Training of Multi-Layer Neural Network

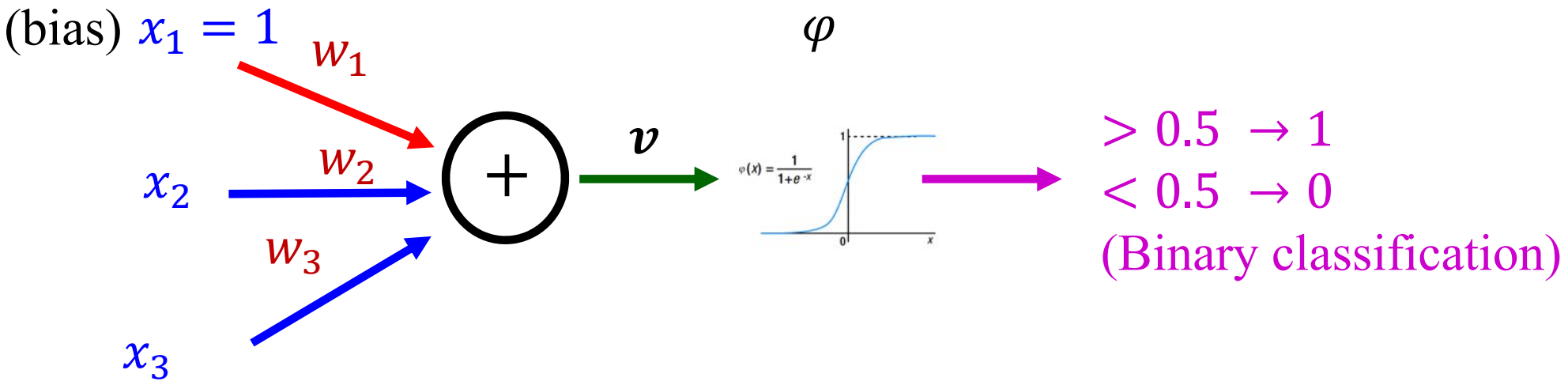
生醫光電所 吳育德

Paolo Perrotta, Programming Machine Learning: From Coding to Deep Learning (2020)

Outline

- Limitation of single layer: cannot classify nonlinearly separable data
- Multi-layer neural network, computational graph and back-propagation
- Example of input-layer-one-hidden-layer-output-layer neural network
- Initializing the weights: random and small
- Tuning hyper-parameters: batch size, learning rates, # of hidden nodes
- Multiclass classification in recognizing all MNIST characters.
- Overcome overfitting: ReLu and dropout
- Example of input-layer-three-hidden-layer-output-layer neural network

Perceptron for Binary Classification



$$v = x_1 w_1 + x_2 w_2 + x_{3 w_3} = \boldsymbol{x}\boldsymbol{w}$$

where $\boldsymbol{x} = [x_1\ x_2\ x_3]$, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$

$y = \varphi(v)$, φ is the sigmoid activation function

Minimize Mean Cross Entropy Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \frac{-1}{N}\sum_{n=1}^{N}(y_n \ln(\hat{y}_n) + (1-y_n)\ln(1-\hat{y}_n))$$

- Minimize the loss function *Loss* w.r.t w_1, w_2, w_3

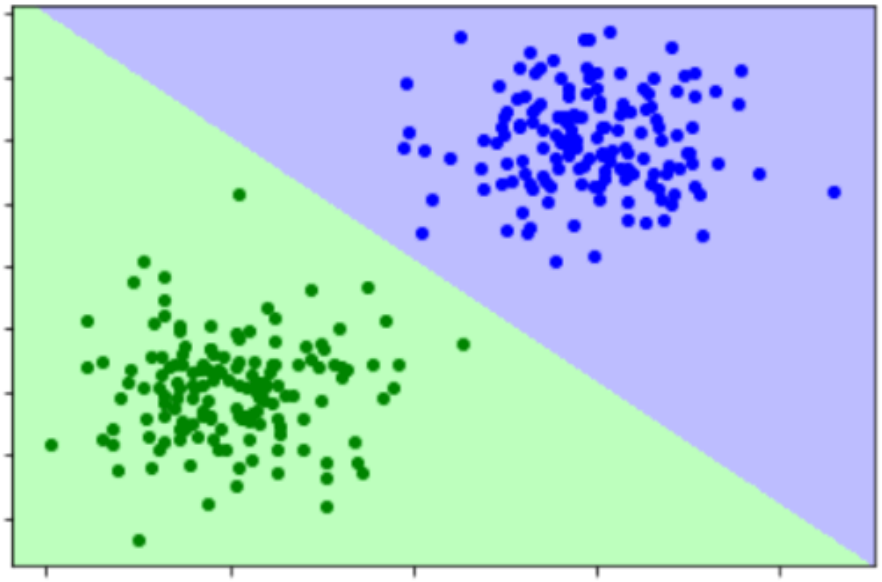
$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \frac{\partial Loss}{\partial w_3} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} e_n \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n2} \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n3} \end{bmatrix} = \frac{1}{N}[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 2}]^{\mathrm{T}}_{3\times \mathrm{N}} e_{N\times 1}$$

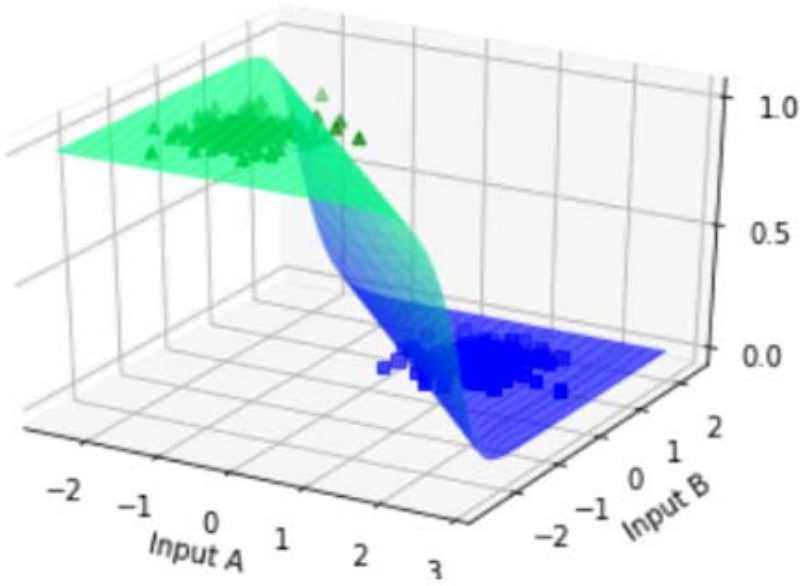
- The batch steepest (gradient) decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \frac{\partial Loss}{\partial \boldsymbol{w}}$$

Limitations of the Single Layer: 1_perceptron_classifier.jpynb

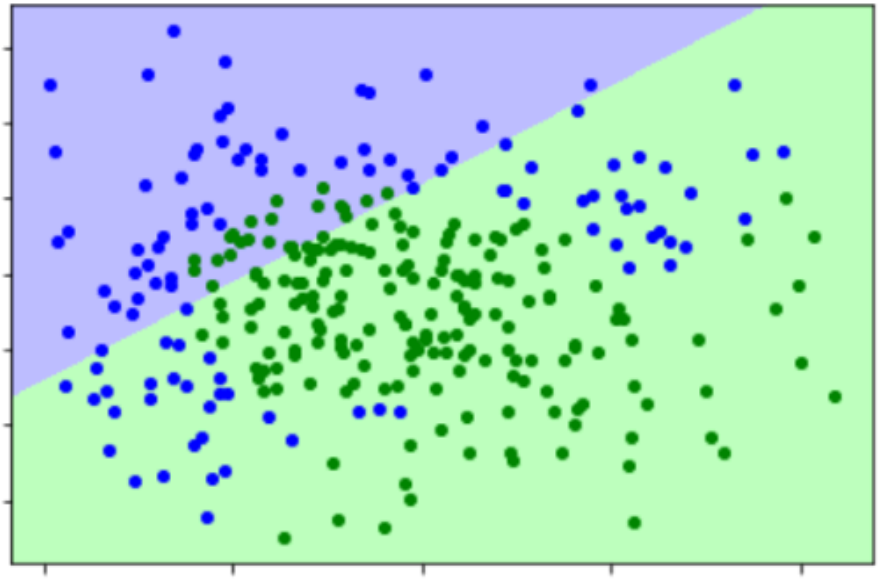
linearly_separable.txt

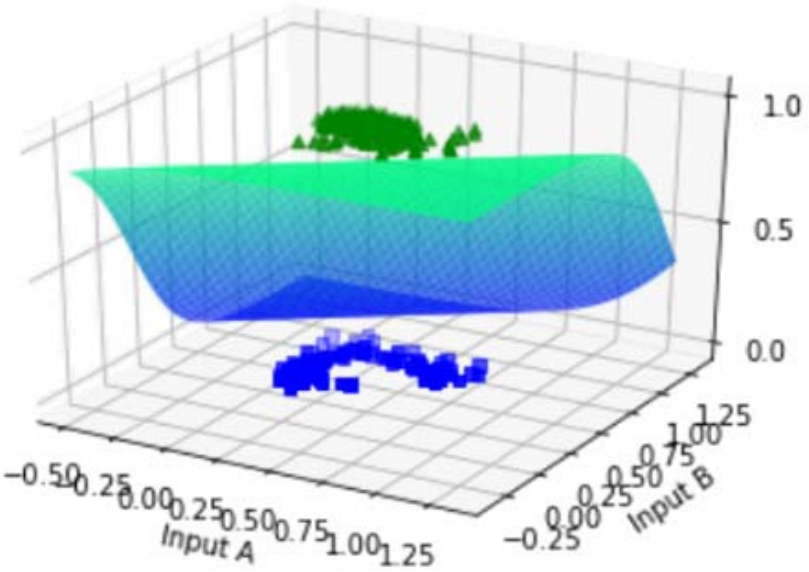




Limitations of the Single Layer: 1_perceptron_classifier.jpynb

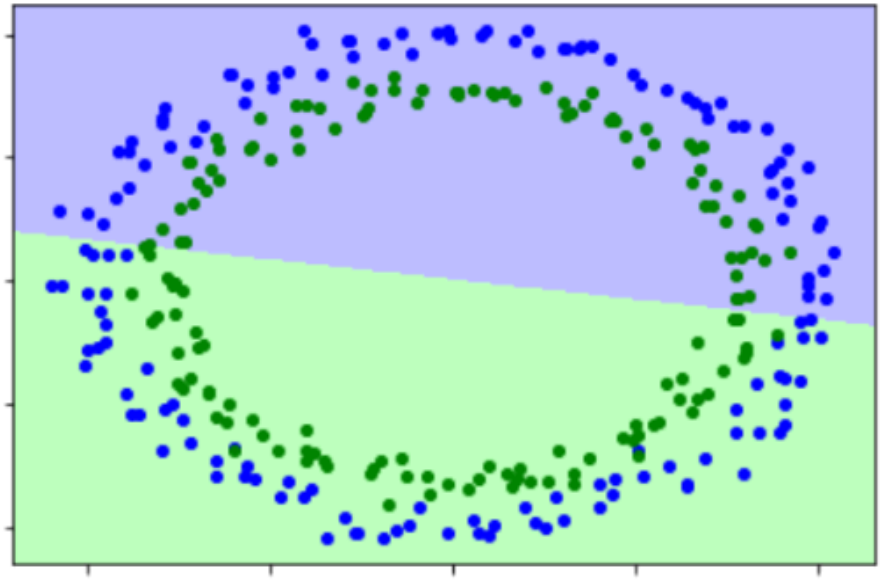
non_linearly_separable.txt

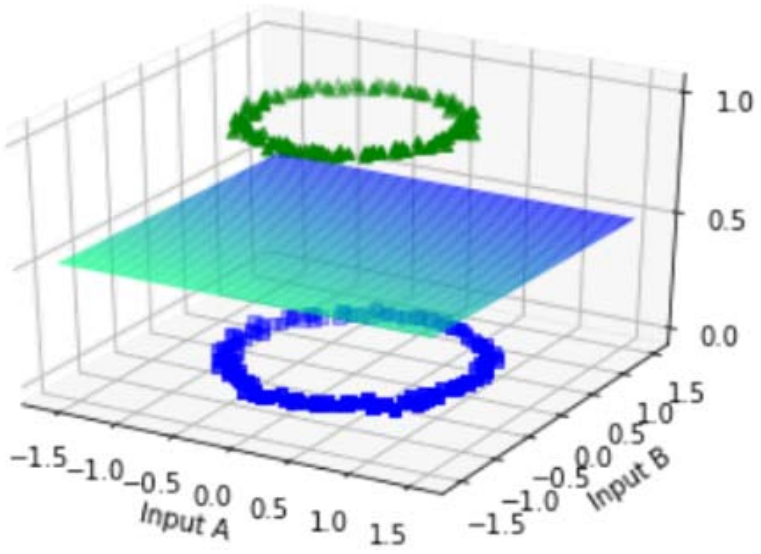




Limitations of the Single Layer: 1_perceptron_classifier.jpynb

circles.txt



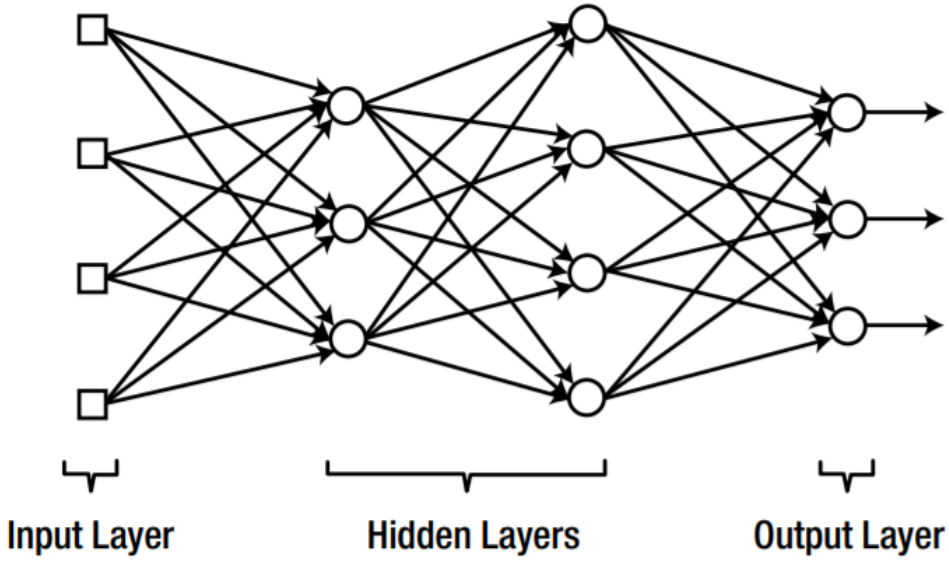


Layers of Neural Network

- The group of square nodes is called the **input layer**. They do not calculate the weighted sum and activation function.

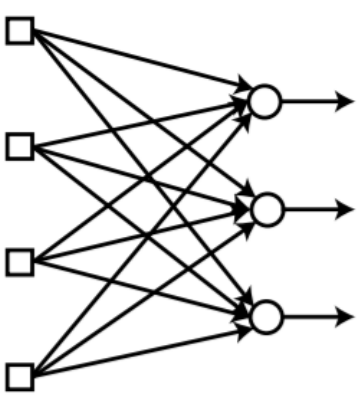
- The group of the rightmost nodes is called the **output layer**. The output from these nodes becomes the final result of the neural network.

- The layers in between the input and output layers are called **hidden layers**.

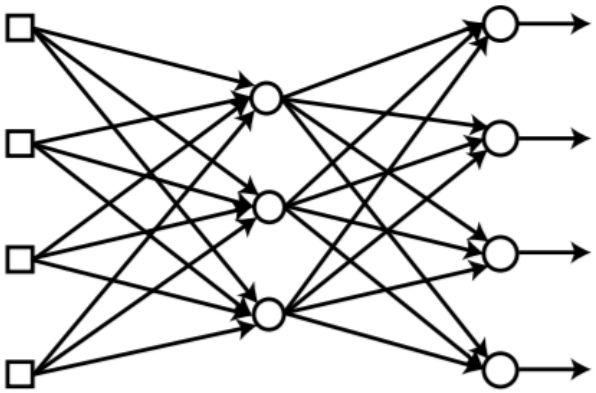


Single-Layer, Shallow, and Deep Neural Networks

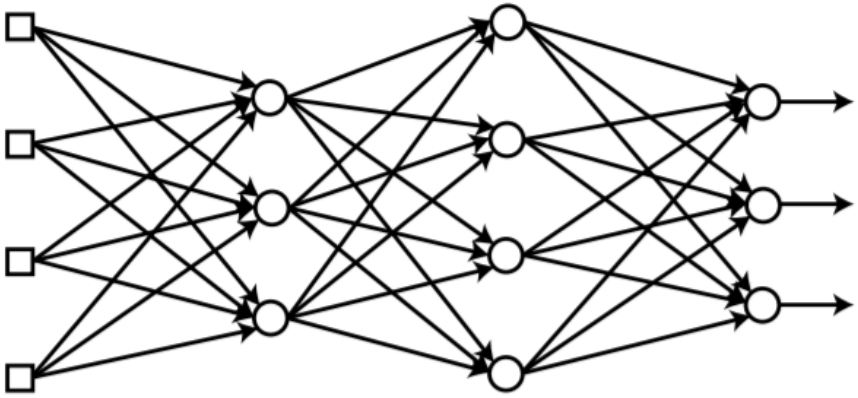
Single-Layer Neural Network		Input Layer – Output Layer
Multi-Layer Neural Network	Shallow Neural Network	Input Layer – Hidden Layer – Output Layer
	Deep Neural Network	Input Layer – Hidden Layers – Output Layers



Single-layer Neural Network



(Shallow) Multi-layer Neural Network



Deep Neural Network

Multi-layer neural network

- In an effort to overcome the practical limitations of the single-layer, the neural network evolved into a multi-layer architecture.

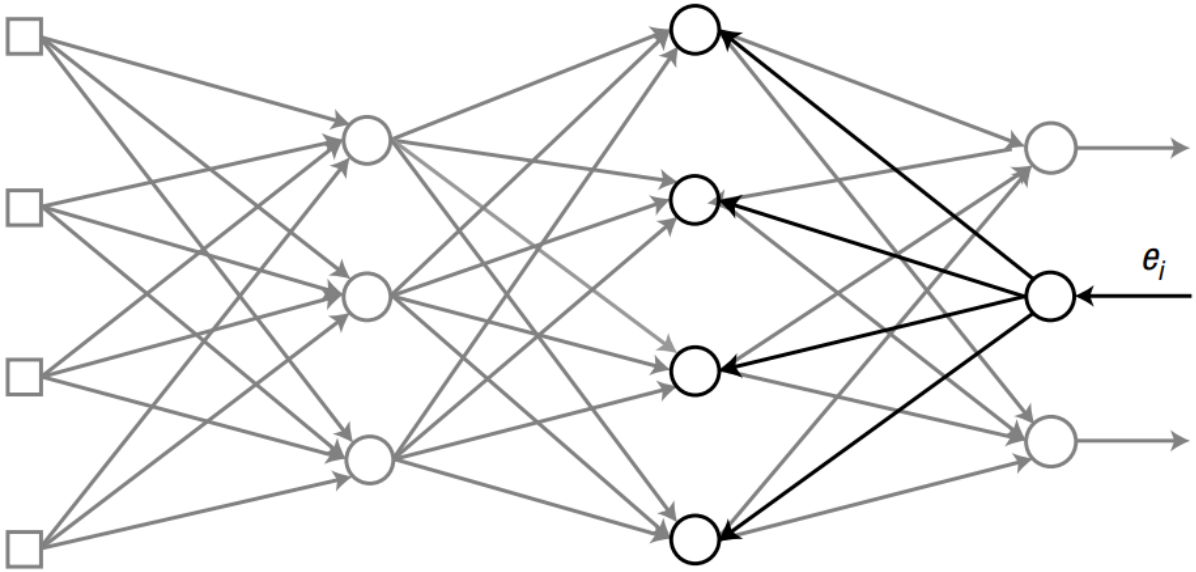
- The previously introduced method **is ineffective for training of the multi-layer neural network** because the error is **not defined in the hidden layers**.

- **Back-propagation algorithm** provided a systematic method to determine the error of the hidden nodes.

Multi-layer neural network

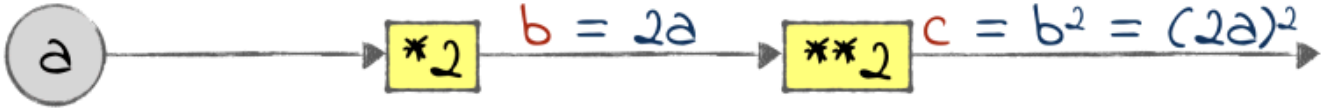
- In the back-propagation algorithm, the output error starts from the output layer and **moves backward** until it reaches the left next hidden layer to the input layer.

- In back-propagation, the signal still flows through the connecting lines and the weights are multiplied.



The Chain Rule on a Simple Network

A computational graph:



$$\frac{\partial c}{\partial a} = ?$$

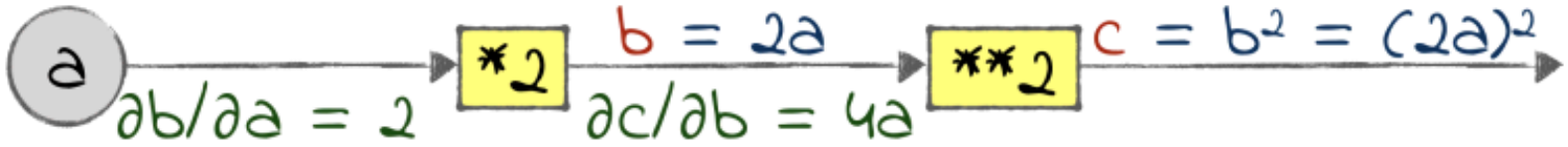
1. Walk the graph back from c to a.

2. For each operation along the way, calculate its *local gradient*—the derivative of the operation's output with respect to its input.

