Universität Konstanz

04 | Regularization of Neural Networks

Giordano De Marzo

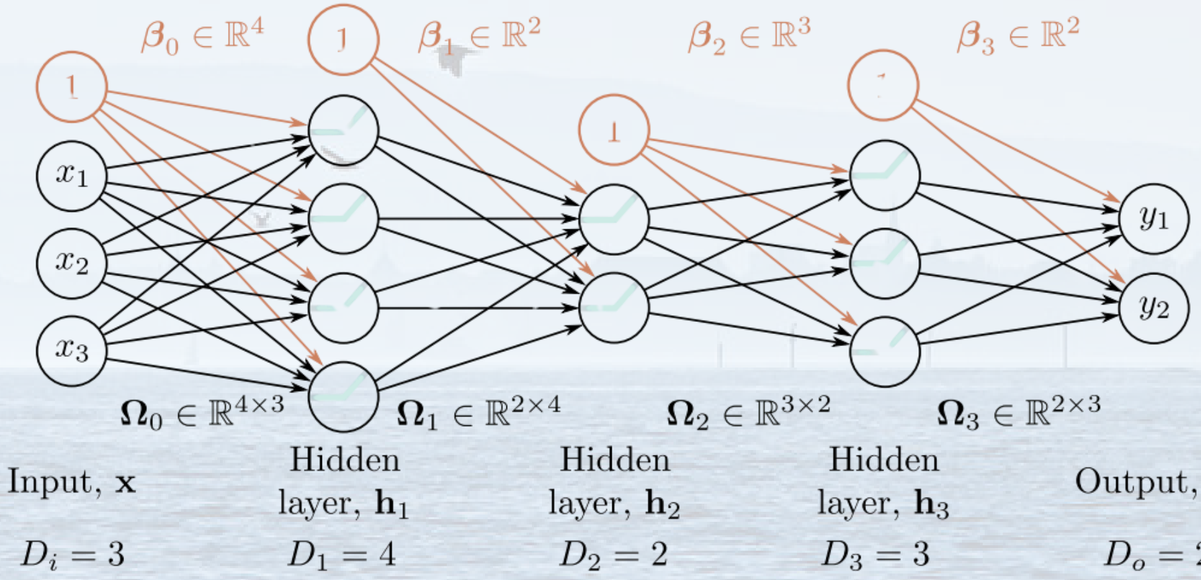
https://giordano-demarzo.github.io/

Deep Learning for the Social Sciences

Recap: Multilayer Perceptron

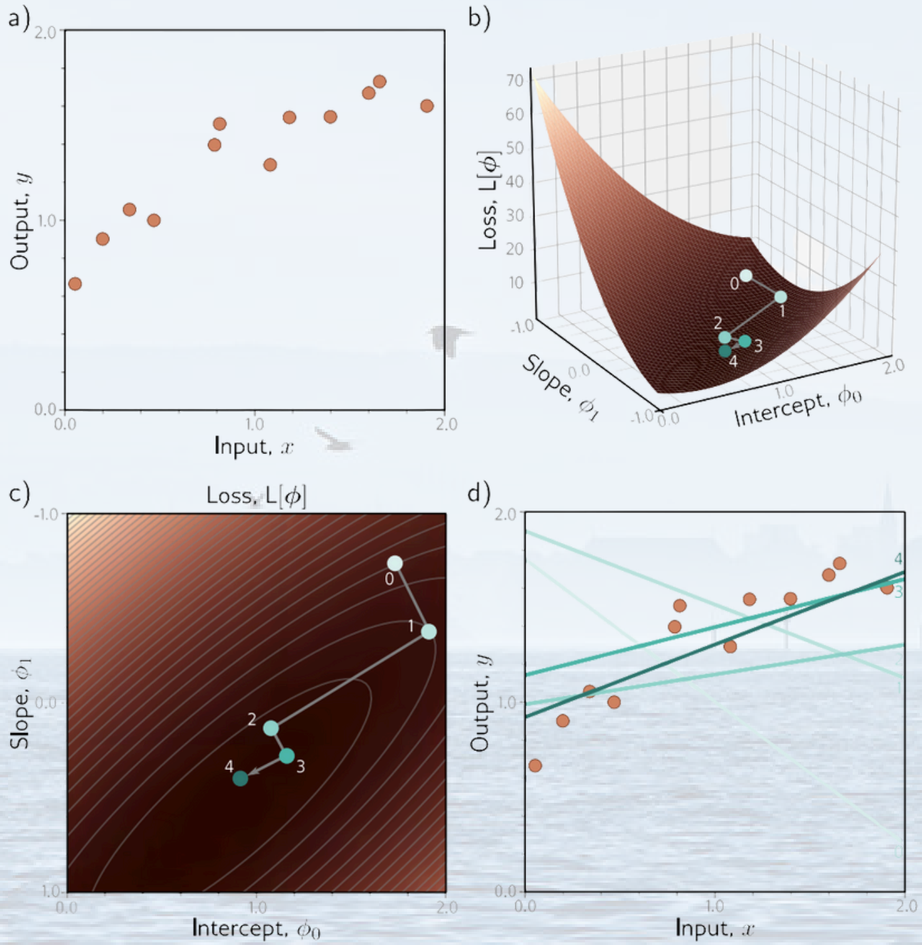
A general Multilayer Perceptron consists of K hidden layers $\mathbf{h}_1 ... \mathbf{h}_k$, for a total of K+1 weight matrices $\boldsymbol{\Omega}_0 ... \boldsymbol{\Omega}_k$ and bias vectors $\boldsymbol{\beta}_0 ... \boldsymbol{\beta}_k$.

