Logistic Regression

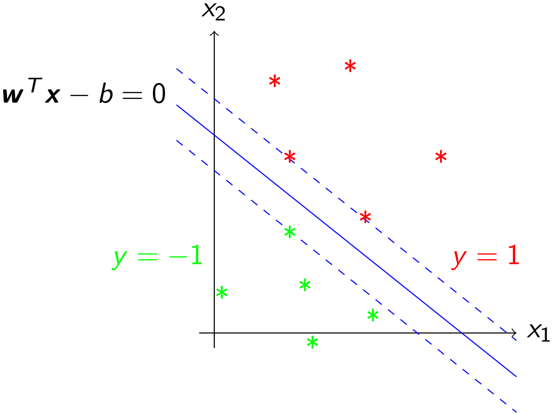
Machine Learning

Axel Carlier, Jean-Yves Tourneret

2024

Outline

Reminder: Support Vector Machines



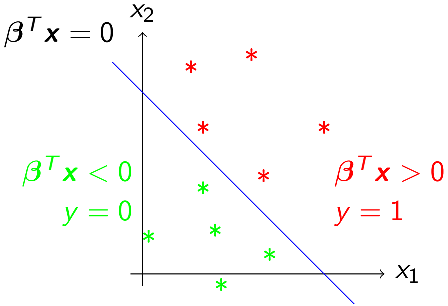
Find an optimal separating hyperplane, such that $|\boldsymbol{w}^T\boldsymbol{x}_i - b| \geq 1$.

Logistic regression for binary classification

For linear regression, we again define a linear model:

$$z = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p = \boldsymbol{\beta}^T \boldsymbol{x}, \text{ with } \boldsymbol{x} = [1, x_1, ..., x_p]^T$$

This linear model acts as a **separator** for the 2 classes: $y \in \{0, 1\}$



Back to the Bayesian classifier

Bayesian rule for two classes ω_1 and ω_2

$$\begin{aligned} d^*(\boldsymbol{x}) = \ \omega_1 &\Leftrightarrow f(\boldsymbol{x}|\omega_1)P(\omega_1) \geq f(\boldsymbol{x}|\omega_2)P(\omega_2) \\ &\Leftrightarrow \frac{f(\boldsymbol{x}|\omega_1)P(\omega_1)}{f(\boldsymbol{x}|\omega_2)P(\omega_2)} \geq 1 \\ &\Leftrightarrow \frac{\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{m}_1)\right]P(\omega_1)}{\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{m}_2)\right]P(\omega_2)} \geq 0 \end{aligned}$$

In the Gaussian case with $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}$, one obtains

$$d^*(\boldsymbol{x}) = \ \omega_1 \Leftrightarrow \sigma(\boldsymbol{\beta}^T\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\beta}^T\boldsymbol{x}}} \leq 0.5$$

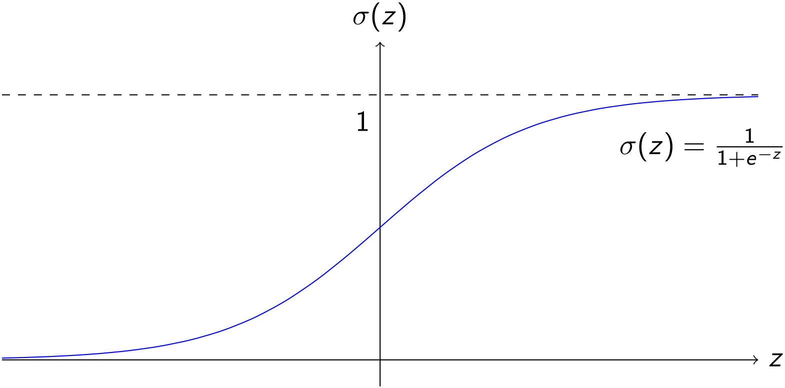
with

$$\boldsymbol{\beta}^T\boldsymbol{x} = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{m}_1-\boldsymbol{m}_2\right) + \frac{1}{2}\boldsymbol{m}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{m}_1 - \frac{1}{2}\boldsymbol{m}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{m}_2 + \ln\left[\frac{P(\omega_1)}{P(\omega_2)}\right].$$

Logistic regression for binary classification

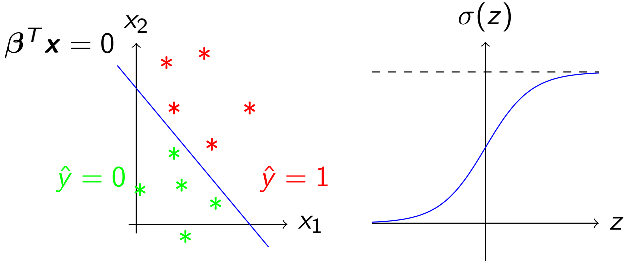
The **sigmoid** or **logistic** function is defined as:

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$



Logistic regression for binary classification

2D case where data is linearly separable:



$$h_\beta(\boldsymbol{x}) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

- We predict $\hat{y} = 1$ if $\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0 \Leftrightarrow \sigma(\boldsymbol{\beta}^T \boldsymbol{x}) > 0.5$
- We predict $\hat{y} = 0$ if $\beta_0 + \beta_1 x_1 + \beta_2 x_2 \leq 0 \Leftrightarrow \sigma(\boldsymbol{\beta}^T \boldsymbol{x}) \leq 0.5$

Logistic regression for binary classification

How to estimate $\boldsymbol{\beta}$ in this framework?

Let $\mathcal{D} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$ be our **training set**, with n examples:

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_p^{(i)} \end{bmatrix} \in \mathbb{R}^p \text{ and } y^{(i)} \in \{0,1\}, \forall i \in \{1, .., n\}$$

and the **model** is defined by the sigmoid function:

$$P(y = 1 | \boldsymbol{x}; \boldsymbol{\beta}) = \sigma(\boldsymbol{\beta}^T \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \boldsymbol{x}}}$$

One can define the following classification rule

- Predict $\hat{y} = 1$ if $\sigma(\boldsymbol{\beta}^T \boldsymbol{x}) > 0.5$
- Predict $\hat{y} = 0$ if $\sigma(\boldsymbol{\beta}^T \boldsymbol{x}) \leq 0.5$

Logistic regression for binary classification

We define an **objective function** that we want to minimize:

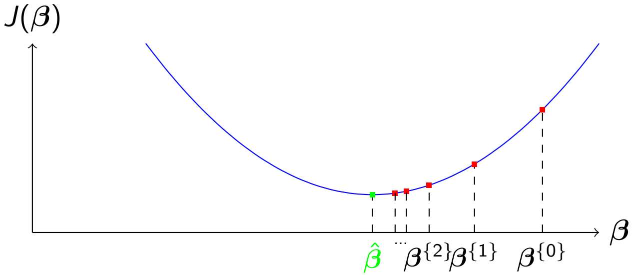
$$J(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} loss\left(h_{\beta}(\boldsymbol{x}^{(i)}), y^{(i)}\right)$$

The loss function should be small when $h_{\beta}(\boldsymbol{x}^{(i)})$ is close to $y^{(i)}$ and large otherwise.

We want to estimate:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$$

Gradient Descent



Algorithm: Gradient descent $(\mathcal{D}, \alpha)$

Initialize $\boldsymbol{\beta}^{\{0\}} \leftarrow 0$, $k \leftarrow 0$

WHILE no convergence **DO**

  FOR j from 1 to p **DO**

$$\boldsymbol{\beta}_j^{\{k+1\}} \leftarrow \boldsymbol{\beta}_j^{\{k\}} - \alpha \frac{\partial J(\boldsymbol{\beta}^{\{k\}})}{\partial \boldsymbol{\beta}_j}$$

  END FOR

  $k \leftarrow k + 1$

END WHILE

Logistic regression for binary classification

Back to our problem:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} loss \left(h_\beta(\boldsymbol{x}^{(i)}), y^{(i)} \right)$$

One could use for example the squared error:

$$loss \left(h_\beta(x^{(i)}), y^{(i)} \right) = \frac{1}{2} \left(h_\beta \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 = \frac{1}{2} \left(\frac{1}{1 + e^{-\boldsymbol{\beta}^T \boldsymbol{x}^{(i)}}} - y^{(i)} \right)^2$$

Unfortunately, this function is **not convex** which means the minimum that can be found with gradient descent is not necessary global!

Another idea: maximum likelihood estimation

Assuming that

$$P(Y_i = 1|\boldsymbol{x_i}; \boldsymbol{\beta}) = \sigma(\boldsymbol{\beta}^T \boldsymbol{x}_i) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \boldsymbol{x}_i}} = h_\beta(\boldsymbol{x}_i)$$

and

$$P(Y_i = 0|\boldsymbol{x_i}; \boldsymbol{\beta}) = 1 - P(Y_i = 1|\boldsymbol{x_i}; \boldsymbol{\beta}) = 1 - h_\beta(\boldsymbol{x}_i),$$

the **likelihood** of the vector $y_1, ..., y_n$ can be written

$$L(y_1, ..., y_n; \boldsymbol{\beta}) = \prod_{i=1}^{n} P(Y_i = y_i|\boldsymbol{x_i}; \boldsymbol{\beta}) = \prod_{i=1}^{n} [h_\beta(\boldsymbol{x}_i)]^{y_i} [1 - h_\beta(\boldsymbol{x}_i)]^{1-y_i}.$$

