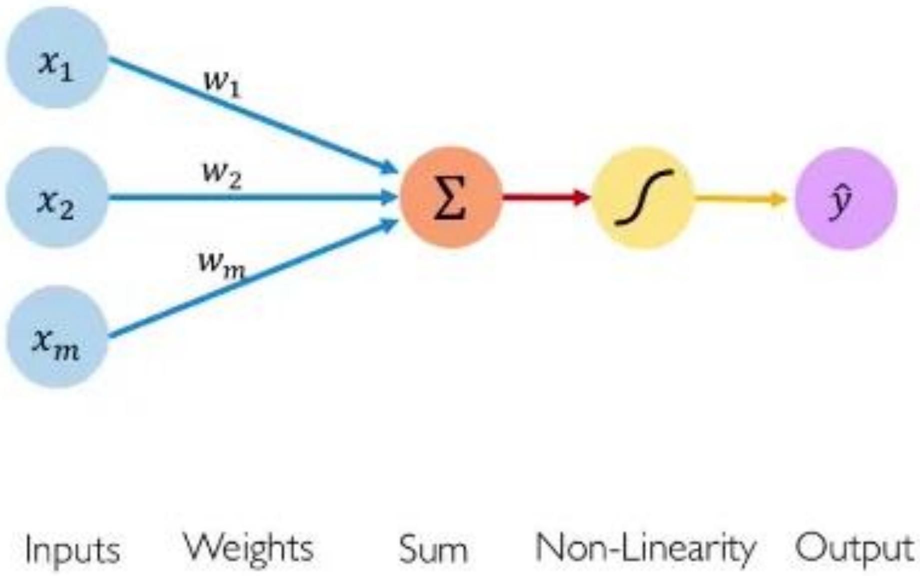
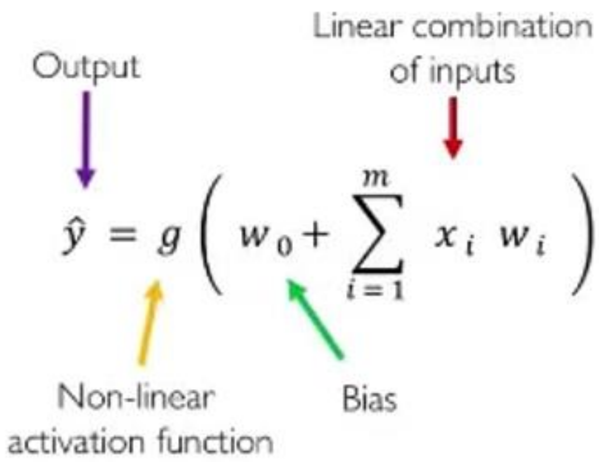
Multi Layer Perceptron

NASTARAN SHAHPARIAN |SHARCNET | COMPUTE ONTARIO |
COMPUTE CANADA |YORK UNIVERSITY

Simple Perceptron

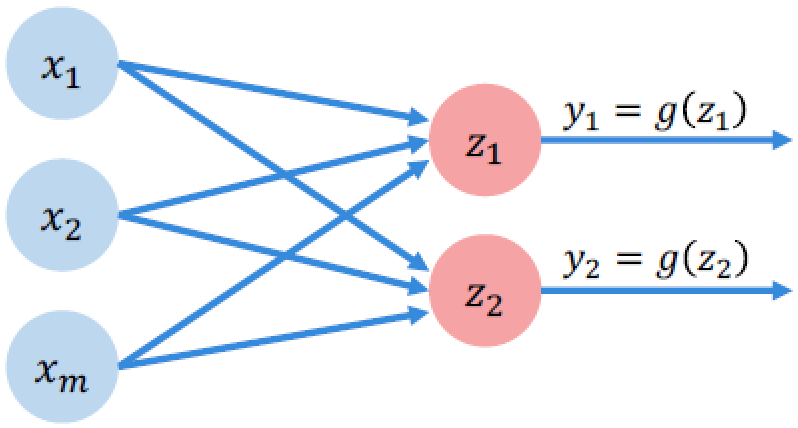
https://medium.com/analytics-vidhya/neural-network-part1-inside-a-single-neuron-fee5e44f1e





