# STOCHASTIC GRADIENT DESCENT, ENTROPY-SGD

PRATIK CHAUDHARI
UNIVERSITY OF PENNSYLVANIA
NOVEMBER 1, 2019

## Reading

- "Stochastic gradient descent tricks" by Bottou (2012). Great paper with lots of little tricks of how to use SGD in practice.
- Till Section 4.2 of "Optimization methods for large-scale machine learning" by Bottou et al. (2018).
- "Entropy-SGD: Biasing gradient descent into wide valleys" by Chaudhari et al. (2016)

---

## 1. Stochastic gradient descent

SGD has its roots in stochastic optimization (Robbins and Monro, 1951). A stochastic optimization problem looks like

$$x^* = \operatorname*{argmin}_x \mathop{\mathbb{E}}_{\xi} \left[ f(x;\xi) \right]$$

where $\xi$ is a random variable. This is a very old and rich area, there was lots of action in it already in the 1950s, e.g., (Kushner and Yin, 2003, Robbins and Monro, 1951). It is also a highly relevant problem: for instance, when a plane goes from Los Angeles to Philadelphia, the route that the plane takes depends on the local weather conditions along its path and airlines will optimize this route using a stochastic optimization problem of the above form. The variable $x$ will be the trajectory of the plane and $\xi$ are the weather conditions which we do not know exactly but may perhaps have estimated a distribution for them.

In machine learning today, we solve a slightly different problem, namely the finite-sum problem. Given a finite dataset $D = \{(\xi_i, y_i)\}_{i=1,\dots,n}$ we minimize

$$f(x) := \frac{1}{n} \sum_{i=1}^{n} \ell(x; \xi_i, y_i).$$

The variable $x$ denotes the weights/parameters of the model and $\xi_i$ denotes the $i^{\text{th}}$ datum. As we have discussed before, the number of data $n$ can be very large in modern machine learning (why?). It is difficult to do gradient descent in this case because the gradient

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(x; \xi_i, y_i).$$

involves a sum of the entire dataset. Note that the convergence rate of optimization algorithms we have look at is independent of the dimension; this is true for most gradient-based optimization algorithms.

Stochastic gradient descent for the finite-sum case performs the following iterations

$$x^{(t+1)} = x^{(t)} - \alpha \nabla \ell(x^{(t)}; \xi_{\omega_t}, y_{\omega_t}) \tag{1}$$

The datum $(x_{\omega_t}, y_{\omega_t})$ over which we compute the gradient before updating the weights is picked randomly from the dataset $D$. So each iteration of SGD is a factor of $n$ times faster than that of gradient descent. Is this the direction of steepest descent though?

**Example 1.** Play with the step-size at
http://fa.bianp.net/teaching/2018/eecs227at/gradient_descent.html.

**Example 2.** Let us take a quadratic loss function with scalar data $\{(a_i, b_i)\}$.

$$f(x) = \frac{1}{2n} \sum_{i=1}^{n} (a_i x - b_i)^2 .$$

Discuss the solution of this problem and the fact that SGD will find solutions that try to fit each datum at each iteration but bounce around.

- Very fast speed outside the region of confusion.
- Once you get close to the optimum, the gradients keep fighting at successive iterations.

**Remark 3.** Sampling with and without replacement.

**Remark 4** (Mini-batch version of SGD)**.**

$$x^{(t+1)} = x^{(t)} - \frac{\eta}{\mathscr{b}} \sum_{k=1}^{\mathscr{b}} \nabla \ell(x^{(t)}; \xi_{\omega_t^k}, y_{\omega_t^k}). \tag{2}$$

Samples in the mini-batch $\left\{(\xi_{\omega_t^k}, y_{\omega_t^k})\right\}_{k=1,\ldots,\mathscr{b}}$ are picked randomly from the dataset at each iteration. Note that if $\mathscr{b} = n$, this is exactly gradient descent.

