Math/ML Review II

Generalization, gradient descent and SGD

Announcements

- Homework 1 is out, due Sep 25
  - Homework submission is on Gradescope
- My office hours are Mon 10am-11am
- Today: continue review; stochastic gradient descent (SGD), regularization and generalization.

Review: supervised learning

- Supervised learning:
  1. Gather training data $S_{train} = \{(x_1, y_1), \dots, (x_N, y_N)\}$
  2. Choose parametrized model $f: X \times \Theta \to Y$
  3. Train the model: find $\hat{\theta} \in \Theta$ such that $f\left(x, \hat{\theta}\right) \approx y$
  4. Test on a new test set.
- How do you do step 3?
  1. Maximum Likelihood Estimator: use when $f(x, \theta)$ is a parameterized family of probability distributions, maximize the probability of the training data.
  2. Empirical Risk Minimization: decide a "cost" or "loss" function $\ell(x, y, \theta)$, minimize the average training loss.

How to Maximize Likelihood/Minimize Loss?

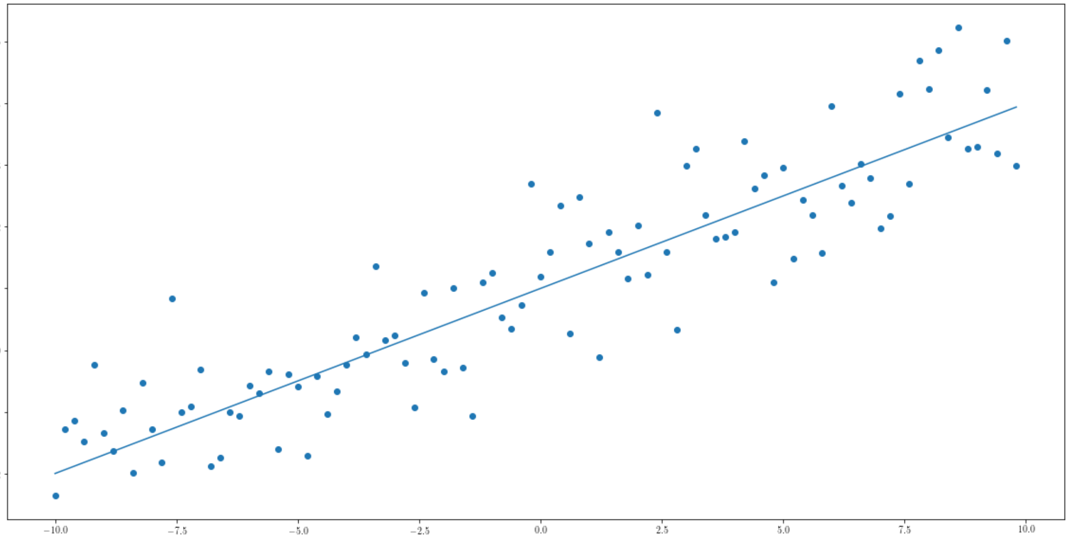
- Linear-regression problem: $y \approx \theta_1 + \theta_2 x$.
- The loss and gradients are:

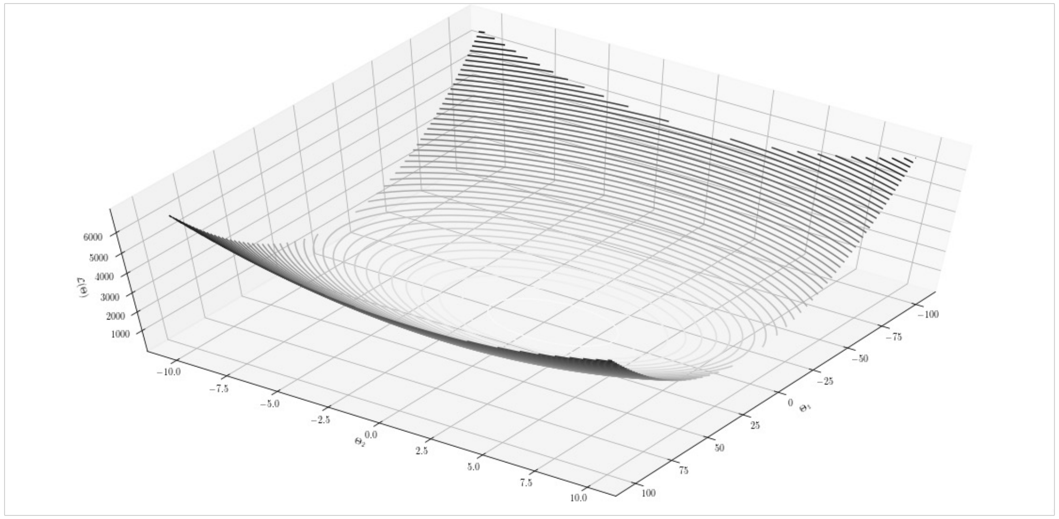
$$L_S(\theta) = \frac{1}{2} \sum_{(x,y)\in S} [y - (\theta_1 + \theta_2 x)]^2$$

$$\frac{dL_S(\theta)}{d\theta_1} = -\sum_{(x,y)\in S} y - (\theta_1 + \theta_2 x)$$

$$\frac{dL_S(\theta)}{d\theta_2} = -\sum_{(x,y)\in S} x[y - (\theta_1 + \theta_2 x)]$$

