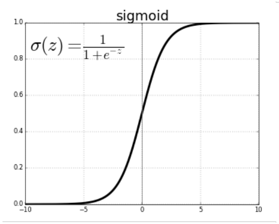
# Lec 24: Stochastic Gradient Descent

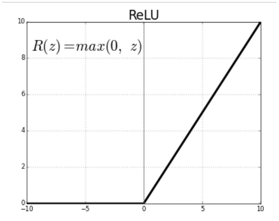
Ailin Zhang

## Agenda

- Recap: Back-propagation
- Multi-class classification
- Implicit gradient regularization
- Stochastic gradient descent
- Momentum, Adagrad, RMSprop, Adam

## Sigmoid vs RELU





The sigmoid function saturates at the two ends, causing the gradient to vanish. The ReLU does not saturate for big positive input.

$$y_i \sim \text{Bernoulli}(p_i),$$

$$p_i = \text{sigmoid}(h_i^\top \beta) = \text{sigmoid}\left(\sum_{k=1}^{d} \beta_k h_{ik}\right),$$

$$h_{ik} = \max(x_i^\top \alpha_k, 0) = \max\left(\sum_{j=1}^{p} \alpha_{kj} x_{ij}, 0\right).$$

## Back propagation with RELU

Q: How do you modify $\frac{\partial l}{\partial \beta}$ and $\frac{\partial l}{\partial \alpha_k}$ for RELU?

## Back propagation with RELU

Q: How do you modify $\frac{\partial l}{\partial \beta}$ and $\frac{\partial l}{\partial \alpha_k}$ for RELU?

For ReLU $\max(0, a)$, we should replace $h_{ik}(1 - h_{ik})$ in the back-propagation by $1(h_{ik} > 0)$, which is a binary detector.

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{n} (y_i - p_i) h_i$$

$$\frac{\partial l}{\partial \alpha_k} = \frac{\partial l}{\partial h_k} \frac{\partial h_k}{\partial \alpha_k} = \sum_{i=1}^{n} (y_i - p_i) \beta_k 1(h_{ik} > 0) X_i$$

The ReLU function does not saturate for big positive input, which indicates the existence of a certain patten. This help avoids the vanishing gradient problem.

## Multi-layer back-propagation

We may write the forward pass as

$$\begin{tikzcd}
x \arrow[r] & h_1 \arrow[r] & \dots \arrow[r] & h_{l-1} \arrow[r] & h_l & \dots \arrow[r] & L. \\
 & W_1 \arrow[u] & \dots & W_{l-1} \arrow[u] & W_l \arrow[u] & &
\end{tikzcd}$$

The back-propagation pass is

$$\begin{tikzcd}
x & h_1 \arrow[l] \arrow[d] & \dots \arrow[l] & h_{l-1} \arrow[l] \arrow[d] & h_l \arrow[l] \arrow[d] & \dots & L. \arrow[l] \\
 & W_1 & \dots & W_{l-1} & W_l & &
\end{tikzcd}$$

This is a process of assigning blame. If the loss function $L$ finds something wrong, he will blame $h_L$ and $W_{L+1}$. This layer is usually the soft-max layer. $h_L$ will then blame $h_{L-1}$ and $W_L$, and so on. This process follows the chain rule.

## Multi-layer back-propagation

$$
\begin{aligned}
\frac{\partial L}{\partial h_{l-1}^{\top}} &= \sum_{k=1}^{d} \frac{\partial L}{\partial h_{l,k}} \frac{\partial h_{l,k}}{\partial s_{l,k}} \frac{\partial s_{l,k}}{\partial h_{l-1}^{\top}} \\
&= \sum_{k=1}^{d} \frac{\partial L}{\partial h_{l,k}} f_l'(s_{l,k}) W_{l,k} \\
&= \frac{\partial L}{\partial h_l^{\top}} f_l' W_l,
\end{aligned}
$$

where $W_{l,k}$ is the $k$-th row of $W_l$, and $f_l' = \mathrm{diag}(f_l'(s_{l,k}), k = 1, ..., d)$.