Let us denote the mini-batch gradient with a batch-size of size $\mathscr{b}$ to be

$$\nabla f_{\mathscr{b}}(x^{(t)}) := \frac{1}{\mathscr{b}} \sum_{k=1}^{\mathscr{b}} \nabla \ell(x^{(t)}; \xi_{\omega_t^k}, y_{\omega_t^k}).$$

This will help us write SGD in short-form as

$$x^{(t+1)} = x^{(t)} - \eta \nabla f_{\mathscr{b}}(x^{(t)}).$$

This looks very similar to gradient descent, except that there is a $\mathcal{b}$ at the subscript. To make our notation in the following a bit clearer, we will also use SGD with a batch-size of $\mathcal{b} = 1$ but denote it as

$$\nabla f_\omega(x^{(t)}) := \nabla \ell(x^{(t)}; \xi_{\omega_t}, y_{\omega_t}).$$

In other words $\nabla f_\omega$ is simply the stochastic gradient $\nabla f_{\mathcal{b}}$ with $\mathcal{b} = 1$.

## 2. Convergence rate for SGD

We will only consider strongly convex functions for analyzing SGD. The proofs are much more tedious for the others. Remember that gradient descent requires $\mathcal{O}(n)$ operations each iteration. For strongly convex functions, gradient descent converges at a rate

$$f(x^{(t)}) - f(x^*) \leq \mathcal{O}(c^t).$$

The number of steps required to reach an error $\epsilon$ is $\mathcal{O}(\log(1/\epsilon))$. Each step needs $\mathcal{O}(n)$ operations and therefore the total computation used by GD is

$$\mathcal{O}(n \log(1/\epsilon)).$$

The updates of SGD are stochastic so

$$f(x^{(t)}) - f(x^*)$$

is a random variable. It depends on the initial condition and the random data that were chosen in the first $t$ iterations to compute the gradients. How should we understand the convergence of SGD then? SGD clearly does not "converge" in the zone of confusion in the sense of iterates stopping to move. Every time we pick a new example or a new mini-batch, you get some gradient and you move. There are many notions of convergence for a sequence of random variables $X_1, X_2, \dots, X_n$, for SGD, we will use convergence in the first moment

$$\mathbb{E}\left[|X_n - X^*|\right] \to 0.$$

As ususal let's use $L$-smooth and $m$-strongly convex function $f(x)$ (we do not need strong convexity for the following two lemmas but will use it later). Note this is $f(x)$, not $\ell(x, \xi, y)$. We can get a descent-lemma style result now.

**Lemma 5** (Descent lemma for stochastic updates)**.** *The next update for SGD satisfies*

$$\mathop{\mathbb{E}}_{\omega_t}\left[f(x^{(t+1)})\right] - f(x^{(t)}) \leq -\alpha \nabla f(x^{(t)})^\top \mathop{\mathbb{E}}_{\omega_t}\left[\nabla f_{\omega_t}(x^{(t)})\right] + \frac{L\alpha^2}{2} \mathop{\mathbb{E}}_{\omega_t}\left[\|\nabla f_{\omega_t}(x^{(t)})\|^2\right].$$

Compare this with the descent lemma for gradient descent

$$f(x^{(t+1)}) \leq f(x^{(t)}) - \alpha \nabla f(x^{(t)})^\top \nabla f(x^{(t)}) + \frac{L\alpha^2}{2}\|\nabla f(x^{(t)})\|^2.$$

**Proof**. Use the descent lemma for GD, substitute the iterates of SGD and take an expectation on both sides over the index of the datum $\omega_t$. ■

**Typical assumptions in SGD analysis.** Based on the previous lemma, we can construct a set of assumptions. Remember that if we can bound the right hand side of the above inequality using some deterministic quantity, we should be good to prove the convergence of SGD and obtain a rate. This is similar to what we did for gradient descent.