Data and Loss





Closed Form Solution

$$L_S(\theta) = \frac{1}{2} \sum_{(x,y)\in S} [y - (\theta_1 + \theta_2 x)]^2$$

$$\bar{y} := \frac{\sum_S y}{|S|}, \qquad \bar{x} := \frac{\sum_S x}{|S|}, \qquad \overline{x^2} := \frac{\sum_S x^2}{|S|}$$

$$\hat{\theta}_2 = \left(1 - \frac{(\bar{x})^2}{\overline{x^2}}\right)^{-1} \sum_S x(y - \bar{y})$$

$$\hat{\theta}_1 = \bar{y} - \hat{\theta}_2 \bar{x}$$

Logistic Regression

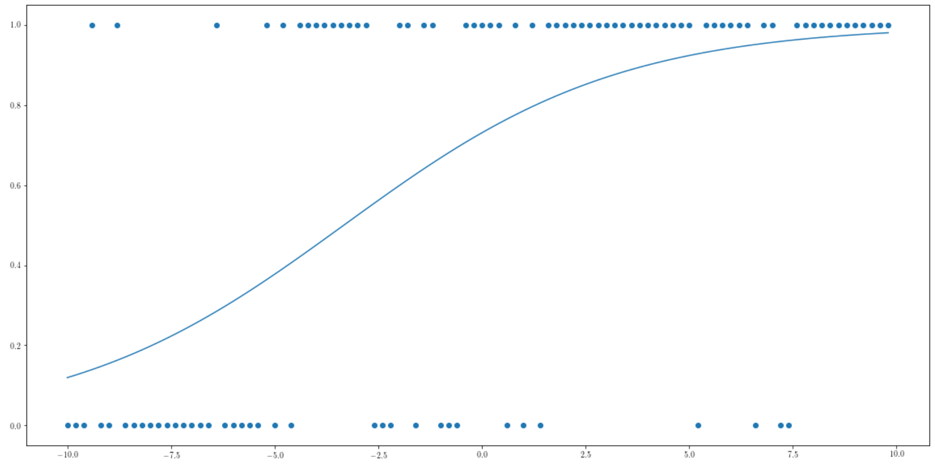
- Binary classification problem: $y = \pm 1$
- Logistic model: $p(y|x;\theta) \propto \exp(y(\theta_1 + \theta_2 x))$.

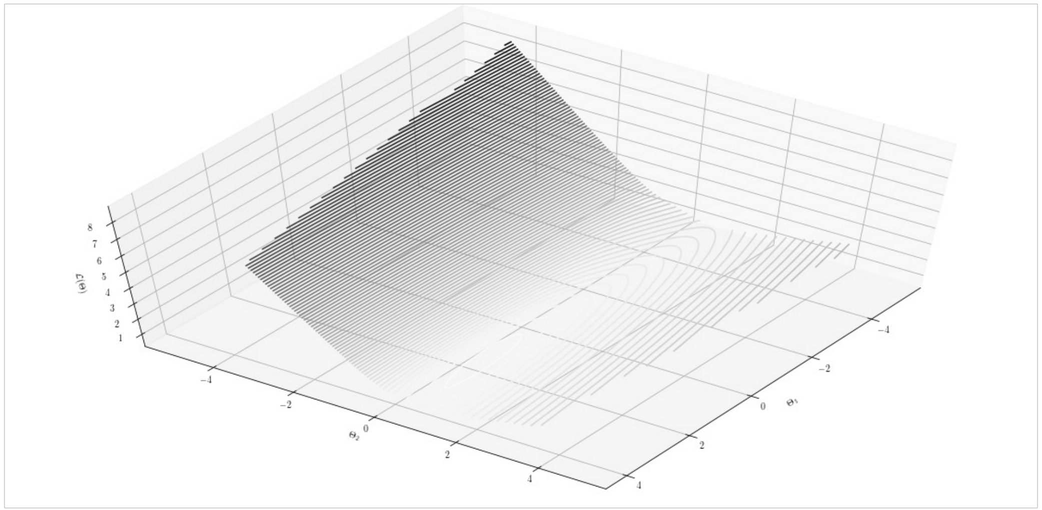
- Logistic loss: $\ell(x, y, \theta) = -\log\big(p(y|x;\theta)\big)$

