# Neural Networks

CS344
Deep Learning

WELLESLEY

# Supervised Machine Learning

**Input (Observed data)**

X

| | Years Experience | Has Required Skills | Favorite Pet (Dog/Cat/Fish) | GPA | Embezzled |
|---|---|---|---|---|---|
| Lavina | 6 | Yes | Cat | 2.8 | No |
| Alba | 2 | Yes | Dog | 3.7 | No |
| Brian | 0 | No | Cat | 0.3 | Yes |
| Zuri | 5 | Yes | Fish | 3.9 | No |
| Sarahi | 9 | No | Dog | 4.0 | No |
| Tao | 8 | No | Cat | 3.2 | No |
| Sakura | 1 | Yes | Dog | 2.5 | No |

$f(X) = y$

What is relationship?

**Output (Label)**

y

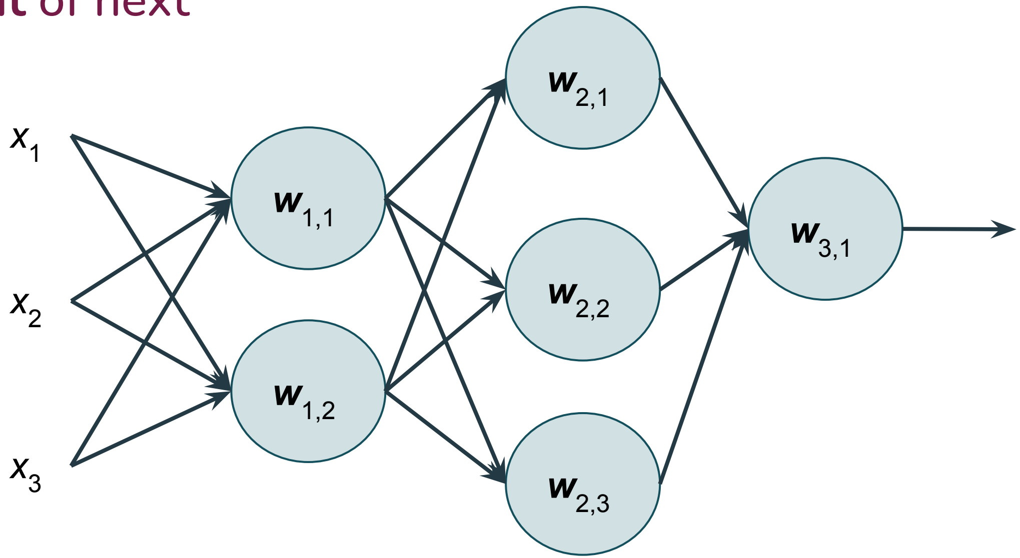
| Interview |
|---|
| Yes |
| Yes |
| No |
| Yes |
| Yes |
| No |
| Yes |

Linear models (like logistic regression) learn linear relationships

Non-linear models (like neural networks) learn more complicated relationships

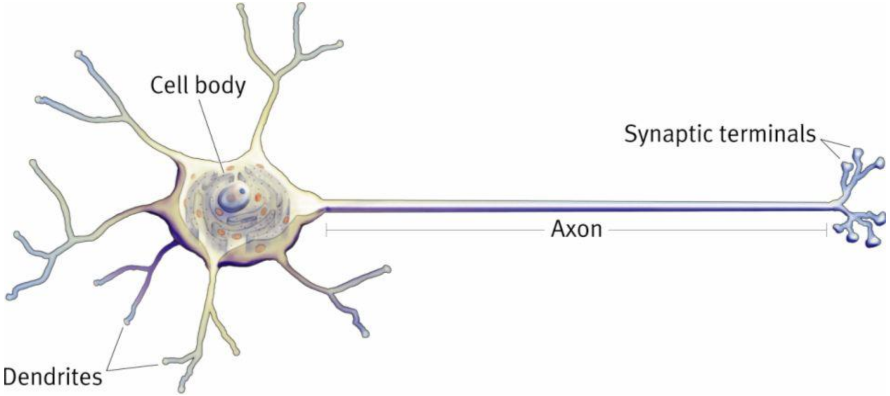
# What is an *artificial* neural network?

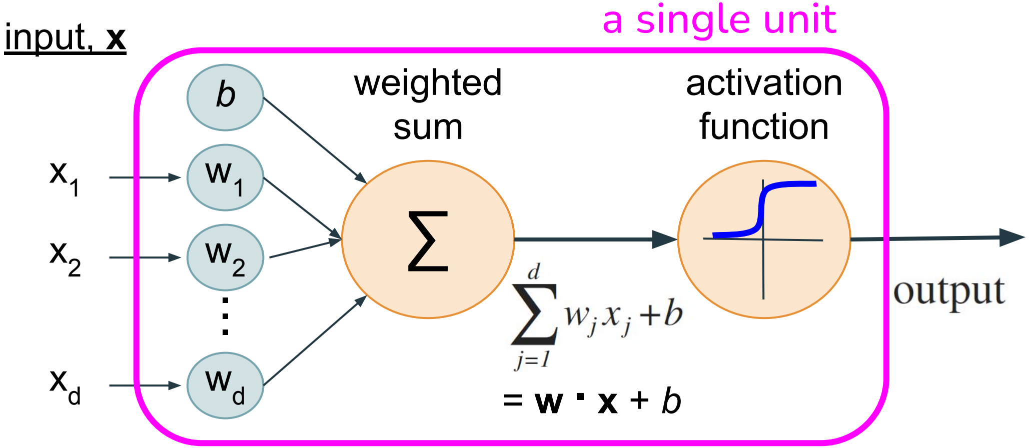
- Stacked layers of linear models
  - **Output** of each layer is **input** of next
- **Training**
  - Given labeled data and an architecture, learn the weight parameters
- **Prediction**
  - Given all the weight parameters, compute the final output (predicted class label)



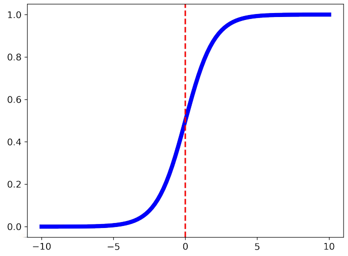
# Logistic Regression

Poor analogy
Not like a neuron



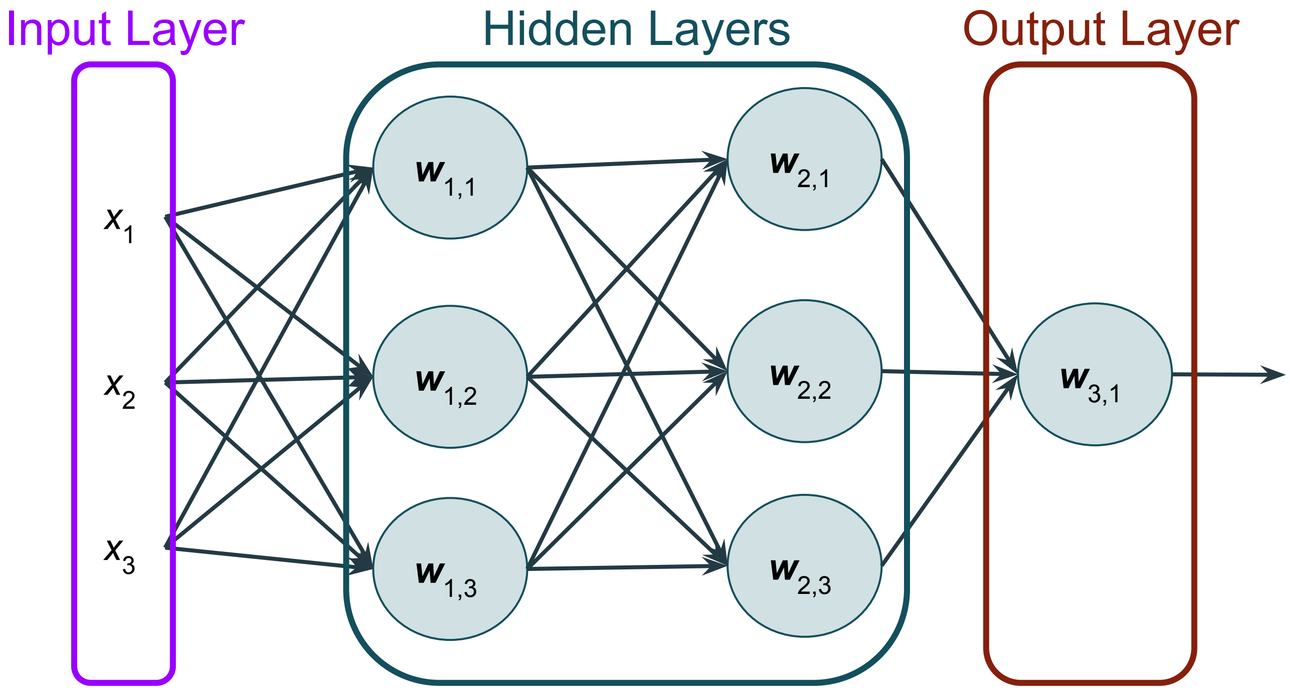

$$g(z) = \frac{1}{1+e^{-z}}$$



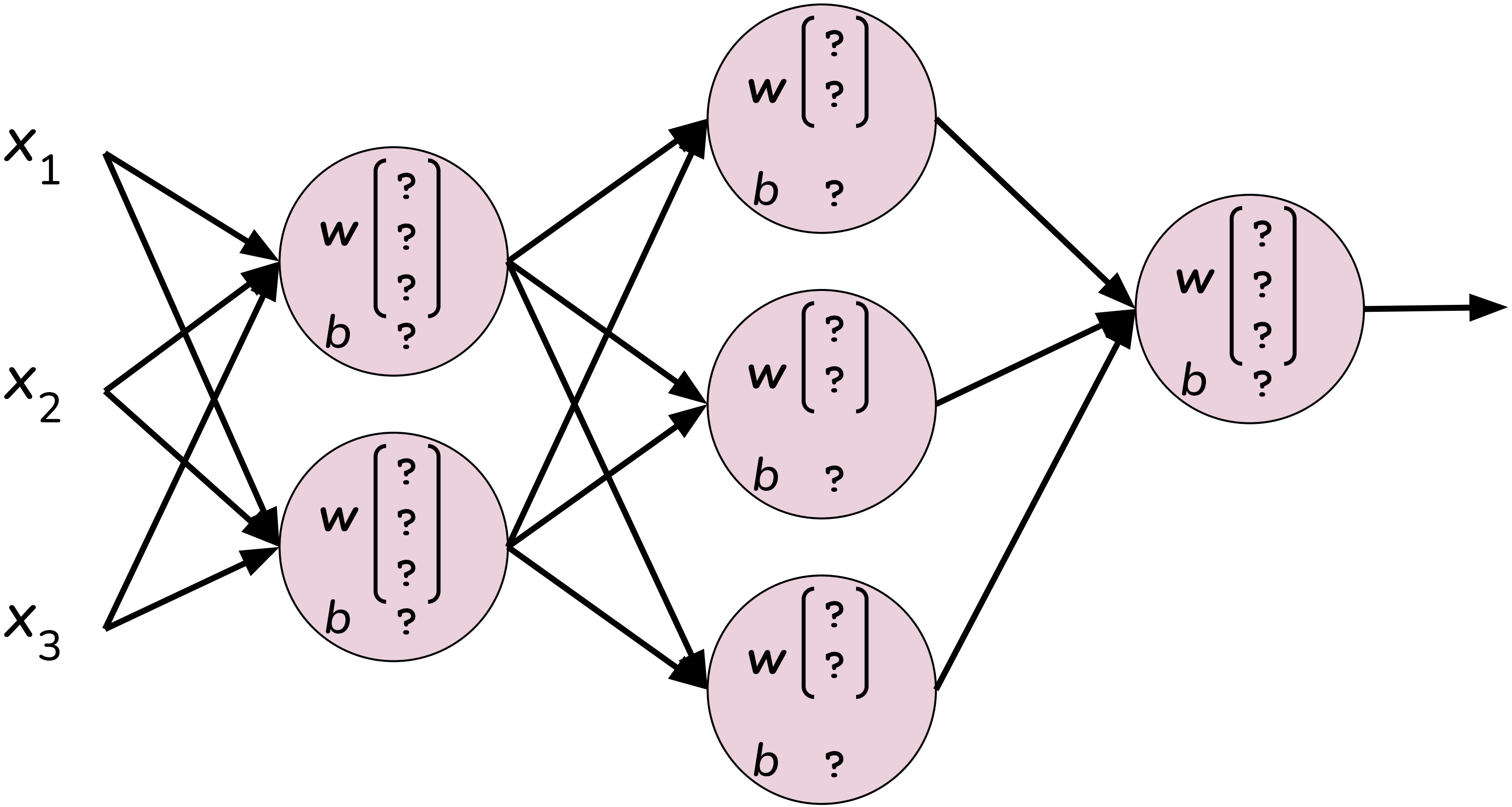

$$\text{output} = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$= \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

