Multi-Layer Perceptrons

Johanni Brea

Introduction à l'apprentissage automatique

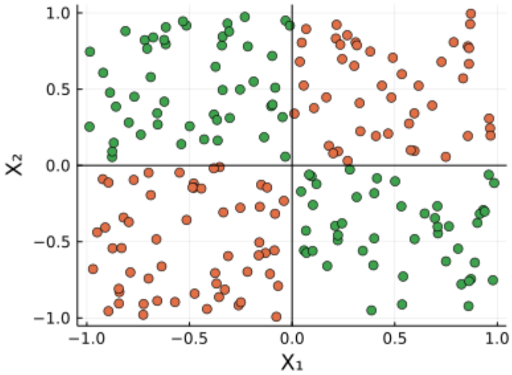
GYMINF 2021

Table of Contents

1. Solving the XOR Problem Without Feature Engineering
2. Artificial Neurons
3. Multilayer Perceptrons
4. Regression with Multilayer Perceptrons
5. Classification with Multilayer Perceptrons

Recap: Vector-Features

XOR-Problem
Training Data

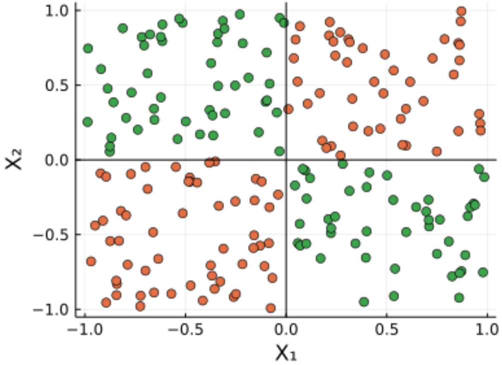


Logistic Regression fails:

There is no linear decision boundary.

Recap: Vector-Features

Project data to a higher dimensional space by computing the scalar products between feature vectors $w_1, \dots, w_q$ and input vectors x_i and thresholding.



For example $h_{21} = \max(0, w_1^T x_2)$.

Logistic Regression on the features works.

Solving the XOR Problem without Feature Engineering

Logistic Regression: $P(Y=1|x,\beta) = \sigma(\beta_0 + \beta_1 x_1 + \cdots \beta_p x_p)$

Solving the XOR Problem without Feature Engineering

Logistic Regression: $P(Y=1|x,\beta) = \sigma(\beta_0 + \beta_1 x_1 + \cdots \beta_p x_p)$

Logistic Regression on features:

$$P(Y=1|x,\beta) = \sigma\Big(\beta_0 + \beta_1 \underbrace{g(w_1^T x)}_{h_1} + \cdots \beta_q \underbrace{g(w_q^T x)}_{h_q}\Big)$$

with hand-picked feature vectors $w_1, \ldots, w_q$ and activation function

$$g(x) = \mathrm{relu}(x) = \max(0, x).$$

Solving the XOR Problem without Feature Engineering

Logistic Regression: $P(Y=1|x,\beta) = \sigma(\beta_0 + \beta_1 x_1 + \cdots \beta_p x_p)$

Logistic Regression on features:

$$P(Y=1|x,\beta) = \sigma\Big(\beta_0 + \beta_1 \underbrace{g(w_1^T x)}_{h_1} + \cdots \beta_q \underbrace{g(w_q^T x)}_{h_q}\Big)$$

with hand-picked feature vectors $w_1, \ldots, w_q$ and activation function

$$g(x) = \mathrm{relu}(x) = \max(0, x).$$

Idea

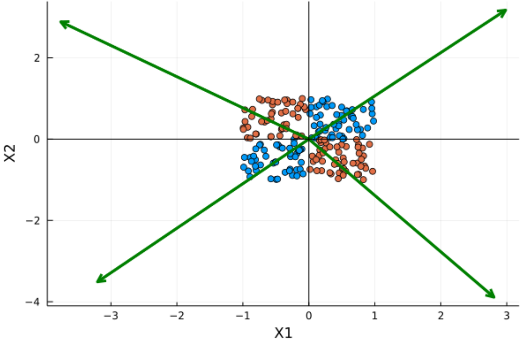
Why don't we learn the features with gradient descent?