$$L_S(\theta) = \frac{1}{|S|} \sum_{(x,y)\in S} \ell(x, y, \theta)$$

- There is no longer a closed-form solution for $argmin\, L_S(\theta)$

Data and Loss



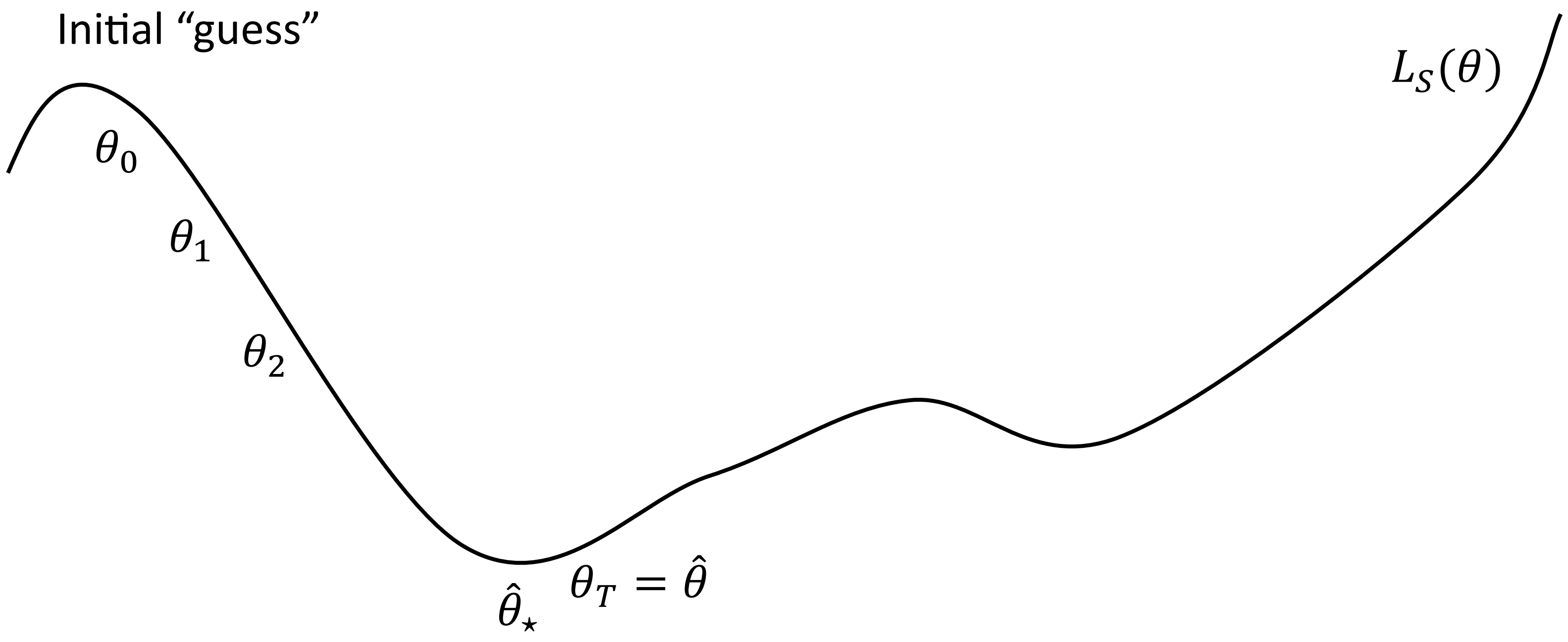


Gradient Descent

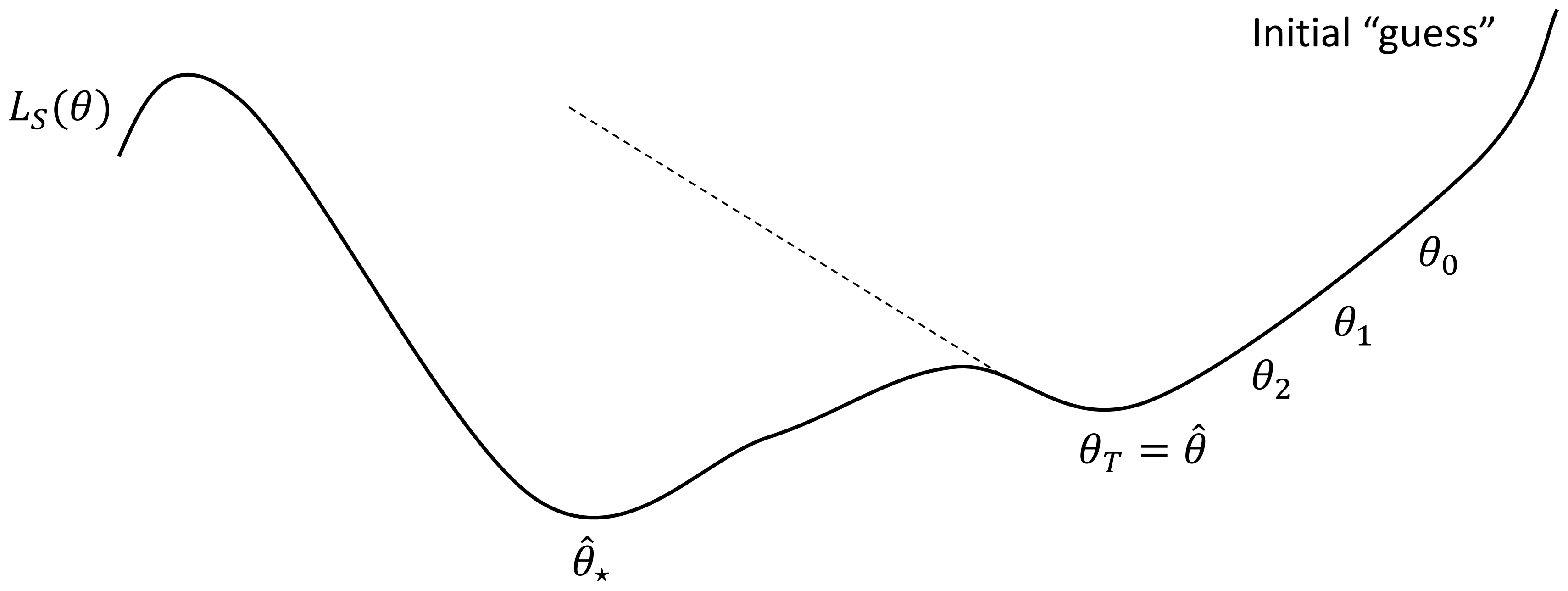
First: Non-stochastic Gradient Descent (just GD)

- “Stochastic” = “random”, GD is a deterministic algorithm.
- GD is much slower than SGD on large datasets, and is rarely used today.
- GD is easier to understand than SGD, so we will do this first.