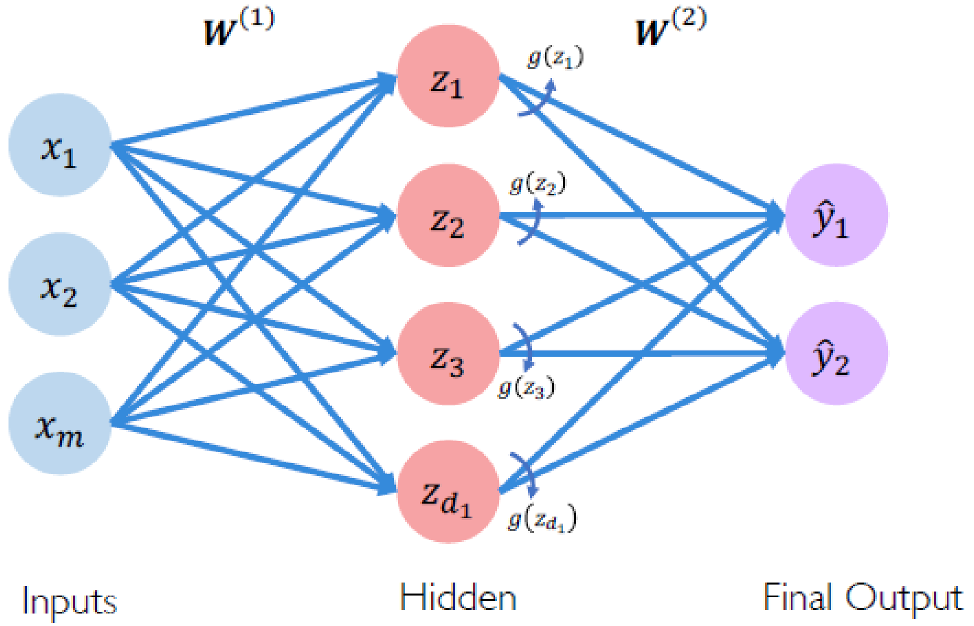
Multi Output Perceptron

Since all inputs are densely connected to outputs, this layer is called "dense" layer



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

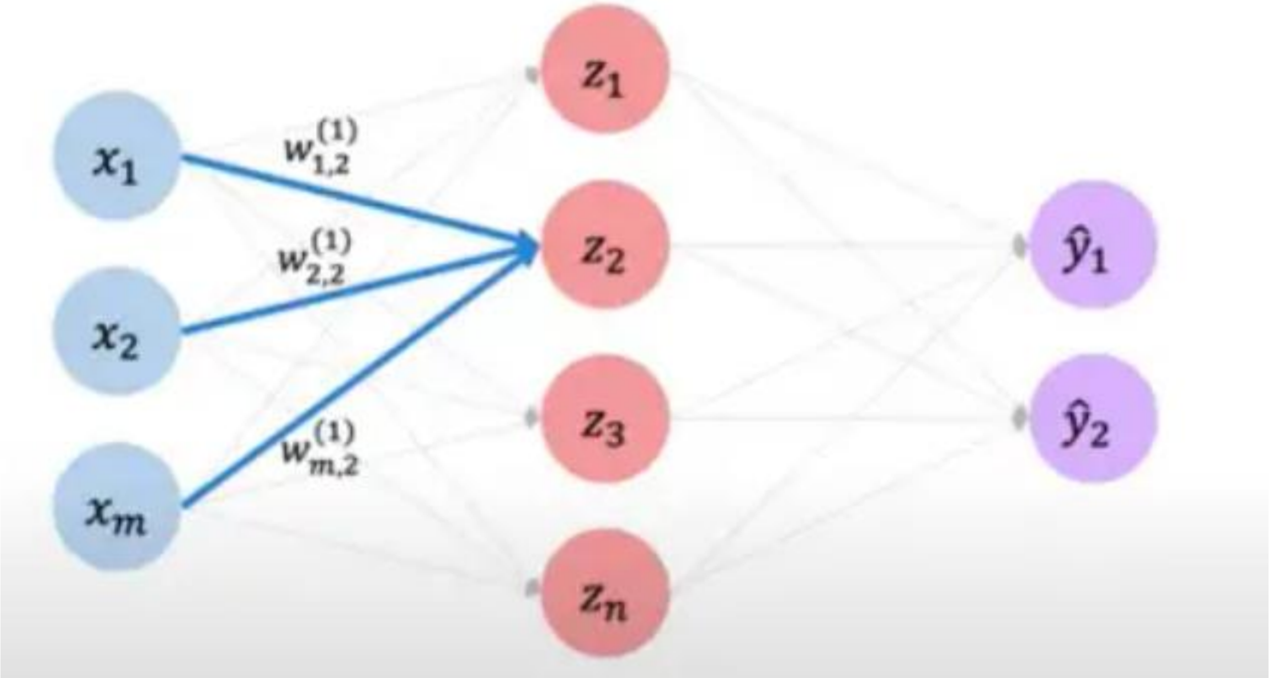
Single layer Neural network



$$Z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)}$$

$$y_i^{\wedge} = g\left(W_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) W_{j,i}^{(2)}\right)$$

