Beyond the Hype: An Introduction to Machine Learning and Neural Networks

Joël Marbet
CEMFI

CEMFI Undergraduate Internship 2023

June 29, 2023

About this Short Course

About this Short Course

- Brief introduction to machine learning with some applications
- Short course consists of three parts
  1. Introduction to Machine Learning
  2. Basics of Neural Networks
  3. Putting it into Action
- Our main goal is to be able to implement a neural network and get a basic understanding of recent advances in machine learning

How Are We Going To Do It?

- Implementation in Julia: Why not Matlab, Python, R, ...?
  - High-level + High-performance
  - Geared towards scientific computing
  - Solves two-language problem: All code in Julia, no need for fast low-level C/C++ code
- A very short introduction to Julia will be provided

Part 1: Introduction to Machine Learning

Introduction to Machine Learning I

- Hype around artificial intelligence (AI) reached new highs with OpenAI's ChatGPT
- Made possible by machine learning (ML) methods that have become popular in recent years...
- ...and have a wide variety of applications
  - Computer vision, speech recognition, data mining, and many more
  - Many potential applications in economics

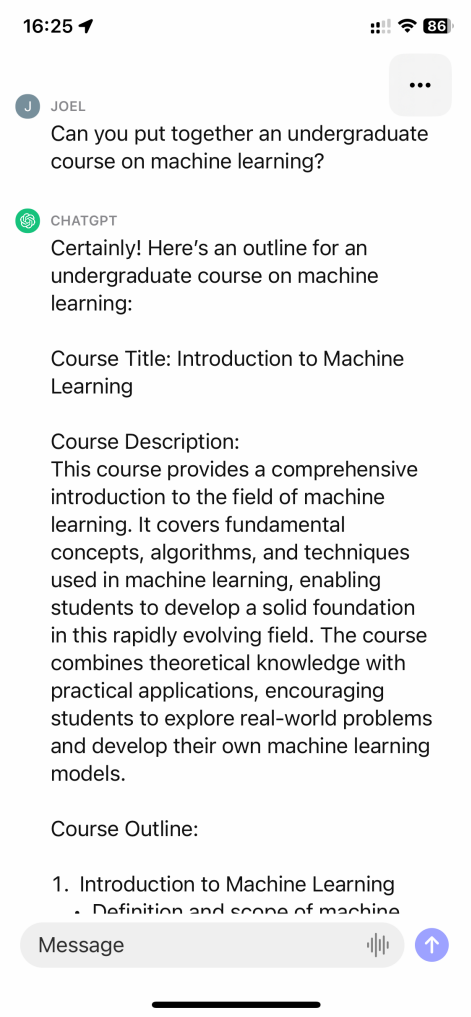


Figure 1: ChatGPT ML Course

Introduction to Machine Learning II

Figure 2: Go board (Source: Wikipedia)

- Well-known examples from recent years
  - DeepMind's AlphaGo is able to beat the best human Go players
  - OpenAI's ChatGPT responds to complex text prompts
  - Midjourney, DALL-E, and Stable Diffusion generate images from text
  - ...
- The field can be very technical, but barriers to entry not as high as they may seem with the mathematical background of an economist

What Is Machine Learning? I

Murphy (2012)

[...] a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty [...]

- Machine learning (ML) provides a range of methods for data analysis
- In that sense, it is similar to statistics or econometrics

What Is Machine Learning? II

- However, the focus of ML is more on prediction rather than causality
- Example from Varian (2014)
  - **Causal impact:** Change in sales associated with change in advertising expenditure everything else held constant
  - **Prediction:** Change in sales you would expect to observe when advertising expenditure changes

  $\rightarrow$ To decide whether to increase advertising expenditures, we are interested in the former rather than the latter
- Nevertheless, there is a large range of problems where ML methods can be very useful

Why Has Machine Learning Become Popular Only Recently?

- Early contributions to the field reach back at least to McCulloch and Pitts (1943) and Rosenblatt (1958)

  → Attempts to find mathematical representations of information processing in biological systems (Bishop, 2006)

- Field has grown substantially mainly in recent years due to
  - Advances in computational power of personal computers
  - Increased availability of large datasets → "big data"
  - Improvements in algorithms
- Need for large data sets still limits the applicability to certain fields

  → **Example:** In macroeconomic forecasting, we usually only have quarterly data for 40-50 years. Conventional time series methods (e.g., ARIMA) tend to perform better than ML methods (e.g., neural networks)

Artificial Intelligence vs. Machine Learning vs. Deep Learning

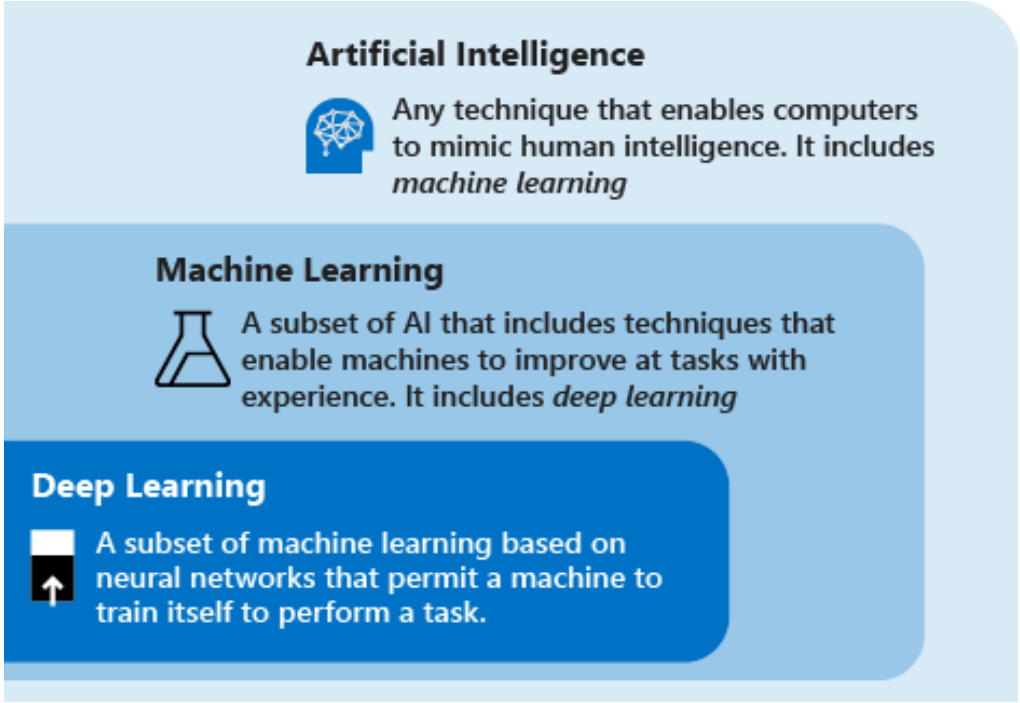


Figure 3: AI vs. ML vs. DL (Source: Microsoft)

- These terms are often used as synonyms in the media but, essentially, each term describes a more narrow subfield
- We will have a look at neural networks and deep learning later on

How Do Machines Learn?

Mitchell (1997)

A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

- Machine learning methods are commonly categorized into
  - **Supervised Learning:** Learn function $y = f(x)$ from data for x and y
  - **Unsupervised Learning:** "Make sense" of data x
  - **Reinforcement Learning:** Learn how to interact with environment
- We have a look at some examples of each category relating it to the definition of Mitchell (1997) above

Supervised Learning I

- Probably most common form of machine learning
- Task T: Learn mapping f from inputs $x \in \mathcal{X}$ into outputs $y \in \mathcal{Y}$
- Experience E is given by a training data set $\{(x_n, y_n\}_{n=1}^N$ of N input-output pairs (x_n, y_n)
- The type of function f might be incredibly complex, e.g.
  - From text input x to some coherent text response y ($\rightarrow$ ChatGPT)
  - From text input x to a generated image y ($\rightarrow$ Midjourney)
  - From bank loan application form x to a loan decision y

Supervised Learning II

- Note that sometimes
  - Inputs $\boldsymbol{x}$ are called features, predictors, or covariates
  - Outputs $\boldsymbol{y}$ are called labels, targets, or responses
- Based on the type of output, we can distinguish between
  - **Classification:** Output $\boldsymbol{y}$ is in a set of mutually exclusive labels (i.e., classes), i.e. $\mathcal{Y} = \{1, 2, 3, \ldots, C\}$
  - **Regression:** Output $\boldsymbol{y}$ is a real-valued quantity, i.e. $\boldsymbol{y} \in \mathbb{R}$

