MATH 6222 LECTURE NOTE 3: INTRODUCTION TO NEURAL NETWORK AND MACHINE LEARNING

JINGRONG WEI

ABSTRACT. This lecture note mainly introduces basic concepts in machine learning. Section 2 and Section 3 follow [2]. The proofs in Section 4 are taken from [3].

CONTENTS

1. NEURAL NETWORKS

Let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$ be a data set containing N samples. Each instance $\boldsymbol{x}_i$ is a feature vector in the feature space $\mathcal{X} \subseteq \mathbb{R}^{d_x}$. $\boldsymbol{y}_i$ is called label and $\mathcal{Y} \subseteq \mathbb{R}^{d_y}$ is the label space.

A neural network is given by specifying a sequence of $L+1$ natural numbers: $n_0 = d_x, n_1, n_2, \ldots, n_L = d_y$, and a set of L affine maps

$$T_k : \mathbb{R}^{n_k} \to \mathbb{R}^{n_{k+1}}, \quad T_k(x) = W_k x + b_k$$

for $k = 0, 1, \ldots, L-1$.

A function from $\mathcal{X} \to \mathcal{Y}$ represented by this neural network is in the form

$$f_{L,n} = T_L \circ \sigma \circ T_{L-1} \circ \sigma \cdots \circ T_2 \circ \sigma \circ T_1$$

where σ is an activation function. The first space $\mathcal{X} \subseteq \mathbb{R}^{n_0}$ is the input layer and the last one $\mathcal{Y} \subseteq \mathbb{R}^{n_L}$ is the output layer and the rest are called hidden layers. L is the number of layers and $n = \sum_i n_i$ is the size of this neural network. A *neuron* (Figure 1) is an information-processing unit that is fundamental to the operation of a neural network (Figure 2). Historically the activation function is the sigmoid function $\sigma(x) = 1/(1+e^{-x})$ and the more popular one is the rectified linear unit (ReLU) activation $\sigma(x) = \max\{0, x\}$.

Date: January 22, 2025.

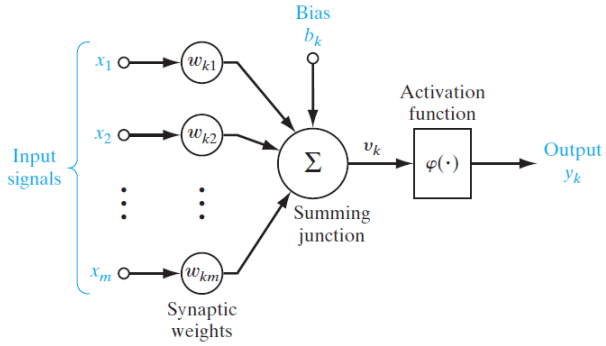

FIGURE 5 Nonlinear model of a neuron, labeled k.

FIGURE 1. Nonlinear model of a neuron, labeled k.

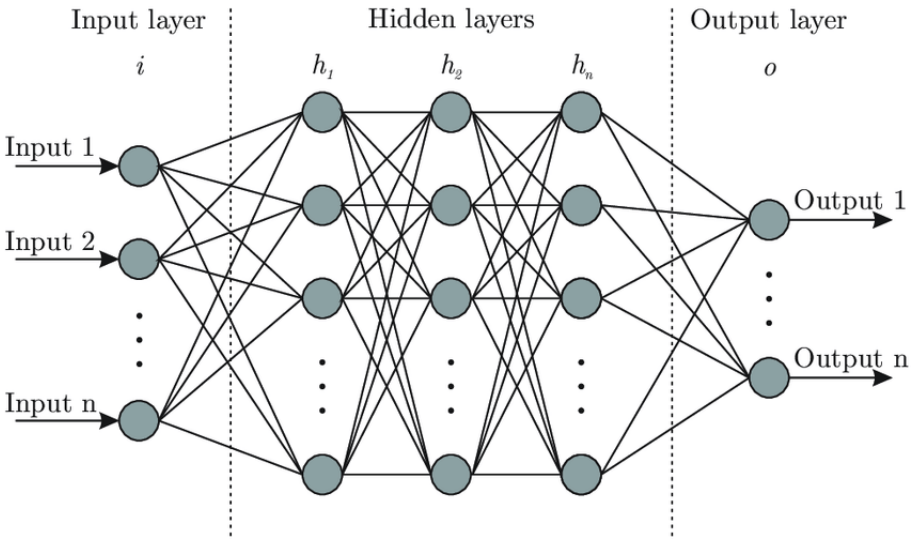

FIGURE 2. A fully connected neural network.

