UniTs - University of Trieste

Faculty of Scientific and Data Intensive Computing

Department of mathematics informatics and geosciences

Deep Learning

Lecturer:
Prof. Alessio Ansuini

Author:
Andrea Spinelli

March 27, 2025

github.com/Spina02 andreaspinelli2002@gmail.com

Preface

As a student of Scientific and Data Intensive Computing, I've created these notes while attending the **Deep Learning** course.

The prerequisites of the course are basic knowledge of:

- Linear Algebra (eigenvalue problems, SVD, etc.)
- Mathematical Analysis (multivariate differential calculus, etc.)
- Probability (chain rule, Bayes theorem, etc.)
- Machine Learning (logistic regression, PCA, etc.)
- Programming (Python, Linux Shell, etc.)

While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in Deep Learning.

Contents

1 Introduction

1.1 What is Deep Learning?

Definition: ***Deep Learning***

*"**Deep Learning** is constructing networks of parametrized functional modules and training them from examples using gradient-based optimization."*

~ Yann LeCun

In other words, Deep Learning is a collection of tools to build complex modular differentiable functions. These tools are devoid of meaning, it is pointless to discuss what DL can or cannot do. What gives meaning to it is how it is trained and how the data is fed to it.

Deep learning models usually have an architecture that is composed of multiple layers of functions. These functions are called **modules** or **layers**. Each layer is a function that takes an input and produces an output. The output of one layer is the input of the next layer. The output of the last layer is the output of the model.

Practical Applications of Deep Learning

Deep Learning has revolutionized numerous fields by achieving unprecedented performance on complex tasks. In **computer vision**, convolutional neural networks can recognize objects, detect faces, segment images, and even generate realistic images. **Protein structure prediction** has seen remarkable advances with models like AlphaFold, which can accurately predict 3D protein structures from amino acid sequences, fundamentally changing molecular biology and drug discovery. **Speech recognition and synthesis** systems powered by deep learning can transcribe spoken language with near-human accuracy and generate natural-sounding speech, enabling voice assistants and accessibility tools. Other applications include natural language processing (powering chatbots and translation systems), recommendation systems, anomaly detection in cybersecurity, weather forecasting, and autonomous driving. The versatility of deep learning comes from its ability to learn meaningful representations directly from data, reducing the need for manual feature engineering while achieving superior performance across diverse domains.

1.2 Family of linear functions

Let's consider a simple modell

$$y = \Phi_0 + \Phi_1 x$$

This model is a linear function of x with parameters Φ_0 and Φ_1. The model can be represented as a line in the $x - y$ plane. The parameters Φ_0 and Φ_1 determine the slope and the intercept of the line.

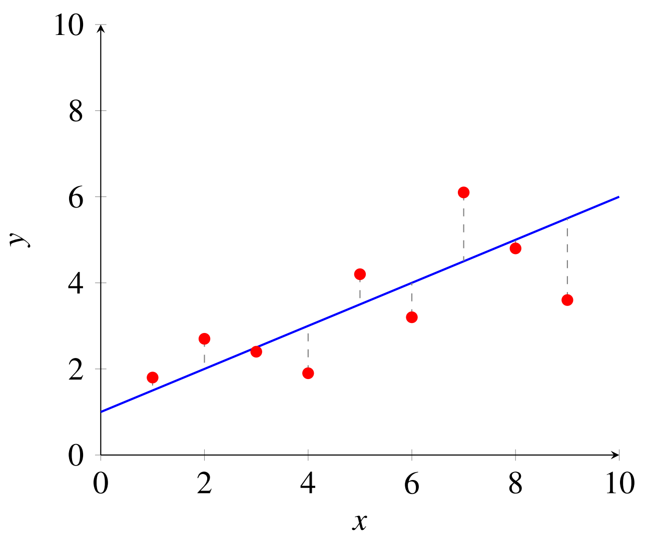


Figure 1.1: Linear model with scatter points and error segments

Loss Function

The loss function for this model is the mean squared error (MSE) between the predicted values and the actual values:

$$L = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\Phi_0 + \Phi_1 x_i}_{=P_i} - y_i)^2$$

where N is the number of data points, x_i is the i-th input, y_i is the i-th target, and P_i is the predicted value for the i-th data point.

