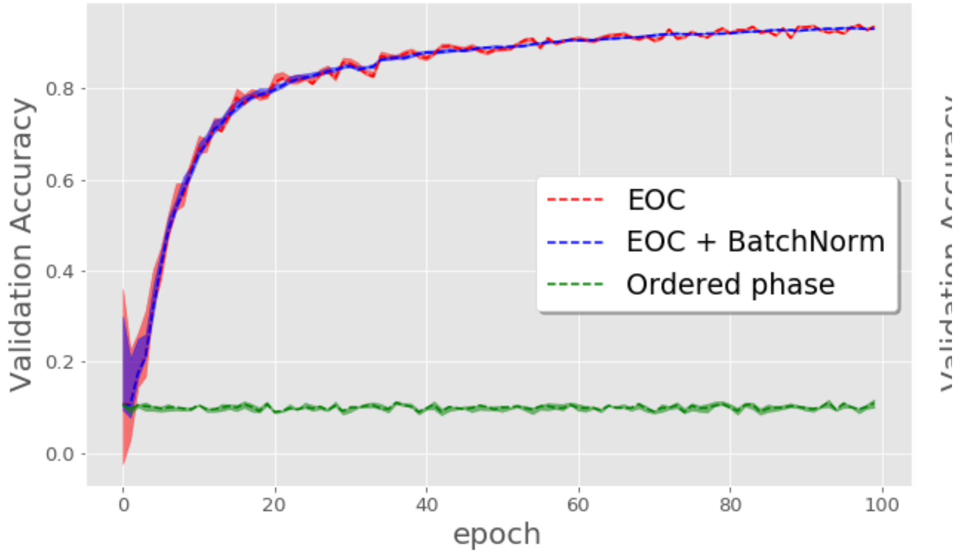
this adapts learning rate to gradient size

- **Adam**: even more refined; adapts learning rate and momentum

Known problems/challenges

- Gradients at layer i is proportional to gradient at layer $i + 1$
  → gradients can vanish/explode at early layers
- Counteract this using
  - gradient clipping,
  - batch normalisation,
  - NN initialized on the “edge of chaos”





Hayou et al 1902.06853

Hyperparameters and network architectures

- **Hyperparameters**: width, depth, learning rate, …
  How get optimal values? Need (systematic) experiments!

- Fully connected NNs is the simplest **architecture**
- More advanced layers:
  Convolutional, dropout, recurrent, …
  Transformer architecture with attention mechanism --> (chat)-GP**T**
- We will not explain this here. Lot’s of literature & online resources!

Summary of this lecture

- Neural Nets are universal function approximators
- NNs are parametrized maps $f_\theta\colon\ \mathbb{R}^n \to \mathbb{R}^m$ w billion parameters θ
  Layers with nodes; affine tf - weight & bias; non-linear activations
- Stochastic Gradient Descent with Backpropagation
- Use ML libraries: PyTorch, JAX, TensorFlow/Keras
- Hyperparameter optimization and network architectures
  → you need time (and algorithms) to experiment

Plan for afternoon studies

- Reading:
  - R. Schneider "Heterotic Compactifications in the Era of Data Science", ch. 2 http://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A1649343&dswid=-2157
  - F. Ruehle. "Data science applications to string theory" ch 2-3 https://www.sciencedirect.com/science/article/pii/S0370157319303072
  - P. Mehta, *et.al* "A high-bias, low-variance introduction to ML for physicists" https://www.sciencedirect.com/science/article/pii/S0370157319300766
- Online tutorials:
  - ML and backprop by Callum Brodie
  - Simple classification NN: ch 2 of https://github.com/ruehlef/Physics-Reports by Fabian Ruehle (physics context 1706.07024)