Multinomial logistic regression for multiple classes

- Logistic regression can be easily extended to multiple labels $2 < |\mathcal{Y}| < \infty$

- Assume we have $n = |\mathcal{Y}| > 2$ labels, then we want to predict the conditional probabilities

$$h(\mathbf{x}) \approx \begin{bmatrix} \mathbb{P}(Y = y_1 \mid X = \mathbf{x}) \\ \vdots \\ \mathbb{P}(Y = y_n \mid X = \mathbf{x}) \end{bmatrix} \in \mathbb{R}^n,$$

- Thus, the output of the hypothesis h is a vector $\mathbf{p} \in [0,1]^n$ such that $p_1 + \ldots + p_n = 1$ (i.e., we predict a discrete probability distribution $\mathbf{p} \in \Delta_n$ on $\mathcal{Y}$)

- To this end, we use n affine-linear functions $f_{\mathbf{w}_1,b_1}, \ldots, f_{\mathbf{w}_n,b_n}$ in combination with a suitable activation function ϕ — this time the softmax function $\boldsymbol{\sigma}: \mathbb{R}^n \to \Delta_n$

$$\boldsymbol{\sigma}(\mathbf{z}) = \left[\frac{\exp(z_j)}{\sum_{k=1}^n \exp(z_k)}\right]_{j=1,\ldots,n}$$

The softmax function is a way to convert a list of numbers (called scores or logits) into a list of probabilities. It's mainly used in situations where you want to classify something into multiple categories

- Thus, we obtain as the ansatz of multinomial logistic regression (MLR)

$$\mathbb{P}(Y = y_j \mid X = \mathbf{x}) = \frac{\exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}{\sum_{k=1}^n \exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)}$$

we can use the Softmax function in order to convert the list of numbers to probabilities. which we would have the MLR method. but we have to also consider the the Loss function for that then we use the Cross Entropy Loss (a form that is usually can be examined
for the models that predict the probabilities. lower value indicates the predicted to be close to the actual lables. A higher value indicates a larger discrepancy)

- The hypothesis $h\colon \mathbb{R}^d \to \Delta_n$ takes then the compact form

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \boldsymbol{\sigma}\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right), \qquad \mathbf{W} = \left[\mathbf{w}_1, \ldots, \mathbf{w}_n\right]^\top \in \mathbb{R}^{n\times d}, \quad \mathbf{b} = \left[b_1, \ldots, b_n\right]^\top \in \mathbb{R}^n$$

where the matrix $\mathbf{W}$ collects all weight vectors $\mathbf{w}_k$, analogously for $\mathbf{b}$

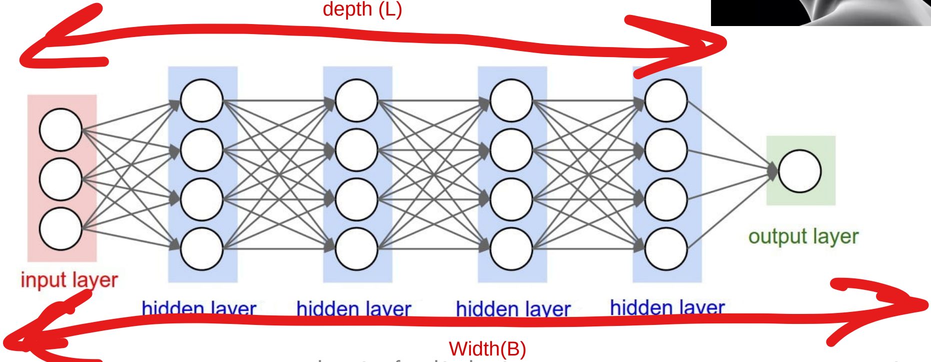
- The ERM rule for MLR is again derived from a maximum likelihood principle. This time we use the cross entropy loss

$$\ell\left(h_{\mathbf{W},\mathbf{b}}, (\mathbf{x}, y)\right) = -\sum_{j=1}^n \mathbf{1}_{\{y_j\}}(y) \ \ln\left(\frac{\exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}{\sum_{k=1}^n \exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)}\right)$$

(note, that only one summand is non-zero)

2.3 Outlook: Neural networks as nonlinear predictors

- (Artificial) neural networks (NN) are (nonlinear) hypotheses $h\colon \mathcal{X} \to \mathcal{Y}$, $\mathcal{X} \subseteq \mathbb{R}^d$, with a particular structure...
- ...which mimics the interaction of neurons in the human brain.
- Here, simple computational nodes (neurons) are connected by a communication network and can thus solve complex tasks.



