Machine Learning (CSE 446): Non-convex optimization; Deep Learning; Tips

Sham M Kakade

University of Washington
cse446-staff@cs.washington.edu

Announcements

- Kevin Jamieson lecture this friday (on structured neural nets).
- EC due Sun

review: multi-layer perceptrons (MLPs)

- (typically) indexes: layer l, nodes: i or j or k
- input activations: given outputs $\{z_j^{(l)}\}$ from layer $l-1$, the *input* activations are:

$$a_j^{(l)} = \sum_{i=1}^{d^{(l-1)}} w_{ji}^{(l)} z_i^{(l-1)}$$

- the *output* activation of each node is:

$$z_j^{(l)} = h(a_j^{(l)})$$

- The target function/output, after we go through L-hidden layers, is then:

$$\widehat{y}(x) = a^{(L+1)} = \sum_{i=1}^{d^{(L)}} w_i^{(L+1)} z_i^{(L)},$$

the 'backprop' algorithm

params $w^{(1)} \dots w^{(L+1)}$ given (x,y)

Computing a gradient on a single datapoint:

- suppose $\ell(y, \widehat{y}(x)) = \frac{1}{2}(y - \widehat{y}(x))^2$.
- Backprop computes $\nabla \ell(y, \widehat{y}(x))$ very efficiently!
- it does this by recursively computing: $\delta_j^{(l)} := \frac{\partial \ell(y,\widehat{y})}{\partial a_j^{(l)}}$ in a 'backwards' pass.

The Forward Pass:

1. Starting with the input x, go forward (from the input to the output layer), compute and store in memory the variables
   $z^{(0)} = x$ $a^{(1)}, z^{(1)}, a^{(2)}, z^{(2)}, \dots a^{(L)}, z^{(L)}, a^{(L+1)}$ $\widehat{y}(x)$

continued...

The Backward Pass:

1. Initialize as follows:

$$\delta^{(L+1)} = -(y - \widehat{y}) = -(y - a^{(L+1)})$$

and compute the derivatives at the output layer:

$$\frac{\partial \ell(y, \widehat{y})}{\partial w_j^{(L+1)}} = -(y - \widehat{y}) z_j^{(L)}$$

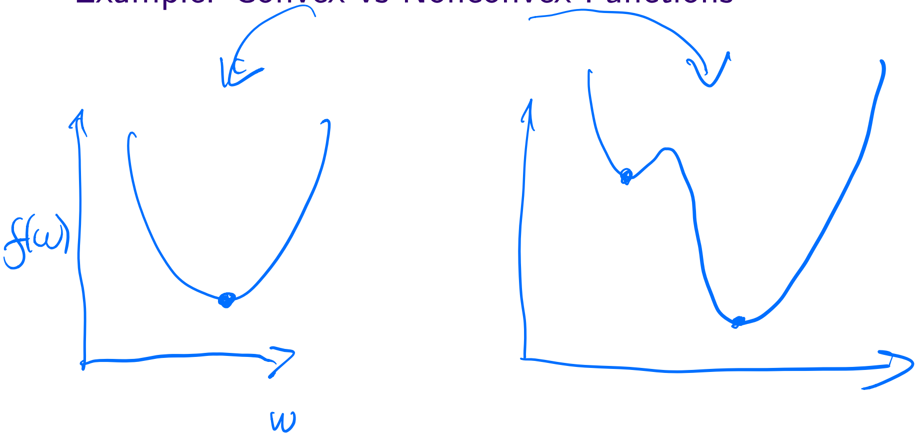
2. **From** $l = L, \ldots 1$

$$\delta_j^{(l)} = \boxed{h'(a_j^{(l)})} \sum_{k=1}^{d^{(l+1)}} w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

then compute the derivatives at layer l:

$$\frac{\partial \ell(y, \widehat{y})}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$$

Example: Convex vs Nonconvex Functions



Gradient descent (or SGD): convexity vs. non-convexity?