- Assume that the stochastic gradient is unbiased
$$\nabla f(x) = \mathop{\mathbb{E}}_{\omega} [\nabla f_\omega(x)]$$
for all $x$ in the domain. This is akin to assuming that the way we sample images in the mini-batch is such that the average is always pointing towards the true gradient with a similar magnitude. This is a natural condition and you will change it only if you are doing tricks with the sampling distribution, e.g., boosting.
- The second condition is important. There exist scalars $\sigma_0$ and $\sigma$ such that
$$\mathop{\mathbb{E}}_{\omega_t} \left[ \|\nabla f_\omega(x)\|^2 \right] \leq \sigma_0 + \sigma \|\nabla f(x)\|^2.$$
This condition assumes something about the second term in the descent lemma for SGD. It assumes that the stochastic gradient is not too bad: near a critical point (locations where $\nabla f(x) = 0$), it is allowed to grow in a similar fashion as the true gradient except with a scaling factor $\sigma > 0$ and a constant $\sigma_0$.

**Lemma 6.** *Under the above assumptions on the loss function in SGD, we have*
$$\begin{aligned}\mathop{\mathbb{E}}_{\omega_t} \left[ f(x^{(t+1)}) \right] - f(x^{(t)}) &\leq -\alpha \|\nabla f(x^{(t)})\|^2 + \frac{L\alpha^2}{2} \mathop{\mathbb{E}}_{\omega_t} \left[ \|\nabla f_{\omega_t}\|^2 \right] \\ &\leq - \left(1 - \frac{L\alpha\sigma}{2}\right) \alpha \|\nabla f(x^{(t)})\|^2 + \frac{\alpha^2 L \sigma_0}{2}.\end{aligned}$$

To prove this simply substitute the assumptions into Lemma 5. Compare this to the corresponding result we derived for gradient descent in Lecture 9
$$f(x^{(t+1)}) - f(x^{(t)}) \leq -\frac{\alpha}{2} \|\nabla f(x^{(t)})\|^2.$$
For strongly convex functions, you can pick a larger step-size in gradient descent $\alpha < 2/L$ and obtain a similar expression as Lemma 6.

Notice that the full objective at the next step depends only on the index of the datum we picked at this iteration and the step-size $\alpha$. It does not depend on any of the past iterates. We have essentially achieved our goal. We upper bounded the left hand side using a deterministic quantity. Notice that for small $\alpha$, the first term is strictly negative. However the second term might be quite large if $\sigma_0$ is large. Picking the step-size $\alpha$ in such a way that it balances of these two terms in SGD is critical to get good performance in practice.

**Theorem 7** (Convergence rate of SGD for smooth, strongly-convex functions)**.** *If we pick a step-size*
$$\alpha \leq \frac{1}{L\sigma}$$

*then the expected optimality gap satisfies*

$$\mathop{\mathbb{E}}_{\omega_1,\omega_2,\ldots,\omega_t}\left[f(x^{(t+1)})\right] - f(x^*) \leq \frac{\alpha L \sigma_0}{2m} + (1-\alpha m)^t \left(f(x^0) - f(x^*) - \frac{\alpha L \sigma_0}{2m}\right).$$

**Proof**. The proof follows by a direct application of Lemma 6, see Bottou et al. (2018) Theorem 4.6 for the proof. ■

This theorem beautifully demonstrates the interplay between the step-size and and the variance of stochastic gradients. If $\sigma, \sigma_0 = 0$, we get the same result as that of gradient descent, namely, the function value $f(x^{(t+1)})$ converges at a linear rate $(1-\alpha m)^t$. Some points to notice

- When gradient computation is noisy, we have a non-zero $\sigma_0$ we can no longer get to the global minimum, there is a first term which does not decay with time.
- If we pick a small $\alpha$, we get closer to the global minimum but go there quite slowly. On the other hand, we can pick a large $\alpha$ and get to a neighborhood of the global minimum quickly but we will then have a large error leftover at the end.