# Network Architecture

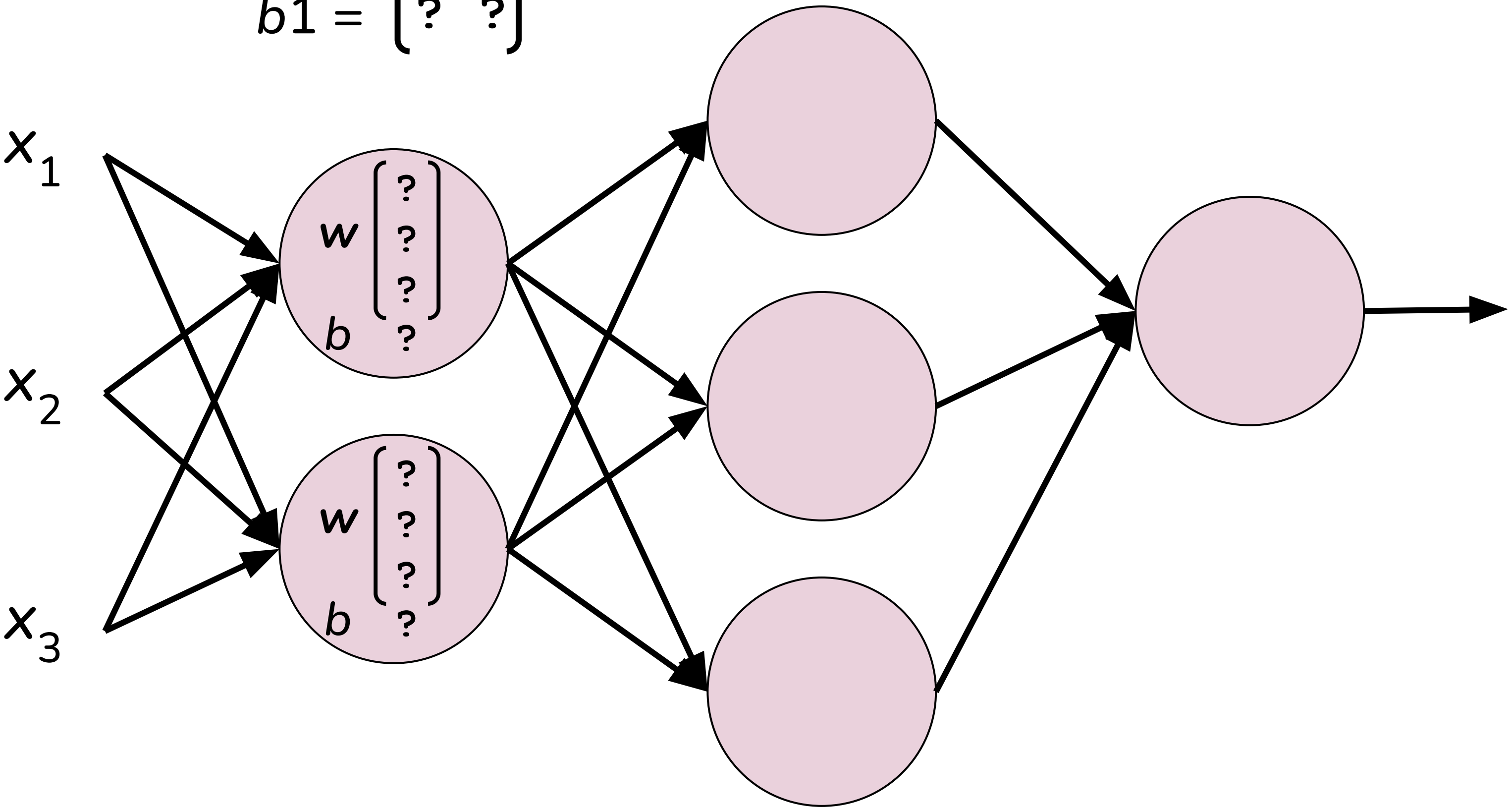


# Parameters



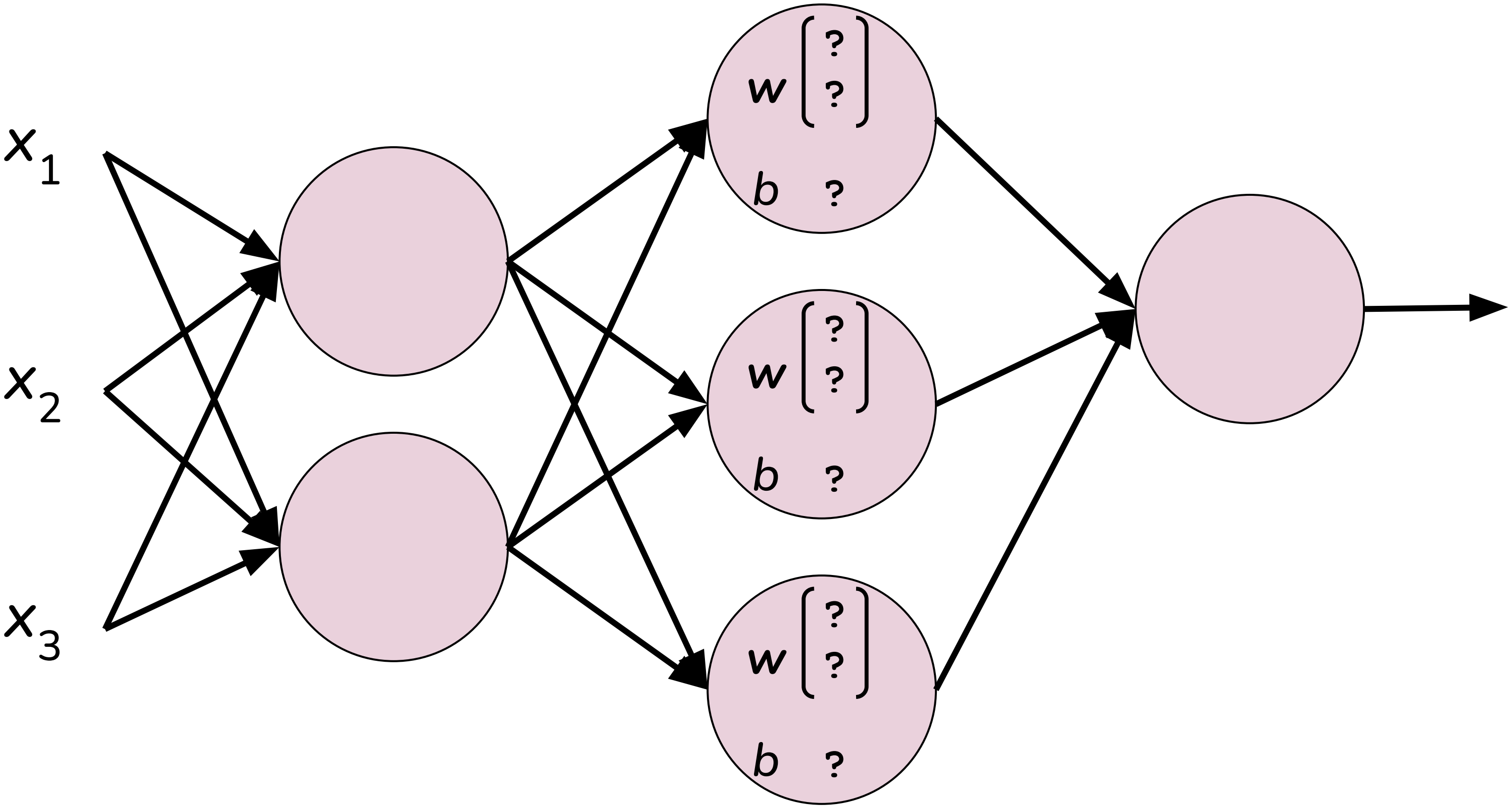
$$W1 = \begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \end{bmatrix}$$

$$b1 = \begin{bmatrix} ? & ? \end{bmatrix}$$



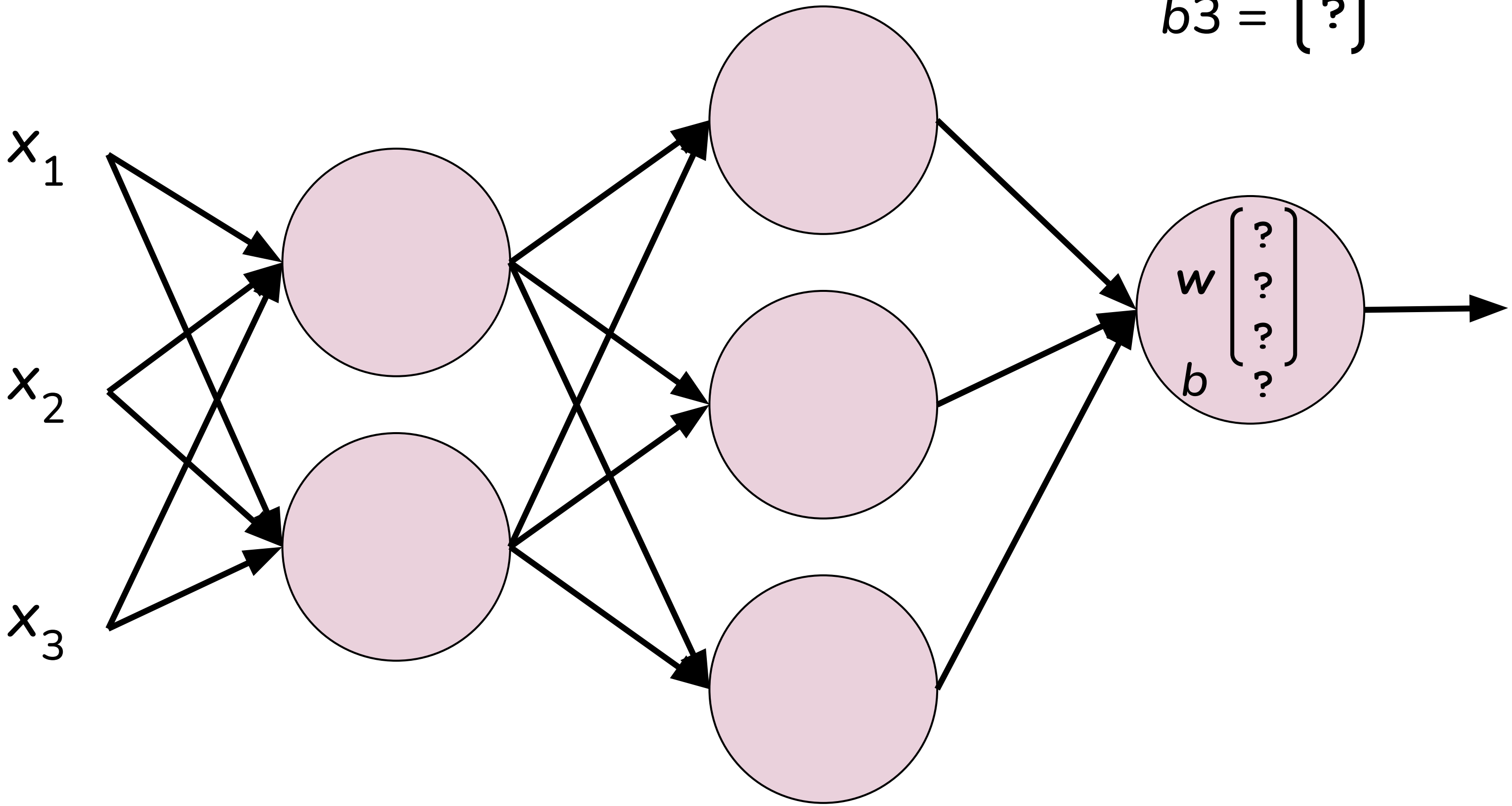
$$W2 = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \end{pmatrix}$$