Supervised Learning: Classification

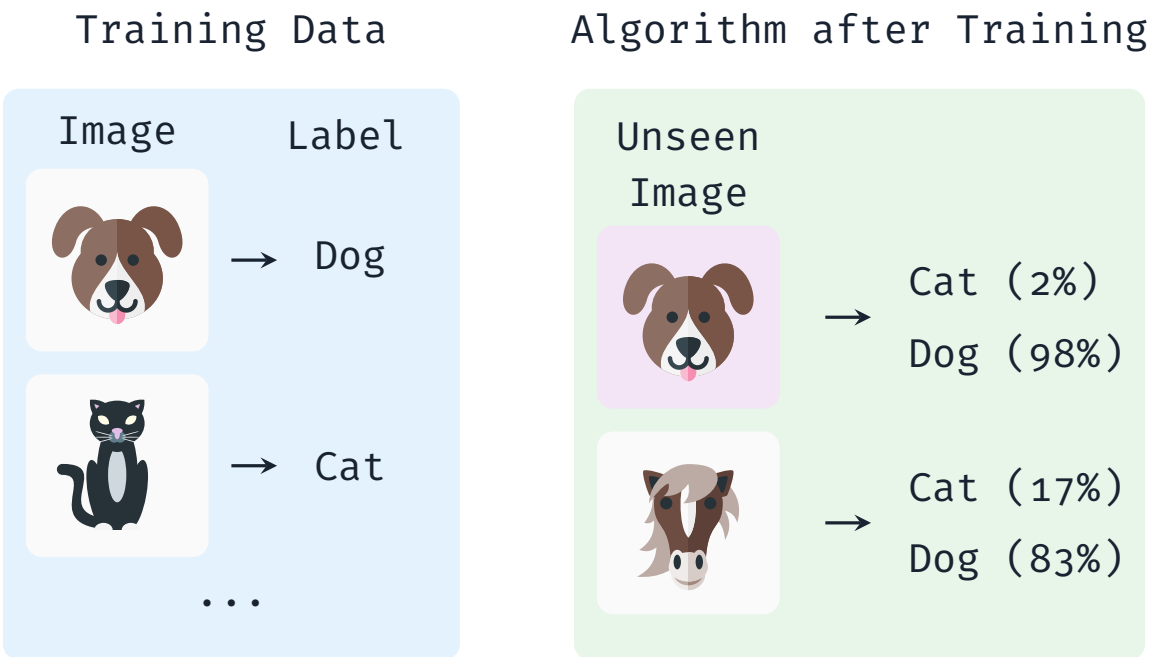


Figure 4: Classification Example

- **Example:** Using the training data set, a ML algorithm (e.g., a neural network) learns to recognize images of cats and dogs
- Accuracy evaluated on "new" images → test data set

Supervised Learning: Regression

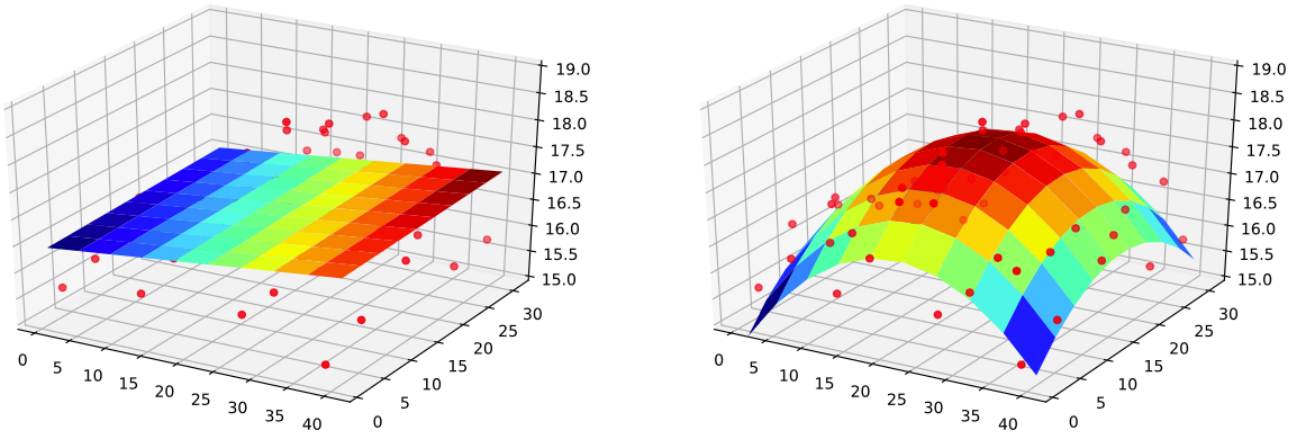


Figure 5: Linear and Polynomial Regression (Source: Murphy, 2022)

- Linear regression is a special form of supervised learning
- A more common form of regression in ML uses neural networks
- Neural networks can learn highly non-linear relationships

Unsupervised Learning I

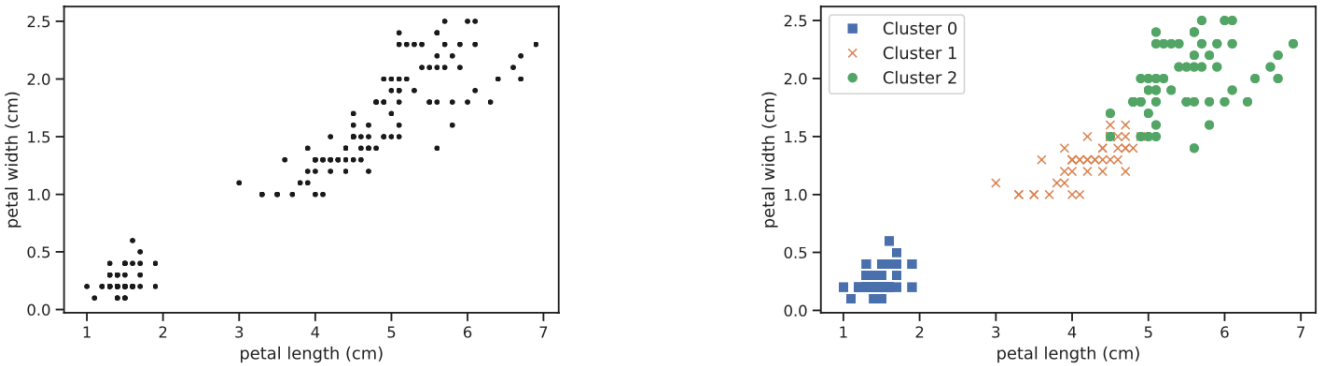


Figure 6: Clusters in data on Iris flowers (Source: Murphy, 2022)

- Task T: "make sense of" data
- Training data set is given by "inputs" only $\{x_n\}_{n=1}^N$
- "Making sense" could mean many things, for example,
  - Finding clusters in the data, i.e. finding data points that are "similar"
  - Finding latent factors that capture the "essence" of the data
  - ...

Unsupervised Learning II

- **Example:** Suppose you observe data on house prices and many variables describing each house
  - You might observe, e.g., property size, number of rooms, room sizes, proximity to the closest supermarket, and hundreds of variables more
  - A ML algorithm (e.g., principal component analysis or autoencoders) can find the unobserved factors that determine house prices
  - These factors sometimes (but not always) have an interpretation
  - For example, a factor driving house prices could be *amenities*
    - → This factor could summarize variables such as proximity to the closest supermarket, number of nearby restaurants, etc.
  - Ultimately, the hundreds of explanatory variables in the data set might be represented by a small number of factors

Reinforcement Learning I

- Task T: Learn policy $a = \pi(x)$ which specifies action to take in response to input x, e.g., an agent has to learn how to interact with its environment
- **Example:** Learning how to play chess
  - Input x would be the current position (i.e., pos. of pieces on the board)
  - Action a would be the next move to make given the position
  - One also needs to define a reward (e.g., winning the game at the end)