We should therefore pick the step-size to decay with time if we actually want to converge. But we do not want to decay too quickly, which might slow down the progress. A good schedule to pick is such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty.$$

**Theorem 8** (SGD convergence with decaying step-size)**.** *If we pick a step-size schedule*

$$\alpha_t = \frac{\alpha_0}{(t+t_0)} \quad \textit{where} \quad \alpha_0 > 1/m,\ t_0 \textit{ is such that } \alpha_1 < \frac{1}{L\sigma} < \frac{1}{L}.$$

*then the expected optimality gap satisfies*

$$\mathop{\mathbb{E}}_{\omega_1,\ldots,\omega_t}\left[f(x^{(t+1)}) - f(x^*)\right] \leq \mathcal{O}\left(\frac{1}{t+t_0}\right).$$

We will not do the proof. If you are interested, see Theorem 4.7 in Bottou et al. (2018). Notice that by decaying the step-size we converge only at a sub-linear rate even for strongly convex loss functions.

**Remark 9** (Mini-batching in SGD)**.** Does mini-batching provide an improvement in the number of iterations? If we use the gradient

$$\nabla f_{\mathscr{b}}$$

instead of $\nabla f_{\omega_t}$ to make updates, the variance of the stochastic gradients decreases by a factor of $\mathscr{b}$. The constants $\sigma$ and $\sigma_0$ therefore decrease by a factor of $\mathscr{b}$ as well (remember that we want as tight an inequality as we can get when we want to analyze the converge rate of an algorithm). This modifies the theorem for expected optimality gap to

$$\mathop{\mathbb{E}}_{\omega_1,\omega_2,\ldots,\omega_t}\left[f(x^{(t+1)})\right] - f(x^*) \leq \frac{\alpha L \sigma_0}{2m\mathscr{b}} + (1-\alpha m)^t\left(f(x^0) - f(x^*) - \frac{\alpha L\sigma_0}{2m\mathscr{b}}\right).$$

Compare to the convergence rate for single-sample SGD update and notice that if we use a step-size $\alpha/\mathscr{b}$ in single-sample SGD, we obtain a similar expression expect that the contraction rate $(1-\alpha m)^t$ for single-sample SGD with this step-size is

$$1 - \frac{\alpha}{m\mathscr{b}}.$$

This roughly indicates that if we were to use single-sample SGD, we need to run $O(\mathscr{b})$ more iterations to achieve the same optimality gap as mini-batch SGD with a batch-size of $\mathscr{b}$. Each iteration of mini-batch SGD is $\mathscr{b}$ times more expensive than one iteration of single-sample SGD. We do not gain anything by using mini-batch SGD. Then why do we use it in practice?

**Remark 10.** First show using the arithmetic-mean greater than or equal to geometric-mean inequality that $\sigma \geq 1$. It is reasonable to imagine that if we use mini-batch SGD, we could use $\mathscr{b}$ times larger step-size than the single-sample SGD. Is this correct? Notice that the condition in Theorem 7 indicates that the largest initial step-size we are allowed is $\frac{1}{L}$. Effectively, we could imagine increasing the step-size with time if we are using mini-batch SGD, we just need to careful not to use too large a step-size at the beginning.

**Remark 11.** Compare the number of computations of GD with SGD. If the average sub-optimality you desire on every sample in the dataset is less than $\mathcal{O}(1/n)$ then you should use stochastic gradient descent. If you want better accuracy, gradient descent is faster. This explains in a very simple way why stochastic gradient descent is so powerful for machine learning. We do not care about getting a very low error on the training set.

## 3. SGD AS A MARKOV PROCESS

The continuous-time point of view for gradient descent gives very quick and clean results as compared to using the discrete-time update equations. The analysis of stochastic gradient descent is quite new (most advancements like SAG, SVRG, proximal terms to control the covariance etc. are within the last 10 years). The fashion of using continuous-time analysis for stochastic algorithms is much newer, within the past 3 years, but it gives an equally powerful understanding of these algorithms.