$$b2 = \begin{pmatrix} ? & ? & ? \end{pmatrix}$$



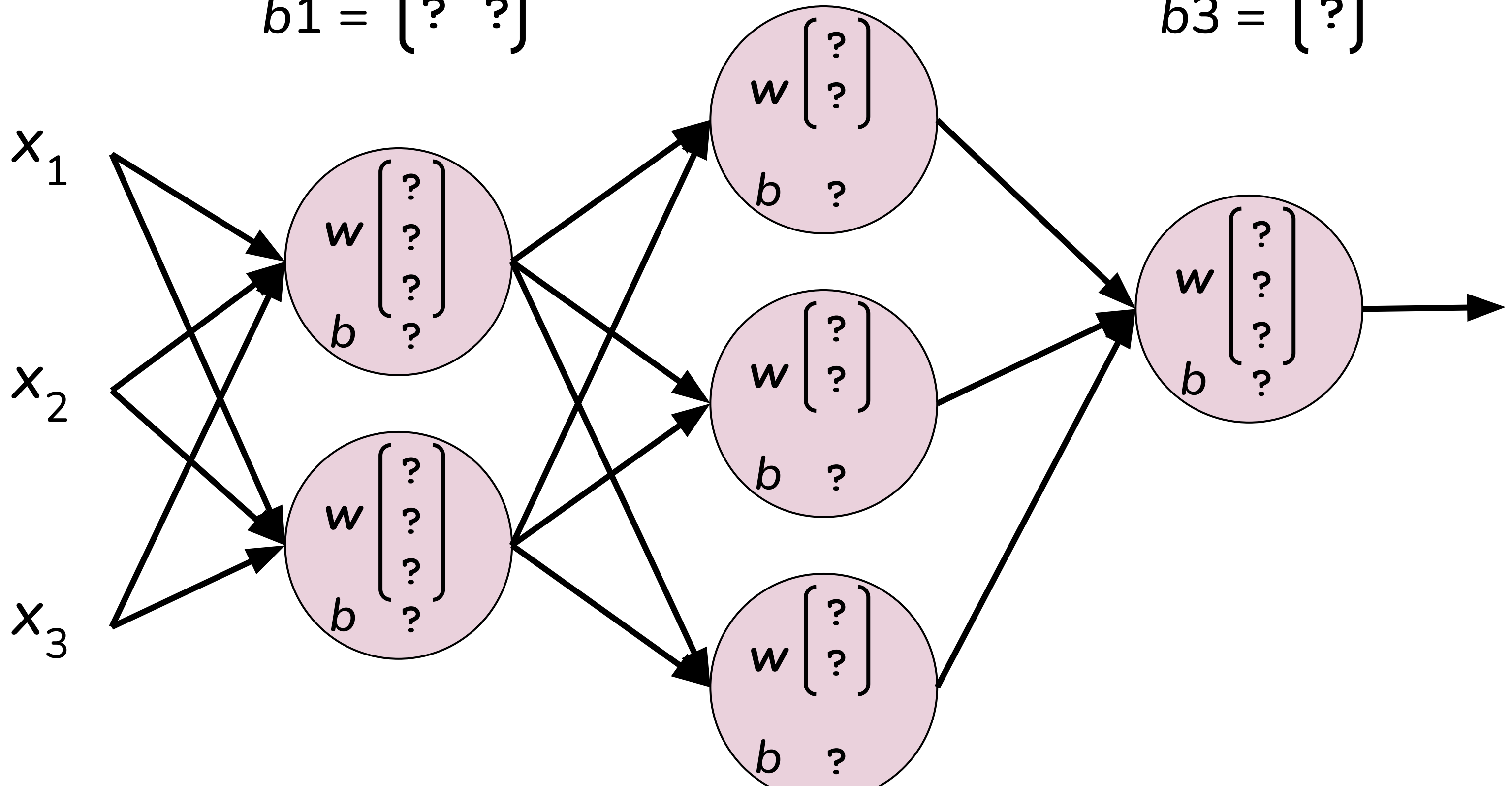
$$W3 = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix}$$

$$b3 = \begin{bmatrix} ? \end{bmatrix}$$



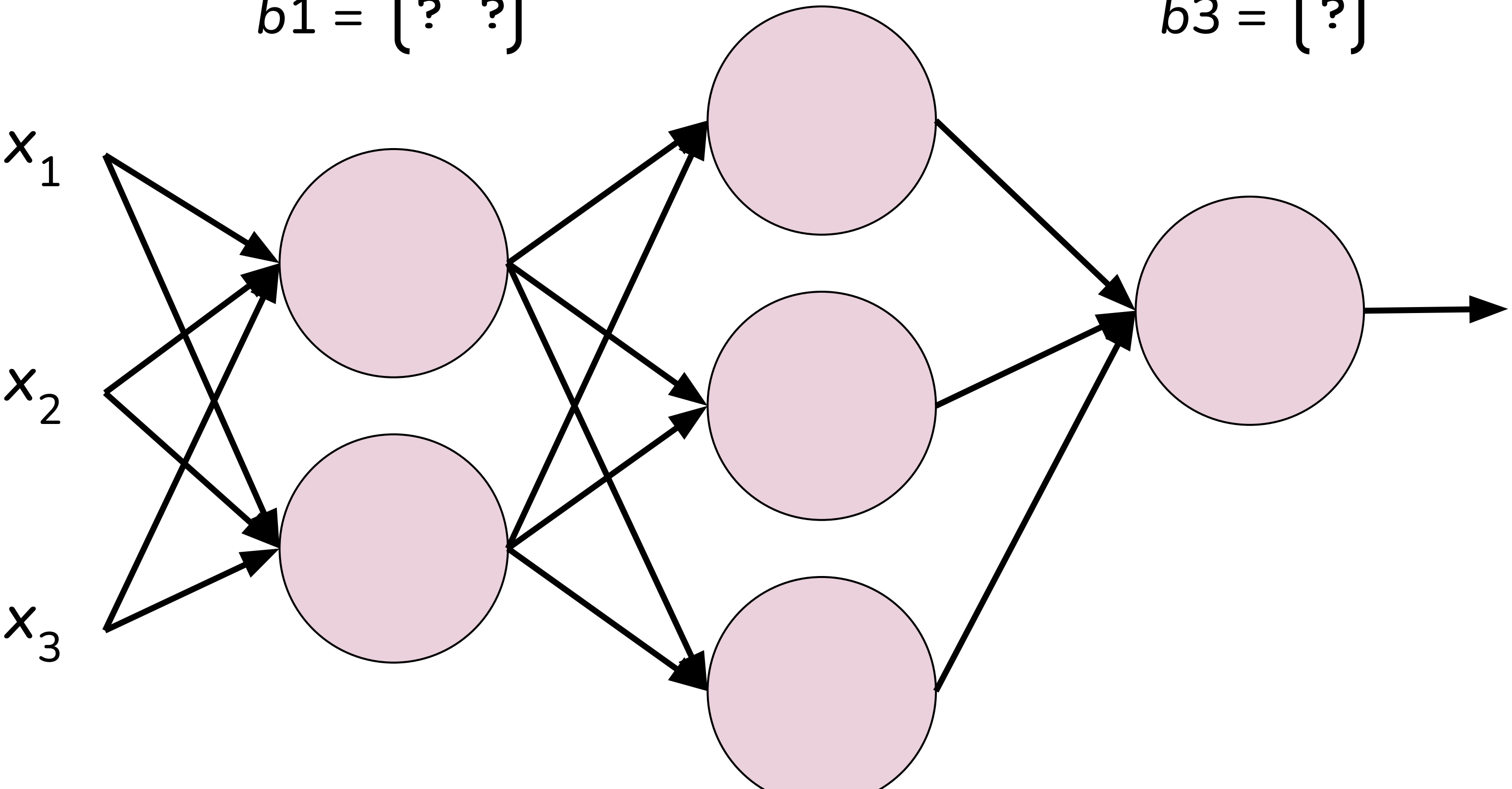
$$W1 = \begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \end{bmatrix} \qquad W2 = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \end{bmatrix} \qquad W3 = \begin{bmatrix} ? \\ ? \\ ? \end{bmatrix}$$

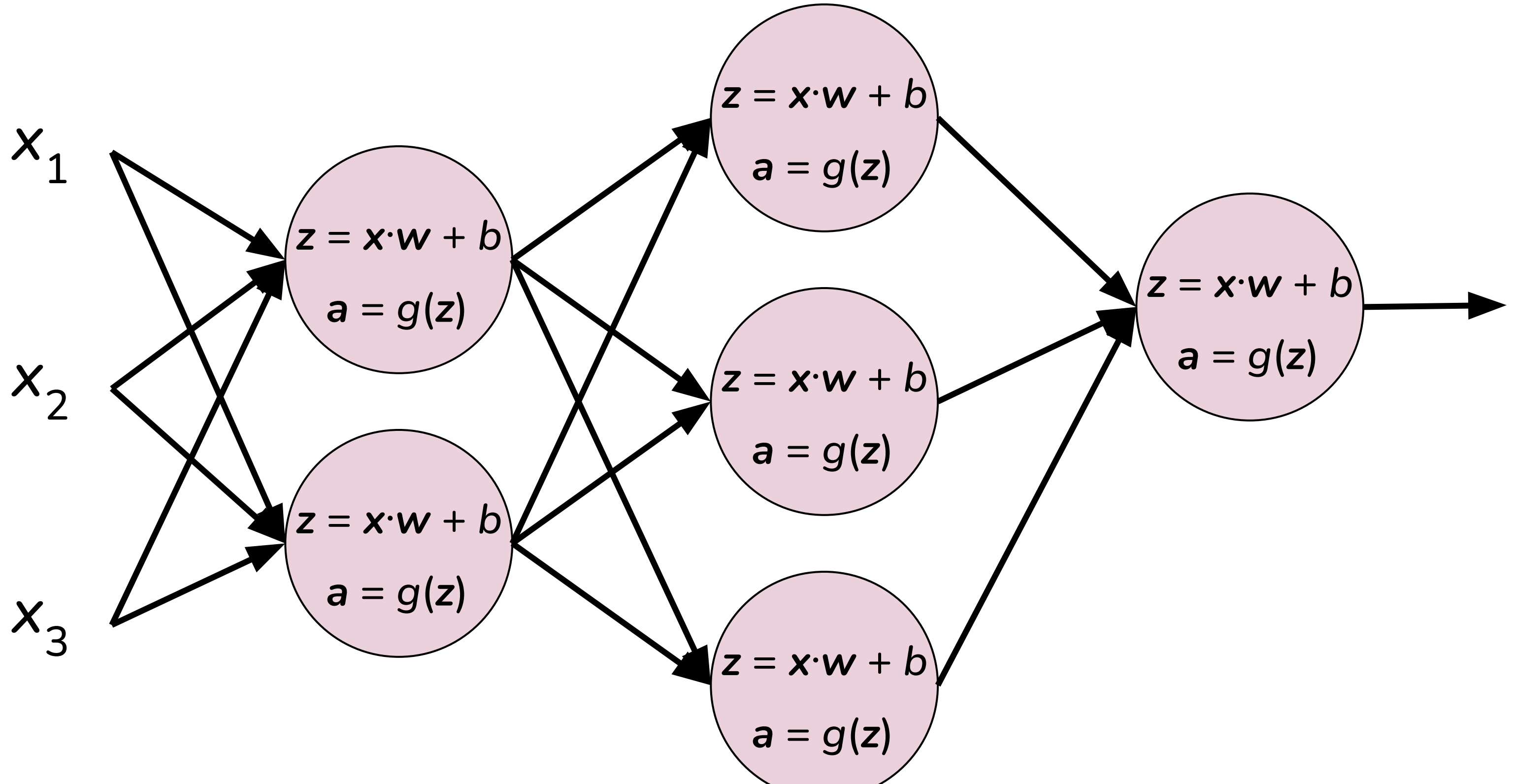
$$b1 = \begin{bmatrix} ? & ? \end{bmatrix} \qquad b2 = \begin{bmatrix} ? & ? & ? \end{bmatrix} \qquad b3 = \begin{bmatrix} ? \end{bmatrix}$$



$$W1 = \begin{pmatrix} ? & ? \\ ? & ? \\ ? & ? \end{pmatrix} \quad W2 = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \end{pmatrix} \quad W3 = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$$