$$P(Y=1|x,\beta,w_1,\ldots,w_q) = \sigma\big(\beta_0 + \beta_1 g(w_1^T x) + \cdots \beta_q g(w_q^T x)\big)$$

$$\Rightarrow \quad \hat{\beta}, \hat{w} = \arg\min_{\beta, w_1, \ldots, w_q} \sum_{i=1}^{n} \log P(y_i|x_i, \beta, w_1, \ldots, w_q)$$

Solving the XOR Problem without Feature Engineering

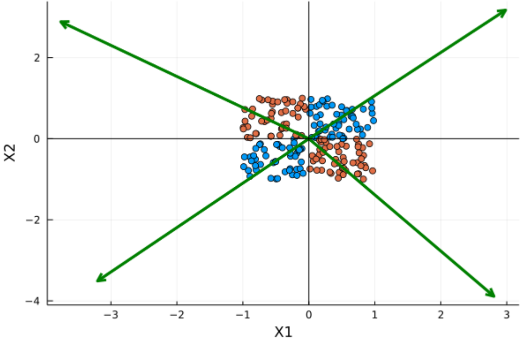
It also works with learned features



- We just fitted our first neural network ☺.

Solving the XOR Problem without Feature Engineering

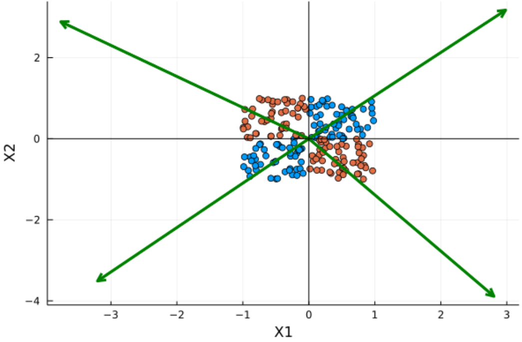
It also works with learned features



- We just fitted our first neural network ☺.
- The loss function has local minima; gradient descent does not find for all initial guesses a good solution.

Solving the XOR Problem without Feature Engineering

It also works with learned features



- We just fitted our first neural network ☺.
- The loss function has local minima; gradient descent does not find for all initial guesses a good solution.
- With more than 4 feature vectors, gradient descent finds good solutions for most initial guesses.

Table of Contents

1. Solving the XOR Problem Without Feature Engineering

2. Artificial Neurons

3. Multilayer Perceptrons

4. Regression with Multilayer Perceptrons

5. Classification with Multilayer Perceptrons

Artificial Neurons

Artificial neurons take a d-dimensional input $x = (x_1, \ldots, x_d)^T$ and output a scalar

$$a = g(w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d)$$

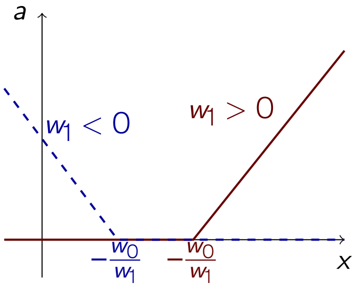
with **parameters (or weights)** $w_0, w_1, \ldots, w_d$ and **activation function** g.
w_0 is also called **bias** (instead of intercept).