The big problem however is that continuous-time stochastic processes are difficult to handle mathematically and involve different kinds of calculi (you may have heard of names like Ito calculus or Stratonovich calculus). Further, mathematically stochastic differential equations do not allow us to talk about one trajectory of optimization (like we always did for gradient descent). So this style of analysis does not easily give a convergence rate but it is a great mental picture and is useful for other kinds of analyses.

### 3.1. **How do our convergence results change if the function is not convex?** (picture of a convex loss function, sgd bounces around. What happens if we never decrease the step-size?)

### 3.2. **Markov process for SGD.** (picture of a discrete-state, discrete-time version of SGD)

The updates of SGD are given by

$$x^{(t+1)} = x^{(t)} - \alpha \nabla f_{\omega_t}(x^{(t)}).$$

The iterate $x^{(t+1)}$ is independent of the previous iterate $x^{(t-1)}$ given the current iterate $x^{(t)}$.

Let us imagine a Markov chain with a trajectory $X^{(0)}, X^{(1)}, \ldots, X^{(t)}$. It proceeds as follows:

$$\begin{aligned} X^{(0)} &= X_0 \\ X^{(t+1)} &\sim \mathbb{P}(X^{(t+1)} | X^{(t)}). \end{aligned}$$

This trajectory depends both on the initial condition $X_0$ and the sample from the transition kernel $\mathbb{P}(\cdot \mid X^{(t)})$. What is the transition kernel

$$\mathbb{P}\left(X^{(t+1)} | X^{(t)}\right)$$

for SGD? If you take the expectation with respect to $\omega_t$ conditioned on $x^{(t)}$ in the update equation we have

$$\mathop{\mathbb{E}}_{\omega_t}\left[X^{(t+1)} \mid X^{(t)}\right] = X^{(t)} - \alpha \nabla f(X^{(t)}),$$

as expected the update of SGD is the update of gradient descent in expectation. What is the second moment?

$$\begin{aligned}\mathrm{Var}_{\omega_t}\left(X^{(t+1)} \mid X^{(t)}\right) &= \mathrm{Var}_{\omega_t}\left(X^{(t+1)} - X^{(t)} \mid X^{(t)}\right)\\ &= \mathrm{Var}_{\omega_t}\left(-\alpha \nabla f_\omega(X^{(t)})\right)\\ &= \alpha^2 \mathop{\mathbb{E}}_{\omega}\left[\left(\nabla f_\omega(x^{(t)}) - \nabla f(X^{(t)})\right)\left(\nabla f_\omega(X^{(t)}) - \nabla f(X^{(t)})\right)^\top\right].\end{aligned}$$

Again as expected the variance of the stochastic update is proportional to $\alpha^2$. The standard deviation is this proportional to $\alpha$ and we have seen this before: there is a factor of $\alpha$ in the first term for the expression of the convergence rate of SGD.

**Assumption 12.** The function $f(x)$ is smooth.

**Assumption 13.** We are sampling with replacement, so the two gradients $\nabla f_\omega$ and $\nabla f_{\omega'}$ are independent, i.e.,

$$\mathop{\mathbb{E}}_{\omega,\omega'}\left[\left(\nabla f_\omega(x^{(t)}) - \nabla f(x^{(t)})\right)\left(\nabla f_{\omega'}(x^{(t)}) - \nabla f(x^{(t)})\right)^\top\right] = 0.$$

Using the second assumption above we can obtain the variance for mini-batch SGD as

$$\begin{aligned}&\mathrm{Var}\left(X^{(t+1)} - X^{(t)} \mid X^{(t)}\right)\\ &= \alpha^2 \mathrm{Var}_{\omega^1,\ldots,\omega^{\mathscr{b}}}\left[\frac{1}{\mathscr{b}}\sum_{i=1}^{\mathscr{b}} \nabla f_{\omega^i}(X^{(t)})\right]\\ &= \frac{\alpha^2}{\mathscr{b}^2}\sum_{i=1}^{\mathscr{b}} \mathrm{Var}(\nabla f_{\omega^i}(X^{(t)}))\\ &= \frac{\alpha^2}{\mathscr{b}}\mathrm{Var}(\nabla f_\omega(X^{(t)})).\end{aligned}$$

The last equality follows because we are sampling with replacement, each of the $\omega^i$ are identically distributed. Next we will assume something quite brutal. We will assume that the matrix $\mathrm{Var}(\nabla f_\omega(X^{(t)}))$ does not depend on $X^{(t)}$, and moreover, that it is simply identity.