Optimization

The goal of optimization is to find the values of Φ_0 and Φ_1 that minimize the loss function. This is done by computing the gradient of the loss function with respect to the parameters and updating the parameters in the opposite direction of the gradient. The update rule for the parameters is given by:

Let's calculate the gradient of the losso function:

$$\nabla_{\{\Phi\}} L = \left[\frac{\partial L}{\partial \Phi_0}, \frac{\partial L}{\partial \Phi_1}\right]$$

Then we can update the parameters as follows:

$$\begin{cases} \Phi_0 \leftarrow \Phi_0 - \lambda \dfrac{\partial L}{\partial \Phi_0} \\ \Phi_1 \leftarrow \Phi_1 - \lambda \dfrac{\partial L}{\partial \Phi_1} \end{cases} \quad \Rightarrow \quad \Phi^{new} = \Phi^{old} - \lambda \nabla_{\{\Phi\}} L$$

where λ is the **learning rate**, a hyperparameter that controls the size of the parameter updates.

This is only a step in the optimization process. The optimization algorithm iteratively updates the parameters until the loss converges to a minimum. But when shell we stop?

$$|L^{new} - L^{old}| < \varepsilon$$

where ε is a small positive number that determines the convergence threshold.

Exercises

Let's talk further about the loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N}(\underbrace{\Phi_0 + \Phi_1 x_i}_{= P_i} - y_i)^2$$

The aim is to minimize the loss function. In this case the loss is a paraboloid function of Φ_0 and Φ_1, so it has a single minimum.

Questions

1. Calculate the gradient of the Loss function
2. Find the minimum of the loss function
3. Show that the gradient of L is orthogonal to the level lines
4. Estimate the computational complexity of the exact solution of the linear regression problem in the general case (take into account the number of data samples and the number of dimensions/features used)

$$\Phi = (X^\top X)^{-1} X^\top y$$

Solutions

1. Calculate the gradient of the Loss function
2. Find the minimum of the loss function
3. Show that the gradient of L is orthogonal to the level lines
4. Estimate the computational complexity of the exact solution of the linear regression problem in the general case (take into account the number of data samples and the number of dimensions/features used)

$$\Phi = (X^\top X)^{-1} X^\top y$$

   Let's consider a $X_{N\times D}$ matrix and let's computate the complexity of the single operations:
   - $(x^\top X)$ is a $D \times D$ matrix, so the complexity is $O(ND^2)$
   - $(x^\top X)^{-1}$ requires $O(D^3)$ operations
   - $(x^\top y)$ is a $D \times 1$ matrix, so the complexity is $O(ND)$

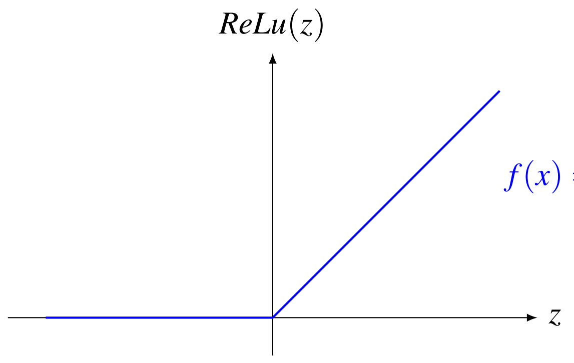
   The total complexity is $O(ND^2 + D^3 + \cancel{ND}) = O(ND^2 + D^3)$

2 Shallow Neural Networks

2.1 Activation Functions

2.1.1 Rectified Linear Unit (ReLU)

The ReLU activation function is defined as:



$$f(x) = \max(0, x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

Figure 2.1: Rectified Linear Unit (ReLU) activation function

2.1.2 Elements of the family F

$$y = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11} x] + \phi_2 a[\theta_{20} + \theta_{21} x] + \phi_3 a[\theta_{30} + \theta_{31} x]$$

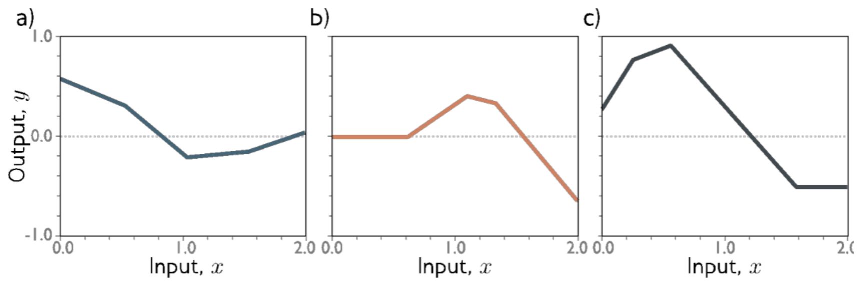


Figure 2.2: Family of functions F

$$\begin{cases} z_1 = \phi(a_1) = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11} x] \\ z_2 = \phi(a_2) = \phi_0 + \phi_2 a[\theta_{20} + \theta_{21} x] \\ z_3 = \phi(a_3) = \phi_0 + \phi_3 a[\theta_{30} + \theta_{31} x] \end{cases}$$