  $\rightarrow$ Goal is then to find $a = \pi(x)$ to maximize the reward

Reinforcement Learning II

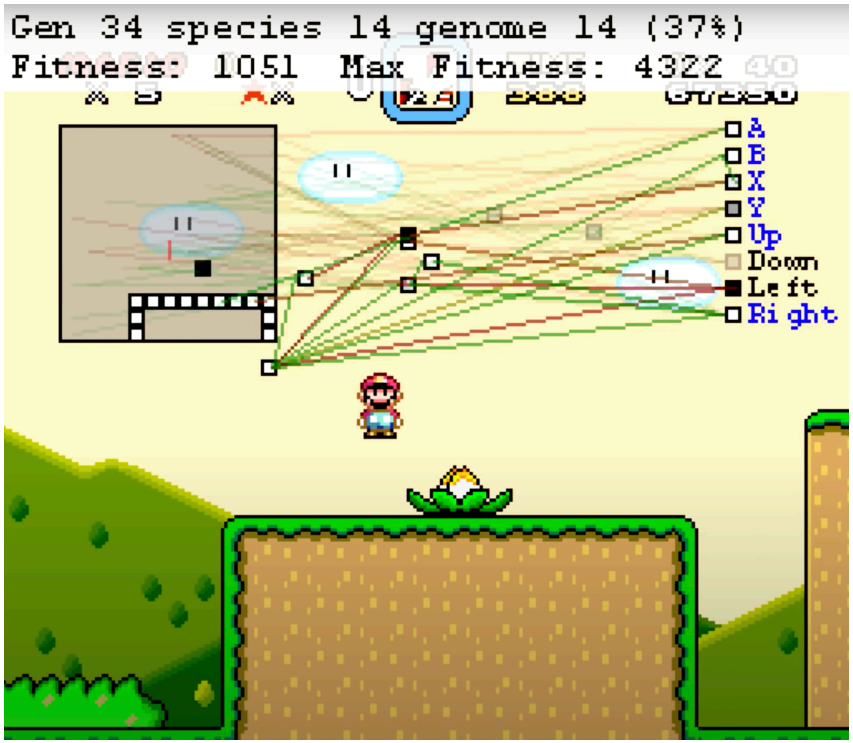


Figure 7: MarI/O (Source: Youtube)

- **Example 2:** Learning how to play Super Mario World: `https://www.youtube.com/watch?v=qv6UVOQ0F44`

Types of Learning in Practice

- Machine learning models might combine different types of learning

  → **Example:** ChatGPT is trained using a combination of supervised and reinforcement learning

- Some machine learning methods, such as neural networks, might be used as part of different types of learning

Commonly used Datasets I

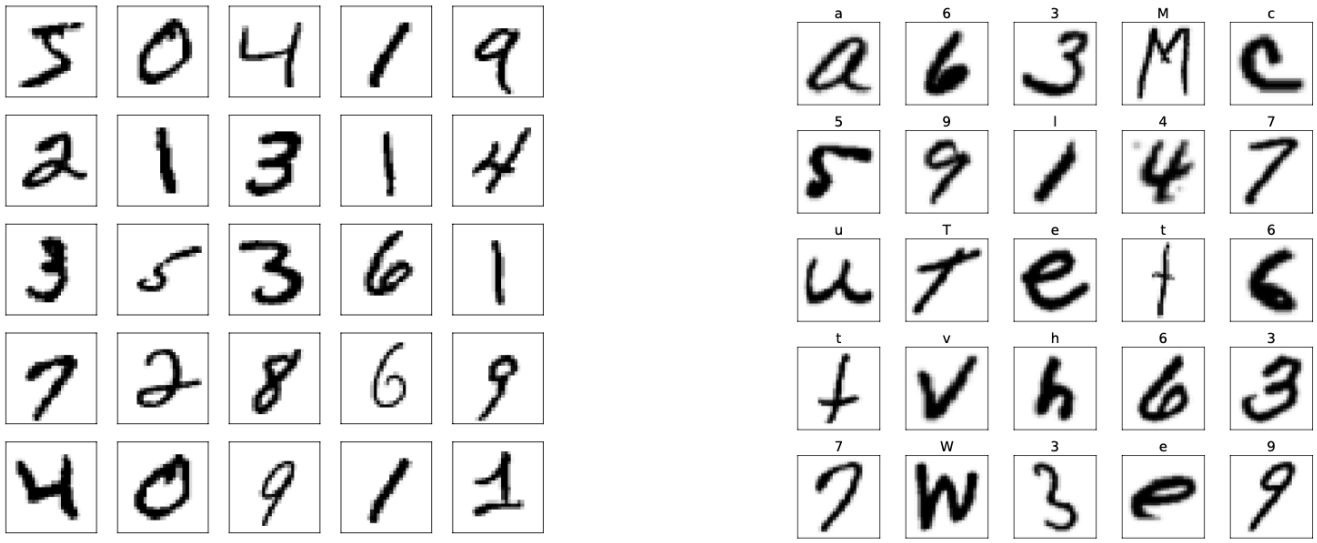


Figure 8: MNIST and EMNIST Datasets (Source: Murphy (2022))

- There are many publicly available datasets to get you started
- See section 1.5 in Murphy (2022): “Probabilistic Machine Learning: An Introduction”

Commonly used Datasets II

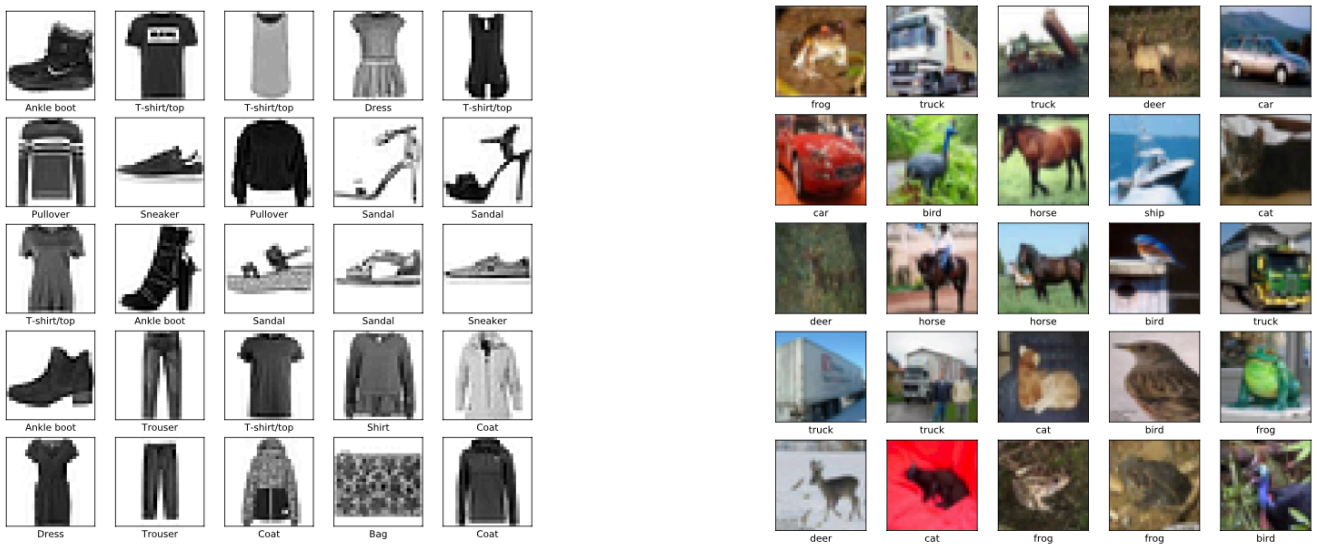


Figure 9: Fashion-MNIST and CIFAR-10 Datasets (Source: Murphy (2022))

Recent Applications in Economics

- Machine learning methods are increasingly used in economics
- Recent examples include
  - Non-linear solution of heterogeneous agent models
    Fernández-Villaverde, Hurtado, and Nuño (2023); *Fernández-Villaverde, Marbet, Nuño, and Rachedi (2023)*; *Kase, Melosi, and Rottner (2022)*; *Maliar, Maliar and Winant (2021)*
  - Deep-learning model to detect emotions in voices during press conferences after FOMC meetings
    Gorodnichenko, Pham, and Talavera (2021)
  - Identifying Monetary Policy Shocks using Natural Language Processing
    Aruoba, and Drechsel (2022)
  - Estimation of structural models with the help of neural networks
    Kaji, Pouliot, and Manresa (2020)

but there are many more...

