Introduction to Machine Learning for Economists and Business Analysts

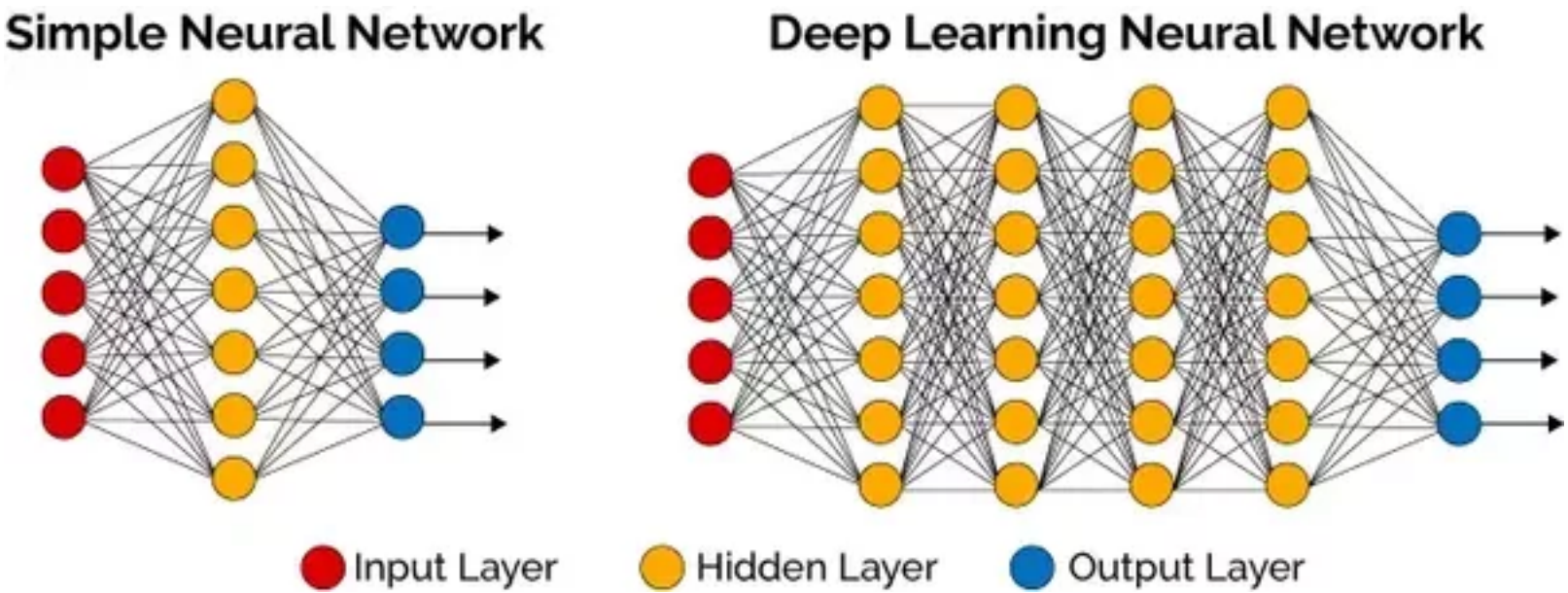
Neural Networks

Anthony Strittmatter

Literature

- Hastie, Tibshirani, and Friedman (2009): "Elements of Statistical Learning", 2nd ed., Springer, download,
  Chapter 11.1-11.8.
- Goodfellow, Bengio, and Courville (2016): "Deep Learning", MIT Press, download.
- Gurney (1997): "An Introduction to Neural Networks", UCL Press, download.