**Assumption 14.** The variance of the SGD update at time $t$ is such that

$$\mathrm{Var}\left(X^{(t+1)} - X^{(t)} \mid X^{(t)}\right) = \frac{\alpha^2}{\mathscr{b}} I_{d\times d}.$$

We are now ready to *approximate* the updates of SGD.

$$\begin{aligned}X^{(t+1)} &= \underbrace{X^{(t)} - \alpha \nabla f(X^{(t)})}_{\text{mean}} + \underbrace{\text{noise}}_{\text{standard deviation}}\\ &= X^{(t)} - \alpha \nabla f(X^{(t)}) + \sqrt{\frac{\alpha}{\mathscr{b}}}\,\zeta^{(t)}.\end{aligned} \tag{3}$$

where $\zeta^{(t)} \sim N(0, \alpha I_{d\times d})$ is a Gaussian random variable which is zero mean and variance $\alpha$.

**Remark 15.** The above equation is a discrete-time equation but the state of the Markov chain $X^{(t)}$ is no longer finite. Simulating this equation is easy. Notice that if we have a gradient flow

$$\dot{X} = -\nabla f(X)$$

we simulate it by discretizing time as

$$x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}).$$

We can think of the step-size $\alpha$ in our standard gradient descent update as the discretization interval of the time variable. We can also simulate this stochastic update equation by taking the full-gradient $\nabla f(X^{(t)})$ and adding Gaussian noise to it of variance $\alpha^2/\mathcal{b}$. This is not exactly SGD but only a model for it.

3.3. **Gibbs distribution.** We now have a model for SGD as a discrete-time Markov chain. What is the steady-state distribution of this Markov chain? The answer is given by what is known as the Gibbs distribution. If we have a Markov chain where each successive update is given by

$$X^{(t+1)} = X^{(t)} - \alpha \nabla f(X^{(t)}) + \sqrt{\frac{2}{\beta}}\, \zeta^{(t)} \tag{4}$$

and $\zeta^{(t)}$ is a Gaussian random variable with mean zero and variance $\alpha I$, the Gibbs distribution is

$$\rho^{\infty}(x) := \lim_{t\to\infty} \mathbb{P}(X^{(t+1)} = x) = \frac{1}{Z(\beta)} e^{-\beta f(x)}.$$

The normalization constant $Z(\beta)$ ensures that the Gibbs distribution is a legitimate probability density, i.e., $\int \rho^{\infty}(x)\, \mathrm{d}x = 1$. We therefore have

$$Z(\beta) = \int e^{-\beta f(x)}\, \mathrm{d}x.$$

This distribution exists uniquely under certain technical conditions on $f(x)$ which we will implicitly assume. None of these conditions however require that $f(x)$ be convex. For our model of SGD notice that we have

$$\beta^{-1} = \frac{\alpha}{2\mathcal{b}}.$$

(picture of Gibbs distribution)

**Remark 16.** Let us list a few properties of the Gibbs distribution that are apparent simply by looking at the formula.

- The probability that the iterates of SGD are found at a location $x$ is proportional to $e^{-\beta f(x)}$. If the training loss $f(x)$ is high, this probability is low and if the training loss is low, the probability is high. The Gibbs distribution therefore shows that if we let SGD run until it equilibriates, i.e., the limit $t \to \infty$ is achieved, we have a high chance of finding the iterates that have a small training loss. This observation is powerful because it does not require us to assume that $f(x)$ is convex. However this statement does require the assumption that the steps-size $\alpha$ of SGD does not go to zero.