Thus the **negative log-likelihood** is

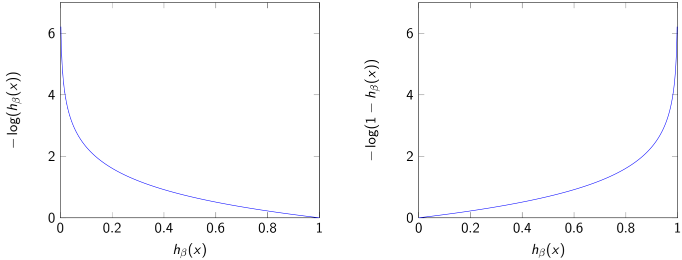
$$-\ln [L(y_1, ..., y_n; \boldsymbol{\beta})] = \sum_{i=1}^{n} -y_i \ln [h_\beta(\boldsymbol{x}_i)] - (1 - y_i) \ln [1 - h_\beta(\boldsymbol{x}_i)],$$

We would like to maximize the likelihood of observing the training samples, i.e. minimize the negative log-likelihood.

Logistic regression for binary classification

We introduce the **logistic loss function (or binary cross entropy)** defined as:

$$\begin{aligned} loss[h_\beta(\boldsymbol{x}), y] &= \begin{cases} -\ln[h_\beta(\boldsymbol{x})] \text{ if } y = 1 \\ -\ln[1 - h_\beta(\boldsymbol{x})] \text{ if } y = 0 \end{cases} \\ &= -y \ln[h_\beta(\boldsymbol{x})] - (1-y) \ln[1 - h_\beta(\boldsymbol{x})] \end{aligned}$$



This loss function is small when $y = 1$ and $h_{\boldsymbol{\beta}}(\boldsymbol{x})$ is large and when $y = 0$ and $h_{\boldsymbol{\beta}}(\boldsymbol{x})$ is small, i.e., when there are few classification errors!

Logistic regression for binary classification

The objective function $J(\beta)$ can be written:

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} -y^{(i)} \ln(h_\beta(\boldsymbol{x}^{(i)})) - (1 - y^{(i)}) \ln(1 - h_\beta(\boldsymbol{x}^{(i)})).$$

This function is **CONVEX** !

$\rightarrow$ We can use gradient descent to find its minimum.

Logistic regression for binary classification

Gradient descent requires partial derivatives to be computed:

$$\begin{aligned}
\frac{\partial J}{\partial \beta_j} &= \frac{\partial}{\partial \beta_j} \frac{1}{n} \sum_{i=1}^{n} \left\{ -y^{(i)} \ln(h_\beta(\boldsymbol{x}^{(i)})) - (1 - y^{(i)}) \ln(1 - h_\beta(\boldsymbol{x}^{(i)})) \right\} \\
&= \frac{1}{n} \sum_{i=1}^{n} \left\{ -y^{(i)} \frac{\partial}{\partial \beta_j} \ln(h_\beta(\boldsymbol{x}^{(i)})) - (1 - y^{(i)}) \frac{\partial}{\partial \beta_j} \ln(1 - h_\beta(\boldsymbol{x}^{(i)})) \right\} \\
&= \frac{1}{n} \sum_{i=1}^{n} \left[h_\beta(\boldsymbol{x}^{(i)}) - y^{(i)} \right] \boldsymbol{x}_j^{(i)} \qquad \text{(all computations made)}
\end{aligned} \tag{1}$$

Comments on the previous slide

In order to compute $\frac{\partial}{\partial \beta_j} \ln(h_\beta(\boldsymbol{x}))$, we define $z = \boldsymbol{\beta}^T \boldsymbol{x}$, $u = \sigma(z)$ and $v = \ln(u)$

$$
\begin{aligned}
\frac{\partial}{\partial \beta_j} \ln[h_\beta(\boldsymbol{x})] &= \frac{\partial}{\partial \beta_j} \ln(\sigma(\boldsymbol{\beta}^T \boldsymbol{x})) \\
&= \frac{\partial v}{\partial u} \frac{\partial u}{\partial z} \frac{\partial z}{\partial \beta_j} && \text{chain-rule} \\
&= \frac{1}{\sigma(z)} \sigma(z)(1 - \sigma(z)) x_j && \text{since } \sigma'(z) = \sigma(z)[1 - \sigma(z)] \\
&= (1 - h_\beta(\boldsymbol{x})) x_j
\end{aligned}
$$

Similarly, we prove $\frac{\partial}{\partial \beta_j} \ln(1 - h_{\boldsymbol{\beta}}(\boldsymbol{x})) = -h_{\boldsymbol{\beta}}(\boldsymbol{x}) x_j$

Logistic regression for multinomial classification

How to proceed with $k > 2$ **classes** ?