$$\begin{tikzcd}
& H_1 \arrow[dr] & \\
X \arrow[ur] \arrow[r] \arrow[dr] & H_2 \arrow[r] & Y \\
& H_3 \arrow[ur] &
\end{tikzcd}$$

$$y = W^{(2)}\phi(W^{(1)}x + b^{(1)}) + b^{(2)}$$

$$W^{*} = W^{(2)}W^{(1)} \qquad \text{rank}(W^{*}) = \min$$

…

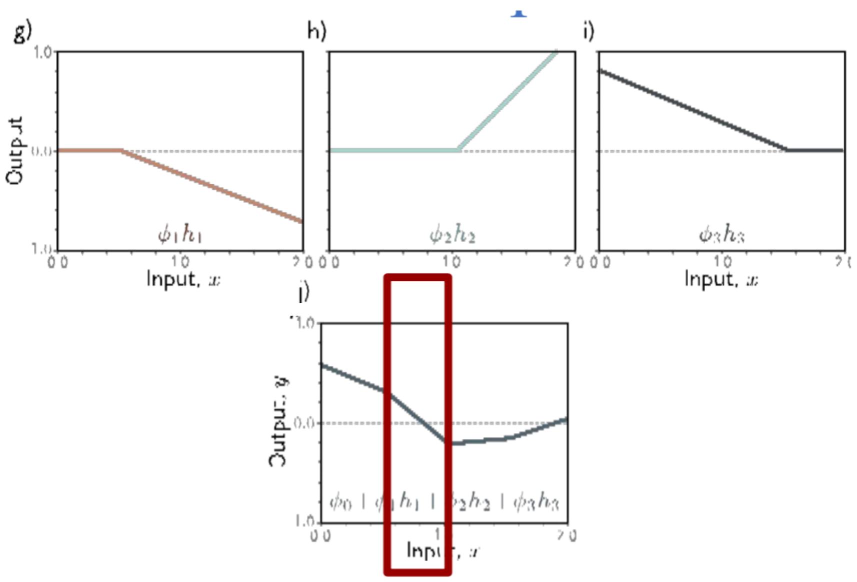


Figure 2.3: Flow computation

In the hilighted section, the contribute of the 2nd neuron of the hidden layer is zero, so the output is only influenced by the 1st and 3rd neurons of the hidden layer.

This is a consequence of the ReLU activation function, which deactivates the second node in that interval of the input space:

$$\begin{tikzcd}
& H_1 \arrow[dr] & \\
X \arrow[ur] \arrow[dr] & \xcancel{H_2} & Y \\
& H_3 \arrow[ur] &
\end{tikzcd}$$

Multivariate Output

For multivariate output, the situation is similar to the previous case. In this case, the output layer has two neurons, Y_1 and Y_2:

$$\begin{tikzcd}
 & H_1 \arrow[dr] \arrow[dddr] & \\
 & & Y_1 \\
X \arrow[uur] \arrow[r] \arrow[ddr] & H_2 \arrow[ur] \arrow[dr] & \\
 & & Y_2 \\
 & H_3 \arrow[uuur] \arrow[ur] &
\end{tikzcd}$$

We obtain the output as:

