Feed-Forward Neural Networks

Logistic Regression

We use the logistic function to estimate the probability that example *x* belongs to class 1 using the logistic or sigmoid function:

$$p(c(x) == 1) = \sigma(x \cdot w) = \frac{1}{1 + e^{-x \cdot w}}$$

Since classes are binary, we apply a threshold to the probability to make a prediction:

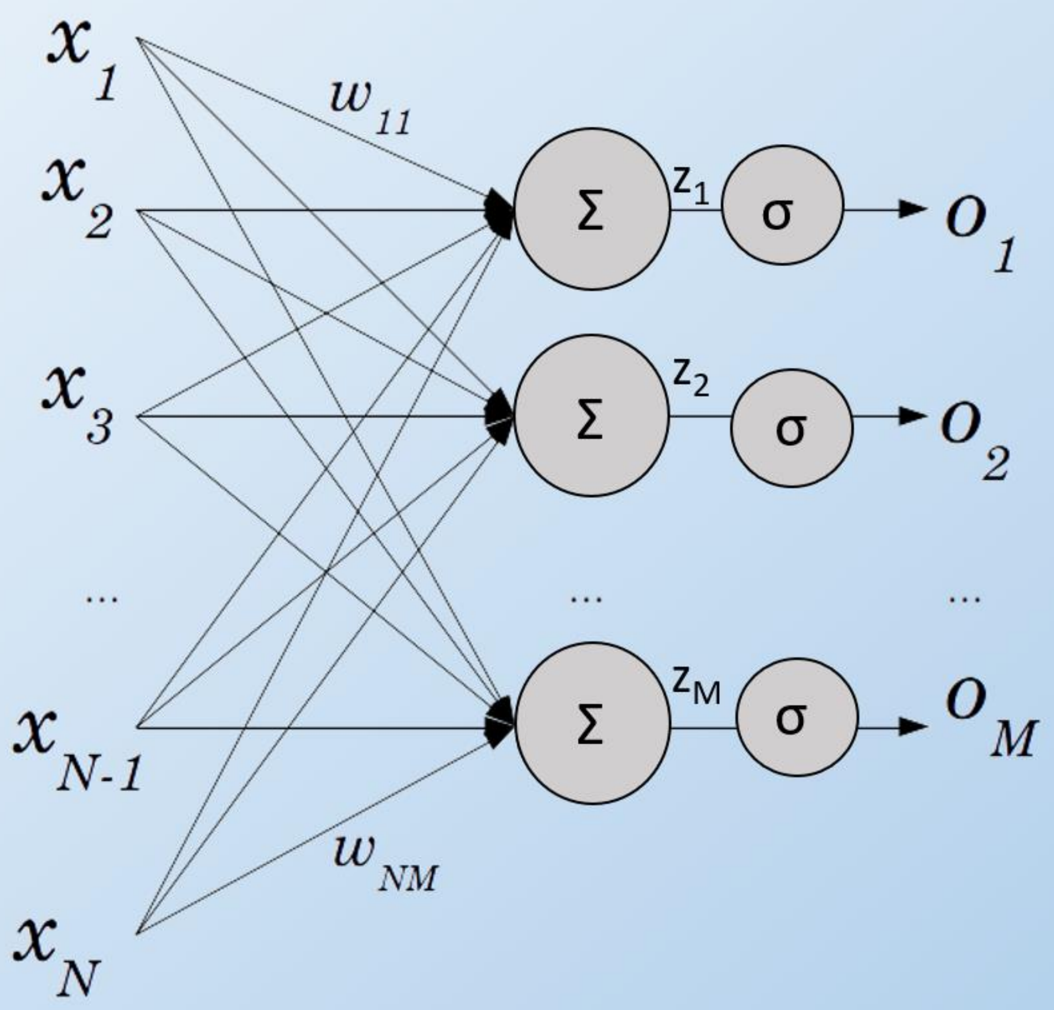
$$pred(x) = \begin{cases} 1 & \text{if } p(c(x) == 1) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