We define the **softmax** function:

$$P(y = i|\boldsymbol{x}, \beta) = \frac{e^{\boldsymbol{\beta}_i^T \boldsymbol{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\beta}_j^T \boldsymbol{x}}}$$

For a sample from class j, we define a one-hot vector $\boldsymbol{y}$ of dimension k such that

$$y_i = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ otherwise} \end{cases} \tag{2}$$

Logistic regression for multinomial classification

For each sample $\boldsymbol{x}^{(i)}$, we can predict its probability to belong to each of the k classes.

$$h_\beta(\boldsymbol{x}^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|\boldsymbol{x}^{(i)}; \boldsymbol{\beta}) \\ p(y^{(i)} = 2|\boldsymbol{x}^{(i)}; \boldsymbol{\beta}) \\ \vdots \\ p(y^{(i)} = k|\boldsymbol{x}^{(i)}; \boldsymbol{\beta}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\boldsymbol{\beta}_j^T \boldsymbol{x}^{(i)}}} \begin{bmatrix} e^{\boldsymbol{\beta}_1^T \boldsymbol{x}^{(i)}} \\ e^{\boldsymbol{\beta}_2^T \boldsymbol{x}^{(i)}} \\ \vdots \\ e^{\boldsymbol{\beta}_k^T \boldsymbol{x}^{(i)}} \end{bmatrix}$$

Logistic regression for multinomial classification

The objective function can be written as:

$$J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\mathbb{I}(y^{(i)} = j)\ln\left[\frac{e^{\boldsymbol{\beta}_j^T\boldsymbol{x}^{(i)}}}{\sum_{l=1}^{k} e^{\boldsymbol{\beta}_l^T\boldsymbol{x}^{(i)}}}\right]. \tag{3}$$

Its gradient is:

$$\nabla_{\beta_j} J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}^{(i)}\left[\mathbb{I}(y^{(i)} = j) - P(y^{(i)} = j|\boldsymbol{x}^{(i)};\boldsymbol{\beta})\right].$$

Outline

Artificial Neuron

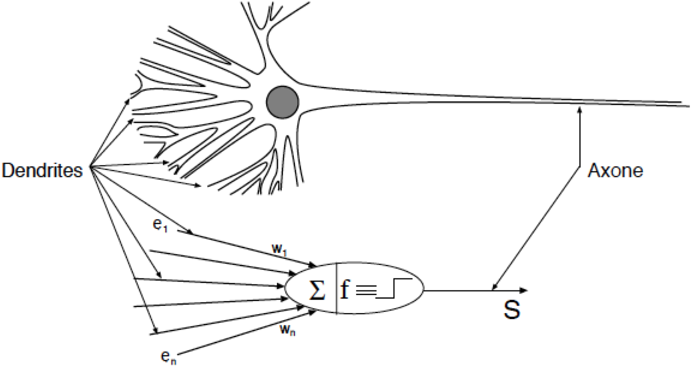
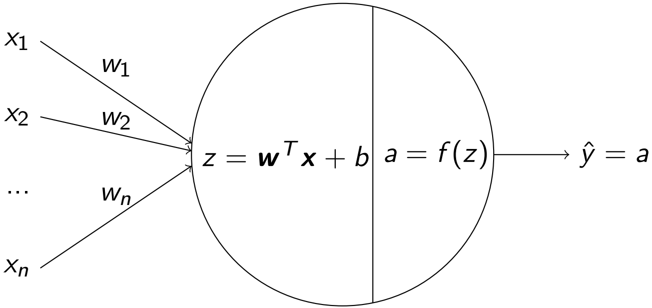


Figure: Structure of biological and artificial neurones

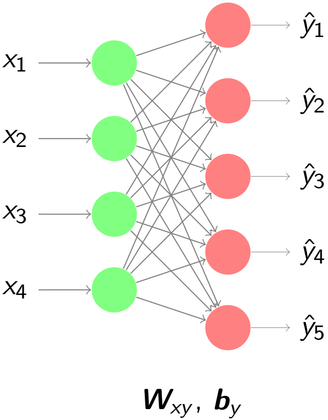
Artificial Neuron



An artificial neuron is defined as a triplet of synaptic weights $\boldsymbol{w}$, bias b and activation function f.
The activation a of a neuron answering to an input $\boldsymbol{x}$ is:

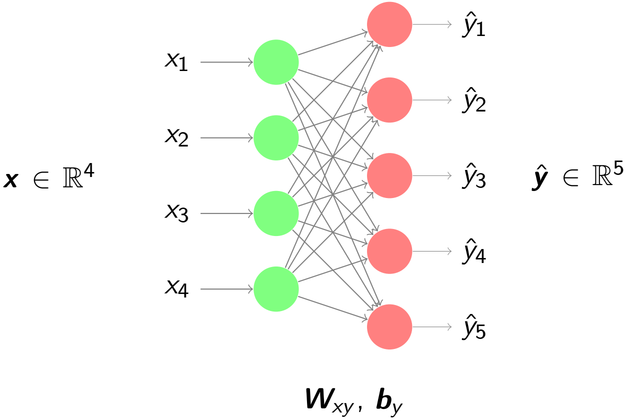
$$a = f(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Neural network: Single-layer perceptron



A single-layer perceptron is in fact a little bit more general and can be composed of several neurons assembled in a single layer ($p = 4$ is the data dimension, and $K = 5$ is the number of classes).

Neural network: Single-layer perceptron

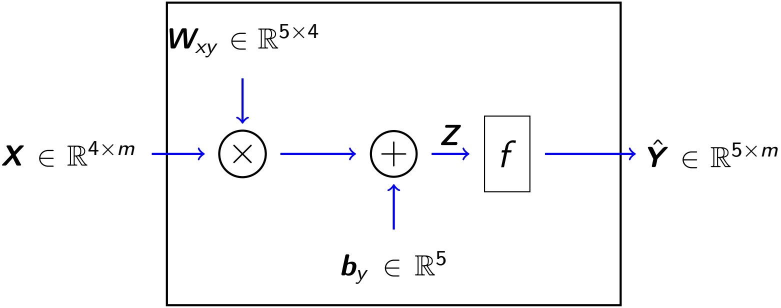


The perceptron computation is implemented as a matrix product:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{W}_{xy}\boldsymbol{x} + \boldsymbol{b}_{\boldsymbol{y}})$$

where f is an activation function

Neural network: Single-layer perceptron

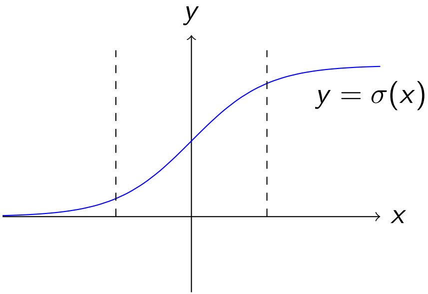


Another representation of the single-layer perceptron for prediction and classification.

Single-layer perceptron

Activation functions, denoted as f, can often be decomposed in three subdomains, as for sigmoid functions below.

- a non-activated part, below a certain threshold;
- a transition part, in the threshold neighbourhood;
- an activated part, above the threshold.



Single-layer perceptron

Algorithm 1 Single-layer perceptron training

1. Initialization of weights $\boldsymbol{W}_{xy}^{\{0\}}$ and biases $\boldsymbol{b}_{y}^{\{0\}}$.
2. Presentation of a set of inputs $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}$ and corresponding outputs $\boldsymbol{y}^{(1)}, .., \boldsymbol{y}^{(n)}$
3. Model prediction and objective function computation:

$$\hat{\boldsymbol{y}}^{(i)\{k\}} = f(\boldsymbol{W}_{xy}^{\{k\}}\boldsymbol{x}^{(i)} + \boldsymbol{b}_{y}^{\{k\}}) \text{ et } J = \sum_{i=1}^{n} \text{loss}(\hat{\boldsymbol{y}}^{(i)\{k\}}, \boldsymbol{y}^{(i)})$$

4. Parameter updates

$$\boldsymbol{W}_{xy}^{\{k+1\}} = \boldsymbol{W}_{xy}^{\{k\}} - \alpha \frac{\partial J}{\partial \boldsymbol{W}_{xy}}, \text{ and } \boldsymbol{b}_{y}^{\{k+1\}} = \boldsymbol{b}_{y}^{\{k\}} - \alpha \frac{\partial J}{\partial \boldsymbol{b}_{y}}$$

   where α is the learning rate $(0 \leq \alpha \leq 1)$.

5. Loop to 2 until convergence (i.e., $\hat{\boldsymbol{y}}^{(i)\{k\}} \approx \boldsymbol{y}^{(i)}$).

Visualization

https://playground.tensorflow.org/