2. APPROXIMATION PROPERTY

A fundamental question is the approximation property of a neural network. Let us consider the simplest case: one single hidden layer, i.e., a 3-layer neural network, and the output layer is $\mathbb{R}$, the neural network function of size n is in the form

$$f_n(x) = \sum_{i=1}^{n} c_i \sigma\left(\boldsymbol{a}_i \cdot \boldsymbol{x} + b_i\right) + c_0, \quad \boldsymbol{a}_i \in \mathbb{R}^d, b_i, c_i \in \mathbb{R}. \tag{1}$$

It is shown by Barron [1] that using one single hidden layer and the sigmoidal function, there exists a n-term linear combination of sigmoidal functions f_n of the form (1) such that

$$\|f - f_n\| \leq \frac{C_f}{\sqrt{n}}. \tag{2}$$

The approximation rate $n^{-1/2}$ is universal since it is independent of the dimension in contrast to a typical rate $n^{-1/d}$, known as "the curse of dimensionality", in the classical approximation theory.

We first present an approximation result in Hilbert spaces. Let $\mathcal{H}$ be a separable Hilbert space with inner product $(\cdot,\cdot)$ and norm $\|\cdot\|$. Let $\{\phi_k\}_{k=1,2,\dots}$ be an orthonormal basis for $\mathcal{H}$. As $\{\phi_k\}_{k=1,2,\dots}$ is an orthonormal basis, we have the representation

$$f = \sum_{k=1}^{\infty} f^k \phi_k := \sum_{k=1}^{\infty} (f, \phi_k)\, \phi_k$$

and $\left\|\left\{f^k\right\}\right\|_{l_2} = \|f\| < \infty$. In Barron's result, the assumption is essentially to ask $\left\|\left\{f^k\right\}\right\|_{l_1} < \infty$ which requires additional smoothness of f as for sequences

$$\left\|\left\{f^k\right\}\right\|_{l_2}^2 = \sum_{k=1}^{\infty} \left|f^k\right|^2 \le \left(\sum_{k=1}^{\infty} \left|f^k\right|\right)^2 = \left\|\left\{f^k\right\}\right\|_{l_1}^2$$

Theorem 2.1. *Given a function $f \in \mathcal{H}$, if $\left\|\left\{f^k\right\}\right\|_{l_1} < \infty$, then for any positive integer n, there exists an n-term approximation*

$$f_n = \frac{\left\|\left\{f^k\right\}\right\|_{l_1}}{n} \sum_{i=1}^{n} \operatorname{sign}\left(f^{k_i}\right) \phi_{k_i},$$

such that

$$\|f - f_n\| \le \frac{\left(\left\|\left\{f^k\right\}\right\|_{l_1}^2 - \left\|\left\{f^k\right\}\right\|_{l_2}^2\right)^{1/2}}{\sqrt{n}}.$$

Proof. The proof is a simple modification of Barron's. Write

$$f = \sum_{k=1}^{\infty} f^k \phi_k = \sum_{k=1}^{\infty} \left|f^k\right| \operatorname{sign}\left(f^k\right) \phi_k = \sum_{k=1}^{\infty} p_k \psi_k$$

where, as $\left\|\left\{f^k\right\}\right\|_{l_1} < \infty, p_i := \left|f^k\right| / \left\|\left\{f^k\right\}\right\|_{l_1}, i = 1, 2, \dots$ defines a probability density function, and the rescaled basis $\left\{\psi_i := \left\|\left\{f^k\right\}\right\|_{l_1} \operatorname{sign}\left(f^i\right) \phi_i\right\}$.

