Machine learning: prediction, classification and clustering

UBB Faculty of Sociology

Course Agenda

#1 Intro, Simple Linear Regression

#2 Python recap, Git, Handling data, EDA

#3 Regression, Decision Trees

#4 Bias, Variance, Overfitting, Classification, Metrics

#5 Random Forest Classifier, Clustering

#6 Neural Newtorks

#7 Help Final Project

#8 Help Final Project

6. Neural Networks

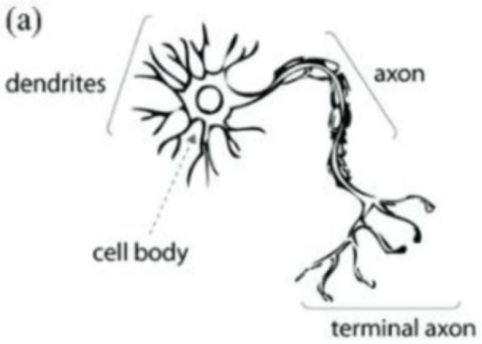
#6.1 Catch Up
#6.2 Neural Networks
#6.3 Gradient Descent
#6.4 Backpropagation
#6.5 NN Examples
#6.6 Final Project Task 5: Modeling NN - Optional
#6.7 Questions & Further reading

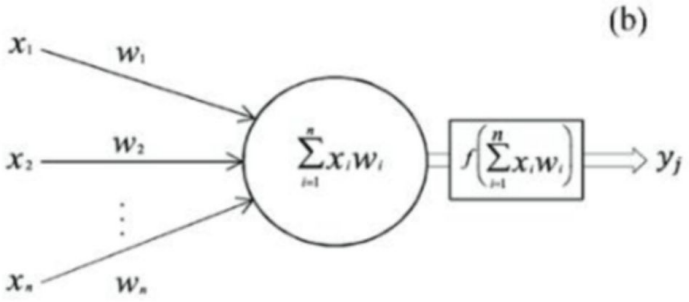
6.1 Catch up

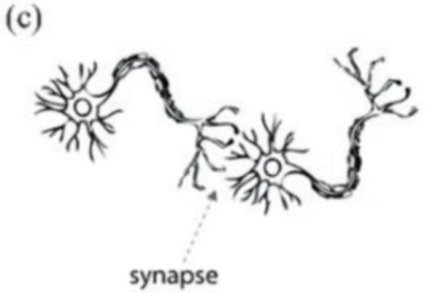
Share one thing that stood out to you:

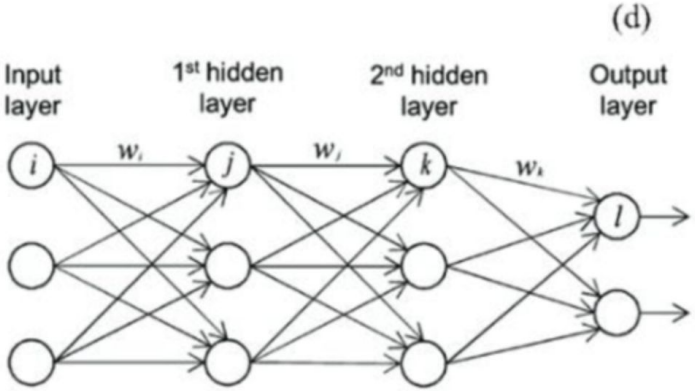
one thing you found surprising / interesting / useful etc.