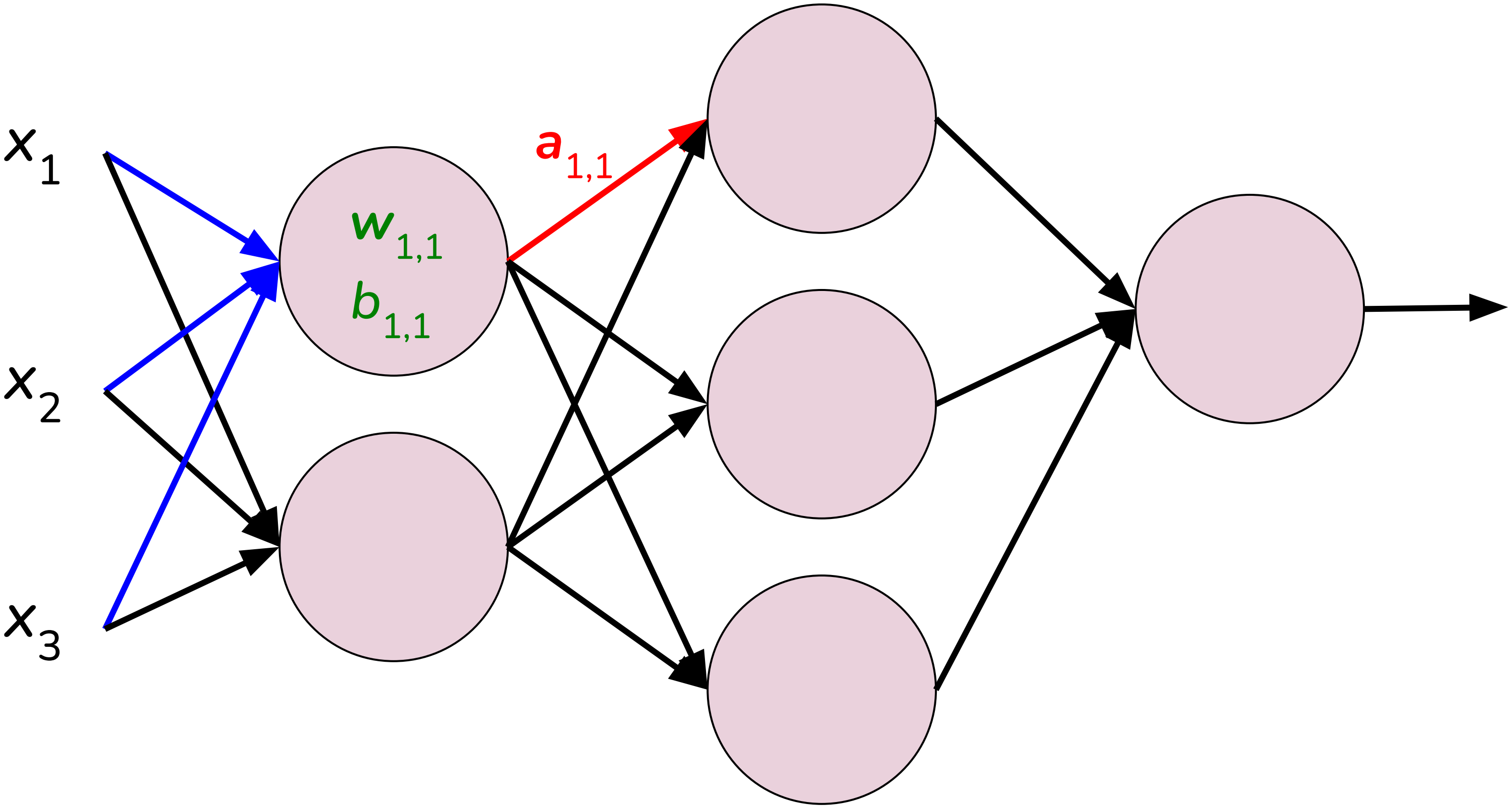
$$b1 = \begin{pmatrix} ? & ? \end{pmatrix} \quad b2 = \begin{pmatrix} ? & ? & ? \end{pmatrix} \quad b3 = \begin{pmatrix} ? \end{pmatrix}$$




x1
x2
x3
z = x·w + b
a = g(z)
z = x·w + b
a = g(z)
z = x·w + b
a = g(z)
z = x·w + b
a = g(z)
z = x·w + b
a = g(z)
z = x·w + b
a = g(z)