## Multi-layer back-propagation

$$\begin{aligned}\frac{\partial L}{\partial W_{l,k}} &= \frac{\partial L}{\partial h_{l,k}}\frac{\partial h_{l,k}}{\partial s_{l,k}}\frac{\partial s_{l,k}}{\partial W_{l,k}} \\ &= \frac{\partial L}{\partial h_{l,k}} f_l'(s_{l,k}) h_{l-1}^\top,\end{aligned}$$

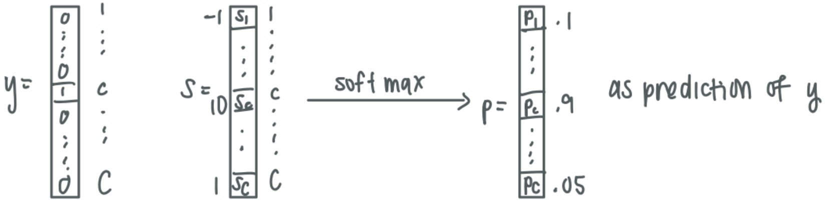
thus,

$$\frac{\partial L}{\partial W_l} = f_l' \frac{\partial L}{\partial h_l} h_{l-1}^\top.$$

Similarly,

$$\frac{\partial L}{\partial b_l} = f_l' \frac{\partial L}{\partial h_l}.$$

## Multi-class classification: softmax function



1. Exponentiate every element of the output layer and sum the results
2. Take each element of the output layer, exponentiate it and divide by the sum obtained in step 1

$$\Pr(y = k) = \frac{\exp(s_k)}{\sum_{k'} \exp(s_{k'})}.$$

## Multi-class classification: gradient descent

## Implicit gradient regularization

For simplicity, assume $x$ is one-dimensional. $h_k = \max(0, a_k x + b_k)$, and $s = f(x) = \sum_{k=1}^{d} \beta_k h_k$.

Suppose we initialize at $(a_k, b_k) \sim p_0(a, b)$ independently for $k = 1, ..., d$ and freeze them, and we only learn $\beta_k$ by gradient descent, starting from $\beta_k^{(0)} = 0$. The loss function is

$$L(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i)^\top \beta)^2.$$

- If $d \leq n$, there is a single minimum, and gradient descent converges to this minimum.
- When $d > n$, the model is over-parametrized, i.e., the number of parameters is greater than the number of training examples. Then there are infinitely many $\beta$ that gives $L(\beta) = 0$.

## Implicit gradient regularization

For simplicity, assume $x$ is one-dimensional. $h_k = \max(0, a_k x + b_k)$, and $s = f(x) = \sum_{k=1}^{d} \beta_k h_k$.

Suppose we initialize at $(a_k, b_k) \sim p_0(a, b)$ independently for $k = 1, ..., d$ and freeze them, and we only learn $\beta_k$ by gradient descent, starting from $\beta_k^{(0)} = 0$. The loss function is

$$L(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i)^\top \beta)^2.$$

- If $d \leq n$, there is a single minimum, and gradient descent converges to this minimum.
- When $d > n$, the model is over-parametrized, i.e., the number of parameters is greater than the number of training examples. Then there are infinitely many $\beta$ that gives $L(\beta) = 0$.

## Implicit gradient regularization

- Gradient descent will converge to one such $\beta$ with minimal $\ell_2$ norm.

Proof:

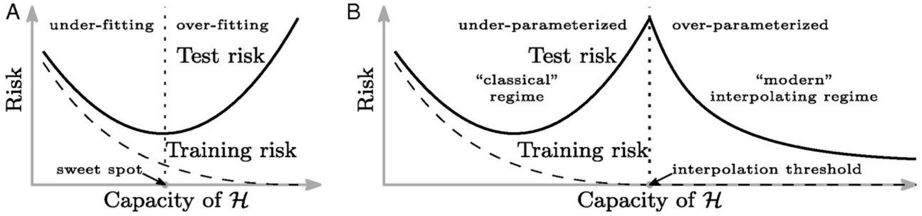
Gradient descent is

$$\Delta\beta \propto \frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i)^\top \beta)h(x_i).$$

- It will converge to a $\hat{\beta} = \sum_i c_i h(x_i)$ that satisfies $y_i = h(x_i)^\top \hat{\beta}$ for all $i$.
- Suppose there is another $\tilde{\beta}$ that satisfies $y_i = h(x_i)^\top \tilde{\beta}$ for all $i$.
- Let $\tilde{\beta} = \hat{\beta} + \Delta$, then $\Delta \perp h(x_i)$ for all $i$, i.e., $\Delta \perp \hat{\beta}$.
- Thus $\|\tilde{\beta}\|^2 = \|\hat{\beta}\|^2 + \|\Delta\|^2$. Thus gradient descent will converge to $\hat{\beta}$ with minimal $\ell_2$ norm.

## Implicit gradient regularization

- When $p \leq n$, there is only one minimum of $L(\beta)$, and there is no regularization, thus the overfitting becomes severe as $p$ approaches $n$.
- But as $p > n$, there are infinitely many $\beta$ with $L(\beta) = 0$, and gradient descent provides implicit regularization, so that testing error starts to decreases again.