6.2 Neural Networks







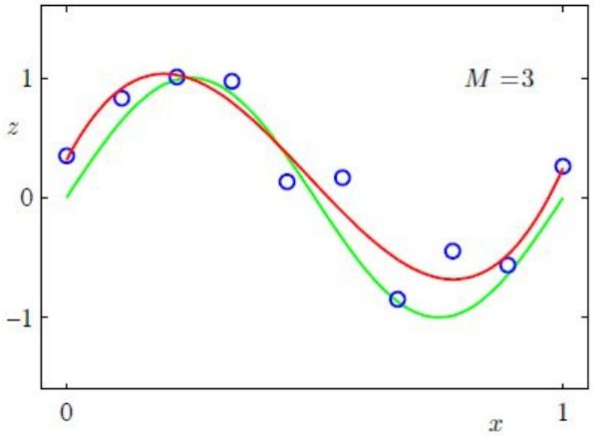


Linear regression (n features)

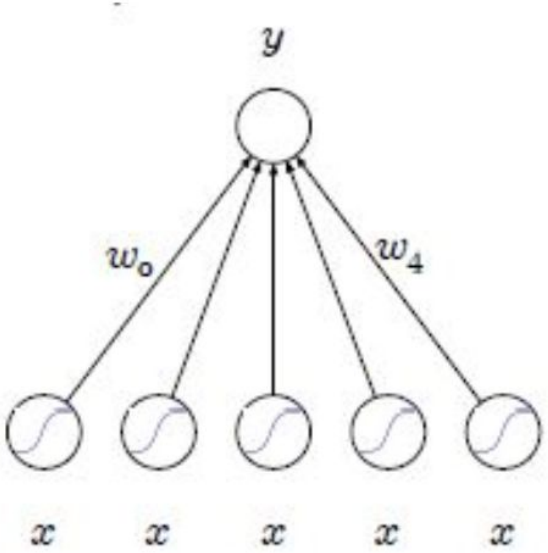
$$\hat{y} = f(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Polynomial regression (up to 3rd degree, 1 feature)

$$\hat{y} = f(x, w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



We can view this function as:



`A non-linear problem` When we are trying to map a 1:1 relationship (direct correlation between two variables) we call this a **linear problem**. It represents a linear relationship between our input data and our output data. It can be solved by linear algorithms.

A **non-linear problem**, in turn, must combine multiple inputs to predict an output. Only certain configurations of inputs yield the pattern we are searching for.

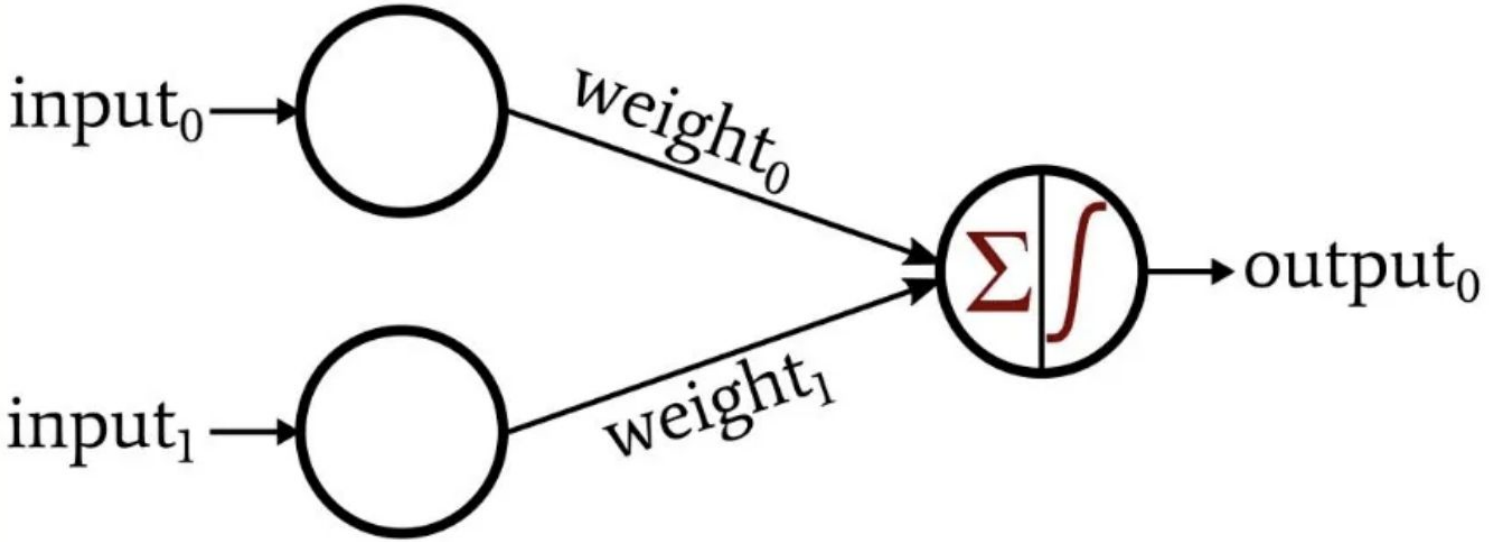
Linear regression (n features)