Gradient Descent: Local-Search



Gradient Descent: Local-Search



First-Order Stationary Points

- Finding $argmin\ L_S(\theta)$ can be very hard.
  - It can be NP-hard in general.
  - In the special case that $L_S(\theta)$ is a ***convex*** function, then the argmin can be found (and GD will do it roughly as fast as possible).
  - Many classical models in machine learning (like SVMs) yield convex losses, but the loss of the deep neural networks is almost always **non-convex**
- We will instead try to find a point with $\nabla L_S(\theta)$ very small.
  - This is *necessary* but not *sufficient* for finding the true argmin.

Key Assumptions for GD analysis

- $L_S(\theta) \geq 0$ for all θ
  - This basically always true
- We can prove that gradient descent works under the assumption that the second derivative of the objective is bounded, i.e.

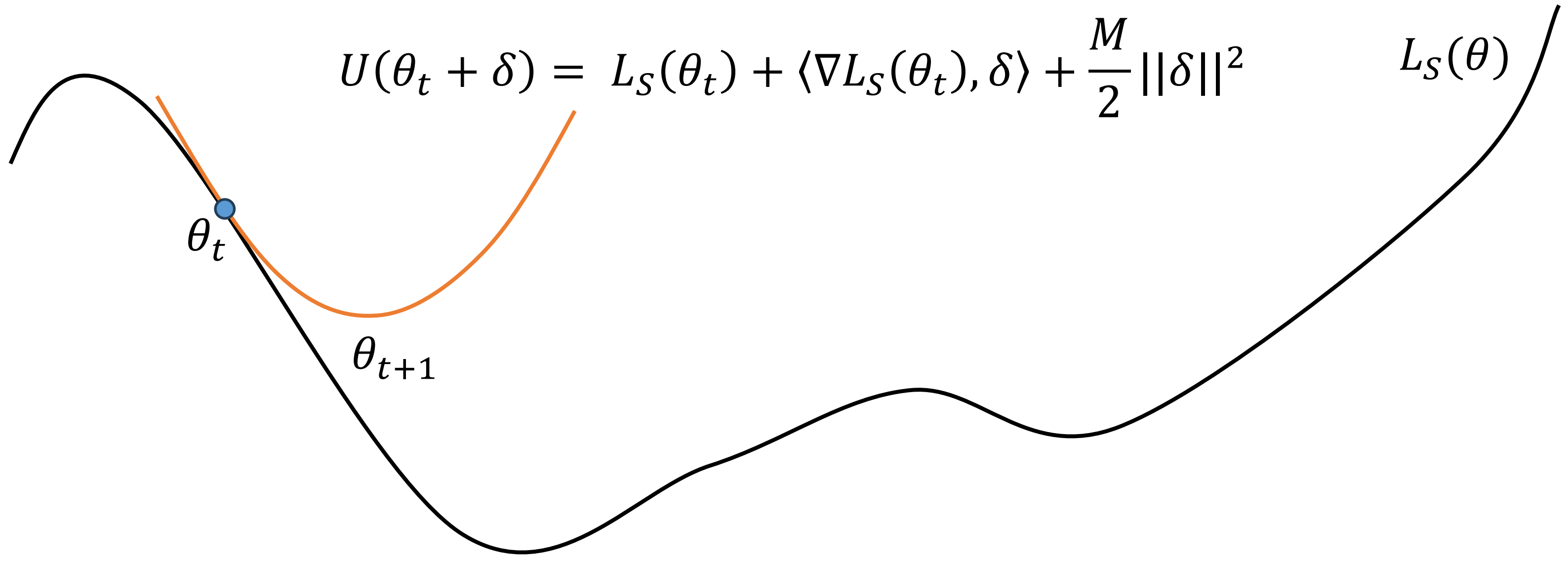
  $||\nabla^2 L_S(\theta)||_{op} \leq M$ for all θ for some M.
  - $\nabla^2 f$ denotes the Hessian (matrix of second partial derivatives) of f
  - The operator norm is also the maximum eigenvalue.
  - M is a measure of "difficulty" of minimizing the loss L_S.
  - This assumption is frequently only true for most θ, or only true for very large values of M for all θ.

Smoothness Parameter M

- The "first-order approximation" to $L_S(\theta)$ at some point θ_t is
$$L_S(\theta_t + \delta) \approx L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle$$
- M measures how large δ can be before this linear approximation becomes very bad: for some $\theta' \in [\theta_t, \theta_t + \delta]$,
$$\begin{aligned} &L_S(\theta_t + \delta) \\ &= L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{1}{2} \delta^\top \nabla^2 L_S(\theta') \delta \\ &= L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + O(M||\delta||^2) \end{aligned}$$

Gradient Descent Algorithm

- Starting from some "guess" θ_t, we will make a "better" guess θ_{t+1}, and then repeat the process with θ_{t+1} etc.