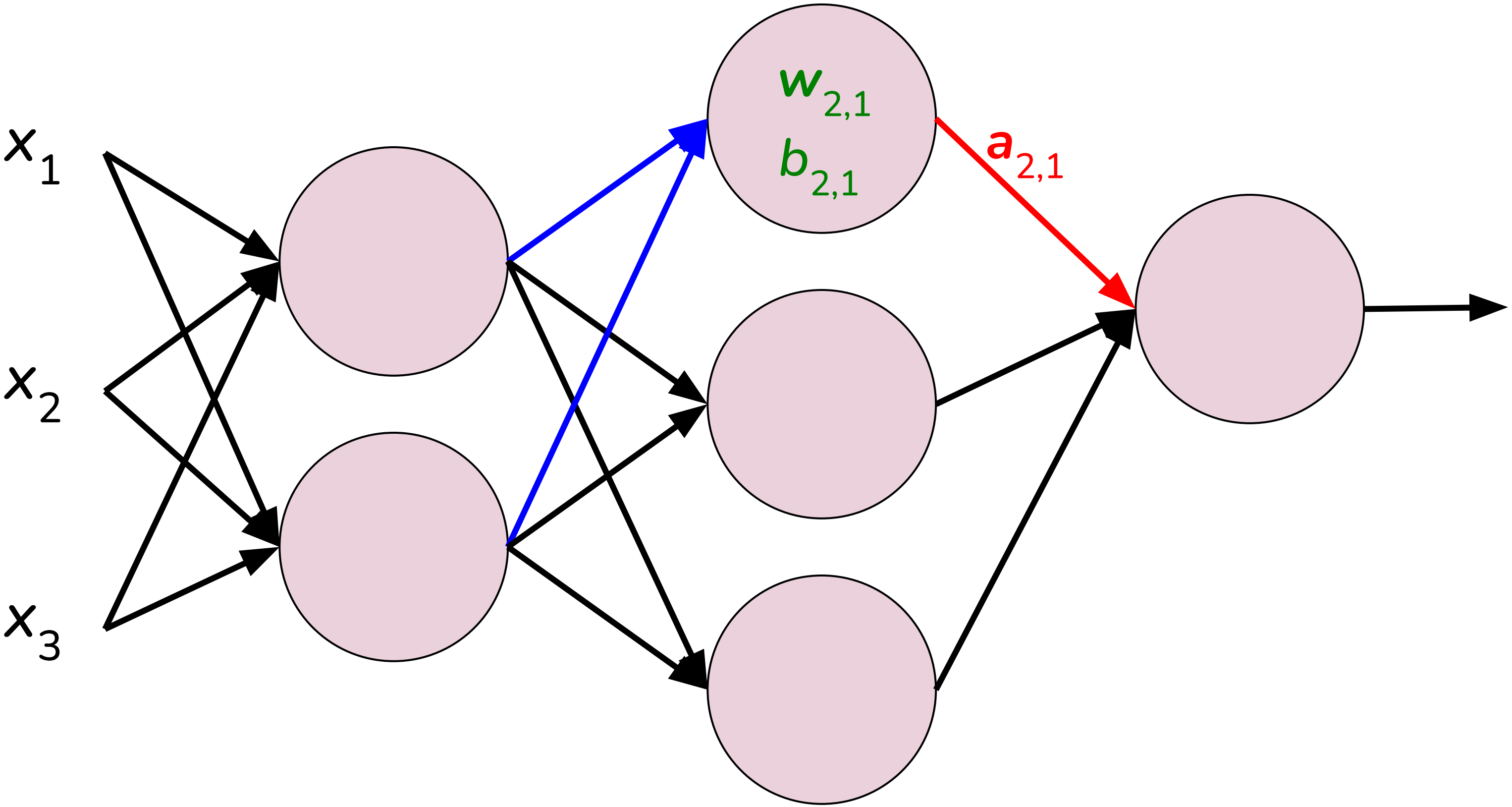
# Forward Propagation

$$a_{1,1} = g(\mathbf{x} \cdot w_{1,1} + b_{1,1})$$



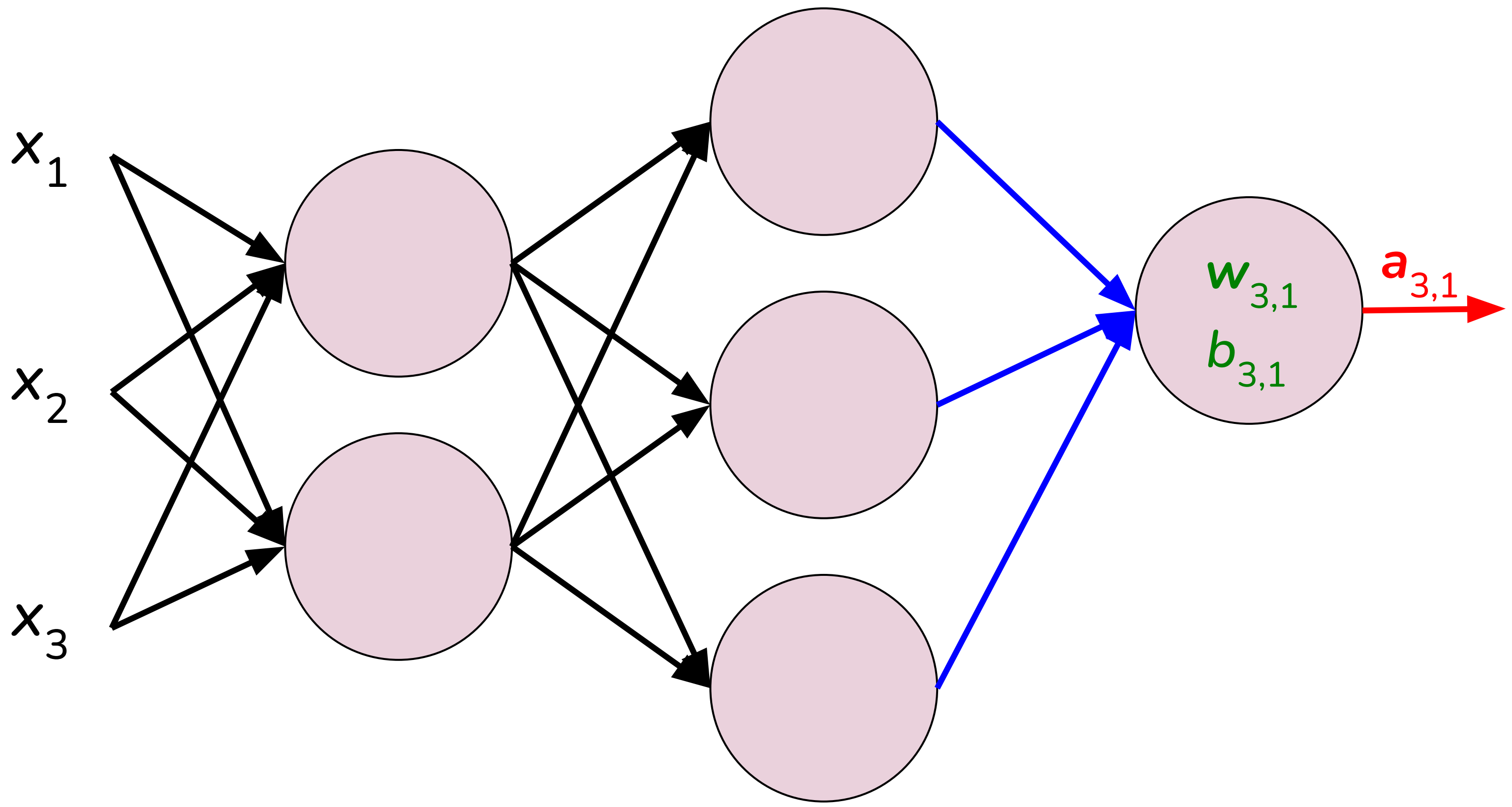
# Forward Propagation

$$a_{2,1} = g(a_1 \cdot w_{2,1} + b_{2,1})$$



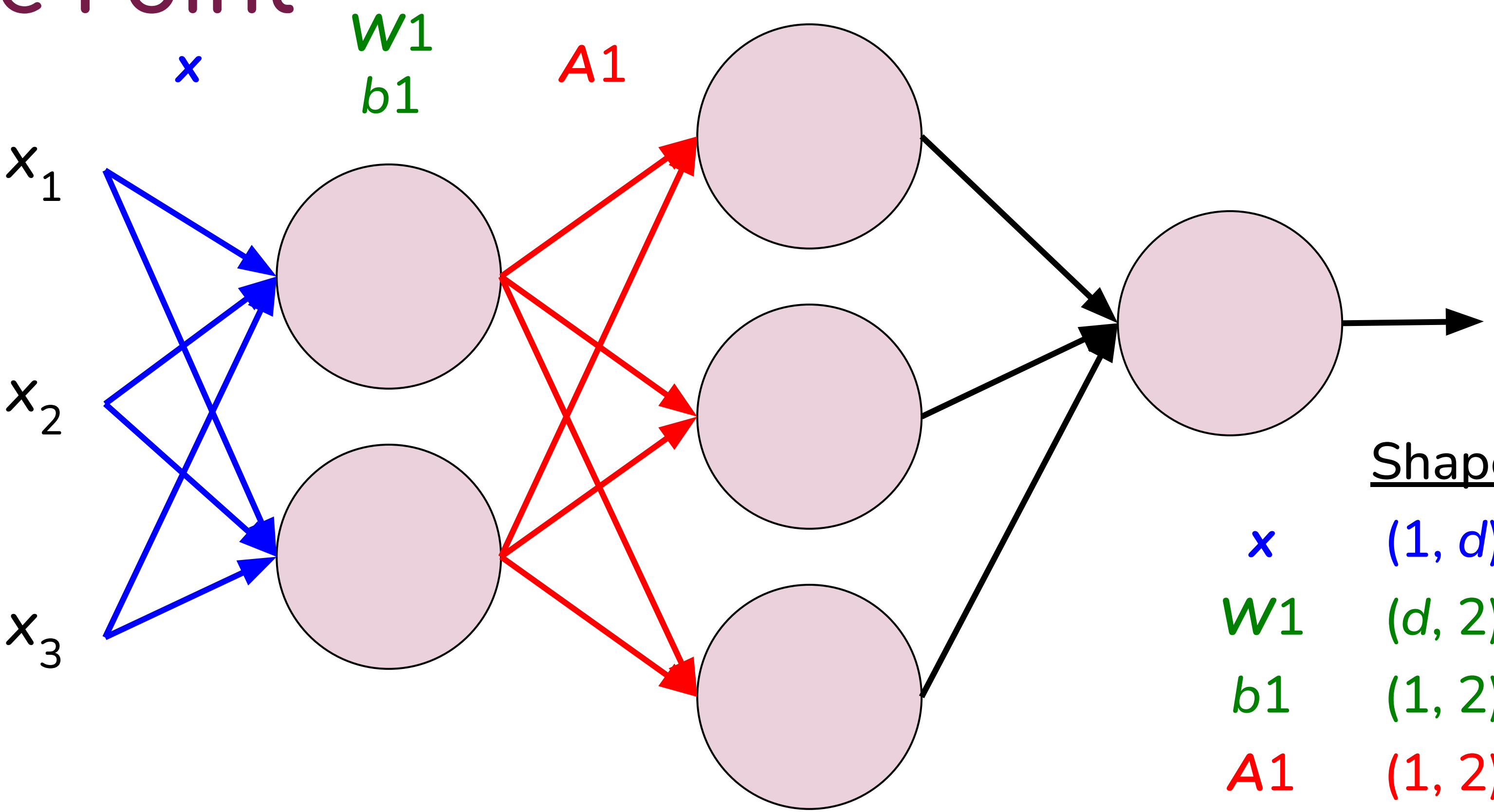
# Forward Propagation

$$a_{3,1} = g(a_2 \cdot w_{3,1} + b_{3,1})$$



# Forward Propagation: One Point

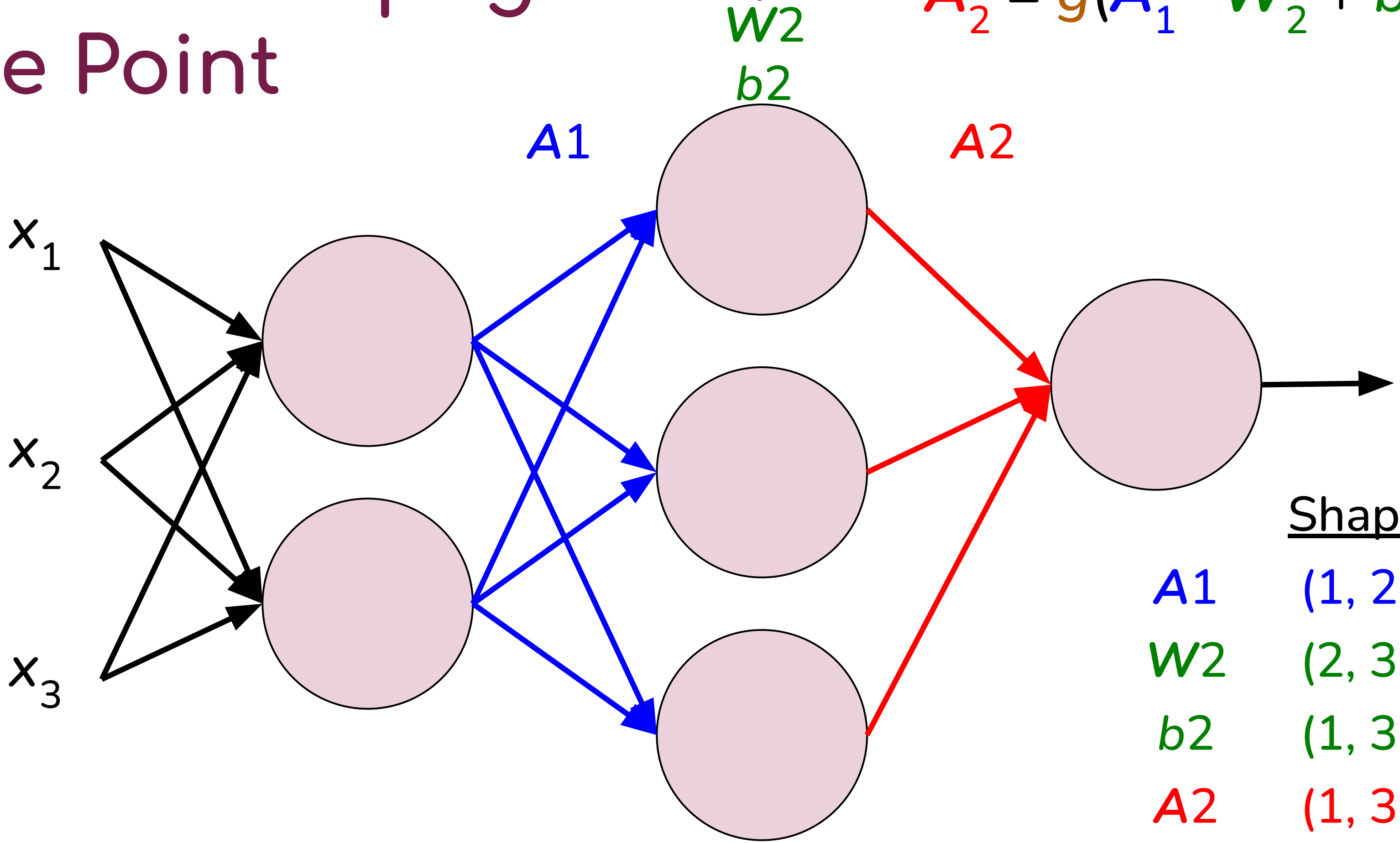
$$A_1 = g(x \cdot W_1 + b_1)$$




| Shape | |
|---|---|
| $x$ | $(1, d)$ |
| $W1$ | $(d, 2)$ |
| $b1$ | $(1, 2)$ |
| $A1$ | $(1, 2)$ |

