1 Convex Optimization

We will consider convex optimization problems. Namely, minimization problems where the objective is convex (we assume no constraints for now). Such problems often arise in machine learning. For example, the SVM optimization problem is convex.

Recall that $f(\boldsymbol{w})$ is convex if for all $\boldsymbol{w}_1, \boldsymbol{w}_2$:

$$f(\boldsymbol{w}_2) \geq f(\boldsymbol{w}_1) + \nabla f(\boldsymbol{w}_1) \cdot (\boldsymbol{w}_2 - \boldsymbol{w}_1) \tag{1}$$

An alternative condition for convexity is that $\forall \boldsymbol{w}_1, \boldsymbol{w}_2$ it holds that:

$$(\nabla f(\boldsymbol{w}_1) - \nabla f(\boldsymbol{w}_2)) \cdot (\boldsymbol{w}_1 - \boldsymbol{w}_2) \geq 0 \tag{2}$$

1.1 Stochastic Gradient Descent

The derivation here follows [1], Chapter 3. Consider minimizing the function $f(\boldsymbol{w})$, which is given as a sum:

$$f(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} f_i(\boldsymbol{w}) \tag{3}$$

where $f_i(\boldsymbol{x})$ are convex functions.

Now consider the SGD algorithm defined as follows. We use W_i for the i^{th} value, since it is a random variable.

- Set $W_1 = 0$
- Sample V_t such that $\mathbb{E}\left[V_t | W_t\right] = \nabla f(W_t)$. Update $W_{t+1} = W_t - \eta V_t$.

In particular, V_t can be obtained by sampling an index j uniformly in $\{1, \ldots, m\}$ and returning $V_t = \nabla f_j(\boldsymbol{w}_t)$ (see slides for why its expected value is indeed $\nabla f(W_t)$)

Now let's analyze the quality of the averaged vector $\bar{W} = \frac{1}{T} \sum_{t=1}^{T} W_t$. We will show the following result.

Theorem 1.1. *Denote by $\boldsymbol{w}^*$ the minimizer of $f(\boldsymbol{w})$ and assume $\|\boldsymbol{w}^*\|_2 \leq B$. Also assume $\|V_t\|_2 \leq G$ for all i. Then:*

$$\mathbb{E}\left[f(\bar{W}) - f(\boldsymbol{w}^*)\right] \leq \frac{BG}{\sqrt{T}} \tag{4}$$

Proof.

$$f(\bar{W}) = f\left(\frac{1}{T} \sum_{t=1}^{T} W_t\right) \leq \frac{1}{T} \sum_{t=1} f(W_t) \tag{5}$$

where we have used the Jensen inequality, which says that for a convex function, the average of its values is greater than its value on the average.

We would like to compare this to the optimum, so:

$$f(\bar{W}) - f(\boldsymbol{w}^*) \leq \frac{1}{T}\sum_{t=1}[f(W_t) - f(\boldsymbol{w}^*)] \tag{6}$$

Using convexity we have:

$$\frac{1}{T}\sum_{i}[f(W_t) - f(\boldsymbol{w}^*)] \leq \frac{1}{T}\sum_{i}\nabla f(W_t)(W_t - \boldsymbol{w}^*) \tag{7}$$

Now, in SGD, the $W_1, \ldots, W_n$ are random variables, because of the randomness of the gradient estimate. Denote these by $W_1, \ldots, W_T$. Thus, the above difference is a random variable, and we would like to say that it is small. Here we will show this in the expected sense. Denote by $V_1, \ldots, V_n$ the stochastic gradient estimates, as sampled during the algorithm. Note that each V_t is sampled based on W_t such that:

$$\mathbb{E}\left[V_t | W_t = \boldsymbol{w}_t\right] = \nabla f(\boldsymbol{w}_t) \tag{8}$$

Similarly:

$$\mathbb{E}\left[V_t W_t | W_t = \boldsymbol{w}_t\right] = \nabla f(\boldsymbol{w}_t)\boldsymbol{w}_t \tag{9}$$

And:

$$\mathbb{E}\left[V_t W_t\right] = \mathbb{E}\left[\nabla f(W_t) W_t\right] \tag{10}$$

We can now take the expected value of Eq. (6) to get:

$$\mathbb{E}\left[f(\bar{W}) - f(\boldsymbol{w}^*)\right] \leq \frac{1}{T}\sum_{i}\mathbb{E}\left[\nabla f(W_t)(W_t - \boldsymbol{w}^*)\right] \tag{11}$$

and using Eq. (10) this is equivalent to:

$$\mathbb{E}\left[f(\bar{W}) - f(\boldsymbol{w}^*)\right] \leq \frac{1}{T}\sum_{i}\mathbb{E}\left[V_t(W_t - \boldsymbol{w}^*)\right] \tag{12}$$