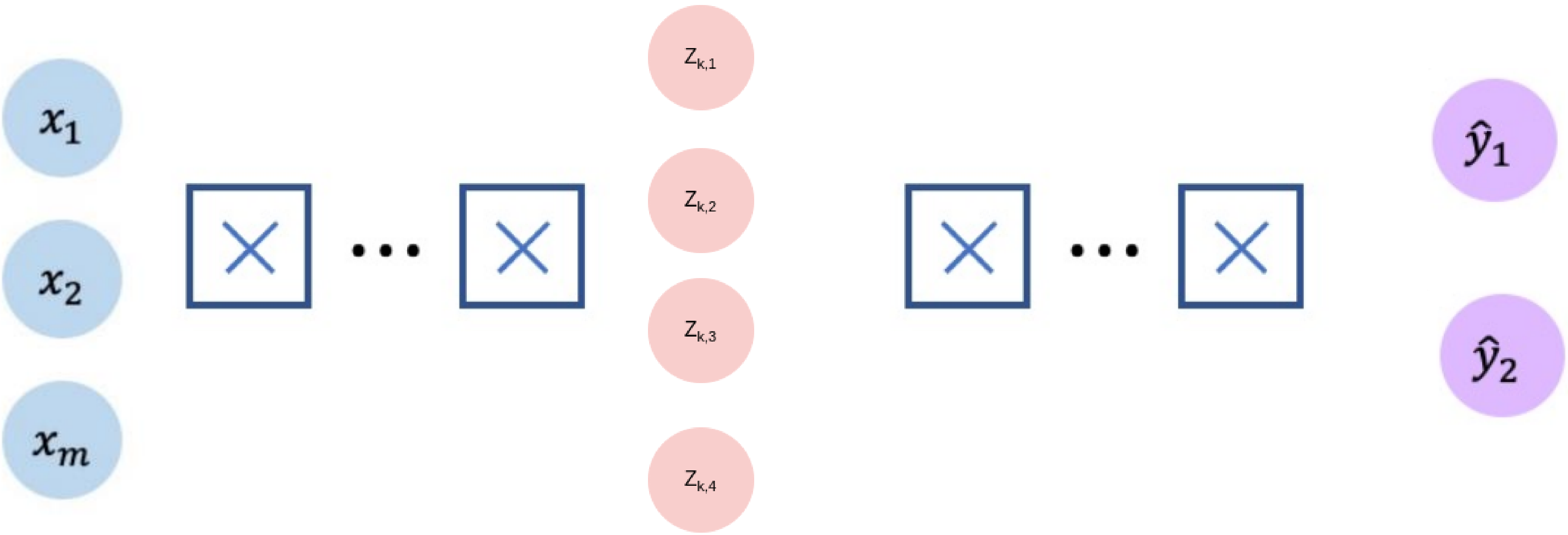
Single layer Neural Network



$$Z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, W_{j,i}^{(2)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)}$$

Deep Neural Network

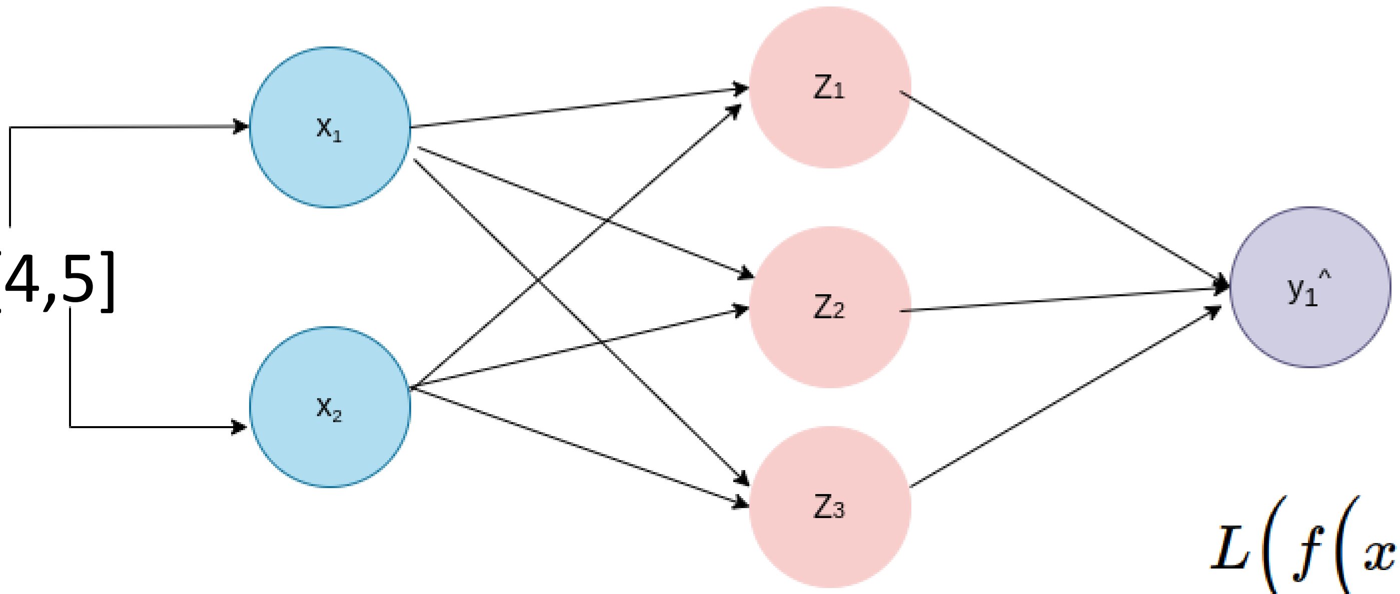


$$Z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(Z_{k-1,j}) W_{j,i}^{(k)}$$

Inputs Hidden Output

Example problem: Will I pass?