- The term $\beta^{-1}$ is quite common in physics where it is called the "temperature". The temperature $\beta^{-1} = \frac{\alpha}{b}$ governs fundamentally how the Gibbs distribution looks. Higher the temperature, more the noise in the iterates and vice-versa. As you can imagine, if there is lots of noise in the SGD updates, if the learning rate $\alpha$ is large or the batch-size $b$ is small, it is easy for SGD to jump over hills. This is the reason why the Gibbs distribution will be spread around the entire energy landscape at high temperatures. The Gibbs distribution is essentially uniform over the entire state-space and does not care what the loss $f(x)$ at a location is in this case. On the other hand, if the temperature is very small, the Gibbs distribution cares a lot about the training loss and the probability of finding SGD at other places diminishes. In particular, if $\beta \to \infty$, the Gibbs distribution only puts non-zero probability on the global minima of the loss function $f(x)$.
- Written in another way, if we want the Gibbs distribution to remain the same we should ensure that
$$\beta^{-1} = \frac{\alpha}{2b} \text{ is a constant.}$$
If you increased the batch-size by two times, you should also double the learning rate if you desire that the solutions of SGD are qualitatively similar.
- We have achieved something remarkable by looking at the Gibbs distribution. We have an algorithm to find the global minimum of a non-convex loss function.
  - Start from some initial condition $x^0$
  - Take lots of steps of SGD with some fixed step-size $\alpha$ until SGD equilibriates
  - Reduce the step-size $\alpha$, and take lots of steps of SGD again until it equilibriates;
  - Repeat the previous step

  This is a formal algorithm but it will converge to the global minimum of a non-convex function $f(x)$. The catch of course is that at each step we have to wait until SGD equilibriates. Remember that the Gibbs distribution is defined as the limit as $t \to \infty$, it may take an inordinately long amount of time to SGD to equilibriate. How much time does it take to equilibriate for a convex loss function?

## 4. Entropy-SGD

4.1. **Topology of the energy landscape of neural networks.** You have probably seen in previous lectures of this course how the energy landscape of deep networks looks like. Some key points about the energy landscape.

- Linear neural networks have a unique "valley" even though the shape of this valley is non-convex (Baldi and Hornik, 1989, Kawaguchi, 2016). If you want to draw a mental picture of the energy landscape, it kind of looks like the Colorado river flowing through a gorge

- We know that regularization in general breaks these connected regions. For example, weight decay will create bias SGD into regions that are closer to the origin.
- For two-layer neural networks, one can show that adding weight decay does not destroy this connectedness property. This is however quite a special property of the $\ell_2$ norm.
- Nonlinear models are generally disconnected. In the case of ReLU non-linearities with one hidden layer, we can show that if the dimensionality of the hidden layer is large enough, i.e., if the neural network is fat enough, the level set becomes more and more connected. More precisely, a path inside $L_f(\epsilon + \tau)$ exists between any two points $x^1, x^2 \in L_f(\tau)$ if
$$\epsilon \approx p^{-1/d}.$$
In other words, one has to climb a tiny hill (the height of the hill becomes tinier if $p$ is large) to go from any point to any other point in the level set. This has not been shown for deeper networks although it is expected that it is true (Freeman and Bruna, 2016, Venturi et al., 2018).

- Neural networks with polynomial activation functions lie somewhere in between linear neural networks and ones with ReLU non-linearities. We essentially use the kernel trick to understand this. If the hidden layers of the network are wide enough, we again have a connected global minimum. Roughly, the bounds indicate that the number of hidden neurons should be twice the dimensionality of the data.

$$p \geq 2d.$$

  Does this hold in practice? The answer is more or less yes for well-performing networks. The All-CNN network you used in your problem set has some layers that are wider than $d = 3 \times 32 \times 32 = 3072$. One may draw the level set for the loss function of a deep nonlinear neural network as

