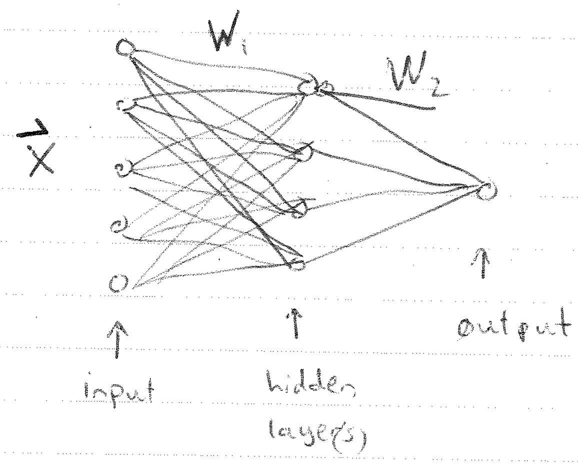
Deep Learning

Mathematical Background - Lecture 2



$$\vec{x} \rightarrow W_1\vec{x} \rightarrow \sigma(W_1\vec{x}) \rightarrow W_2\,\sigma_1(W_1\vec{x}) \rightarrow \sigma_2(W_2\,\sigma_1(W_1 \ldots$$

Or... output is $f(\vec{x})$ [typically an error function]

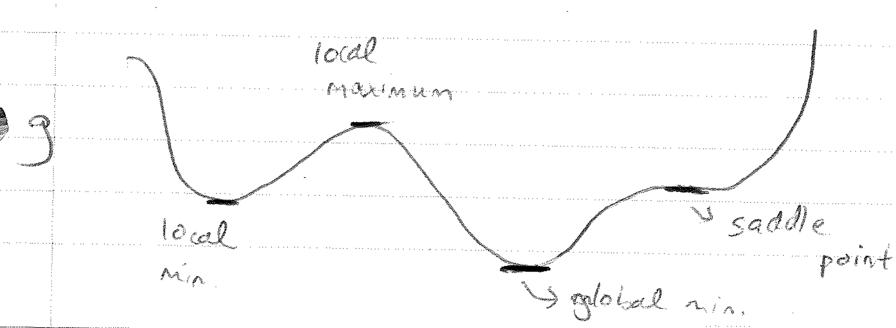
Typically, what is unknown are the weights of the network, and this is what is learned during supervised training.

Abstractly, if w are the parameters, often want to minimize

$$g(w) = \frac{1}{n}\sum_i f(x_i; w)$$

We will study a little about numerical optimization, i.e. finding $\vec{w}$ to minimize $g(\vec{w})$.

How do we find the minimum of g?



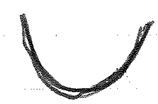
In 1-d: $g'(w) = 0$

In higher-dim, $\nabla g(\vec{w}) = 0$

$$\begin{bmatrix} \frac{\partial g(\vec{w})}{\partial w_1} \\ \vdots \end{bmatrix}$$

In some cases, it turns out to be easy to compute:

- May be simple to compute all points where $\nabla g(\vec{w}) = 0$ (e.g. low-degree polynomial)
- If the function is <u>convex</u>



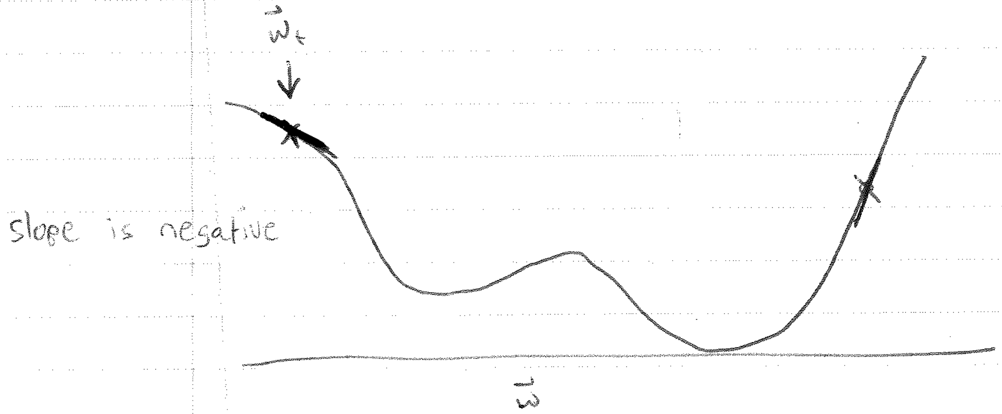
In cases where we will be interested, we cannot easily compute the global minimum. Need a procedure

0^{th}-order : can $\boxed{\text{evaluate}}$ compute $g(\vec{w})$ at a point $\vec{w}$

1^{st}-order : can compute $g(\vec{w})$ and $\nabla g(\vec{w})$

2^{nd}-order : can compute $g(\vec{w})$, $\nabla g(\vec{w})$ and $\nabla^2 g(\vec{w})$

→ most of the focus will be here



$$\vec{w}_{t+1} = \vec{w}_t - \epsilon \cdot \nabla g(\vec{w}_t)$$

ϵ → learning rate

If slope is negative, want to go →

If slope is positive, want to go ←

Called the <u>gradient descent rule</u>.