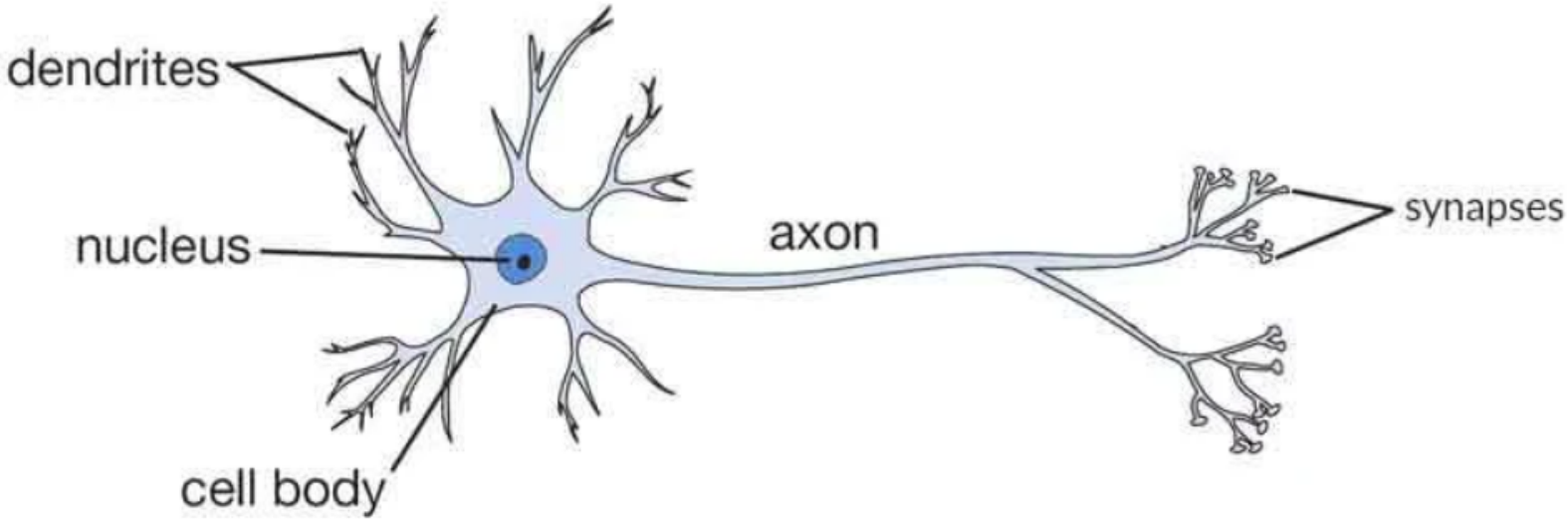
Neural Networks



Basic Idea of Neural Networks

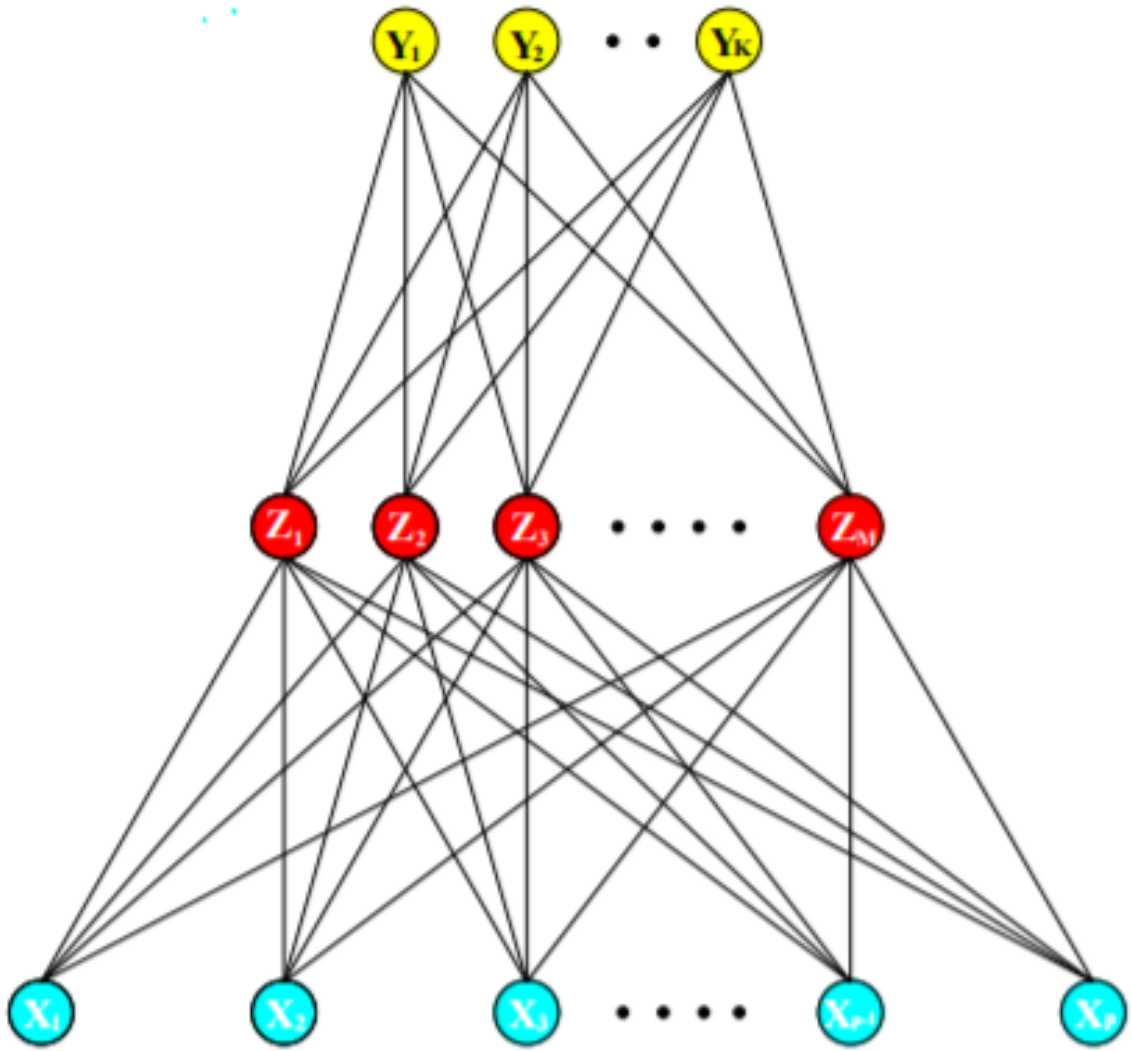
- Neural networks are complex (non-linear) adaptive systems
- Adaptive means it has the ability to change its internal structure by adjusting weights of inputs
- Neural networks were first developed to model human brains
- They are composed of a large number of highly interconnected processing elements known as the neurons
- The connections between neurons represent synapses
- Early neurons fired when the signal passed a certain threshold, which was mimicked by step-functions
- Nowadays, neurons use smoother threshold functions like the sigmoid

Biological Neuron



- Human brains consist of billions of neural cells that process information
- Each neural cell considered a simple processing system
- Dendrites of a neuron receive input signals from another neuron
- Based on those inputs, fire an output signal via an axon

Single Hidden Layer Network



Source: Hastie, Tibshirani, and Friedman (2009)

Two Step Linear Regression or Classification Model

- $X = (X_1, ..., X_P)$ are the **input** layers
- $Z = (Z_1, ..., Z_M)$ are the **neurons or hidden units** of the hidden layer (here only 1 hidden layer)
- $Y = (Y_1, ..., Y_K)$ are the **output** layers
- Predicted output $f_k(X) = (f_1(X), ..., f_K(X))$

$$Z_m = \alpha_{0m} + X\alpha_m \text{ for all } m = 1, ..., M,$$
$$f_k(X) = \beta_{0k} + Z\beta_k \text{ for all } k = 1, ..., K$$

- $\alpha_m = (\alpha_{1m}, ..., \alpha_{Pm})'$ and $\beta_k = (\beta_{1k}, ..., \beta_{Mk})'$ are coefficient vectors (called weights)
- α_{0m} and β_{0k} are intercepts (called bias terms)

Two Step Linear Regression or Classification Model

Merging the two equations:

$$f_k(X) = \beta_{0k} + (\alpha_{01} + X\alpha_1)\beta_{1k} + ... + (\alpha_{0M} + X\alpha_M)\beta_{Mk}$$

$$f_k(X) = \underbrace{\beta_{0k} + \sum_{m=1}^{M} \alpha_{0m}\beta_{1k}}_{=\gamma_0} + \sum_{m=1}^{M} X \underbrace{\alpha_m\beta_{mk}}_{=\gamma_{mk}}$$

$$f_k(X) = \gamma_0 + \sum_{m=1}^{M} X\gamma_{mk}$$

$\Rightarrow$ Breaks down to a simple linear model

Single Hidden Layer Network

Neural networks are non-linear modelling tools:

- $\sigma(\cdot)$ activation function
- $g_k(\cdot)$ is the outcome transformation function

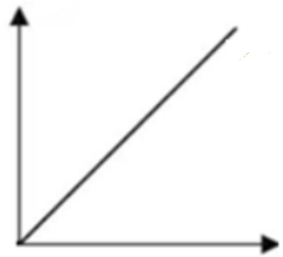
$$Z_m = \sigma(\alpha_{0m} + X\alpha_m) \text{ for all } m = 1, ..., M,$$
$$f_k(X) = g_k(\underbrace{\beta_{0k} + Z\beta_k}_{=T_k}) \text{ for all } k = 1, ..., K$$

Merging the two equations:

$$f_k(X) = g_k(\beta_{0k} + \sigma(\alpha_{01} + X\alpha_1)\beta_{1k} + ... + \sigma(\alpha_{0M} + X\alpha_M)\beta_{Mk})$$

Selection of Activation Functions

- **Identity functions** map input to the same output value, $\sigma(v) = v$.