We now use the SGD update form to write:

$$\begin{aligned} \|W_{t+1} - \boldsymbol{w}^*\|_2^2 &= \|W_i - \eta V_t - \boldsymbol{w}^*\|_2^2 \\ &= \|W_t - \boldsymbol{w}^*\|_2^2 + \eta^2\|V_t\|_2^2 - 2\eta\left(W_i - \boldsymbol{w}^*\right)V_t \end{aligned}$$

Rearranging we have:

$$V_t(W_t - \boldsymbol{w}^*) = \frac{\|W_t - \boldsymbol{w}^*\|_2^2 - \|W_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta} + 0.5\eta\|V_t\|_2^2 \tag{13}$$

From Eq. (13) we get:

$$\begin{aligned} \mathbb{E}\left[f(\bar{W}) - f(\boldsymbol{w}^*)\right] &\leq \frac{1}{T}\sum_{t}\mathbb{E}\left[\frac{\|W_t - \boldsymbol{w}^*\|_2^2 - \|W_{t+1} - \boldsymbol{w}^*\|_2^2}{2\eta} + \frac{\eta}{2}\|V_t\|_2^2\right] \\ &= \frac{1}{2\eta T}\|\boldsymbol{w}^*\|_2^2 - \frac{1}{2\eta T}\mathbb{E}\left[\|W_{T+1} - \boldsymbol{w}^*\|_2^2\right] + \frac{\eta}{2T}\sum_{t}\mathbb{E}\left[\|V_t\|_2^2\right] \\ &\leq \frac{1}{2\eta T}\|\boldsymbol{w}^*\|_2^2 + \frac{\eta}{2T}\sum_{t}\mathbb{E}\left[\|V_t\|_2^2\right] \leq \frac{B^2}{2\eta T} + \frac{\eta G^2}{2} \end{aligned}$$

where we used the fact the series is telescoping, and for the last inequality we dropped a negative term. Setting $\eta = \frac{B}{G\sqrt{T}}$ we get:

$$\mathbb{E}\left[f(\bar{W}) - f(\boldsymbol{w}^*)\right] \leq \frac{BG}{\sqrt{T}} \tag{14}$$

Namely, after T iterations we have $O\left(\frac{1}{\sqrt{T}}\right)$ error. □

2 Deep Learning

In deep learning we are interested in functions that correspond to sequences of linear transformation followed by a non-linearity.

Formally, assume $\boldsymbol{x}$ is out input, and denote $\boldsymbol{z}_0 = \boldsymbol{x}$. Now define the calculation recursively:

$$\boldsymbol{z}_{t+1} = h(W_{t+1}\boldsymbol{z}_t) \tag{15}$$

Here $h : \mathbb{R} \to \mathbb{R}$ is a non-linear function. Some examples are:

$$\begin{aligned} h(z) &= \mathrm{R}ELU(z) = \max[0, z] \\ h(z) &= \mathrm{S}IGMOID(z) = \frac{1}{1+e^{-z}} \\ h(z) &= \mathrm{T}ANH(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \end{aligned}$$

The output of the neural net depends on the task we want to perform. Let us focus for simplicity on binary classification. In which case the output is a scalar, obtained as the transformation:

$$z_L = h(\boldsymbol{w}_L \cdot \boldsymbol{z}_{L-1}) \tag{16}$$

Note that often this last transformation is taken to be linear. For ease of derivation we assume it is the same non-linear transformation as we had in the previous layers.

The resulting classifier is then taken to be:

$$f(\boldsymbol{x}; W) = \mathrm{sign}\,(z_L) \tag{17}$$

where W stands for all the parameters of the model. Note that it would also make sense to add a bias term here so that classification is $\mathrm{sign}\,(z_L - b)$ (of course this will be absolutely required if h has non-negative outputs).

2.1 Training Deep Learning by SGD and Backpropagation

Assume we have training data in the form of pairs $\boldsymbol{x}_i, y_i$ for $y_i \in \{+1, -1\}$. Then we can follow the standard ERM approach of minimizing some approximation

of the classification error. As we discussed earlier, two natural loss function are the hinge loss or the logistic loss.

$$\begin{aligned} \ell_{\text{ hinge}}(y, z_L) &= \max[1 - yz_L, 0] \\ \ell_{\text{ logistic}}(y, z_L) &= \log\left(1 + e^{-yz_L}\right) \end{aligned}$$

Note that the logistic loss will go to zero when y, z have the same sign and $z \to \infty$.

Introduce some notation:

- Denote $\boldsymbol{v}_t = W_{t+1}\boldsymbol{z}_t$. Namely, these are the values of the neurons in layer t before the activation function.
- Let $\boldsymbol{w}_{t,i}$ denote the i^{th} row of the matrix W_t. So, these are the input weights of neuron i in the layer t.