Ingredients

- An artificial neural network is determined by

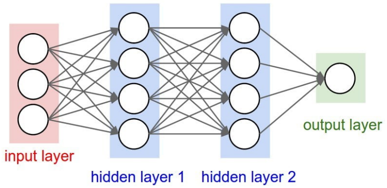
  1. its architecture, i.e., which neurons are connected to which ones,
  2. its parameters or weights $\mathbf{w} \in \mathbb{R}^p$, which determine the computation or information processing per neuron,
  3. and activation functions $\phi, \rho \colon \mathbb{R} \to \mathbb{R}$ which convert the neuron's computational results into outputs.

- The architecture as well as the activation function is determined in advance and the parameters are learned from training data.

- There are different architecture types of NN. We will deal with so-called feedforward NN (FNN).

The architecture

- FNN are constructed from nodes or neurons v and directed edges $e = (v, v')$ between nodes.
- Let the set of all neurons be V and E the set of all edges in the network.



- The nodes are organized in disjoint layers $V_k = \{v_{k,1}, \dots, v_{k,n_k}\}$, $k = 0, \dots, L$:

$$V = (V_0, \dots, V_L), \qquad V_k \cap V_{k'} = \emptyset, \qquad |V_k| = n_k.$$

- Communication edges exists only between adjacent layers:

$$E \subseteq \{(v_{k,i}, v_{k+1,j}) \colon v_{k,i} \in V_k \text{ and } v_{k+1,j} \in V_{k+1}\}.$$

What happens in a neuron?

- Following the model by McCulloch & Pitts each neuron $v_{k,i}$ is a linear hypothesis

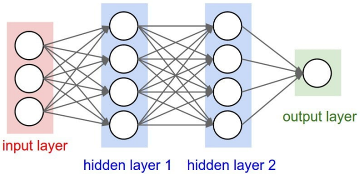
$$y_{k,i} = v_{k,i}(\mathbf{y}_{k-1}) := \phi\left(\sum_{j=1}^{n} w_j \; y_{k-1,j} + b\right) \in \mathbb{R},$$

  where the inputs are the incoming signals/outputs from the previous layer

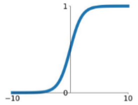
- Neural networks are thus also called multilayer perceptrons

- Each neuron $v_{k,i}$ has its own weights $w_j = w_j^{(k,i)}$ and bias $b = b^{(k,i)}$ which are learned during the training

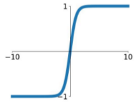
- But it is common to use the same activation function ϕ for all neurons in hidden layers



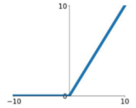
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



tanh
$\tanh(x)$



ReLU
$\max(0, x)$



Formal description of artifical neural networks

A feedforward neural network is a hypothesis $h\colon \mathcal{X} \to \mathcal{Y}$ of the form

$$h(\mathbf{x}) = \rho \circ f_{\mathbf{W}_L,\mathbf{b}_L} \circ \phi \circ f_{\mathbf{W}_{L-1},\mathbf{b}_{L-1}} \circ \phi \circ \cdots \circ \phi \circ f_{\mathbf{W}_1,\mathbf{b}_1}(\mathbf{x}),$$

where

- $\phi\colon \mathbb{R} \to \mathbb{R}$ as well as $\rho\colon \mathbb{R} \to \mathcal{Y}$ are chosen activation functions whose applications are to be understood componentwise,
- given layerwise weight matrices $\mathbf{W}_k \in \mathbb{R}^{n_k \times n_{k-1}}$ and bias vectors $\mathbf{b}_k \in \mathbb{R}^{n_k}$

$$f_{\mathbf{W}_k,\mathbf{b}_k}(\mathbf{y}) := \mathbf{W}_k\ \mathbf{y} + \mathbf{b}_k$$

- with $n_k \in \mathbb{N}$ denotes the size of the k-th layer V_k.