Matrix notation is useful to represent large neural networks in a compact notation.



$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}] \\
\mathbf{h}_2 &= \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1] \\
\mathbf{h}_3 &= \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2] \\
&\vdots \\
\mathbf{h}_K &= \mathbf{a}[\boldsymbol{\beta}_{K-1} + \boldsymbol{\Omega}_{K-1} \mathbf{h}_{K-1}] \\
\mathbf{y} &= \boldsymbol{\beta}_K + \boldsymbol{\Omega}_K \mathbf{h}_K.
\end{aligned}
$$

Recap: Gradient Descent



The **Gradient Descent** is an iterative algorithm that uses the gradient of the Loss function to compute how to update the weights in order to make the Loss decrease. It consists of two steps

- we denote by $\mathbf{W_t}$ the set of all weights and biases at iteration t and we compute the gradient of the Loss in $\mathbf{W_t}$

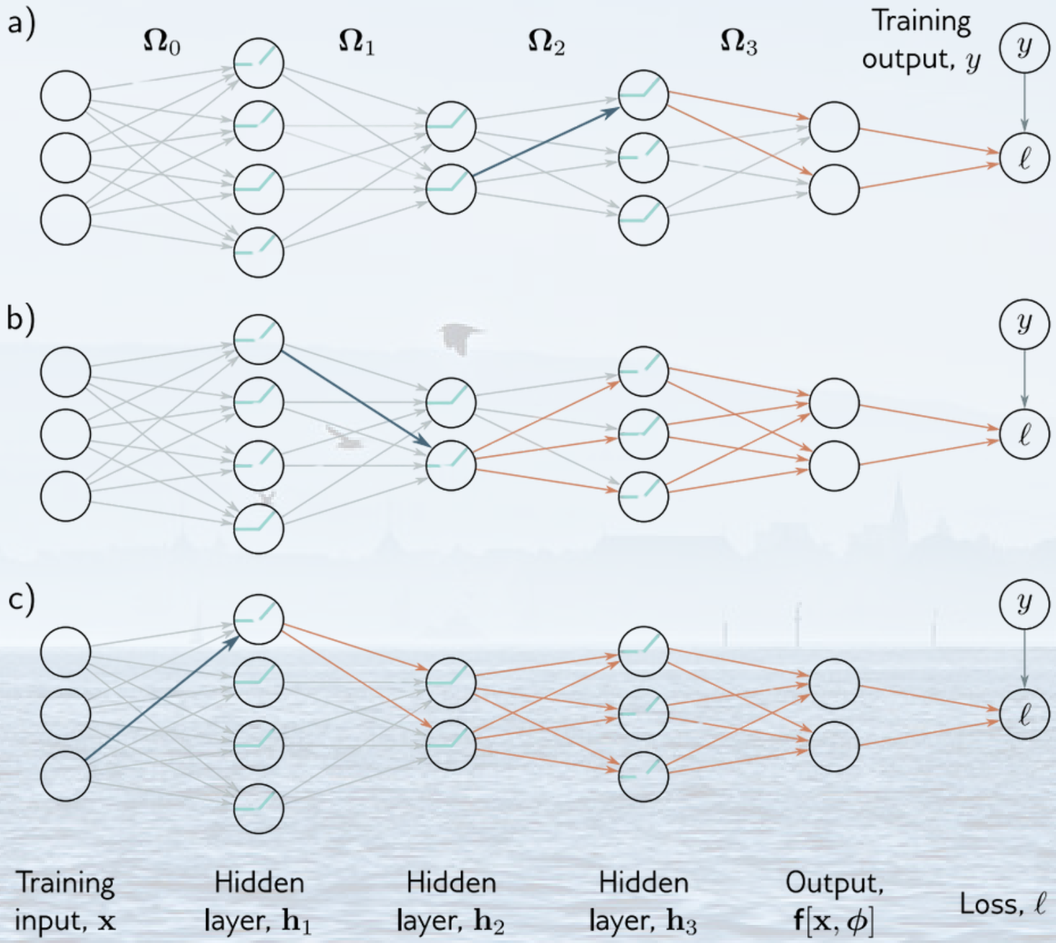
$$\nabla L(\mathbf{W}_t) = \left(\frac{dL}{d\Omega_{0,0}}, \frac{dL}{d\Omega_{0,1}} \dots \frac{dL}{d\beta_{0,0}}, \frac{dL}{d\beta_{0,1}} \dots . \right)_{\mathbf{W}_t}$$