$$\hat{y} = f(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Polynomial regression (up to 3rd degree, 1 feature)

$$\hat{y} = f(x, w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

Single Layer Perceptron



Each node/neuron:

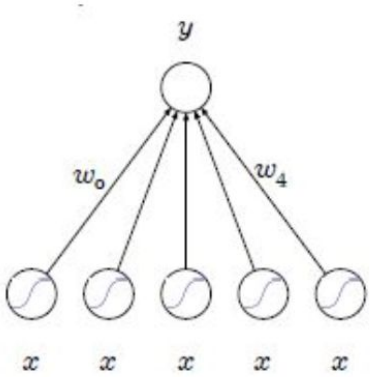
- **Summation Part**: This computes a **linear combination** of the inputs, similar to what happens in a linear regression model.
- **Activation Function**: This introduces **non-linearity**, allowing the network to model complex patterns.

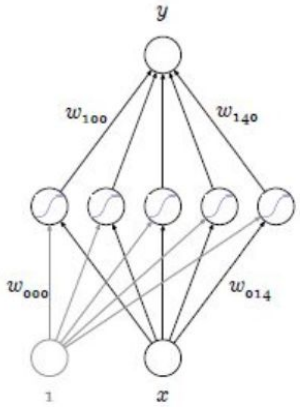
$$z = \sum_{i=1}^{n} w_i x_i + b$$

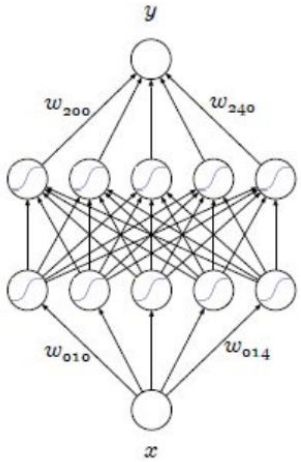
$$y = f(z)$$

Multi Layer Perceptron

We can continue adding layers.







$$Y = f(X) \rightarrow Y = f_1(f_0(X))$$

- Each neuron receives all the input data and uses the same summation and activation functions.
- Each neuron has its own unique set of weights and a bias, which are trained during learning.
- This allows each neuron to focus on different features or patterns in the data.
- The output of a neuron becomes the input for neurons in the next layer, enabling the network to build progressively more complex patterns from simpler ones.

How Neural Networks Solve Non-Linear Problems

Neural networks are well-suited to solving non-linear problems because they can combine inputs in complex ways and learn to detect patterns that are not obvious or straightforward.

How?

- Adding an activation function
Introduces non-linear relationships and capture complex interactions between inputs.

- Add more layers (with activation functions)

- **Early Layers**: Learn simple patterns (edges in an image or simple relationships in data).
- **Later Layers**: Combine these simple patterns to form more complex ones (e.g., shapes, objects, or high-level features).

Play TIME!

https://playground.tensorflow.org/

6.3 Gradient Descent

How ML Models Learn

Machine learning models learn a function that maps input data to output data by identifying patterns in the data.

They use mathematical operations, such as matrix transformations and differentiable functions, with a fixed number of parameters to make predictions or uncover relationships.

Neural networks (NNs), a subset of ML models, stand out by using multiple interconnected layers to learn hierarchical patterns, making them especially powerful for capturing complex, nonlinear relationships in data, unlike simpler traditional ML models.