# Forward Propagation: One Point

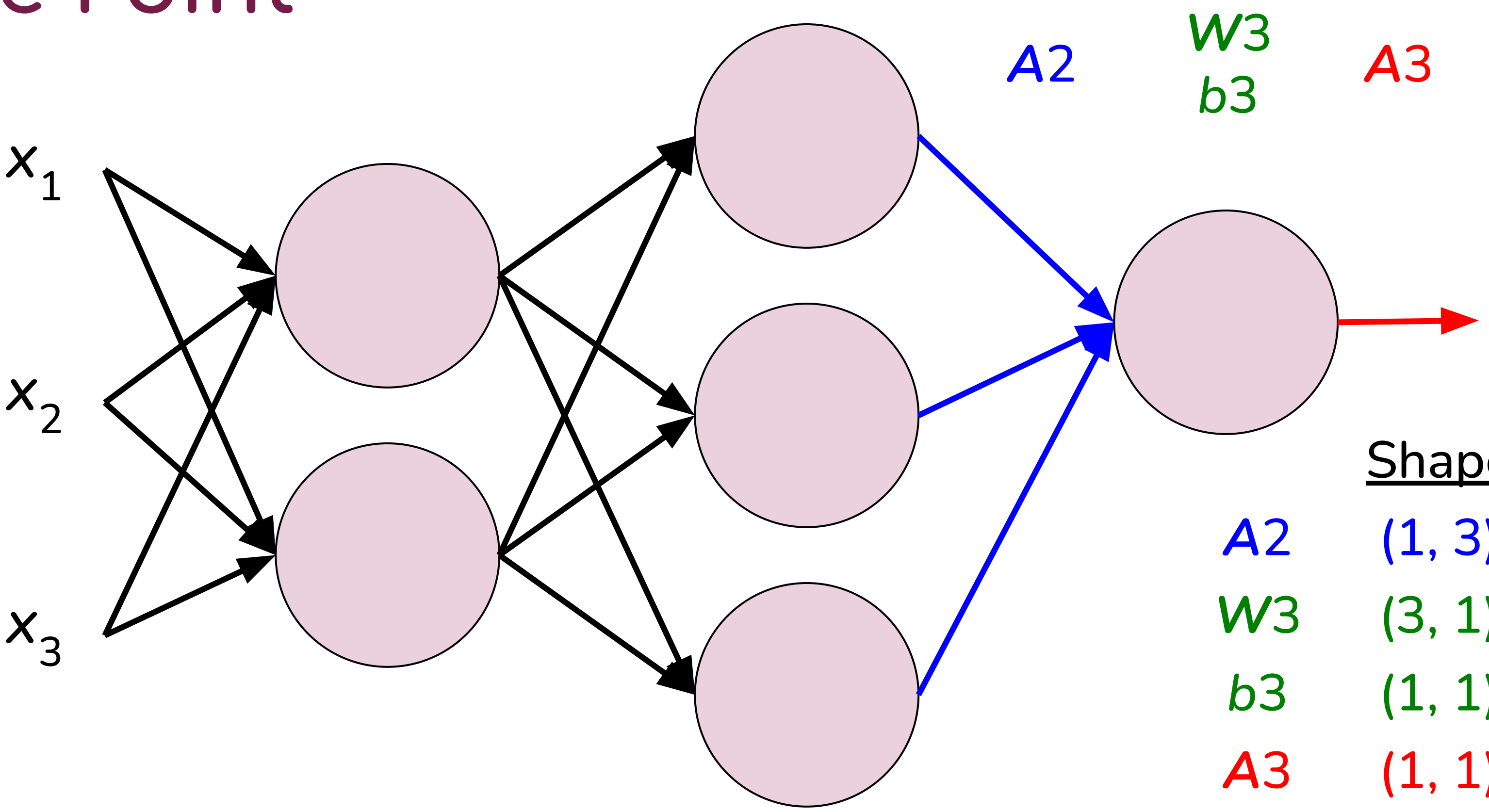
$$A_2 = g(A_1 \cdot W_2 + b_2)$$




| Shape | |
|---|---|
| A1 | (1, 2) |
| W2 | (2, 3) |
| b2 | (1, 3) |
| A2 | (1, 3) |

# Forward Propagation: One Point

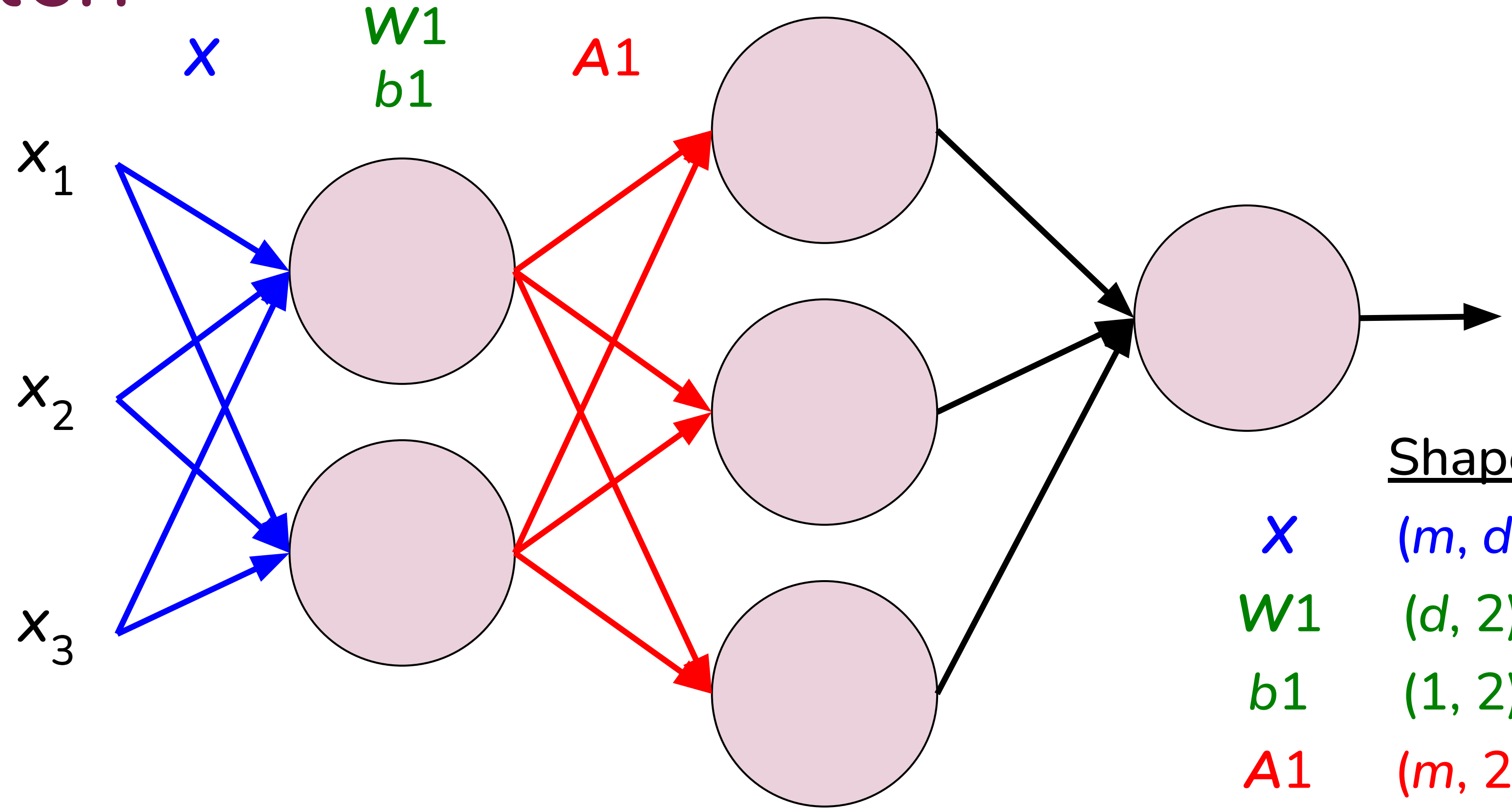
$$A_3 = g(A_2 \cdot W_3 + b_3)$$




| Shape | |
|---|---|
| A2 | (1, 3) |
| W3 | (3, 1) |
| b3 | (1, 1) |
| A3 | (1, 1) |

# Forward Propagation: Batch

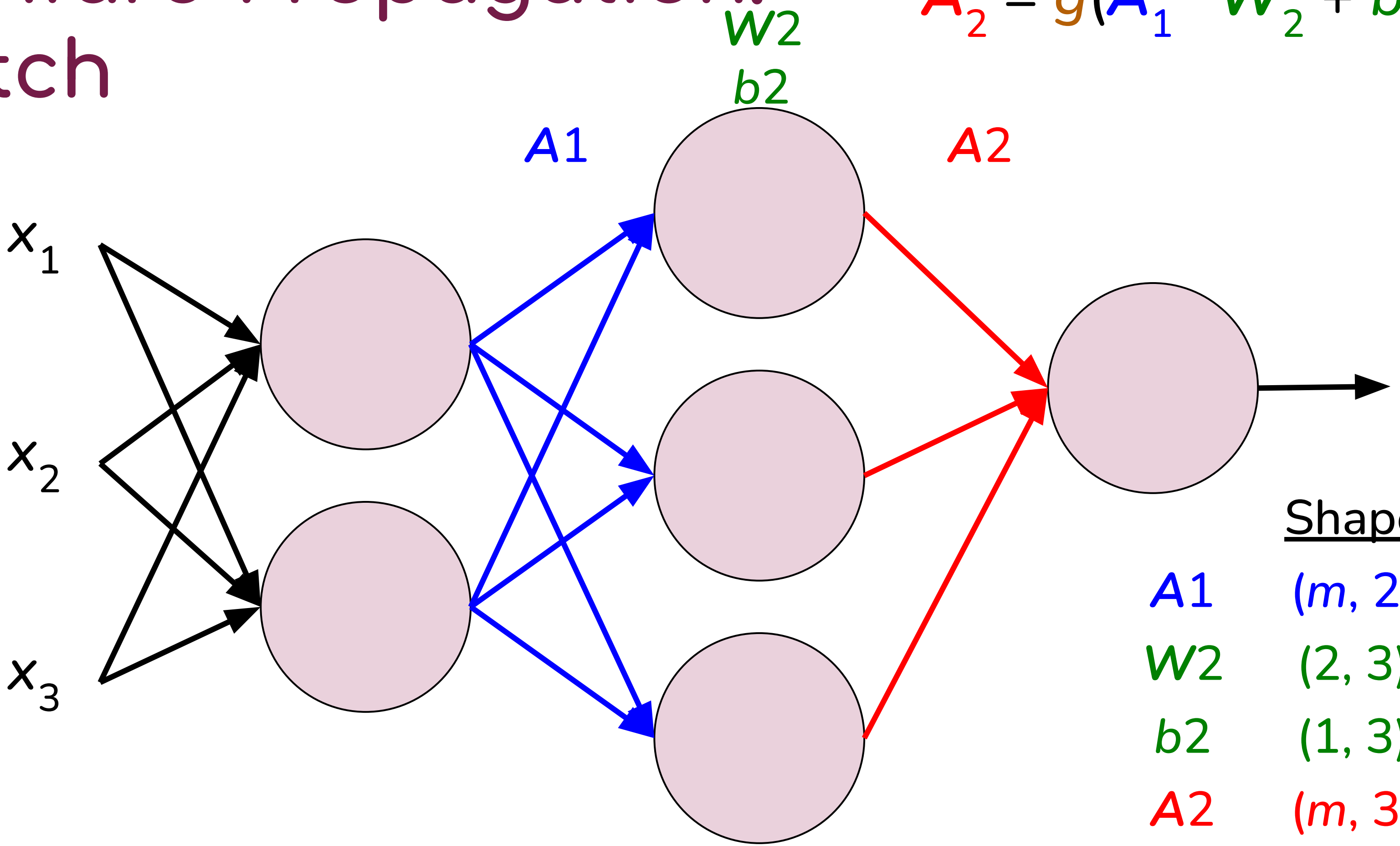
$$A_1 = g(X \cdot W_1 + b_1)$$




| Shape | |
|---|---|
| $X$ | $(m, d)$ |
| $W1$ | $(d, 2)$ |
| $b1$ | $(1, 2)$ |
| $A1$ | $(m, 2)$ |