Reference: "Reconciling modern machine-learning practice and the classical bias–variance trade-off"

## Stochastic gradient descent

Gradient descent:

$$\theta_{t+1} = \theta_t - \eta L'(\theta_t),$$

where $\eta$ is the step size or learning rate, and $L'(\theta)$ is the gradient.

- It may be time consuming because we need to compute $L'(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i'(\theta)$ by summing over all the examples.
- Stochastic gradient descent: at each step, we randomly select $i$ from $\{1, 2, ..., n\}$. Then we update $\theta$ by

  $$\theta_{t+1} = \theta_t - \eta_t L_i'(\theta_t),$$

  where $\eta_t$ is the step size or learning rate, and $L_i'(\theta_t)$ is the gradient only for the $i$-th example.
- Because $i$ is randomly selected, the above algorithm is called the stochastic gradient descent algorithm.

## Mini-batch gradient descent

Instead of randomly selecting a single example, we may randomly select a mini-batch, and replace $L_i'(\theta_t)$ by the average of this mini-batch.

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{m} \sum_{i=1}^{m} L_i'(\theta)$$

Question: Any constraint on $\eta_t$?

## Mini-batch gradient descent

Instead of randomly selecting a single example, we may randomly select a mini-batch, and replace $L_i'(\theta_t)$ by the average of this mini-batch.

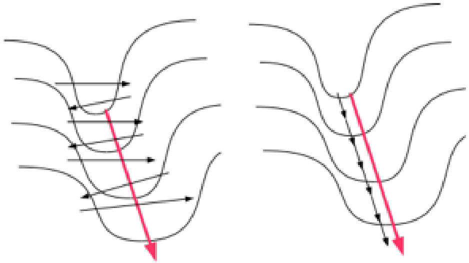
$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{m} \sum_{i=1}^{m} L_i'(\theta)$$

Question: Any constraint on $\eta_t$?

1. $\sum_{t=1}^{\infty} \eta_t = \infty$. Ensures that the algorithm can go the distance toward the minimum.
2. $\sum_{t=1}^{\infty} \eta_t^2 < \infty$. Ensures that the algorithm will not run away from the local minimum once it arrives.

## Gradient descent may not be the best direction

The gradient descent algorithm goes downhills in the steepest direction in each step. However, the steepest direction may not be the best direction.



The red arrows are the preferred direction, which is the direction of momentum. It is better to move along the direction of the momentum, as illustrated by the right figure. We want to accumulate the momentum, and let it guide the descent.

## Momentum

$$v_t = \gamma v_{t-1} + \eta_t g_t,$$
$$\theta_t = \theta_{t-1} - v_t.$$

- $g_t$ is the average gradient computed from the current mini-batch.
- $v_t$ is the momentum or velocity.
- $\gamma$ is usually set at .9, for accumulating the momentum.

## Adagrad

Adagrad modifies the gradient descent algorithm in another direction. The magnitudes of the components of $g_t$ may be very uneven, and we need to be adaptive to that. The Adagrad let

$$\begin{aligned} G_t &= G_{t-1} + g_t^2, \\ \theta_{t+1} &= \theta_t - \eta_t \frac{g_t}{\sqrt{G_t + \epsilon}}, \end{aligned}$$

- $\epsilon$ is a small number to avoid dividing by 0.
- In the above formula, $g_t^2$ and $g_t/\sqrt{G_t + \epsilon}$ denote component-wise square and division.

## RMSprop

In Adagrad, $G_t$ is the sum over all the time steps. It is better to sum over the recent time steps. RMSprop use the following scheme:

$$G_t \quad = \quad \beta G_{t-1} + (1-\beta) g_t^2,$$

- $\beta$ can be set at .9.
- It can be shown that

$$G_t = (1-\beta)(\beta^{t-1} g_1^2 + \beta^{t-2} g_2^2 + ... + \beta g_{t-1}^2 + g_t^2),$$

  which is a sum over time with decaying weights.

## Adam optimizer

Adam optimizer combines the idea of RMSprop and the idea of momentum.

$$
\begin{aligned}
&v_t = \gamma v_{t-1} + (1-\gamma) g_t, \\
&G_t = \beta G_{t-1} + (1-\beta) g_t^2, \\
&v_t \leftarrow v_t/(1-\gamma), \\
&G_t \leftarrow G_t/(1-\beta), \\
&\theta_{t+1} = \theta_t - \eta_t \frac{v_t}{\sqrt{G_t + \epsilon}}.
\end{aligned}
$$