$U(\theta_t + \delta) = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2}||\delta||^2$
$L_S(\theta)$
θ_t
θ_{t+1}

Gradient Descent Algorithm

- $U(\theta_t + \delta) = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2} ||\delta||^2$

$$
\begin{aligned}
& L_S(\theta_t + \delta) \\
& = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{\delta^\top \nabla^2 L_S(\theta')\delta}{2} \\
& \leq L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2} ||\delta||^2 \\
& = U(\theta_t + \delta)
\end{aligned}
$$

If I set M large enough, I can eventually bound $\nabla^2 L_S(\theta_t)$

Gradient Descent Algorithm

- By setting $\theta_{t+1} = argmin\ U(\theta)$, we guarantee $L_S(\theta_{t+1}) \leq L_S(\theta_t)$, because $U(\theta_t) = L_S(\theta_t)$.
- Gradient descent guarantees that $L_S(\theta_t)$ decreases with each iteration.
- What is the actual algorithm? We need to compute $argmin\ U(\theta)$.

Computing the GD update

$$U(\theta_t + \delta) = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2} ||\delta||^2$$

$$\nabla U(\theta_t + \delta) = \nabla L_S(\theta_t) + M\delta$$

$$\delta_\star = -\frac{\nabla L_S(\theta_t)}{M}$$

$$\theta_{t+1} = \theta_t + \delta_\star$$

$$= \theta_t - \frac{1}{\boxed{M}} \nabla L_S(\theta_t)$$

If I set M large enough, I can eventually bound $\nabla^2 L_S(\theta_t)$ and hence guarantee to reduce the function

Key Assumptions for GD analysis

- $L_S(\theta) \geq 0$ for all θ
  - This basically always true
- $||\nabla^2 L_S(\theta)||_{op} \leq M$ for all θ for some M.
  - The operator norm is also the maximum eigenvalue.
  - M is a measure of "difficulty" of minimizing the loss L_S.
  - This assumption is frequently only true for most θ, or only true for very large values of M for all θ.

How fast does GD make progress? (skip)

Remember $\theta_{t+1} = \theta_t + \delta_\star$ with $\delta_\star = -\frac{1}{M}\nabla L_S(\theta_t)$.

$$L_S(\theta_{t+1}) \leq U(\theta_{t+1}) = U(\theta_t + \delta_\star)$$
$$= L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta_\star \rangle + \frac{M}{2}||\delta_\star||^2$$
$$= L_S(\theta_t) - \frac{1}{2M}||\nabla L_S(\theta_t)||^2$$

How fast does GD make progress? (skip)

$$L_S(\theta_{t+1}) \leq L_S(\theta_t) - \frac{1}{2M} ||\nabla L_S(\theta_t)||^2$$

$$L_S(\theta_{t+2}) \leq L_S(\theta_t) - \frac{1}{2M} (||\nabla L_S(\theta_t)||^2 + ||\nabla L_S(\theta_{t+1})||^2)$$

$$L_S(\theta_T) \leq L_S(\theta_0) - \frac{1}{2M} \sum_{t=0}^{T-1} ||\nabla L_S(\theta_t)||^2$$

$$\sum_{t=0}^{T-1} ||\nabla L_S(\theta_t)||^2 \leq 2M\left(L_S(\theta_0) - L_S(\theta_T)\right)$$

How fast does GD make progress? (skip)

$$\sum_{t=0}^{T-1} ||\nabla L_S(\theta_t)||^2 \leq 2M\big(L_S(\theta_0) - L_S(\theta_T)\big)$$

$$\leq 2ML_S(\theta_0)$$

$$\underbrace{\boxed{\frac{1}{T}\sum_{t=0}^{T-1} ||\nabla L_S(\theta_t)||^2}}_{\text{Convergent sum}} \leq \underbrace{\boxed{\frac{\overset{\text{constant}}{2ML_S(\theta_0)}}{T}}}_{\text{Decays to zero}}$$

Since the *average* is at most $O(1/T)$, the minimum is also at most $O(1/T)$:

$$\min_T ||\nabla L_S(\theta_t)|| \leq O\left(\frac{1}{\sqrt{T}}\right)$$

GD summary

- Gradient Descent uses the update:

$$\theta_{t+1} = \theta_t - \frac{1}{M}\nabla L_S(\theta_t)$$

- After T iterations, gradient descent finds a point with

$$||\nabla L_S(\theta_T)|| \leq O\left(\frac{1}{\sqrt{T}}\right)$$

- Each iteration of gradient descent takes time $O(|S|)$.

Stochastic Gradient Descent

Stochastic Gradient Descent: a simple approximate way to find minimizers

- Only requires O(1) memory overhead for large datasets.
- Relatively easy to code up.
- In a variety of settings, SGD is in some sense the *fastest* algorithm for finding the minimum.
- Minimizing the loss can take a long time! SGD provides successively better solutions: you can stop the algorithm at any time and get a reasonable answer.

From deterministic to stochastic

- Key idea for speeding things up:

$$L_S(\theta) = \frac{1}{|S|} \sum_S \ell(x, y, \theta)$$

- Instead of computing a full pass over the dataset each iteration, we will *randomly* choose a datapoint $(x, y) \in S$ to build an approximation to $L_S(\theta)$.