Part 2: Neural Networks

Why can Neural Networks be useful to Economists? I

- NN are at the core of most cutting-edge machine learning models
- As we will see later on, under certain (relatively weak) conditions
  - Neural networks are universal approximators
    $\rightarrow$ Can approximate any (Borel measurable) function
  - Neural networks break the curse of dimensionality
    $\rightarrow$ Can handle very high dimensional functions
- NN are also easy to code, stable, and scalable for multiprocessing
- Interesting for a wide range of fields in economics, e.g., quantitative macroeconomics or econometrics

Why can Neural Networks be useful to Economists? II

- NN are not a magic bullet, and there are some downsides
  - Large data requirements may make it infeasible to properly train the NN in some applications (e.g. GDP forecasting)
  - Slow training may make other approaches more sensible in order to approximate simple functions
- In this part of the course, we will try to understand
  - How neural networks are built, and
  - How they can be trained to approximate functions

Origins of the Term “Neural Network”

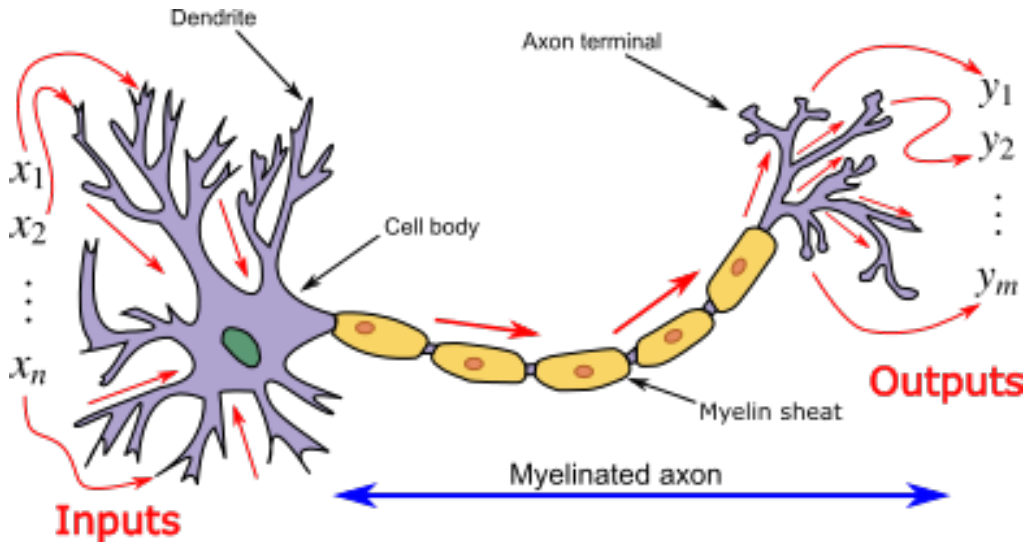
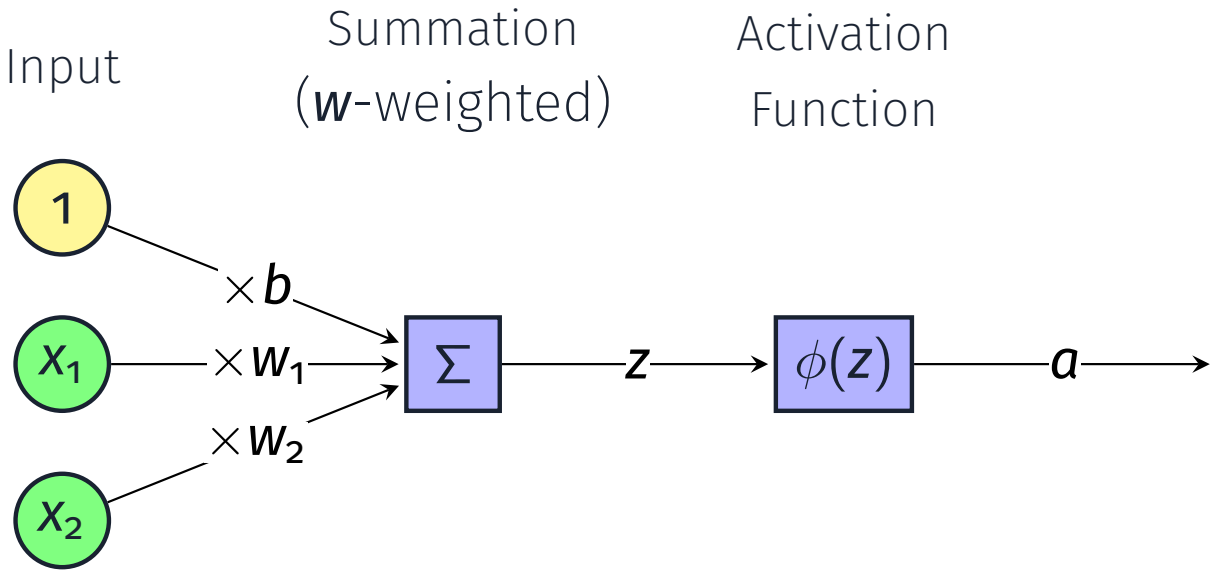


Figure 10: A biological neuron (Source: Wikipedia)

- Origins in attempts to find mathematical representations of information processing in biological systems (Bishop, 2006)
- Biological interpretation not very important for research anymore
- But the interpretation can be useful when starting to learn about NN

An Artificial Neuron I



- Artifical neurons are the basic building blocks of neural networks
- N inputs denoted $x = (x_1, x_2, \ldots, x_N)'$ and a single output denoted a
- Inputs are linearly combined into z using weights w_i and bias b

$$z = b + \sum_{i=1}^{N} w_i x_i = \sum_{i=0}^{N} w_i x_i$$

where we defined an additional input $x_0 = 1$ and $w_0 = b$

An Artificial Neuron II

- Linear combination z is transformed using an activation function

$$a = \phi(z) = \phi\left(\sum_{i=0}^{N} w_i x_i\right)$$

- Activation function introduces non-linearity into the NN and allows the NN to learn highly non-linear functions
- Particular choice of activation function depends on the application

A Special Case: Perceptron

- Perceptrons were developed in the 1950s and have only one neuron
- They use a step function as an activation function

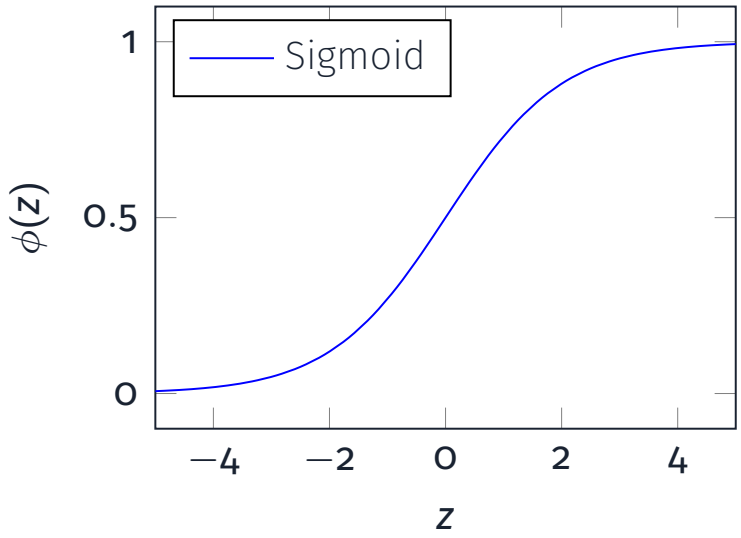
$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise}, \end{cases}$$

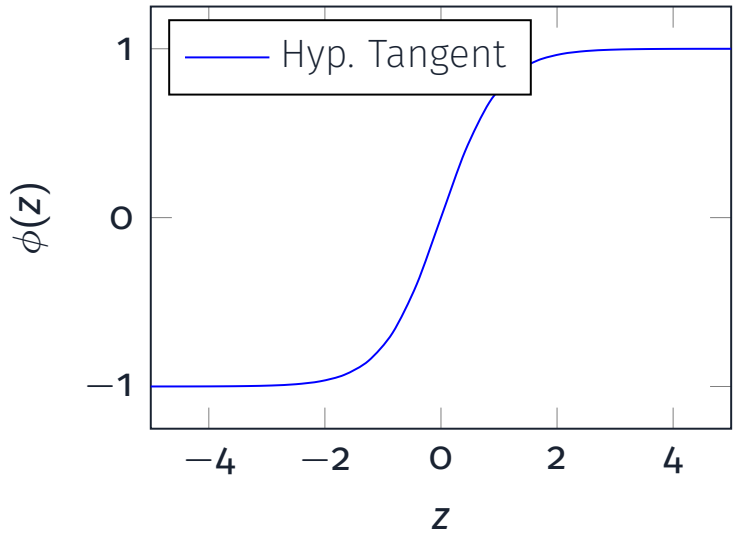
- Perceptrons can be used for basic classification
- Note that the step function is usually not used in neural networks
  - Derivative is not defined at $z = 0$ and zero everywhere else
  - Thus, it cannot be used for the back-propagation algorithm, which is used for determining the network weights

