Non convex cost functions have multiple optima, only one of which is the global optimum. The cost functions used in deep learning are mostly non-convex and can be very difficult to converge to the global optimum. There are multiple challenges associated with training a neural network

- Learning rate selection
- Avoiding poor local minimas
- Avoiding drastic changes to the loss surface

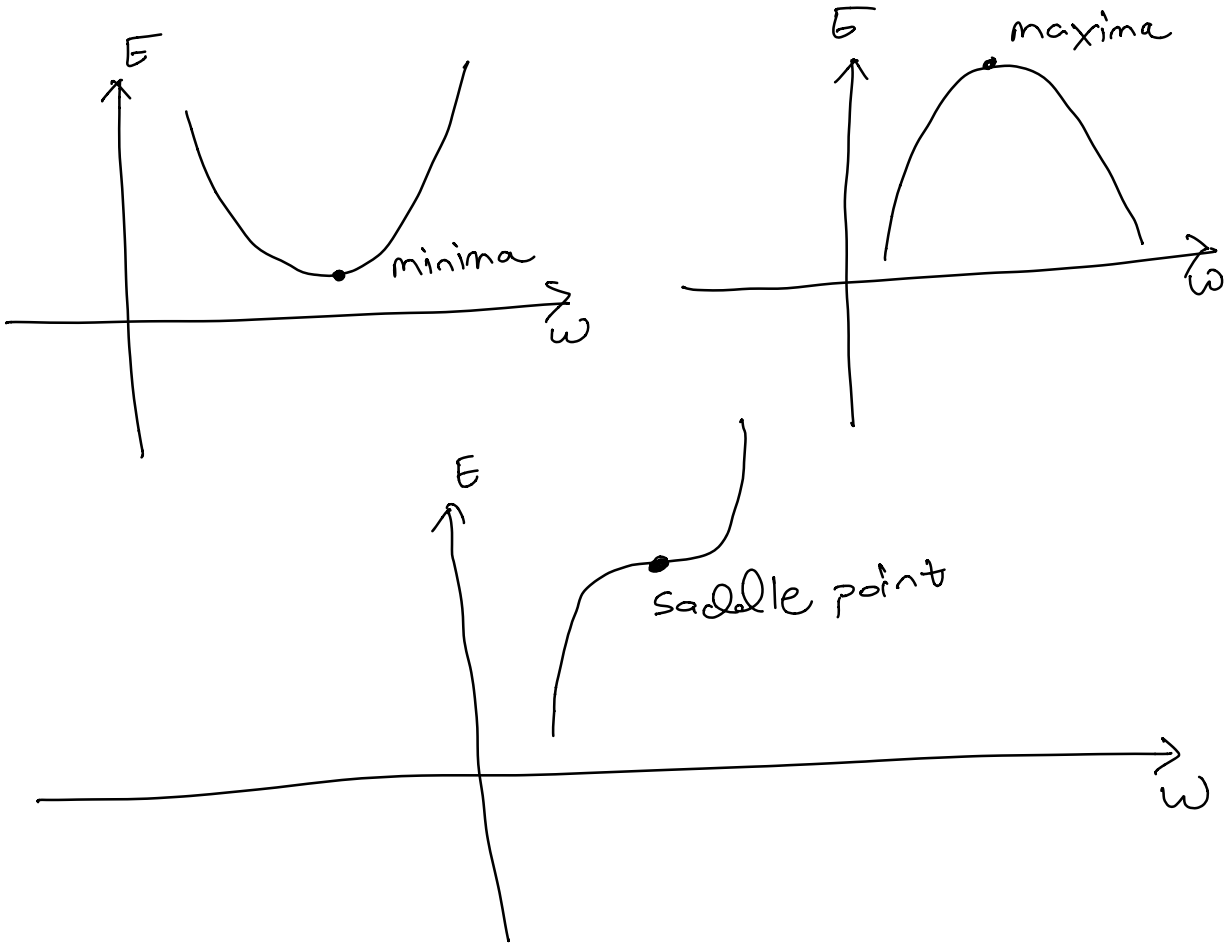
Local minima in deep networks:

Till today it remains an open question whether local minima with a high error rate relative to the global minima are common in practical deep networks.