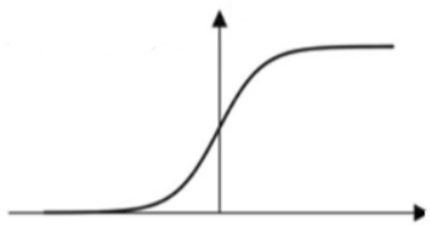


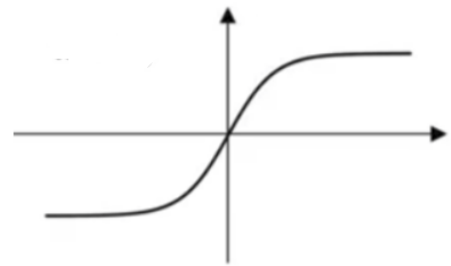
- **Binary step functions** set the output True (or activated) if the value of s is above a certain threshold value c and the output is false (or not activated) if the value of v is below the threshold, $\sigma(v) = 1\{v > s\}$.

Selection of Activation Functions

- **S-shaped functions** use often the sigmoid or the hyperbolic tangent transformation:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \quad \text{or} \quad \sigma(v) = \tanh(v) = \frac{\sin(v)}{\cos(v)}.$$

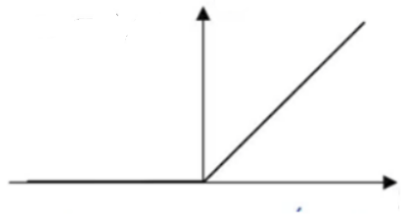




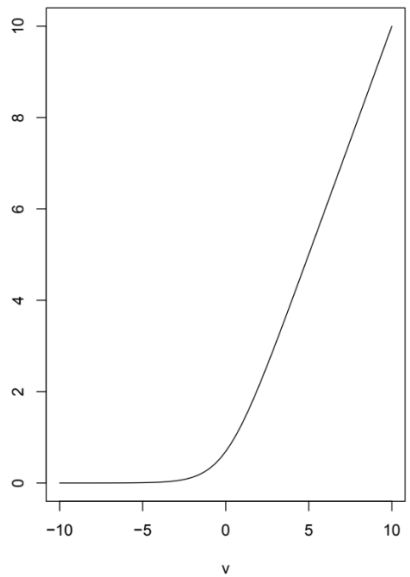
- **Radial basis functions** transform the input based on the distance (e.g. Euclidean or Mahalanobis) to some center vector c, $\sigma(v) = \rho(\|v - c\|)$. The radial basis function $\rho(\cdot)$ is commonly taken to be Gaussian.

Selection of Activation Functions

- **Ramp functions** map negative inputs to 0 and positive inputs to the same output, $\sigma(v) = max(0, v)$.



- **Rectified linear unit (ReLU) functions**, $\sigma(v) = log(1 + e^v)$, are similar to ramp functions but without kink.



Outcome Transformation Function

- The output transformation allows to account for the distribution of the output (e.g., continuous or binary).
- This is relevant to distinguish between regression and classification problems.
- For regression problems we typically choose the identity function, $g_k(T_k) = T_k$.
- For classification problems the *softmax* function

$$g_k(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

is often used. This is the same transformation that is used for the multinomial Logit. It produces positive estimates that sum to one across the K outcomes (e.g. categorical outcome variables).

Optimization of the Network

- We need to estimate values for the coefficients (or weights) $\theta = \{\alpha_{0m}, \alpha_m, \beta_{0k}, \beta_k : m = 1, ..., M; k = 1, ..., K\}$.
- For regressions, we minimize the sum of squared errors

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_{ik}))^2.$$

- For classification, often the cross-entropy is minimized

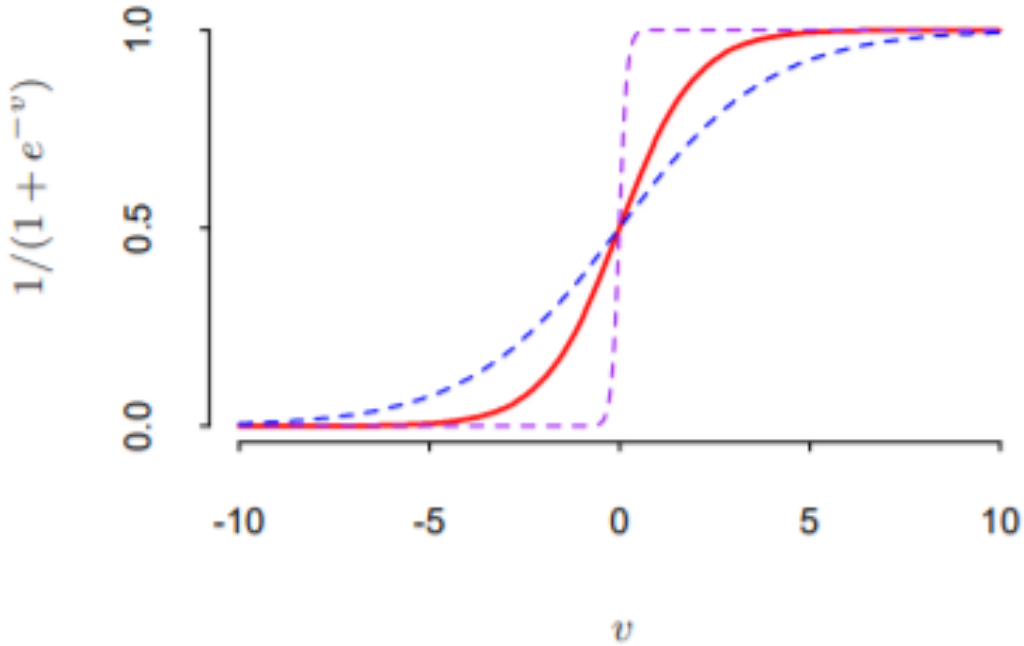
$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} log(f_k(x_{ik})).$$

- $R(\theta)$ is minimized with a specific gradient descent algorithm, which is called *back-propagation algorithm*.