Where $\boldsymbol{x} \cdot \boldsymbol{w}$ is the dot product:

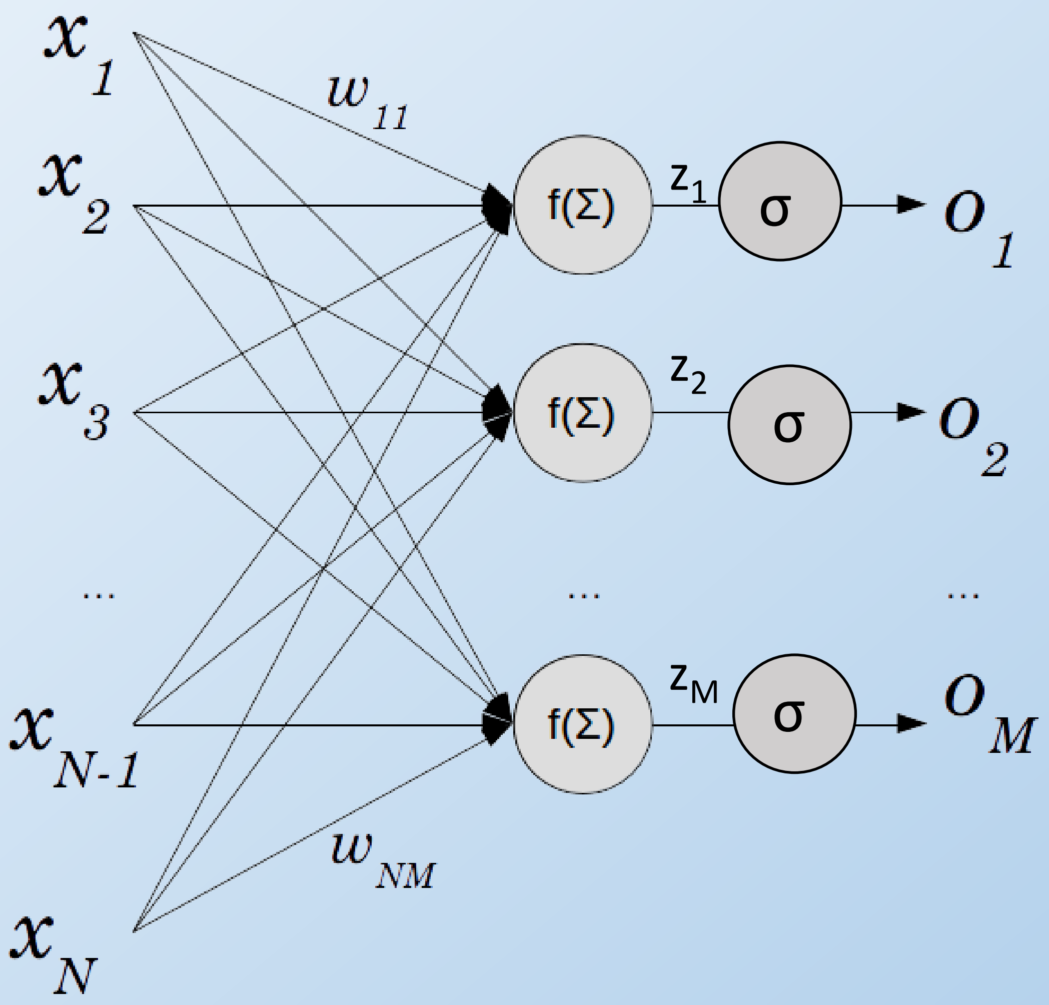
$\boldsymbol{x} \cdot \boldsymbol{w} = x_0 \cdot w_0 + x_1 \cdot w_1 + \ldots + x_{n-2} \cdot w_{n-2} + x_{n-1} \cdot w_{n-1}$

and the parameters $w_0, w_1, \ldots, w_{n-2}, w_{n-1}$ are learned from the training data