However, many recent studies seem to indicate that most local minima have error rates and generalization characteristics that are very similar to global minima.

Critical points:

Given an arbitrary function, a point at which the gradient is the zero vector is called a critical point.



for a 1-D cost function, a critical point can take one of the three forms as shown above. Assuming each of these 3 configurations is equally likely then the probability of

a randomly selected critical point being a minima is 1/3. Hence if we have a total of K critical points then on average we will have a total of $\frac{K}{3}$ minimas.

Now let's consider a cost function in a d-dimensional space. In general, in a d-dimensional space we can slice through a critical point on d-different axes. A critical point can only be a local minima if it appears as a local minima in every single one of the

d one-dimensional subspaces. Hence using the result from earlier we have that the probability that a randomly selected critical point in a d-dimensional space is a local minima is $\frac{1}{3^d}$. $\therefore$ As d increases local minima become exponentially more rare.

Vanishing and exploding gradients:

$$h^1 = Wx$$

$$= \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} x$$

$$h^2 = Wh^1$$

$$= WWx = \begin{bmatrix} a^2 & 0 \\ 0 & b \end{bmatrix} x$$

In general,

$$h^n = W^n x$$

$$= \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix} x$$

So, $h_1^n = a^n x_1$

$h_2^n = b^n x_2$

Now, $y = h_1^n + h_2^n$

$\Rightarrow \quad y = a^n x_1 + b^n x_2$

now, $\frac{\partial y}{\partial a} = n a^{n-1} x_1$

$\frac{\partial y}{\partial b} = n b^{n-1} x_2$

So, $\nabla y = \begin{bmatrix} n a^{n-1} x_1 \\ n b^{n-1} x_2 \end{bmatrix}$

now,
Assuming $x_1 = x_2 = 1$ and $a = 1, b = 2$, we have

$y = 1 + 2^n$

$\nabla y = \begin{bmatrix} n \\ n 2^{n-1} \end{bmatrix}$

As $n \to \infty$ then ∇y explodes

Assuming $x_1 = x_2 = 1$ and $a = 0.5$, $b = 0.9$

we have

$$y = (0.5)^n + (0.9)^n$$

$$\nabla y = \begin{bmatrix} n(0.5)^{n-1} \\ n(0.9)^{n-1} \end{bmatrix}$$

As $n \to \infty$ then ∇y vanishes.

Momentum:

In lecture, we observed that loss surfaces with high curvature results in oscillations of the SGD updates. The oscillatory nature leads to a slower convergence of the SGD.

Since SGD will move in the right direction but with oscillations so in order to dampen out the oscillations we will use the average gradient to make the updates.

$$v \leftarrow \alpha v - \epsilon g$$
$$\theta \leftarrow \theta + v$$

v is the velocity and more weight is applied to more recent gradients

creating an exponentially decaying average of gradients.

Adaptive gradients:

A major challenge for training deep networks is appropriately selecting the learning rate. The basic concept behind learning rate adaptation is that the optimal learning rate is appropriately modified over the span of learning to achieve good convergence properties.

Adagrad:

- Adapt global learning rate over time using an accumulation of historical gradients.
- Keep track of a learning rate for each parameter.

$$a \leftarrow a + g \odot g$$

a is the gradient accumulation vector

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{a} + \upsilon} \odot g$$

- Parameters with largest gradients experience a rapid decrease in their learning rates while parameters with smaller gradients only observe a small decrease in their learning rates.

- Since $g \odot g \geq 0$, so a keeps growing and in turn causes the learning rate to shrink and eventually become infinitesimally small.

- Flat regions may force AdaGrad to decrease learning rate before it reaches a minima.

RMS Prop:

- Use a exponentially weighted moving average of gradients

$$a \leftarrow \beta a + (1-\beta)\, g \odot g$$

where β is the decay factor

Smaller the decay factor the shorter the effective window

Adam:

- A variant combination of RMSProp and momentum

- Keep track of an exponentially weighted moving average of the gradient

$$v \leftarrow \beta_1 v + (1-\beta_1) g$$

- Keep track of an exponentially weighted moving average of historical gradients.

$$a \leftarrow \beta_2 a + (1-\beta_2) g \odot g$$