Lecture 2. What to do if my network fails to train

Young, Shen

August 11, 2024

1 General Guide

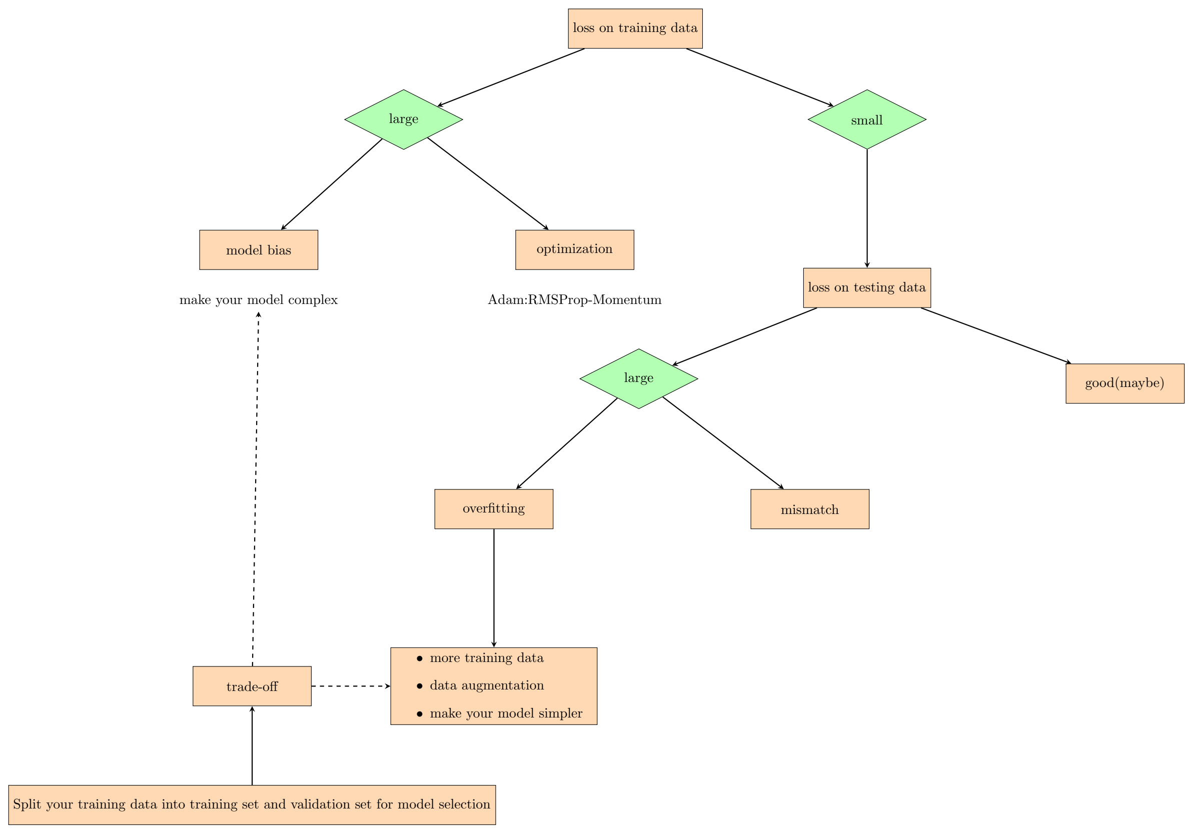


Fig. 1. General Guide when train a model

2 Local Minima and Saddle Point

When the gradient of loss function is 0 e.g. $\nabla L(\theta_0) = 0$, it's probably not local minima. In fact, there are 3 conditions where the gradient is 0:

- local maxima
- local minima
- saddle point

How can we distinguish between them mathematically? Considering the Taylor Series expansion near $\theta = \theta_0$ of loss function:

$$L(\theta) = L(\theta_0) + (\theta - \theta_0)^\top \nabla_\theta L(\theta_0) + (\theta - \theta_0)^\top H(\theta - \theta_0) + ...$$

Where H is called **Hessian Matrix**:

$$H_{ij} = \frac{\partial^2 L}{\partial\theta_i \partial\theta_j}|_{\theta=\theta_0}$$

For critical points(points where the gradient is 0), the second term is zero, so we only need to care about the third term. It is obvious that Hessian Matrix is symmetry, and $\theta^\top H\theta$ is a quadratic form. We define H is **positive definite** when: for every $x \neq 0$:

$$x^\top Hx > 0$$

Most commonly used way to examine whether a matrix is positive definite is by its eigenvalue. **Theorem**. H is positive definite if and only if all of its eigenvalues are positive.

