Deep Equilibrium Nets for Solving Dynamic Stochastic Models

MCC, 03-04 September 2024

Lecture 1: Intorduction to deep neural nets

Aleksandra Friedl

https://github.com/alexmalova/DEQN_workshop

friedl@ifo.de

This course is inspired by Simon Scheidegger and based on his various teaching materials that I was relying on as a student and those which came as a result of our scientific collaboration

Course information, day 1

Day 1, Tuesday, September 3rd, 2024

Time	Main Topics
09:00 - 10:00	Introduction to ML and deep learning
10:00 - 11:00	Practical Session on SGD
10:40 - 12:00	Training the Neural Net
12:00 - 13:30	Lunch Break
13:30 - 14:30	Practical Session on DNN
14:30 - 15:30	DEQN
15:30 - 16:30	Practical Session on DEQN

Course information, day 2

Day 2, Wednesday, September 4th, 2024

Time	Main Topics
09:00 - 10:00	Stochastic Problems
10:00 - 11:00	Practical Session on DEQN, stochastic
11:00 - 12:00	Intorduction to DEQN software
12:00 - 13:30	Lunch Break
13:30 - 14:30	Practical Session, DEQN, Brock-Mirman 1972
14:30 - 15:30	Practical Session, DEQN, Ramsey model
15:30 - 16:30	Practical Session, DEQN, stochastic Ramsey model

Most recent info and course materials

- **Github**: https://github.com/alexmalova/DEQN_workshop
- **Nuvolos** enrollement key: https://app.nuvolos.cloud/enroll/class/c7Jt2zmJ4iM

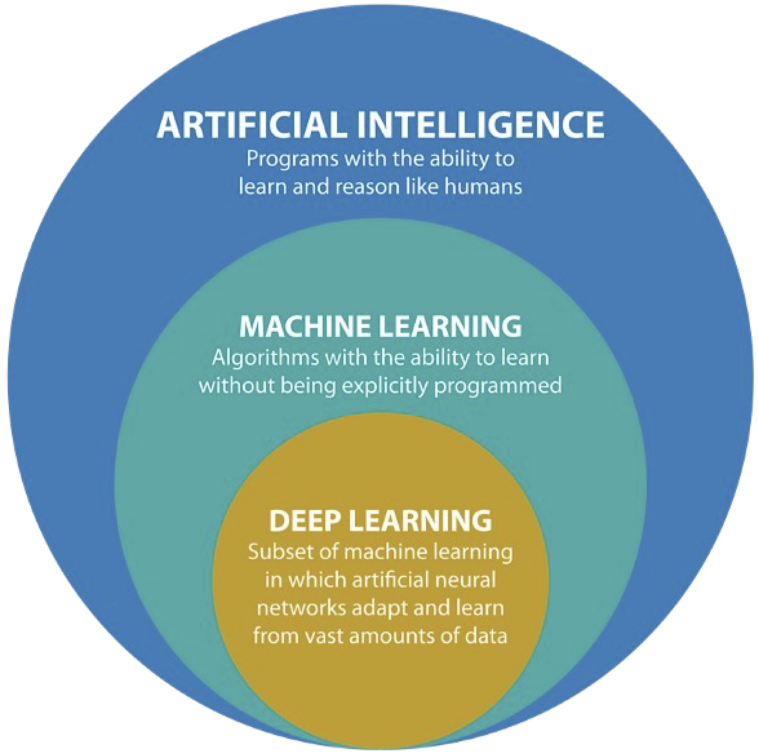
Info about me

- **Assistant Professor** (non-tenure track) at the University of Munich, **Economist** at ifo Institute
- **Ph.D. in Economics** from the University of Lausanne, 2023 *(supervisors: Simon Scheidegger, Kenza Benhima)*
- **Research interests**: macroeconomics of climate change, green energy transition, growth and development, applied quantitative economics
- **Webpage**: https://sites.google.com/view/aleksandrafriedl
- **Email**: friedl@ifo.de

Some terminology

- **Artificial Intelligence (AI)**: the effort to automate intellectual tasks normally performed by humans
- **Machine Learning (ML)**:

- **Deep Learning**: a particular example of an ML technique



figures from F. Chollet (2018)

Types of Machine Learning

- **Supervised learning**: the training data you feed to the algorithm includes the desired solutions, called labels
  - Classification: e.g. spam filter
  - Regression: e.g. car price based on car features
- **Unsupervised learning**: the training happens without labeling the data
  - Clustering: e.g. customer segmentation
- **Reinforcement learning**: the learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return

Building an ML algorithm I

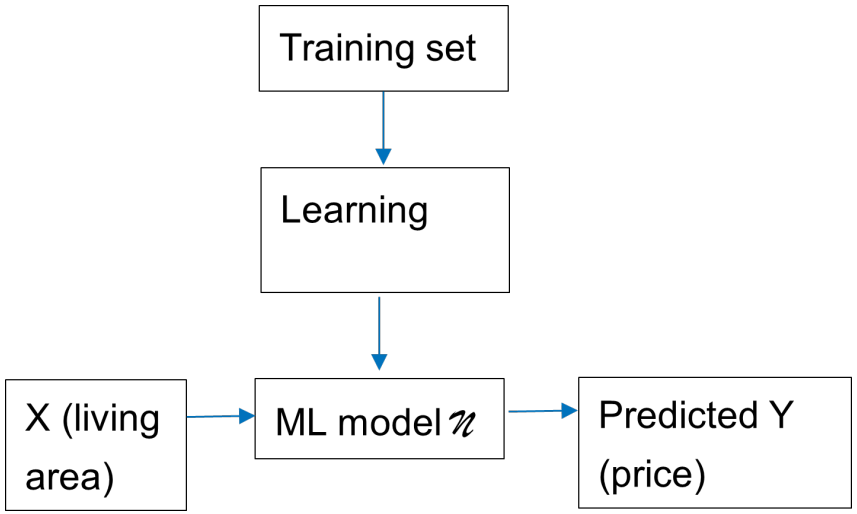
Living area (sq.feet)	Price (1000USD)
2104	400
1600	330
2400	369
1416	232
3000	540
...	...

Question

Given data like this, how can we learn to predict the prices of other houses as a function of the size of their living areas?

Building an ML algorithm II

- x_i **input variables** (input features)
- y_i **output variable** (target variable)
- (x_i, y_i) **training example**
- $\{(x_i, y_i) : i = 1, 2, 3, ..., m\}$ **training set**



Training model

To perform supervised learning, we must decide how we're going to represent functions/hypotheses/model $\mathcal{N}$ in a computer.