The loss we want to minimize is then (we just use ℓ for the loss. It can be hinge, logistic or something else):

$$f(W) = \sum_{i=1} \ell(y_i, z_L(\boldsymbol{x}_i, W)) \tag{18}$$

Note we could have included a regularization term, and this is indeed often used.

The simplest and most effective way of minimizing the above is SGD. To implement it, we only need to calculate the gradient $\nabla\ell(\boldsymbol{x}_i, y_i, W)$. There are many software tools that can do this differentiation automatically. Here we derive the gradient, to better understand its structure. This gradient calculation, which is fundumental to any deep learning optimization, is called backpropagation for reasons that will soon be clear.

We first recall the chain rule. Start with a simple case where:

$$f(x) = y(z(x)) \tag{19}$$

Then:

$$\frac{\partial f}{\partial y} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} \tag{20}$$

Similarly in the multivariate case:

$$f(x) = y(z_1(x), \ldots, z_m(x)) \tag{21}$$

Then:

$$\frac{\partial f}{\partial x} = \sum_i \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial x} \tag{22}$$

Now say we want the gradient wrt $W_{t,i,j}$ or equivalently the j^{th} weight in the vector $\boldsymbol{w}_{t,i}$. We recall that $\boldsymbol{w}_{t,i}$ appears in the objective only in $z_{t,i}$. Thus we can use the chain rule to write:

$$\frac{\partial\ell(y, z_L(\boldsymbol{x}, W))}{\partial W_{t,i,j}} = \frac{\partial\ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t,i}}\frac{\partial z_{t,i}}{\partial W_{t,i,j}} \tag{23}$$

Now recall:

$$z_{t,i} = h(\boldsymbol{w}_{t,i} \cdot \boldsymbol{z}_{t-1}) = h(v_{t,i}) \tag{24}$$

So we have:

$$\frac{\partial z_{t,i}}{\partial W_{t,i,j}} = h'(v_{t,i}) z_{t-1,i,j} \tag{25}$$

Putting things together we have:

$$\frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial W_{t,i,j}} = \frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t,i}} h'(v_{t,i}) z_{t-1,j} \tag{26}$$

Note that our result depends on the derivative of the loss wrt $\boldsymbol{z}_t$. We turn to show that this can be calculated recursively (from the top layer to the input).

$$\frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t,i}} = \sum_r \frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t+1,r}} \frac{\partial z_{t+1,r}}{\partial z_{t,i}} \tag{27}$$

Again using Eq. (24) we get:

$$\frac{\partial z_{t+1,r}}{\partial z_{t,i}} = h'(v_{t+1,r}) W_{t+1,r,i} \tag{28}$$

So together with Eq. (27) we have:

$$\sum_r \frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t+1,r}} h'(v_{t+1,r}) W_{t+1,r,i} \tag{29}$$

To simplify things denote:

$$\delta_{t,r} = \frac{\partial \ell(y, z_L(\boldsymbol{x}, W))}{\partial z_{t,r}} \tag{30}$$

And denote $\boldsymbol{\delta}_t$ the vector of all values for given t.

Then the backpropagation calculation for calculating the gradients at weight W amounts to the following steps:

- Use W to run the network forward on $\boldsymbol{x}$ and calculate all $\boldsymbol{v}_t$ values (i.e., the outputs of the linear calculation at each step).
- Recursively calculate $\boldsymbol{\delta}_t$ using:

$$\delta_{t,i} = \sum_r h'(v_{t+1,r}) \delta_{t+1,r} W_{t+1,r,i} \tag{31}$$

  In vector form we would have:

$$\boldsymbol{\delta}_t = \left(h'(\boldsymbol{v}_{t+1}) \circ \boldsymbol{\delta}_{t+1}^T\right) W_{t+1} \tag{32}$$

  The base of the recursion is simply (note it is a scalar because there is only one z_L):

$$\delta_L = \ell'(y, z_L(\boldsymbol{x}, W)) \tag{33}$$

- Calculate the desired gradients via:

$$\frac{\partial \ell}{\partial W_{t,i,j}} = \delta_{t,i} h'(v_{t,i}) z_{t-1,j} \tag{34}$$

  Or in vector form:

$$\frac{\partial \ell}{\partial W_t} = (\boldsymbol{\delta}_t \circ h'(\boldsymbol{v}_t)) \boldsymbol{z}_{t-1}^T \tag{35}$$

You can now see why the algorithm is called backpropagation. After the forward pass, it starts from the last layer, calculates the gradient of the loss wrt z_L and then begins to propagate the errors backwards, so that each weight matrix W_t gets a signal as to how it should change to decrease the error.

References

[1] E. Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.