3. Multiply all the local gradients together.

The Chain Rule on a Simple Network

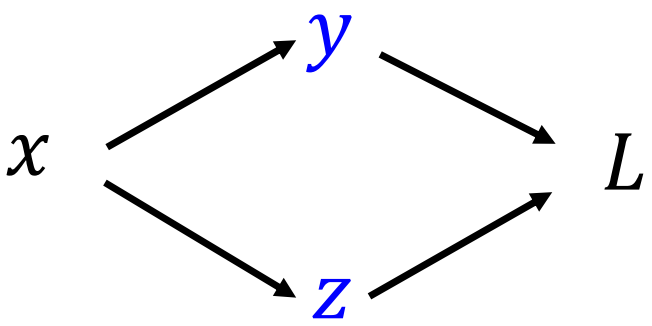
A computational graph:



$$\frac{\partial c}{\partial a} = \frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = \frac{\partial (b^2)}{\partial b}\frac{\partial (2a)}{\partial a} = 2b \times 2 = 2(2a) \times 2 = 8a$$

Chain Rule in Back Propagation

- Multivariate chain rule



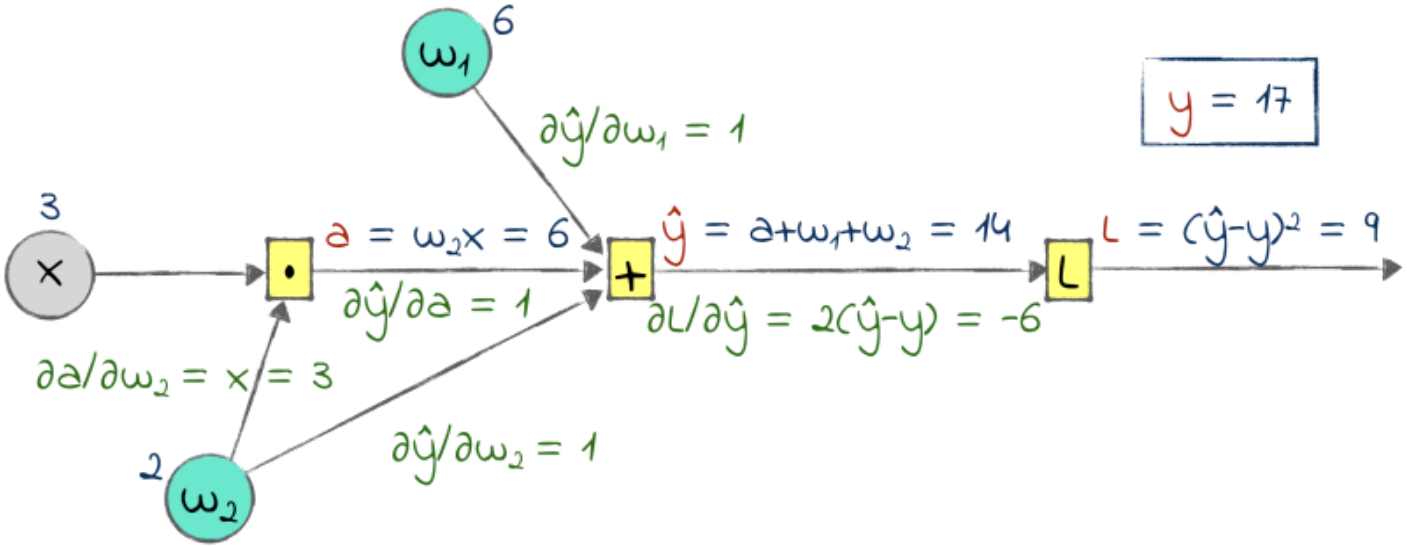
To be calculated

$$\frac{dL}{dx} = \frac{dL}{dy}\frac{dy}{dx} + \frac{dL}{dz}\frac{dz}{dx}$$

Has been computed

$$\bar{L} = \bar{y}\frac{dy}{dx} + \bar{z}\frac{dz}{dx}$$

The 2nd example of backpropagation



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1} = 2(\hat{y} - y) \times 1 = 2(14 - 17) \times 1 = -6$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a} \frac{\partial a}{\partial w_2} = -6 \times 1 + (-6) \times 1 \times 3 = -24$$

Back-Propagation Algorithm

- Consider a univariate logistic least-square model $L = \frac{1}{2}(\varphi(wx+b)-y)^2$

$$\frac{\partial L}{\partial w} = \frac{1}{2}\frac{\partial}{\partial w}(\varphi(wx+b)-y)^2$$
$$= (\varphi(wx+b)-y) \times \frac{\partial}{\partial w}(\varphi(wx+b)-y)$$
$$= (\varphi(wx+b)-y) \times \varphi'(wx+b) \times \frac{\partial}{\partial w}(wx+b)$$
$$= (\varphi(wx+b)-y) \times \varphi'(wx+b) \times x$$

$$\frac{\partial L}{\partial b} = \frac{1}{2}\frac{\partial}{\partial b}(\varphi(wx+b)-y)^2$$
$$= (\varphi(wx+b)-y) \times \frac{\partial}{\partial b}(\varphi(wx+b)-y)$$
$$= (\varphi(wx+b)-y) \times \varphi'(wx+b) \times \frac{\partial}{\partial b}(wx+b)$$
$$= (\varphi(wx+b)-y) \times \varphi'(wx+b)$$

- Disadvantages: Cumbersome calculation
  Two derivations are nearly identical (redundant)
  Repeated terms

Back-Propagation Algorithm

- Consider a univariate logistic least-square model $L = \frac{1}{2}(\varphi(xw + b) - y)^2$
- A more structural way

Compute the loss

$z = xw + b$

$\hat{y} = \varphi(z)$

$L = \frac{1}{2}(\hat{y} - y)^2$

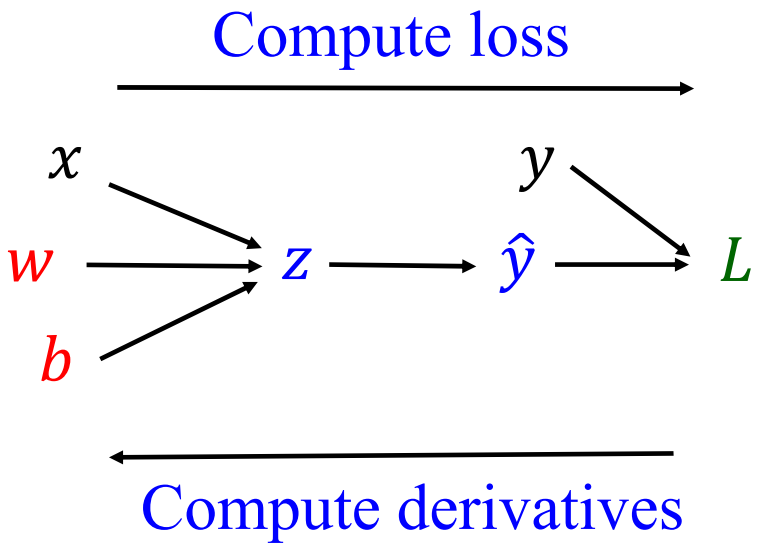
Compute the derivatives:

$\bar{y} \equiv \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$

$\bar{z} \equiv \frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \bar{y}\, \varphi'(z)$

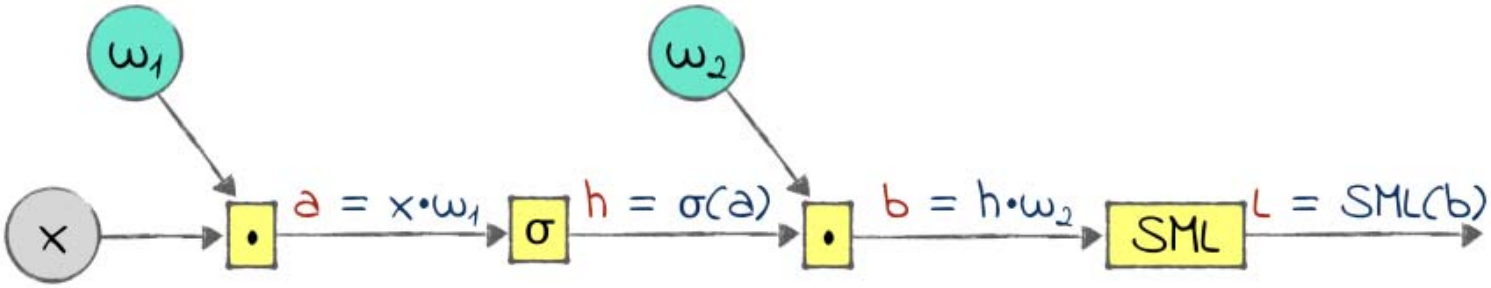
$\bar{w} \equiv \frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w} = \bar{z}\, x$

$\bar{b} \equiv \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial b} = \bar{z}\, 1$



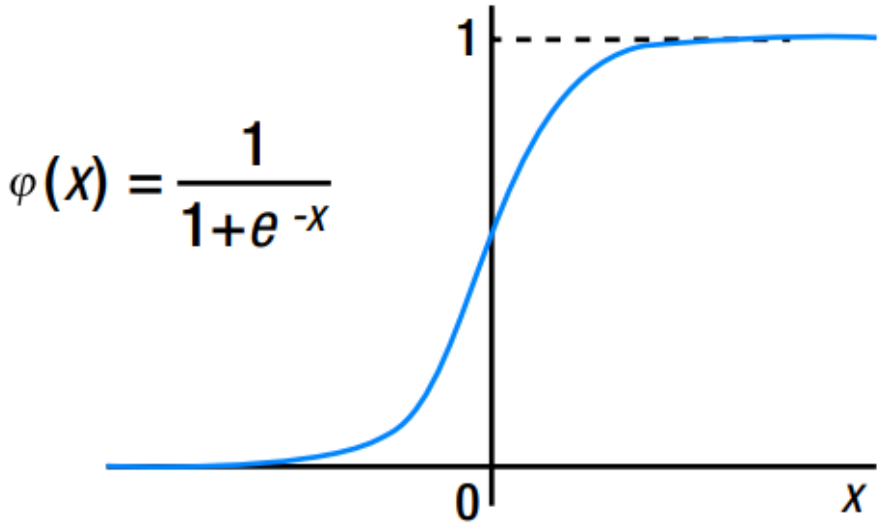
- $\bar{y}$, $\bar{z}$, $\bar{w}$ and $\bar{b}$ are computed by program

Forward of a simplified Multilayer Network



$$L = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}),$$
$$\hat{y} = softmax(b) = \sigma(b)$$

Derivative of the sigmoid function



$$\frac{d(1+e^{-x})^{-1}}{dx} = -(1+e^{-x})^{-2}(-e^{-x}) = \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right)$$

$$\varphi'(x) = \varphi(x)\big(1-\varphi(x)\big)$$

Derivative of the Softmax function

$$y_i = \sigma(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \cdots + e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}} = e^{v_i}\left(\sum_{k=1}^{M} e^{v_k}\right)^{-1}$$

$$\frac{\partial y_i}{\partial v_i} = \frac{\partial e^{v_i}}{\partial v_i}\left(\sum_{k=1}^{M} e^{v_k}\right)^{-1} + e^{v_i}\frac{\partial \left(\sum_{k=1}^{M} e^{v_k}\right)^{-1}}{\partial v_i},$$

$$= \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}} - \frac{e^{v_i} e^{v_i}}{\left(\sum_{k=1}^{M} e^{v_k}\right)^2}$$

$$= \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}\left(1 - \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}\right)$$

$$= \sigma(v_i)\big(1 - \sigma(v_i)\big)$$

$$= y_i(1 - y_i)$$

note $\frac{d(f(x)g(x))}{dx} = \frac{d(f(x))}{dx}g(x) + f(x)\frac{d(g(x))}{dx}$

Derivative of the Binary Cross-Entropy Loss

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial(-y\ln(\hat{y})-(1-y)\ln(1-\hat{y}))}{\partial \hat{y}}$$

$$= -y\frac{1}{\hat{y}} - (1-y)\frac{-1}{1-\hat{y}}$$

$$= \frac{-y(1-\hat{y})+(1-y)\hat{y}}{\hat{y}(1-\hat{y})}$$

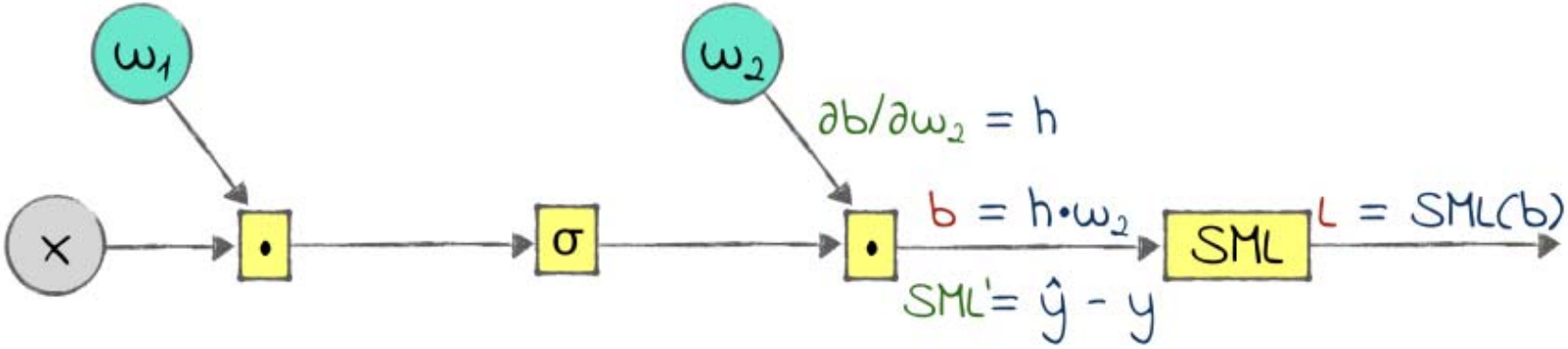
$$= \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})}$$

$$\hat{y} = \sigma(b)$$

$$\frac{\partial \hat{y}}{\partial b} = \sigma(b)\big(1-\sigma(b)\big) = \hat{y}(1-\hat{y})$$

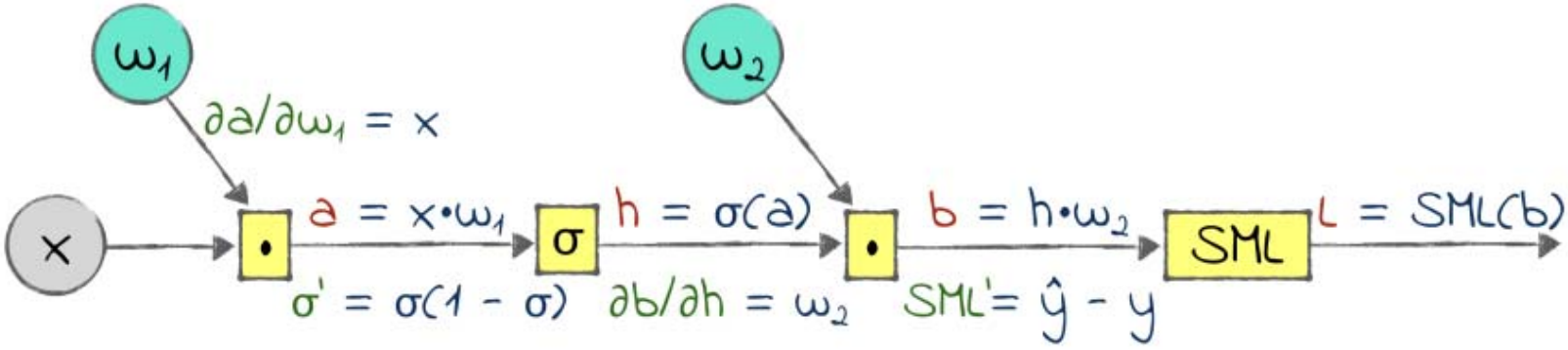
$$\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \times \hat{y}(1-\hat{y}) = -(y-\hat{y}) = \hat{y}-y = \sigma(b)-y \to e$$

Backpropagation of a simplified Multilayer Network



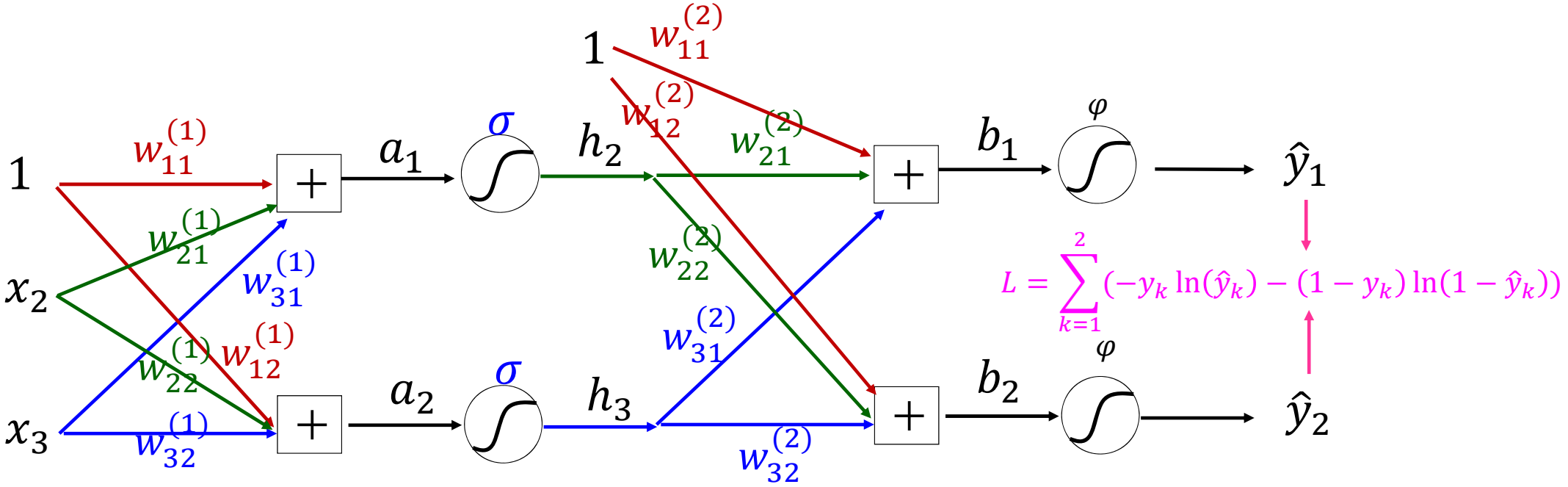
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} \frac{\partial b}{\partial w_2} = (\hat{y} - y)h$$

Backpropagation of a simplified Multilayer Network



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} \frac{\partial b}{\partial h} \frac{\partial h}{\partial a} \frac{\partial a}{\partial w_1} = (\hat{y} - y) w_2 \sigma(a)(1 - \sigma(a)) x$$

Multi-layer Neural Network: forward pass



$$[a_1 \quad a_2] = [1 \quad x_2 \quad x_3] \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} \triangleq [1\ X]W_1$$

$$[h_2 \quad h_3] = [\sigma(a_1) \quad \sigma(a_2)] \triangleq H$$

$$[b_1 \quad b_2] = [1 \quad h_2 \quad h_3] \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} \triangleq [1\ H]W_2$$

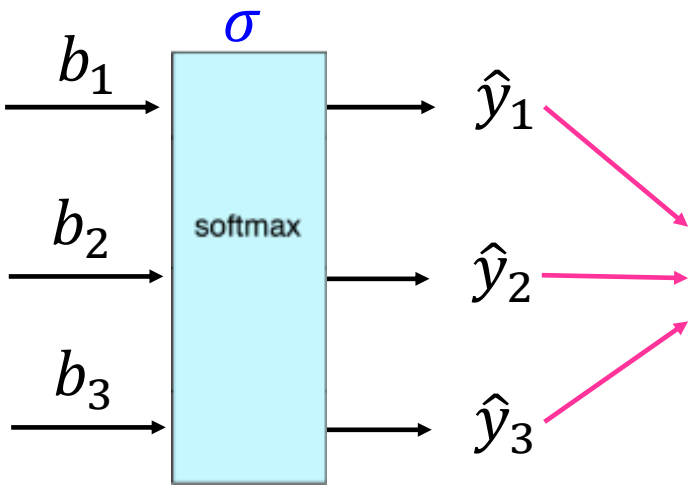
$$[\hat{y}_1 \quad \hat{y}_2] = [\varphi(b_1) \quad \varphi(b_2)] \triangleq \hat{Y}$$

Update weights using Gradient Descent

$$\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} \leftarrow \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(2)}} & \frac{\partial L}{\partial w_{12}^{(2)}} \\ \frac{\partial L}{\partial w_{21}^{(2)}} & \frac{\partial L}{\partial w_{22}^{(2)}} \\ \frac{\partial L}{\partial w_{31}^{(2)}} & \frac{\partial L}{\partial w_{32}^{(2)}} \end{bmatrix}$$

$$\begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial w_{12}^{(1)}} \\ \frac{\partial L}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial w_{22}^{(1)}} \\ \frac{\partial L}{\partial w_{31}^{(1)}} & \frac{\partial L}{\partial w_{32}^{(1)}} \end{bmatrix}$$

Derivative of the Categorical Cross-Entropy Loss



$$L(b_1, b_2, b_2) = -\sum_{k=1}^{3} y_k \ln(\hat{y}_k),$$

One-Hot Encoding

$$\begin{aligned}[y_1 \quad y_2 \quad y_3] \\ = [1 \quad 0 \quad 0], \\ = [0 \quad 1 \quad 0], \\ = [0 \quad 0 \quad 1]\end{aligned}$$

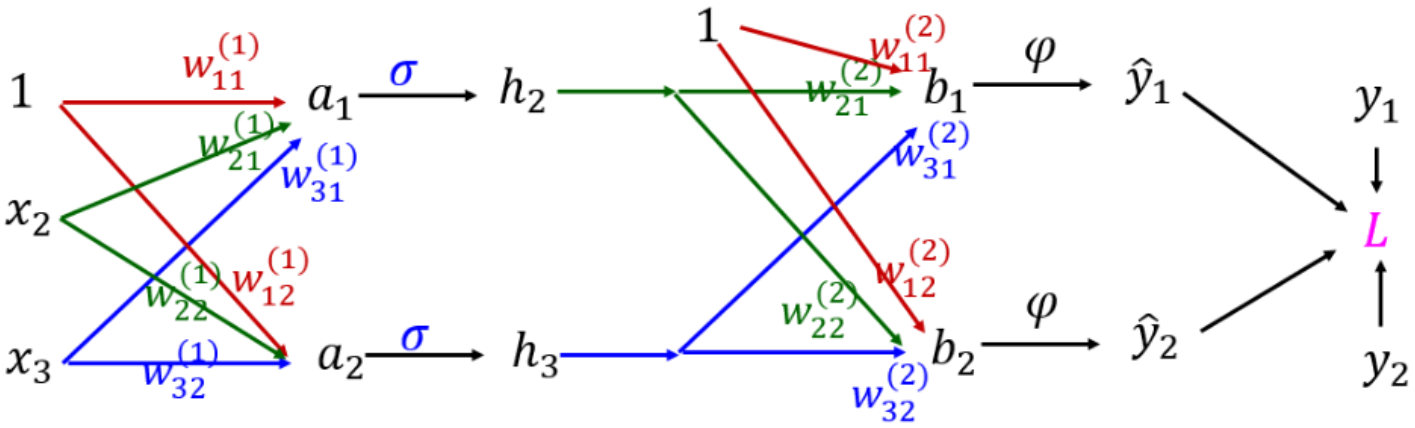
$$\frac{\partial L}{\partial b_j} = \frac{\partial}{\partial b_j}\left(-\sum_{k=1}^{3} y_k \ln(\hat{y}_k)\right) = -\sum_{k=1}^{3} y_k \frac{\partial \ln(\hat{y}_k)}{\partial b_j},$$

$$\hat{y}_k = \frac{e^{b_k}}{e^{b_1} + e^{b_2} + e^{b_3}}$$

$$\ln(\hat{y}_k) = b_k - \ln\left(\sum_{i=1}^{3} e^{b_i}\right) \Rightarrow \frac{\partial \ln(\hat{y}_k)}{\partial b_j} = \frac{\partial b_k}{\partial b_j} - \frac{\partial}{\partial b_j}\ln\left(\sum_{i=1}^{3} e^{b_i}\right) = 1\{k = j\} - \frac{e^{b_j}}{\sum_{i=1}^{3} e^{b_i}}$$

$$\Rightarrow \frac{\partial L}{\partial b_j} = -\sum_{k=1}^{3} y_k \left(1\{k = j\} - \hat{y}_j\right) = -\sum_{k=1}^{3} y_k 1\{k = j\} + \hat{y}_j \sum_{k=1}^{3} y_k = -y_j + \hat{y}_j \times 1 \equiv e_j$$