Building an ML algorithm II

- **Model**: $\mathcal{N}_\rho(x) = \rho_0 + \rho_1 x_1$, where $\rho = (\rho_0, \rho_1)$ are parameters of the model
- **Cost (loss) function**: $J(\rho) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathcal{N}_\rho \left(x^{(i)} \right) - y^{(i)} \right)^2$
- We want to minimize $J(\rho)$ to obtain coefficients ρ

Available frameworks for ML

- Keras
- **Tensorflow**
- Pytorch
- Caffe
- Scikit-learn
- ... you name it

Outline of the presentation

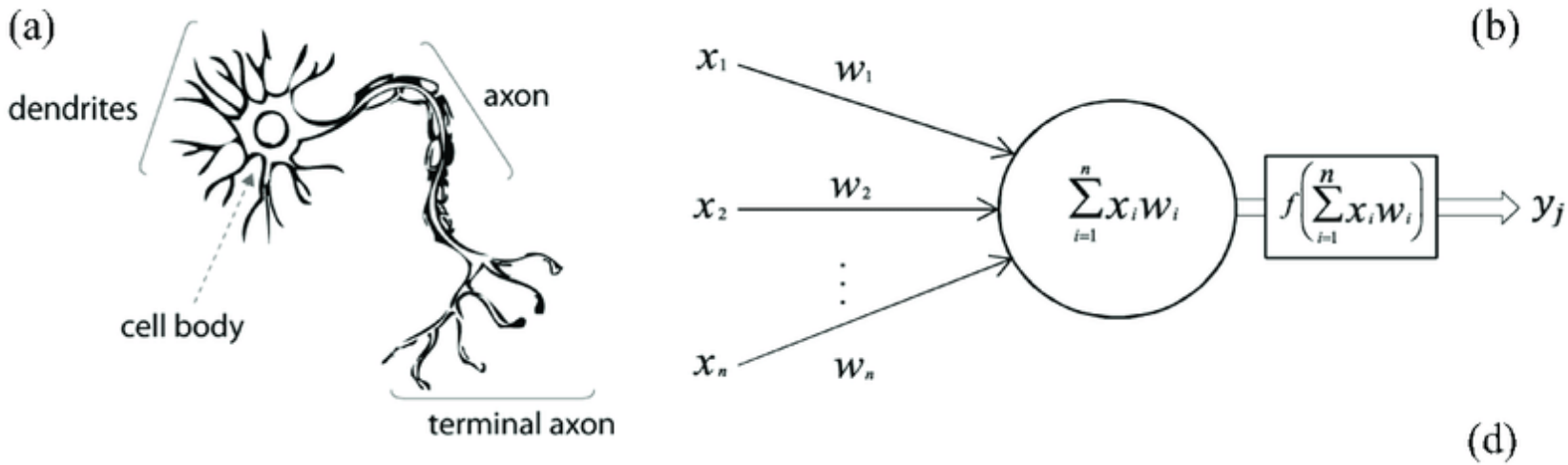
1 Course information

2 Brief (largely incomplete) overview of Machine Learning

3 Introduction to Artificial Neural Networks (ANNs)
- Building a NN
- Loss function
- (Stochastic) Gradient Descent
- Backpropagation

4 Practical session

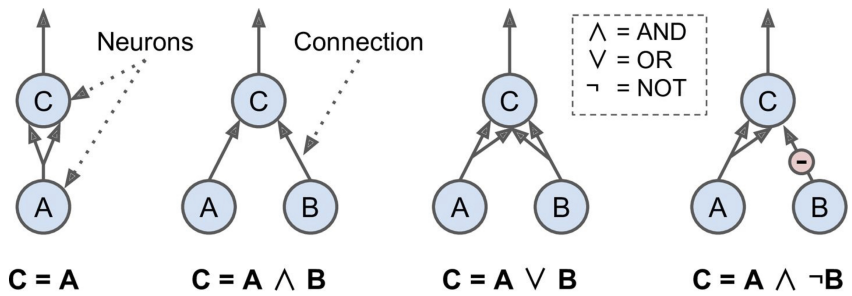
Artificial Neural Networks



[Meng, Hu, and Ancey(2020)]

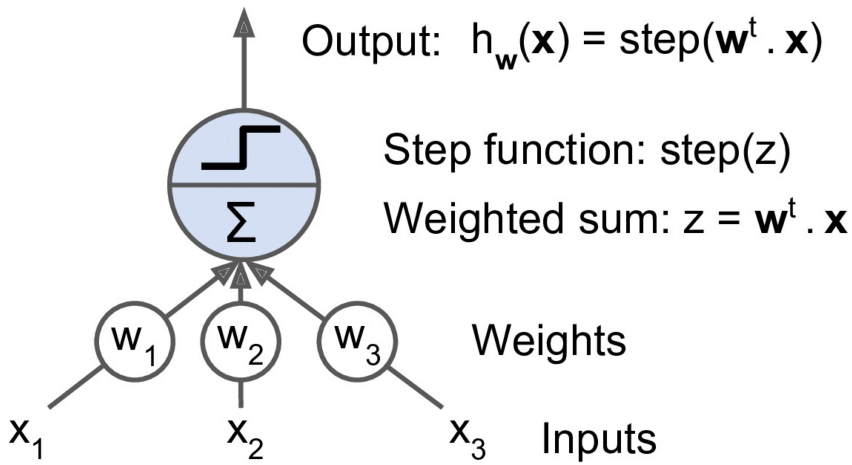
First artificial neuron (1943)

- McCulloch and Pitts' (1943) suggested a first mathematical model of a neuron:
  - one or more binary (on/off) inputs and one binary output
  - The artificial neuron simply activates its output when more than a certain number of its inputs are active.

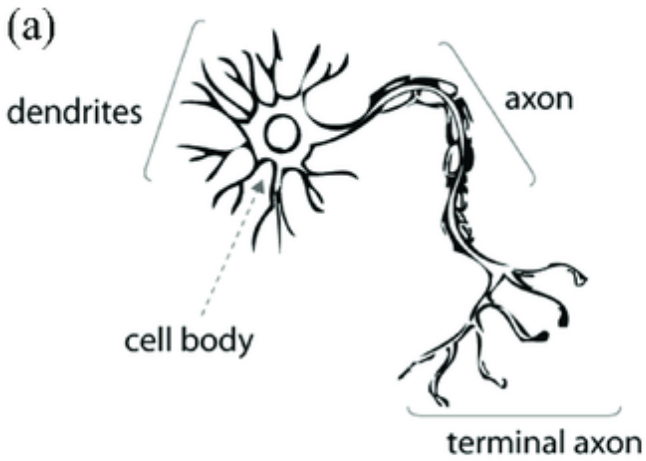


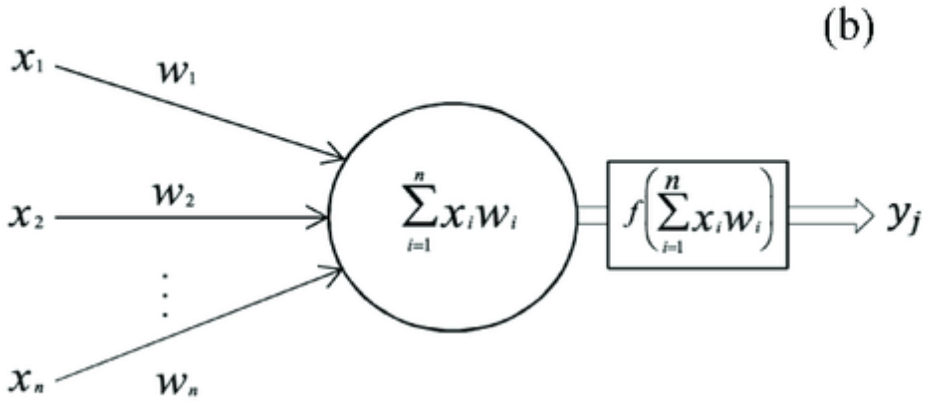
Perceptron (1957)