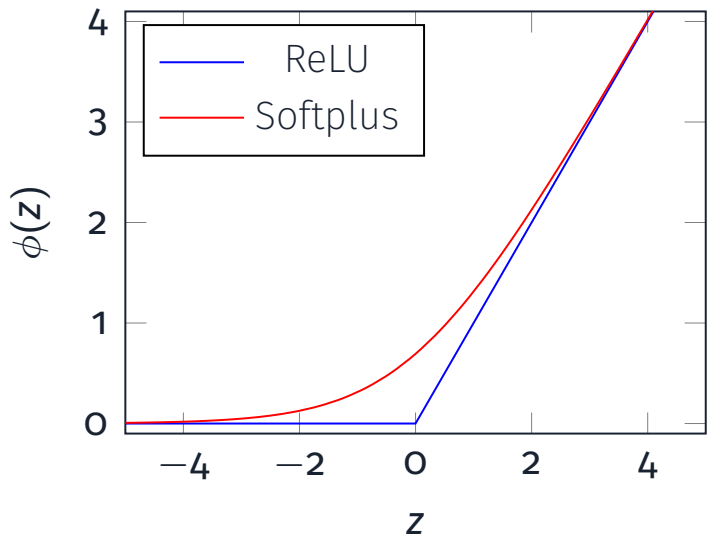
Activation Functions

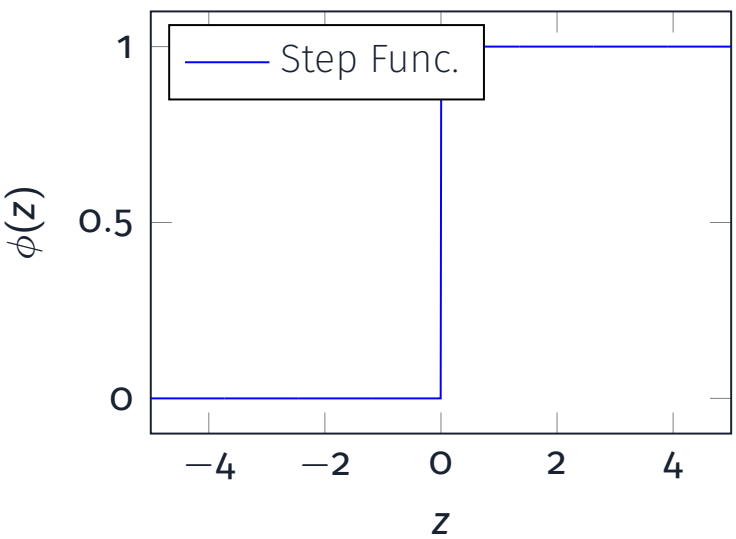
- Common activation functions include
  - Sigmoid: $\phi(z) = \frac{1}{1+e^{-z}}$
  - Hyperbolic tangent: $\phi(z) = tanh(z)$
  - Rectified linear unit (ReLU): $\phi(z) = \max(0, z)$
  - Softplus: $\phi(z) = \log(1 + e^z)$
- ReLU has become a popular in deep neural networks in recent years
- For approximations of smooth functions, softplus can be a good alternative to ReLU

Examples of Activation Functions









Building a Neural Network from Artificial Neurons

- A single-layer neural network is a linear combination of M artificial neurons a_j

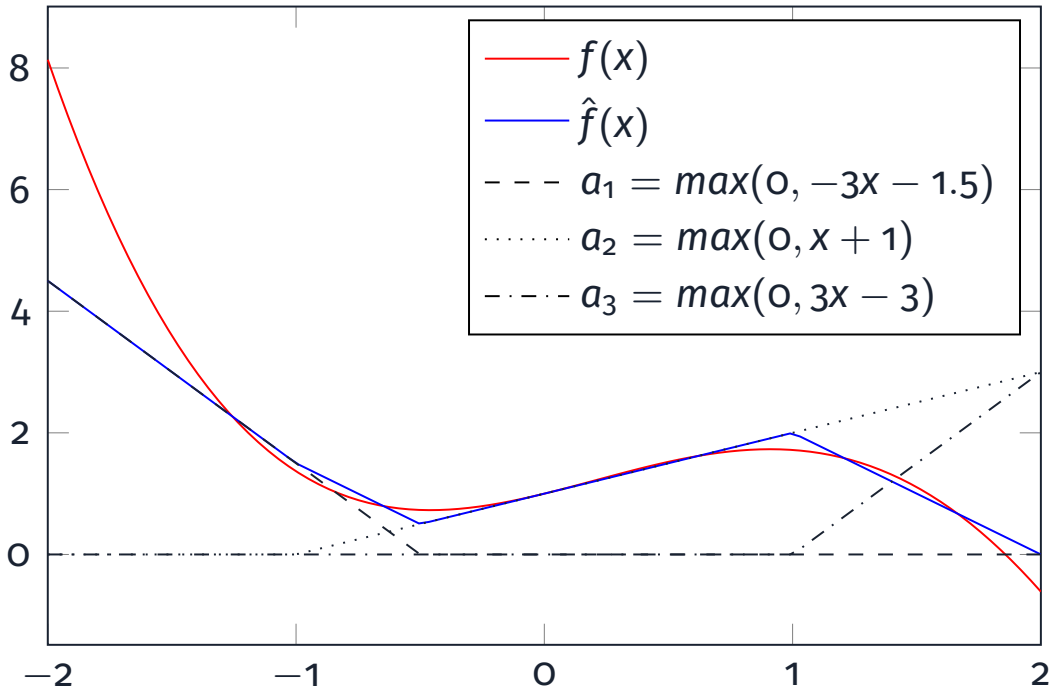
$$a_j = \phi(z_j) = \phi\left(b_j^1 + \sum_{i=1}^{N} w_{ji}^1 x_i\right)$$

with the output defined as

$$g(x; w) = b^2 + \sum_{j=1}^{M} w_j^2 a_j$$

- This neural network has N inputs, a single hidden layer with M nodes/neurons, and a single output
- M is also called the width of the neural network

Building a Neural Network from Artificial Neurons: Example



- Suppose we want to approximate $f(x) = exp(x) - x^3$ with 3 neurons
- Our (admittely bad) approximation might be $\hat{f}(x) = a_1 + a_2 - a_3$

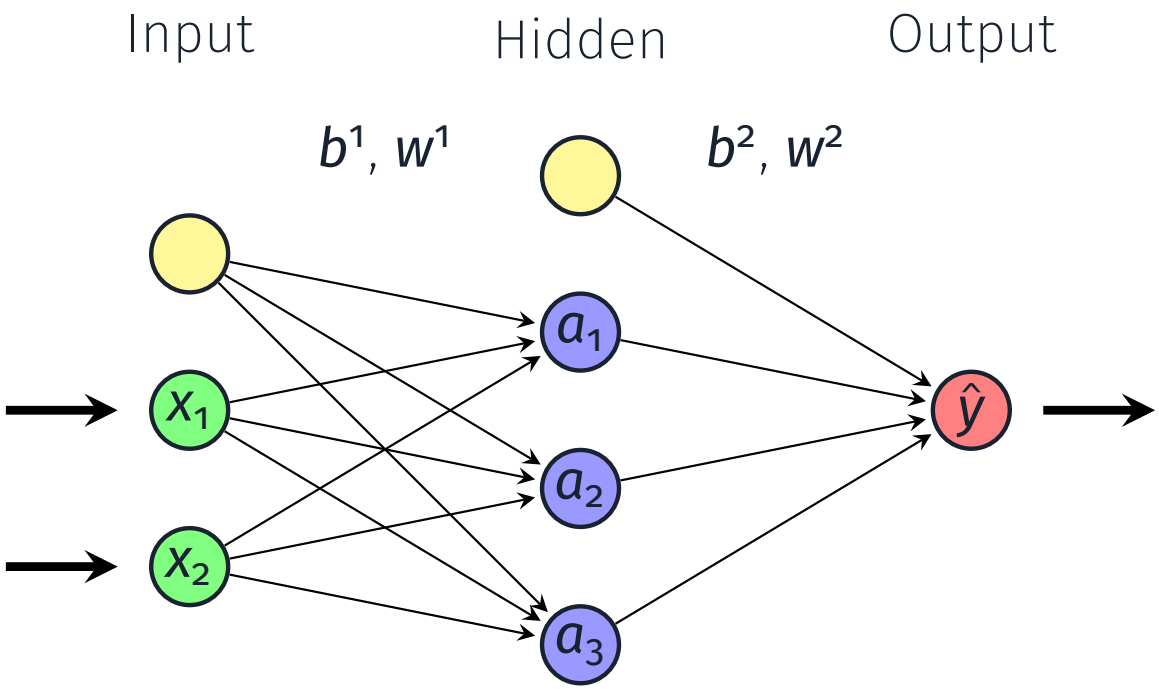
Relation to Linear Regression

- Suppose we use a linear activation function, e.g. $\phi(x) = x$
- Then, the neural network collapses to a linear regression