Starting Values and Local Minima

- Usually starting values for the coefficients (weights) are chosen to be random values near zero.
- The sigmoid is then an almost linear function. Accordingly, we start (almost) linearly and move to a non-linear function.
- The algorithm does not converge when the starting values are exactly zero.
- Often the error function $R(\theta)$ is is non-convex and has many local minima.
- The final solution depends on the choice of the starting values.
- We have to try several starting values and select the results with the smallest error function $R(\theta)$.
- Alternatively, we can average the results across the different starting values (*bagging*), which reduces the out-of-sample variance.

Sigmoid Function



Source: Hastie, Tibshirani, and Friedman (2009)

- Red curve is the sigmoid function $\sigma(v) = \frac{1}{1+e^{-v}}$.
- The dashed lines show weighted sigmoid functions $\sigma(sv)$, where the blue line shows the function for $s = 0.5$ and the violet line for $s = 10$.

Rescaling

- The scale of the inputs affects the scaling of the coefficients (weights).
- This matters particular when we choose the starting values.
- Accordingly, all input layers $X = (X_1, ..., X_P)$ should be standardized (mean of zero and variance of one).
- With standardized input layers, the starting values are typically chosen from a uniform distribution with the range $[-0.7, +0.7]$.

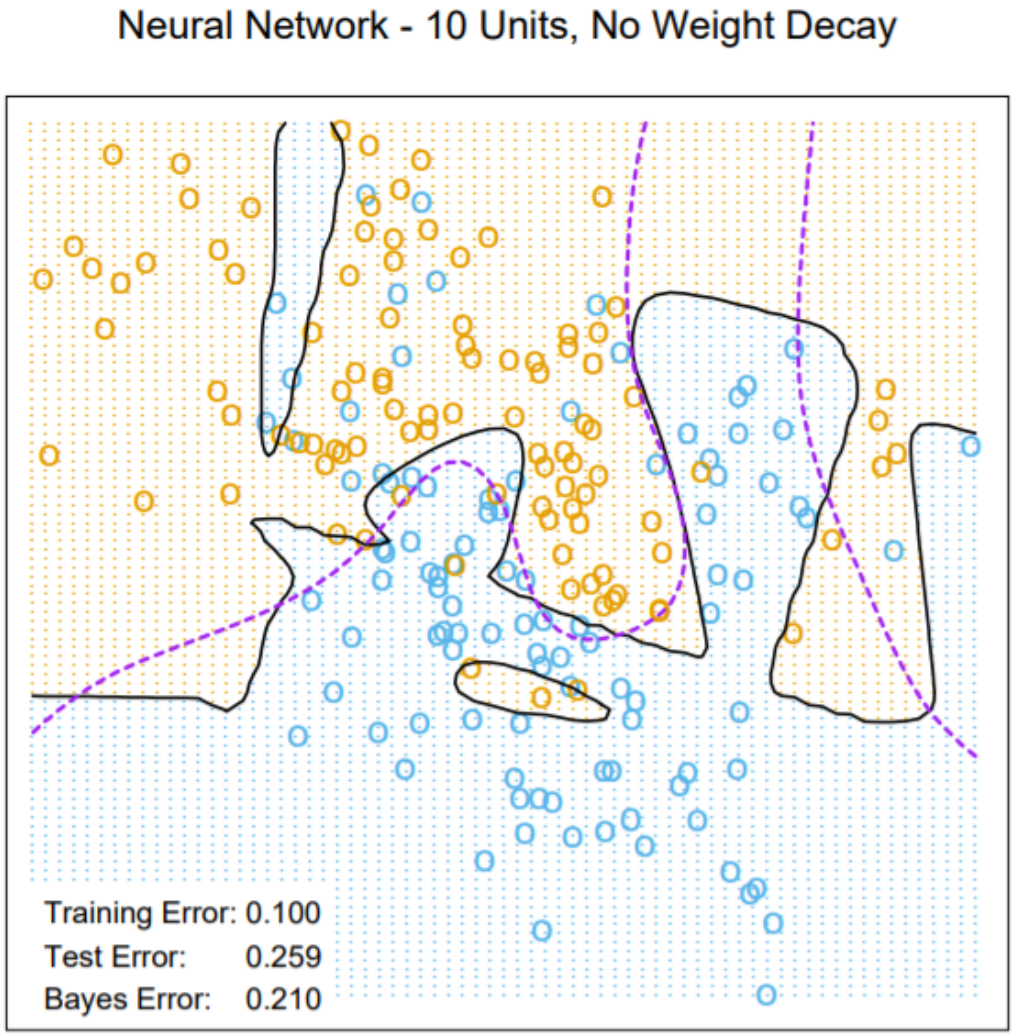
Overfitting

- Neural nets notoriously overfit the data.
- We can use *early stopping rules* to avoid overfitting. A validation data set can be used to determine the optimal time to stop.
- A more explicit method to avoid overfitting is *regularization* or *weight decay*. We add to the error function a penalty term, $R(\theta) + \lambda J(\theta)$.
- A common approach is to use the analogous of the Ridge regression, with the penalty