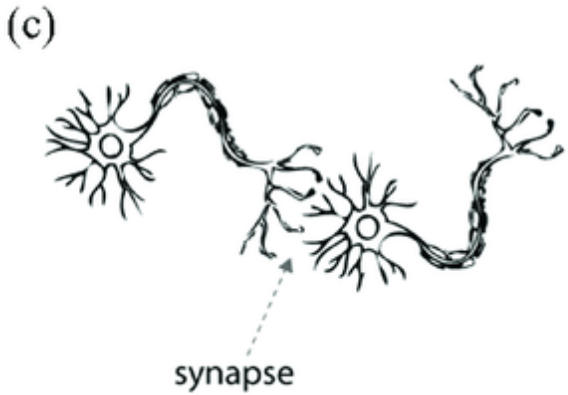
- Frank Rosenblatt (1957) suggested a simplest ANN architecture.
- The inputs x_i are multiplied by the weights w_i , and the neurons sum their values as z.
- If this sum is greater than the threshold ($step(z)$) then the neuron fires
- Neuron training:
  $w_{i,j}^{next} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right) x_i$, where
  η - learning rate

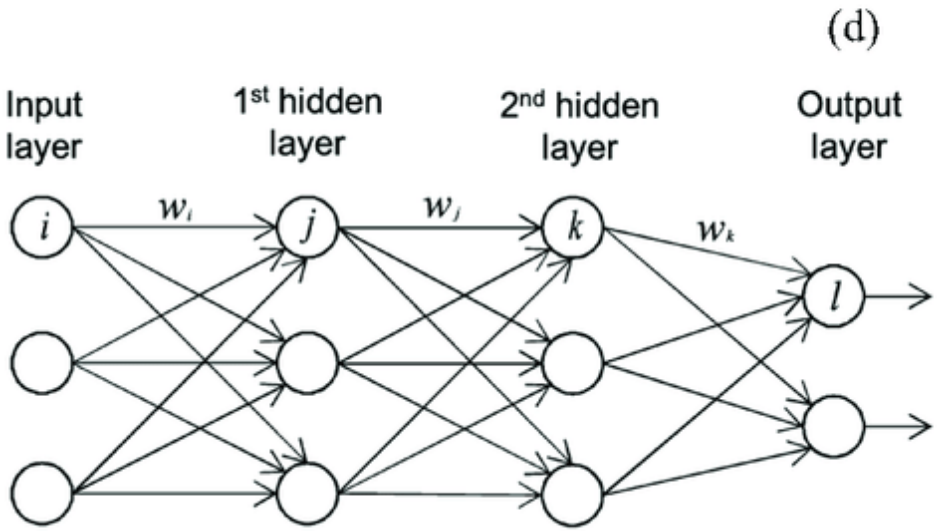


Artificial Neural Networks



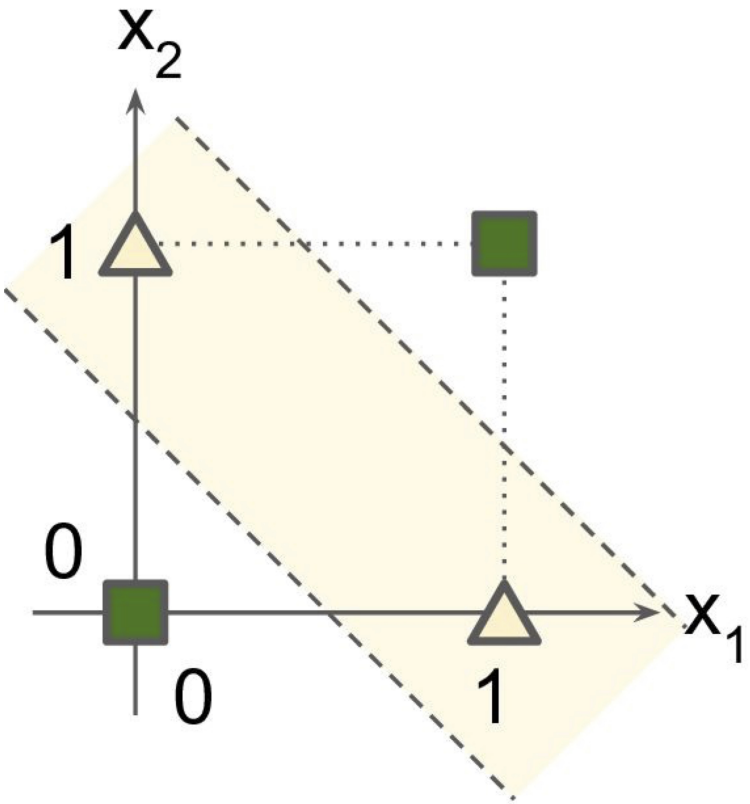


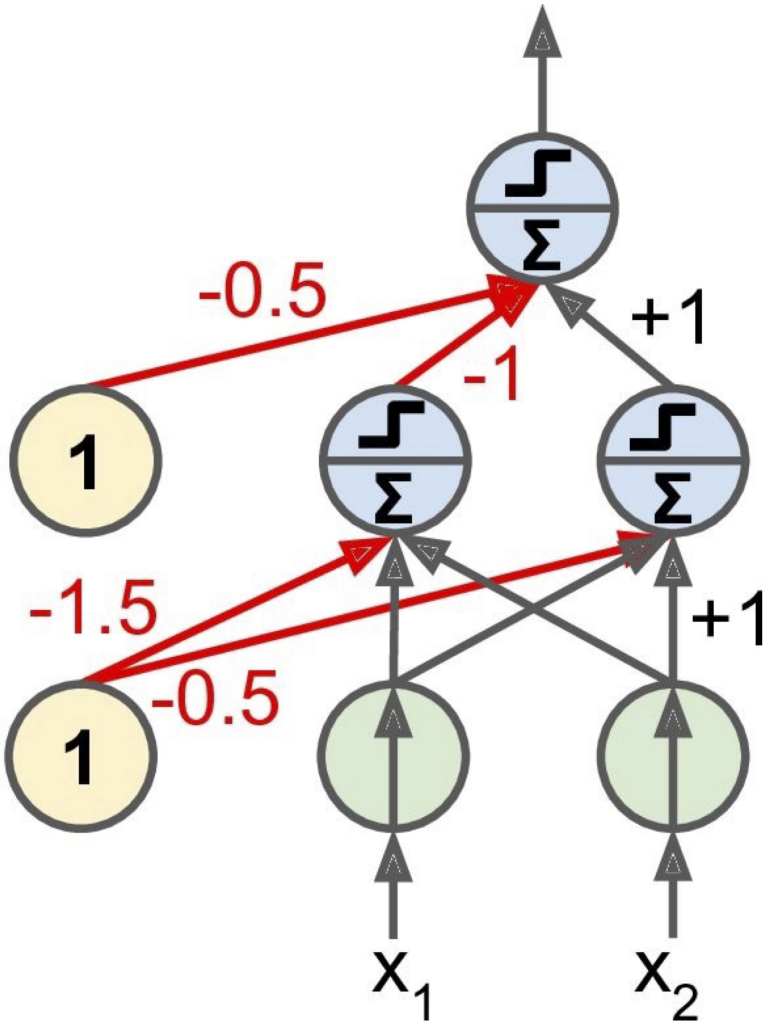




[Meng, Hu, and Ancey(2020)]