Popular Activation Functions

rectified linear unit	$\mathrm{relu}(x) = \max(0, x) = \begin{cases} x & x \geq 0, \\ 0 & x < 0 \end{cases}$
scaled exponential linear unit	$\mathrm{selu}(x) = \lambda \begin{cases} x & x \geq 0, \\ \alpha e^x - \alpha & x < 0 \end{cases}$
sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
tangent hyperbolic	$\tanh(x)$
heaviside (perceptron)	$H(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0 \end{cases}$

Artificial relu-Neurons

$$a = \mathrm{relu}(w_0 + w_1 x)$$



$$a = \mathrm{relu}(w_0 + w_1 x_1 + w_2 x_2)$$

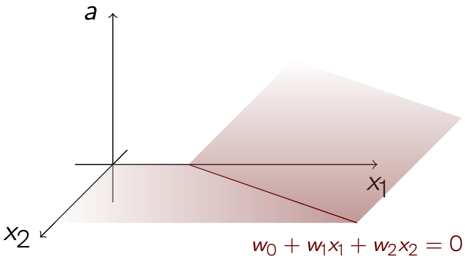


Table of Contents

1. Solving the XOR Problem Without Feature Engineering

2. Artificial Neurons

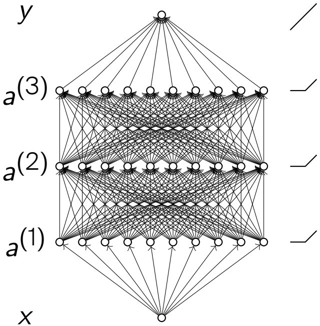
3. Multilayer Perceptrons

4. Regression with Multilayer Perceptrons

5. Classification with Multilayer Perceptrons

Multilayer Perceptrons

Multilayer Perceptrons (MLP) consist of multiple neurons organized in layers $1, 2, \ldots, L$.
Each **layer** has $d^{(l)}$ neurons and activation $g^{(l)}$.



one **input neuron** x
one linear **output neuron** y
3 **hidden layers** of 10 **relu-neurons**

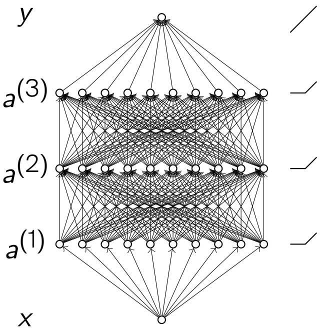
Multilayer Perceptrons

Multilayer Perceptrons (MLP) consist of multiple neurons organized in layers $1, 2, \ldots, L$.
Each **layer** has $d^{(l)}$ neurons and activation $g^{(l)}$.

output $a_k^{(l)}$ of k-th neuron in l-th layer

$$a_k^{(l)} = g^{(l)}\left(w_{k0}^{(l)} + w_{k1}^{(l)} a_1^{(l-1)} + \cdots + w_{kd^{(l-1)}}^{(l)} a_{d^{(l-1)}}^{(l-1)}\right)$$

input layer $a_k^{(0)} = x_k$.



one **input neuron** x
one linear **output neuron** y
3 **hidden layers** of 10 **relu-neurons**

Multilayer Perceptrons

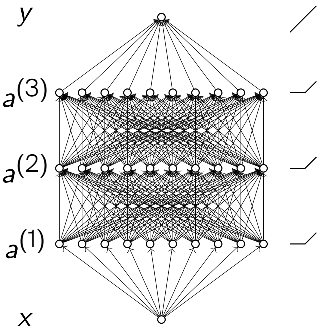
Multilayer Perceptrons (MLP) consist of multiple neurons organized in layers $1, 2, \ldots, L$.
Each **layer** has $d^{(l)}$ neurons and activation $g^{(l)}$.

output $a_k^{(l)}$ of k-th neuron in l-th layer

$$a_k^{(l)} = g^{(l)}\left(w_{k0}^{(l)} + w_{k1}^{(l)} a_1^{(l-1)} + \cdots + w_{kd^{(l-1)}}^{(l)} a_{d^{(l-1)}}^{(l-1)}\right)$$

input layer $a_k^{(0)} = x_k$.

