ECE/CS 559 Lecture 13 T 10/8

Last time: Regularization

- Idea: Avoid overexplaining by limiting to reasonable explanations: prior knowledge / constraints on weights / biases
- $\min_w R(w)$ for $\|w\|_2 \le C$ $\Leftrightarrow$ $\min_w R(w) + \lambda \|w\|_2^2$ (L_2)
- During gradient descent simply add: $\nabla_w(R(w) + \lambda\|w\|_2^2) = \nabla_w R + 2\lambda w$ ($\|w\|_1$: L_1, $\text{sign}(w)$)
- Dropout: Randomly prune weights during training (with prob. p). Include all weights (scaled by p) at test time.

① Optimization

ⓐ Stochastic Gradient Descent

Empirical risk $R(w) = \frac{1}{n}\sum_{(x,y)\in Data} \ell(y, f(x;w))$

"True" risk $R'(w) = \mathbb{E}[\ell(Y, f(X;w))]$ — what we really want to minimize, behavior on future points.

Note: $R(w)$ is an unbiased estimate of $R'(w)$: $\mathbb{E}[R(w)] = R'(w)$

This means the gradients are unbiased: $\mathbb{E}[\nabla_w R(w)] = \nabla_w R'(w)$ (random/stochastic gradients)

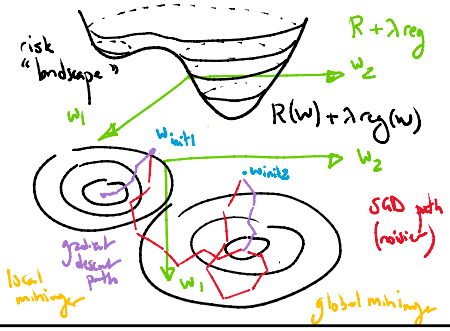
- But... this remains true for even a single point $\mathbb{E}[\nabla_w \ell(Y, f(x;w))] = \nabla_w R(w)$ or a mini batch of points $\mathbb{E}\left[\nabla_w \sum_{(x,y)\in B} \ell(y, f(x;w))\right] = \nabla_w R(w)$
- What changes? The variance. # points ↓ variance ↑ noisiness ↑

Then why don't we always use all the points?

- Reasons for minibatch SGD:
  - Computation: It's faster to get an update with small batches
  - Statistics: Homogeneous data ⇒ small batches give a glimpse of the whole
  - Optimization: The variance can be helpful to get out of local minima.

  (choose batch size to trade off this part with convergence speed.)
- General form of SGD:

```
for epoch in #of epochs:
    t=0 ; Σ∇ = 0
    For (x,y) in Data:
        Forward(x) ; Backward(y)   gives ∇R + λ∇reg
        Σ∇ += ∇R + λ∇reg ; t += 1
        if (t+1) % B == 0:
            W += -η Σ∇
            Σ∇ = 0
```



ⓑ Choice of η, the "learning rate"

- Because SGD introduces noise, η should be adjusted
  - Otherwise, performance can worsen/improve/worsen (oscillate)
- Typically: Reduce η by a factor (e.g. 90%) if sustained worsening.
- Advanced: Adapt to the landscape (e.g. flatness, steepness)
- Vary η by layer: Later layers ∇_w ↑, make η ↓
  by neuron: # Inputs ↑, make η ↓ $\propto 1/\sqrt{\#\text{ inputs}}$ (heuristic)

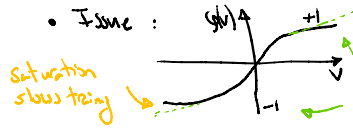
ⓒ Controlling instability:

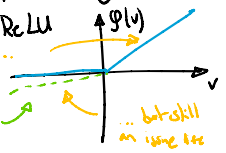
- Gradients are repeatedly multiplied during back propagation.
  - As a result, they could vanish ($\times 0.9 \times 0.9 \times \ldots$) or explode ($\times 1.1 \times 1.1 \times \ldots$)
  - Solution: normalize gradients (across a batch or a layer), clip gradients

② Heuristics

The "craft" of training neural networks.

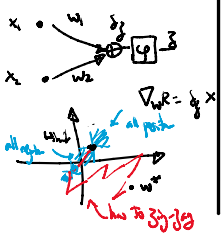
ⓐ Choosing the activation function:

- For the longest time: Sigmoid σ, hyperbolic tangent tanh
  - Issue: 

  - Instead: ReLU 

ⓑ Preprocessing inputs:

- Ideally, input coordinates are uncorrelated/centered
  - Otherwise, may slow down learning
  - Example: if x_i's are always + or − all together
- Preprocess to make them have 0 mean, unit variance, 0 covariance: "whitened"



- How? $\mu = \frac{1}{|Data|}\sum_{x\in Data} x$ (mean (vector)) $C = \frac{1}{|Data|}\sum_{x\in Data}(x-\mu)(x-\mu)^T$ (covariance (matrix))

C is positive semidefinite: can be diagonalized $C = UDU^T$, $UU^T = U^TU = I$

Let $x_{new} = D^{-\frac{1}{2}}U^T(x-\mu)$ ← has 0 mean, covariance = I

ⓒ Initializing the weights:

- Intuition: maintain similar statistics from layer to layer.
- Ignoring activation, assuming whitened inputs, covariance I, next layer's inputs is:

$$C_i = \frac{1}{|Data|}\sum_{x\in Data}(Wx)(Wx)^T = WW^T = n\left[\;\right]n\left[\;\right]$$

Ideally, we want to make $\mathbb{E}[WW^T] = I$

- If w have 0 mean + uncorrelated, then off-diagonals are 0 ✓
- On the diagonal, we'd be adding n, $\mathbb{E}[w^2]$, so we get $n\,\text{var}(w)$.
- To make it identity, we just have to choose $\text{var}(w) = 1$
- E.g., choose w to be i.i.d. $N\left(0, \frac{1}{\sqrt{\#\text{ inputs}}}\right)$