Obviously, if H is positive definite, the critical point is local minima, since for every θ near θ_0, the quadratic form is positive, then

$$L(\theta) > L(\theta_0), for\, all\, \theta\, near\, \theta_0$$

More generally, if H negative definite, then the critical point is local maxima. If some of the eigenvalue are positive, some are negative, then it is the saddle point.

Let's deal with saddle point. Considering one of the negative eigenvalue λ and the corresponding eigenvector $\mathbf{u}$, we have:

$$\mathbf{u}^\top H\mathbf{u} = \lambda\mathbf{u}^\top\mathbf{u} = \lambda||\mathbf{u}||^2 < 0$$

If we let $\theta - \theta_0 = \mathbf{u}$, then the quadratic form will be negative, meaning that $L(\theta) < L(\theta_0)$. This is a way to escape from saddle point:

$$\theta = \theta_0 + \alpha\mathbf{u}$$

In fact, most of the times when the gradient is close to 0 are caused by saddle point.

3 Batch and Momentum

3.1 Small Batch and Big Batch

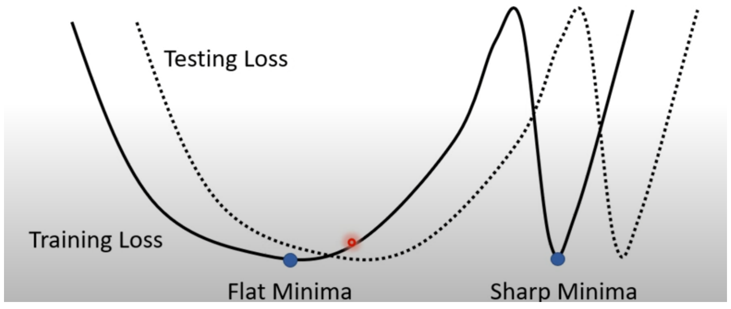


Fig. 2. flat-sharp-minima

Testing set and training set may be different. As a result, their loss functions are different. Supposing moving the loss function of training set gets the testing loss. It can be seen from 2 that the sharp minima performs poorly on test set, while the flat minima performs better. Researches show that when using big batch, you tend to fall in sharp minima, while small batch brings you to flat minima. Moreover, the time it consumes using big and small batch when training a model is almost the same. In all, **small batch is preferred**.

3.2 Escape from Critical Point: Momentum and Adaptive Learning Rate

3.2.1 Vanilla Gradient Descent

$$\theta^{i+1} = \theta^i - \nabla L(\theta_i)$$

Drawbacks: Stuck in critical point, even if it's not local minima.

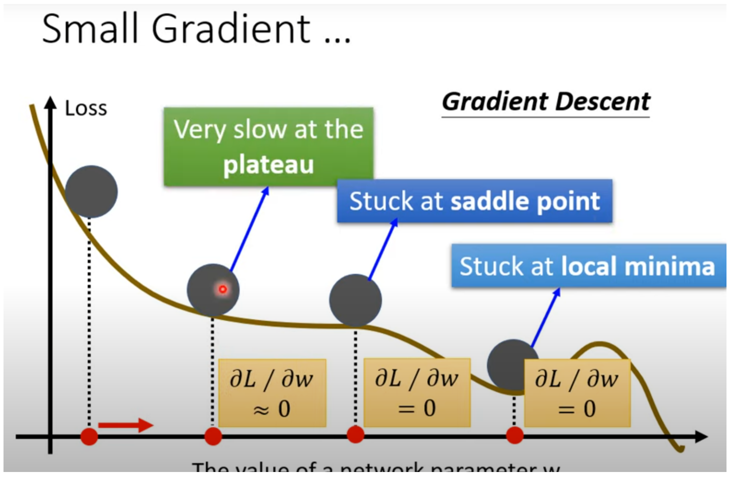


Fig. 3. Vanilla Gradient Descent sticks when gradient is small

3.2.2 Momentum

In real world, when a little ball rolls down along a curve, its speed is not only related to the gradient of its current position, but also related to the gradient of the curve it has maved through. To avoid the optimizer sticking at somewhere not the minima, we introduce the concept of **Momentum**.

The movement now consists not only gradient, but also the movement of last step, namely:

$$\mathbf{m^{i+1}} = \lambda \mathbf{m^i} - \alpha \mathbf{g^i}$$

with the updating rule being:

$$\theta^{i+1} = \theta^i + \mathbf{m^{i+1}}$$

where $\mathbf{m^{i+1}}$ is the movement of the $(i+1)$th update, g^i is the gradient of loss function calculated at θ^I.