- we use the gradient to update the weights

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla L(\mathbf{W}_t)$$

The process is repeated until the weights stop to change (the minimum is reached).

Recap: Backpropagation



In order to compute the gradient of the loss efficiently we use the Backpropagation Algorithm:

- complex name, simple concept: compute derivatives
- it relies on the chain rule of derivatives
- it consists of two steps:
  - a forward pass
  - a backward pass

Using the Backpropagation makes computing the gradients as time consuming as computing the Loss, but a lot of memory is required (~10x model size).

Universität Konstanz

Outline

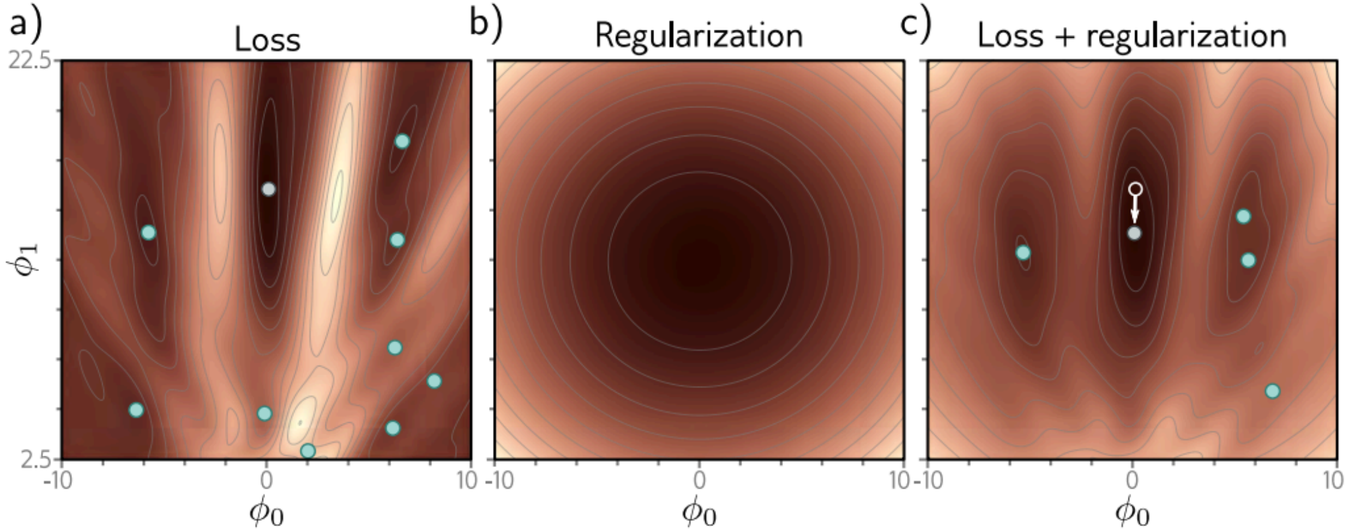
1. Regularizing Neural Networks
2. The Autoencoder
3. Coding Session