Backpropagation: Gradient of W_2 in matrix form

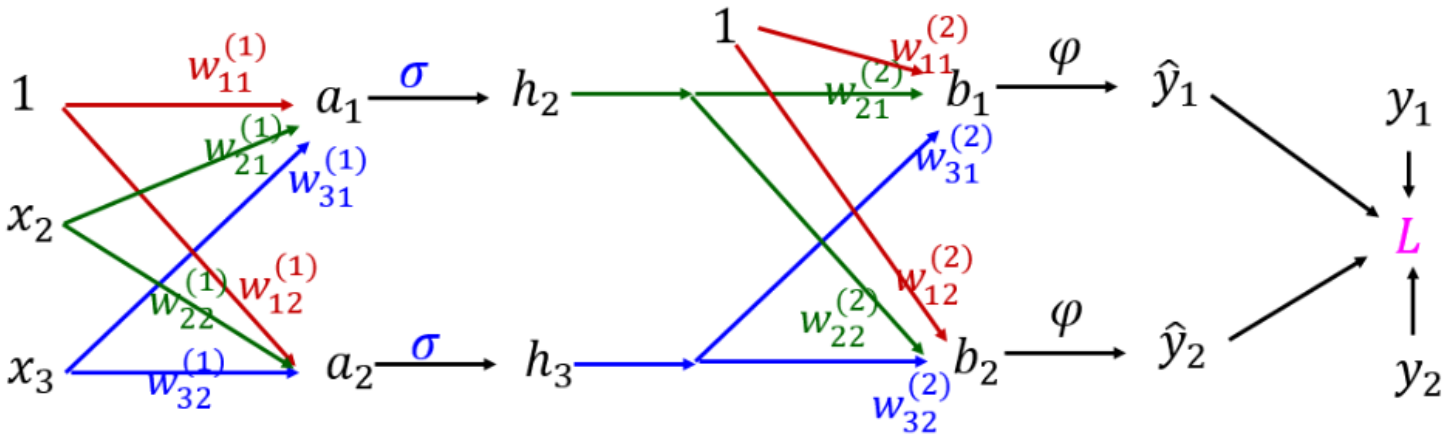


$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial b_1} = \hat{y}_1 - y_1,$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial b_2} = \hat{y}_2 - y_2$$

$$\begin{bmatrix} \frac{\partial L}{\partial b_1} & \frac{\partial L}{\partial b_2} \end{bmatrix} = [\hat{y}_1 - y_1 \quad \hat{y}_2 - y_2] = \hat{\mathrm{Y}} - \mathrm{Y}, \qquad \mathrm{Y} \triangleq [y_1 \quad y_2]$$

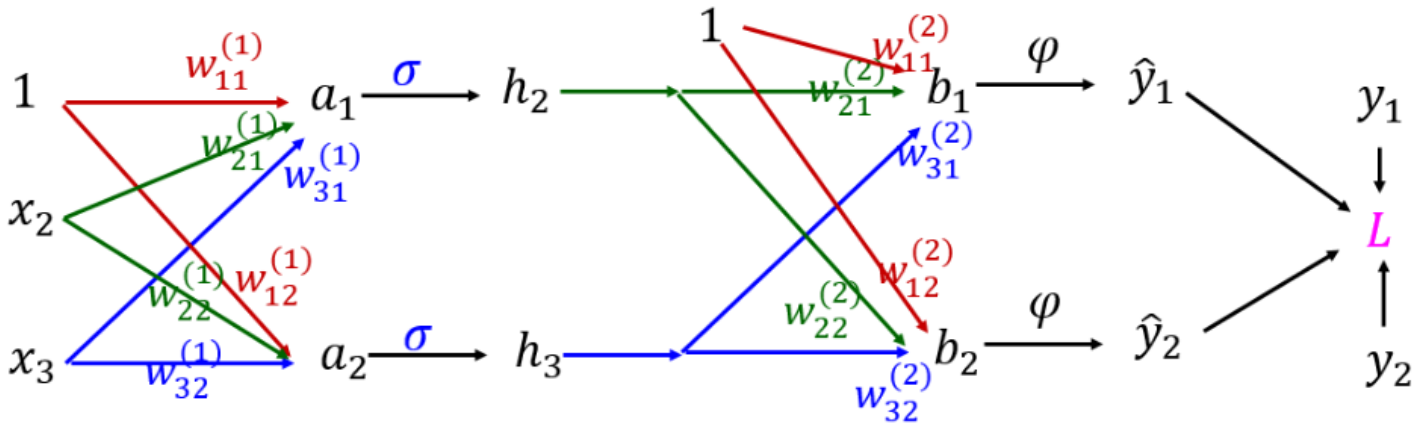
Backpropagation: Gradient of W_2 in matrix form



$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(2)}} & \frac{\partial L}{\partial w_{12}^{(2)}} \\ \frac{\partial L}{\partial w_{21}^{(2)}} & \frac{\partial L}{\partial w_{22}^{(2)}} \\ \frac{\partial L}{\partial w_{31}^{(2)}} & \frac{\partial L}{\partial w_{32}^{(2)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{11}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{12}^{(2)}} \\ \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{21}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{22}^{(2)}} \\ \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{31}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{32}^{(2)}} \end{bmatrix} = \begin{bmatrix} 1 \\ h_2 \\ h_3 \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial b_1} & \frac{\partial L}{\partial b_2} \end{bmatrix} = \begin{bmatrix} 1 \\ h_2 \\ h_3 \end{bmatrix} [\hat{y}_1 - y_1 \quad \hat{y}_2 - y_2] = [1\ H]^T(\hat{Y} - Y)$$

dW2 = np.matmul(prepend_bias(h).T, Y_hat-Y)/X.shape[0]

Backpropagation: Gradient of W_1 in matrix form



$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial h_2} + \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial h_2} = (\hat{y}_1 - y_1)w_{21}^{(2)} + (\hat{y}_2 - y_2)w_{22}^{(2)}$$

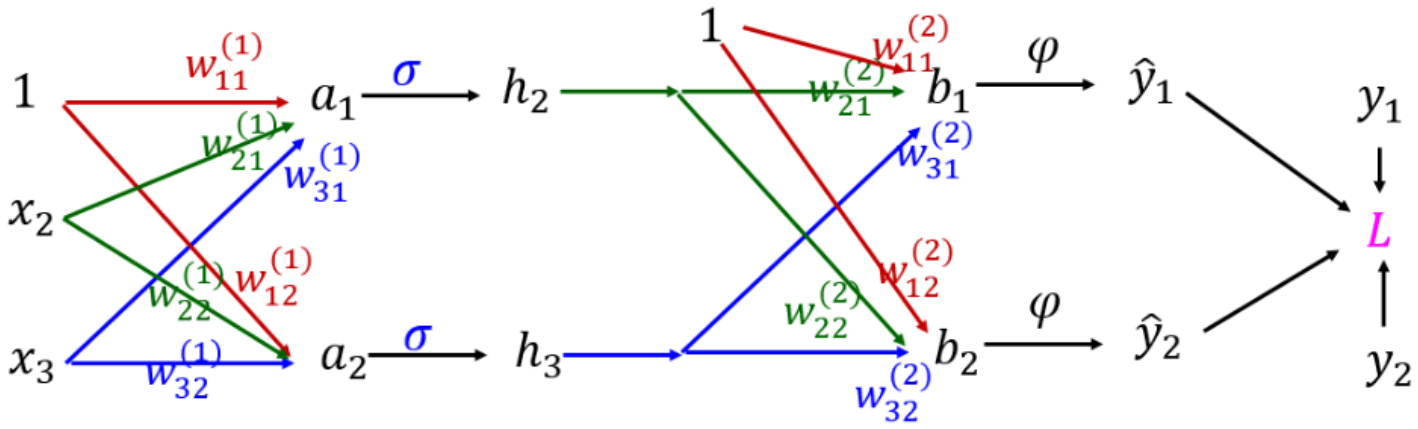
$$\frac{\partial L}{\partial h_3} = \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial h_3} + \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial h_3} = (\hat{y}_1 - y_1)w_{31}^{(2)} + (\hat{y}_2 - y_2)w_{32}^{(2)}$$

$$\begin{bmatrix}\frac{\partial L}{\partial h_2} & \frac{\partial L}{\partial h_3}\end{bmatrix} = [\hat{y}_1 - y_1 \quad \hat{y}_2 - y_2]\begin{bmatrix} w_{21}^{(2)} & w_{31}^{(2)} \\ w_{22}^{(2)} & w_{32}^{(2)}\end{bmatrix} = (\hat{\mathrm{Y}} - \mathrm{Y})W_2[1:]^T$$

Note $W_2[1:]$ represents all the columns from the second column $= W_2[1:2]$

dh=np.matmul(Y_hat – Y, $W_2[1:]$.T)

Backpropagation: Gradient of W_1 in matrix form



$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_1} = \frac{\partial L}{\partial h_2}\sigma'(h_2), \qquad \sigma'(h_2) = \sigma(h_2)(1-\sigma(h_2))$$

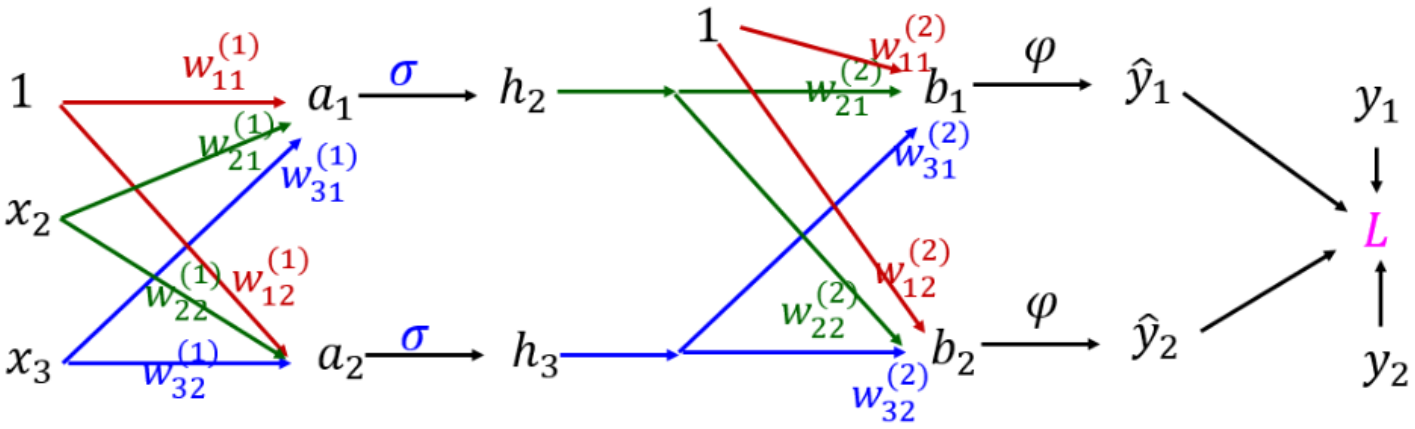
$$\frac{\partial L}{\partial a_2} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial a_2} = \frac{\partial L}{\partial h_3}\sigma'(h_3), \qquad \sigma'(h_3) = \sigma(h_3)(1-\sigma(h_3))$$

$$\left[\frac{\partial L}{\partial a_1} \quad \frac{\partial L}{\partial a_2}\right] = \left[\frac{\partial L}{\partial h_2} \quad \frac{\partial L}{\partial h_3}\right] * [\sigma'(h_2) \quad \sigma'(h_3)] = (\hat{\mathrm{Y}} - \mathrm{Y})W_2[1:]^T * [\sigma'(h_2) \quad \sigma'(h_3)]$$

Point-wise multiplication

da=dh $*\sigma'(H)$

Backpropagation: Gradient of W_1 in matrix form

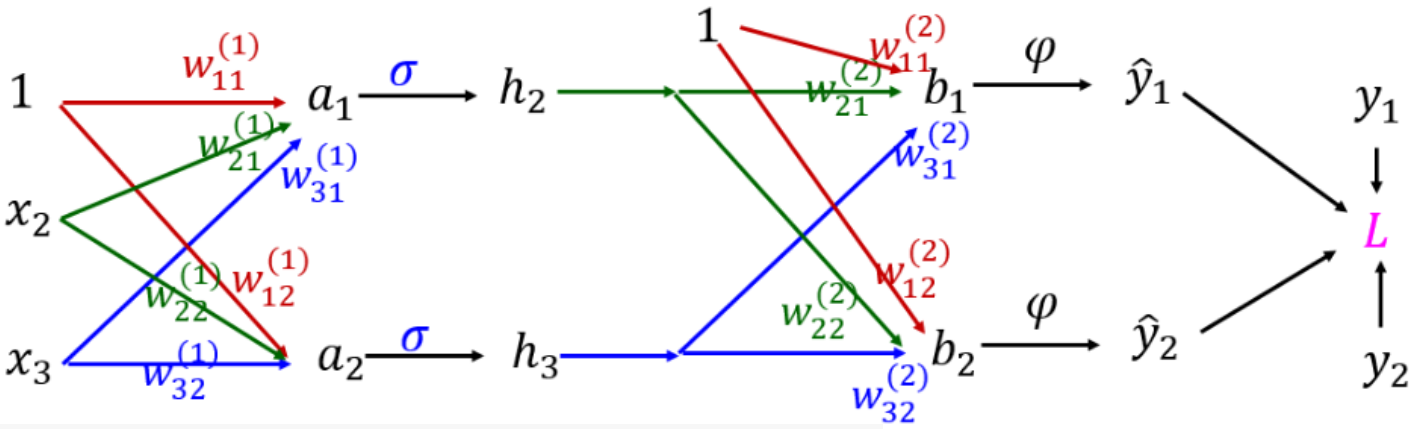


$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial w_{12}^{(1)}} \\ \frac{\partial L}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial w_{22}^{(1)}} \\ \frac{\partial L}{\partial w_{31}^{(1)}} & \frac{\partial L}{\partial w_{32}^{(1)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{12}^{(1)}} \\ \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{22}^{(1)}} \\ \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{31}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{32}^{(1)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1}1 & \frac{\partial L}{\partial a_2}1 \\ \frac{\partial L}{\partial a_1}x_2 & \frac{\partial L}{\partial a_2}x_2 \\ \frac{\partial L}{\partial a_1}x_3 & \frac{\partial L}{\partial a_2}x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial a_1} & \frac{\partial L}{\partial a_2} \end{bmatrix}$$

$$= [1\ X]^T(\hat{\mathrm{Y}} - \mathrm{Y})W_2[1:]^T * [\sigma'(h_2) \quad \sigma'(h_3)]$$

dW1 = np.matmul(prepend_bias(X).T, da)/X.shape[0]

2_neural_network_classifier.jpynb

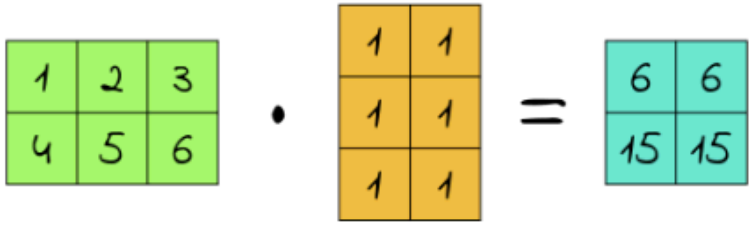



```
def forward(X, w1, w2):
    h = sigmoid(np.matmul(prepend_bias(X), w1))
    y_hat = softmax(np.matmul(prepend_bias(h), w2))
    return (y_hat, h)
```

```
def sigmoid_gradient(sigmoid):
    return np.multiply(sigmoid, (1 - sigmoid))
```

```
def back(X, Y, y_hat, w2, h):
    dw2 = np.matmul(prepend_bias(h).T, (y_hat - Y)) / X.shape[0]
    dh  = np.matmul(y_hat - Y, w2[1:].T)
    da  = dh * sigmoid_gradient(h)
    dw1 = np.matmul(prepend_bias(X).T, da)/X.shape[0]

#    dW1 = np.matmul(prepend_bias(X).T, np.matmul(y_hat-Y, w2[1:].T)* sigmoid_gradient(h))/X.shape[0]
    return (dw1, dw2)
```

Initializations of weights



- Assume the 1st and 2nd matrices are input X and the 1st layer's weights W_1.
- The resulting matrix passes through a sigmoid to become the hidden layer h, which has the same values in each row, meaning that all the hidden nodes of the network have the same value.
- By initializing all the weights with the same value, we forced our network to behave as if it had only one hidden node.
- We shouldn't initialize the weights with large values. Large weights generate large values which can cause problems if the network's function are not numerically stable: they can push those functions past the brink, making them overflow.

Initializations of weights

- Another reason is the larger the weight, the flatter the sigmoid; the flatter the sigmoid, the smaller the gradient; the smaller the gradient, the slower GD; and the slower GD, the less the weight changes.
- We should initialize weights with values that are:
  - Random (to break symmetry)
  - Small (to speed up training and avoid dead neurons)
- The absolute value of each weight shouldn't be much bigger than the square root of the inverse of rows r.
- As the weight matrix gets bigger, individual weights should become smaller.

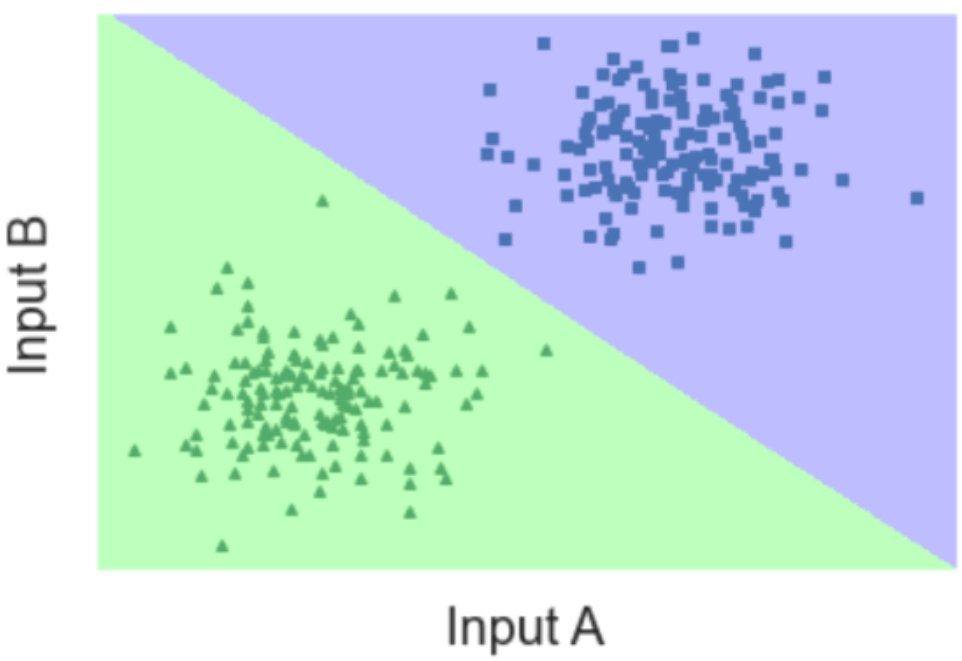
$$w \approx \pm\sqrt{\frac{1}{r}}$$

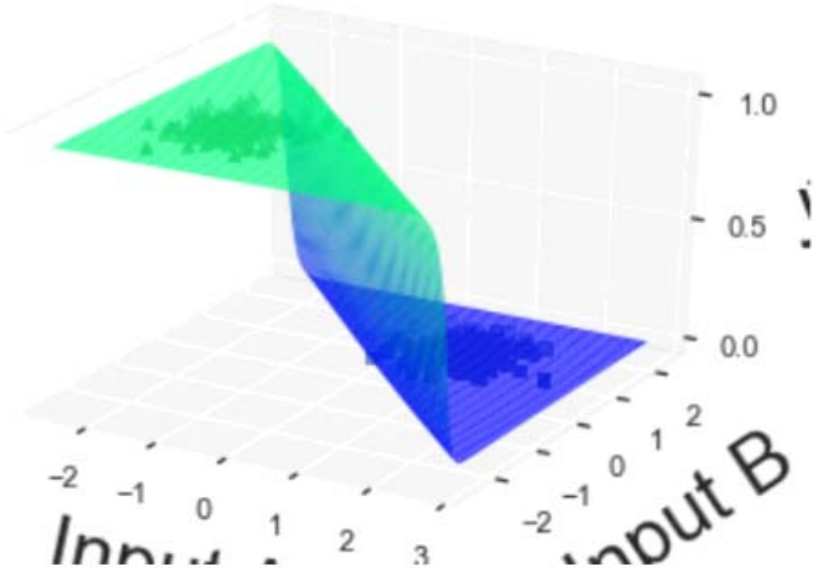
Initializations of weights

```
def initialize_weights(n_input_variables, n_hidden_nodes, n_classes):
    w1_rows = n_input_variables + 1
    w1 = np.random.randn(w1_rows, n_hidden_nodes) * np.sqrt(1 / w1_rows)

    w2_rows = n_hidden_nodes + 1
    w2 = np.random.randn(w2_rows, n_classes) * np.sqrt(1 / w2_rows)

    return (w1, w2)
```