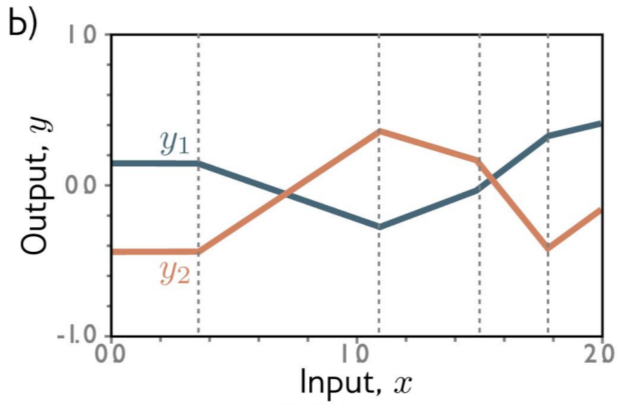
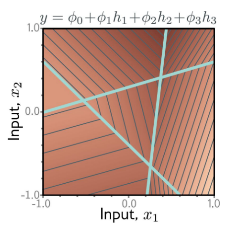
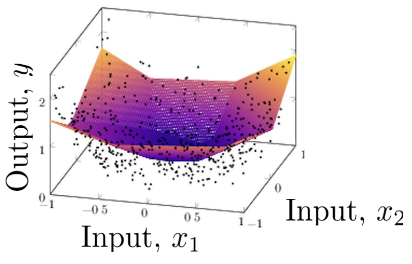


Figure 2.4: Flow computation for multivariate output

Multivariate Input

$$\begin{tikzcd}
 & H_1 \arrow[ddr] & \\
x_1 \arrow[ur] \arrow[dr] \arrow[dddr] & & \\
 & H_2 \arrow[r] & y \\
x_2 \arrow[uuur] \arrow[ur] \arrow[dr] & & \\
 & H_3 \arrow[uur] &
\end{tikzcd}$$



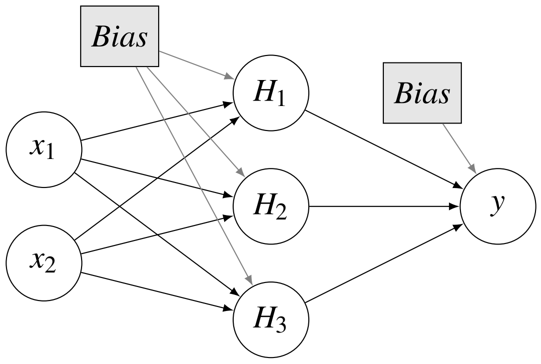


In this case we have 13 parameters:

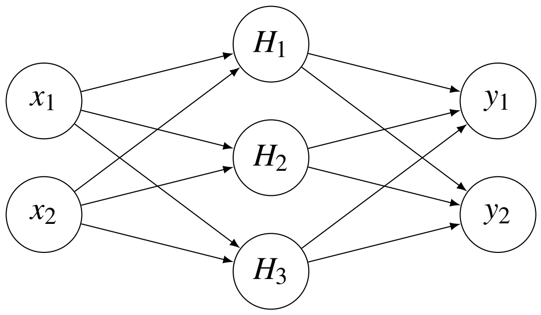
- We have 3 parameters from X_1 and X_2 to connect each node of the hidden layer (2 are the weights and 1 is the bias), for a total of 9 parameters.
- We have 4 parameters from the hidden layer to the output layer (3 weights and the bias term).

The general formula to count the number of parameters is:

#



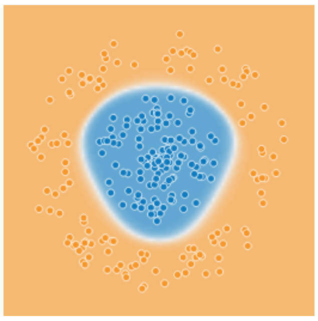
General Case



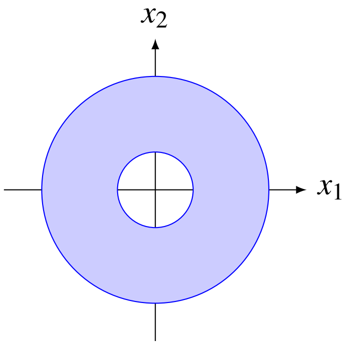
3 Lecture 18/03/2025

Polar Coordinates

Let's consider data disposed as in figure:

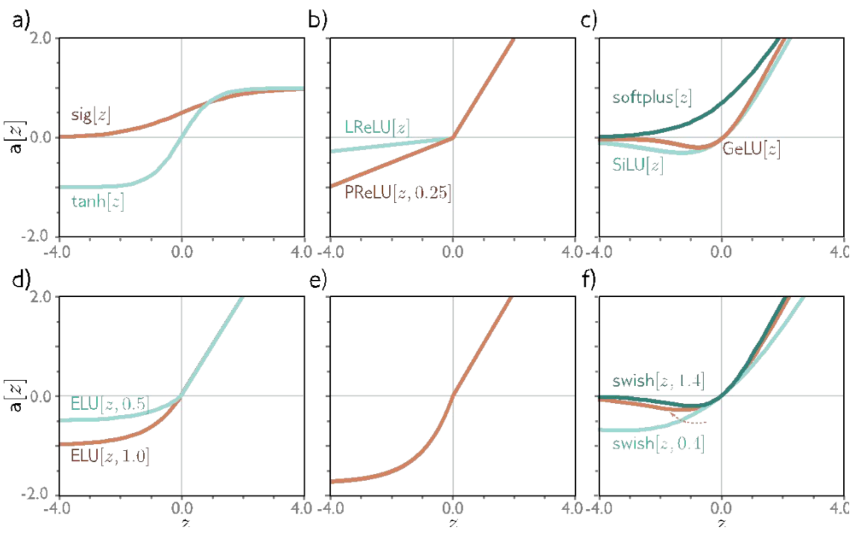


We can convert the data from Cartesian to Polar coordinates as follows:



$$\begin{cases} x_1 \\ x_2 \end{cases} \rightarrow \begin{cases} r = \sqrt{x_1^2 + x_2^2} \\ \theta = arctg\dfrac{x_2}{x_1} \end{cases}$$

Other Non-Linearities



3.1 Universal Approximation Theorem

Theorem 3.1. *Let σ be any continuous discriminatory function.*

$$\textit{Then finite sums of the form} \quad G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_i^\top x + \theta_j) \quad \textit{are dense in } C(I_n).$$

In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for wich:

$$|G(x) - f(x)| < \varepsilon \qquad \forall x \in I_n$$

Bounds of the Universal Approximation Theorem

Any function $f(x)$ on $\mathbb{R}^d$ with some smoothness conditions can be approximated by a single hidden layer sigmoidal neural network $f_n(x)$ with n hidden units such that:

$$\int_{B_r} (f(x) - f_n(x))^2 \mu(dx) \leq \frac{c}{n}$$

where μ is a probability measure on ball $B_r = \{x : |x| < r\}$, c is a constant independent of n and r.

Observation: ***Feature Learning Advantage***

A result of the Universal Approximation Theorem is that no linear combinain of n fixed basis functions yells integrated square error smaller than order:

$$\left(\frac{1}{n}\right)^{\frac{2}{d}}$$

3.2 Number of Linear Regions

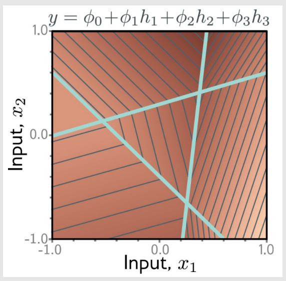
Be D_i the size of the input layer, D the size of the hidden layer, Then, given $D_i \leq D$.
The number of linear regions N is given by:

$$N \leq \sum_{j=0}^{D_i} \binom{D}{j}$$

Example:

Let's consider $D_i = 2$ and $D = 3$. The number of linear regions is given by:

$$N \leq \binom{3}{0} + \binom{3}{1} + \binom{3}{2} = 1 + 3 + 3 = 7$$



Fixed Functions

$$y = \sum_{w_i} \phi(W_{i_{D_i}} x_{D_i} + b_i) \sim \{B_i(x)\}$$

Polynomial case

If the dimension is $DIM = 1$ we have something of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M$$

If the dimension is $DIM = D$, we have:

$$y = w_o + \sum_{i=1}^{D} w_i x_i + \sum_{i,j=1}^{D} w_{ij} x_i x_j + \cdots + \sum_{i_1,\ldots,i_D=1}^{D} w_{i_1,\ldots,i_D} x_{i_1} \ldots x_{i_D}$$

The consequence is that the number of parameters grows exponentially with the dimension.

Curse of dimensionality

$$B_i(x) = \begin{cases} 0 & \text{if } x \text{ does not face in the block } b_i \\ majority & \text{if the class of } x \text{ in the block } A \end{cases}$$

Limitations of Fixed Basis Functions

High dimensionality introduces several challenges. One intuitive explanation comes from examining the probability density function in high-dimensional spaces.

Consider the marginal density of the radial coordinate:

$$p(r) = \int_{d\Omega} d\Omega \int p(x)\, dx,$$

where $d\Omega$ denotes the differential solid angle.