Universität
Konstanz

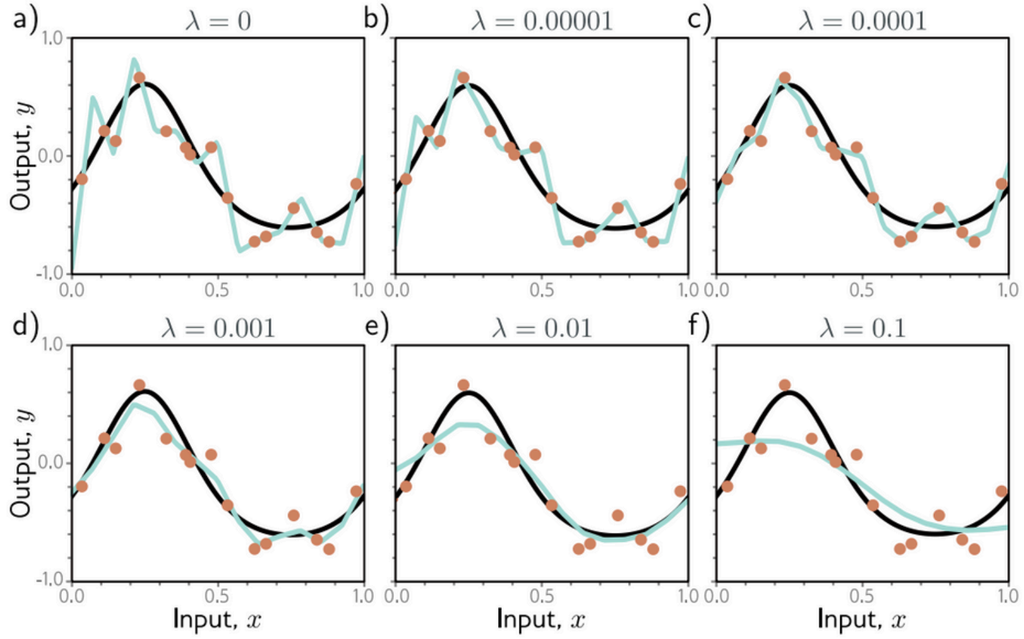
Regularizing Neural Networks

Regularization Techniques

Theoretically speaking a MLP with two hidden layers and many enough neurons can perform any arbitrary complex regression or classification task. In practice finding the best parameters achieving this is very hard. This is mainly due to the fact that the loss is a very irregular function with many local minima. Regularization techniques are used to make the loss more smooth and to improve the performances of DNN.



L2 Regularization



L2 (and other norm regularizations) work by adding a penalty term to the loss that avoids weights to be too much large

$$L' = L + \lambda \sum w_i^2$$

Benefits:

- Reduces Model Complexity
- Reduces Overfitting

Implementation:

- Modify the Loss
- Train and Test the Model as Usual

Dropout

Dropout prevents overfitting by randomly dropping units (neurons) during training.

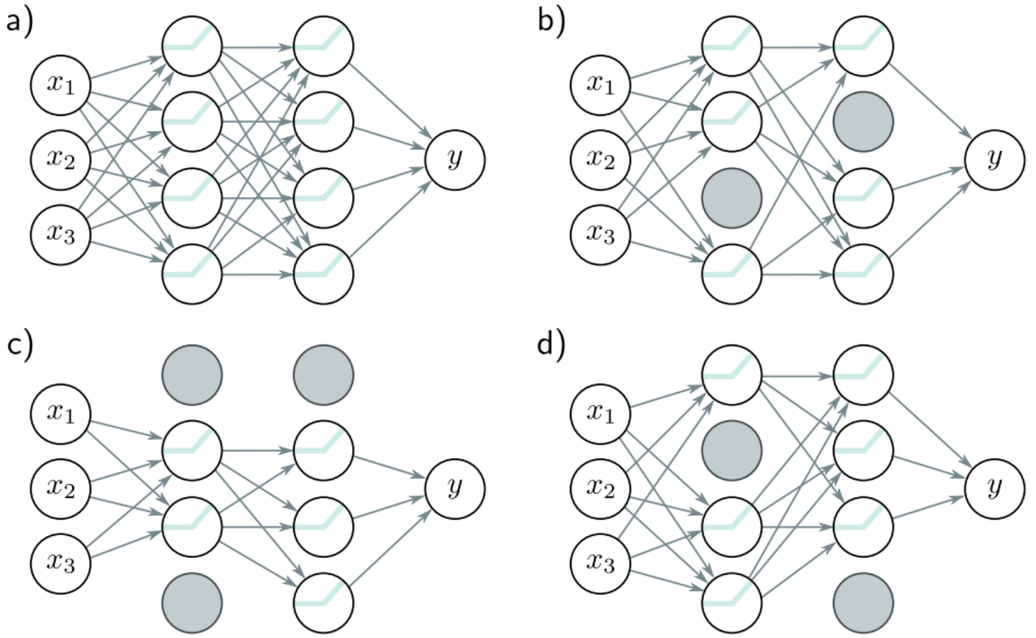
Benefits:

- Reduces Overfitting
- Increases Robustness

Implementation:

1. Apply Dropout:
2. Set Dropout Rate (0.1-0.5)
3. Training Phase Only

Conceptually, dropout effectively simulates training a large number of neural networks with different architectures in parallel.