2_neural_network_classifier.jpynb

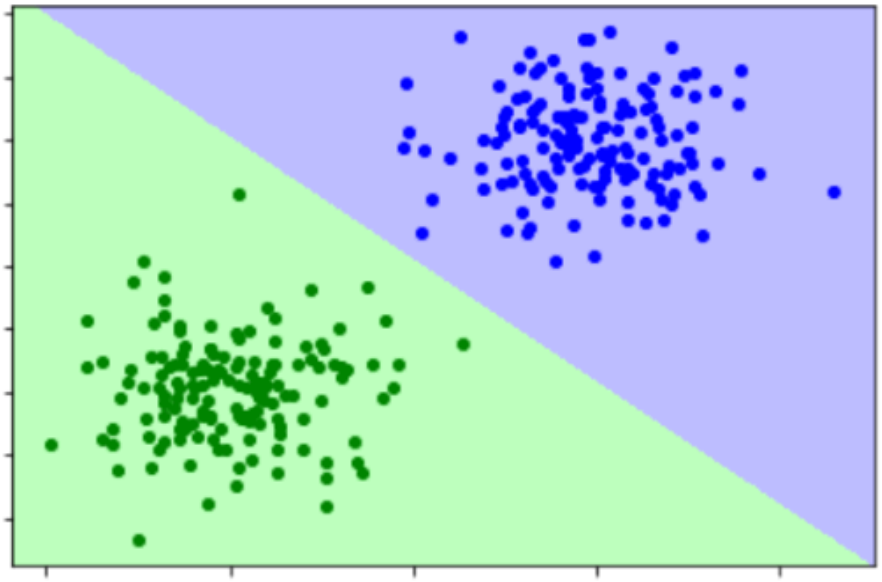
linearly_separable.txt

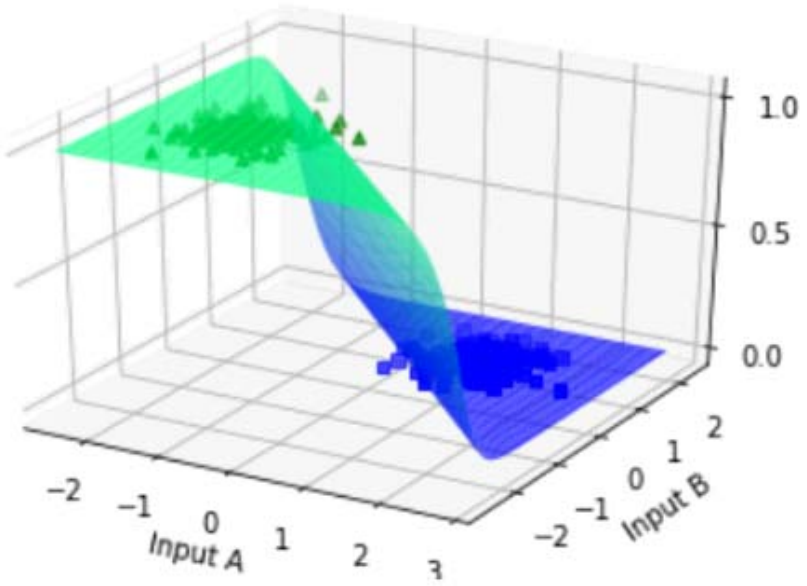




Limitations of the Single Layer: 1_perceptron_classifier.jpynb

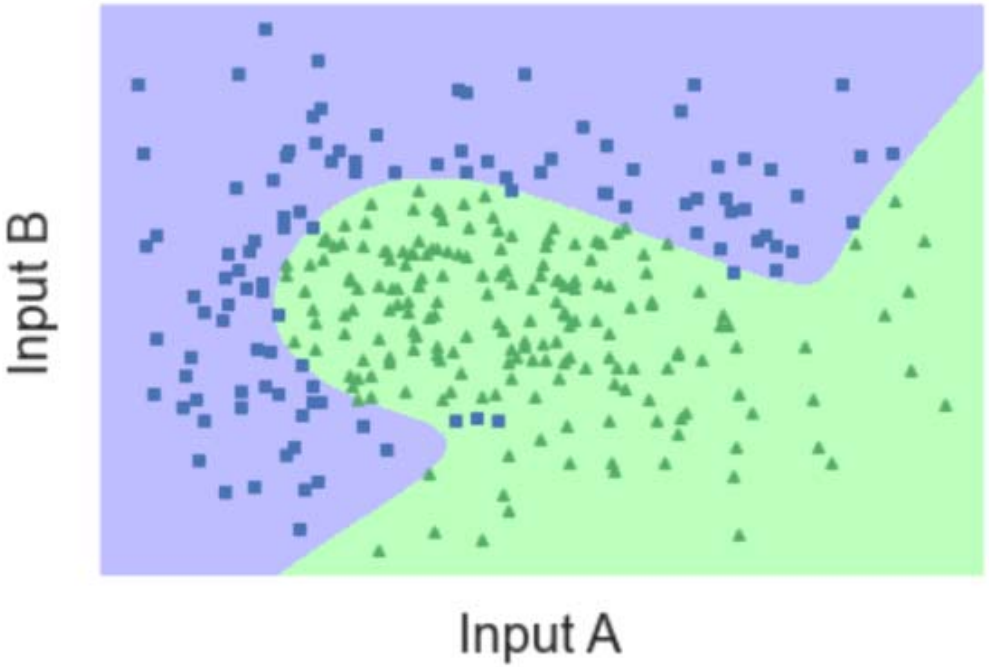
linearly_separable.txt

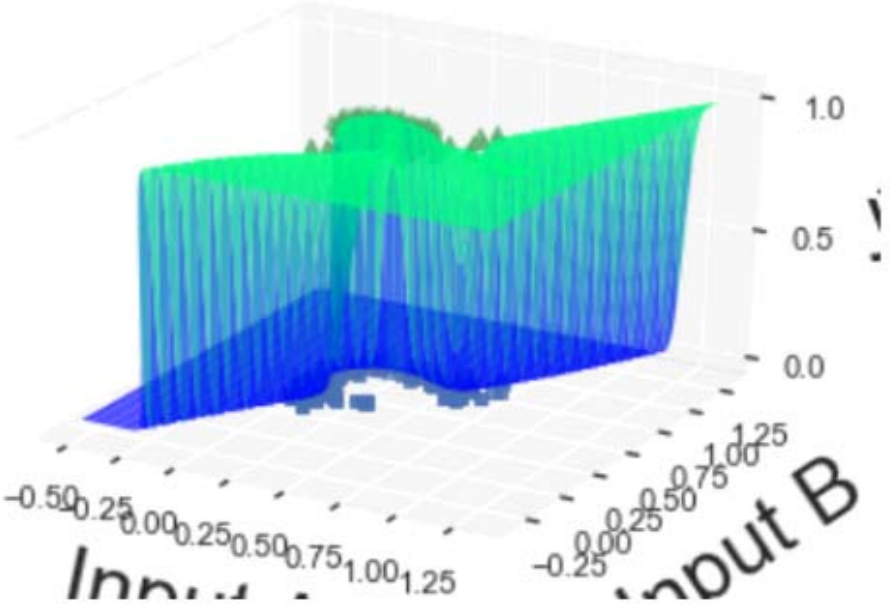




2_neural_network_classifier.jpynb

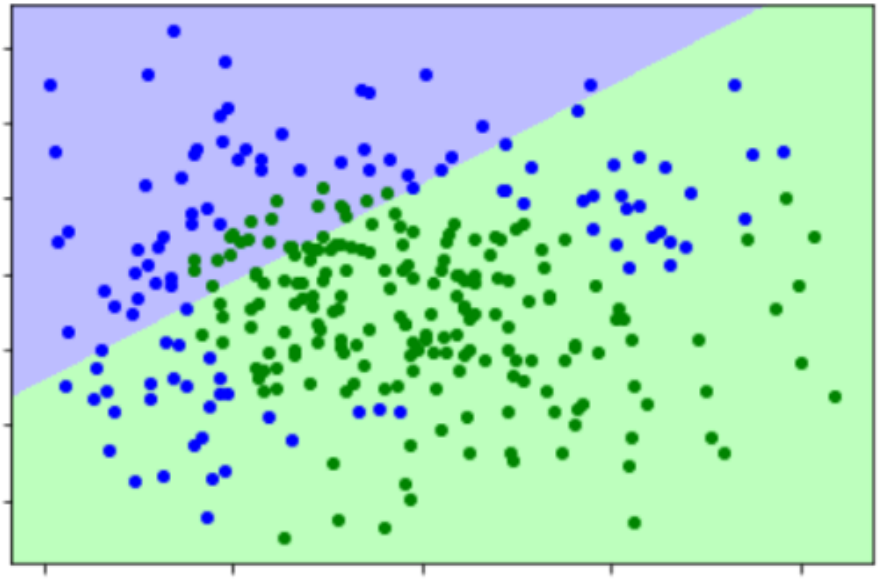
non_linearly_separable.txt

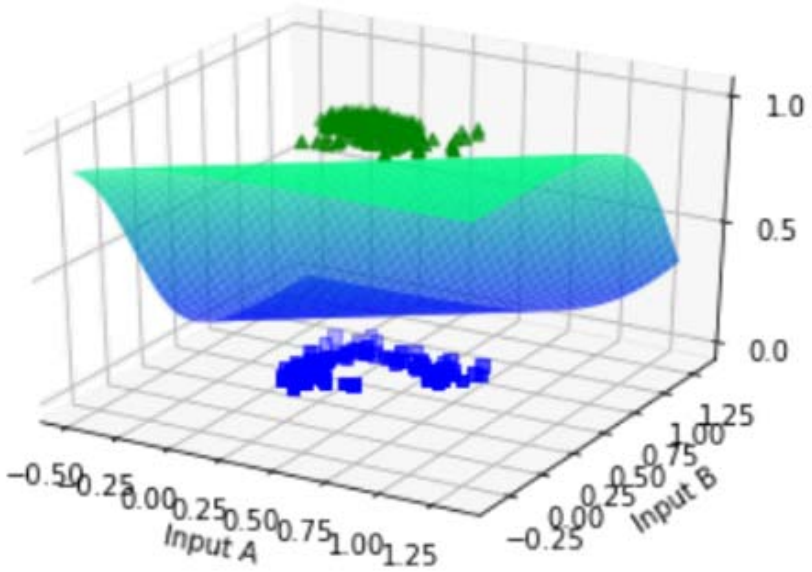




Limitations of the Single Layer: 1_perceptron_classifier.jpynb

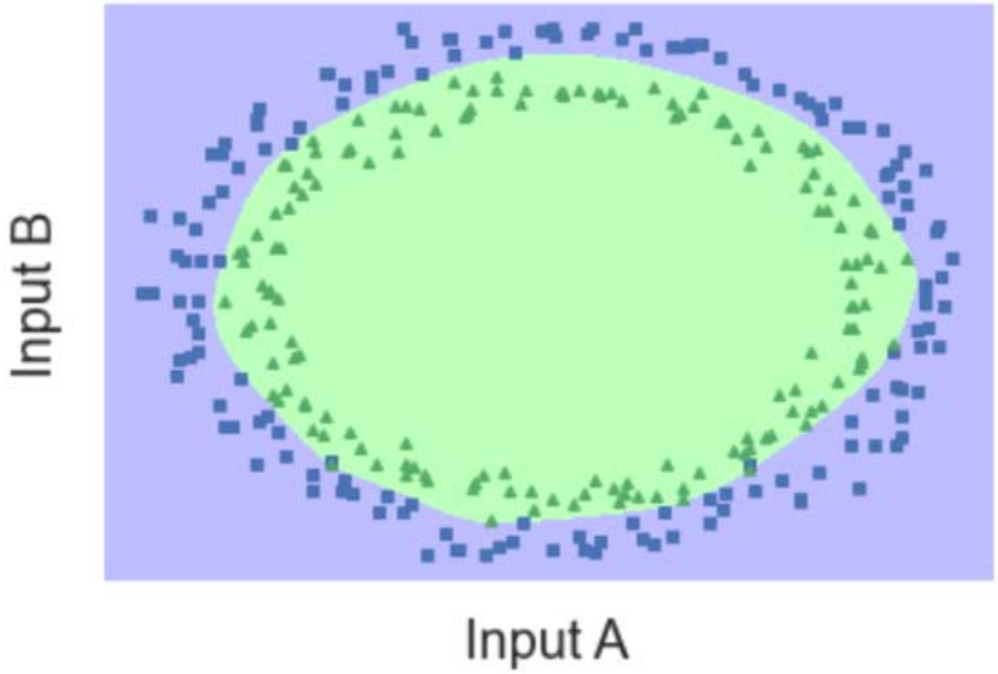
non_linearly_separable.txt

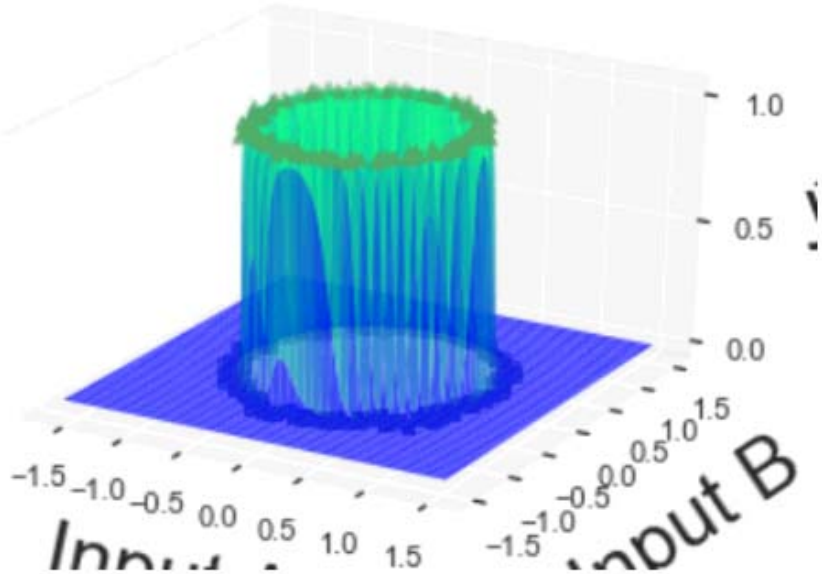




2_neural_network_classifier.jpynb

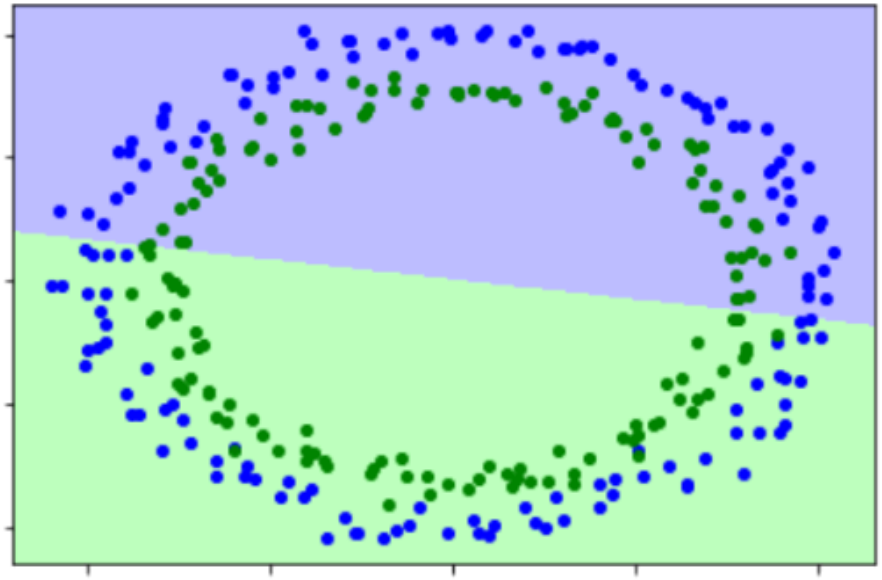
circles.txt

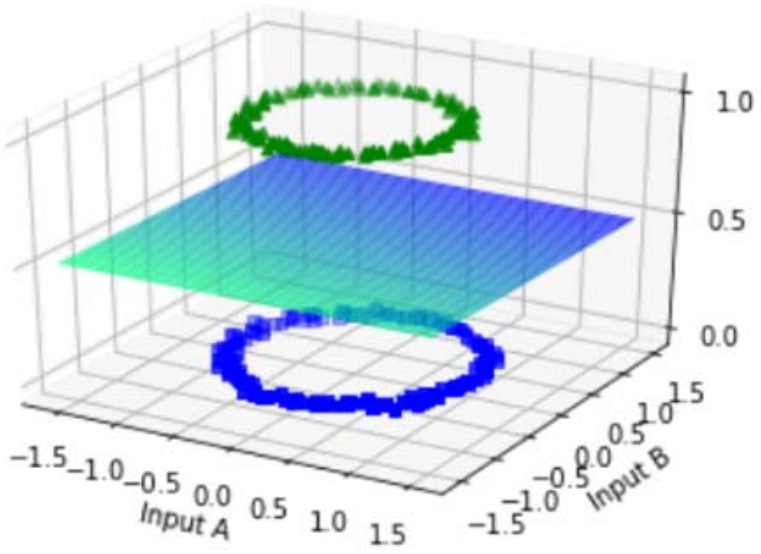




Limitations of the Single Layer: 1_perceptron_classifier.jpynb

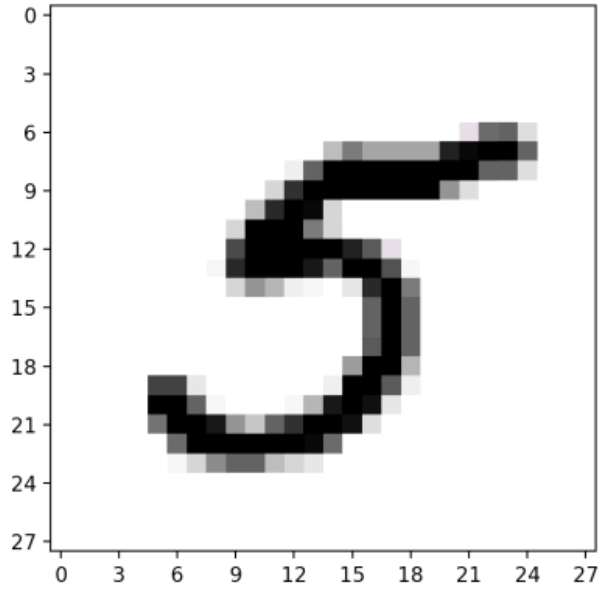
circles.txt



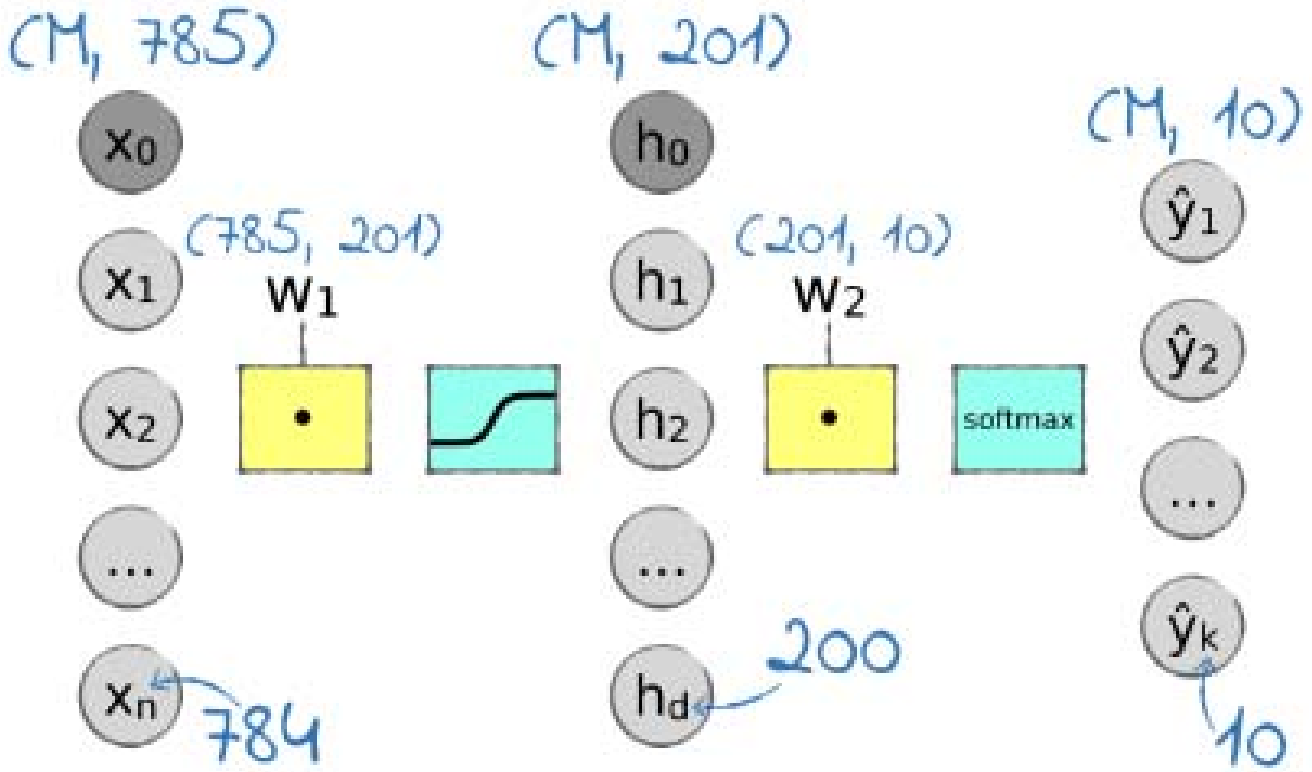


Multiple Classification of MNIST

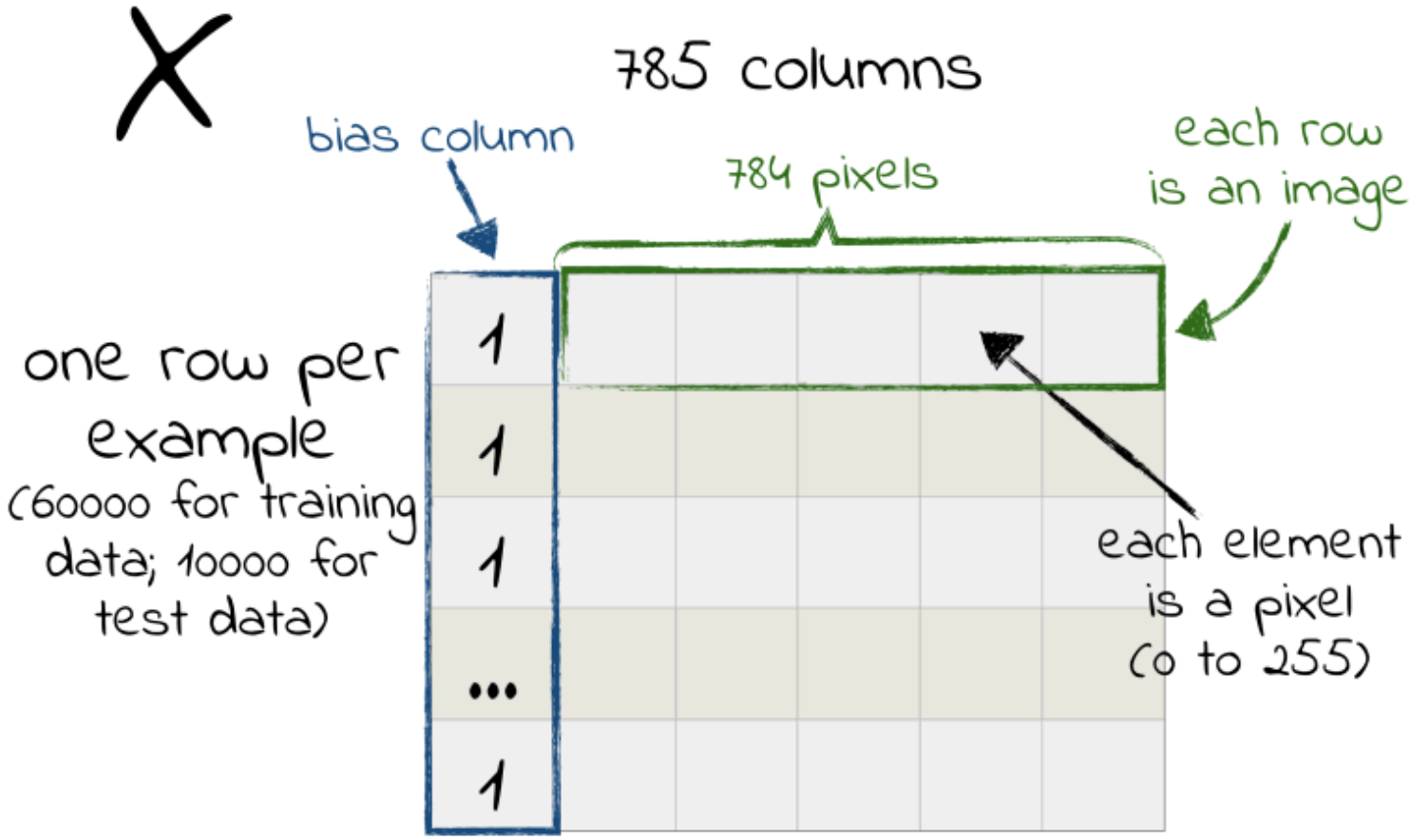
- Digits are made up of 28 by 28 grayscale pixels, each represented by one byte.
- In MNIST's grayscale, 0 stands for "perfect background white," and 255 stands for "perfect foreground black."
- It contains 70,000 examples, neatly partitioned into 7,000 examples for each digit from 0 to 9.



