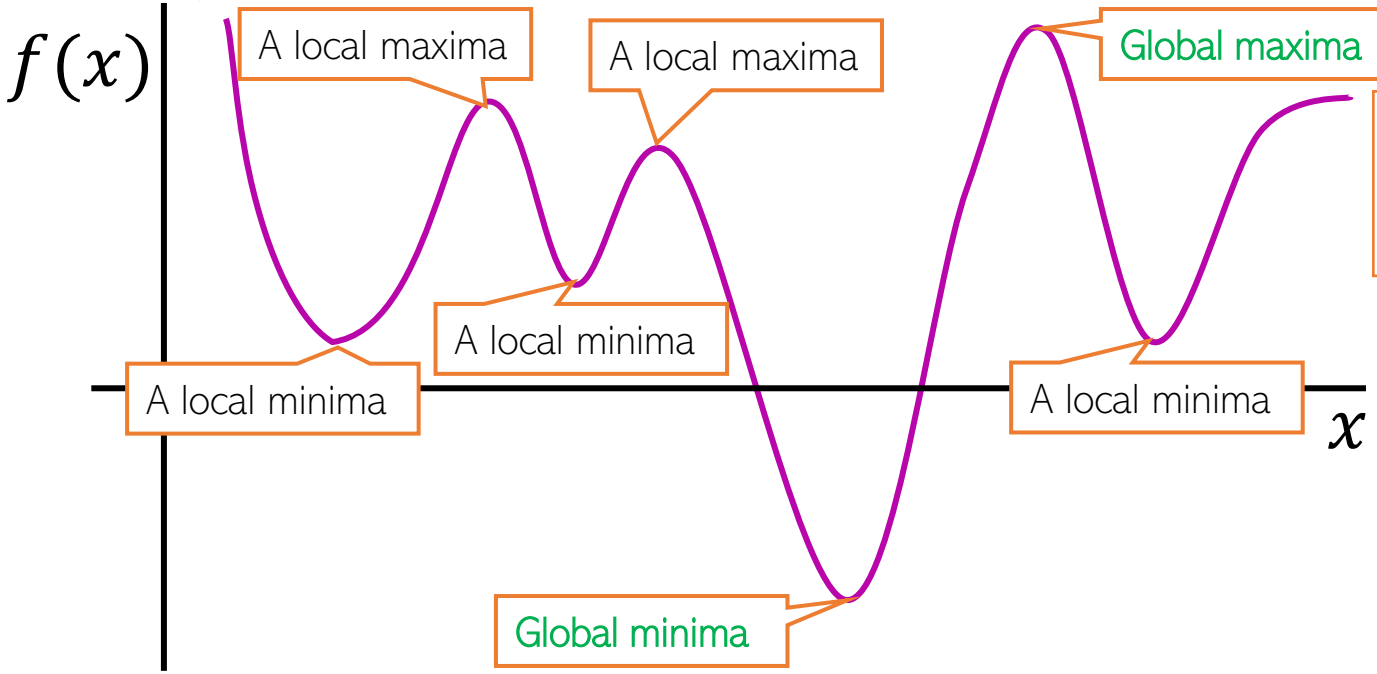
# optimization

# Functions and their optima

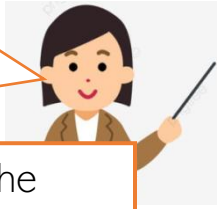
The objective function of the ML problem we are solving (e.g., squared loss for regression)

Assume unconstrained for now, i.e., just a real-valued number/vector

- Many ML problems require us to optimize a function $f$ of some variable(s) $x$
- For simplicity, assume $f$ is a scalar-valued function of a scalar $x$ ($f: \mathbb{R} \rightarrow \mathbb{R}$)




Usually interested in global optima but often want to find local optima, too



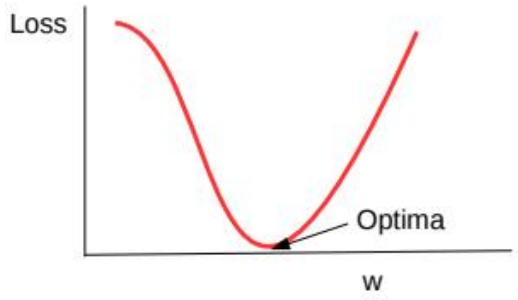
For deep learning models, often the local optima are what we can find (and they usually suffice) – more later