Characteristics of FNN

$$h(\mathbf{x}) = \rho \circ f_{\mathbf{W}_L,\mathbf{b}_L} \circ \sigma \circ f_{\mathbf{W}_{L-1},\mathbf{b}_{L-1}} \circ \sigma \circ \cdots \circ \sigma \circ f_{\mathbf{W}_1,\mathbf{b}_1}(\mathbf{x}),$$

- **Depth:** L
- **Width:** $B := \max_{k=0,\dots,L} n_k$
- **Size:** $n := n_0 + n_1 + \ldots + n_L$
- **Architecture:** (V, E) with $V = (V_0, \ldots, V_L)$
- **Hypothesis class:** $\mathcal{H}_{V,E,\sigma,\rho} \subseteq \mathcal{Y}^{\mathcal{X}}$
- **Number of parameters:**

$$p_{V,E} := |E| + |V_1| + \ldots + |V_L| \leq L\,(B^2 + B)$$

Shallow and deep neural networks

- An FNN with no hidden layer ($L = 1$) is nothing but a linear hypothesis.
- Shallow networks have one hidden layer ($L = 2$)

$$h(\mathbf{x}) = \rho\left(\beta + \sum_{i=1}^{n} \alpha_i \, \sigma\left(\mathbf{w}_i^\top \mathbf{x} + b_i\right)\right), \qquad \alpha_i, \beta, b_i \in \mathbb{R}, \;\; \mathbf{w}_i \in \mathbb{R}^d,$$

  where $|V_1| = n$ and $E = (V_0 \times V_1) \cup (V_1 \times V_2)$
- Deep networks have two or more hidden layers ($L \geq 3$)
- Google's AlphaGo: two NNs with $L = 13$ layers each. ChatGPT3: $L \approx 100$ layers.

inputs in RNN, are the
outputs of the next input
in the hidden layers

- Recurrent neural networks allow the output of V_k as input to previous layers V_{k-1}, e.g., LSTM networks
- Convolutional neural networks implement convolutional filters in each layer and are special cases of FNN

Structure of the hypothesis class of FNN

- FNN are basically a composition of vector-valued linear hypotheses

- In particular, we have for fully connected FNN $\mathcal{H} = \mathcal{H}_{V,E,\phi,\rho}$ of depth L

$$\mathcal{H} = \mathcal{L}_{n_{L-1},\rho} \circ \mathcal{H}_{L-1} \cdots \circ \mathcal{H}_1, \qquad \mathcal{H}_k = \mathcal{L}_{n_{k-1},\phi} \times \cdots \times \mathcal{L}_{n_{k-1},\phi}$$

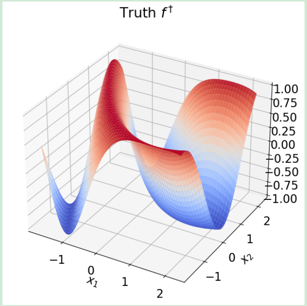
- This structure will become useful in the analysis of feedforward neural networks

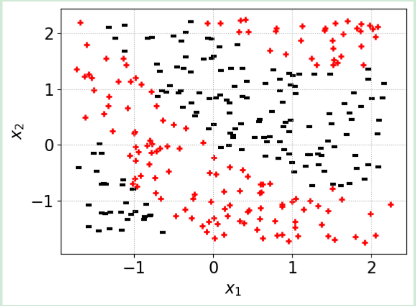
- Nonetheless, due to the nonlinearity of the activiation function ϕ a multilayer perceptron $(L \geq 2)$ is a nonlinear hypothesis

Example: Synthetic data

- We consider nonlinearly separable training data in $\mathcal{X} = \mathbb{R}^2$ using a true hypothesis, i.e., $(X, Y) \sim \mu$ is given by

$$Y = \operatorname{sgn}(f^\dagger(X)), \qquad X \sim \mathrm{U}[-1.75, 2.25]^2$$

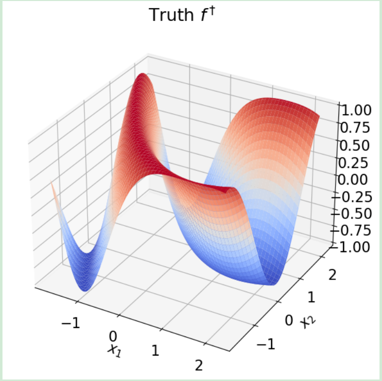


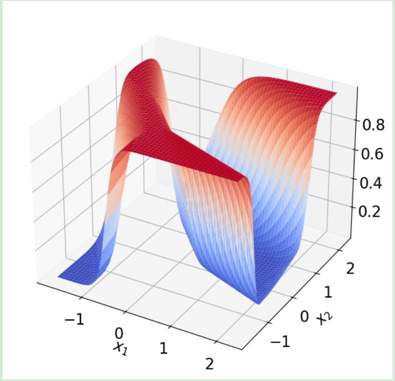


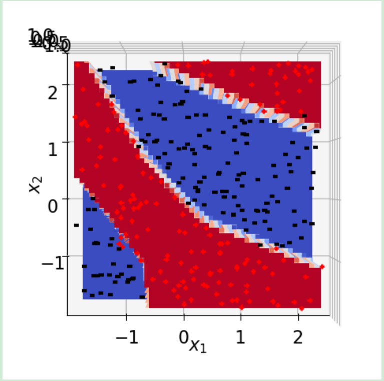
- We then train a fully connected shallow neural network with $n_1 = 25$ neurons using ReLU activation function ϕ and $\rho = \mathrm{sig}$.
- These are the default choices for ϕ and ρ in the `scikit-learn` routine `MLPClassifier`.