Multi-layer neural network



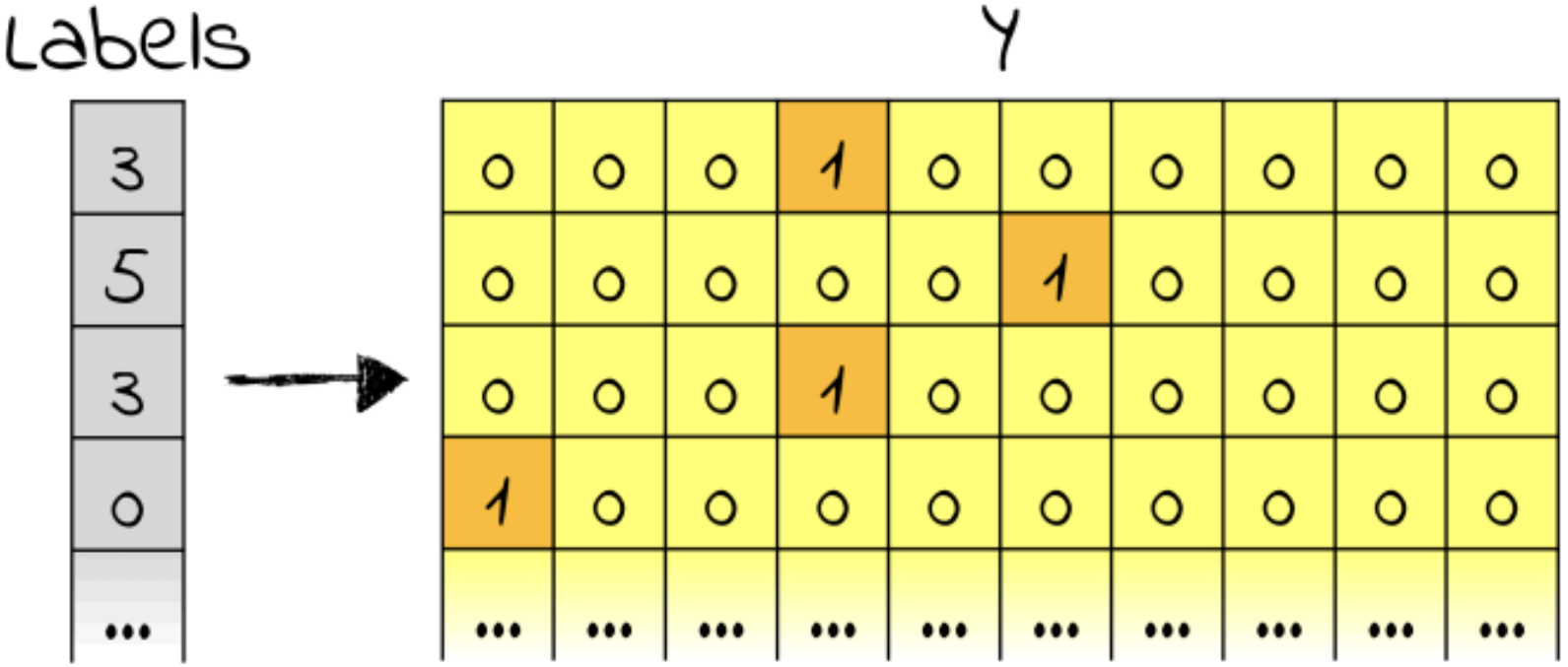
MNIST Training and Testing Data



One-Hot Encoding

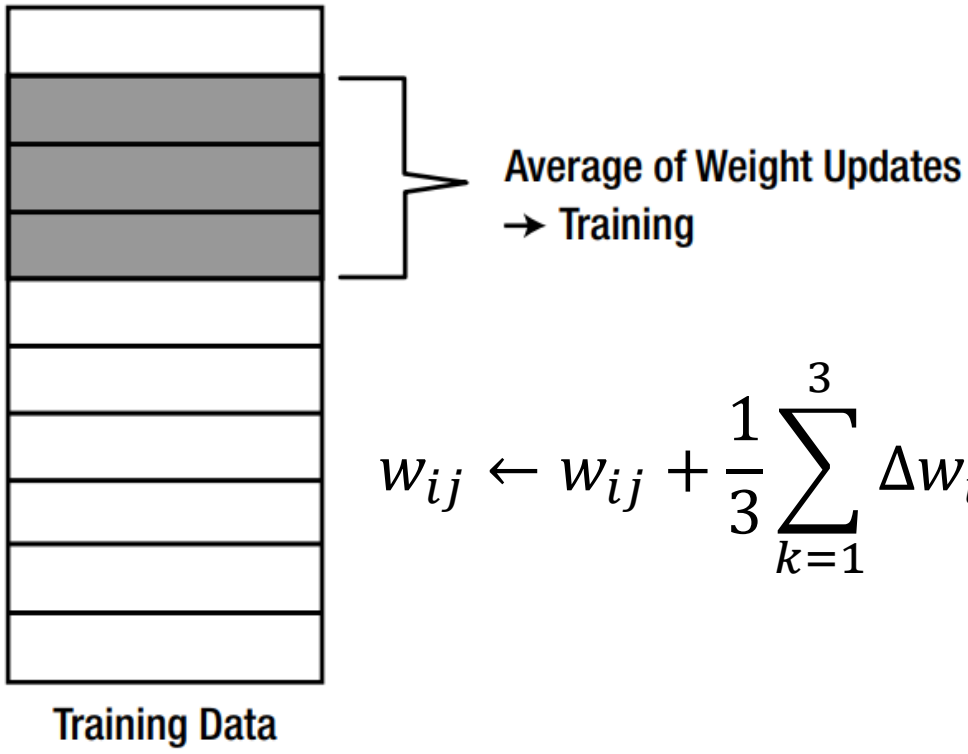
A possible output $\hat{y}_1, \cdots, \hat{y}_{10}$

"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
0.111	0.005	0.787	0.170	0.001	0.176	0.352	0.001	0.073	0.003



Mini-batch gradient descent with batch size 3

- Assume 99 training data, the weights are updated 33 iterations for each epoch
- Training 10 epochs needs 330 iterations



$$w_{ij} \leftarrow w_{ij} + \frac{1}{3}\sum_{k=1}^{3} \Delta w_{ij}\,(k)$$

```python
def prepare_batches(X_train, Y_train, batch_size):
    x_batches = []
    y_batches = []
    n_examples = X_train.shape[0]
    for batch in range(0, n_examples, batch_size):
        batch_end = batch + batch_size
        x_batches.append(X_train[batch:batch_end])
        y_batches.append(Y_train[batch:batch_end])
    return x_batches, y_batches
```

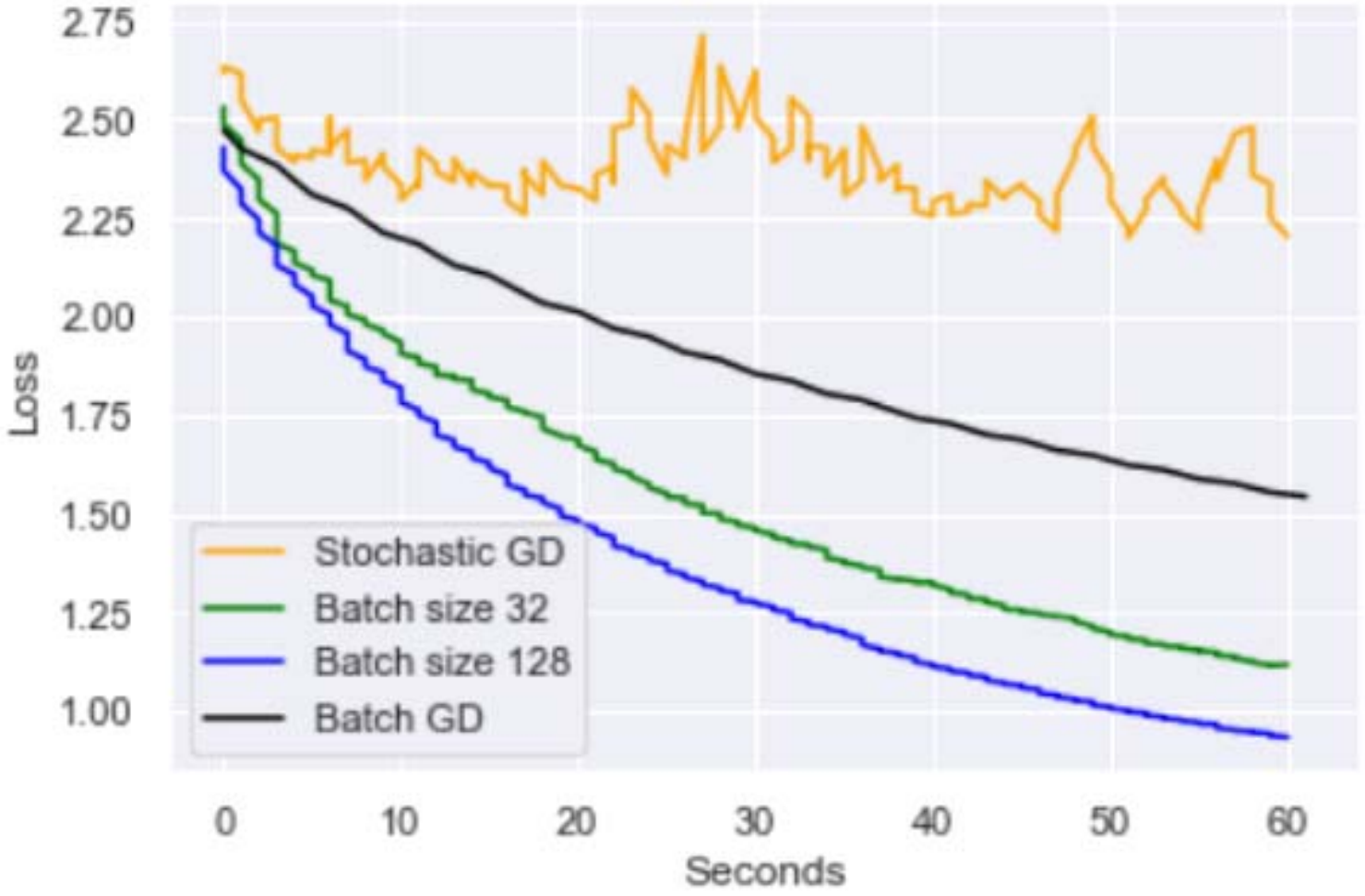
```python
tmpx = np.array([[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]])
tmpy = np.array([[0], [1], [0], [1], [0], [1], [0], [1], [0], [1]])
print(tmpx)
tmpx_batches, tmpy_batches = prepare_batches(tmpx, tmpy, 2)
tmpx_batches
```

```
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
```

```
[array([[0],
        [1]]),
 array([[2],
        [3]]),
 array([[4],
        [5]]),
 array([[6],
        [7]]),
 array([[8],
        [9]])]
```

```
def train(X_train, Y_train, X_test, Y_test, n_hidden_nodes,epochs, batch_size, lr):
    n_input_variables = X_train.shape[1]
    n_classes = Y_train.shape[1]

    w1, w2 = initialize_weights(n_input_variables, n_hidden_nodes, n_classes)
    x_batches, y_batches = prepare_batches(X_train, Y_train, batch_size)
    for epoch in range(epochs):
        for batch in range(len(x_batches)):
            y_hat, h = forward(x_batches[batch], w1, w2)
            w1_gradient, w2_gradient = back(x_batches[batch], y_batches[batch],
                                            y_hat, w2, h)
            w1 = w1 - (w1_gradient * lr)
            w2 = w2 - (w2_gradient * lr)
            report(epoch, batch, X_train, Y_train, X_test, Y_test, w1, w2)
    return (w1, w2)
```

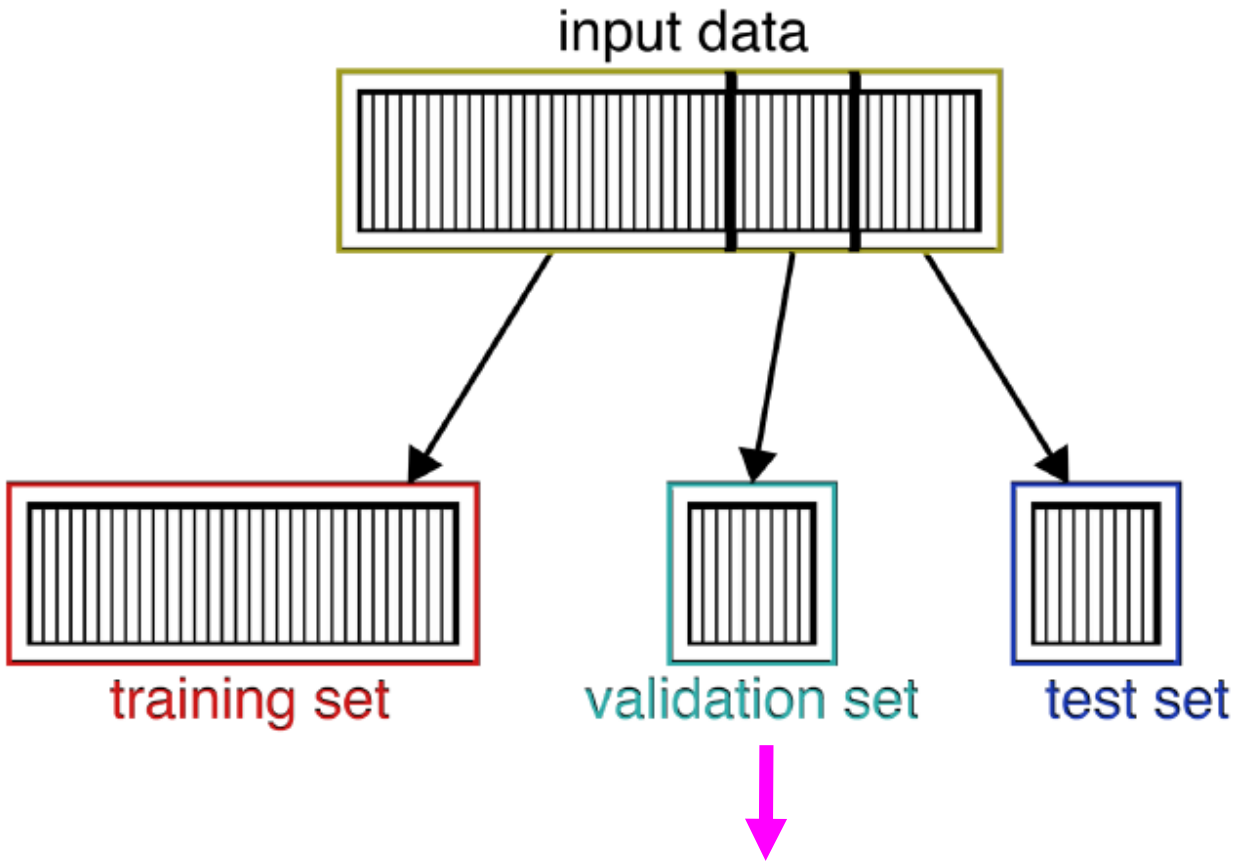
03_batching.jpynb



Splitting input data into a training, validation, and test set

1. The setup: we put the test set aside. We'll never look at it until the very end.
2. The development cycle: we train the network on the training set as usual, but we use the validation set to gauge its performance.
3. The final test: after we've tweaked and tuned our hyperparameters, we test the network on the test set, which gives us an objective idea of how it will perform in production.

Splitting input data into a training, validation, and test set

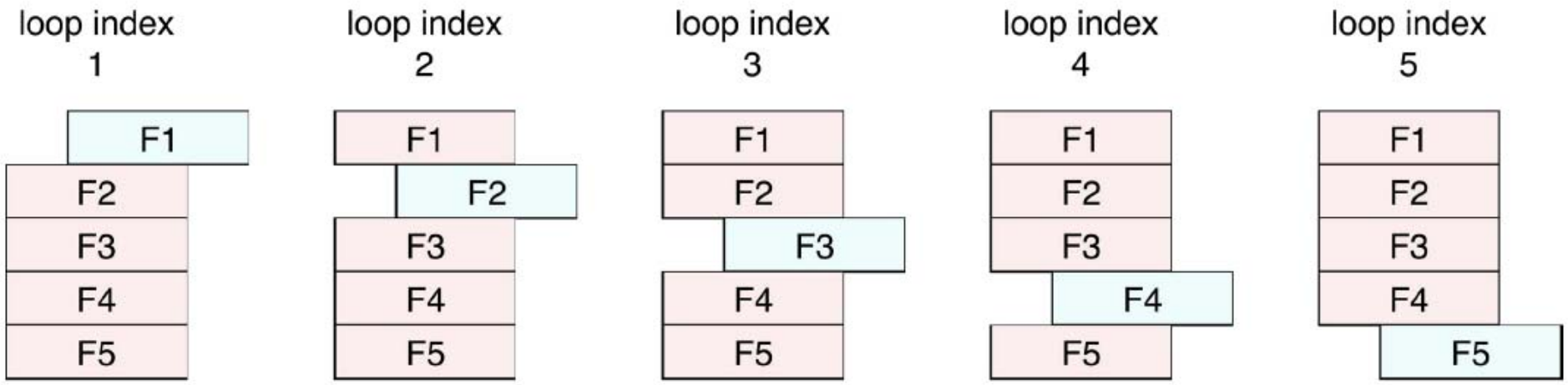


Determine hyper-parameters: learning rate, batch size, …

Andrew Glassner - Deep Learning_ A Visual Approach-No Starch Press (2021)

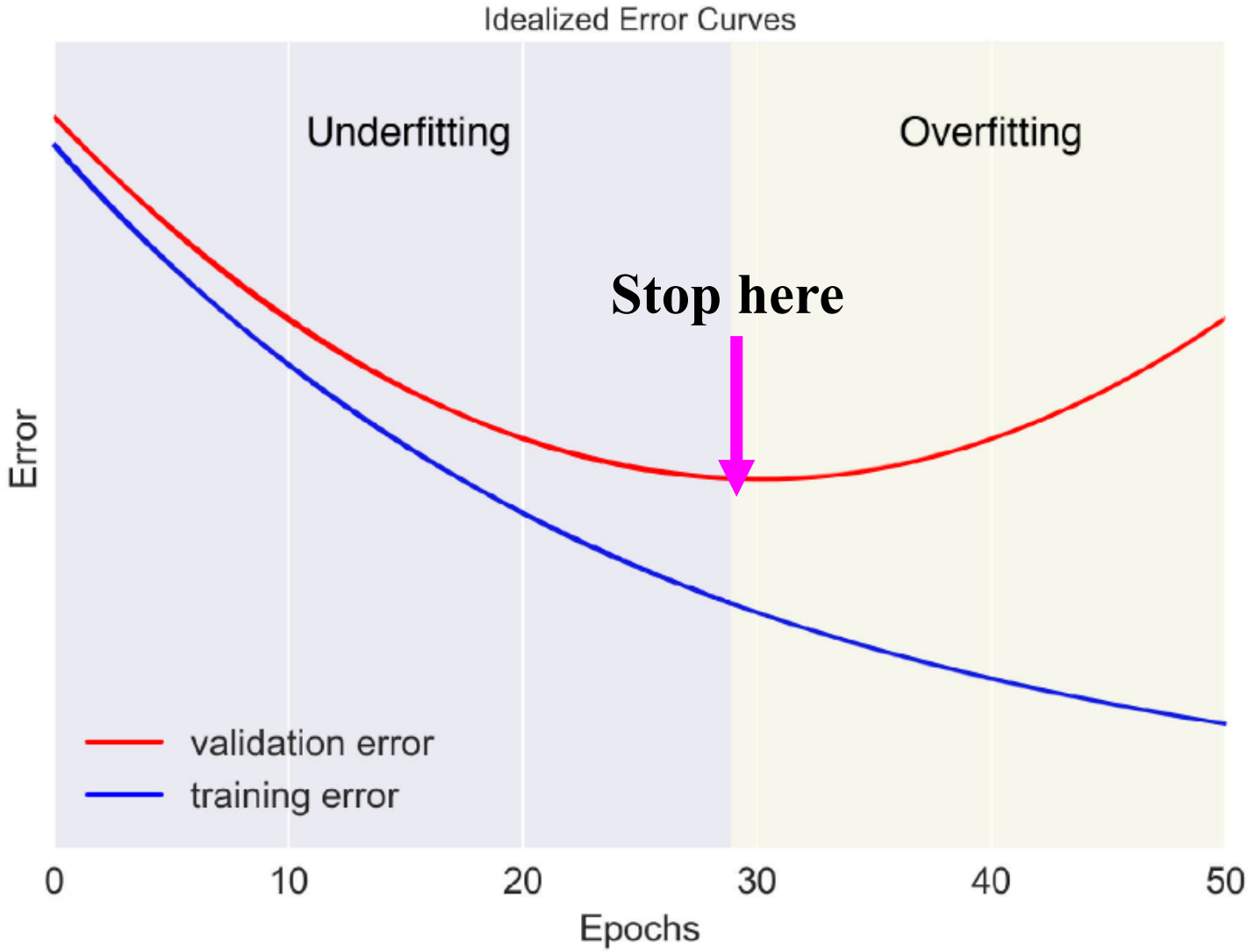
K-fold cross-validation

In each pass through the loop, we choose one fold for validation (in blue), and train with the others (in red).