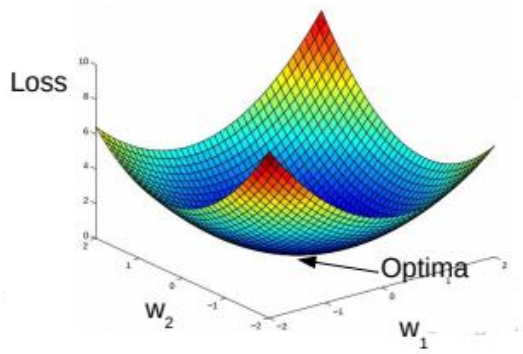
Will see what these are later

- Any function has one/more optima (maxima, minima), and maybe saddle points
- Finding the optima or saddles requires derivatives/gradients of the function

# Convex and Non-Convex Functions

- A function being optimized can be either convex or non-convex
- Here are a couple of examples of convex functions

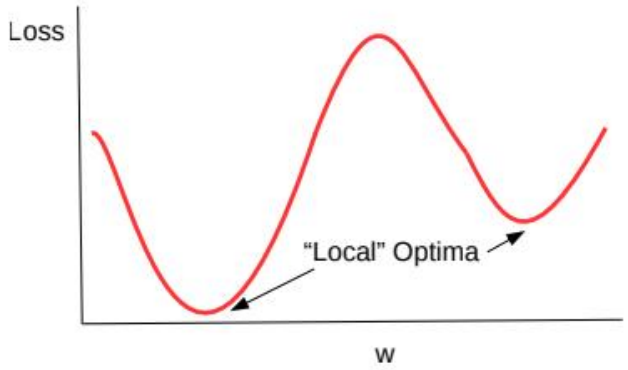


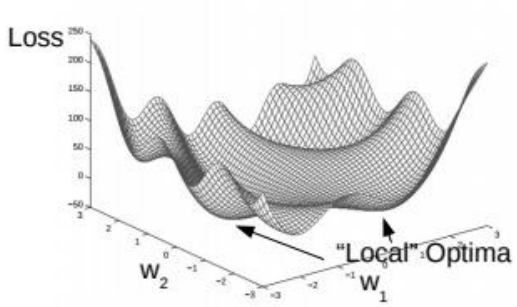




Convex functions are bowl-shaped. They have a unique optima (minima)

Negative of a convex function is called a concave function, which also has a unique optima (maxima)

- Here are a couple of examples of non-convex functions



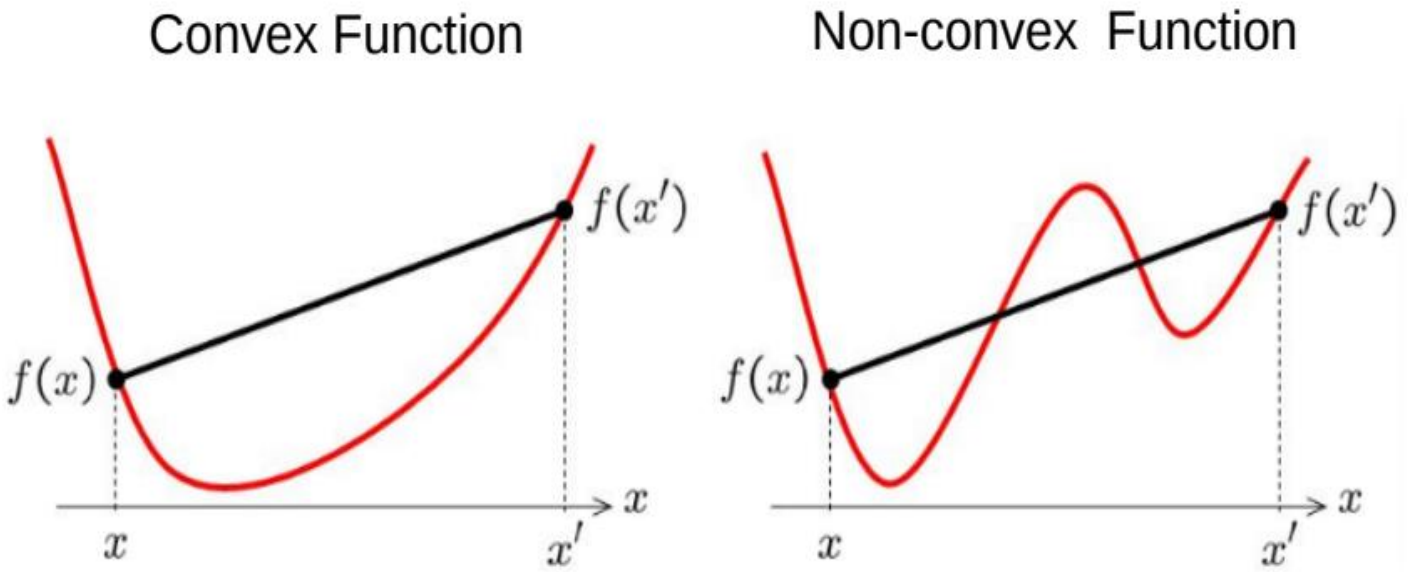




Non-convex functions have multiple minima. Usually harder to optimize as compared to convex functions