Model Ensembling

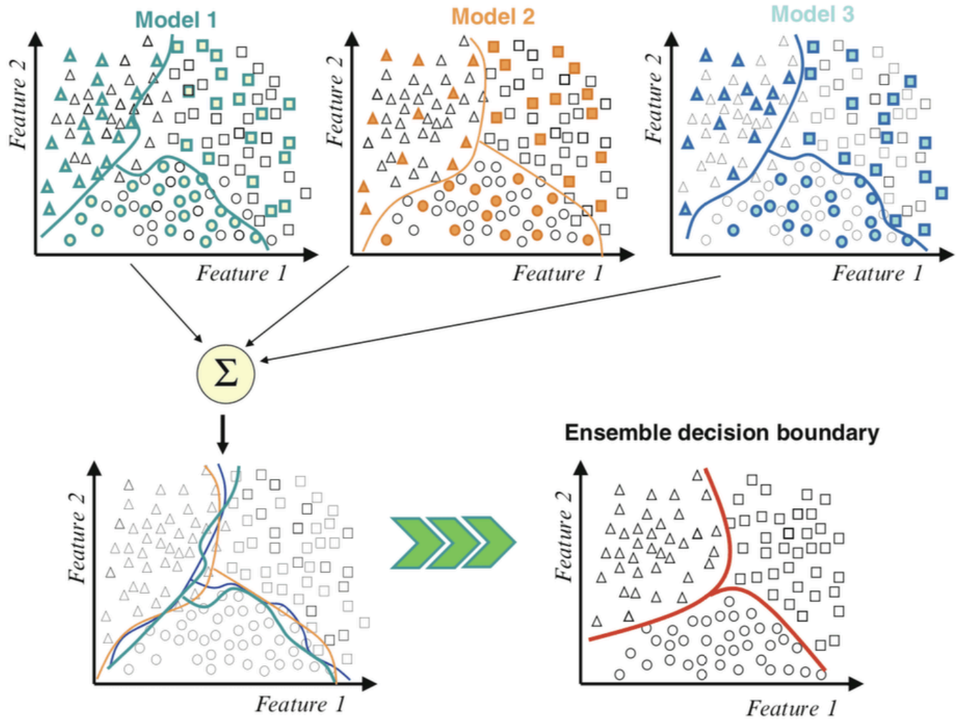
Model ensembling consists in independently training a group of different ML models on the same data. These models are then combined into a single more powerful model (random forest is one example).

Benefits:

- Reduces Overfitting
- Increases Robustness

Implementation:

1. Train Different Models
2. Combine Models
3. Test Combined Model



Early Stopping

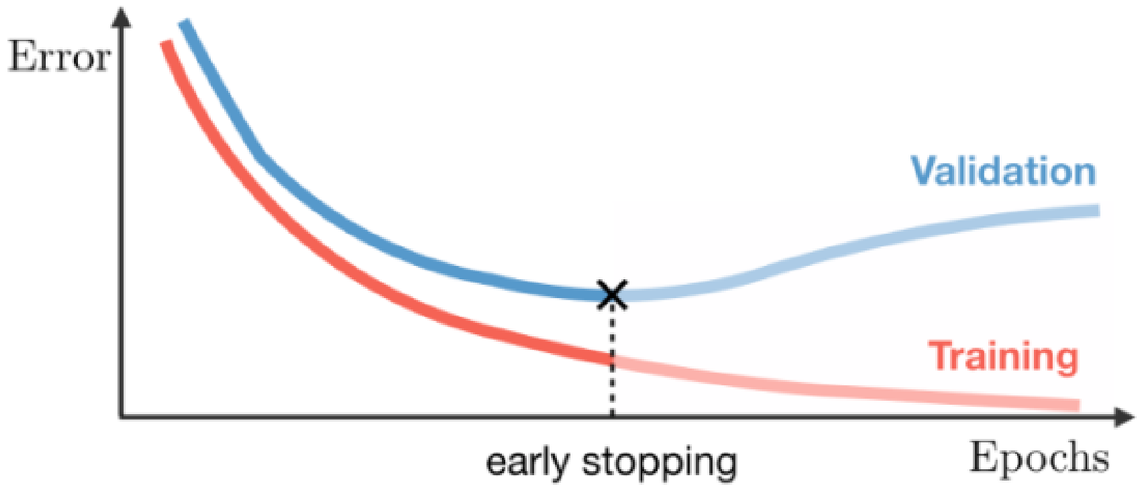
Early stopping prevents overfitting in training DNN. It works by monitoring the model's performance on a validation set during training and stopping the training process if the validation performance begins to degrade

Benefits:

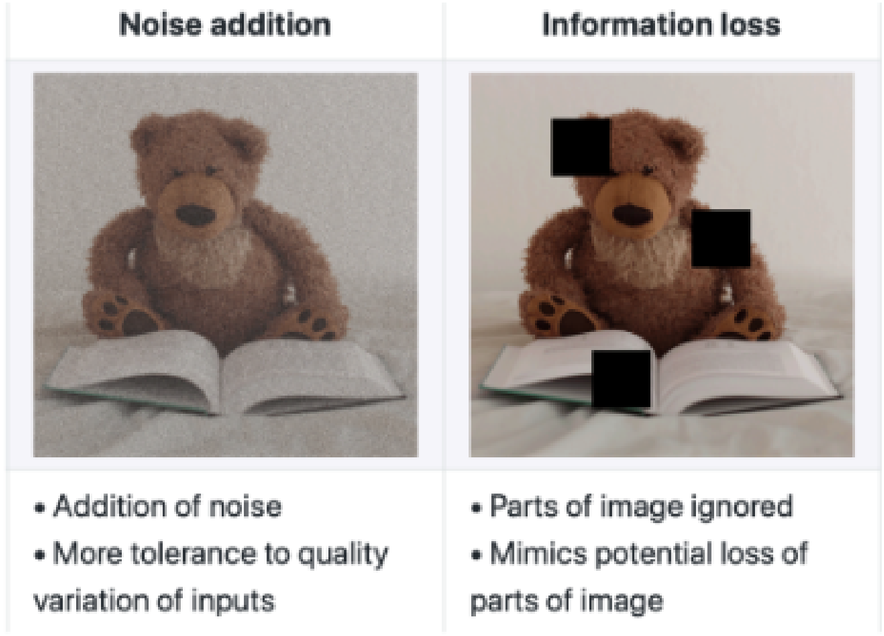
- Prevents Overfitting
- Optimizes Training Time

Implementation:

1. Monitor Performance
2. Patience Counter
3. Model Checkpointing



Noise Injection



Noise injection is a regularization technique that involves adding random noise to the inputs, weights, or activations of a neural network during training. This method helps improve the robustness and generalization of the model by forcing it to learn to cope with small, random changes in the training data.

Benefits:

- Reduces Overfitting
- Enhances Robustness

Implementation:

1. Choose Target for Noise
2. Set Noise Type and Level
3. Training Phase Only

Data Augmentation

Data augmentation is a technique used to increase the diversity and amount of training data by creating modified versions of existing data or artificially generating new data. Important in unbalanced datasets. Some simple techniques are Random Oversampling and SMOTE

Benefits:

- Makes Dataset Balanced
- Enhances Robustness

Implementation:

- Augment Data
- Train on Augmented Data
- Validate and Test on non-Augmented Data

Weights Initialization

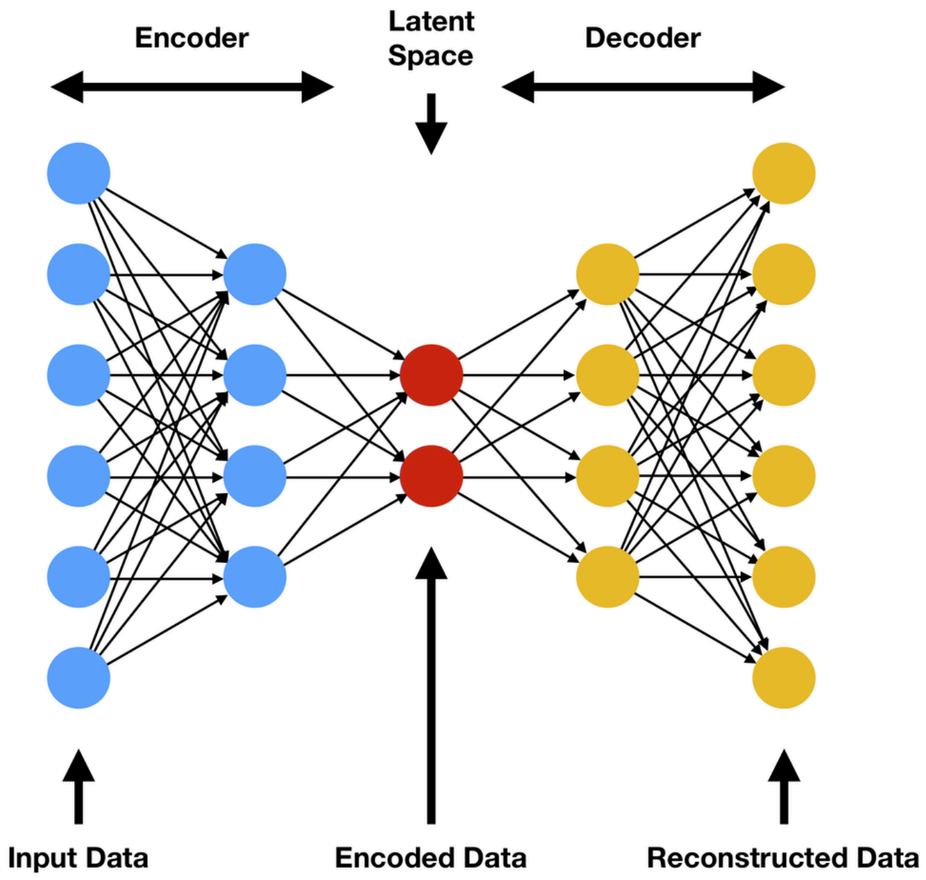
Proper weights initialization is crucial in deep neural networks as it sets the initial state from which training starts. Good initialization can help in breaking symmetry between neurons of the same layer, ensuring that each neuron learns a unique aspect of the data during training. Without this, neurons might end up learning the same features, stalling the learning process. There are several approaches:

- **Zero Initialization:** Generally avoided as it maintains symmetry, causing neurons to learn identical features.
- **Random Initialization:** Assigns weights with small random numbers. Common methods include using a Gaussian or uniform distribution, often scaled by the size of the previous layer (e.g., Xavier/Glorot or He initialization).
    - **Xavier/Glorot Initialization:** Ideal for layers with Sigmoid or Tanh activations.
    - **He Initialization:** Designed for ReLU activations, considering the variance of neurons in the network to maintain activation scales.

Universität
Konstanz

Autoencoder Architecture

The Autoencoder



The Autoencoder is one of the most important MLP architectures for unsupervised learning. It is composed of three sections:

- **Encoder** Encodes the data into a latent representation
- **Latent Space** Space where the encoded data live
- **Decoder** Convert back the data from the latent space to the standard representation

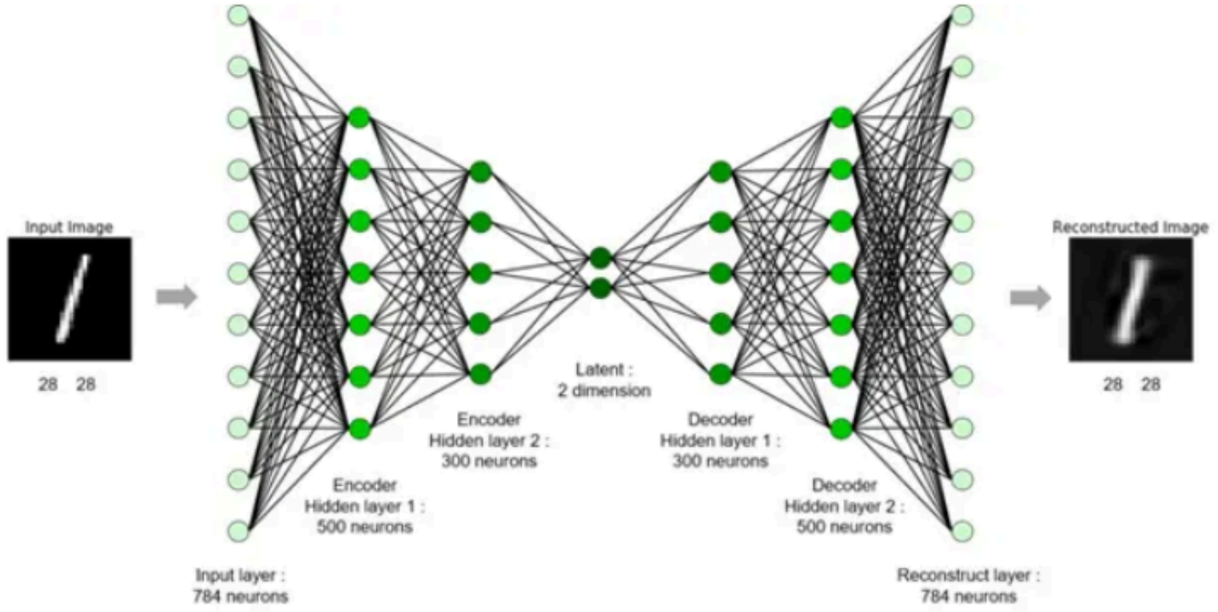
The Autoencoder can be used for several tasks