From deterministic to stochastic

- How long does each iteration of GD take?

$$O(|S|)$$

- Solution: add some randomization:
  1. Choose a datapoint $(x, y) \in S$ at random.
  2. Set $\theta_{t+1} = \theta_t - \eta \nabla_\theta \ell(x, y, \theta)$

- What is $E[\theta_{t+1}]$?

$$\begin{aligned} E[\theta_{t+1}] &= E[\theta_t - \eta \nabla_\theta \ell(x, y, \theta)] \\ &= \theta_t - \eta E[\nabla_\theta \ell(x, y, \theta)] \\ &= \theta_t - \eta \nabla L_S(\theta) \end{aligned}$$

Expectation of Gradient

$$E[\nabla_\theta \ell(x,y,\theta)] \approx \frac{1}{|S|}\sum_S \nabla_\theta \ell(x,y,\theta)$$

$$= \nabla_\theta \frac{1}{|S|}\sum_S \ell(x,y,\theta)$$

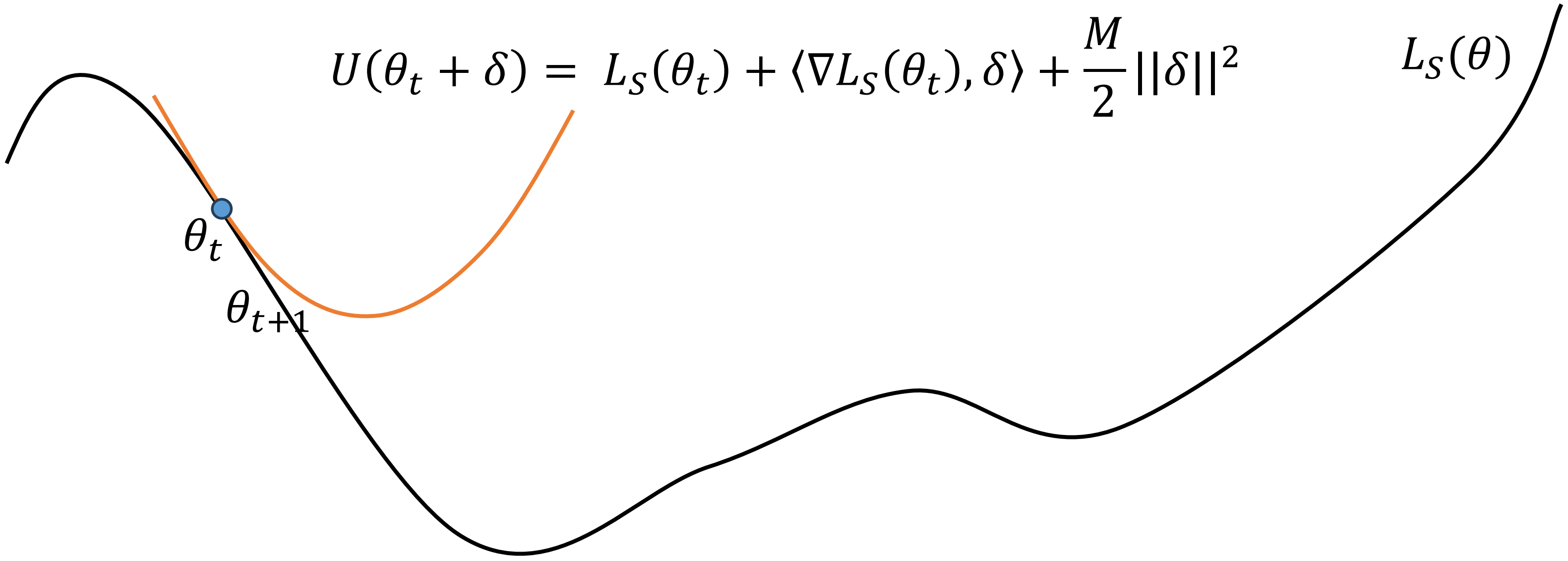
$$= \nabla_\theta L_S(\theta)$$

Key Assumptions for SGD analysis

- $L_S(\theta) \geq 0$ for all θ
  - This basically always true
- $||\nabla^2 L_S(\theta)||_{op} \leq M$ for all θ for some M.
- $||\nabla_\theta \ell(x, y, \theta)|| \leq G$ for all x, y, θ for some G.
  - This is another measure of difficulty: it essentially bounds the amount of "noise" in the problem.

Stochastic Gradient Descent

- The gradients are "noisy", so we trust them less by having a small "learning rate" η



$U(\theta_t + \delta) = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2}||\delta||^2$
$L_S(\theta)$
θ_t
θ_{t+1}

Stochastic Gradient Descent (skip)

$$U(\theta_t + \delta) = L_S(\theta_t) + \langle \nabla L_S(\theta_t), \delta \rangle + \frac{M}{2} ||\delta||^2$$

$$U(\theta_{t+1})$$
$$= L_S(\theta_t) - \eta \langle \nabla L_S(\theta_t), \nabla \ell(x, y, \theta_t) \rangle$$
$$+ \frac{M\eta^2}{2} ||\nabla \ell(x, y, \theta_t)||^2$$

$$E[U(\theta_{t+1})] = L_S(\theta_t) - \eta ||\nabla L_S(\theta_t)||^2 + \frac{M\eta^2 G^2}{2}$$

Progress

Movement Penalty

How fast does SGD make progress? (skip)

$$E[L_S(\theta_{t+1})]$$
$$\leq E[L_S(\theta_t)] - \eta E[||\nabla L_S(\theta_t)||^2] + \frac{M\eta^2 G^2}{2}$$

$$E[L_S(\theta_T)]$$
$$\leq L_S(\theta_0) - \eta \sum_t E[||\nabla L_S(\theta_t)||^2] + \frac{TM\eta^2 G^2}{2}$$

$$\frac{1}{T}\sum_t E[||\nabla L_S(\theta_t)||^2 \leq \frac{L_S(\theta_0)}{\eta T} + \frac{M\eta G^2}{2}$$