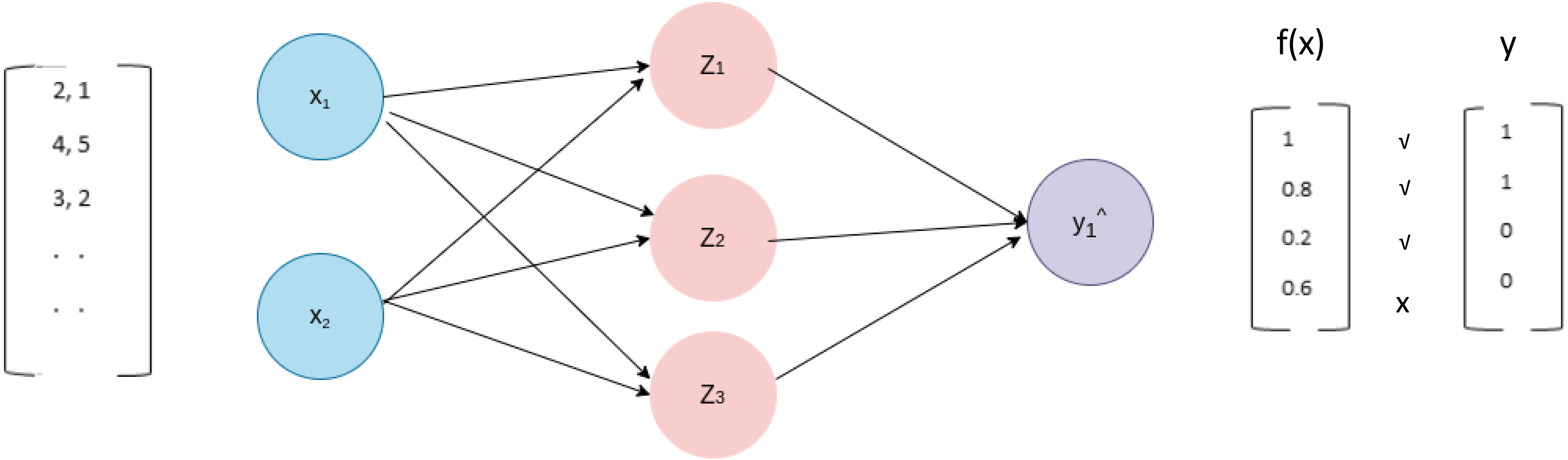
$X^{(1)} = [4,5]$



Predicted = 0.1
Actual =1

$$L\Big(\underbrace{f\big(x^{(i)};W\big)}_{\text{Predicted}},\underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

Empirical Loss



$$J(w) = \frac{1}{n}\sum_{i=1}^{n} l\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

Different Cost Functions

- Mean Square Error (MSE)

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - y_i^{\wedge})^2$$

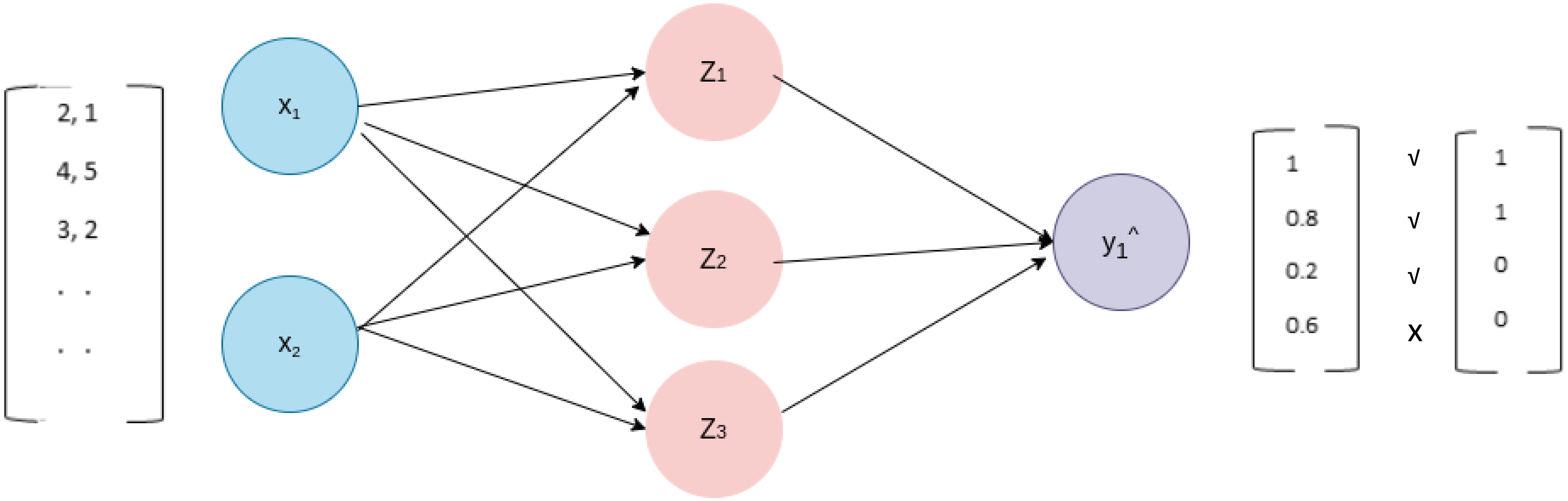
- Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} * \sum_{i=1}^{n} |y_i - y_i^{\wedge}|$$