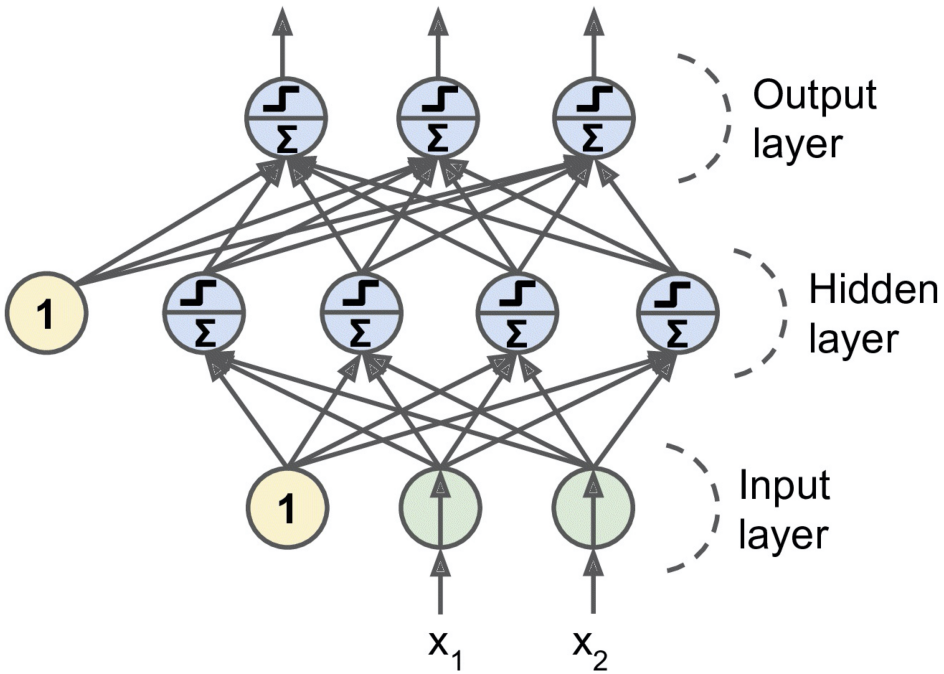
Multi-Layer Perceptron (example)



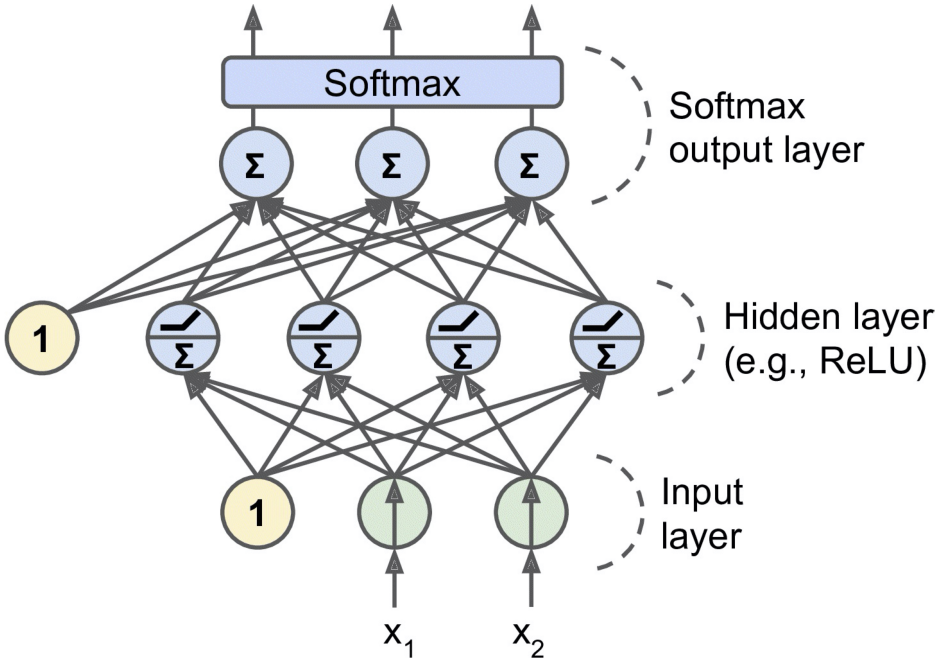


Activation function and backpropagation (1986)

In 1986, Rumelhart et al. suggested a modification to the ANN to enable backpropagation:



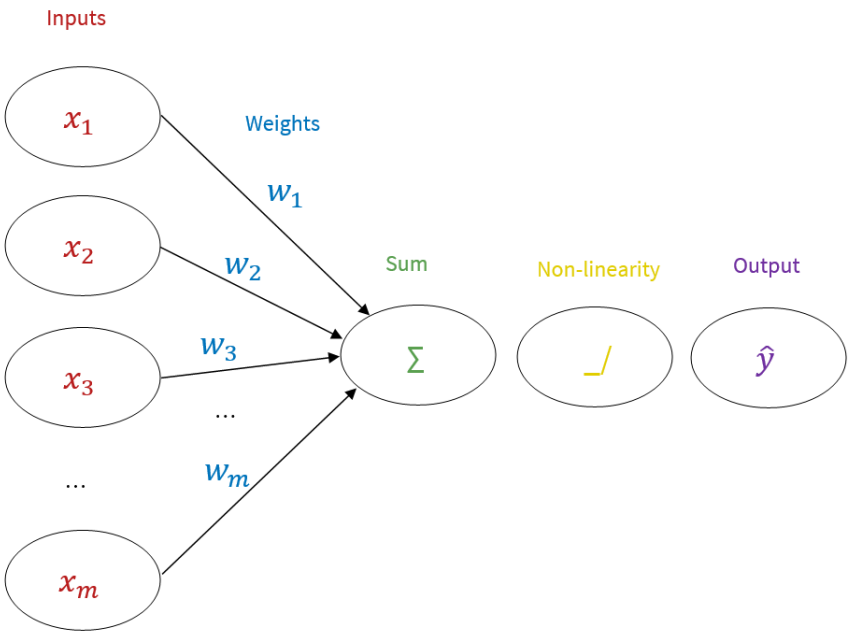
Step-function is flat - no gradients possible!



Introduce non-linear activation function which enables backpropagation

Perceptron: forward propagation

A modern neuron (perceptron) structure:



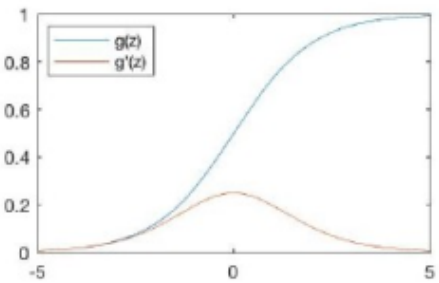
A way to get an output mathematically:

- $\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$
- Activation function $g(z)$
  - Sigmoid
  - tanH
  - ReLU
  - seLU
  - etc...