How ML Models Learn

Learning = update “weights”

- so that the output to the error function becomes zero.
- an error of 0 means we predicted perfectly.

How to find the right weights?
- Gradient Descent

How ML Models Learn

Machine learning models, including neural networks, learn by minimizing error, using a loss function, which measures how well our model is predicting the target values

1. **Trial and Error**: The model predicts, compares its output to actual values, and adjusts its parameters (weights) to improve accuracy.

2. **Gradient Descent**: Models use gradient descent to update parameters (weights) by minimizing the loss, similar to finding the lowest point of a hill.

    1. **Traditional ML Models**: Apply **gradient descent** to optimize a small number of parameters through a single transformation, making them ideal for simpler, linear patterns.
    2. **Neural Networks**: Use **gradient descent** in combination with **backpropagation**, which calculates how errors flow backward through the layers. Backpropagation identifies how much each weight contributes to the error, allowing gradient descent to adjust weights layer by layer and capture complex, nonlinear patterns.

Through repeated updates, both traditional ML models and neural networks reduce error and improve their predictions. However, neural networks excel at uncovering intricate relationships due to their multi-layered architecture.

Gradient descent is a method used to minimize error (loss) by adjusting a model's parameters (weights).

It calculates how much a small change in the weights affects the error and updates the weights to reduce the error.

Using derivatives, gradient descent moves "downhill" on the error function to find its minimum.

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{\partial L}{\partial w}$$

Where:

- w_{new}: The updated weight.
- w_{old}: The current weight.
- α: The learning rate, controlling how big the update step is.
- $\frac{\partial L}{\partial w}$: The derivative of the loss function L with respect to the weight w, showing how L changes as w changes.

How to change a “weight” so that the “error” moves in a particular direction?
- through a “derivative” = the sensitivity to change of a quantity which is determined by another quantity.

How to use a derivative to learn?