$$y \cong g(x; w) = \tilde{w}_0 + \sum_{i=1}^{N} \tilde{w}_i x_i$$

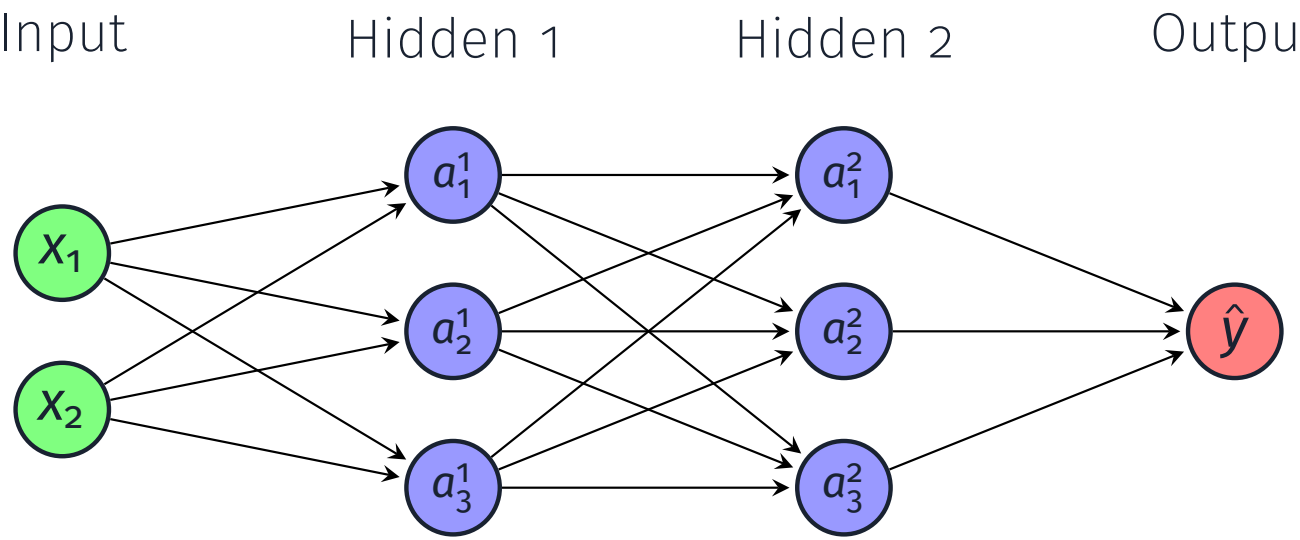
  with appropriately defined regression coefficients $\tilde{w}$
- NN is a linear combination of M generalized linear models of x
- Non-linear activation function is crucial to approximate non-linear functions

A Graphical Representation of Neural Networks



- Common to represent NN as directed graphs (here $N = 2$ and $M = 3$)
- We will only consider neural networks
  - that are feedforward (i.e. their graphs are acyclical),
  - with dense layers (i.e. each layer is fully connected to the prev.), and
  - without connections that skip layers

Deep Neural Networks



- Deep neural networks have more than one hidden layer
- Number of hidden layers is also called the depth of the NN
- Deep neural networks can learn more complicated things
- For simple function approximation, a single hidden layer is sufficient

Deep Neural Networks II

- The first hidden layer consists of M_1 artificial neurons with inputs $x_1, x_2, \ldots, x_N$

$$a_j^1 = \phi\left(b_j^1 + \sum_{i=1}^{N} w_{ji}^1 x_i\right)$$

- The second hidden layer consists of M_2 artificial neurons with inputs $a_1^1, a_2^1, \ldots, a_{M_1}^1$

$$a_k^2 = \phi\left(b_k^2 + \sum_{j=1}^{M_1} w_{kj}^2 a_j^1\right)$$

- After Q hidden layers the output defined as

$$y \cong g(x; w) = b^{Q+1} + \sum_{j=1}^{M_Q} w_j^{Q+1} a_j^Q$$

- Activation functions do not need to be the same everywhere

Why Is This Useful?

- Recall that we want to approximate an unknown function

$$y = f(x)$$

  where $y = (y_1, y_2, \ldots, y_K)'$ and $x = (x_1, x_2, \ldots, x_N)'$ are vectors

- $f(x)$ could stand for many different functions in economics (e.g. a value function, a policy function, a conditional expectation, a classifier, ...)

- It turns out that neural networks are universal approximators and break the curse of dimensionality

Universal approximation theorem (Hornik, Stinchcombe, and White, 1989)

A neural network with at least one hidden layer can approximate any Borel measurable function mapping finite-dimensional spaces to any desired degree of accuracy.

Relation to Other Approximation Methods I

- Recall the expression for our basic neural network

$$y \cong g(x; w) = b^2 + \sum_{j=1}^{M} w_j^2 \phi \left(b_j^1 + \sum_{i=1}^{N} w_{ji}^1 x_i \right)$$

  where for notational simplicity we assumed that y is scalar

- This looks similar to a projection

$$y \cong \tilde{g}(x; w) = w_0 + \sum_{j=1}^{M} w_j \phi_j (x)$$

  where ϕ_j is, for example, a Chebyshev polynomial

Relation to Other Approximation Methods II

Breaking the curse of dimensionality (Barron, 1993)

A one-layer NN achieves integrated square errors of order $O(1/M)$, where M is the number of nodes. In comparison, for series approximations, the integrated square error is of order $O(1/(M^{2/N}))$ where N is the dimensions of the function to be approximated.

- Crucial difference between NN and traditional series approx.
  - In series approximation coefficients should increase exponentially with dimensions to preserve a given precision (this is not the case for NN)
  - NN require much more samples to train, given a number of parameters, than traditional series approximation
- Large data requirements may make it infeasible to properly train the NN in some applications (e.g. GDP forecasting)
- In some cases (e.g. macro-modelling), we can generate data

Training a Neural Network: Determining Weights and Biases

- We have not yet discussed how to determine the weights and biases
- The weights and biases w are selected to minimize a loss function

$$E(w; X, Y) = \frac{1}{N} \sum_{n=1}^{N} E_n(w; x_n, y_n)$$

  where N refers to the number of input-output pairs that we use for training and $E_n(w; x_n, y_n)$ refers to the loss of an individual pair n

- For notational simplicity, I will write $E(w)$ and $E_n(w)$ in the following or in some cases even omit argument w

Choice of Loss Function

- The choice of loss function depends on the problem at hand
- In regressions, one often uses a mean squared error (MSE) loss