$$J(\theta) = \sum_{m=1}^{M} \alpha_m^2 + \sum_{k=1}^{K} \beta_k^2$$

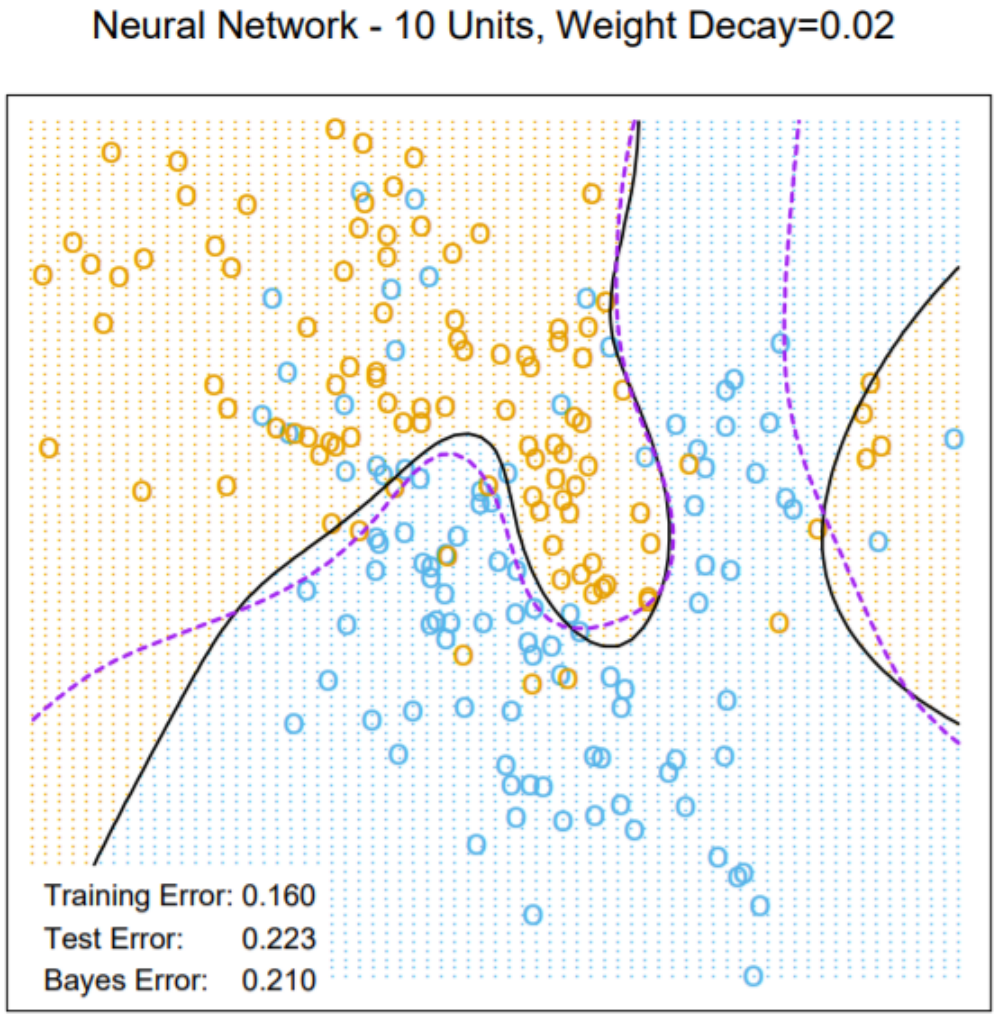
- The tuning parameter $\lambda \geq 0$ is specified with a cross-validation procedure.

Example of an Overfitted Network



Source: Hastie, Tibshirani, and Friedman (2009)

Example with Weight Decay



Source: Hastie, Tibshirani, and Friedman (2009)

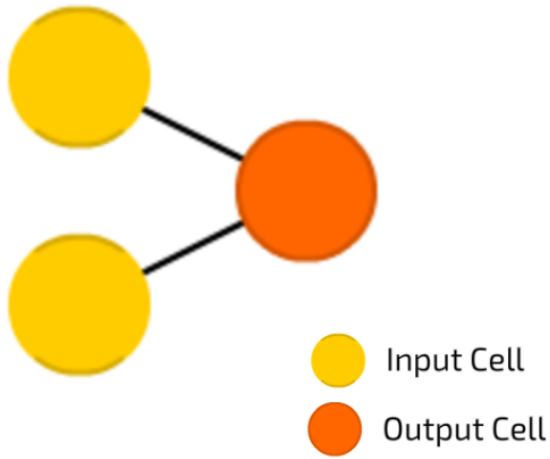
Number of Hidden Units or Neurons

- It is better to have too many than too few hidden units.
- With too few hidden units the model does not have enough flexibility.
- With too many hidden units, we can shrink the extra weights towards zero by choosing the appropriate regularization.
- Typically, the number of hidden units is between 2 and 100; and increases with the number of input layers and the sample size.
- Some people say that the number of hidden units should be between the size of the input layer and the output layer (but this is not always the case).

Number of Hidden Layers

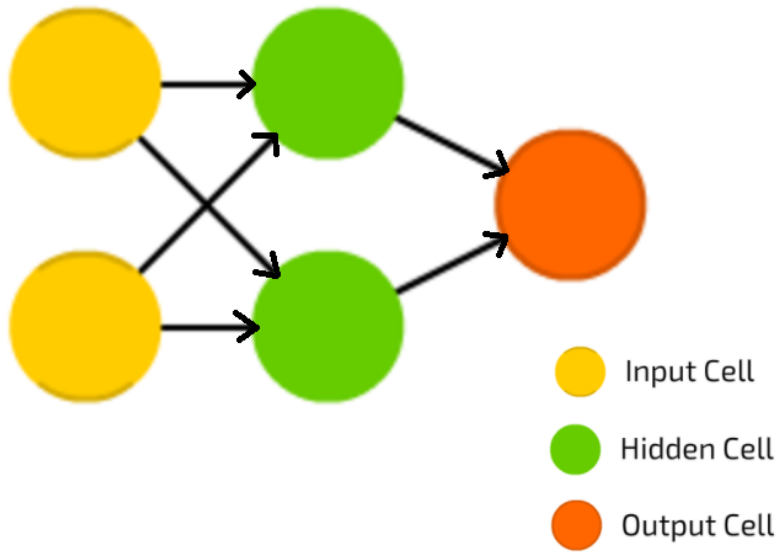
- The choice of the number of hidden layers is guided by background knowledge, experience, and experimentation.
- Using multiple hidden layers (as opposed to the single hidden layer we considered until now) allows to build hierarchical models or to model different resolution levels.
- Networks with 3 or more hidden layers are called *deep neural networks*.