Introduce a random variable g with distribution $\Pr\{g = \psi_i\} = p_i$ for $i = 1, 2, \dots$. Then

$$\mathbb{E}(g) = \sum_{i=1}^{\infty} p_i \psi_i = f$$

and

$$\operatorname{Var}(g) = \mathbb{E}\left(\|g\|^2\right) - \|\mathbb{E}(g)\|^2 = \left\|\left\{f^k\right\}\right\|_{l_1}^2 - \left\|\left\{f^k\right\}\right\|_{l_2}^2$$

Using this distribution, we draw n i.i.d. samples g_i and let $f_n = \frac{1}{n}\sum_{i=1}^{n} g_i$ be the sampled mean. Then by the linearity of expectation, $\mathbb{E}(f_n) = f$. By the independence of g_i, we compute the variance

$$\mathbb{E}\left(\|f_n - f\|^2\right) = \operatorname{Var}(f_n) = \frac{1}{n}\operatorname{Var}(g) \le \frac{1}{n}\left(\left\|\left\{f^k\right\}\right\|_{l_1}^2 - \left\|\left\{f^k\right\}\right\|_{l_2}^2\right).$$

Therefore there exists a realization of the sampled mean satisfies the estimate. □

Then (2) can be proved by showing that the function in the form of (1) is dense in $C(\mathbb{R}^d$ which contains the Fourier basis. Indeed it can be shown that such density result holds for any non-polynomial activation function.

3. Loss Function and Back Propogation

A loss function is usually in the form

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} J_i, \quad \text{with } J_i = J\left(W, b; \boldsymbol{x}_i, \boldsymbol{y}_i\right)$$

where we use a single letter for all weights and bias for all layers

$$W = \left(W_1, \ldots, W_L\right), \quad b = \left(b_1, \ldots, b_L\right)$$

If we denote by $\hat{\boldsymbol{y}}_i = f_{L,n}\left(\boldsymbol{x}_i; W, b\right)$, the loss function is a comparison of the true function value $\boldsymbol{y}_i$ with the predicted value $\hat{\boldsymbol{y}}_i$. The simplest example is the least square problem:

$$\min_{W,b} \mathcal{L}(W, b) = \frac{1}{N}\sum_{i=1}^{N} \left\|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i\right\|^2, \tag{3}$$

where $J_i = \left\|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i\right\|^2$ is the l^2-difference. Loss functions are designed based on the problem to be solved.

To find the optimal weights, we can apply the gradient descent method to solve (3). In machine learning tasks, this is called the training process.

To apply gradient descent, we explain the way to compute the derivative $\nabla_{W,b}\mathcal{L}$. We write the variables in a *forward pass* of the neural network:

$$\boldsymbol{x} \xrightarrow{W_1,b_1} \alpha_1 \xrightarrow{\sigma} \beta_1 \xrightarrow{W_2,b_2} \alpha_2 \xrightarrow{\sigma} \beta_2 \cdots \xrightarrow{W_{L-1},b_{L-1}} \alpha_{L-1} \xrightarrow{\sigma} \beta_{L-1} \xrightarrow{W_L,b_L} \hat{\boldsymbol{y}} \xrightarrow{J} \mathcal{L}.$$

Given weights $W_k, k = 1, \ldots, L$, we can evaluate all intermediate variables α_k, β_k.

To compute the derivative, we shall go backwards and the corresponding algorithm is known as back propagation. We start from the first two weights matrices. To be precise, we first write the component-wise formula and then abbreviate by matrix notation.