- Cross Entropy

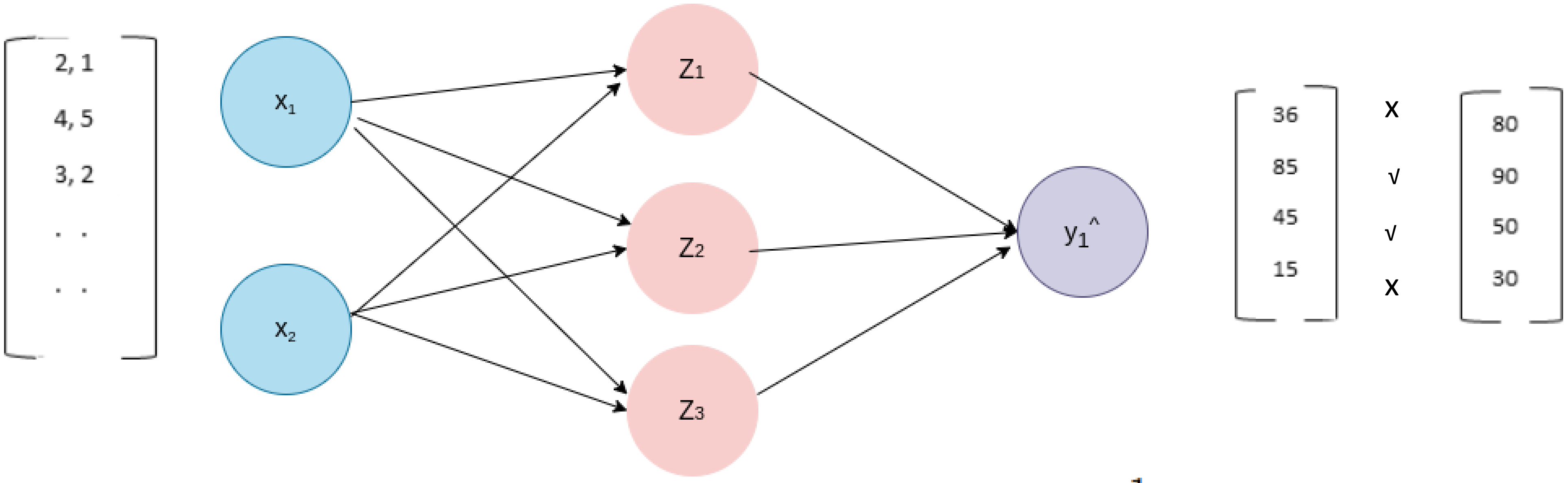
$$J(W) = -\frac{1}{n} \sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)};\ W\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; W\right)\right)$$

Binary Cross Entropy loss



$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)};\ W\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f\left(x^{(i)}; W\right)\right)$$