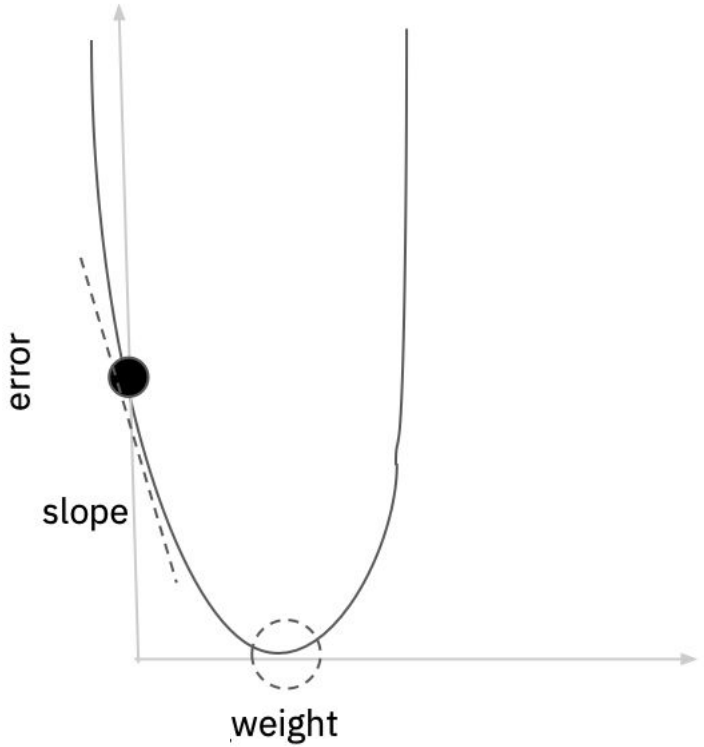
- Using gradient descen

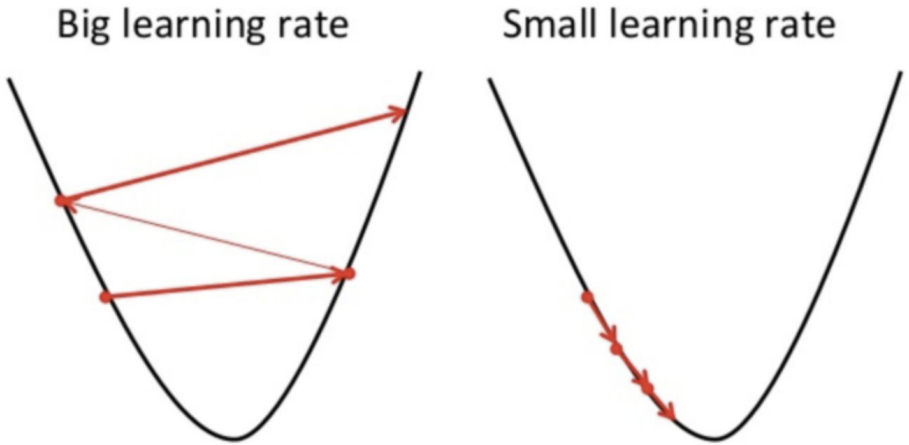
$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \frac{\partial L}{\partial w}$$

Adjusting our weights to reduce our error over a series of training examples ultimately just searches for CORRELATION between out input and our output layers.

If there is no correlation > the error will never reach 0.

Loss error function and Learning rate





This graph represents every value of error for every weight according to the relationship in the formula.

The black dot is at the point of BOTH our current weight and error. The dotted circle is where we want to be (error==0).

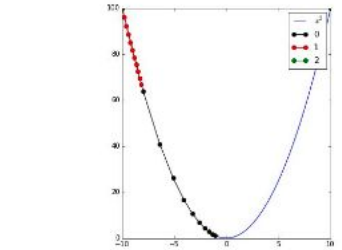
More problems?

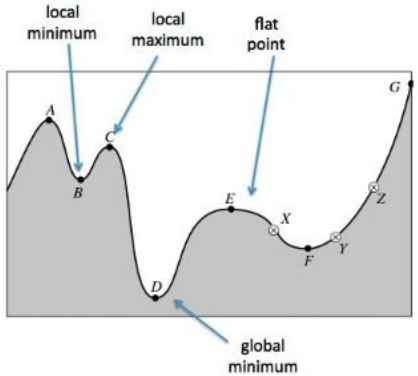
Since the function is not linear anymore, the solution space is not simple (convex).

You see a very small part of the function space.

You have no guarantee that you ever reach the global solution

Stuff that might help:
- Mini-batch learning : train with multiple samples at the same time
- advanced **update methods for gradient descent**:
    - Adagrad - Adaptive gradient: “Remember where you came from”
    - Adam - Adaptive moment estimation: ”Learn from the past and adapt your steps wisely”





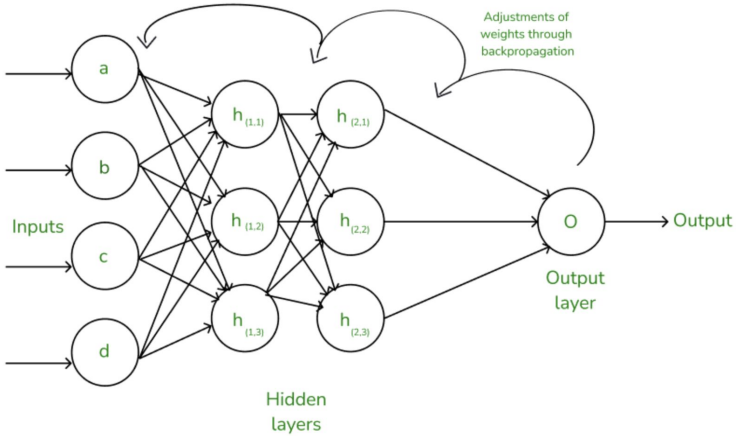
[Code on Colab Gradient Descent dummy example]

6.4 Backpropagation

Backpropagation (Simplified Explanation)

Backpropagation is the process neural networks use to learn from mistakes and improve predictions.

It adjusts the weights in the network layer by layer, working backward from the output to the input.