Activation functions

Here are some examples for the most popular activation functions:

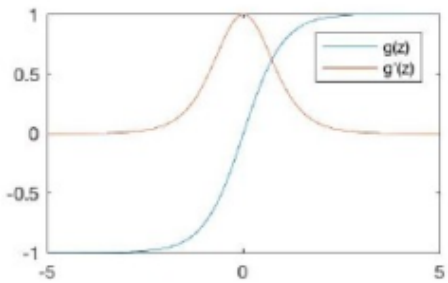
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

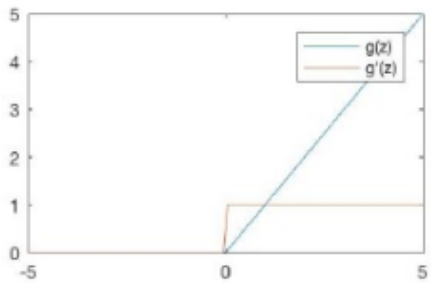
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



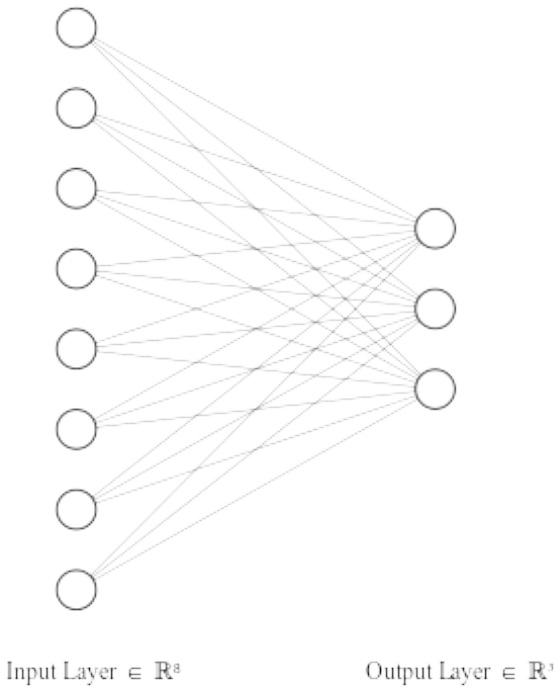
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

There is plenty of cheat sheets with activation functions and their properties...

Building a FNN: multioutput perceptron

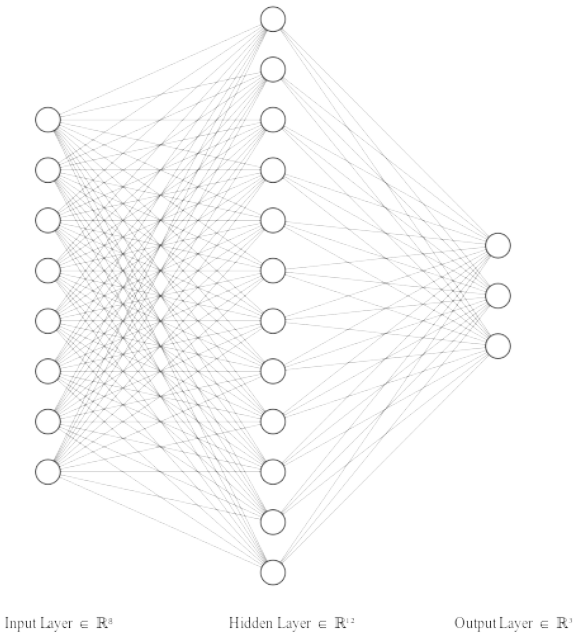
An FNN stands for Feed-Forward Neural Network (our focus during the course). Let's start with a single neuron (perceptron) with multiple outputs:



- $\hat{y}_1 = g\left(w_{0,1} + \sum_{j=1}^{m} x_j w_{j,1}\right)$
- $\hat{y}_2 = g\left(w_{0,2} + \sum_{j=1}^{m} x_j w_{j,2}\right)$
- $\hat{y}_3 = g\left(w_{0,3} + \sum_{j=1}^{m} x_j w_{j,3}\right)$

Building a FNN: single hidden layer NN

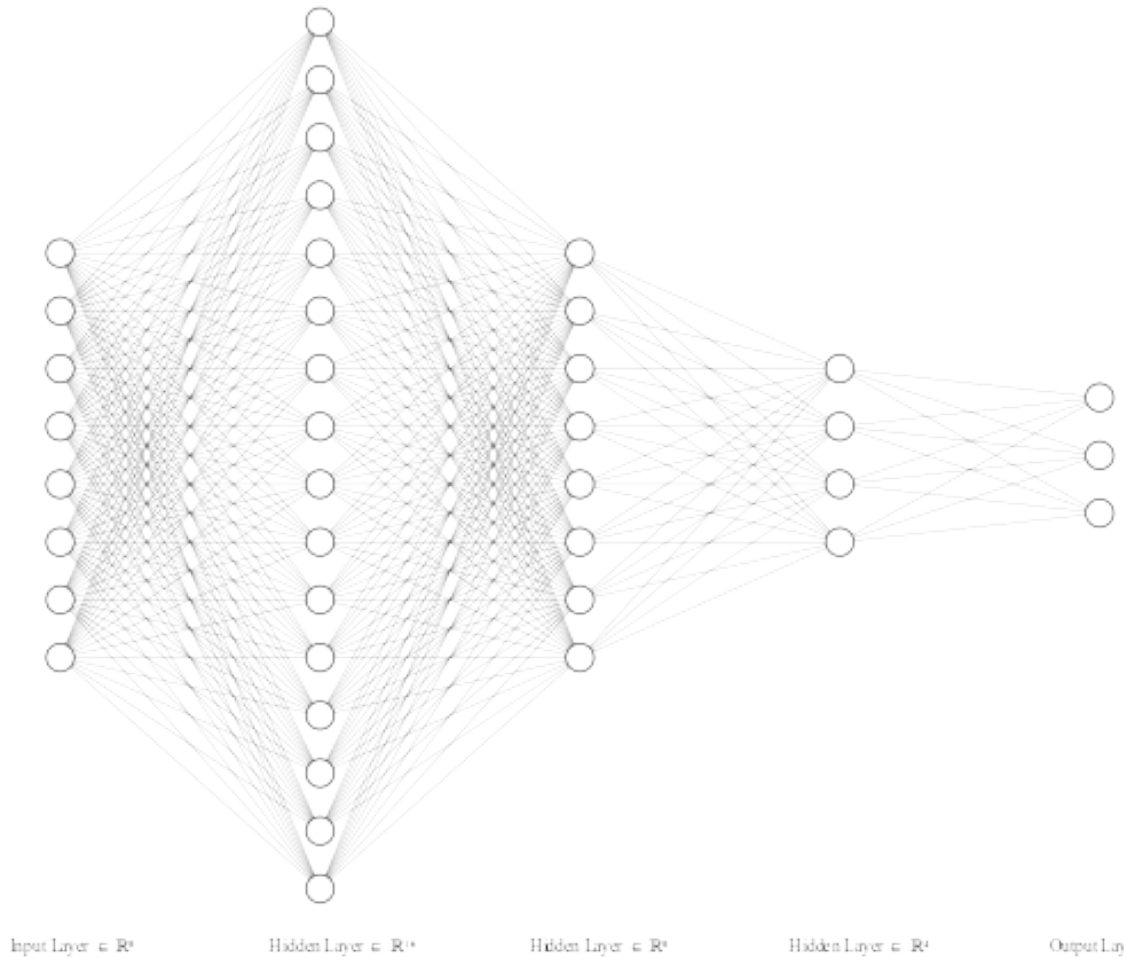
Let's take a look at with a single hidden layer NN:



- Hidden layer:
  $y_1 = g\left(w_{0,1}^{(1)} + \sum_{j=1}^{m} x_j w_{j,1}^{(1)}\right)$
- Output layer:
  $\hat{y}_1 = f\left(w_{0,1}^{(2)} + \sum_{j=1}^{m} x_j w_{j,1}^{(2)}\right)$
- Activation functions $g(z)$ and $f(z)$ can (and probably should) be different

Building a FNN: deep fully connected NN

Let's wrap up a deep fully connected NN:



Capabilities of ANNs

- **Boolean functions**:
  - Every Boolean function can be represented by a network with a single hidden layer.
  - Might require exponential (in number of inputs) hidden units.
- **Continuous functions**:
  - Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989].
- **Deep NN are in practice superior to other ML methods** in presence of large data sets.