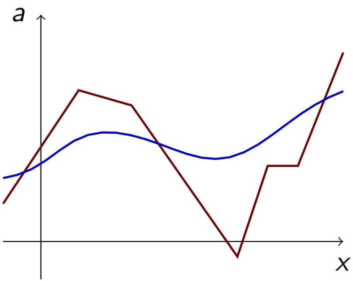
matrix notation $a^{(l)} = g^{(l)}\left(w^{(l)} a^{(l-1)} + b^{(l)}\right)$
with $b_k^{(l)} = w_{k0}^{(l)}$.



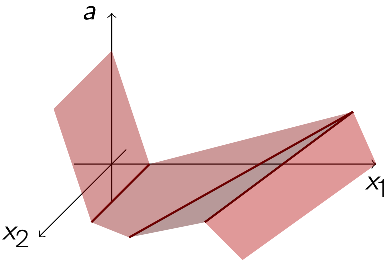
one **input neuron** x
one linear **output neuron** y
3 **hidden layers** of 10 **relu-neurons**

Multilayer Perceptrons

relu-neurons tanh-neurons

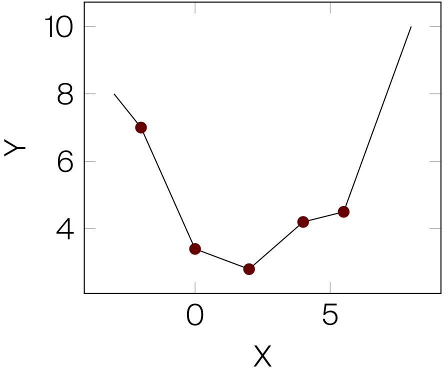


2D input, relu-neurons



Depth versus Width

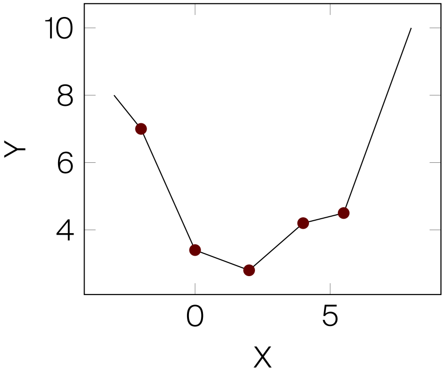
1 hidden layer with 5 neurons



$10 + 6 = 16$ parameters

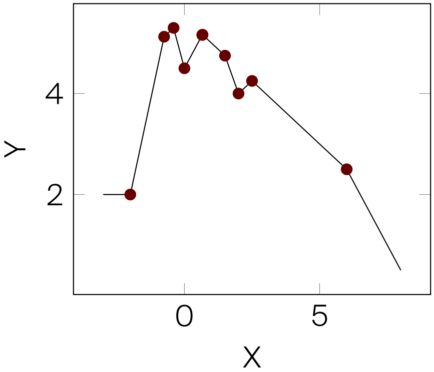
Depth versus Width

1 hidden layer with 5 neurons



$10 + 6 = 16$ parameters

two hidden layers with 3 and 2 neurons



$6 + 8 + 3 = 17$ parameters

Table of Contents

1. Solving the XOR Problem Without Feature Engineering
2. Artificial Neurons
3. Multilayer Perceptrons
4. **Regression with Multilayer Perceptrons**
5. Classification with Multilayer Perceptrons

Regression with Multilayer Perceptrons

The output of the neural network is used to parametrize the conditional density. The parameters are fitted with gradient descent on the negative log-loglikelihood loss.

For example: Assume the wind speed in Luzern is distributed normally around some mean that correlates with the measurements done 5 hours earlier.

We take a neural network with as many input neurons as measurements, some hidden neurons and one output neuron. Gradient descent finds the parameters such that the output activity approaches the mean of the conditional density.

$$p(y|x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2}\left(y - g^{(2)}(b^{(2)} + w^{(2)} g^{(1)}(b^{(1)} + w^{(1)} x))\right)^2\right)$$

Regression with Multilayer Perceptrons

A neural network with more outputs can be used to predict more complex densities.