Mean Squared Error Loss



$$MSE = \frac{1}{n} * \sum_{i=1}^{n} \left(y_i - y_i^{\wedge}\right)^2$$

Loss Optimization

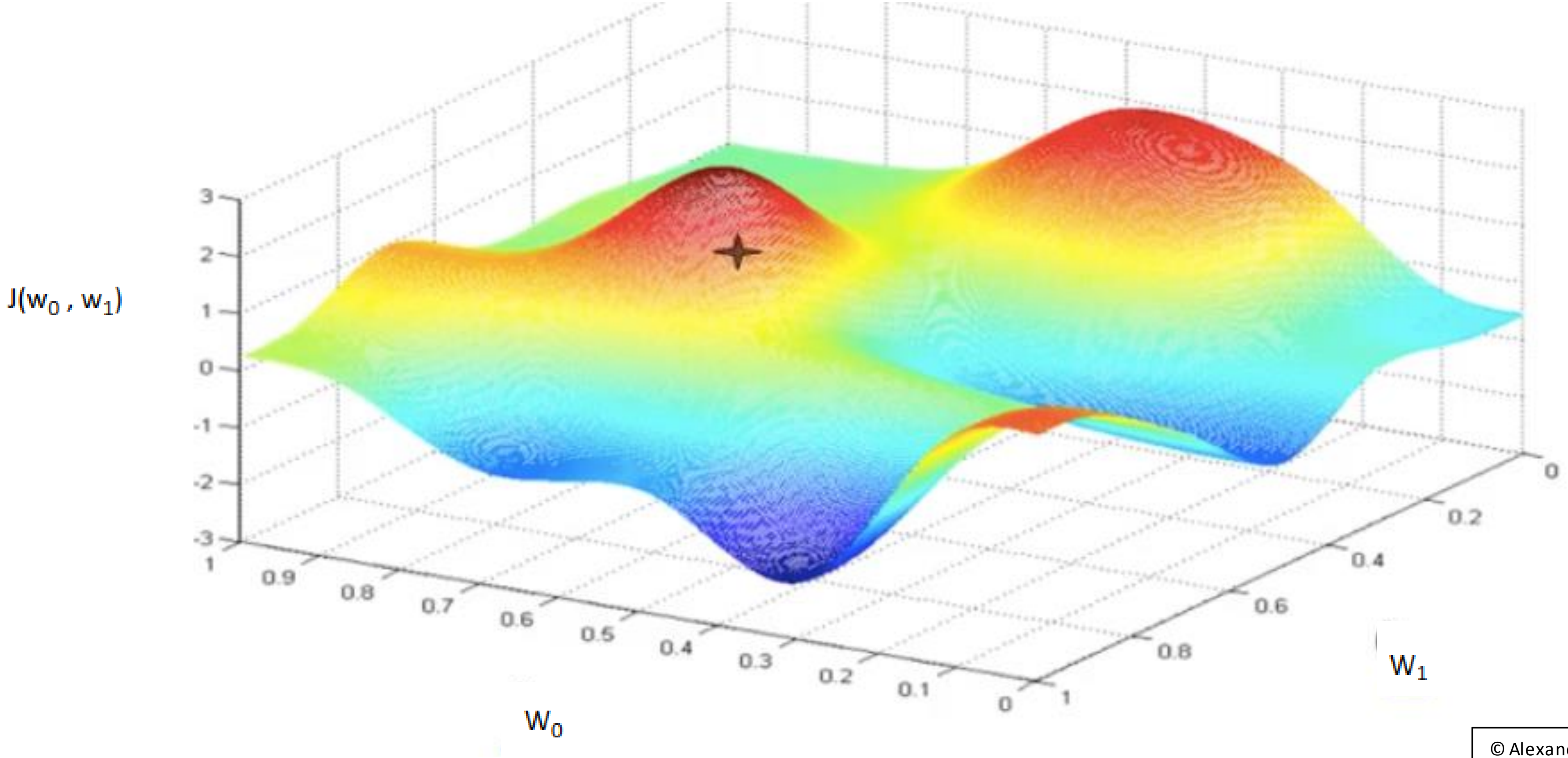
The goal is to find the weights (w) that minimize the loss

$$w^* = \text{argmin}_w \; \frac{1}{n}\sum_{i=1}^{n} l\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \text{argmin}_w \; J(w)$$