# Forward Propagation: Batch

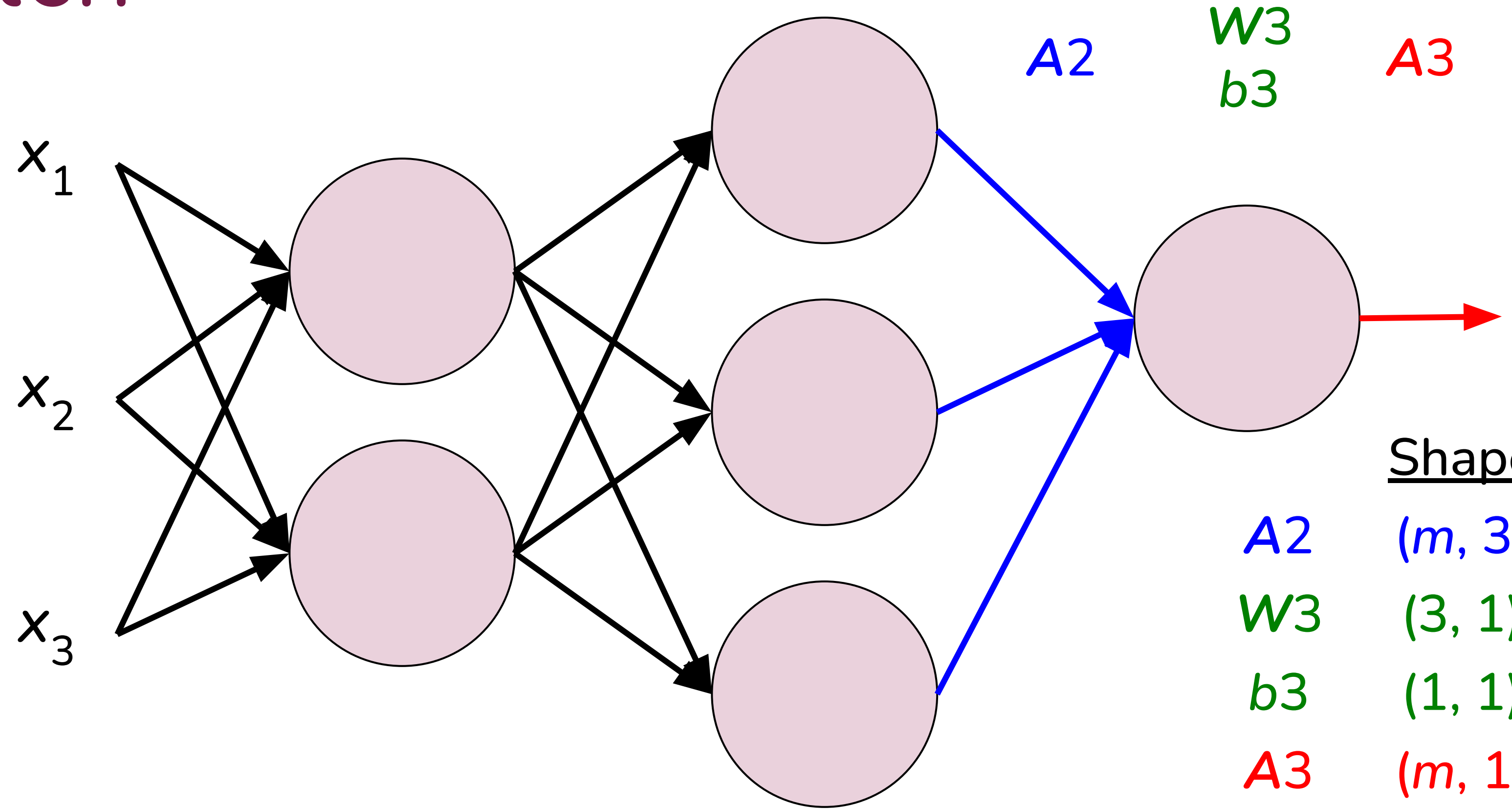
$$A_2 = g(A_1 \cdot W_2 + b_2)$$




| Shape | |
|---|---|
| A1 | $(m, 2)$ |
| W2 | $(2, 3)$ |
| b2 | $(1, 3)$ |
| A2 | $(m, 3)$ |

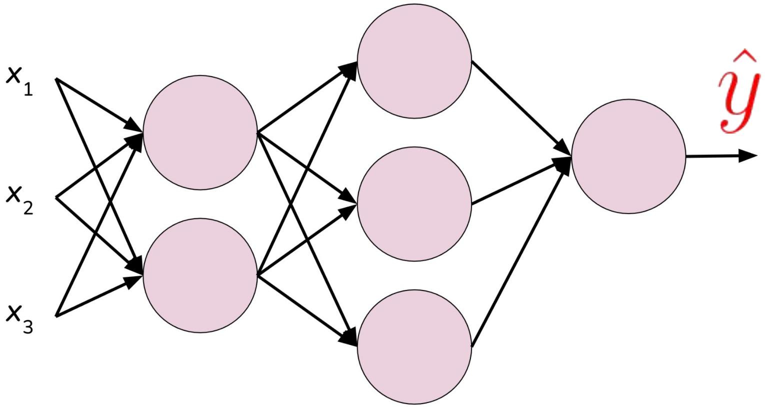
# Forward Propagation: Batch

$$A_3 = g(A_2 \cdot W_3 + b_3)$$




| | Shape |
|---|---|
| A2 | $(m, 3)$ |
| W3 | $(3, 1)$ |
| b3 | $(1, 1)$ |
| A3 | $(m, 1)$ |

# Loss and Cost




The *loss function*, $L$, quantifies the error, i.e., how far our prediction $\hat{y}$ is from the true label $y$

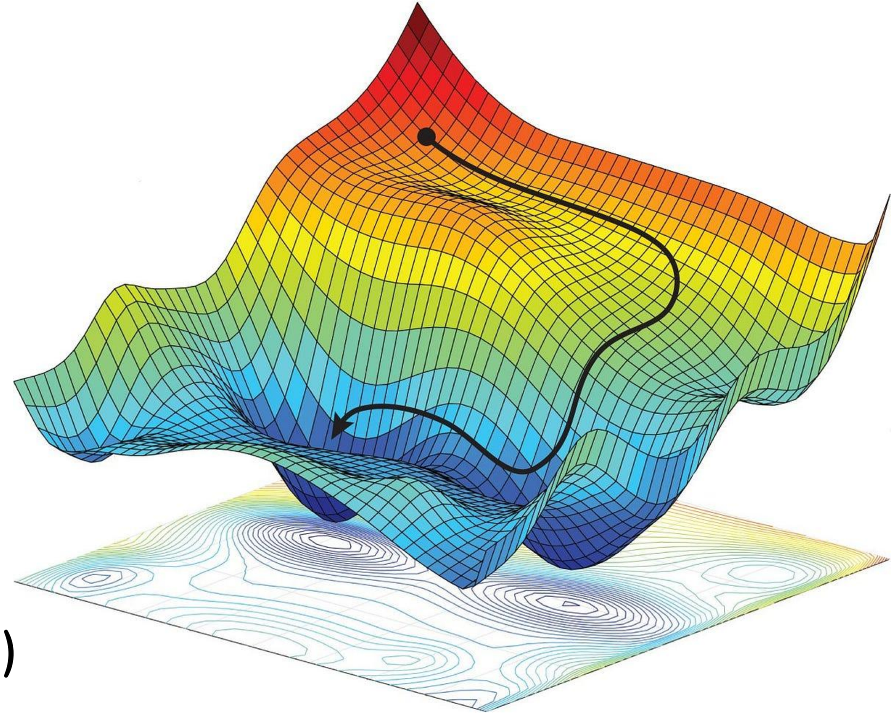
$$L = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

The *cost function*, $J$, is the average loss (error) over all data points

$$J = \frac{1}{m}\sum_{i=1}^{m} L \; = \frac{1}{m}\sum_{i=1}^{m} -y_i\log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$$

# Training

We want to find parameters (***W***'s and *b*'s) that minimize the cost, $J$



Gradient Descent Algorithm

- Initialize parameters (***W***'s and *b*'s)
- Repeat until converge:
  - Update parameters (***W***'s and *b*'s) to reduce the cost, $J$