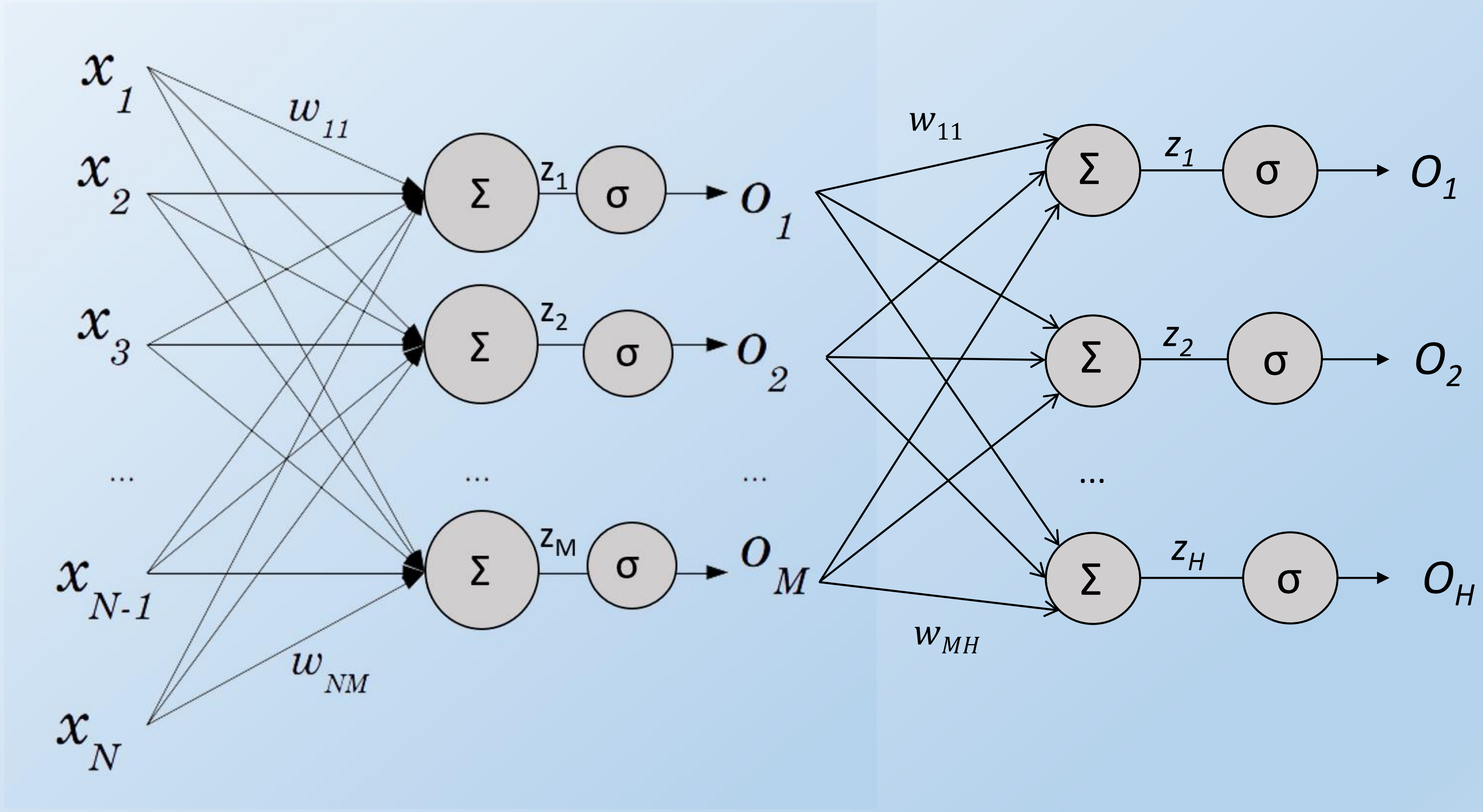
Logistic Regression



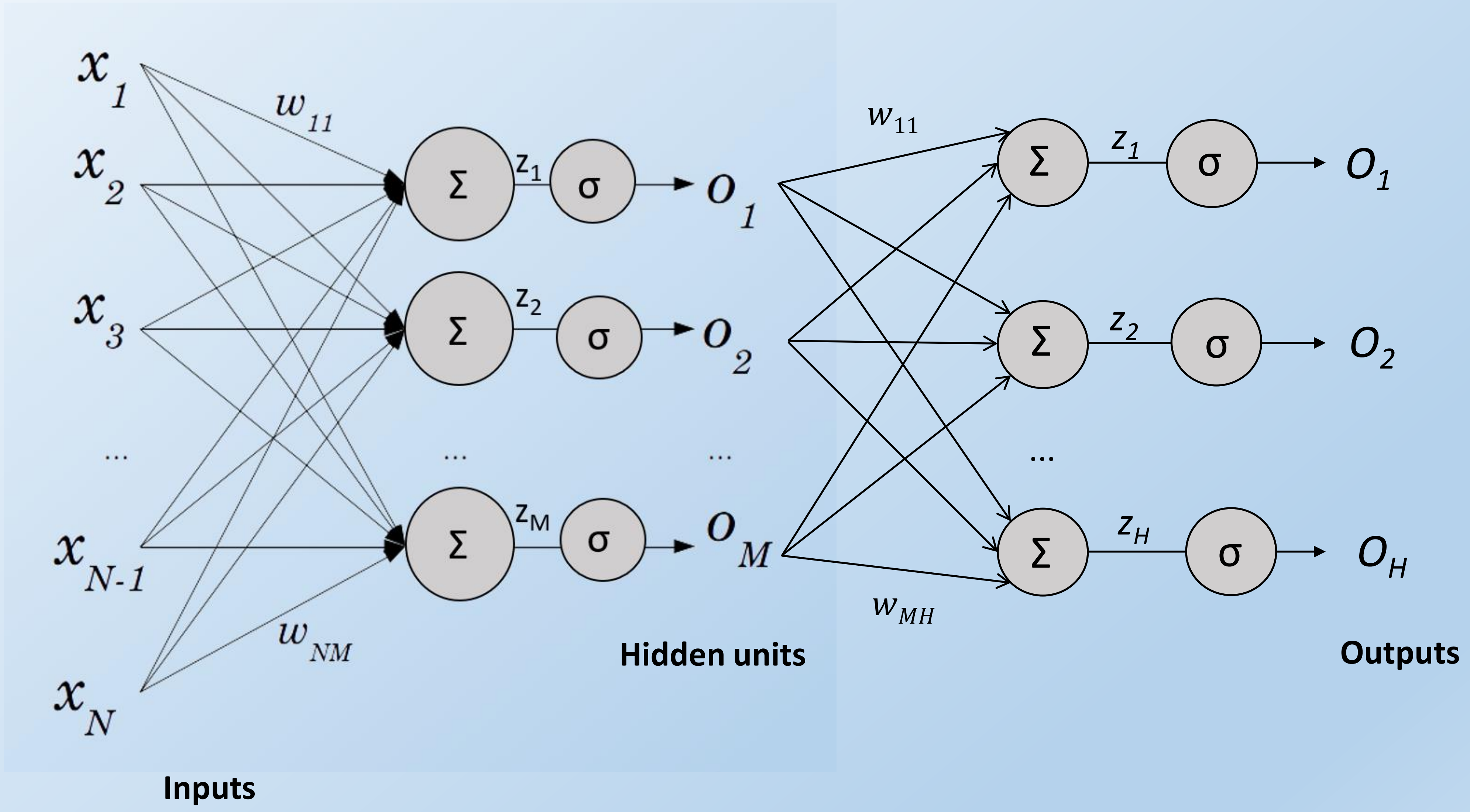
Logistic Regression



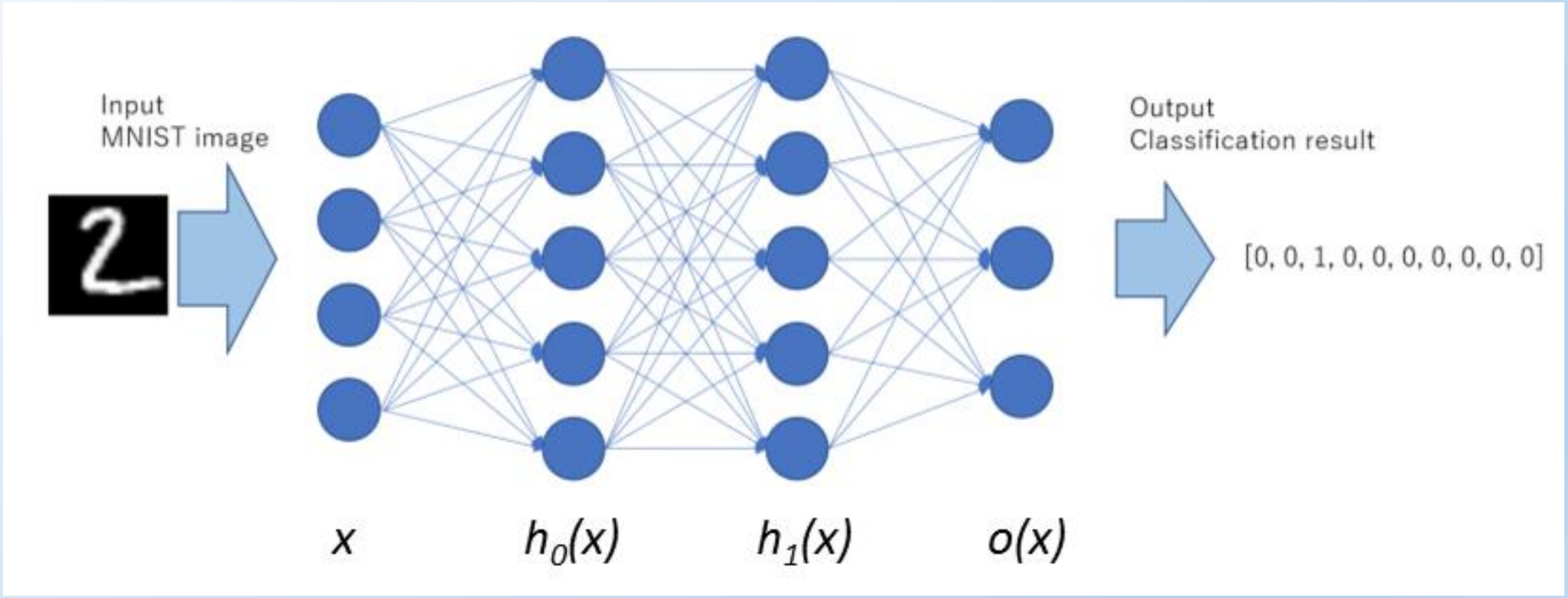
Stacking 2 logistic regressors



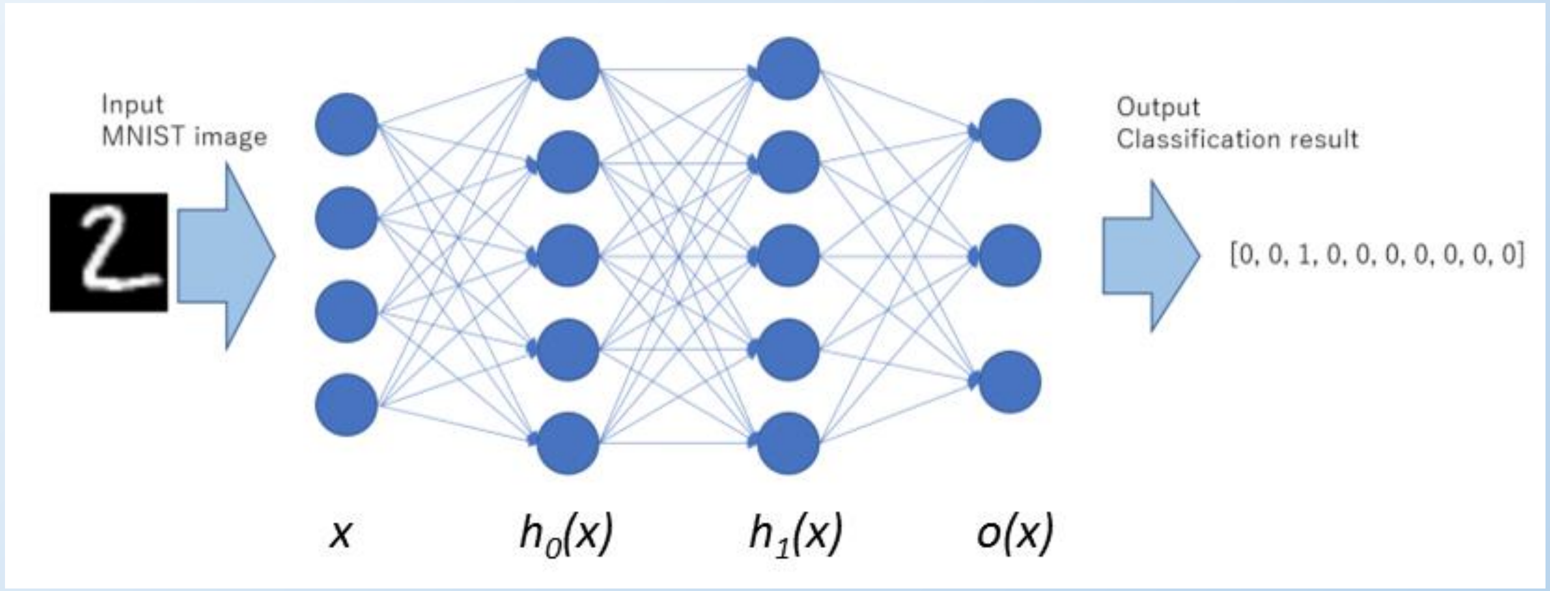
Neural Network with one hidden layer



Neural Network with two hidden layers

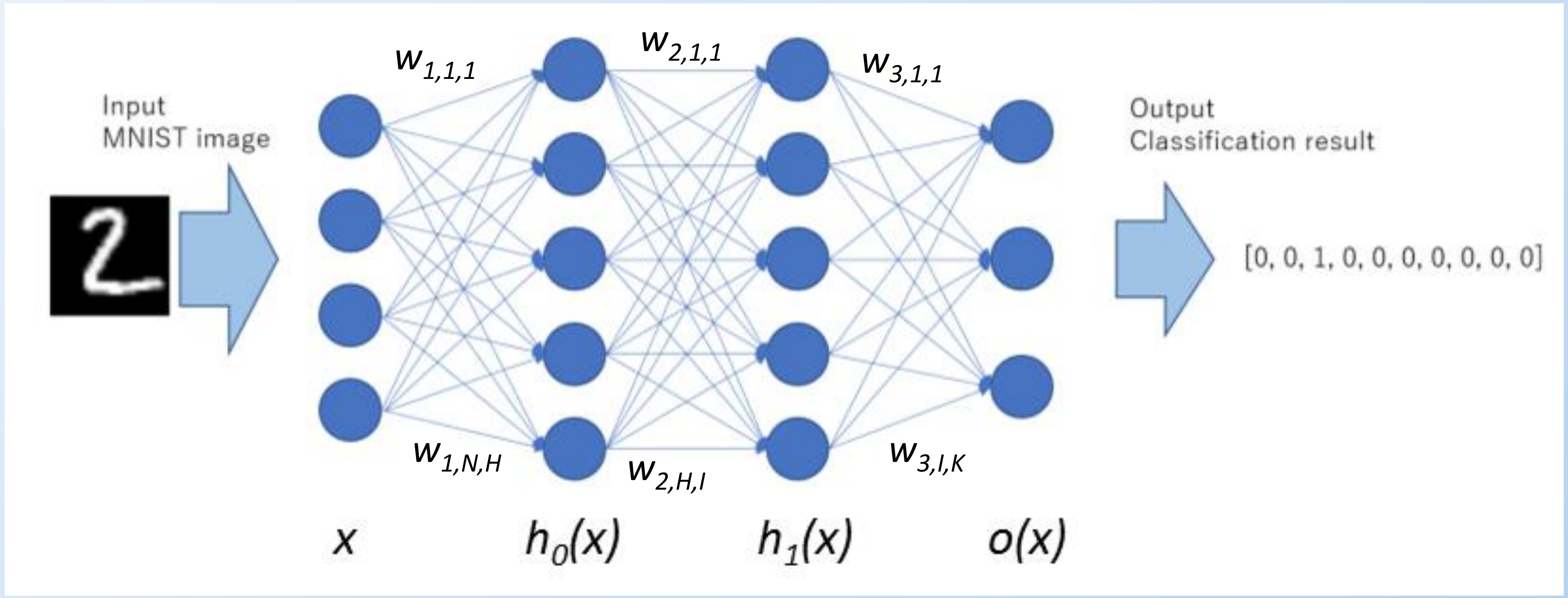


Learning in Neural Networks with hidden layers



- As with logistic regression, the goal is to find the values of the weight matrices that minimize the error
- We know the values of x and o(x) – (X,y in our notation)
- We don't know what the values of $h_0(x)$ and $h_1(x)$ should be