Andrew Glassner - Deep Learning_ A Visual Approach-No Starch Press (2021)

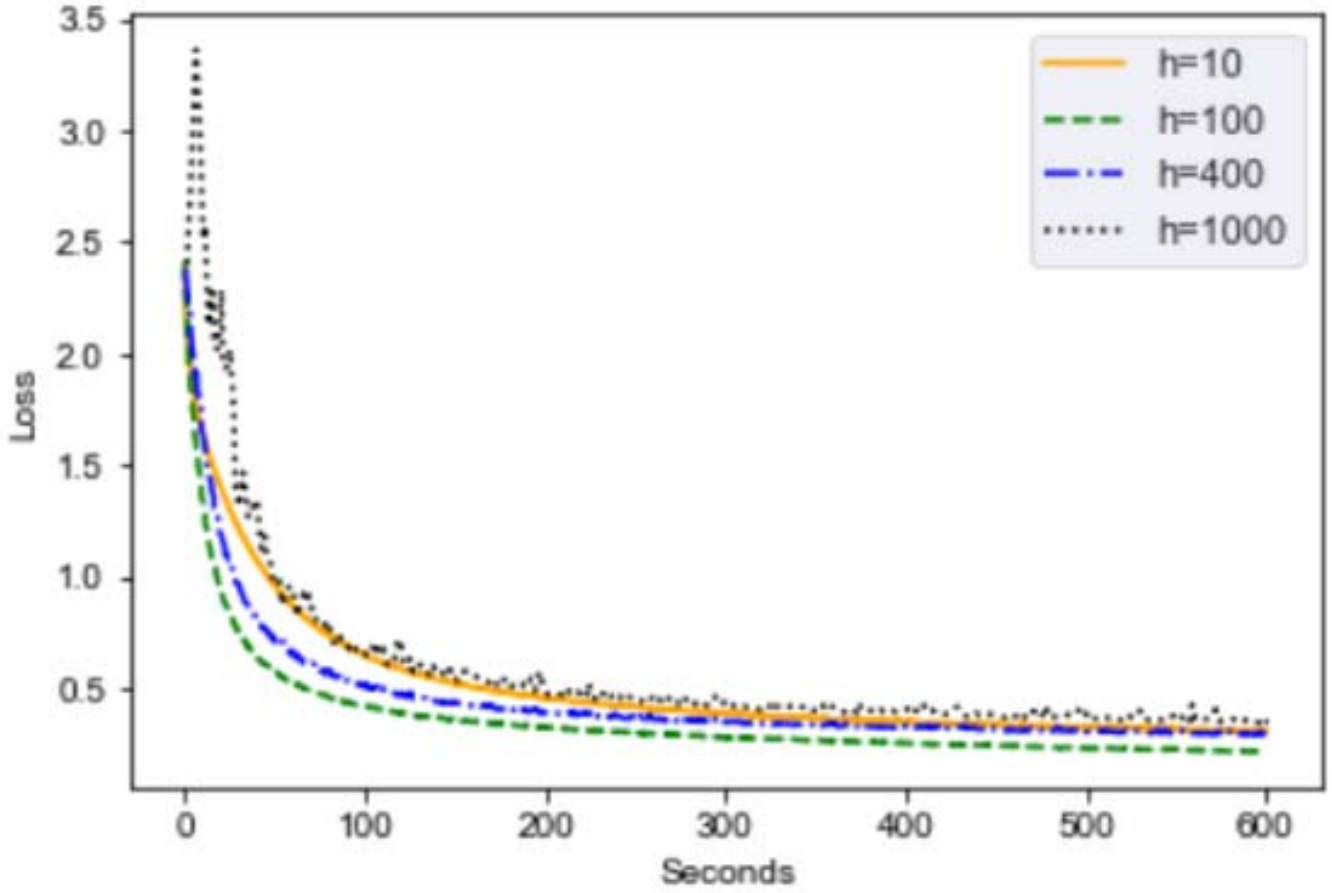
An idealized set of error curves



Andrew Glassner - Deep Learning_ A Visual Approach-No Starch Press (2021)

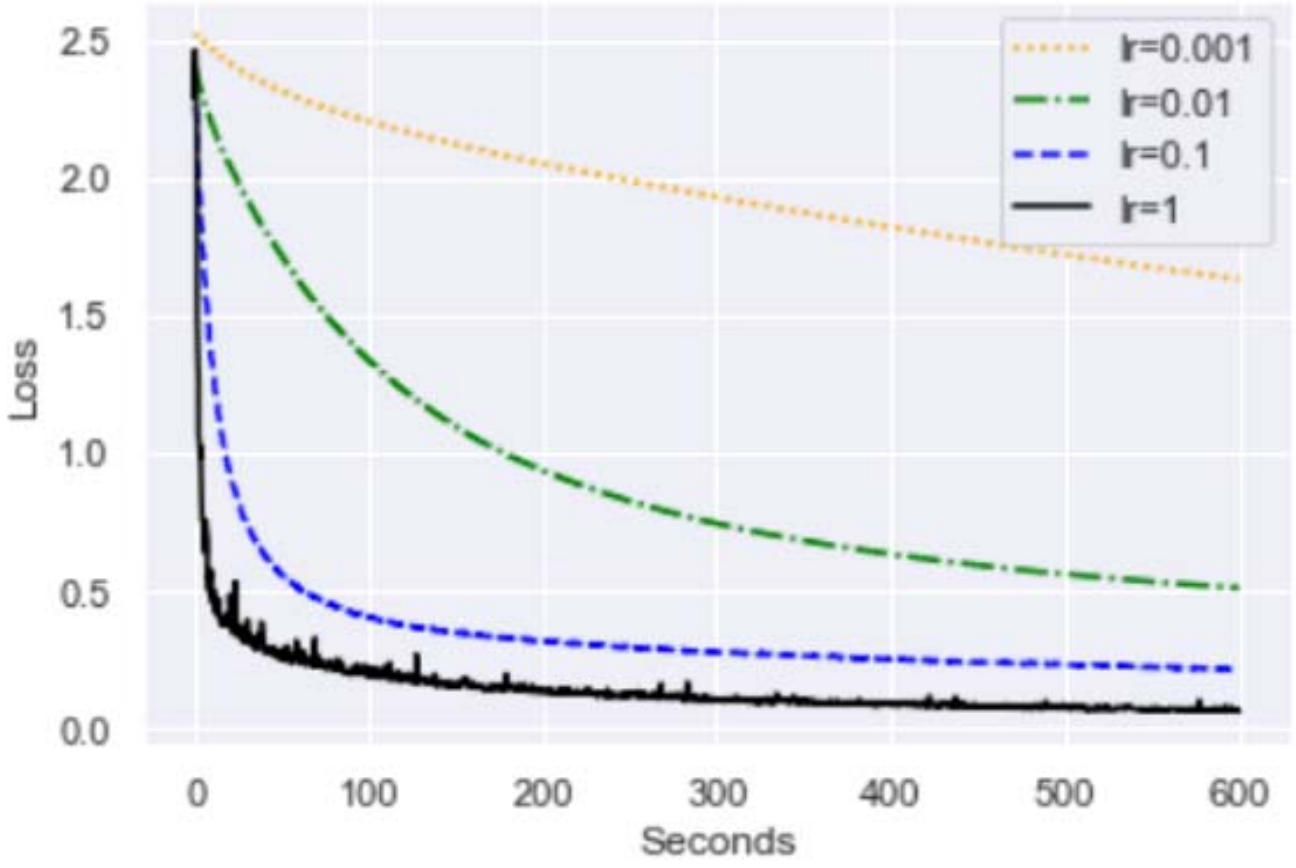
Tuning Hyperparameters: 5_development.jpynb

Tuning the Number of Hidden Nodes



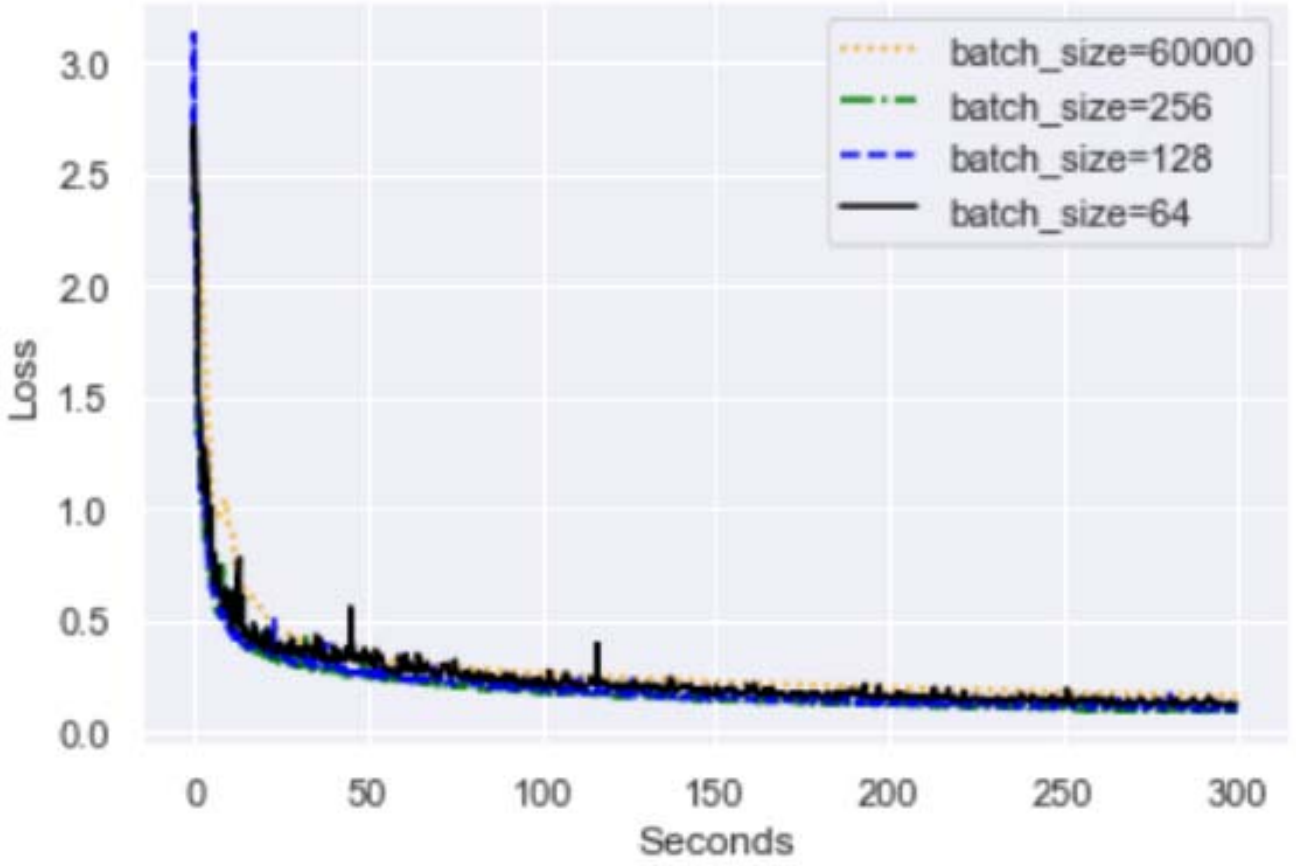
Tuning Hyperparameters: 5_development.jpynb

Tuning the Learning Rate



Tuning Hyperparameters: 5_development.jpynb

Tuning the Batch Size



Tuning Hyperparameters: 5_development.jpynb

```
train(X_train, Y_train, X_test, Y_test,
      n_hidden_nodes=100, epochs=10, batch_size=256, lr=1)
print("Done.")
```

```
   0-0 > Loss: 2.51126276, Accuracy: 23.72%
   0-60 > Loss: 0.33310378, Accuracy: 93.48%
   0-120 > Loss: 0.30845169, Accuracy: 93.60%
   0-180 > Loss: 0.23942956, Accuracy: 95.40%
   1-0 > Loss: 0.20253645, Accuracy: 95.98%
   1-60 > Loss: 0.17660872, Accuracy: 96.48%
   1-120 > Loss: 0.17063644, Accuracy: 96.62%
   1-180 > Loss: 0.15613097, Accuracy: 96.96%
   2-0 > Loss: 0.14175469, Accuracy: 97.18%
   9-0 > Loss: 0.04812531, Accuracy: 98.38%
   9-60 > Loss: 0.04472160, Accuracy: 98.40%
   9-120 > Loss: 0.04300803, Accuracy: 98.52%
   9-180 > Loss: 0.04358591, Accuracy: 98.28%
Done.
```

Vanishing Gradient and Overfitting

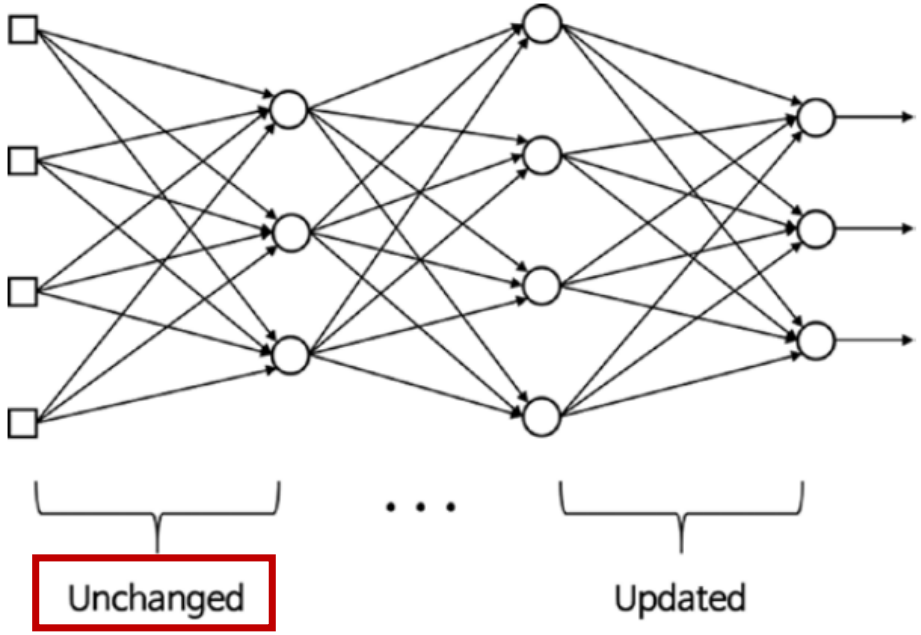
- The neural network with deeper layers yielded poorer performance was that the network was not properly trained.

- The backpropagation algorithm experiences two difficulties in training deep neural network：
  - **Vanishing gradient**
  - **Overfitting**

Vanishing Gradient

- The vanishing gradient in the training process occurs when the output error is more likely to **fail to reach the farther nodes**.

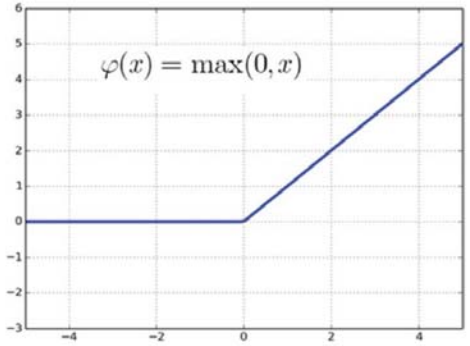
- As the error hardly reaches the first hidden layer, **the weight cannot be adjusted**.



Vanishing Gradient： ReLU

- A solution to the vanishing gradient is using the Rectified Linear Unit (**ReLU**) function as the activation function.

$$\rho(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} = max(0, x)$$



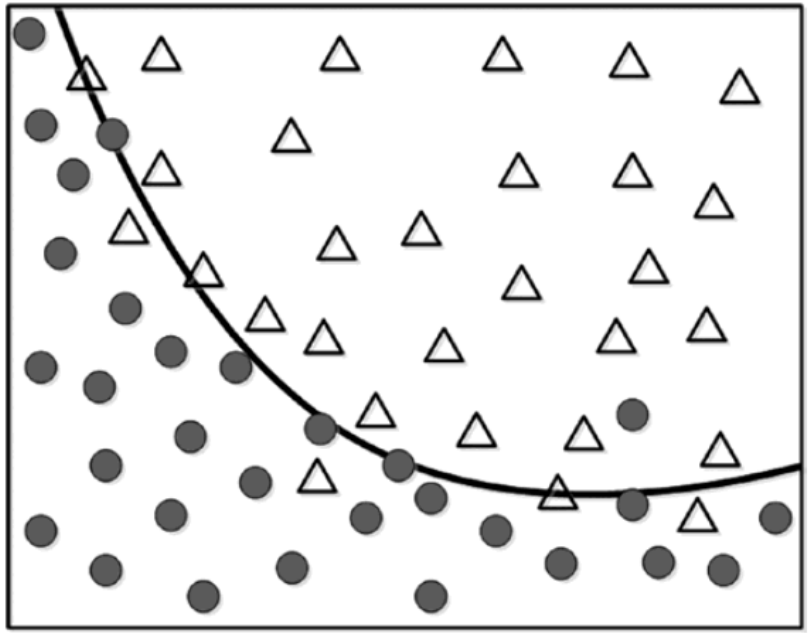
- The sigmoid function limits the node's outputs to the unity, the ReLU function does not exert such limits and better transmit the error than the sigmoid function.

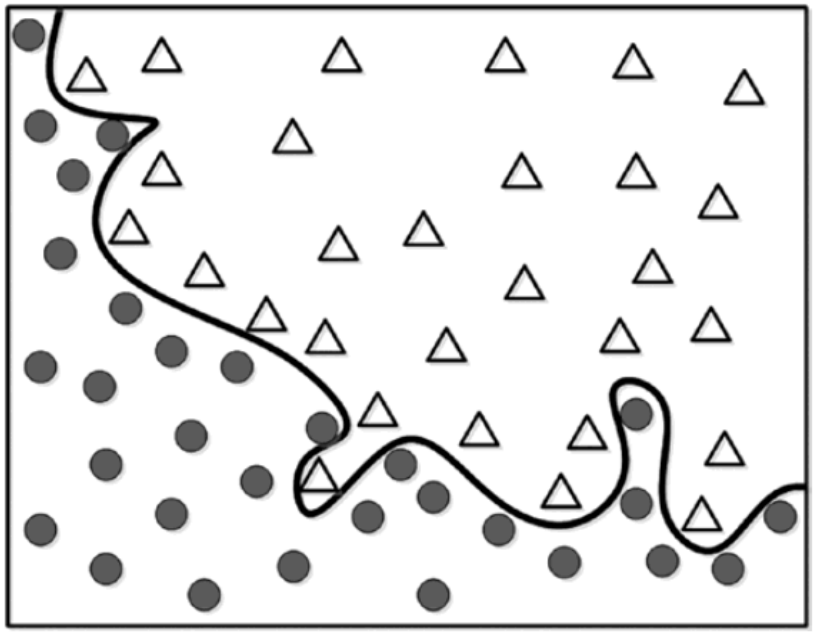
- We also need the derivative of the ReLU function.

$$\rho'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Overfitting

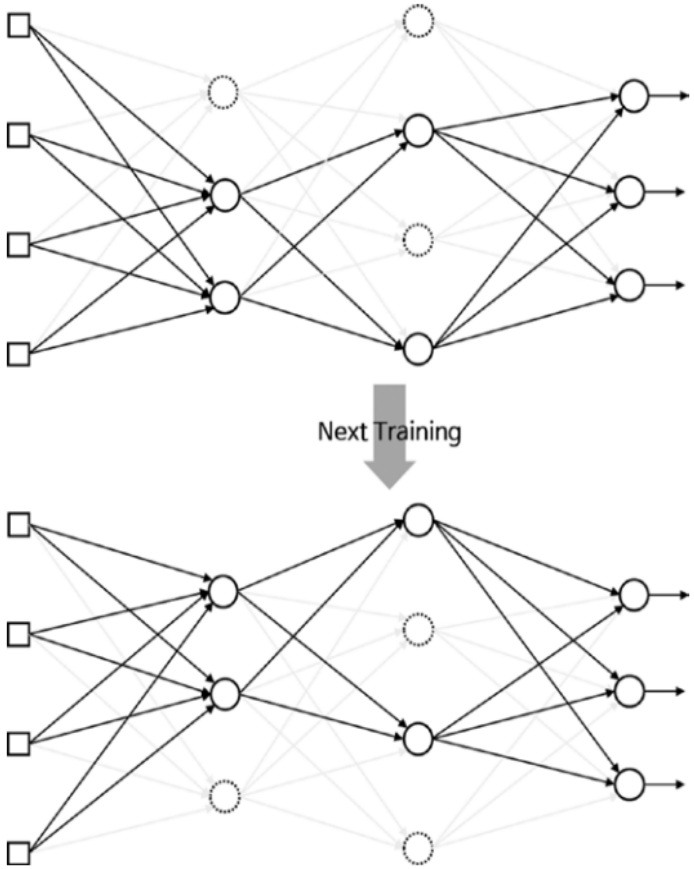
- The reason that the deep neural network is especially vulnerable to overfitting is that the model becomes **more complicated as it includes more hidden layers**, and hence more weight.