The component form of $\hat{\boldsymbol{y}} = W_L\beta_{L-1} + b_L$ is: for $i = 1, 2, \ldots, n_L$:

$$\hat{\boldsymbol{y}}^i = \sum_{j=1}^{n_{L-1}} W_L^{i,j}\beta_{L-1}^j + b_L^i,$$

where $W_L = \left(W_L^{i,j}\right)$ and $b_L = \left(b_L^1, x_L^2, \ldots, b_L^{n_L}\right)^\top$. Then for $i = 1, 2, \ldots, n_L, j = 1, 2, \ldots, n_{L-1}$:

$$\frac{\partial J_i}{\partial W_L^{i,j}} = \frac{\partial J_i}{\partial \hat{\boldsymbol{y}}^i}\frac{\partial \hat{\boldsymbol{y}}^i}{\partial W_L^{i,j}} = \frac{\partial J_i}{\partial \hat{\boldsymbol{y}}^i}\beta_{L-1}^j,$$

which can be written as

$$\frac{\partial \mathcal{L}}{\partial W_L} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial W_L} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}\beta_{L-1}^\top.$$

Here the gradient $\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}})$ is a column vector of size n_L evaluated at $\hat{\boldsymbol{y}}$ and $\beta_{L-1}^\top$ is a row vector of size n_{L-1} so that the product is a matrix with consistent dimension of W_L.

To go further, by chain rule, formally we have

$$\frac{\partial \mathcal{L}}{\partial W_{L-1}^{i,j}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \beta_{L-1}^i}\frac{\partial \beta_{L-1}^i}{\partial \alpha_{L-1}^i}\frac{\partial \alpha_{L-1}^i}{\partial W_{L-1}^{i,j}} = \frac{\partial \mathcal{L}}{\partial \beta_{L-1}^i}\frac{\partial \alpha_{L-1}^i}{\partial W_{L-1}^{i,j}}$$

for $i = 1, 2, \ldots, n_L, j = 1, 2, \ldots, n_{L-1}$, which can be written as

$$\frac{\partial \mathcal{L}}{\partial W_{L-1}} = \left(\frac{\partial \mathcal{L}}{\partial \beta_{L-1}} .* \frac{\partial \beta_{L-1}}{\partial \alpha_{L-1}}\right)\frac{\partial \alpha_{L-1}}{\partial W_{L-1}} = \left(\frac{\partial \mathcal{L}}{\partial \beta_{L-1}} .* \frac{\partial \beta_{L-1}}{\partial \alpha_{L-1}}\right)\beta_{L-2}^\top.$$

We now introduce notation to simplify the computation. Denoted by $g_L = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}, g_k = \frac{\partial \mathcal{L}}{\partial \alpha_k}$ and $l_k = \frac{\partial \mathcal{L}}{\partial \beta_k}$ for $k = L-1, \ldots, 1$. The vectors g_k, l_k can be computed by the backward pass (back propagation). Using g_L, we have

$$l_{L-1} = \frac{\partial \mathcal{L}}{\partial \beta_{L-1}} = W_L^\top \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} = W_L^\top g_L.$$

The derivative $\frac{\partial \beta_{L-1}^i}{\partial \alpha_{L-1}^i} = \sigma'\left(\alpha_{L-1}^i\right)$, and the action $l_{L-1}. * \frac{\partial \beta_{L-1}}{\partial \alpha_{L-1}} := l_{L-1}. * \sigma'\left(\alpha_{L-1}\right)$ is the component-wise product (so-called Hadamard product) and . * is the MATLAB notation for such product. For sigmoidal function σ, there is a simple formula on its derivative

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

For ReLU, the derivative is even simpler $\sigma'(x) = 1$ if $x \geq 0$ and $\sigma'(x) = 0$ otherwise.

Inductively going backword, we have the diagram

$$x \overset{W_1^\top}{\leftarrow} g_1 \overset{\sigma'}{\leftarrow} l_1 \overset{W_2^\top}{\leftarrow} g_2 \overset{\sigma'}{\leftarrow} l_2 \cdots \overset{W_{L-1}^\top}{\leftarrow} g_{L-1} \overset{\sigma'}{\leftarrow} l_{L-1} \overset{W_L^\top}{\leftarrow} g_L \overset{J'}{\leftarrow} \mathcal{L}.$$

If we collect the forward pass and backward pass together, we can find they are kind of dual to each other which makes the back propagation is easy to derive.