Outline of the presentation

1 Course information

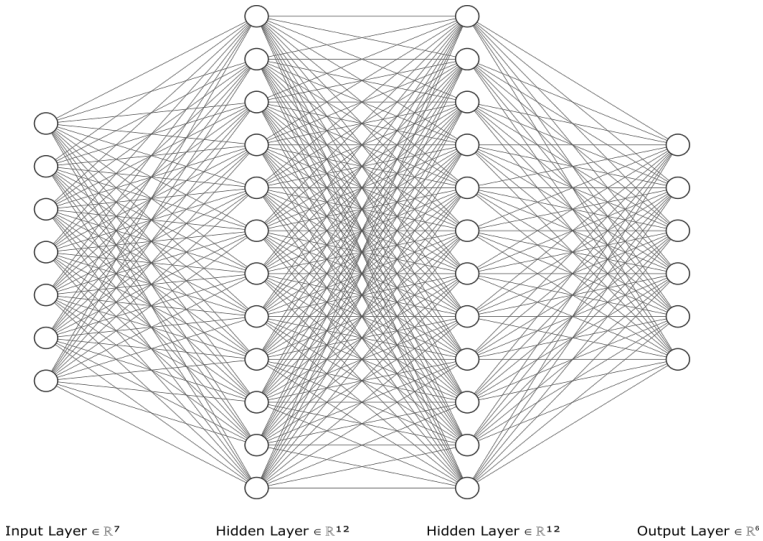
2 Brief (largely incomplete) overview of Machine Learning

3 Introduction to Artificial Neural Networks (ANNs)
- Building a NN
- Loss function
- (Stochastic) Gradient Descent
- Backpropagation

4 Practical session

Loss function: MSE

Training a model consists of minimizing a loss $\mathcal{L}_w$ which reflects the performance of the predictor $\mathcal{N}_w(x)$ on a training data set.



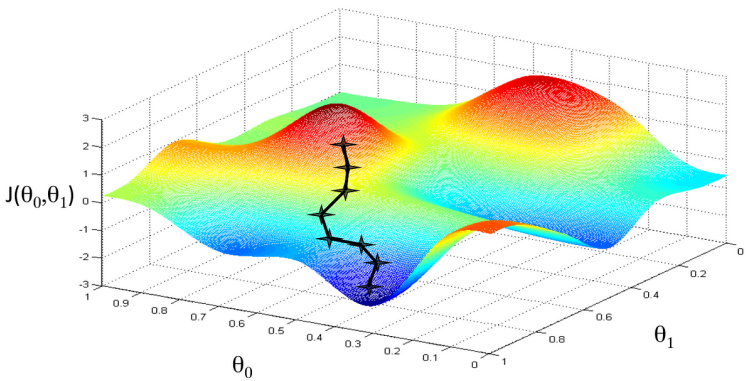
- Empirical loss:
  $\mathcal{L}_w = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\mathcal{N}_w\left(x^{(i)}\right), \hat{y}^{(i)}\right)$
- Mean Squared Error:
  $J(\rho) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathcal{N}_w\left(x^{(i)}\right) - \hat{y}^{(i)}\right)^2$

Outline of the presentation

1. Course information
2. Brief (largely incomplete) overview of Machine Learning
3. Introduction to Artificial Neural Networks (ANNs)
   - Building a NN
   - Loss function
   - (Stochastic) Gradient Descent
   - Backpropagation
4. Practical session

Gradient descent: general idea

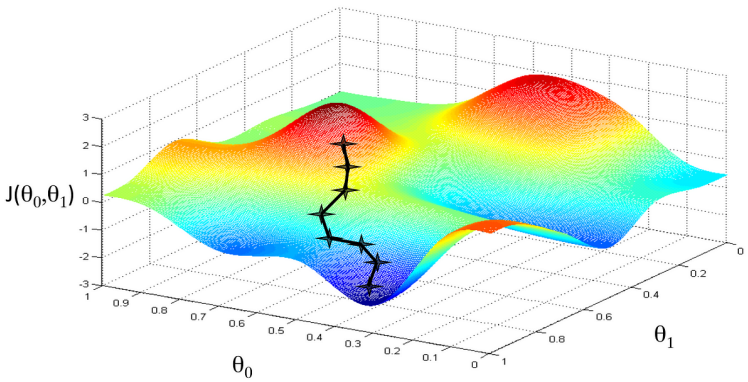
We want to find the network weights θ^* that achieve the lowest loss $\mathcal{L}_\theta$:
$\theta^* = \arg\min_\theta \mathcal{L}_\theta$



- Randomly pick initial weights (θ_0, θ_1)
- Compute gradient
- Take small steps in the opposite direction of gradient
- Repeat until convergence

Gradient descent: more specific

We want to find the network weights θ^* that achieve the lowest loss $\mathcal{L}_\theta$: $\theta^* = \arg\min_\theta \mathcal{L}_\theta$



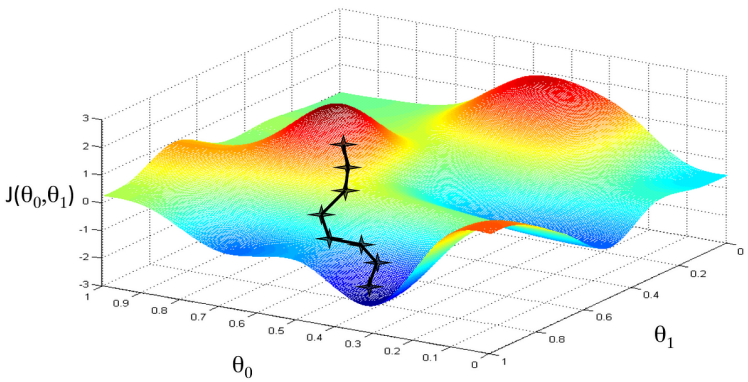
- Initialize weights randomly $\mathcal{N}_\theta$
- Repeat until convergence
  1. Compute gradient: $\frac{\partial \mathcal{L}_\theta}{\partial \theta}$
  2. Update weights: $\theta^{new} = \theta^{old} - \eta \frac{\partial \mathcal{L}_\theta}{\partial \theta}$, where η is a learning rate
  3. Return new weights

Problem with gradient descent

Taking gradients can be computationally expensive

Stochastic gradient descent

We want to find the network weights θ^* that achieve the lowest loss $\mathcal{L}_\theta$:
$\theta^* = \arg\min_\theta \mathcal{L}_\theta$



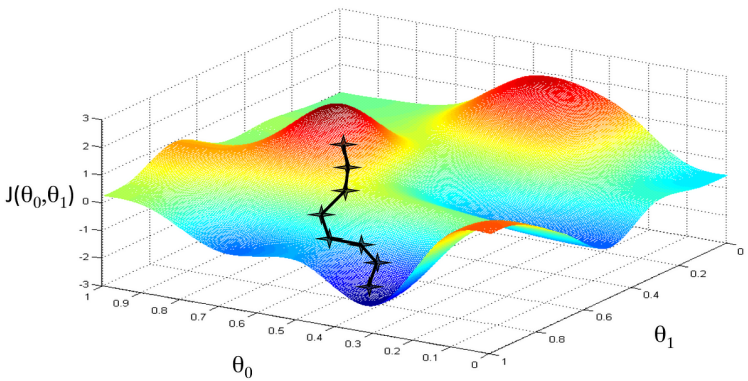
- Initialize weights randomly $\mathcal{N}_\theta$
- Repeat until convergence
  1. Pick a single data point *i*
  2. Compute gradient: $\frac{\partial \mathcal{L}_\theta^i}{\partial \theta}$
  3. Update weights: $\theta^{new} = \theta^{old} - \eta \frac{\partial \mathcal{L}_\theta^i}{\partial \theta}$, where η is a learning rate
  4. Return new weights