Overfitting：Dropout

- **Train only some of the randomly selected nodes**
- 50% and 25% for hidden and input layers are dropped out
- Continuously alters the nodes and weights in the training process
- Use massive training data is very helpful to reduce potential bias



An example of Dropout

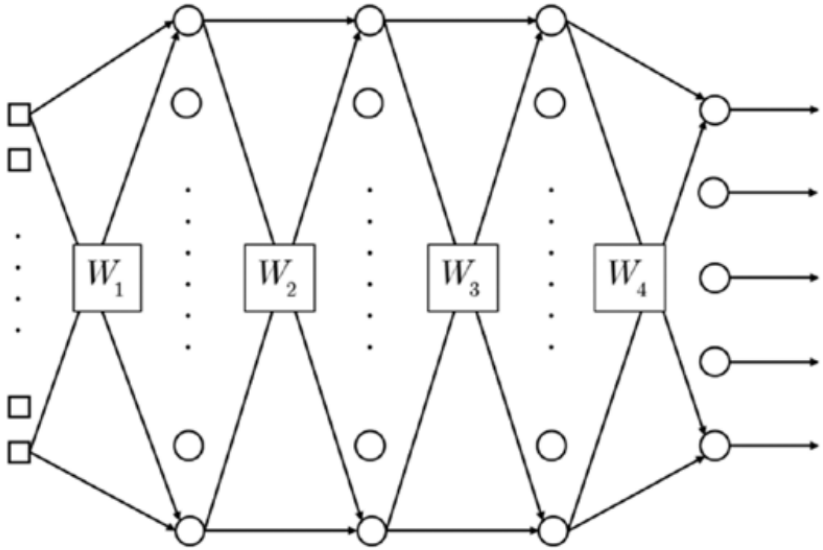
```
ratio=0.2
h1 = np.random.random((2, 3))
print(h1)
yd = np.zeros(h1.size)
print('yd=',ym)
num = round(h1.size*(1-ratio))
print(f'h1.size={h1.size}, num={num}')
idx = np.random.choice(h1.size, num, replace=False)
print('idx=',idx)
yd[idx]= 1.0 / (1.0 - ratio)
print('yd =',yd)
yd=np.reshape(yd,(h1.shape))
print('yd =',yd)
```

```
[[0.04806826 0.43194944 0.98873132]
 [0.30957655 0.98688492 0.77138021]]
yd= [0. 0. 0. 0. 0. 0.]
h1.size=6, num=5
idx= [5 1 2 3 4]
yd = [0.   1.25 1.25 1.25 1.25 1.25]
yd = [[0.   1.25 1.25]
 [1.25 1.25 1.25]]
```

- ratio=0.2 drops out 20% of the hidden nodes
- yd contains zeros for as many elements as the ratio and 1 / (1 - ratio) for the other elements to compensate for the loss of output due to the dropped elements

Example: ReLU and Dropout

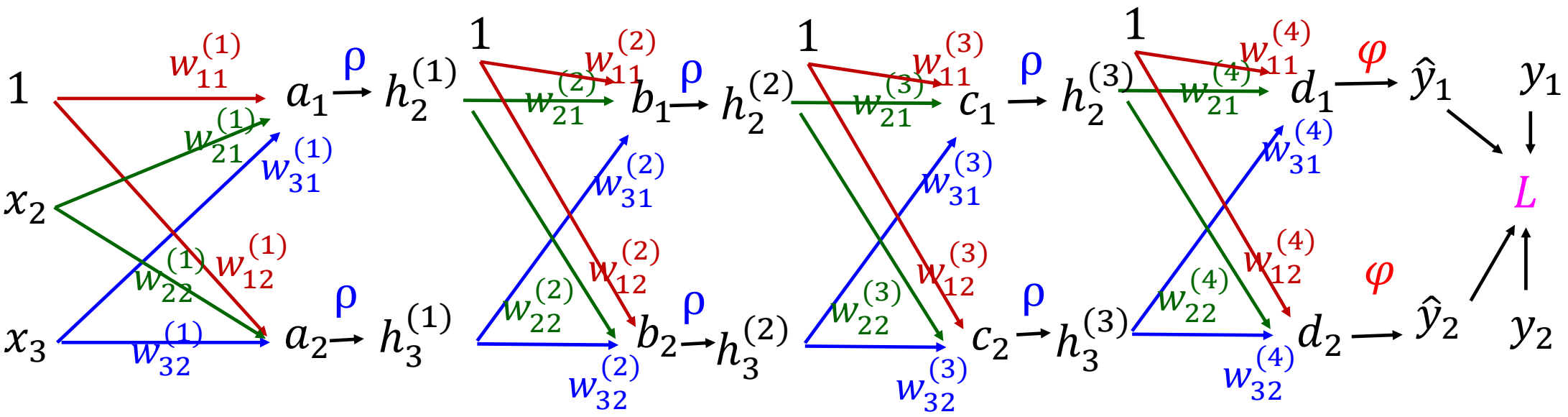
- The network has 25 input nodes
- Five output nodes for the five classes.
- Output nodes employ the softmax activation function.
- Three hidden layers, each hidden layer contains 20 nodes.



Data

```
X = np.zeros((5, 5, 5))

X[0, :, :] = [[0,1,1,0,0],
              [0,0,1,0,0],
              [0,0,1,0,0],
              [0,0,1,0,0],
              [0,1,1,1,0]]

X[1, :, :] = [[1,1,1,1,0],
              [0,0,0,0,1],
              [0,1,1,1,0],
              [1,0,0,0,0],
              [1,1,1,1,1]]

X[2, :, :] = [[1,1,1,1,0],
              [0,0,0,0,1],
              [0,1,1,1,0],
              [0,0,0,0,1],
              [1,1,1,1,0]]
```

```
X[3, :, :] = [[0,0,0,1,0],
              [0,0,1,1,0],
              [0,1,0,1,0],
              [1,1,1,1,1],
              [0,0,0,1,0]]

X[4, :, :] = [[1,1,1,1,1],
              [1,0,0,0,0],
              [1,1,1,1,0],
              [0,0,0,0,1],
              [1,1,1,1,0]]
```



$$[a_1 \quad a_2] = [1 \quad x_2 \quad x_3] \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} \triangleq [1\ X]W_1$$

$$[h_2^{(1)} \quad h_3^{(1)}] = [\rho\,(a_1) \quad \rho\,(a_2)] \triangleq H_1$$

$$[b_1 \quad b_2] = [1 \quad h_2^{(1)} \quad h_3^{(1)}] \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix} \triangleq [1H_1]W_2$$

$$[h_2^{(2)} \quad h_3^{(2)}] = [\rho(b_1) \quad \rho(b_2)] \triangleq H_2$$

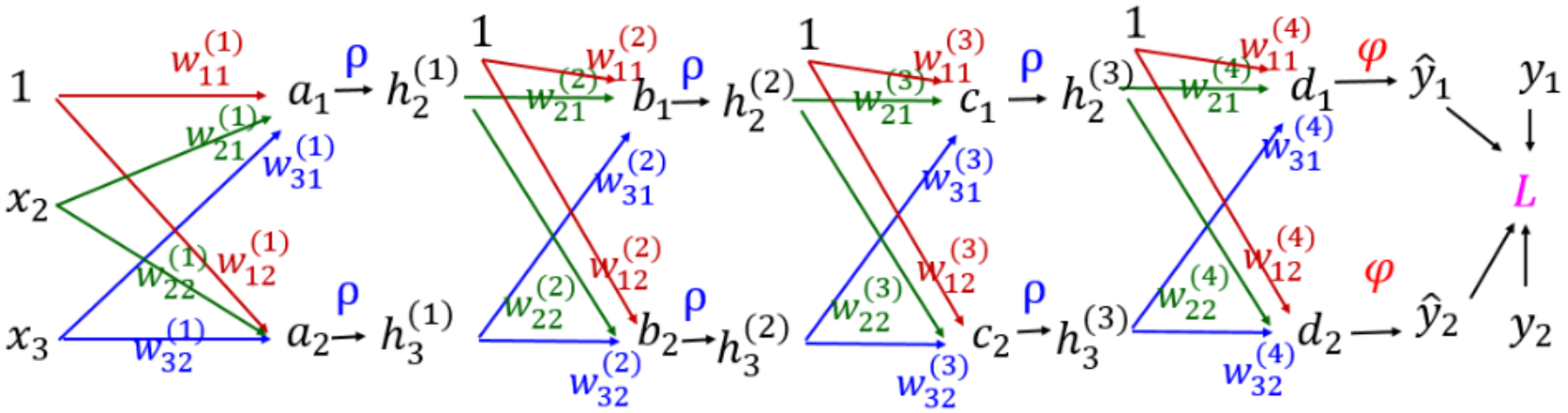
$$[c_1 \quad c_2] = [1 \quad h_2^{(2)} \quad h_3^{(2)}] \begin{bmatrix} w_{11}^{(3)} & w_{12}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} \\ w_{31}^{(3)} & w_{32}^{(3)} \end{bmatrix} \triangleq [1H_2]W_3$$

$$[h_2^{(3)} \quad h_3^{(3)}] = [\rho(c_1) \quad \rho(c_2)] \triangleq H_3$$

$$[d_1 \quad d_2] = [1 \quad h_2^{(3)} \quad h_3^{(3)}] \begin{bmatrix} w_{11}^{(4)} & w_{12}^{(4)} \\ w_{21}^{(4)} & w_{22}^{(4)} \\ w_{31}^{(4)} & w_{32}^{(4)} \end{bmatrix} \triangleq [1H_3]W_4$$

$$[\hat{y}_1 \quad \hat{y}_2] = [\varphi(d_1) \quad \varphi(d_2)] \triangleq \hat{\mathrm{Y}}$$

Backpropagation: Gradient of W_4 in matrix form

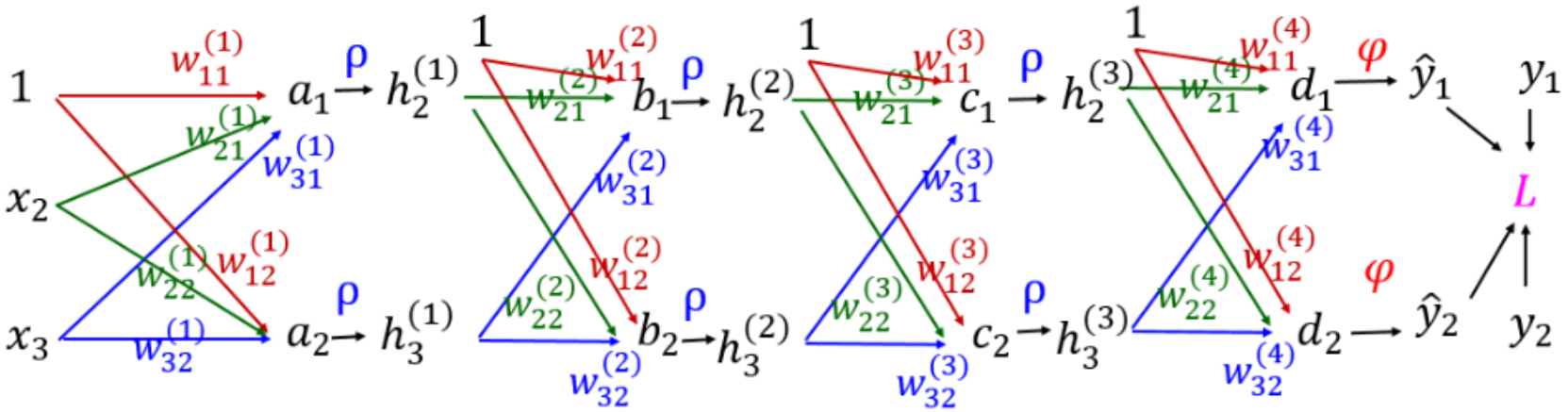


$$\frac{\partial L}{\partial d_1} = \frac{\partial L}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial d_1} = \hat{y}_1 - y_1,$$

$$\frac{\partial L}{\partial d_2} = \frac{\partial L}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial d_2} = \hat{y}_2 - y_2$$

$$\begin{bmatrix} \frac{\partial L}{\partial d_1} & \frac{\partial L}{\partial d_2} \end{bmatrix} = [\hat{y}_1 - y_1 \quad \hat{y}_2 - y_2] = \hat{\mathrm{Y}} - \mathrm{Y}, \qquad \mathrm{Y} \triangleq [y_1 \quad y_2]$$

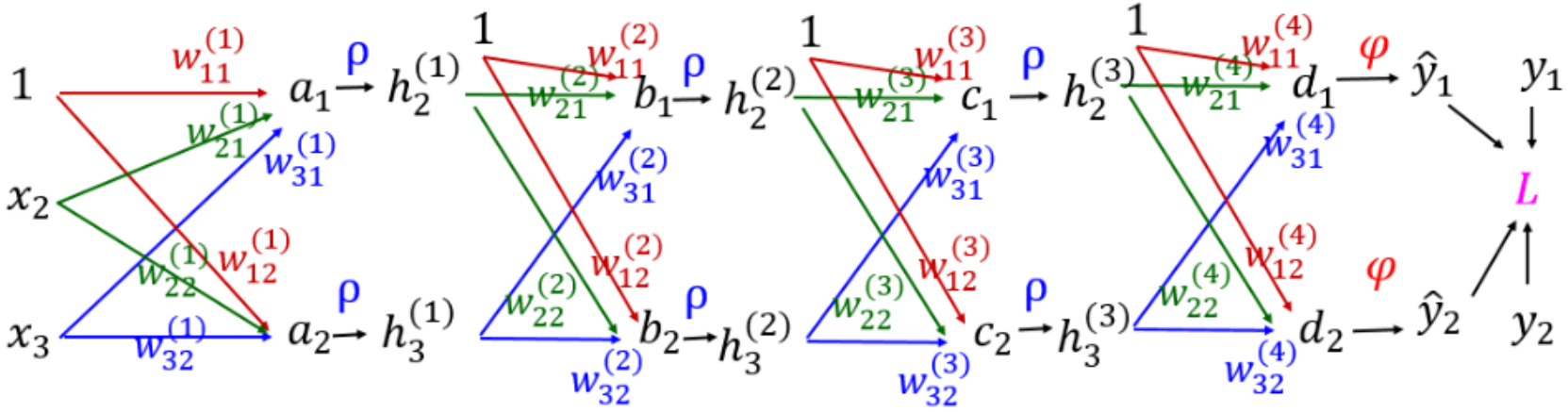
Backpropagation: Gradient of W_4 in matrix form



$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(4)}} & \frac{\partial L}{\partial w_{12}^{(4)}} \\ \frac{\partial L}{\partial w_{21}^{(4)}} & \frac{\partial L}{\partial w_{22}^{(4)}} \\ \frac{\partial L}{\partial w_{31}^{(4)}} & \frac{\partial L}{\partial w_{32}^{(4)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial d_1}\frac{\partial d_1}{\partial w_{11}^{(4)}} & \frac{\partial L}{\partial d_2}\frac{\partial d_2}{\partial w_{12}^{(4)}} \\ \frac{\partial L}{\partial d_1}\frac{\partial d_1}{\partial w_{21}^{(4)}} & \frac{\partial L}{\partial d_2}\frac{\partial d_2}{\partial w_{22}^{(4)}} \\ \frac{\partial L}{\partial d_1}\frac{\partial d_1}{\partial w_{31}^{(4)}} & \frac{\partial L}{\partial d_2}\frac{\partial d_2}{\partial w_{32}^{(4)}} \end{bmatrix} = \begin{bmatrix} 1 \\ h_2^{(3)} \\ h_3^{(3)} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial d_1} & \frac{\partial L}{\partial d_2} \end{bmatrix} = [1 H_3]^T (\hat{Y} - Y)$$

dW4 = np.matmul(prepend_bias(H3).T, Y_hat-Y)/X.shape[0]

Backpropagation: Gradient of W_3 in matrix form



$$\frac{\partial L}{\partial h_2^{(3)}} = \frac{\partial L}{\partial d_1}\frac{\partial d_1}{\partial h_2^{(3)}} + \frac{\partial L}{\partial d_2}\frac{\partial d_2}{\partial h_2^{(3)}} = (\hat{y}_1 - y_1)w_{21}^{(4)} + (\hat{y}_2 - y_2)w_{22}^{(4)}$$

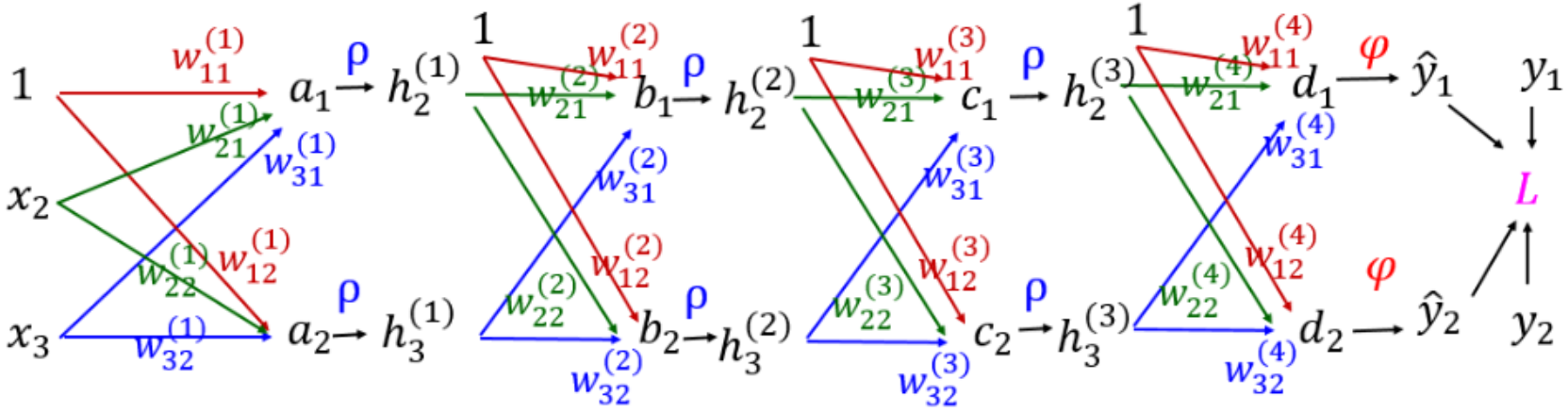
$$\frac{\partial L}{\partial h_3^{(3)}} = \frac{\partial L}{\partial d_1}\frac{\partial d_1}{\partial h_3^{(3)}} + \frac{\partial L}{\partial d_2}\frac{\partial d_2}{\partial h_3^{(3)}} = (\hat{y}_1 - y_1)w_{31}^{(4)} + (\hat{y}_2 - y_2)w_{32}^{(4)}$$

$$\begin{bmatrix}\frac{\partial L}{\partial h_2^{(3)}} & \frac{\partial L}{\partial h_3^{(3)}}\end{bmatrix} = [\hat{y}_1 - y_1 \quad \hat{y}_2 - y_2]\begin{bmatrix} w_{21}^{(4)} & w_{31}^{(4)} \\ w_{22}^{(4)} & w_{32}^{(4)}\end{bmatrix} = (\hat{\mathrm{Y}} - \mathrm{Y})W_4[1:]^T$$

Note $W_4[1:]$ represents all the columns from the second column $= W_4[1:2]$

dH3=np.matmul(Y_hat – Y, $W_4[1:]$.T)

Backpropagation: Gradient of W_3 in matrix form



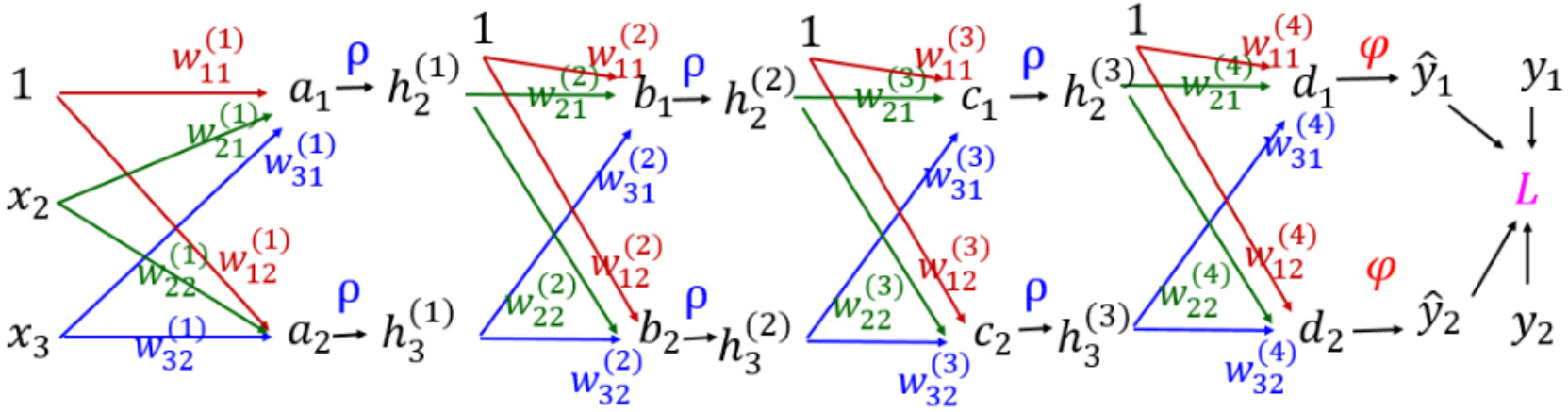
$$\frac{\partial L}{\partial c_1} = \frac{\partial L}{\partial h_2^{(3)}} \frac{\partial h_2^{(3)}}{\partial c_1} = \frac{\partial L}{\partial h_2^{(3)}} \rho'\left(h_2^{(3)}\right), \qquad \rho'\left(h_2^{(3)}\right) = \begin{cases} 1, h_2^{(3)} > 0 \\ 0, h_2^{(3)} \le 0 \end{cases}$$

$$\frac{\partial L}{\partial c_2} = \frac{\partial L}{\partial h_3^{(3)}} \frac{\partial h_3^{(3)}}{\partial c_2} = \frac{\partial L}{\partial h_3^{(3)}} \rho'\left(h_3^{(3)}\right), \qquad \rho'\left(h_3^{(3)}\right) = \begin{cases} 1, h_3^{(3)} > 0 \\ 0, h_3^{(3)} \le 0 \end{cases}$$

$$\left[\frac{\partial L}{\partial c_1} \quad \frac{\partial L}{\partial c_2}\right] = \left[\frac{\partial L}{\partial h_2^{(3)}} \quad \frac{\partial L}{\partial h_3^{(3)}}\right] * \left[\rho'\left(h_2^{(3)}\right) \quad \rho'\left(h_3^{(3)}\right)\right]$$

dc=dH3 $*\rho'$(H3)

Backpropagation: Gradient of W_3 in matrix form



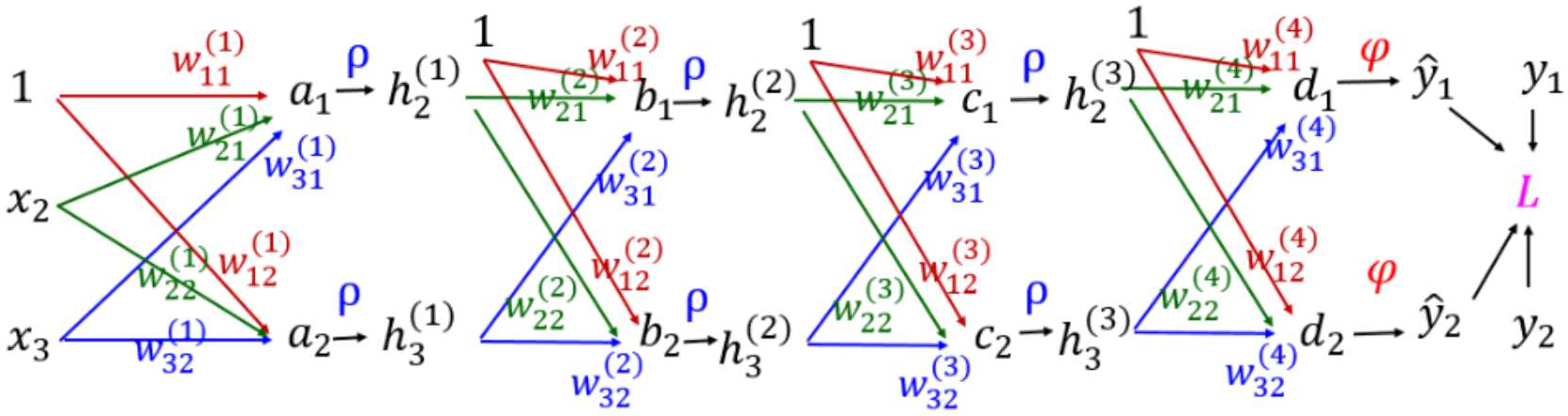
$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(3)}} & \frac{\partial L}{\partial w_{12}^{(3)}} \\ \frac{\partial L}{\partial w_{21}^{(3)}} & \frac{\partial L}{\partial w_{22}^{(3)}} \\ \frac{\partial L}{\partial w_{31}^{(3)}} & \frac{\partial L}{\partial w_{32}^{(3)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial c_1}\frac{\partial c_1}{\partial w_{11}^{(3)}} & \frac{\partial L}{\partial c_2}\frac{\partial c_2}{\partial w_{12}^{(3)}} \\ \frac{\partial L}{\partial c_1}\frac{\partial c_1}{\partial w_{21}^{(3)}} & \frac{\partial L}{\partial c_2}\frac{\partial c_2}{\partial w_{22}^{(3)}} \\ \frac{\partial L}{\partial c_1}\frac{\partial c_1}{\partial w_{31}^{(3)}} & \frac{\partial L}{\partial c_2}\frac{\partial c_2}{\partial w_{32}^{(3)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial c_1}1 & \frac{\partial L}{\partial c_2}1 \\ \frac{\partial L}{\partial c_1}h_2^{(2)} & \frac{\partial L}{\partial c_2}h_2^{(2)} \\ \frac{\partial L}{\partial c_1}h_3^{(2)} & \frac{\partial L}{\partial c_2}h_3^{(2)} \end{bmatrix} = \begin{bmatrix} 1 \\ h_2^{(2)} \\ h_3^{(2)} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial c_1} & \frac{\partial L}{\partial c_2} \end{bmatrix} = [1 H_2]^T \times dc$$

dH3=np.matmul(Y_hat – Y, $W_4[1:]$.T)

dc=dH3 $*\rho'$(H3)

dW3 = np.matmul(prepend_bias(H2).T, dc)/X.shape[0]