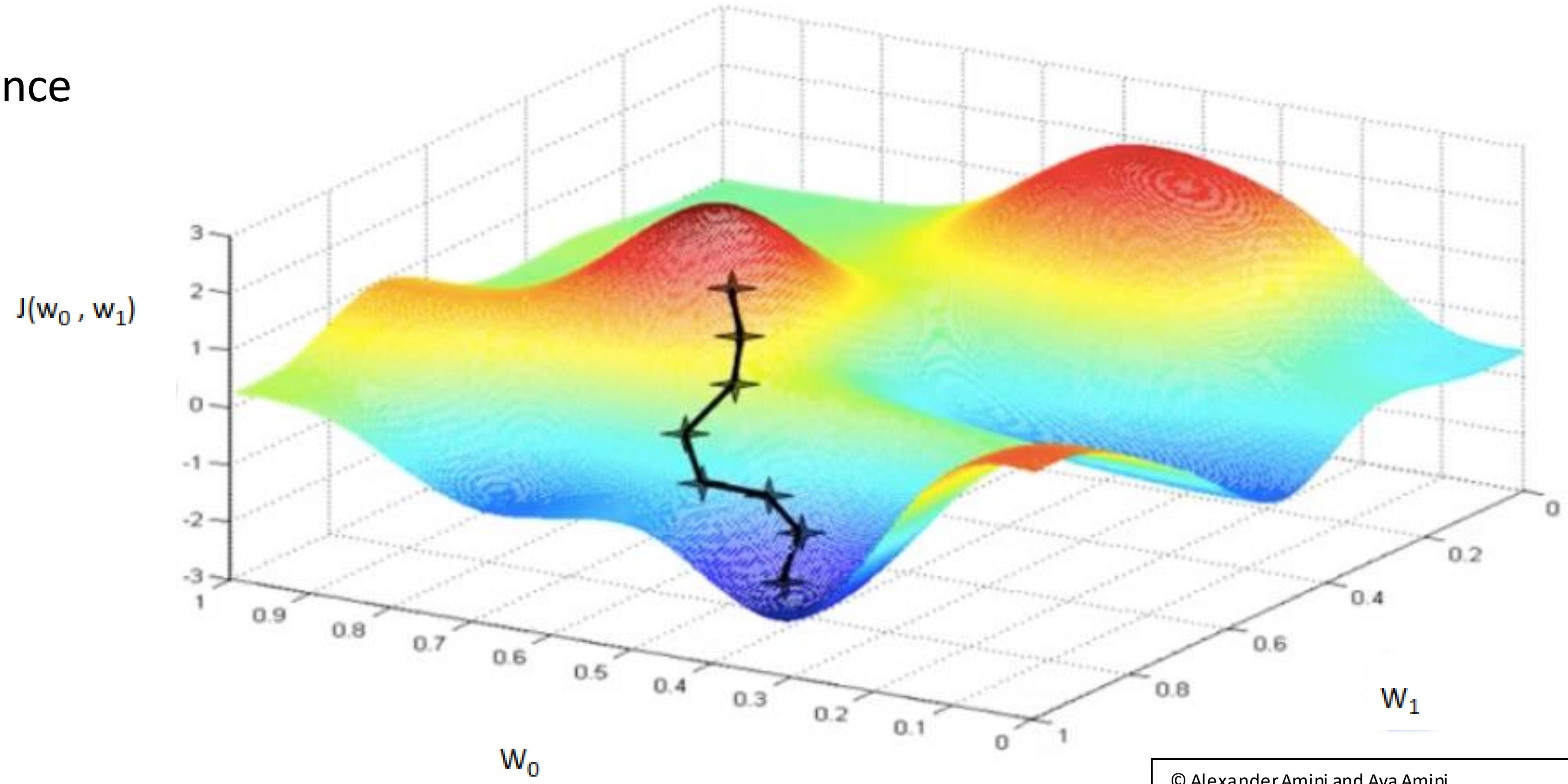
$$W = \{ W^{(0)}, W^{(1)}, \ldots\}$$

Loss Optimization



Gradient Descent

Repeat until convergence

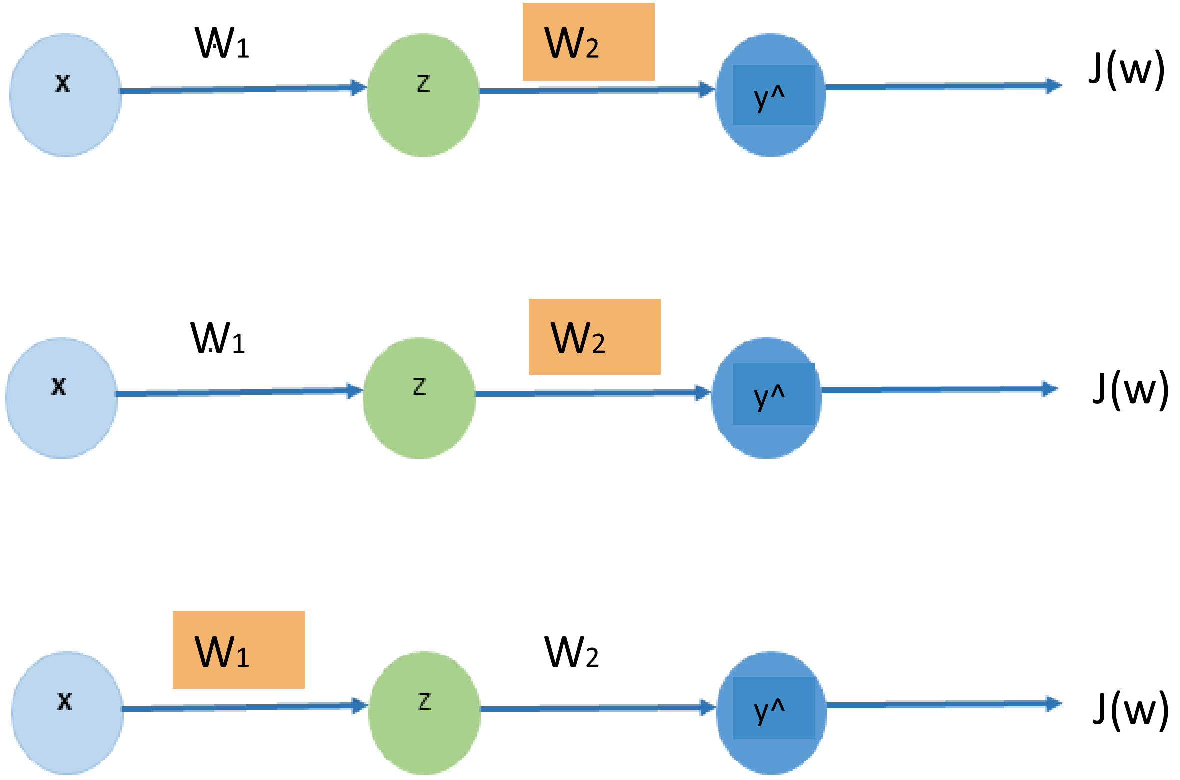


Gradient Descent

Algorithm

1. Initialize weights randomly $\sim N\left(0, \sigma^{(2)}\right)$
2. Lop until convergence
3. Compute gradient, $\frac{\partial J(w)}{\partial w}$
4. Update weights $w \longleftarrow w - \alpha \frac{\partial J(w)}{\partial w}$
5. Return weights