$$\begin{aligned}
&\boldsymbol{x} \xrightarrow{W_1} \alpha_1 \xrightarrow{\sigma} \beta_1 \xrightarrow{W_2} \alpha_2 \xrightarrow{\sigma} \beta_2 \ldots \xrightarrow{W_{L-1}} \alpha_{L-1} \xrightarrow{\sigma} \beta_{L-1} \xrightarrow{W_L} \hat{\boldsymbol{y}} \xrightarrow{J} \mathcal{L} \\
&\boldsymbol{x} \overset{W_1^\top}{\leftarrow} g_1 \overset{\sigma'}{\leftarrow} l_1 \overset{W_2^\top}{\leftarrow} g_2 \overset{\sigma'}{\leftarrow} l_2 \cdots \overset{W_{L-1}^\top}{\leftarrow} g_{L-1} \overset{\sigma'}{\leftarrow} l_{L-1} \overset{W_L^\top}{\leftarrow} g_L \overset{J'}{\leftarrow} \mathcal{L}.
\end{aligned}$$

The derivative to the weights can be computed by

$$\frac{\partial \mathcal{L}}{\partial W_k} = g_k \beta_{k-1}^\top, \quad \frac{\partial \mathcal{L}}{\partial b_k} = g_k.$$

Noted that $\nabla_{(W,b)} \mathcal{L} = \sum_{i=i}^N \nabla_{(W,b)} J_i$. In each iteration of gradient descent, we need to compute the back propagation sequence for each data pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. To save compuatation cost, we shall introduce stochastic gradient descent, which approximates the derivative using only partial data.

4. Stochastic Gradient Descent

We use $x = (W, b)$ to denote the weights and bias. Let $x^0 \in \mathbb{R}^d$ and $\{\gamma_k > 0\}$ a sequence of step sizes. The Stochastic Gradient Descent (SGD) algorithm is given by

$$x^{k+1} = x^k - \gamma_k \nabla J_{i_k}\left(x^k\right), \tag{4}$$

where $i_k \in \{1, \ldots n\}$ is sampled with probability $\frac{1}{n}$.

Given two random variables x, y in $\mathbb{R}^d$, we note:

- the *expectation* of x as $\mathbb{E}[x]$,
- the *expectation of x conditioned to y* as $\mathbb{E}[x|y]$,
- the *variance* of x as $\mathbb{V}[x] := \mathbb{E}\left[\|x - \mathbb{E}[x]\|^2\right]$.

An important feature of the SGD Algorithm is that at each iteration we follow the direction $-\nabla J_{i_k}\left(x^k\right)$, which is an *unbiaised estimator* of $-\nabla \mathcal{L}\left(x^k\right)$. Indeed, since

$$\mathbb{E}\left[\nabla J_i\left(x^k\right) \mid x^k\right] = \sum_{i=1}^n \frac{1}{n} \nabla J_i\left(x^k\right) = \nabla \mathcal{L}\left(x^k\right).$$

Lemma 4.1. *Suppose J_i is convex and L_i-smooth. Let $L_{\max} = \max_i L_i$. Then L is $L_{\max}$-smooth in expectation, in the sense that for all $x, y \in \mathbb{R}^d$,*

$$\frac{1}{2L_{\max}}\mathbb{E}\left[\|\nabla J_i(y) - \nabla J_i(x)\|^2\right] \leq \mathcal{L}(y) - \mathcal{L}(x) - \langle \nabla \mathcal{L}(x), y - x\rangle.$$

Proof. For each $i = 1, 2, \ldots, N$, using co-coercivity applied to J_i, together with the fact that $L_i \leq L_{\max}$, allows us to write

$$\frac{1}{2L_{\max}}\|\nabla J_i(y) - \nabla J_i(x)\|^2 \leq J_i(y) - J_i(x) + \langle \nabla J_i(x), y - x\rangle .$$

To conclude, multiply the above inequality by $\frac{1}{N}$, and sum over i, using the fact that $\frac{1}{N}\sum_i J_i = \mathcal{L}$ and $\frac{1}{N}\sum_i \nabla J_i = \nabla \mathcal{L}$.

□

As direct consequence we have the following bound at x^*.