Problem with stochastic gradient descent based on a single data point

Taking gradients base don the single data point can be noisy

Stochastic gradient descent with batches

We want to find the network weights θ^* that achieve the lowest loss $\mathcal{L}_\theta$:
$\theta^* = \arg\min_\theta \mathcal{L}_\theta$



- Initialize weights randomly $\mathcal{N}_\theta$
- Repeat until convergence
  1. Pick a batch of data points B
  2. Compute gradient: $\frac{\partial \mathcal{L}_\theta}{\partial \theta} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial \mathcal{L}_\theta^k}{\partial \theta}$
  3. Update weights: $\theta^{new} = \theta^{old} - \eta \frac{\partial \mathcal{L}_\theta}{\partial \theta}$, where η is a learning rate
  4. Return new weights

Stochastic gradient descent with batches: justification

Justfication

From Deisenroth et al. (2021) Mathematics for Machine learning, Sec. 7.1.3.: *"The key insight about why taking a subset of data is a sensible is to realize that for gradient descent to converge, we only require that the gradient is an unbiased estimate of the true gradient."*

Training: some terminology

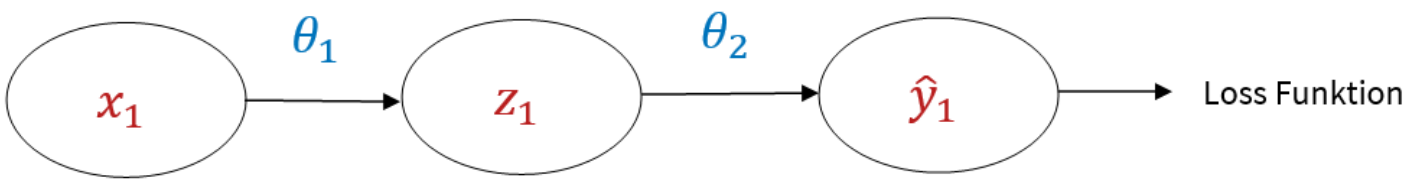
- The batch used for a SGD is called mini-batch
- An epoch refers to a complete pass through the entire dataset, where the algorithm has iterated over all the mini-batches once.

Outline of the presentation

1. Course information
2. Brief (largely incomplete) overview of Machine Learning
3. Introduction to Artificial Neural Networks (ANNs)
   - Building a NN
   - Loss function
   - (Stochastic) Gradient Descent
   - Backpropagation
4. Practical session

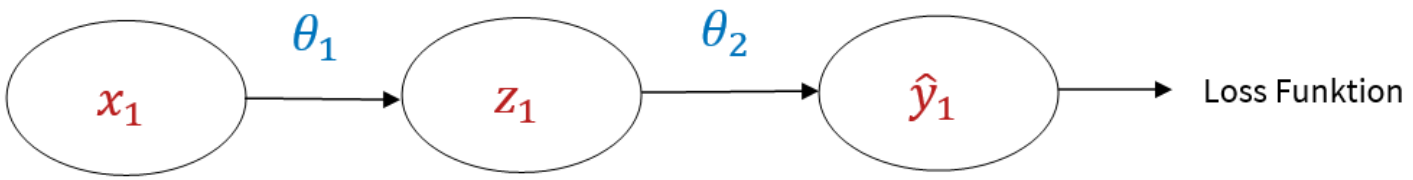
Computing gradients: error backpropagation

- How does a small change in one weight (e.g., θ_2) affect the final loss function $\mathcal{L}_\theta$?



Computing gradients: error backpropagation

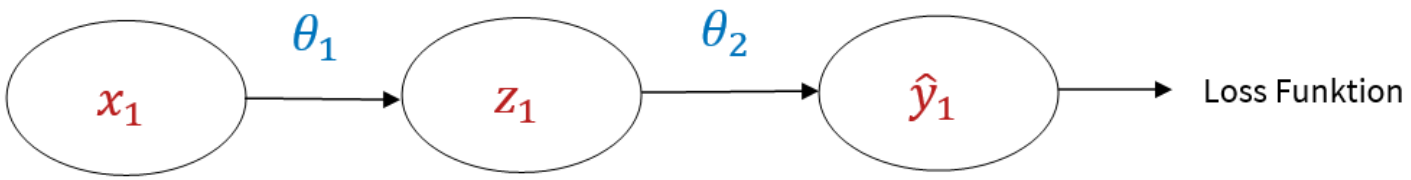
- How does a small change in one weight (e.g., θ_2) affect the final loss function $\mathcal{L}_\theta$?



- Chain rule: $\frac{\partial \mathcal{L}_\theta}{\partial \theta_2} = \frac{\partial \mathcal{L}_\theta}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_2}$

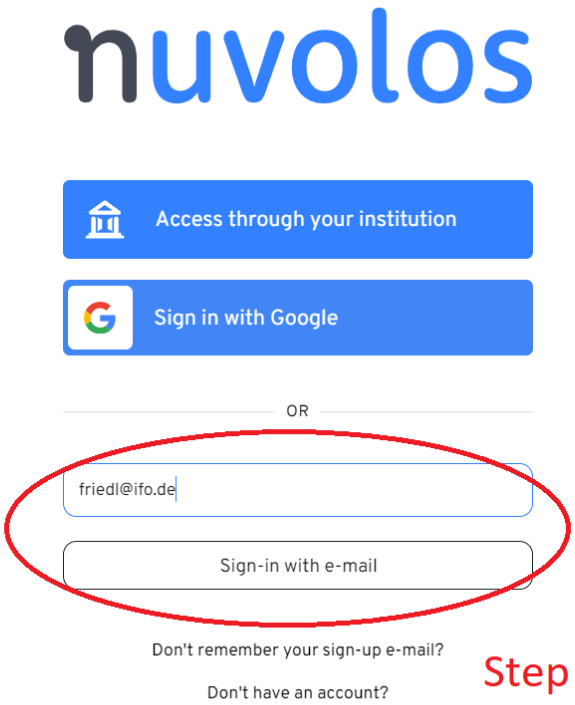
Computing gradients: error backpropagation

- How does a small change in one weight (e.g., θ_2) affect the final loss function $\mathcal{L}_\theta$?