Example: Assume the wind speed in Luzern is log-normally distributed with mean and standard deviation correlating with the pressure in Luzern.

We take a neural network with one input neuron, some hidden neurons and two output neurons. Gradient descent finds the parameters such that the output neurons code for the mean and the variance[1] of the log-normal density.

$$p(y|x) = \frac{1}{\sqrt{2\pi f\left(a_2^{(2)}\right)}} \exp\left(-\frac{1}{2f\left(a_2^{(2)}\right)}\left(\log(y) - a_1^{(2)}\right)^2\right)$$

[1] The output of the second neuron is additionally transformed with function f to be positive.

Initialization and Data Preprocessing

The initialization of neural networks and the choice the hyper-parameters (number of hidden neurons, non-linearities, learning rate, etc.) is an art.

Common choices are:

biases = 0

weights $w_{ij}^{(l)}$ sampled uniformly from $[-x, x]$ with $x = \sqrt{\frac{6}{d^{(l-1)} + d^{(l)}}}$

learning rate between 10^{-4} and 10^{-1}.

These choices work typically well for input data between 0 and 1 or standardized input with each predictor having mean 0 and variance 1.

For regression it is advisable to also scale or standardize the output.

Regularization

For the weights and biases in each layer one can apply L1 or L2 regularization.
Early stopping in gradient descent can be used.
Using fewer hidden neurons also reduces the flexibility of the neural network.

Another popular and effective regularization method is **Dropout**:
During training, for each training example a randomly selected fraction of p neurons is dropped out (inactivated).

This prevents neurons from becoming over-specialized.
All neurons are active when testing, but their weights are scaled by $1 - p$.

Flexibility of a Neural Network and Its Number of Parameters

More layers or more neurons $\Rightarrow$ more parameters.

More parameters $\Rightarrow$ more flexibility?

Flexibility of a Neural Network and Its Number of Parameters

More layers or more neurons $\Rightarrow$ more parameters.

More parameters $\Rightarrow$ more flexibility?

Not necessarily! Regularization has a strong effect on the flexibility.
Even without explicit regularization (L1, L2, Dropout) and without explicitly monitored early stopping one does stop gradient descent usually after some number of iterations and before perfect convergence; therefore one regularizes by implicit early stopping.

Flexibility of a Neural Network and Its Number of Parameters

More layers or more neurons $\Rightarrow$ more parameters.

More parameters $\Rightarrow$ more flexibility?

Not necessarily! Regularization has a strong effect on the flexibility. Even without explicit regularization (L1, L2, Dropout) and without explicitly monitored early stopping one does stop gradient descent usually after some number of iterations and before perfect convergence; therefore one regularizes by implicit early stopping.

Wisely regularized large neural networks often work better than small ones, because they tend not to get stuck at sub-optimal losses.

Why Multilayer Perceptrons?

Flexibility by Composition of Simple Elements.

- Individual neurons should not be simpler: the composition of linear functions is a linear function.
- Individual neurons do not need to be more complex: complexity is achieved by using multiple neurons.
- With sufficiently many neurons one can approximate any function.

Table of Contents

1. Solving the XOR Problem Without Feature Engineering

2. Artificial Neurons

3. Multilayer Perceptrons

4. Regression with Multilayer Perceptrons

5. Classification with Multilayer Perceptrons

Classification with Multilayer Perceptrons

With K output neurons we can use neural networks to parametrize categorical distributions suitable for classification problems.

Example: We take $28 \times 28 = 784$ input neurons, some hidden neurons and 10 output neurons to classify MNIST images. The softmax of the 10 output neurons is the predicted probability of the different class labels.

$$P(C_i|x) = s\left(g^{(2)}(b^{(2)} + w^{(2)}g^{(1)}(b^{(1)} + w^{(1)}x))\right)$$

where s is the softmax function (see slides “Supervised Learning”).

A Quick Introduction to Flux and MLJFlux