1. Forward Pass: Make a Prediction

- Input data is passed through the network, layer by layer
- The network calculates a prediction at the output layer.
- The error (or loss) is calculated by comparing the prediction to the actual target using a loss function.

2. Backward Pass: Error Propagation, Learn from the Mistake

- The network looks at the error and determines how much each weight contributed to it.
- It starts at the output layer and works backward to the earlier layers.
- Propagate error back, to hidden layers.
- At each Layer:

  2.1 Compute gradients (derivatives)

  - The network calculates how much to change each weight to reduce the error.

  - Done using derivatives, which tell how the error changes as the weights change.

  2.2 adjust the Weights

  - The weights are updated using a method called **gradient descent**, which makes small changes in the direction that reduces the error.

6.5 NN Examples

1. **Feedforward Neural Networks (FNN)**
• What they are: The simplest type of neural network, where information moves in one direction—from input to output. There are no loops or cycles.
• Example: Predicting whether a social media post is positive or negative in tone.
• Key Feature: Suitable for tasks where outputs depend only on current inputs.

2. **Convolutional Neural Networks (CNN)**
• What they are: Designed for processing grid-like data such as images or spatial data.
• Key Use Case: Analyzing visual patterns, such as identifying objects in photos or analyzing geographical heatmaps for urban studies.
• Key Feature: Uses layers called convolutions to detect patterns like edges, shapes, or textures in data.

3. **Recurrent Neural Networks (RNN)**
• What they are: Specialized for sequential or time-series data. Unlike feedforward networks, RNNs can "remember" past inputs due to internal loops.
• Key Use Case: Predicting future trends in survey responses or analyzing patterns in longitudinal studies.
• Key Feature: Good at handling time-related patterns, but can struggle with long sequences.

4**. Long Short-Term Memory Networks (LSTM)**
• What they are: A special type of RNN that can handle long-term dependencies. They "remember" important information while forgetting irrelevant details.
• Key Use Case: Modeling complex social phenomena that evolve over long periods, such as migration trends.
• Key Feature: Overcomes RNN limitations by selectively remembering information.

5. **Generative Adversarial Networks (GANs)**

- What they are: Composed of two networks—a generator and a discriminator—that compete against each other. The generator creates data, and the discriminator evaluates its authenticity.
- Key Use Case: Generating realistic fake social media posts or synthetic datasets for research.
- Key Feature: Generates new data that mimics real-world data.

6. **Transformer Networks**

- What they are: Powerful models designed for sequential data but focus more on capturing relationships between all parts of a sequence simultaneously.
- Key Use Case: Analyzing large bodies of text for sociological insights (e.g., GPT-based models).
- Key Feature: Handles long sequences well, often used in modern natural language processing.

7. **Autoencoders (Encoders, Decoders)**

- What they are: Networks designed to compress data into smaller representations and then reconstruct it.
- Key Use Case: Analyzing patterns in survey data or dimensionality reduction for visualization.
- Key Feature: Useful for uncovering hidden structures in data.

Summary of Key Applications (in Sociology)

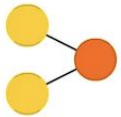
- Feedforward and CNNs: Useful for tasks like image analysis or static predictions.
- RNNs and LSTMs: Ideal for time-series data, such as societal trends or behavioral sequences.
- Transformers: Great for analyzing text-heavy data, such as policy reviews or interviews.
- GANs and Autoencoders: Help with generating synthetic datasets or understanding hidden patterns.

