Neural Network training

Model training steps

	Logistic Regression	TensorFlow + Neural Network

① specify how to compute output given input x and parameters w, b (define model)
 $f_{\vec{w},b}(\vec{x}) = ?$
	$z = np.dot(w, x) + b$
	 $f\_x = 1 / (1 + np.exp(-z))$
	 sigmoid function
		`model = Sequential([`
		 `Dense ...`
		 `Dense ...`
		 `Dense ...`
		 `])`

② Specify loss and cost
 $L(f_{\vec{w},b}(\vec{x}), y)$ 1 example
 $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}), y)$
	logistic loss
	 $loss = -y * np.log(f\_x)$
	 $- (1-y) * np.log(1 - f\_x)$
		binary cross entropy
		 `model.compile(`
		 `loss = BinaryCrossentropy())`

③ Train on data to minimize $J(\vec{w},b)$	$w = w - alpha * dj\_dw$ $b = b - alpha * dj\_db$	`model.fit(X, y, epoch=100)` data → X, target → y

1. Create the Model

define the model

$f(\vec{x}) = ?$


```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units = 25, activation = 'sigmoid'),
    Dense(units = 25, activation = 'sigmoid'),
    Dense(units = 25, activation = 'sigmoid')])
```

2. Loss and cost functions

handwritten digit classification problem → Binary classification

$$L\left(f_{\vec{w},b}(\vec{x}), y\right) = -y \log\left(f_{\vec{w},b}(\vec{x})\right) - (1-y) \log\left(1 - f_{\vec{w},b}(\vec{x})\right)$$

compare prediction vs target

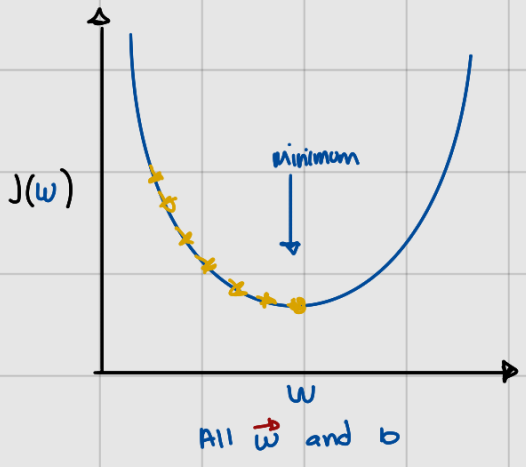
↳ logistic loss / binary cross entropy

```
from tensorflow.keras.layers import BinaryCrossentropy
model.compile(loss = BinaryCrossentropy())
```

Regression (prediction numbers and not categories) → mean squared error

```
from tensorflow.keras.layers import MeanSquaredError
model.compile(loss = MeanSquaredError())
```

3. Gradient Descent



All $\vec{w}$ and b

repeat until convergence {

$$w_j^{[a]} := w_j^{[a]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$$b_j^{[a]} := b_j^{[a]} - \alpha \frac{\partial}{\partial b_j} J(\vec{w}, b) \}$$

Compute derivates for gradient descent using "backpropagation"

```
model.fit(X, y, epoch = 100)
```