Perceptron



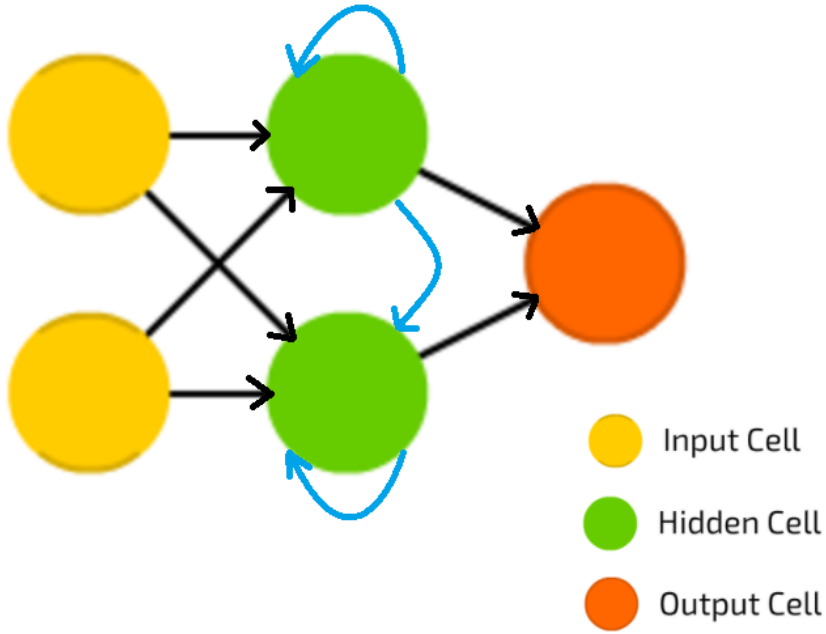
- Has no hidden layers and no hidden units.
- Sums up some inputs, applies the outcome transformation function, and passes them to the output layer.

Feed Forward Neural Network



- Connections between the nodes do not form a cycle.
- Activation flows from input layer to output, without back loops.

Recurrent Neural Network (RNN)

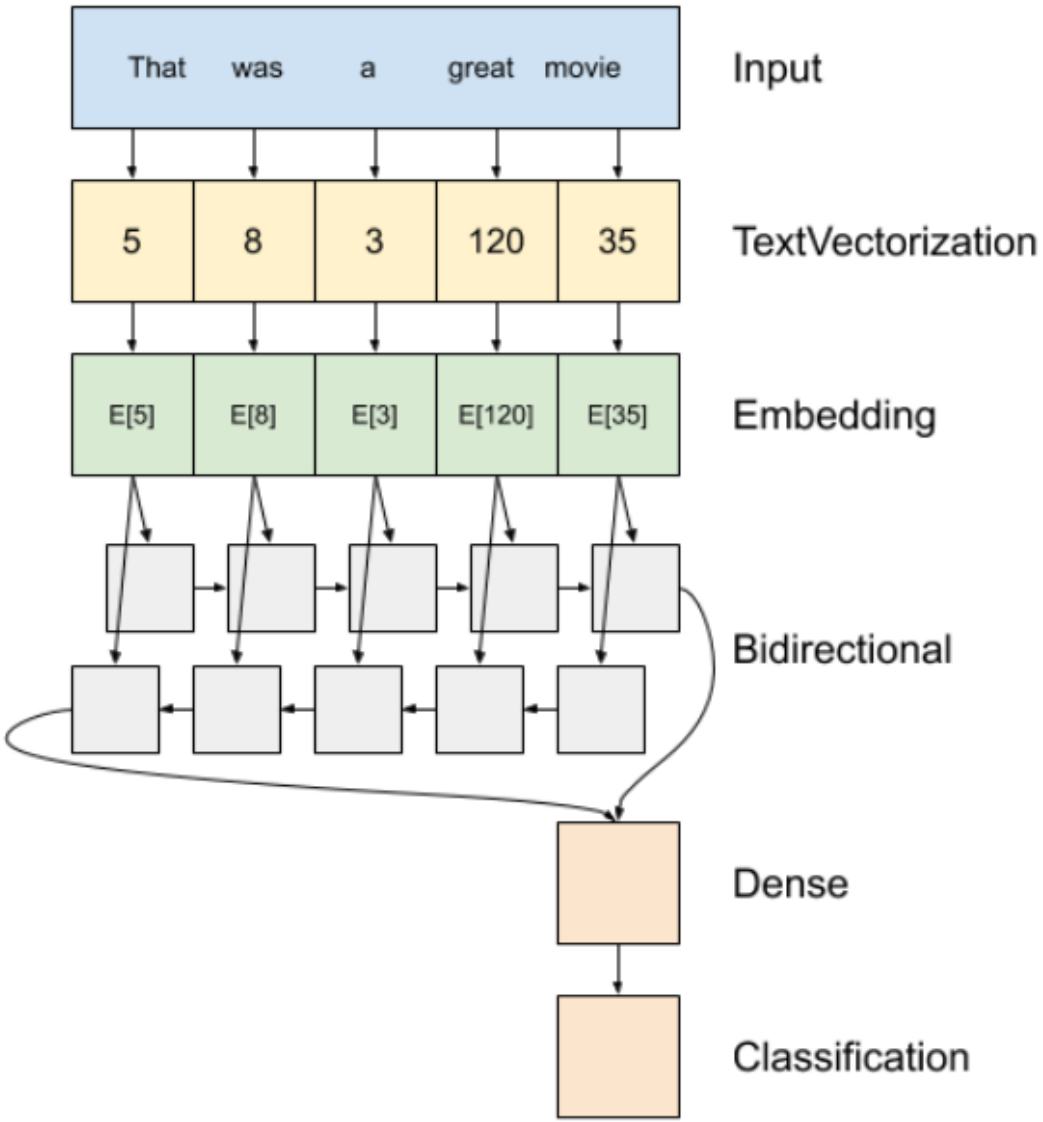


- RNN have connections to between hidden units of the same or previous layers (often with a temporal lag).
- This allows model different aggregation levels jointly or to "remember" the previous input.
- Accordingly, RNNs are suited to model data along a temporal sequence (or sequentially arriving data).