A mostly complete chart of

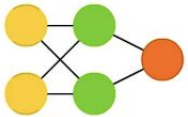
Neural Networks

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

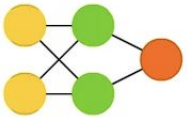
Perceptron (P)



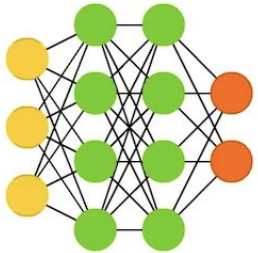
Feed Forward (FF)



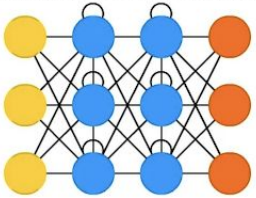
Radial Basis Network (RBF)



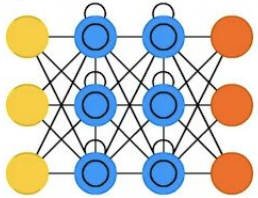
Deep Feed Forward (DFF)



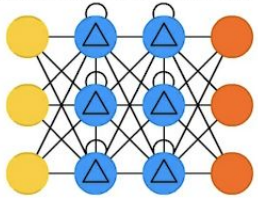
Recurrent Neural Network (RNN)



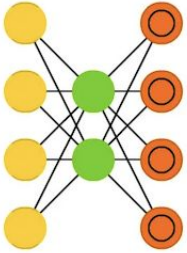
Long / Short Term Memory (LSTM)



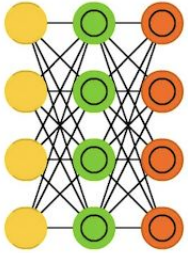
Gated Recurrent Unit (GRU)



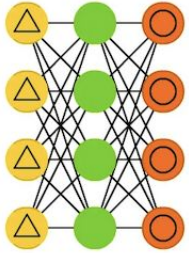
Auto Encoder (AE)



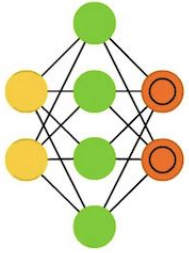
Variational AE (VAE)