By switching to polar coordinates, we have:

$$p(x) \to p(r, \Omega),$$

so that the radial density becomes:

$$p(r) = \frac{S_D\, r^{D-1}}{(2\pi\sigma^2)^{D/2}} e^{-r^2/(2\sigma^2)},$$

with S_D representing the surface area of the unit sphere in D dimensions.

A key insight is that the probability mass concentrates around a specific radius. In fact, the mode of the radial density occurs approximately at:

$$r^* \approx \sigma\sqrt{D}.$$

Moreover, for a small deviation ε from r^*, the density decays as:

$$p(r^* + \varepsilon) = p(r^*) \exp\left(-\frac{3\varepsilon^2}{2\sigma^2}\right).$$

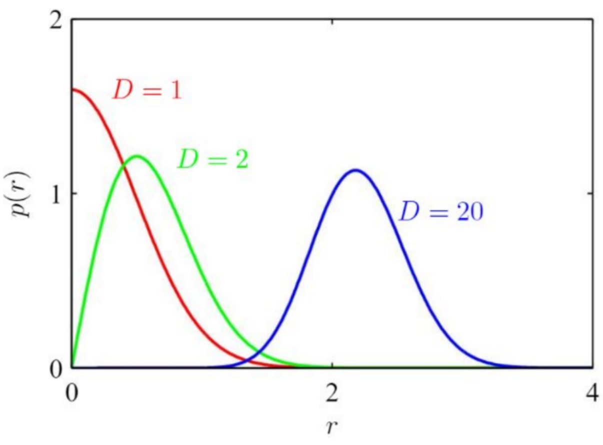


Figure 3.1: Curse of dimensionality.

Mainfolds hypothesis

Most ”naturally occuring” datasets lie on a low-dimensional mainfold

4 Deep Networks

4.1 Composition of shallow networks

The composition of shallow networks is a way to increase the number of linear regions. The idea is to stack multiple shallow networks to create a deep network.

Let’s consider firstly a simple example with two shallow networks:

$$\begin{tikzcd}
 & h_1 \arrow[dr] & & h'_1 \arrow[dr] & \\
X \arrow[ur] \arrow[r] \arrow[dr] & h_2 \arrow[r] & y \arrow[ur] \arrow[r] \arrow[dr] & h'_2 \arrow[r] & y' \\
 & h_3 \arrow[ur] & & h'_3 \arrow[ur] &
\end{tikzcd}$$

In this case we can distinguish how the two network behaves:

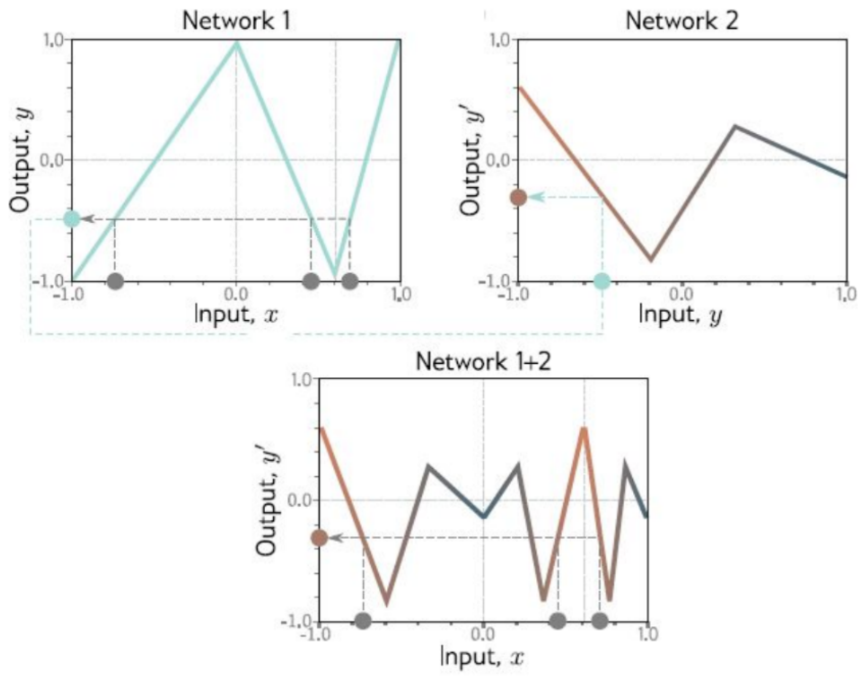
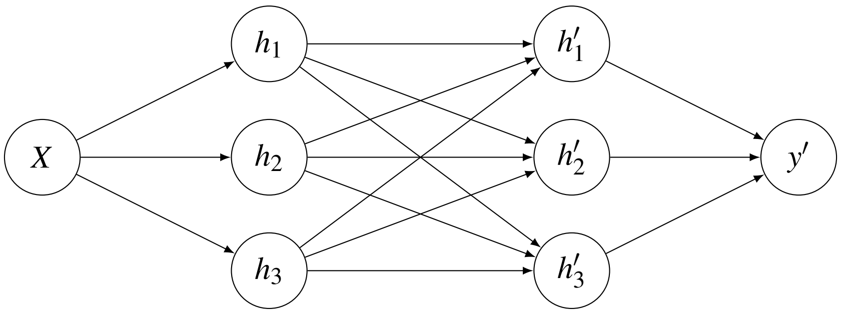