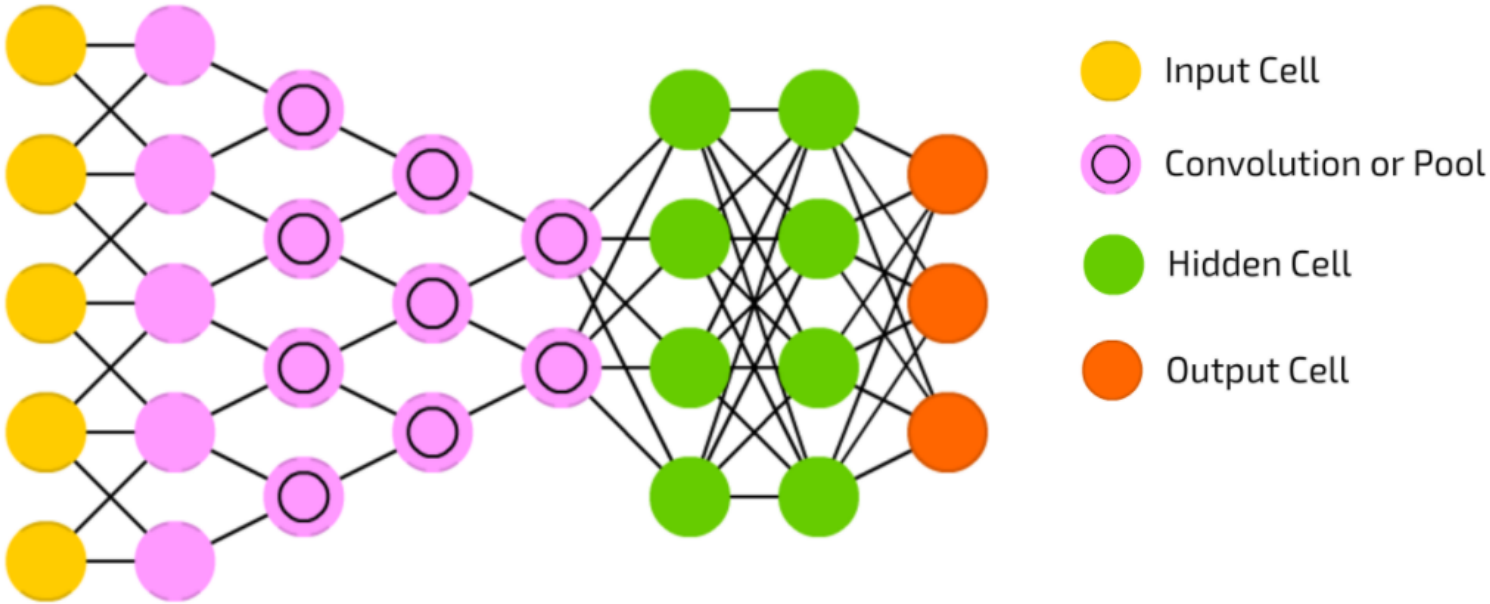
Recurrent Neural Network

- *Direct feedback* is the connection between the output and input of the same neuron.
- *Indirect feedback* is the connection between the output of a neuron and the input of a different neuron of the same layer.
- *Lateral feedback* is the connection between the output of a neuron and the input of a neuron of a previous layer.

Example: Text Recognition

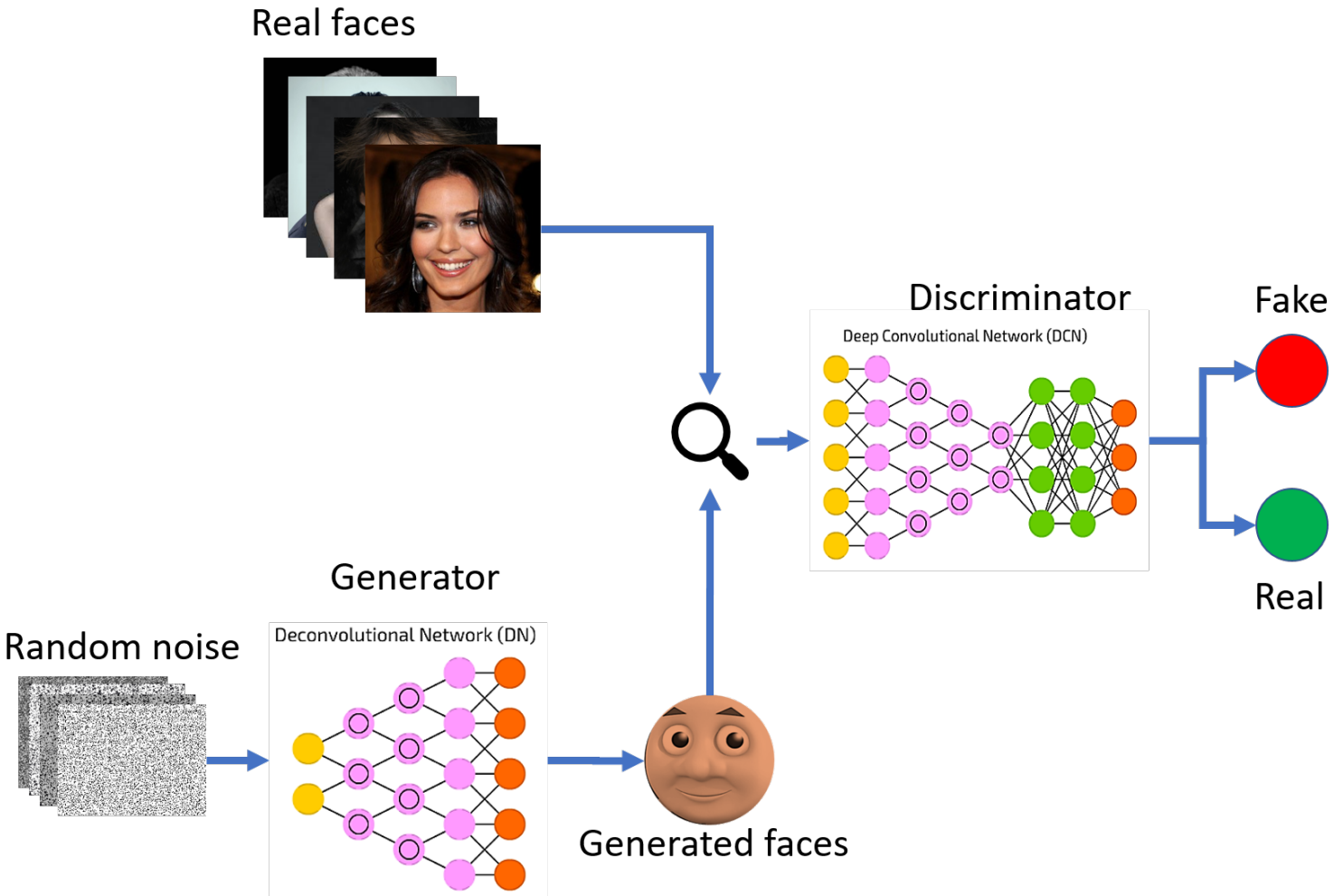


Convolutional Neural Network (CNN)