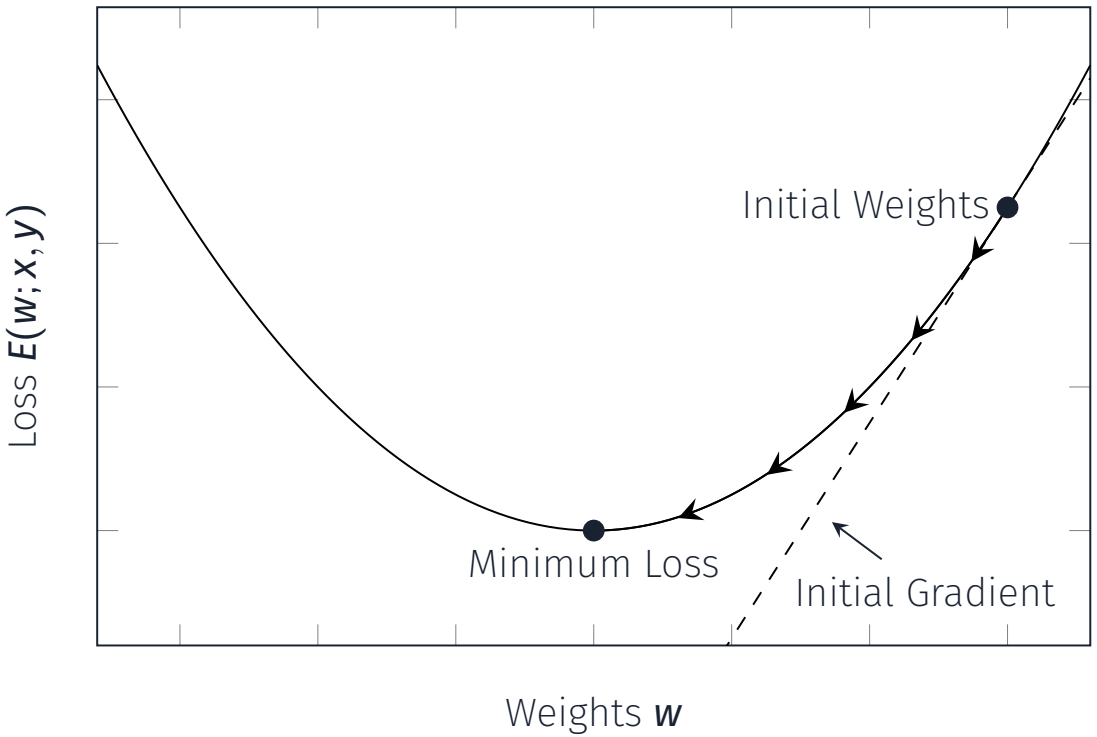
$$E_n(w; x_n, y_n) = \frac{1}{2} \|g(x_n; w) - y_n\|^2$$

- In classification problems, one often uses a cross entropy loss

$$E_n(w; x_n, y_n) = \sum_{k=1}^{K} y_{nk} \log(g_k(x_n; w))$$

where k refers to kth class (or kth element) in the output vector

Gradient Descent



- Loss function is minimized using a gradient descent algorithm
- **Basic Idea:** Compute how loss changes with weights w and step into direction which reduces the loss

Algorithm for Batch Gradient Descent

1. Initialize weights (e.g. draw from Gaussian distribution)

$$w^{(0)} \sim N(0, I)$$

2. Compute gradient of loss function with respect to weights

$$\nabla E(w^{(i)}) = \frac{1}{N} \sum_{n=1}^{N} \nabla E_n \left(w^{(i)} \right)$$

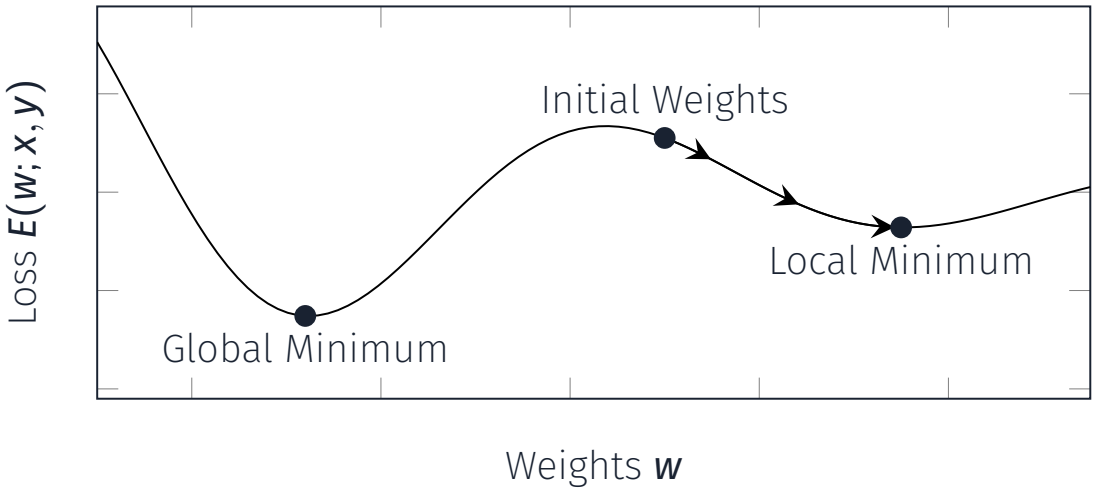
3. Update weights (make small step in direction of negative gradient)

$$w^{(i+1)} = w^{(i)} - \eta \nabla E \left(w^{(i)} \right)$$

   where $\eta > 0$ is the learning rate

4. Repeat step 2 and 3 until a terminal condition (e.g. fixed number of iterations) is reached

Gradient Descent: Practical Issues I

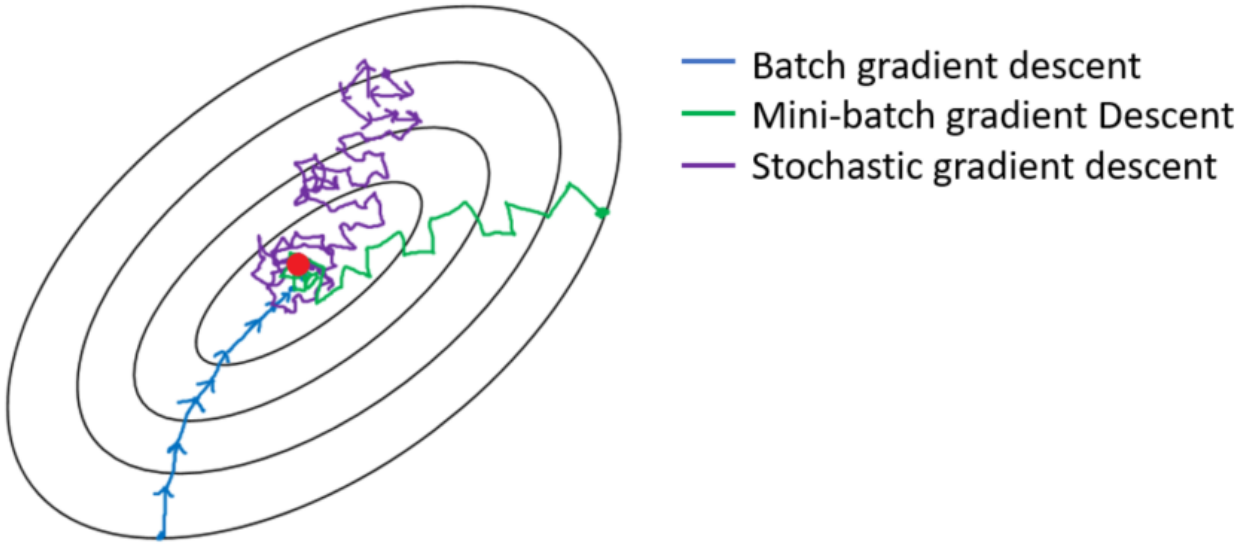


- Using batch gradient descent one might get stuck in local minima
- The following have proven to be less susceptible to this
  - Stochastic gradient descent: Use only a single observation to compute the gradient and update the weights for each observation

$$w^{(i+1)} = w^{(i)} - \eta \nabla E_n \left(w^{(i)} \right)$$

  - Minibatch gradient descent: Use a small batch of observations (e.g. 32) to compute the gradient and update the weights for each minibatch