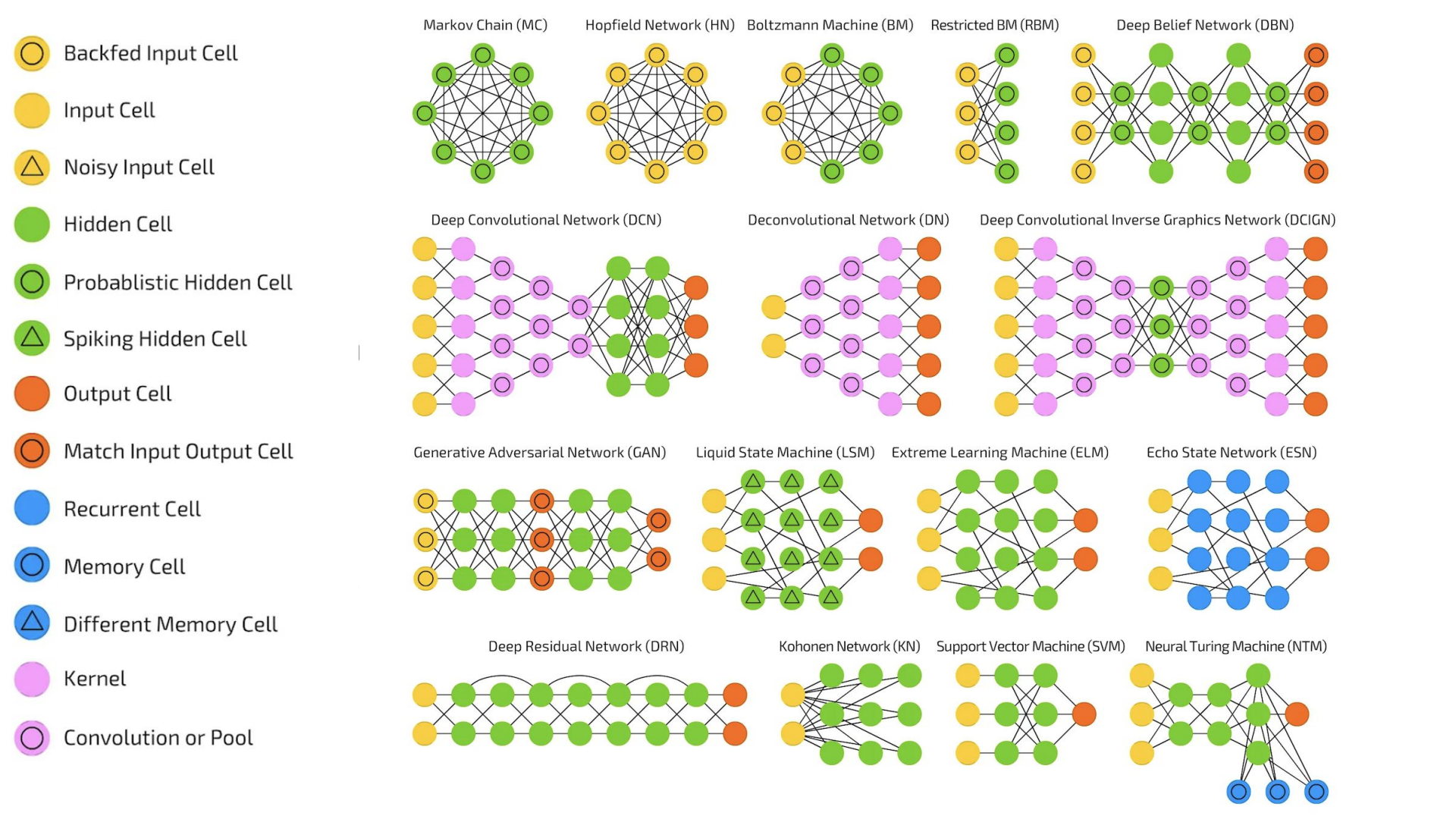


Denoising AE (DAE)



Sparse AE (SAE)




Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Convolutional Neural Network (CNN)

What is a CNN? A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for processing structured grid-like data, such as images. It excels in tasks like image classification, object detection, and facial recognition.

Key Components:

1. **Convolutional Layers:** Extract features from input images by applying filters (kernels). These filters detect patterns like edges, textures, or shapes.
2. **Pooling Layers:** Downsample the feature maps, reducing their size while retaining important information (e.g., max pooling selects the highest value in a region).
3. **Fully Connected Layers:** Flatten the output and perform final classification or regression based on extracted features.

How It Works:

- **Input Image:** Fed as pixel values into the network.
- **Feature Extraction:** Convolutional layers learn hierarchical features (simple to complex).
- **Dimensionality Reduction:** Pooling layers reduce data size, ensuring computational efficiency.
- **Classification:** Fully connected layers use extracted features to classify the input.
- https://poloclub.github.io/cnn-explainer/

Key Advantages:

- Efficient feature extraction and parameter sharing (using kernels).
- Works with minimal preprocessing.
- Handles large-scale image data effectively.

Applications:

- Image recognition, video analysis, autonomous vehicles, medical imaging, and more.

Play time!

Play with NN: https://playground.tensorflow.org/.

Play with CNN: https://poloclub.github.io/cnn-explainer/

Play with GAN: https://poloclub.github.io/ganlab/

Play with RNN: https://damien0x0023.github.io/rnnExplainer/

6.6 Final Project Task 5: Modeling NN - Optional

https://github.com/zahariesergiu/ubb-sociology-ml/blob/main/final_project/Final_Project_Task_5_Census_Modeling_NN_Regression%20-%20Optional.ipynb

6.7 Further Reading & Questions

#1 Neural networks – a guide for my mom https://blog.imaginationtech.com/neural-networks-a-guide-for-my-mom/

#2 Neural Networks an Introduction: https://blog.wolfram.com/2019/05/02/neural-networks-an-introduction/

#3 Neural Networks Deep Learning explained:
https://www.wgu.edu/blog/neural-networks-deep-learning-explained2003.html

#4 An Intuitive Explanation of Convolutional Neural Networks:
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Thank you !!

Machine Learning Engineer / Data Scientist
zahariesergiu@gmail.com
https://www.linkedin.com/in/zahariesergiu/
https://github.com/zahariesergiu/ubb-sociology-ml