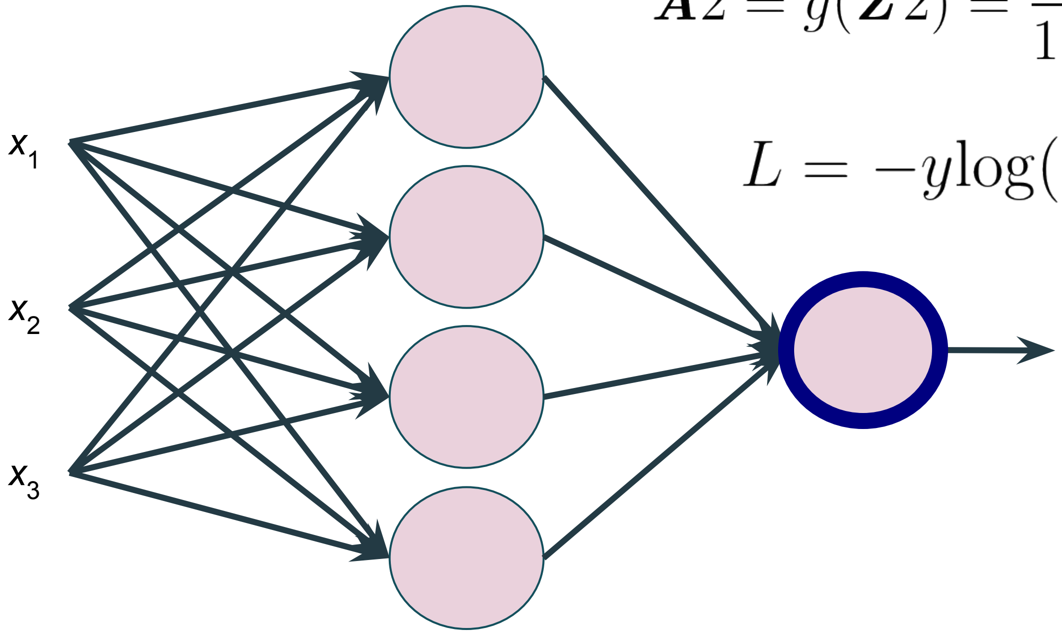
# Backpropagation

$$\boldsymbol{Z}2 = \boldsymbol{A}1 \cdot \boldsymbol{W}2 + b2$$

$$\boldsymbol{A}2 = g(\boldsymbol{Z}2) = \frac{1}{1 + e^{-\boldsymbol{Z}2}}$$

$$L = -y\log(\boldsymbol{A}2) - (1 - y)\log(1 - \boldsymbol{A}2)$$

$$J = \frac{1}{m}\sum_{i=1}^{m} L$$



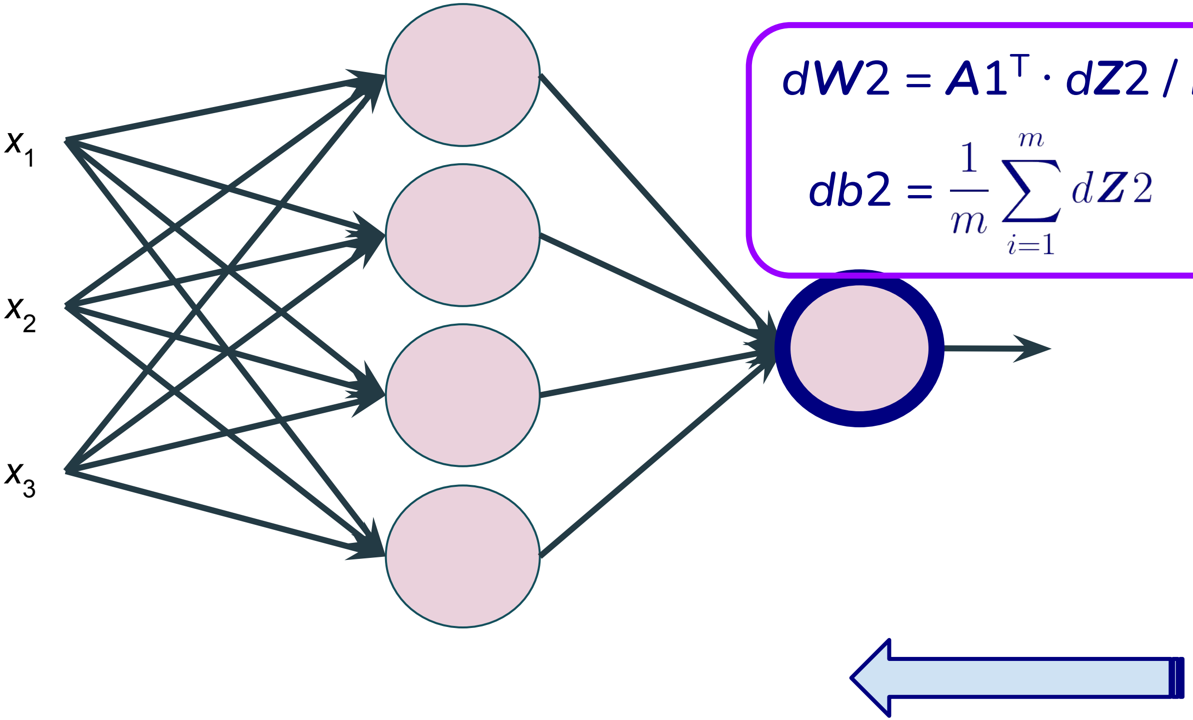
# Backpropagation

$dZ2 = A2 - y$

$dW2 = A1^{T} \cdot dZ2 \,/\, m$

$db2 = \frac{1}{m}\sum_{i=1}^{m} dZ2$

Gradients for updating $W2$ and $b2$



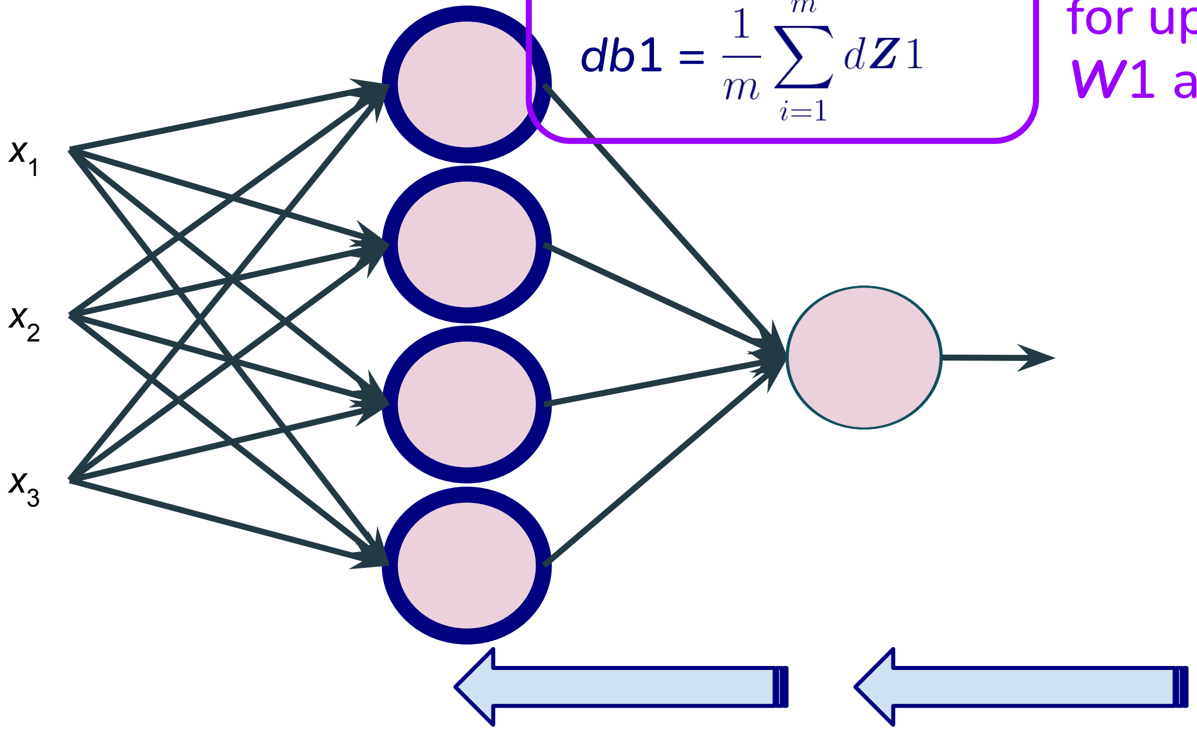
# Backpropagation

$$dZ1 = (dZ2 \cdot W2^T) * (A1 - A1^2)$$

$$dW1 = X^T \cdot dZ1 \,/\, m$$

$$db1 = \frac{1}{m}\sum_{i=1}^{m} dZ1$$

Gradients for updating $W1$ and $b1$



# Gradient Descent

- Initialize parameters (*W*'s and *b*'s)

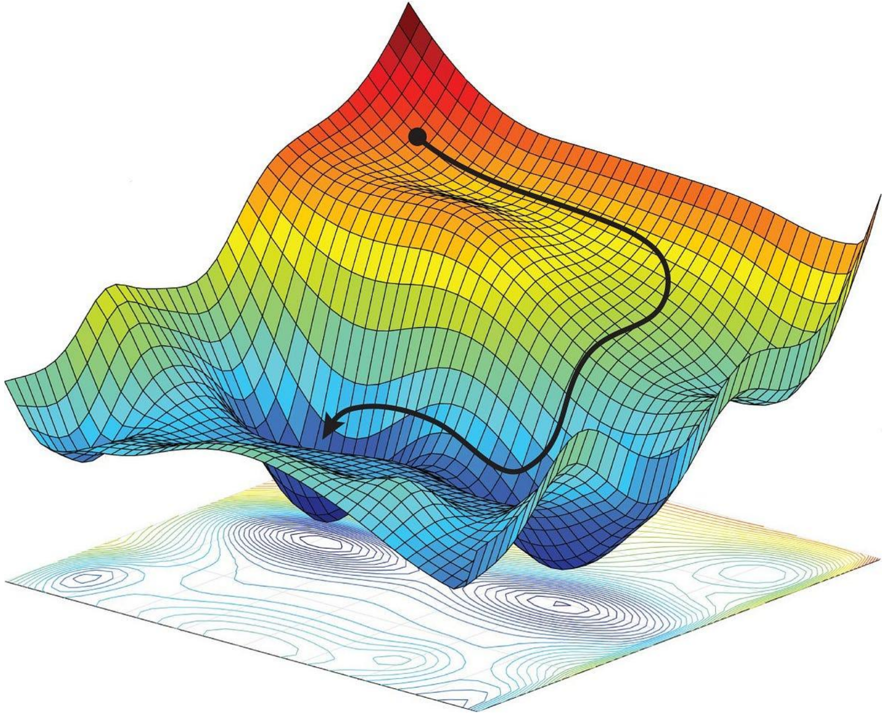
- Repeat until converge:

$$\boldsymbol{W}2 = \boldsymbol{W}2 - \alpha \frac{\partial}{\partial \boldsymbol{W}2} J$$

$$b2 = b2 - \alpha \frac{\partial}{\partial b2} J$$

$$\boldsymbol{W}1 = \boldsymbol{W}1 - \alpha \frac{\partial}{\partial \boldsymbol{W}1} J$$

$$b1 = b1 - \alpha \frac{\partial}{\partial b1} J$$



# Gradient Descent

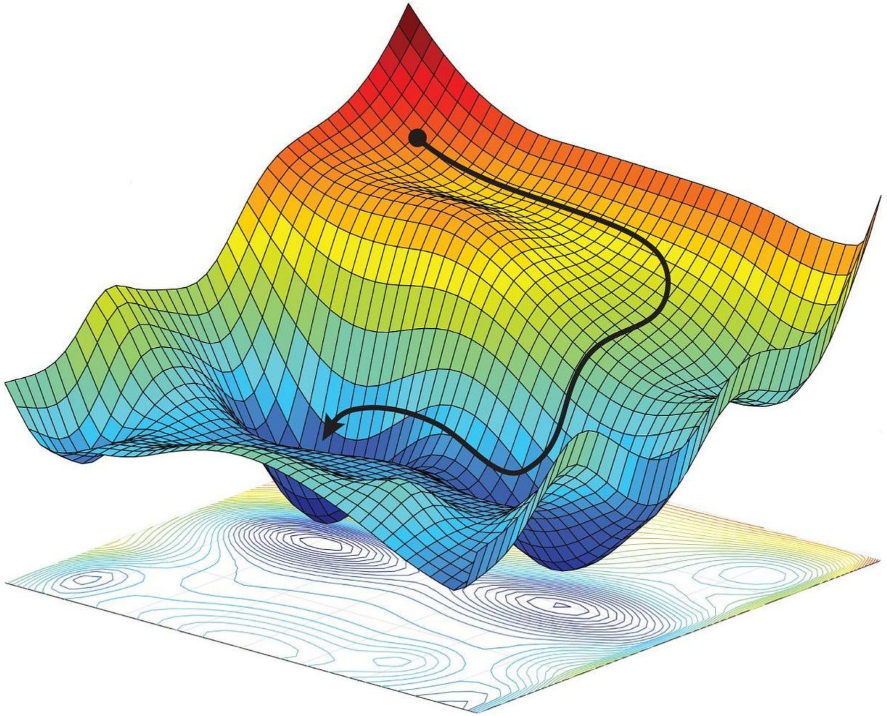
- Initialize parameters (*W*'s and *b*'s)
- Repeat until converge:

$$\boldsymbol{W}2 = \boldsymbol{W}2 - \alpha d\boldsymbol{W}2$$

$$b2 = b2 - \alpha db2$$

$$\boldsymbol{W}1 = \boldsymbol{W}1 - \alpha d\boldsymbol{W}1$$

$$b1 = b1 - \alpha db1$$



# Training (Fitting)

**Training** refers to learning the parameters (*W* and *b*) of the model from the training data*

Gradient Descent Algorithm

- Initialization
- Repeat until convergence:
  - Forward propagation
  - Calculate cost
  - Backpropagation

*Assumes ***X*** refers to training data with *m* rows and *d* columns

*Assumes the number of units in the hidden layer is *units*

# Training

Gradient Descent Algorithm

- ❖ Initialization
- ❖ Repeat until convergence:
  - ➢ Forward propagation
  - ➢ Calculate cost
  - ➢ Backpropagation

Initialize parameters ***W*** and *b*

- ➔ Create (*d*, *units*) array ***W1*** of random numbers
- ➔ Create (1, *units*) array *b*1 of 0's
- ➔ Create (*units*, 1) array ***W***2 of random numbers
- ➔ Create (1, 1) array *b*2 of 0's

# Training

Loop for *max_iter* iterations

Gradient Descent Algorithm

- Initialization
- Repeat until convergence:
  - Forward propagation
  - Calculate cost
  - Backpropagation

# Training

## Gradient Descent Algorithm

- ❖ Initialization
- ❖ Repeat until convergence:
  - ➢ Forward propagation
  - ➢ Calculate cost
  - ➢ Backpropagation

## Compute activations

- ➔ $Z1 = X \cdot W1 + b1$
- ➔ $A1 = g(Z1) = \text{sigmoid}(Z1)$
- ➔ $Z2 = A1 \cdot W2 + b2$
- ➔ $A2 = g(Z2) = \text{sigmoid}(Z2)$

# Training

Gradient Descent Algorithm

- Initialization
- Repeat until convergence:
  - Forward propagation
  - Calculate cost
  - Backpropagation

Cost using current values of ***W*** and *b*

$$\frac{1}{m}\sum_{i=1}^{m} -\boldsymbol{y}\log(\boldsymbol{A}2) - (1-\boldsymbol{y})\log(1-\boldsymbol{A}2)$$

# Training

## Gradient Descent Algorithm

- Initialization
- Repeat until convergence:
  - Forward propagation
  - Calculate cost
  - Backpropagation

**Compute gradients**

→ $dZ2 = A2 - y$

→ $dW2 = A1^T \cdot dZ2 / m$

→ $db2 = \frac{1}{m} \sum_{i=1}^{m} dZ2$

→ $dZ1 = (dZ2 \cdot W2^T) * (A1 - A1^2)$

→ $dW1 = X^T \cdot dZ1 / m$

→ $db1 = \frac{1}{m} \sum_{i=1}^{m} dZ1$

**Update parameters**

→ $W1 = W1 - \alpha \cdot dW1$

→ $b1 = b1 - \alpha \cdot db1$

→ $W2 = W2 - \alpha \cdot dW2$

→ $b2 = b2 - \alpha \cdot db2$

# Testing

***Testing*** refers to evaluating the trained model with testing data*

- Make predictions
- Assess how well predictions correspond to known labels

*Assumes ***X*** refers to testing data

# Testing

- ❖ Make predictions
- ❖ Assess how well predictions correspond to known labels

<u>Predict activations</u>

→ $A1$, $A2$ = Forward propagation of $X$

→ Predictions = Binarize (round) $A2$

# Testing

- Make predictions
- Assess how well predictions correspond to known labels

**Score model**

→ Calculate percentage of predictions that correctly match labels