- Dimensionality Reduction
- Anomaly Detection
- Denoising

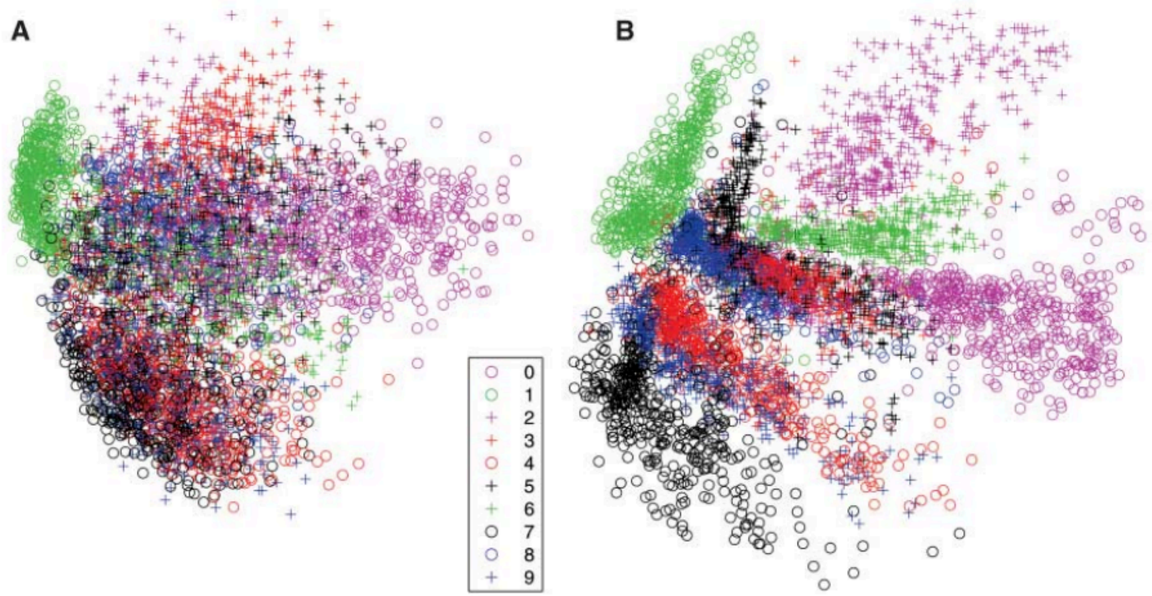
Unsupervised Learning

The Autoencoder is trained in an unsupervised way using a reconstruction loss

- it is trained to reconstruct in output exactly what it is given in input
- the loss quantifies how different is the output from the input
- practically speaking it is like a supervised regression, but there is no need of labelled data



Dimensionality Reduction



The Autoencoder architecture has a bottleneck, the latent space:

- if the reconstruction loss is low, the autoencoder can successfully reconstruct the input
- this means that the latent space contains enough information to reconstruct the input
- the latent space contains a low dimensional representation of the input data

Therefore we can train an autoencoder and then use it encoder to get a dimensionality reduction of the data (similar to PCA or T-SNE)

Anomaly Detection

Autoencoders can be used to perform anomaly detection

- the autoencoder will correctly reconstruct most of the data with low error
- outliers and anomalies, that are rarely (or never) seen in the training data will be harder to reconstruct

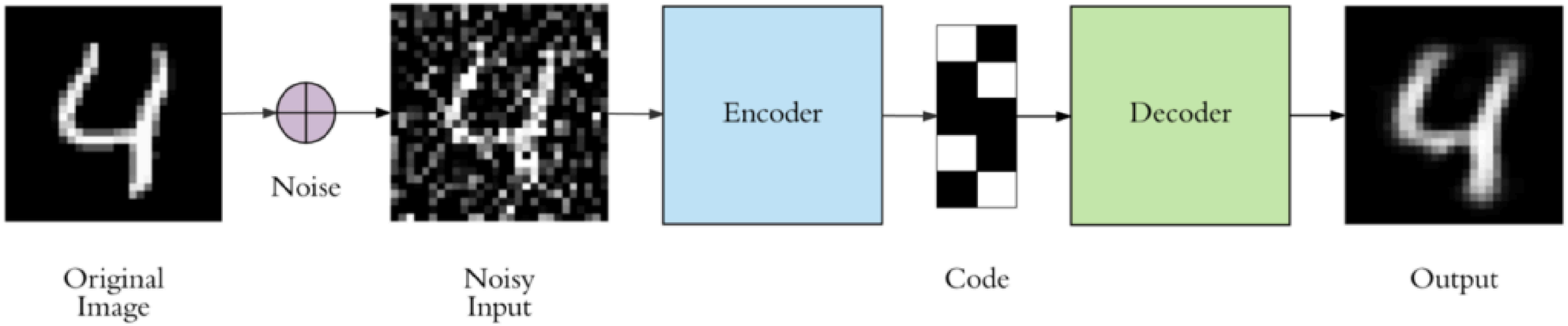
The idea is thus to determine if an event is or not an anomaly (e.g. credit card fraud) based on its reconstruction error. Note that no labels are needed (unsupervised anomaly detection).

Denoising

The Autoencoder can be used to remove noise from data

- training data are corrupted with some noise
- the Autoencoder is feed with the noisy data, but it is trained to output the original data without noise
- since noise is random and contains no information, the latent space does not learn it

As a consequence, when given in input noisy data or images, a denoising autoencoder can remove all noise returning a clean image. Again no supervised learning is required.



Summary

Improving Performances

- The loss function is typically very rough and full of local minima
- Several possible approaches to regularize the loss, they can be used alone or together
- They make finding the global minimum (or a good local minimum) easier

Autoencoder Architecture

- Unsupervised learning model trained to reconstruct the input as output
- Bottleneck structure to extract low-dimension latent representation of the input
- It can be used for
  - Dimensionality reduction
  - Anomaly detection
  - Denoising