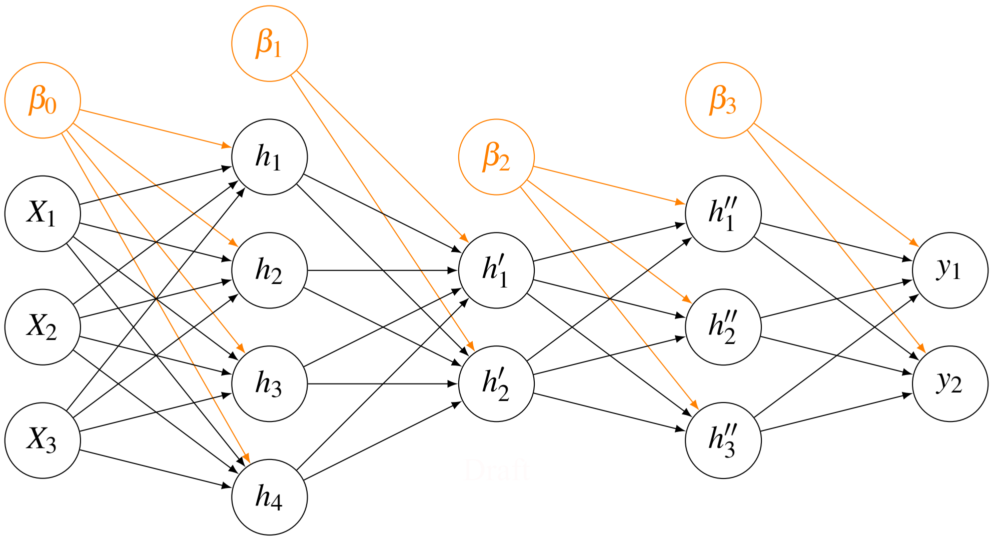
Figure 4.1: Composition of shallow networks.

We can reinterpret the composition of shallow networks as 2-layer a deep network:



4.2 Generic MLP

The generic Multi-Layer Perceptron (MLP) is a deep network with multiple hidden layers.



5 Loss Functions

Mean Squared Error (MSE)

Mean Squared Error is the most common loss function used for regression problems. MSE is the sum of squared distances between our target variable and predicted values.

$$L = \frac{1}{N}\sum_{i}^{N}\left(y_i - f(c^i;\{\phi\})\right)^2$$

Maximum Likelihood Estimation (MLE) is equivalent to minimizing the MSE loss function.

—

Recipe for loss construction:

1. Choose a suitable probability distribution $\Pr(y|\theta)$ that is defined over the domain of the predictions y and has distribution parameters θ
2. Set the mahchine learning model $f[x,\phi]$ to predict one or more of these parameters so $\theta = f[x,\phi]$ and $\Pr(y|\theta) = \Pr(y|f[x,\phi])$.
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood over the training dataset pairs:

$$\hat{\phi} = \arg\min_{\phi}[L[\phi]] = \arg\min_{\phi}\left[-\sum_{i=1}^{N}\log\left[\Pr(y_i|f[x_i,\phi])\right]\right]$$

This is equivalent to maximise the probability of the observed data under the model:

$$\Pr(\{x_i,y_i\}) = \prod_{i=1}^{N}\Pr(y_i|f[x_i;\{\phi\}])$$

...