How fast does SGD make progress? (skip)

$$\frac{1}{T}\sum_t E[||\nabla L_S(\theta_t)||^2 \leq \frac{L_S(\theta_0)}{\eta T} + \frac{M\eta G^2}{2}$$

Set $\eta = \frac{1}{G\sqrt{T}}$.

$$\frac{1}{T}\sum_t E[||\nabla L_S(\theta_t)||^2 \leq O\left(\frac{G}{\sqrt{T}}\right)$$

SGD summary

- Stochastic Gradient Descent uses the update:

$$\theta_{t+1} = \theta_t + \eta \nabla_\theta \ell(x, y, \theta_t)$$

- We can use $\eta = \frac{1}{G\sqrt{T}}$.
    - Other values are possible! In practice, often guess.
- After T iterations, SGD finds a point with

$$\mathrm{E}[||\nabla L_S(\theta_t)||] \leq O\left(\frac{1}{T^{\frac{1}{4}}}\right)$$

- Each iteration of SGD takes time $O(1)$.

SGD vs GD

- GD: every iteration is guaranteed to make progress on the objective L_S, but it take $O(|S|)$ time to compute a gradient and take one step.
- SGD: Some iterations may actually make negative progress, but it takes only $O(1)$ time to compute a gradient and take one step.
- GD: After T steps, we can find a gradient of size $||\nabla L_S(\theta)||^2 \leq O\left(\frac{1}{T}\right)$
- SGD: After T steps, $E\left[||\nabla L_S(\theta)||^2\right] \leq O\left(\frac{1}{\sqrt{T}}\right)$

SGD vs GD

- GD: $O(|S|)$ time per step, finds a gradient of size $||\nabla L_S(\theta)||^2 \leq O\left(\frac{1}{T}\right)$ after T steps
- SGD: $O(1)$ time per step, finds a gradient of size $E[||\nabla L_S(\theta)||^2] \leq O\left(\frac{1}{\sqrt{T}}\right)$ after T steps
- When is GD more efficient than SGD?
- After N total computation time:
- GD: $O\left(\frac{|S|}{N}\right)$, SGD: $O\left(\frac{1}{\sqrt{N}}\right)$, crossover at $N = |S|^2$

Some implementation notes:

- In practice, the values of M and G are not actually known. The learning rate is usually set by some heuristics.
- Instead of randomly selecting a datapoint $(x, y) \in S$ in SGD, most implementations will shuffle S and then iterate over each datapoint one-by-one in shuffled order. In practice, this seems to produce better results, but the theory is less well understood.

Generalization

Overfitting and Generalization

- For large enough computational budget N, it seems that GD is actually better than SGD. However, this is an illusion caused by *overfitting*.
- We train models by minimizing $L_{S_{train}}(\theta)$, but in reality we care about $L_{S_{test}}(\theta)$.
- These functions can be different if the data is not i.i.d., but they will also be different due to ordinary sampling error.

Overfitting/Generalization as Multiple Hypothesis Testing

- You are trying to predict whether the NASDAQ will rise or fall on each day.
- To do this, you go out and poll 10000 experts on 10 different days, each of which tells you “yes”, or “no”.
- Based on these 10 days of data, you discover that Alice always predicts the correct answer.
- The conclusion (based on ERM), is that we should always do what Alice says.

Overfitting/Generalization as Multiple Hypothesis Testing

- Unfortunately, of the 10000 experts, only one, Barb is actually any good. She is correct with probability 9/10 on all days, and made her expected 1 error on the 10 days of data.
- Every other "expert" just randomly flips a coin and says "yes" if it comes up heads.
- This suggests that Alice should have only 1/1024th chance of getting all 10 days right! Were we just really unlucky to pick her instead of Barb?

Multiple Hypothesis Testing

- Although Alice has individually 1/1024 odds of being right every time, the odds that *someone* other than Barb will be perfectly correct on all 10 days is actually very high:

$$P[someone\ is\ perfect] = 1 - \left(\frac{1023}{1024}\right)^{9999} > 0.99$$

- This is a failure of ERM: the testing performance turned out to be very different than the training performance.

Ways to avoid the problem

- How can we make it less likely to choose someone other than Barb?