Corollary 4.2. *Suppose J_i is convex and L_i-smooth. Let $L_{\max} = \max_i L_i$. Then, for every $x \in \mathbb{R}^d$ and every $x^* \in \operatorname{argmin} \mathcal{L}$, we have that*

$$\frac{1}{2L_{\max}}\mathbb{E}\left[\|\nabla J_i(x) - \nabla J_i(x^*)\|^2\right] \leq \mathcal{L}(x) - \mathcal{L}(x^*).$$

Lemma 4.3 (Variance transfer : gradient noise)**.** *Suppose J_i is convex and L_i-smooth. Let $L_{\max} = \max_i L_i$. Then for all $x \in \mathbb{R}^d$ we have that*

$$\mathbb{E}\left[\|\nabla J_i(x)\|^2\right] \leq 4L_{\max}(\mathcal{L}(x) - \inf \mathcal{L}) + 2\sigma_{\mathcal{L}}^*$$

where $\sigma_L^* = \mathbb{V}\left[\nabla J_i(x^*)\right]$ for $x^* \in \operatorname{argmin} \mathcal{L}$.

Proof. Noted that

$$\|\nabla J_i(x)\|^2 \leq 2\|\nabla J_i(x) - \nabla J_i(x^*)\|^2 + 2\|\nabla J_i(x^*)\|^2$$

Taking the expectation over the above inequality, then applying Corollary 4.2 gives the result. □

Theorem 4.4 (Convergence rate of SGD for strongly convex problems)**.** *Assume J_i is convex and L_i-smooth. Suppose $\mathcal{L}$ is μ-strongly convex and let $L_{\max} = \max_i L_i$. Consider $\left\{x^k\right\}_{k=0,1,2,\ldots}$ the sequence generated by the SGD algorithm (4) with a constant stepsize satisfying $0 < \gamma \leq \frac{1}{2L_{\max}}$. It follows that for $k \geq 0$,*

$$\mathbb{E}\left\|x^k - x^*\right\|^2 \leq (1 - \gamma\mu)^k \left\|x^0 - x^*\right\|^2 + \frac{2\gamma}{\mu}\sigma_{\mathcal{L}}^*$$

where $\sigma_{\mathcal{L}}^* = \mathbb{V}\left[\nabla J_i(x^*)\right]$ for $x^* = \operatorname{argmin} \mathcal{L}$.

Proof. We will note $\mathbb{E}_k[\cdot]$ instead of $\mathbb{E}\left[\cdot \mid x^k\right]$, for simplicity. Using the definition of SGD and expanding the squares we have

$$\begin{aligned}\left\|x^{k+1} - x^*\right\|^2 &= \left\|x^k - x^* - \gamma \nabla J_i\left(x^k\right)\right\|^2 \\ &= \left\|x^k - x^*\right\|^2 - 2\gamma\left\langle x^k - x^*, \nabla J_i\left(x^k\right)\right\rangle + \gamma^2\left\|\nabla J_i\left(x^k\right)\right\|^2\end{aligned}$$

Taking expectation conditioned on x^k we obtain

$$\begin{aligned}\mathbb{E}_k\left[\left\|x^{t+1} - x^*\right\|^2\right] &= \left\|x^k - x^*\right\|^2 - 2\gamma\left\langle x^k - x^*, \nabla \mathcal{L}\left(x^k\right)\right\rangle + \gamma^2\mathbb{E}_k\left[\left\|\nabla J_i\left(x^k\right)\right\|^2\right] \\ &\leq (1 - \gamma\mu)\left\|x^k - x^*\right\|^2 - 2\gamma\left[\mathcal{L}\left(x^k\right) - \mathcal{L}\left(x^*\right)\right] + \gamma^2\mathbb{E}_k\left[\left\|\nabla J_i\left(x^k\right)\right\|^2\right]\end{aligned}$$