Loss functions of most deep learning models are non-convex

# Convex Functions

- Informally, $f(x)$ is convex if all of its chords lie above the function everywhere

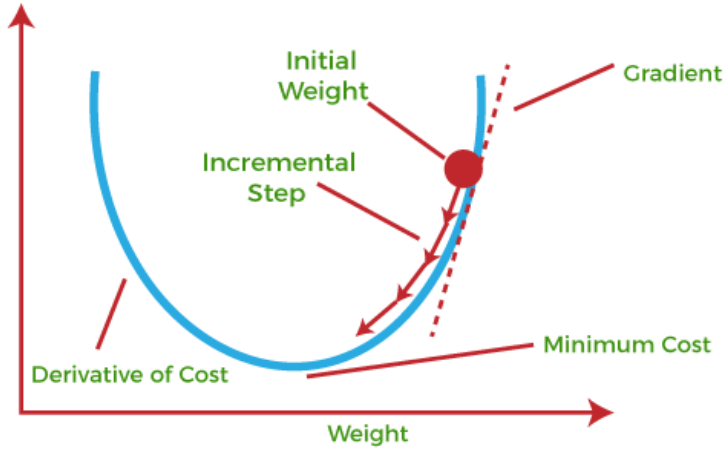


# Gradient Descent

**gradient descent is an iterative optimization algorithm for finding the minimum of an objective function.** In the case of deep learning, the objective function corresponds to the loss function of the network.

**The gradient descent algorithm works in two steps that are performed continuously for a specific number of iterations:**

1. First, we compute the gradient, which is the first-order derivative of the objective function with respect to the variables.
2. Then, we update the variables in the opposite direction of the gradient.




In the case of a neural network, we pass the training data through the hidden layers of the network to compute the value of the loss function and compute the gradients of the loss function with respect to the parameters of the network. Then, we update the parameters accordingly.

# Understanding the partial derivative

If a function $f(x, y)$ depends on multiple variables, its **partial derivatives** tell us how $f$ changes with respect to $x$ while keeping $y$ constant, and vice versa.

### Example:

Let's consider a simple function:

$$f(x, y) = x^2 + 3xy + y^2$$

- **Partial derivative with respect to $x$ ($\frac{\partial f}{\partial x}$):**
  Differentiate with respect to $x$, treating $y$ as a constant:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2 + 3xy + y^2)$$
$$= 2x + 3y + 0$$

- **Partial derivative with respect to $y$ ($\frac{\partial f}{\partial y}$):**
  Differentiate with respect to $y$, treating $x$ as a constant:

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2 + 3xy + y^2)$$
$$= 0 + 3x + 2y$$

Thus, the **gradient** (vector of partial derivatives) is:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2x + 3y, 3x + 2y)$$

# Understanding the partial derivative

Let's compute the **partial derivative of $x$ for the equation:**

$$f(x, y) = (y - x)^2$$

**Step 1: Apply the Power Rule**

Since $(y - x)^2$ is a squared term, we apply the power rule:

$$\frac{\partial}{\partial x}(y - x)^2 = 2(y - x) \cdot \frac{\partial}{\partial x}(y - x)$$

**Step 2: Differentiate the Inner Term $(y - x)$**

Since $y$ is treated as a **constant** with respect to $x$, we differentiate $(y - x)$:

$$\frac{\partial}{\partial x}(y - x) = -1$$

**Step 3: Multiply by the Outer Derivative**

Now, multiplying by $2(y - x)$:

$$\frac{\partial}{\partial x}(y - x)^2 = 2(y - x) \cdot (-1)$$
$$= -2(y - x)$$

**Final Answer**

$$\frac{\partial}{\partial x}(y - x)^2 = -2(y - x)$$

# Gradient Descent

- **Gradient Descent**, used to compute the gradient of a cost function.

For **Mean Squared Error (MSE)**, the cost function is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

where:

- $m$ is the number of data points (tuples),
- $y_i$ is the actual value,
- $\hat{y}_i$ is the predicted value.

## Gradient Descent Update Rule

The gradient for a single variable $\theta$ is:

$$\frac{\partial J}{\partial \theta} = -\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i) x_i$$

And the update step is:

$$\theta := \theta - \alpha \cdot \frac{\partial J}{\partial \theta}$$

where:

- $\alpha$ is the learning rate.