- For training we choose the (subgradient) SGD with constant learning rate η_k and let it run for 100 epochs (comes later).
- We plot the learned FNN $h_s\colon \mathbb{R}^2 \to [0,1]$ and the resulting classifier

$$\operatorname{sgn}(h_s - 0.5)\colon \mathbb{R}^2 \to \{-1,+1\}$$

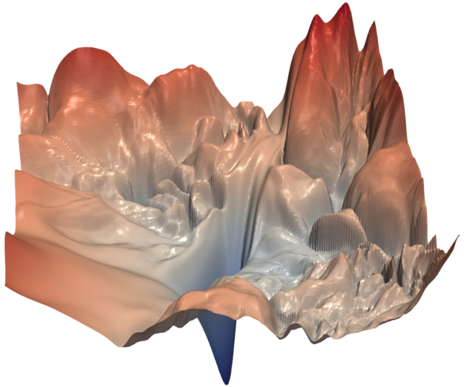


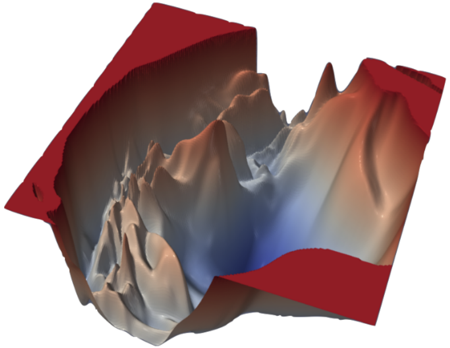




Training neural networks

Here are two local two-dimensional slices through the “loss landscape” for deep neural networks where the x- and y-direction in the plots are two random directions in the parameter space $\mathbb{R}^p$ of the neural network and the z-direction is the empirical risk:





Souce: "Visualizing the Loss Landscape of Neural Nets" (2017)

Summary

Evaluate the three methods we have learned so far by completing the table below using

$$1\ (\text{best}),\quad 2\ (\text{medium}),\quad 3\ (\text{worst})$$

for the performance regarding the corresponding important errors:

Method	ε_{app}	ε_{est}	ε_{opt}
Perceptron	3	3	2
Logistic regression	2	2	1
Neural networks	1	1	3

PERCEPTRON
Error Approximation (3): The perceptron can only approximate linear decision boundaries. It struggles with complex data patterns, making it less effective for error approximation.

Error Optimization (3): The perceptron uses a basic learning rule and does not perform well in optimizing errors beyond simple linear separability. It's limited in its ability to minimize loss effectively.

Error Estimation (3): The perceptron does not provide probability estimates; it simply classifies inputs into two categories (0 or 1). As a result, its error estimation capabilities are quite poor.

LOGISTIC REGRESSION
Error Approximation (2): Logistic regression is better at approximating decision boundaries than a perceptron because it can handle binary outcomes with probabilities. However, it still struggles with non-linear relationships.

Error Optimization (2): Logistic regression uses maximum likelihood estimation to optimize errors, providing a more robust approach than the perceptron but less flexible than neural networks.

Error Estimation (2): Logistic regression provides probabilistic outputs, giving it a moderate rating in error estimation. However, it is limited to binary classification (or can be extended to multiclass using techniques like one-vs-all).

Take home messages/questions

- Linear hypotheses are simply separating hyperplanes, aren't they?
- How does the Perceptron algorithm work and when does it terminate?
- What is the ansatz and purpose of logistic regression?
- What is the log-loss and what is its advantage for training?
- How can we extend logistic regression to multiple classes?
- How is a neural network built up?
- Compared to linear hypotheses neural networks are... ?

NN

Error Approximation (1): Neural networks are highly flexible and can approximate complex functions and non-linear decision boundaries effectively, leading to excellent performance in error approximation.

Error Optimization (1): They utilize advanced optimization techniques (like gradient descent and backpropagation) that allow them to minimize errors effectively across multiple layers and neurons.

Error Estimation (1): Neural networks can produce outputs that can be interpreted as probabilities, especially with activation functions like softmax. This capability allows them to provide precise error estimates.