- Roughly speaking, the way to understand the energy landscape consists of convexifying the problem and mapping the parameters of the neural network to some other space where we can use techniques from convexity, e.g., connectedness of the level sets at all energy levels (remember the over-parametrized one-dimensional example).
- This is what theory tells us. In practice, it seems essentially impossible to detect local minima. Beware of any sentence in any paper that says "we have an algorithm for jumping of local minima/we believe the competitor's algorithm does not work because it is stuck in local minima and ours is not, etc.".
- Among the many conjectures in deep learning is that these level sets are quite regular if the energy/loss is high. They are expected to become more and more irregular as we approach the global minimum. So effectively, if one were to draw a picture of the energy landscape of a nonlinear deep neural network, it looks like lakes descending

all the way down. The ones at the top are the largest, the ones at the bottom are the most disconnected.

- It is very important to remember that such a study of over-parametrization does not tell us anything about the generalization performance of the network. We are simply making statements about the training error.

4.2. **Geometry of the energy landscape of neural networks.** We only want to develop an intuitive understanding of the material here.

Let us consider a simple model: that of a binary perceptron. We are given data $x_i \in \{-1, 1\}^d$, labels $y_i \in \{-1, 1\}$ and would like to fit a one layer neural network with sign non-linearities $\text{sign}(x) = 1$ if $x > 0$ and $\text{sign}(x) = -1$ if $x < 0$. We want to fit discrete valued weights $w \in \{-1, 1\}^d$ that minimize the errors of the perceptron:

$$f(w) = \sum_{i=1}^{n} \mathbf{1}_{\hat{y}_i = y_i}$$

where

$$\hat{y}_i = \text{sign}(w^\top x_i).$$

Everything is discrete in this problem. Each weight is $\pm 1$, each input neuron is $\pm 1$ and so are the labels $y_i = \pm 1$. We cannot take the gradient of the loss function $f(w)$ with respect to $w$ because $f(\cdot)$ is not a continuous function of $w$. Such problems are called discrete optimization problems and are typically much harder than continuous optimization problems which we have seen till now.

**Question 17.** Can you suggest an algorithm to train a binary perceptron?

**Question 18.** Are there symmetries in the binary perceptron like those in deep linear networks?

There are two kinds of questions we are interested in asking for the binary perceptron:

- Given a dataset $\{x_i, y_i\}_{i=1,\ldots,n}$ how many different $w \in \{-1, 1\}^d$ can fit this dataset. This is the discrete analog of the size of the lake in the continuous-valued case.
- How different is each of the solution obtained above? Do some solutions generalize better than others?

In order to understand generalization, we need to first construct a model for the probability distribution that generates the dataset $\{x_i, y_i\}_{i=1,\ldots,n}$. Here's a powerful way to think of generalization. It is called the teacher-student model. We assume that nature, which generates the data, is also a neural network. Let us say that the neural network that nature uses has some weights $w^*$. Nature draws $n$ vectors $x_i \in \{-1, 1\}^d$ uniformly randomly from this discrete space and feeds them through its network to get the true labels

$$y_i = \text{sign}({w^*}^\top x_i).$$

The data is random but the dataset is not random, the labels $y_i$ are a function of the specific $w^*$ that was chosen by nature

$$D = \left\{ x_i, \operatorname{sign}({w^*}^\top x_i) \right\}_{i=1,\ldots,n}.$$

We are the student, we would like to to fit a binary perceptron $w$ on this dataset. Note that if we managed to fit $w = w^*$, we will get perfect generalization, any sample $x \in \{-1,1\}^d$ will be labeled in exactly the same way by the teacher and the student.

## References

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2016). Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv:1611.01838*.

Freeman, C. D. and Bruna, J. (2016). Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*.

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594.

Kushner, H. and Yin, G. G. (2003). *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Venturi, L., Bandeira, A. S., and Bruna, J. (2018). Spurious valleys in two-layer neural network optimization landscapes. *arXiv preprint arXiv:1802.06384*.