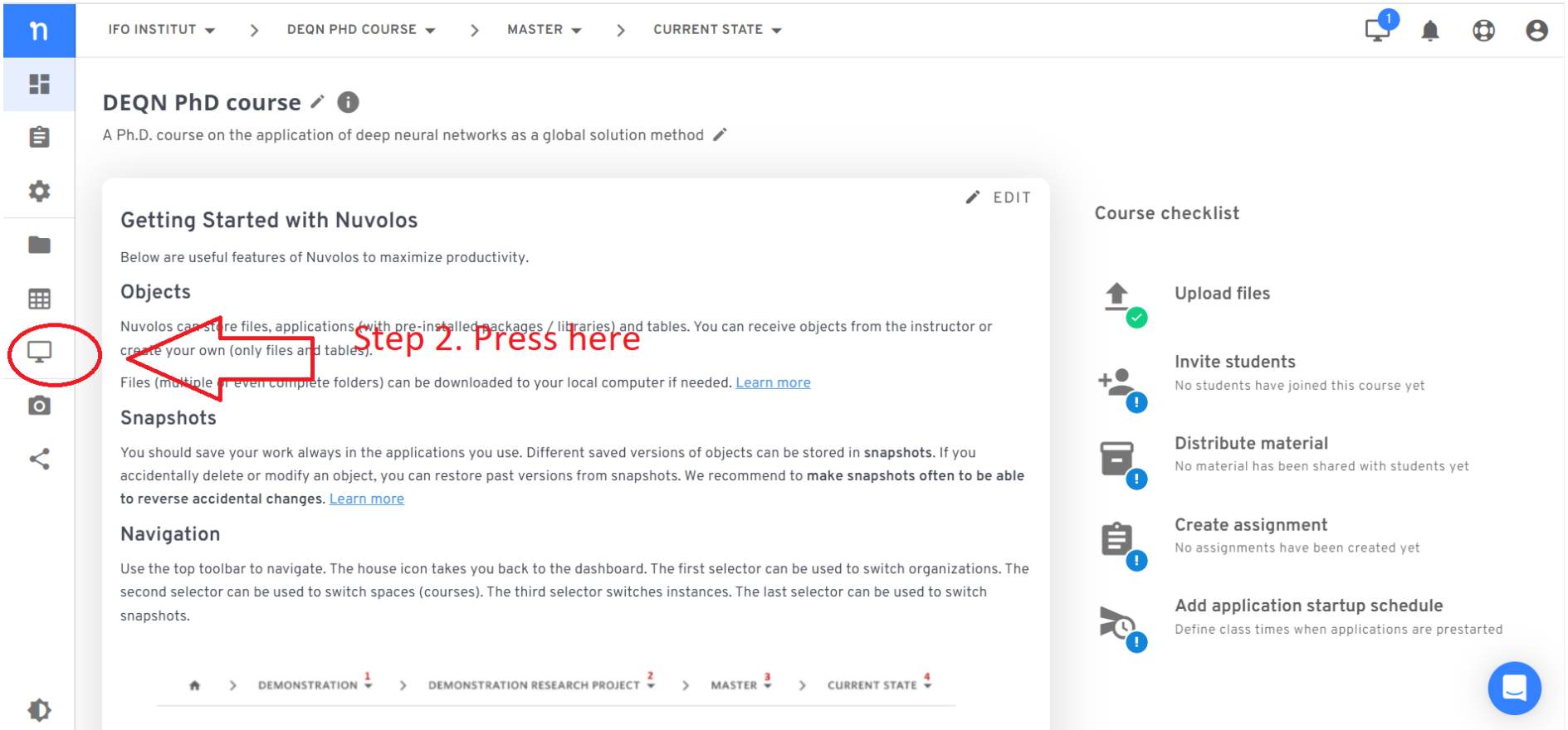
- Chain rule: $\frac{\partial \mathcal{L}_\theta}{\partial \theta_2} = \frac{\partial \mathcal{L}_\theta}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_2}$
- Repeat this for every weight in the network using gradients from later layers: $\frac{\partial \mathcal{L}_\theta}{\partial \theta_1} = \frac{\partial \mathcal{L}_\theta}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta_1} = \frac{\partial \mathcal{L}_\theta}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_1} \frac{\partial z_1}{\partial \theta_1}$

Login to Nuvolos: step 1

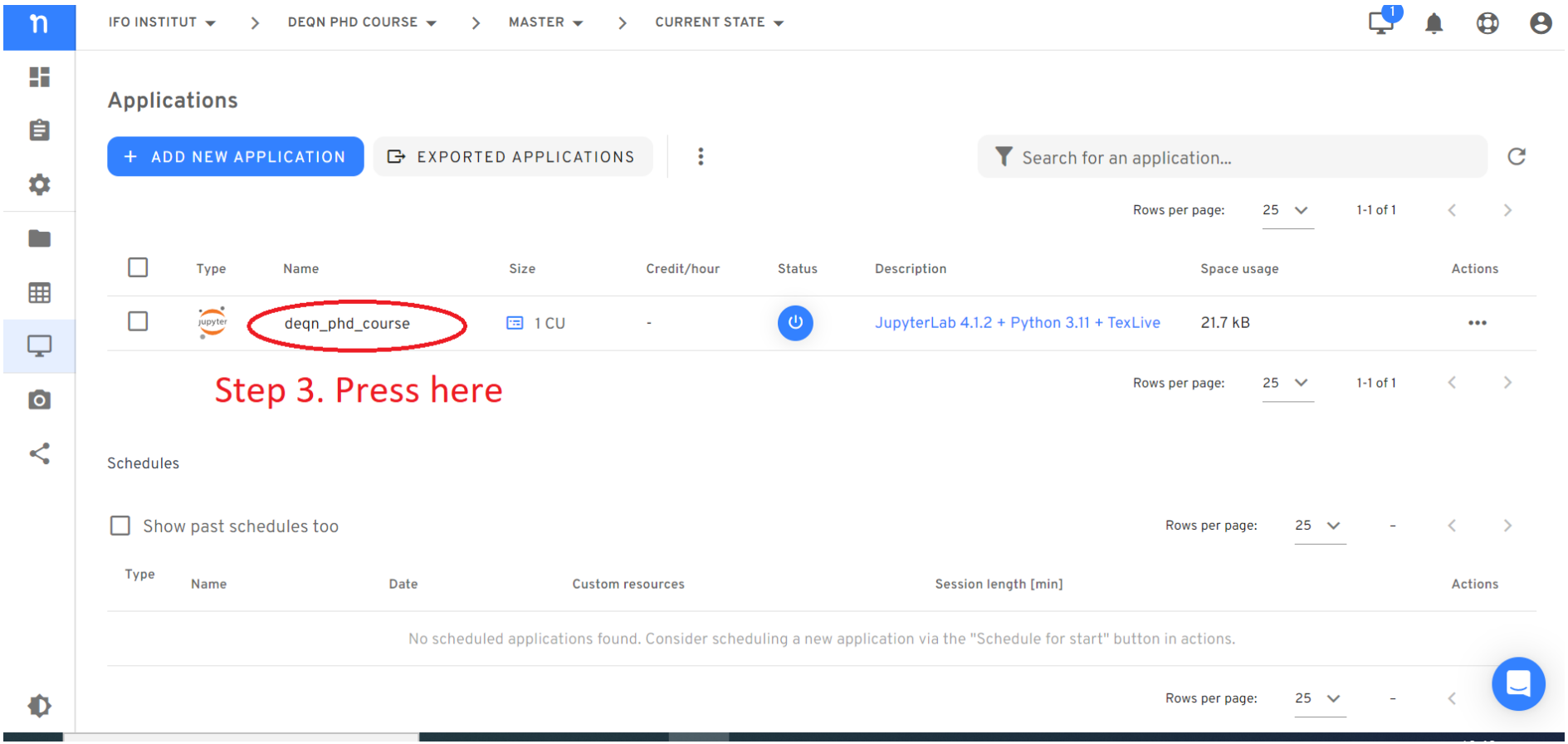


Step 1. Sign-up here

Login to Nuvolos: step 2



Login to Nuvolos: step 3



Let's look at SGD closer

- Let's look at this notebook on Nuvolos.cloud: 01_GradientDescent_and_StochasticGradientDescent.ipynb

Questions?