Gradient Descent: Practical Issues II



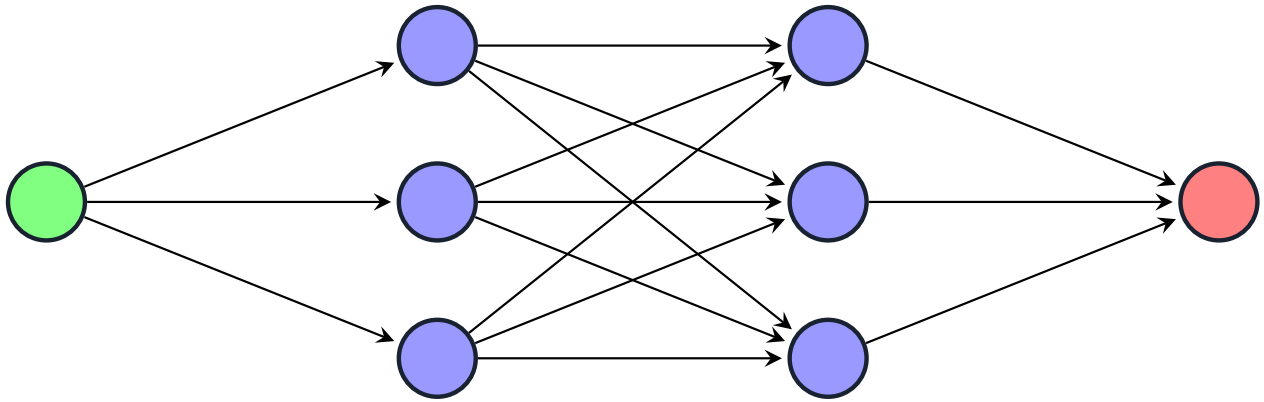
Gradient Descent Comparison (Source: Analytics Vidhya, medium.com)

- Gradient descent becomes “noisier” if not all observations are used

  $\rightarrow$ Less likely to get stuck in shallow local minimum

- Minibatch gradient descent is probably the most commonly used

Backpropagation Algorithm



- Computing the gradient seems to be a daunting task

  $\rightarrow$ A weight in the first layer in a deep neural network affects the loss function through potentially thousands of "paths"

- The backpropagation algorithm (Rumelhart et al., 1986) provides an efficient way to evaluate the gradient

  - Major breakthrough for training deep neural networks
  - **Basic idea:** Go backward through network to evaluate the gradient

Practical Considerations

- From a practical perspective, there are many more things to consider
- Often times it's beneficial to do some (or all) of the following
  - Input/output normalization (e.g. to have unit variance and mean zero) can improve performance of the NN
  - Check for overfitting by splitting the dataset into a training dataset and a validation dataset
  - Use regularization to avoid overfitting (e.g. add term to loss function that penalizes large weights)
  - Adjust the learning rate η during training

Putting It Into Action: Implementing Your Very Own Neural Network

Machine Learning Libraries

- Python is the dominant language for machine learning
  - TensorFlow: Machine learning library developed by Google
  - PyTorch: Another popular machine learning library
  - Keras: Programming interface for several deep learning libraries (incl. TensorFlow)
- Julia has recently become popular in scientific computing due to its speed and ease of use, but it is not that commonly used for ML yet
  - Flux.jl: Machine learning library fully written in Julia

Putting it Into Action

- We will be using Julia today because
  - Flux.jl is very easy to use
  - Knowing Julia will be very valuable if you ever need to solve (macro-)economic models
  - Julia is not taught as a separate course in the internship this year
- Let's move over to the Pluto notebook now...

Conclusions

Conclusions

- In this short course, we had a look at
  1. The field of machine learning in general,
  2. The basics of neural networks, and
  3. How to implement neural networks
- You now have powerful new tools at your disposal. Use them wisely!
- If you have any questions about the course, feel free to come by my office at any time or send an email to joelmarbet@cemfi.edu.es

Resources for Further Learning I

- Useful references for neural networks and machine learning
  - Goodfellow et al. (2016), “Deep Learning” (`https://www.deeplearningbook.org`)
  - Bishop (2006), “Pattern Recognition And Machine Learning”
  - Nielsen (2019), “Neural Networks and Deep Learning” (`http://neuralnetworksanddeeplearning.com/`)
  - Murphy (2012, 2022, 2023), Book Series on Probabilistic ML (`https://probml.github.io/pml-book/`)
  - Sutton and Barto (2018), “Reinforcement Learning: An Introduction” (`http://incompleteideas.net/book/the-book-2nd.html`)
- Note that several of these books are officially available for free

Resources for Further Learning II

- Some useful references for Julia
  - TechyTok! (Good tutorial): `https://techytok.com/from-zero-to-julia/`
  - QuantEcon (Another good tutorial with economic applications): `https://julia.quantecon.org/`
  - Julia Documentation: `https://docs.julialang.org/`
  - Flux.jl: `https://fluxml.ai/`

Thank you!

Appendix: Backpropagation Algorithm I

- Recall how the layers in the neural network were built from neurons

$$a_j^l = \phi\left(z_j^l\right) \quad \text{with} \quad z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l\,,$$

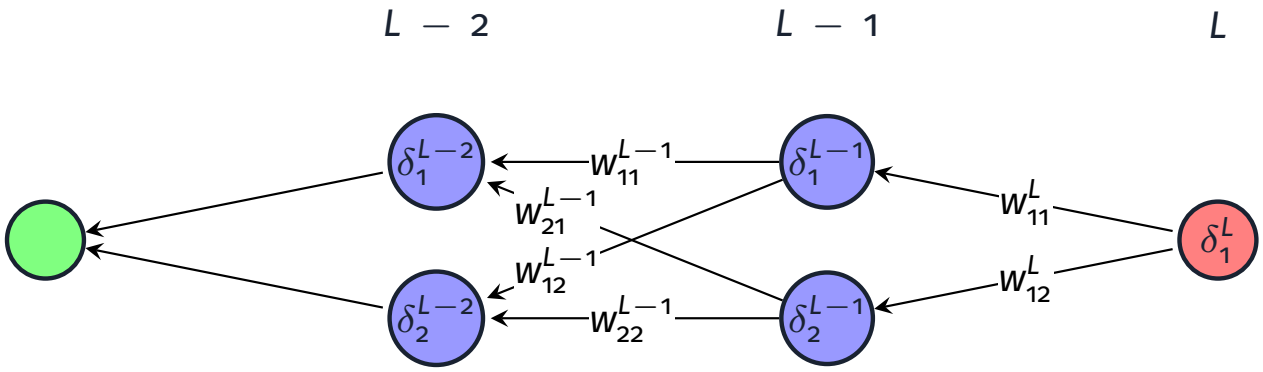
  where w_{ji}^l is associated with the connection from neuron i in layer $l-1$ to neuron j in layer l

- Our goal is to compute $\frac{\partial E_n}{\partial w_{ji}^l}$ for all weights in our network

- We will compute these derivatives starting from the output layer (which has index L) backwards through the network

Back

Appendix: Backpropagation Algorithm II



- It is convenient to define

$$\delta_j^l = \frac{\partial E_n}{\partial z_j^l}$$

which we call the error associated with neuron j in layer l

- Given this definition, note that for $l < L$ we have

$$\delta_j^l = \frac{\partial E_n}{\partial z_j^l} = \sum_k \frac{\partial E_n}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \phi'(z_j^l) \sum_k w_{kj} \delta_k^{l+1}$$

which gives us an expression that relates the errors from layer $l+1$ with the error at layer l

Appendix: Backpropagation Algorithm III

- With a single hidden layer, we would get the following
- We can compute for the output layer

$$\frac{\partial E_n}{\partial w_{ji}^L} = \frac{\partial E_n}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{ji}^L} = (a_j^L - y_{nj}) a_i^{L-1} = \delta_j^L a_i^{L-1}$$

- For the hidden layer, we have

$$\frac{\partial E_n}{\partial w_{ji}^{L-1}} = \frac{\partial E_n}{\partial z_j^{L-1}} \frac{\partial z_j^{L-1}}{\partial w_{ji}^{L-1}} = \delta_j^{L-1} x_{ni}$$

  where we can compute δ_j^{L-1} using the relation from the previous slide

Back

Appendix: Backpropagation Algorithm IV

- All of this can be more conveniently written in matrix notation

$$\begin{aligned}
\delta^L &= \nabla_a E_n = a^L - y_n \\
\delta^l &= \left(\left(w^{l+1} \right)' \delta^{l+1} \right) \odot \phi' \left(z^l \right) \\
\frac{\partial E_n}{\partial b^l} &= \delta^l \\
\frac{\partial E_n}{\partial w^l} &= \delta^l \left(a^{l-1} \right)'
\end{aligned}$$

  where $\delta^l = (\delta^l_1, \delta^l_2, \ldots)'$, $a^l = (a^l_1, a^l_2, \ldots)'$, $z^l = (z^l_1, z^l_2, \ldots)'$, $b^l = (b^l_1, b^l_2, \ldots)'$ and w^l is an appropriately defined matrix

- The gradient for E is then simply the sum of E_n over all observations

Back