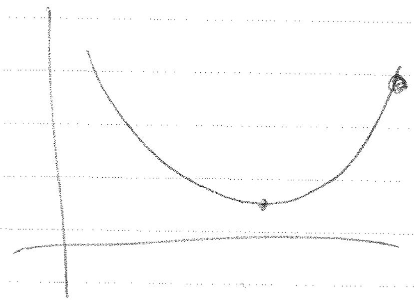
Choosing the learning rate is important

- Steps too large - $\vec{w}_t$ may oscillate
- Steps too small - may not make quick progress

2 General Approaches

- Learning rate schedule that shrinks to 0
- Line search (e.g. Wolfe conditions)

Aside: 2nd-order methods (e.g. Newton's method)



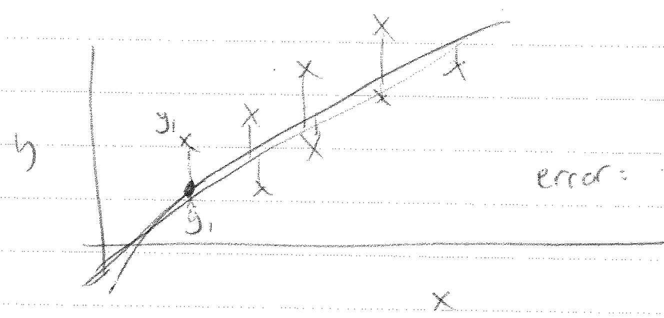
~~$x_{t+1} = x_t - \epsilon f(x_t)$~~

$$\vec{w}_{t+1} = \vec{w}_t - \epsilon \cdot \underbrace{\nabla^2 f(\vec{w}_t)^{-1}}_{\text{may be a massive matrix}} \cdot \nabla f(\vec{w}_t)$$

· Quasi-Newton methods are still used

Machine Learning example

Linear Regression



$\hat{y}_i = w^T x_i + b$

error: $\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = g(\vec{w}, b)$

Minimize error w.r.t. $\vec{w}$ and b

2 comments

- Let's ignore b (can wrap into $\vec{w}$
- Typically solve this simply by computing $\nabla g(\vec{w}) = 0$ and solving for $\vec{w}$.

Let's do gradient descent:

$$\vec{w}_{t+1} = \vec{w}_t - \epsilon \cdot \nabla g(\vec{w}_t)$$

$$\nabla g(\vec{w}_t) = \nabla_{\vec{w}}\left[\frac{1}{2}\sum_{i=1}^{n}(y_i - \vec{w}_t^T \vec{x}_i)^2\right] = \sum_{i=1}^{n}(y_i - \vec{w}^T \vec{x}_i)\vec{x}_i$$

G.D. rule is simply $\vec{w}_{t+1} = \vec{w}_t - \epsilon \cdot \sum_{i=1}^{n} (y_i - \vec{w}^T x_i) \tilde{x}_i$

We may further consider Newton's method on this example

Exercise: Newton's method with a step size of 1 gives you exactly the global minimum, i.e. the least squares solution.

Stepping back a bit - in machine learning, our error functions typically decompose as a sum over the data points, i.e.

$$\text{Err} = \sum_{i=1}^{n} g_i(\vec{x}_i, y_i)$$

~~Alternative to~~ Can be expensive to compute the full gradient (for linear regression - $O(dn)$)
- Instead compute "pieces" of the gradient.

Take a single data point, corresponding to a function g_i, do a G.D. step using only the gradient of g_i:

$$\vec{w}_{t+1} = \vec{w}_t - \epsilon \nabla g_i(\vec{w}_t)$$

In the case of L.R., $\nabla g_i(\vec{w}_t) = (y_i - \vec{w}^T \vec{x}_i) x_i$

$$\vec{w}_{t+1} = \vec{w}_t - \epsilon (y_i - \vec{w}^T \vec{x}_i) x_i$$

Whole algorithm

Initialize $\vec{w}_0$, $t = 0$

repeat

  choose a training pt (x_i, y_i)

  $w_{t+1} = \vec{w}_t - \epsilon (y_i - \vec{w}_t^T \vec{x}_i) \vec{x}_i$

  $t \leftarrow t+1$

end

Intuition - in the time it takes to do a single GD update, you can do many SGD updates (n of them)