Backpropagation: Gradient of W_3 in matrix form



$$\frac{\partial L}{\partial h_2^{(2)}} = \frac{\partial L}{\partial c_1}\frac{\partial c_1}{\partial h_2^{(2)}} + \frac{\partial L}{\partial c_2}\frac{\partial c_2}{\partial h_2} = \frac{\partial L}{\partial c_1} w_{21}^{(3)} + \frac{\partial L}{\partial c_2} w_{22}^{(3)}$$

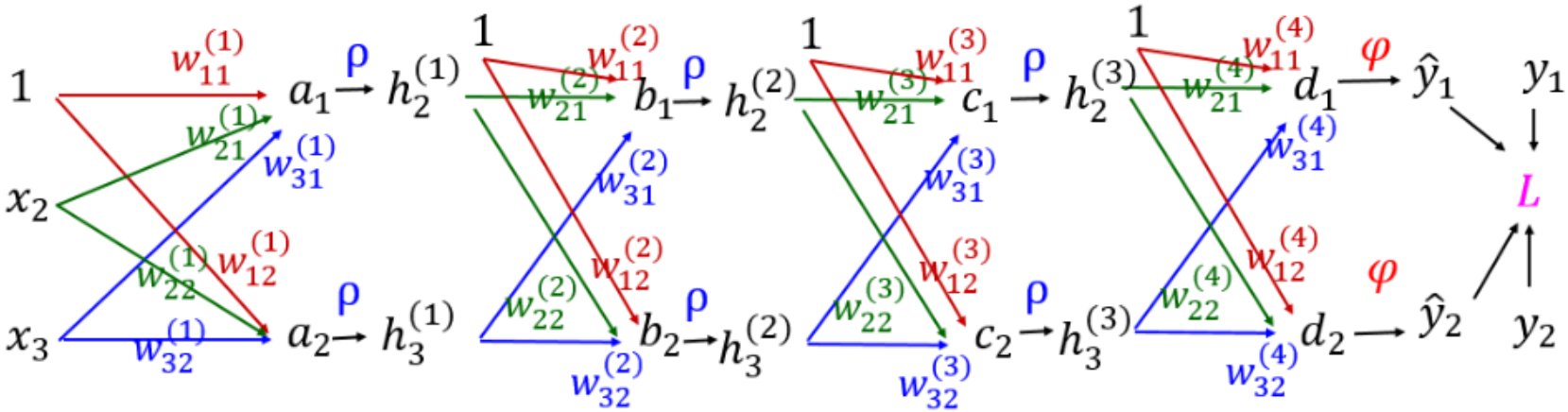
$$\frac{\partial L}{\partial h_3^{(2)}} = \frac{\partial L}{\partial c_1}\frac{\partial c_1}{\partial h_3^{(2)}} + \frac{\partial L}{\partial c_2}\frac{\partial c_2}{\partial h_3^{(2)}} = \frac{\partial L}{\partial c_1} w_{31}^{(3)} + \frac{\partial L}{\partial c_2} w_{32}^{(3)}$$

$$\begin{bmatrix} \frac{\partial L}{\partial h_2^{(2)}} & \frac{\partial L}{\partial h_3^{(2)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial c_1} & \frac{\partial L}{\partial c_2} \end{bmatrix} \begin{bmatrix} w_{21}^{(3)} & w_{31}^{(3)} \\ w_{22}^{(3)} & w_{32}^{(3)} \end{bmatrix} = dc \times W_3[1:]^T$$

Note $W_3[1:]$ represents all the columns from the second column $= W_3[1:2]$

dH2=np.matmul(dc, W_3[1:].T)

Backpropagation: Gradient of W_2 in matrix form



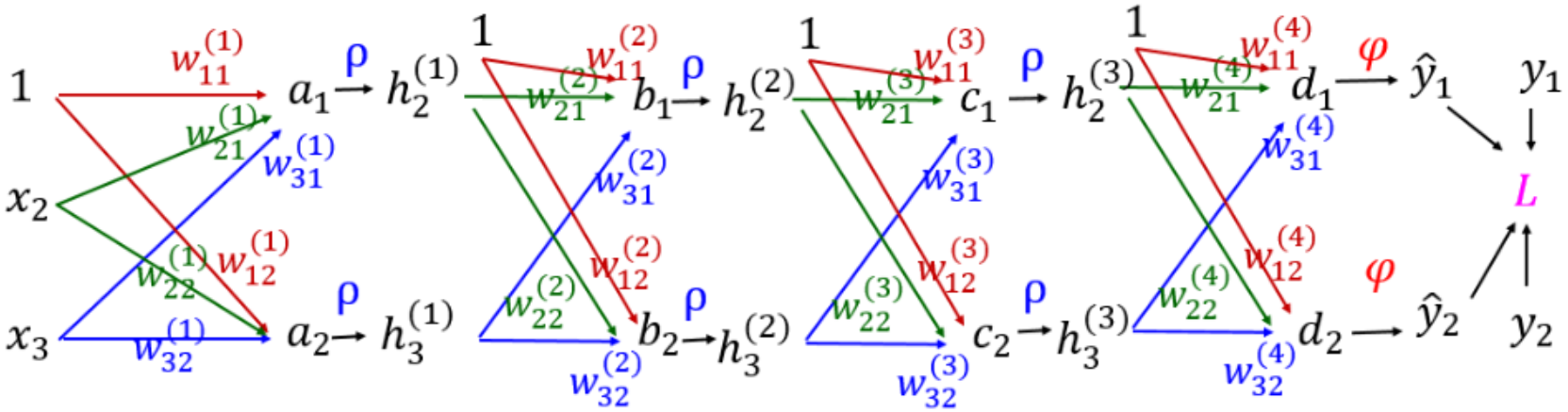
$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(2)}} & \frac{\partial L}{\partial w_{12}^{(2)}} \\ \frac{\partial L}{\partial w_{21}^{(2)}} & \frac{\partial L}{\partial w_{22}^{(2)}} \\ \frac{\partial L}{\partial w_{31}^{(2)}} & \frac{\partial L}{\partial w_{32}^{(2)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{11}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{12}^{(2)}} \\ \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{21}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{22}^{(2)}} \\ \frac{\partial L}{\partial b_1}\frac{\partial b_1}{\partial w_{31}^{(2)}} & \frac{\partial L}{\partial b_2}\frac{\partial b_2}{\partial w_{32}^{(2)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial b_1}1 & \frac{\partial L}{\partial b_2}1 \\ \frac{\partial L}{\partial b_1}h_2^{(1)} & \frac{\partial L}{\partial b_2}h_2^{(1)} \\ \frac{\partial L}{\partial b_1}h_3^{(1)} & \frac{\partial L}{\partial b_2}h_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ h_2^{(1)} \\ h_3^{(1)} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial b_1} & \frac{\partial L}{\partial b_2} \end{bmatrix} = [1H_1]^T \times db$$

dH2=np.matmul(dc, $W_3[1:]$.T)

db=dH2 $*\rho'$(H2)

dW2 = np.matmul(prepend_bias(H1).T, db)/X.shape[0]

Backpropagation: Gradient of W_1 in matrix form



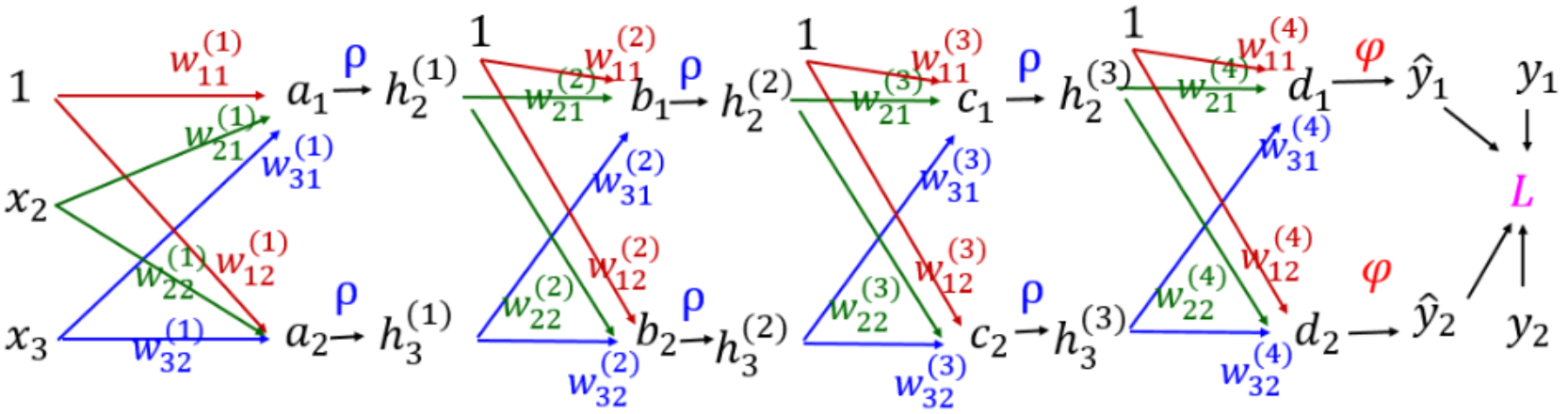
$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial w_{12}^{(1)}} \\ \frac{\partial L}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial w_{22}^{(1)}} \\ \frac{\partial L}{\partial w_{31}^{(1)}} & \frac{\partial L}{\partial w_{32}^{(1)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{12}^{(1)}} \\ \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{22}^{(1)}} \\ \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{31}^{(1)}} & \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_{32}^{(1)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1}1 & \frac{\partial L}{\partial a_2}1 \\ \frac{\partial L}{\partial a_1}x_2 & \frac{\partial L}{\partial a_2}x_2 \\ \frac{\partial L}{\partial a_1}x_3 & \frac{\partial L}{\partial a_2}x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial a_1} & \frac{\partial L}{\partial a_2} \end{bmatrix}$$

dH1=np.matmul(db, $W_2[1:].T$)

da=dH1 $*\rho'$(H1)

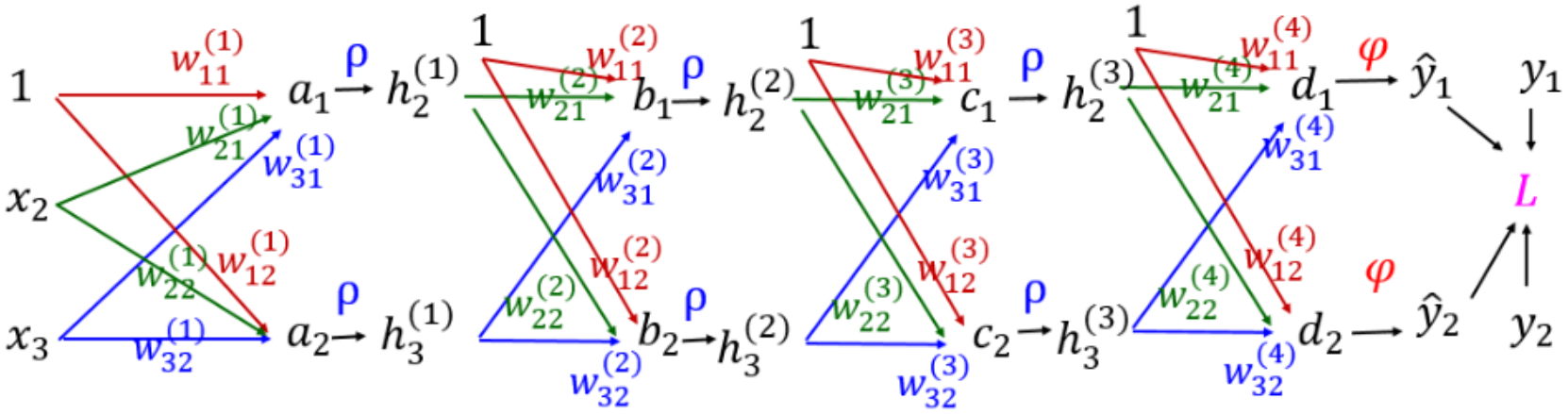
dW1 = np.matmul(prepend_bias(X).T, da)/X.shape[0]

Forward using ReLU



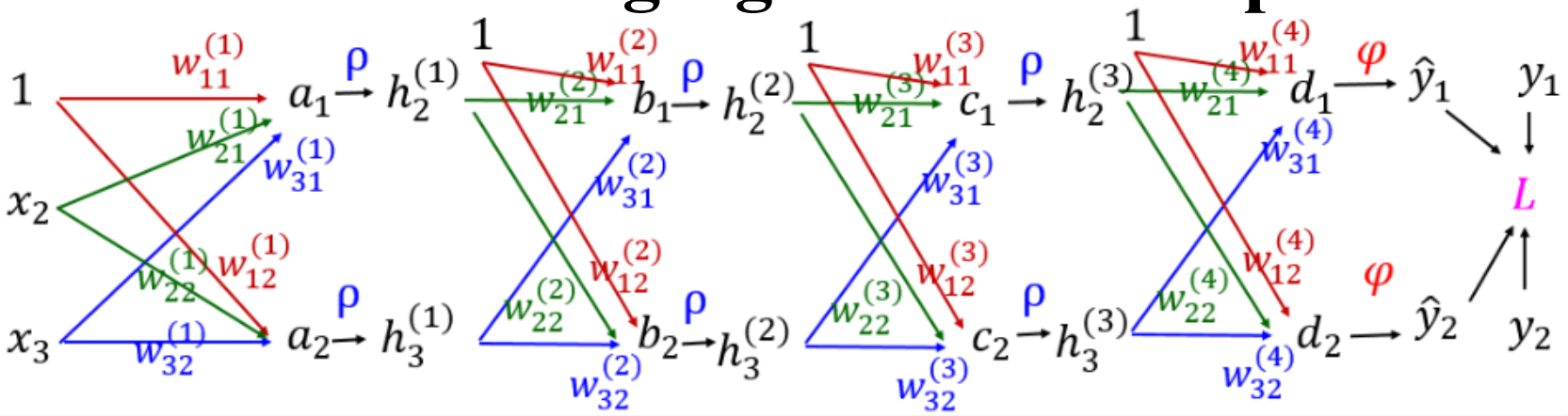

```
def forward(X, w1, w2, w3, w4):
    a  = np.matmul(prepend_bias(X), w1)     # (5 x 26) x (26 x 20) = 5 x 20
    h1 = ReLU(a)                            # 5 x 20
    b  = np.matmul(prepend_bias(h1), w2)    # (5 x 21) x (21 x 20) = 5 x 20
    h2 = ReLU(b)                            # 5 x 20
    c  = np.matmul(prepend_bias(h2), w3)    # (5 x 21) x (21 x 20) = 5 x 20
    h3 = ReLU(c)                            # 5 x 20
    d  = np.matmul(prepend_bias(h3), w4)    # (5 x 21) x (21 x 20) = 5 x 20
    y_hat = softmax(d)
    return (y_hat, d, h3, c, h2, b, h1, a)
```

Backpropagation



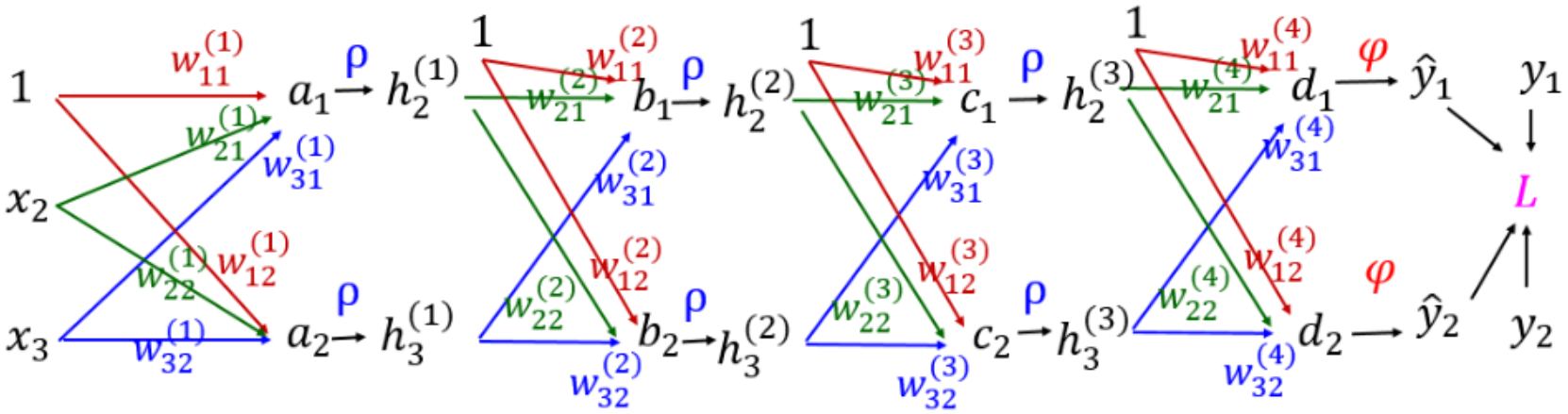

```python
def back(X, Y, y_hat, w1, w2, w3, w4, a, b, c, d, h1, h2, h3):
    dw4 = np.matmul(prepend_bias(h3).T, y_hat-Y)/X.shape[0] # (21 x 5) x (5 x 5)  = 21 x 5
    dh3 = np.matmul(y_hat - Y, w4[1:].T)                    # (5 x 5)  x (5 x 20) = 5  x 20
    dc  = dh3 * (h3 > 0)                                    # (5 x 20) * (5 x 20) = 5  x 20
    dw3 = np.matmul(prepend_bias(h2).T, dc)/X.shape[0]      # (21 x 5) x (5 x 20) = 21 x 20
    dh2 = np.matmul(dc, w3[1:].T)                           # (5 x 20) x (20 x 20)= 5  x 20
    db  = dh2 * (h2 > 0)                                    # (5 x 20) * (5 x 20) = 5  x 20
    dw2 = np.matmul(prepend_bias(h1).T, db)/X.shape[0]      # (21 x 5) x (5 x 20) = 21 x 20
    dh1 = np.matmul(db, w2[1:].T)                           # (5 x 20) x (20 x 20)= 5  x 20
    da  = dh1 * (h1 > 0)                                    # (5 x 20) x (20 x 20)= 5  x 20
    dw1 = np.matmul(prepend_bias(X).T, da)/X.shape[0]       # (26 x 5) x (5  x 20)= 26 x 20
    return (dw1, dw2, dw3, dw4)
```

Forward using sigmoid and Dropout




```
def forward_dropout(X, w1, w2, w3, w4):
    a  = np.matmul(prepend_bias(X), w1)     # (5 x 26) x (26 x 20) = 5 x 20
    h1 = sigmoid(a)                         # 5 x 20
    h1 = h1 * Dropout(h1, 0.2)
    b  = np.matmul(prepend_bias(h1), w2)    # (5 x 21) x (21 x 20) = 5 x 20
    h2 = sigmoid(b)                         # 5 x 20
    h2 = h2 * Dropout(h2, 0.2)
    c  = np.matmul(prepend_bias(h2), w3)    # (5 x 21) x (21 x 20) = 5 x 20
    h3 = sigmoid(c)                         # 5 x 20
    h3 = h3 * Dropout(h3, 0.2)
    d  = np.matmul(prepend_bias(h3), w4)    # (5 x 21) x (21 x 20) = 5 x 20
    y_hat = softmax(d)
    return (y_hat, d, h3, c, h2, b, h1, a)
```

Backpropagation




```
def back_sigmoid(X, Y, y_hat, w1, w2, w3, w4, a, b, c, d, h1, h2, h3):
    dw4 = np.matmul(prepend_bias(h3).T, y_hat-Y)/X.shape[0]     # (21 x 5) x (5 x 5)  = 21 x 5
    dh3 = np.matmul(y_hat - Y, w4[1:].T)                        # (5 x 5)  x (5 x 20) = 5  x 20
    dc  = dh3 * sigmoid(h3) * (1-sigmoid(h3))                   # (5 x 20) * (5 x 20) = 5  x 20
    dw3 = np.matmul(prepend_bias(h2).T, dc)/X.shape[0]          # (21 x 5) x (5 x 20) = 21 x 20
    dh2 = np.matmul(dc, w3[1:].T)                               # (5 x 20) x (20 x 20)= 5  x 20
    db  = dh2 * sigmoid(h2) * (1-sigmoid(h2))                   # (5 x 20) * (5 x 20) = 5  x 20
    dw2 = np.matmul(prepend_bias(h1).T, db)/X.shape[0]          # (21 x 5) x (5 x 20) = 21 x 20
    dh1 = np.matmul(db, w2[1:].T)                               # (5 x 20) x (20 x 20)= 5  x 20
    da  = dh1 * sigmoid(h1) * (1-sigmoid(h1))                   # (5 x 20) x (20 x 20)= 5  x 20
    dw1 = np.matmul(prepend_bias(X).T, da)/X.shape[0]           # (26 x 5) x (5  x 20)= 26 x 20
    return (dw1, dw2, dw3, dw4)
```

Homework #10-1

- Modify the code Lecture 10_4_testing.jpynb as follows:

1. Replace the functions of forward and back in Lecture 10_4_testing.jpynb with forward_dropout and back_sigmoid in Lecture 10_6_drop_out.jpynb
2. Replace the sigmoid activation function in Lecture 10_4_testing.jpynb with ReLU,
3. Rescale the intensity of MNIST training and testing data to be between 0 and 1
4. Set the n_hidden_nodes=50, iterations=2000, lr=0.01
5. Compute the accuracy of MNIST test data to achieve at least 90%
6. Plot the losses of training and testing against iterations

Homework #10-2

- Modify your code in Homework # 10-1 by adding the function configuration in Lecture 10_5_development.jpynb to determine a set of hypermeters including

1. The number of hidden node for each hidden layer,
2. learning rate,
3. batch size
4. epoch

so that the accuracy of MNIST test data can achieve at least 98%

Deadline of Homework #10-1 and #10-2: 2022/12/12 3:30pm