1. Gather more training data (i.e. more than 10 days)
2. Remove some of the experts at random?!?

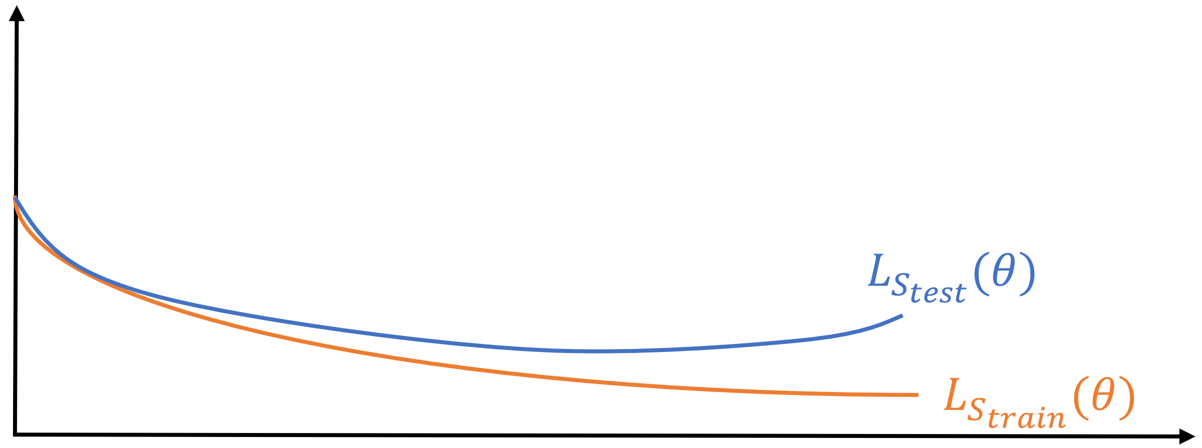
- Say we throw out $\frac{9}{10}$ of experts.
- Probability of not picking Barb is about:

$$1 - \frac{1}{10}\left(\frac{1023}{1024}\right)^{1000} = 0.96$$

Overfitting in Machine Learning

- Each parameter value $\theta \in \Theta$ is like an “expert”.
- If we check “too many” different values of θ, or if $|S_{train}|$ is too small, then some bad value of θ might look really good on the test set.

Overfitting in Machine Learning



Iterations of SGD

Number of θ values tested

How much data do you need?

- You want to estimate the probability that a biased coin will land heads. If you observe N coin flips, how accurate can you be?
- $F_i = 1$ if ith flip is heads, and 0 otherwise.
- $A = \frac{1}{N}(F_1 + \cdots + F_N)$
- $Var(F_i) \leq 1$
- $|A - E[F]| \leq \frac{10}{\sqrt{N}}$ with probability at least 0.99 (you prove something like this in the homework!)
- In general, with N datapoints, you often *cannot* estimate quantities to better than $O\left(\frac{1}{\sqrt{N}}\right)$ error.

Two Myths about overfitting

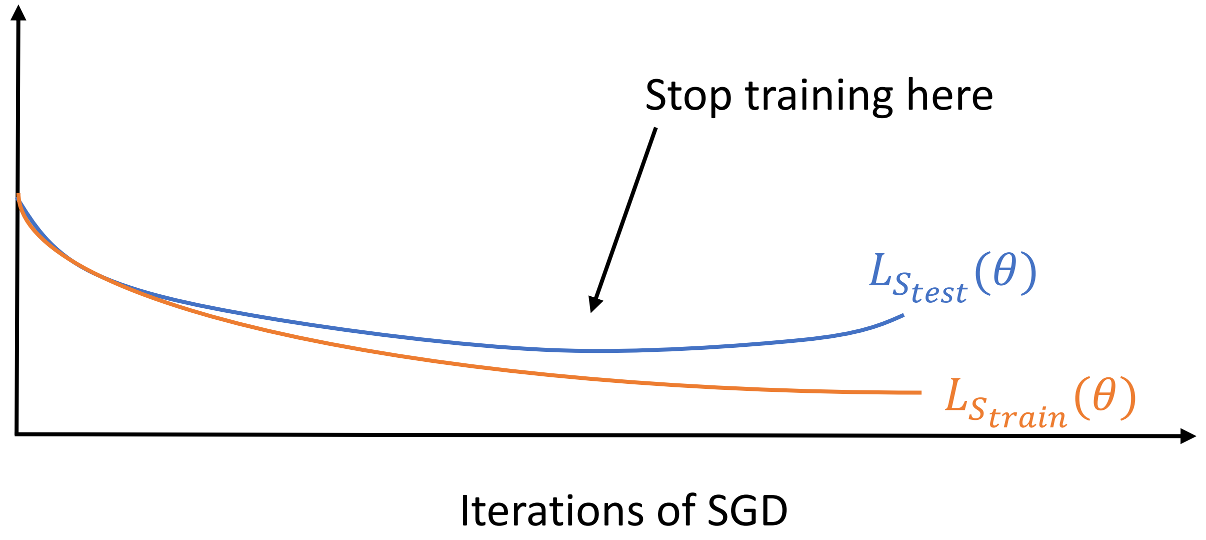
- The complexity term C is equal to the dimension of Θ.
  - C is often a measure of *size* of Θ, but there are much more sophisticated ways to measure size that work much better.
  - Many take the form $C = \max ||\theta||$ for some particular norm $|| \cdot ||$ other than the standard one.
  - Finding the "right" way to measure C is a subject of very active research.
- Deep neural networks never overfit.
  - They often overfit much less than other models, but they do sometimes overfit in practice. You should always check for this.

Combatting Overfitting

- Gather more data!
- All the other methods are essentially more sophisticated versions of “throw out some of the experts”.
  - Even though this sounds bad, they can work quite well.

Combatting Overfitting

- Early stopping



Combatting Overfitting

- Regularization: penalize some values of θ
- Instead of minimizing the empirical risk directly, minimize a penalized empirical risk:
- $L = L_S(\theta) + r(\theta)$
- $r(\theta)$ is called a *regularizer*. One common regularizer is *weight decay*:
$$r(\theta) = \lambda||\theta||^2$$
- Here λ is a *hyperparameter* chosen by the intrepid user (i.e. you!)
- Weight decay makes it less likely to pick large θ: "throw out all large θ unless they have really good training error."