Binary classification Problem

$$\begin{cases}\lambda = \Pr(y=1|x)\\ 1-\lambda = \Pr(y=0|x)\end{cases} \quad\Leftrightarrow\quad \Pr(y|x) = \lambda^y(1-\lambda)^{1-y}$$

$$\min_{\lambda} -\sum_{i}^{N} y^i\log\lambda + (1-y^i)\log(1-\lambda)$$

suppose that you want to classify a $\mathbb{R}^d$ dimentional data

$$\begin{array}{c} x^{(1)}, y^{(1)} \\ x^{(2)}, y^{(2)} \\ \vdots \\ x^{(N)}, y^{(N)} \end{array}$$

where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{0,1\}$

In this case we cannot use a ReLu activation function in the output layer, because the output of the ReLu function is not bounded between 0 and 1. We need to use a ***sigmoid activation*** function in the output layer.

This is for a binary classification problem.

Multi-class classification Problem

For a multi-class classification problem we want to have multiple choices for the output layer.

Let's consider k parameters for the output layer. The values returned by the output layer are not probabilities, but scores, usually called ***logits***.

In this case we need to use a ***softmax activation*** function in the output layer.

$$\begin{array}{ccc} z_o & \rightarrow & e^{z_o} / \sum_i^k e^i \\ z_1 & \rightarrow & e^{z_1} / \sum_i^k e^i \\ & \vdots & \\ z_k & \rightarrow & e^{z_k} / \sum_i^k e^i \end{array}$$

The softmax function is a generalization of the sigmoid function and is used in the output layer of a neural network when we are dealing with a multi-class classification problem.

Information Theoretical Perspective on Loss Functions

We can calculate the empirical probability distribution as follows:

$$\Pr_{emp}(\{x\}) = \sum_{i=1}^{N} \delta(x - x_i)$$

where δ is the ***Dirac delta*** function.

…

We can calculate the "distance" between two probability distributions using the ***Kullback-Leibler divergence***:

$$KL[q][p] = \int_{-\infty}^{\infty} q(z) \log[q(z)]dz - \int_{-\infty}^{\infty} q(z) \log[p(z)]dz$$

This is not symmetric $KL[q][p] \neq KL[p][q]$ but it is always positive $KL[q][p] \geq 0$.

$$\min KL(q|p(\theta)) \qquad \Leftrightarrow \qquad MLE$$

$$KL(q|p(\theta)) = -\int \frac{1}{N}\sum_{i=1}^{N}\delta(x-x_i)\log p(x;\theta)dx + \underbrace{\cancel{\int \frac{1}{N}\sum_{i=1}^{N}\delta(x-x_i)\log\frac{1}{N}\sum_{i=1}^{N}\delta(x-x_i)dx}}_{\text{not dependent on }\theta}$$

So we can minimize the KL divergence by minimizing the negative log-likelihood.

—

We said that the KL divergence is not symmetric:

$$\min KL(p|q) = \min\sum_{i=1}^{N} p_i \log\frac{p_i}{q_i}$$

We will always have a distance between de empirical distribution and the model distribution:

$$KL(p|q) > 0 \qquad = \text{"wrong code"}$$

This is becouse We have limited knowledge of the source, and this reflects in a non optimal compression of the data. The KL divergence is the difference between the optimal compression and the compression we are able to achieve.

5.1 Training

5.1.1 Stochastic Gradient Descent

The ***Stochastic Gradient Descent*** (SGD) is a method to minimize a function by iteratively moving in the direction of the negative gradient of the function at each point.

...

Gabor model

The Gabor model is a linear model that uses a set of Gabor filters to extract features from an image. The Gabor filters are a set of sinusoidal functions that are modulated by a Gaussian function.

$$f[x,\phi] = \sin\left[\phi_0 + \sum_{i=1}^{N}\phi_i x_i\right] e^{-\sum_{i=1}^{N}\phi_i^2 x_i^2}$$

Gradient descent gets to the global minimum if we start in the right "valley", otherwise it will get stuck in a local minimum or near a saddle point.

The idea is to add noise to the gradient to escape from the local minimum:

- Compute the gradient based on only a subset of the points: a ***mini-batch***.
- Work through datsset sampling without replacement.
- One pass through the data is called an ***epoch***.

In this way we calculate the gradient of the loss function for a mini-batch, which will likely be different from the gradient of the whole dataset. The hope is that the noise in the gradient will help us escape from the local minimum.

Property of SGD

- Can escape from local minima
- Adds noise, but still sensible updates as based on part of data
- Uses all data equally
- Less computationally expensive
- Seems to find better solutions

- Doesn’t convere in traditional sense
- Learning rate schedule: learning rate decreases over time

Momentum