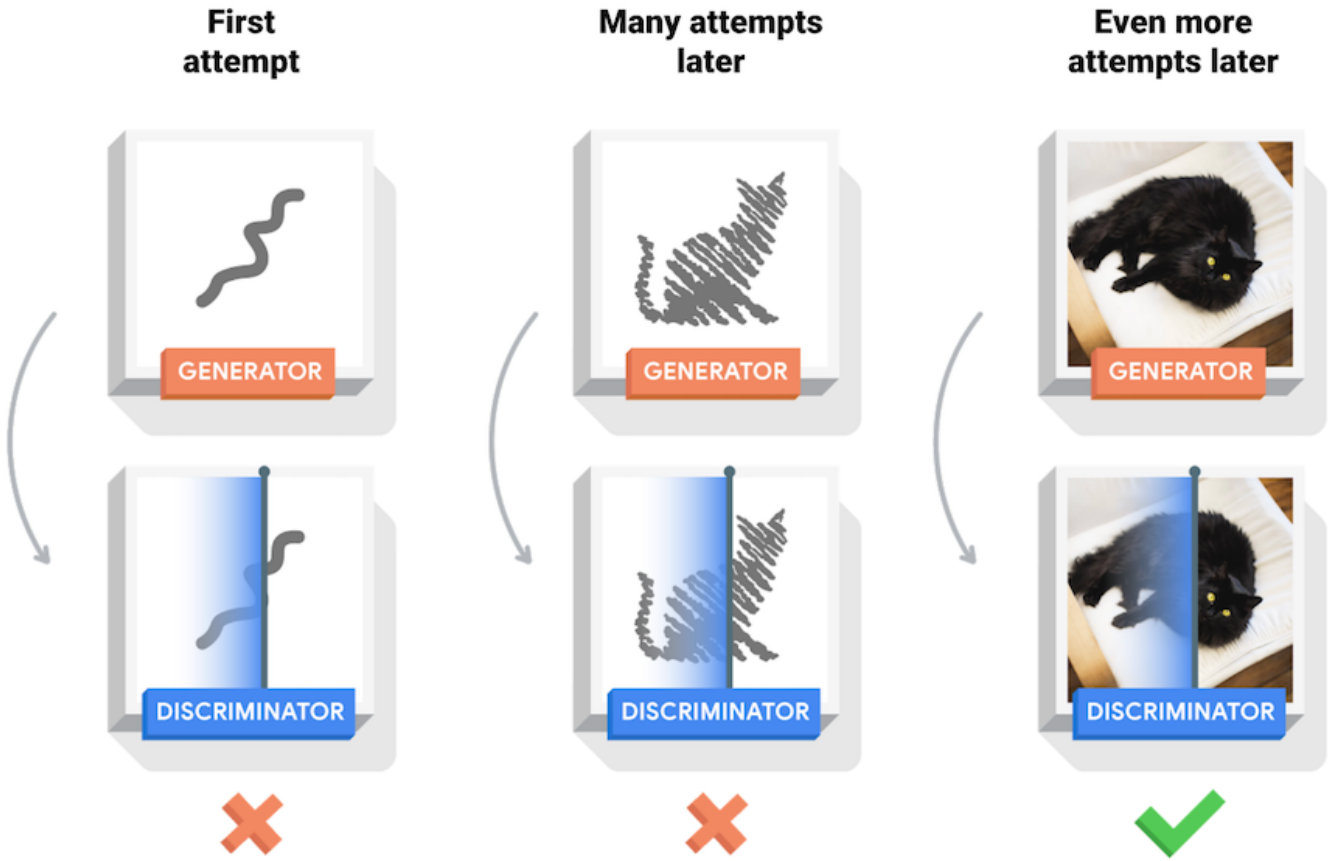


- CNNs reduce the dimension of the input layer without loosing their characteristics.
- They use a hierarchical structure instead of fully connected cells.
- Then they build a neural net using the convolutions as input.

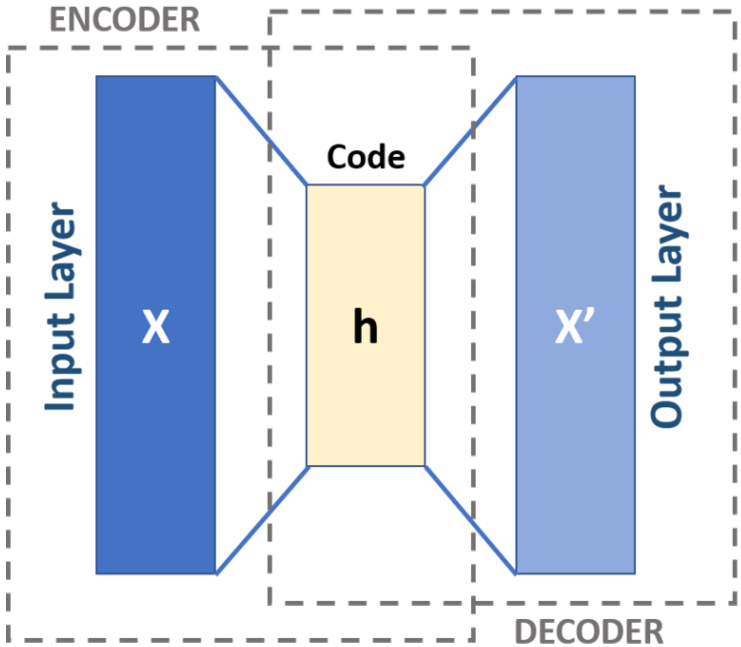
Generative Adversarial Neural Networks (GANs)



Generative Adversarial Neural Networks (GANs)



Autoencoder



- Encoder reduces the dimensionality of the input layer.
- Decoder tries to restore the input layer from the coded data.
- Model accuracy is measured by comparing the input layer X with the output layer X'.

$\Rightarrow$ Unsupervised machine Learning

Tools to Train Deep Neural Nets

Frameworks:

- Tensor Flow (can be deployed in R using the *keras* package)
- H2O.ai (R package *h2o*)

Cloud infrastructure:

- Vertex AI (Google Cloud AI Platform)
- Amazon SageMaker
- RapidMiner