- By applying the chain rule from calculus, we can determine how to modify $h_0(x)$ and $h_1(x)$ in order to reduce the error on the training data
- We modify $h_0(x)$ by changing the values of the first (leftmost) weight matrix
- We modify $h_1(x)$ by changing the values of the second (middle) weight matrix

Neural Network with two hidden layers



Thus, for every weight in the network, we can determine how it affects the error or cost function by multiple applications of the chain rule.

Then we just apply gradient descent, where J is the error or cost function and $w_{i,j,k}$ is the weight in matrix

$$w_{i,j,k} = w_{i,j,k} - \lambda \frac{\partial J}{\partial w_{i,j,k}}$$

Neural Networks for regression

Logistic regression uses a sigmoid function in its output, which restricts the range of outputs to the interval [0,1] – thus logistic regression can only be used for classification – if we removed the sigmoid operation from logistic regression, we would have linear regression.

Feed-forward neural networks can easily be adapted to regression by removing the sigmoid operation in the output layer. Since non-linearities are present in earlier layers, they are still non-linear regressors.

Neural Networks with hidden layers

- We won't cover the details of the derivation of the weight gradients
- We won't implement the learning algorithms
- We will use 2 widely available libraries for neural network learning:
  - sklearn – general purpose machine learning library written in Python
  - tensorflow – high-performance learning for neural networor learning and optimization written in Python

Neural Networks with hidden layers

Our focus will be on understanding and optimizing the choice of:

- Network architecture
  - Number of layers
  - Number of units in each layer
- Optimizer
  - Gradient descent (with and without momentum)
  - Adaptive algorithms – ADAM, RMSPROP, ...
- Nonlinearities
  - We used sigmoid for logistic regression, there are many others: hyperbolic tangent, rectified linear units, ...
- Training parameters:
  - learning rate, batch size, number of epochs, stopping criteria, etc.

Neural Networks with hidden layers

The description in this section has been limited to fully-connected networks – all the units in a layer are connected to all the units in the previous layer and the strength of these connections is given by a weight matrix W.

Later we will study convolutional neural networks, a type of network that is very well suited to image analysis tasks and has been used to achieve super-human performance in several problems.

Feed-forward Neural Networks

Read documentation from sklearn:

- **class sklearn.neural_network.MLPRegressor**(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)

- **class sklearn.neural_network.MLPClassifier**(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000