Taking expectations again and using a variance transfer (see Lemma 4.3) gives

$$\begin{aligned}\mathbb{E}\left[\left\|x^{k+1}-x^*\right\|^2\right] &\leq (1-\gamma\mu)\mathbb{E}\left\|x^k-x^*\right\|^2+2\gamma^2\sigma_f^*+2\gamma\left(2\gamma L_{\max}-1\right)\mathbb{E}\left[\mathcal{L}\left(x^k\right)-\mathcal{L}\left(x^*\right)\right]\\ &\leq (1-\gamma\mu)\mathbb{E}\left[\left\|x^k-x^*\right\|^2\right]+2\gamma^2\sigma_f\mathcal{L}^*,\end{aligned}$$

where we used in the last inequality that $2\gamma L_{\max}\leq 1$ since $\gamma\leq\frac{1}{2L_{\max}}$. Recursively applying the above and summing up the resulting geometric series gives

$$\begin{aligned}\mathbb{E}\left\|x^k-x^*\right\|^2 &\leq (1-\gamma\mu)^k\left\|x^0-x^*\right\|^2+2\sum_{j=0}^{k-1}(1-\gamma\mu)^j\gamma^2\sigma_f^*\\ &\leq (1-\gamma\mu)^k\left\|x^0-x^*\right\|^2+\frac{2\gamma\sigma_f^*}{\mu}.\end{aligned}$$

□

For every $\varepsilon>0$, we can guarantee that $\mathbb{E}\left\|x^K-x^*\right\|^2\leq\varepsilon$ provided that

$$\gamma=\min\left\{\varepsilon\frac{\mu}{4\sigma_f^*},\frac{1}{2L_{\max}}\right\}\quad\text{and}\quad K\geq\max\left\{\frac{1}{\varepsilon}\frac{4\sigma_f^*}{\mu^2},\frac{2L_{\max}}{\mu}\right\}\log\left(\frac{2\left\|x^0-x^*\right\|^2}{\varepsilon}\right).$$

Compared with gradient decent, the step size is further restricted by the variance. To reduce the variance and enlarge the step size, one approach is to use the *minibatching SGD*: let $B_k\subseteq\{1,\dots n\}$ be a set of size b sampled uniformly among $\{1,\dots n\}$, then

$$x^{k+1}=x^k-\gamma_k\nabla\mathcal{L}_{B_k}\left(x^k\right). \tag{5}$$

where $\mathcal{L}_{B_k}(\cdot)=\frac{1}{|B_k|}\sum_{i\in B_k}J_i(\cdot)$.

Theorem 4.5. *Suppose J_i is convex and L_i-smooth. Assume further that $\mathcal{L}$ is μ-strongly convex and L-smooth. Consider $\left\{x^k\right\}_{k=0,1,2,\dots}$ the sequence generated by the minibatching SGD algorithm (5) with a constant stepsize satisfying $0<\gamma\leq\frac{1}{2L_b}$. It follows that for $k\geq 0$,*

$$\mathbb{E}\left[\left\|x^k-x^*\right\|^2\right]\leq(1-\gamma\mu)^t\left\|x^0-x^*\right\|^2+\frac{2\gamma\sigma_b^*}{\mu},$$

where

$$\mathcal{L}_b=\frac{n(b-1)}{b(n-1)}L+\frac{n-b}{b(n-1)}L_{\max},\quad\sigma_b^*=\frac{n-b}{b(n-1)}\sigma_{\mathcal{L}}^*.$$

Remark 4.6. *For $b=1$, minibatching SGD reduces to SGD and the results in 4.4. for $b=n$ minibatching SGD reduces to GD, we see that $L_b=L$ and $\sigma_b^*=0$. We recover the fact that the behavior of (GD) is controlled by the Lipschitz constant L, and has no variance.*

Some online resources:

- A neural network playground using tensorflow: https://playground.tensorflow.org/
- Simple Neural Network for MNIST numpy from scratch: https://www.kaggle.com/code/scaomath

REFERENCES

[1] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993. 2

[2] L. Chen and S. Cao. A brief introduction to deep learning. *Lecture note*, 2020. 1

[3] G. Garrigos and R. M. Gower. Handbook of convergence theorems for (stochastic) gradient methods. *arXiv preprint arXiv:2301.11235*, 2023. 1