Gradient Descent : back Propagation



$$\frac{\partial J(w)}{\partial w_2} = ?$$

Using chain rule

$$\frac{\partial J(w)}{\partial w_2} = \frac{\partial J(w)}{\partial y^\wedge} * \frac{\partial y^\wedge}{\partial w_2}$$

$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial y^\wedge} * \frac{\partial y^\wedge}{\partial w_1}$$

$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial y^\wedge} * \frac{\partial y^\wedge}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Different Gradient Descent

- **Batch gradient descent**: Computes the gradient of the loss function with respect to the parameters using the entire training dataset at once.
- **Stochastic gradient descent (SGD)**: Updates the parameters after each individual training example. This can lead to faster convergence, but the optimization may be more noisy.
- **Mini-batch gradient descent**: In mini-batch gradient descent, the algorithm computes the gradient on a small batch of data at a time, rather than the entire dataset or a single example.
- **RMSprop:**
- **Adam: Adam (adaptive moment estimation)**

Stochastic Gradient Descent

Algorithm

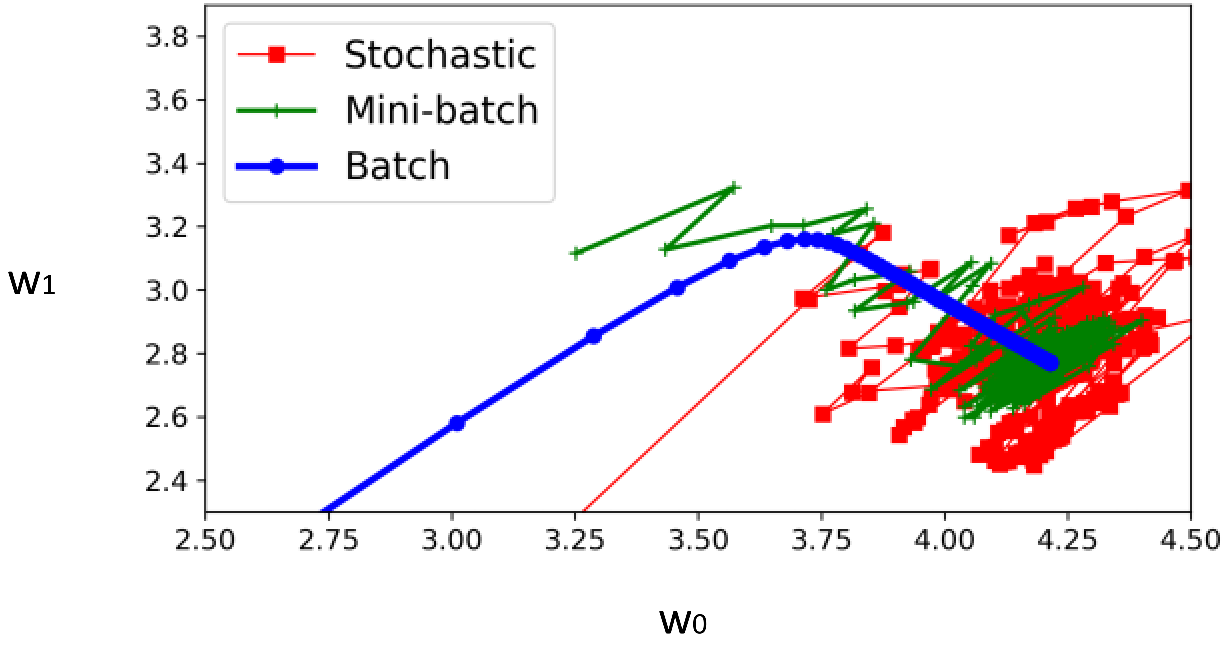
1. Initialize weights randomly $N\left(0, \sigma^{(2)}\right)$
2. Lop until convergence
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_j(w)}{\partial w}$
5. Update weights w $\longleftarrow$ w- α
6. Return weights

Mini-batch Gradient Descent

Algorithm

1. Initialize weights randomly $N\left(0, \sigma^{(2)}\right)$
2. Shuffle the training data.
3. For each mini-batch (B):
    1. Compute gradient $\frac{\partial J(w)}{\partial w} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(w)}{\partial w}$
4. Update weights $w \longleftarrow w - \alpha$

Gradient Descent Comparison



https://medium.com/analytics-vidhya/all-you-need-to-know-about-gradient-descent-f0178c19131d