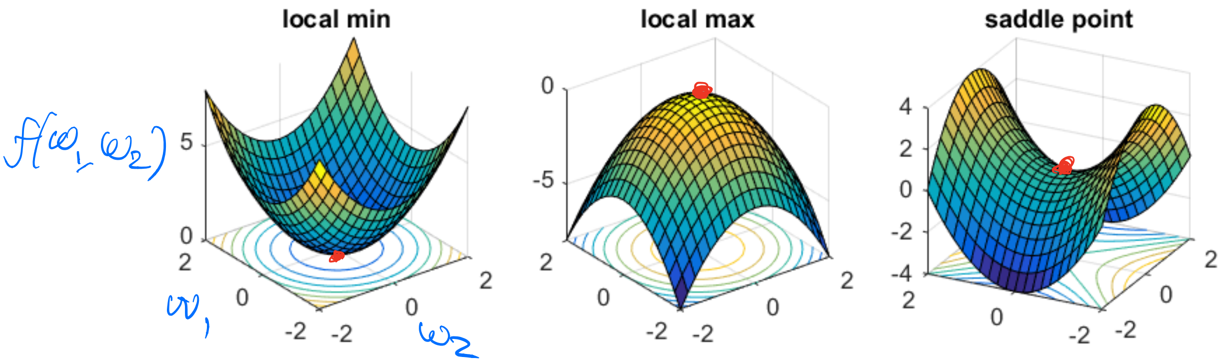
$$w \leftarrow w - \eta \nabla f(w)$$

- Convex problems: gd/sgd will reach the global optima.
- Non-convex problems: we should not (necessarily) expect *any* algorithm to reach the global optima..
- When do we expect GD/SGD to stop? or stop moving 'quickly'?

Terminology: stationary points

- *stationary* (or *critical*) point of $f(w)$: a point which has zero gradient.
- *local minima* of $f(w)$: a point which locally is at a minima (i.e. any infinitesimal change to the point will result in an infinitesimal ~~decrease~~ increase in the function value).
- *global minimum* of $f(w)$: a point w_* which achieves the minimal possible value of $f(w)$ over *all* w.
- (local/global maxima defs are analogous)
- saddle point of $f(w)$: a stationary point that is neither a local maxima or minima.

Stationary points ⟷ 0 gradient



- saddle points could be 'very' flat in some directions.

Fig taken from ``off the convex path'' also see ``escaping saddle points efficiently''.

Things to understand

- Initialization
- Learning rate turning
- saturation/vanishing gradients

Initialization Tips

- convex case: we should start with $w = 0$.
- non-convex case: starting with them all 0 is almost always bad. (often it is a saddle point. why?)
  - too large:
  - too small:
- decay: same heuristics as before

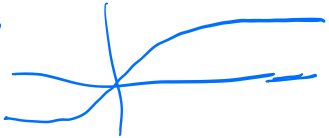
start randomly (from a Gaussian with some variance)

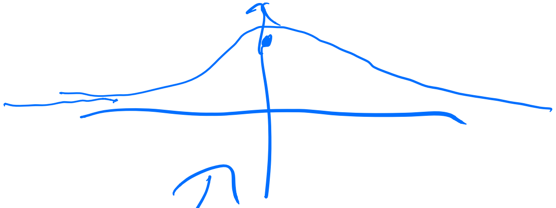
Initialization Tips: ideas based on 'sensitivity'

- we want the starting gradient to not be 0 or "too small". why?
- also, we want the starting gradient not be too large. why?
- instructor: a good starting point is when our *initial loss* is 'slightly' larger than the loss had our weights been all 0's.
- How would we find this setting?

- Also: the "Xavier" initialization.
(more robust version of this idea)

Learning rates: tips





- very similar to before: find a point at which your *loss* decreases quickly (say gives you a large drop in loss after some number of updates).
- be careful about using too a 'large' learning rate: $\tanh(\cdot)$ and $\sigma(\cdot)$ functions *saturate*, meaning that their gradients become small when their inputs are large.
- what does the gradient of $\tanh(\cdot)$ or $\sigma(\cdot)$ look like?

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh'(x) = \frac{\cancel{2}\ 4}{(e^x + e^{-x})^2}$$

Saturation/Vanishing gradients

- (please) do the readings
- in the convex case, a small gradient is good (our loss is nearly optimal).
- 'vanishing gradients': in the non-convex case, for a variety of reasons gradients can become small.
  we could be at a saddle point.
  (please) do the readings

GD/SGD theory: convergence

- you will find a w where $\|\nabla f(w)\|$ is small "quickly" with both GD and SGD.