Note: we define $\mathbf{m^0} = 0$. Using the recursion, we can easily get:

$$\begin{aligned} \mathbf{m^0} &= 0 \\ \mathbf{m^1} &= -\alpha \mathbf{g^0} \\ \mathbf{m^2} &= -\lambda\alpha \mathbf{g^0} - \alpha \mathbf{g^1} \\ &\dots \end{aligned}$$

We can easily find out that $\mathbf{m^i}$ is linear combination of $\mathbf{g^0}, \mathbf{g^1}, \mathbf{g^2}, ..., \mathbf{g^{i-2}}, \mathbf{g^{i-1}}$

With momentum, optimizer can sometimes even get across small hills to find places lower than current local minima.

3.2.3 Adaptive Learning Rate

Sometimes along one parameter the loss decreases quickly, while along some other parameters the loss decreases very slow. Even for one certain parameter, the loss function can sometimes be steep and sometimes flat. Under these circumstance, it is not wise to use all the same learning rate for all parameters for all the time. We need to customize learning rate for each parameter and adapt learning rate in real time. The updating rule now rewrites as :

$$\theta_j^{i+1} = \theta_j^i - \frac{\alpha}{\sigma_j^i} g_j^i$$

The subscripts stands for the jth parameter, while the superscript stands for ith update. Note that σ is related to both parameter and update.

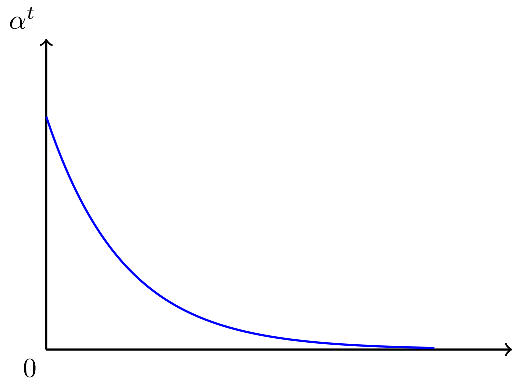
$$\begin{aligned}
\sigma_j^0 &= \sqrt{(g_j^0)^2} \\
\sigma_j^1 &= \sqrt{\frac{1}{2}[(g_j^0)^2 + (g_j^1)^2]} \\
\sigma_j^2 &= \sqrt{\frac{1}{3}[(g_j^0)^2 + (g_j^1)^2 + (g_j^2)^2]} \\
&...
\end{aligned}$$

σ is defined as **root mean square (RMS)** of all the (i+1) gradients before. In practise, **RMSProp** is more commonly used. With the same updating rule as before, σ is defined as:

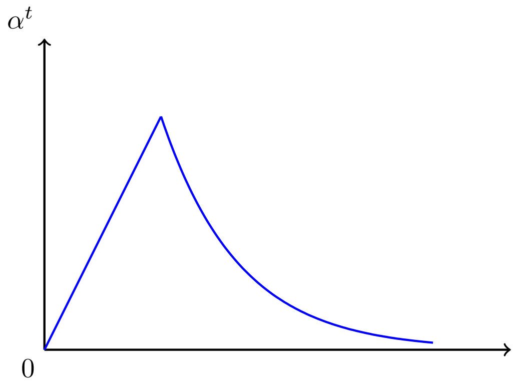
$$\begin{aligned}
\sigma_j^0 &= \sqrt{(g_j^0)^2} \\
\sigma_j^1 &= \sqrt{k(\sigma_j^0)^2 + (1-k)(g_j^1)^2} \\
\sigma_j^2 &= \sqrt{k(\sigma_j^1)^2 + (1-k)(g_j^2)^2} \\
&...
\end{aligned}$$

Where $0 < k < 1$. k is a hyperparameter here. It reflects how much you would the gradient at this point weigh in σ.

This method makes sense. Imagine the gradient before is big, indicating that it is steep along this parameter, so we want σ to be big so that the step is not that big, avoiding overshooting. If the gradient before is small, meaning it is flat along this parameter, the σ is small, the step is big, avoiding optimization terminating.



(a) Learning Rate Decay



(b) Warm Up

Fig. 4. Learning Rate Scheduling

There is also another technique called **Learning Rate Scheduling**4:

$$\theta_j^{i+1} = \theta_j^i - \frac{\alpha^i}{\sigma_j^i} g_j^i$$

Where the learning rate itself changes along with optimization i. Commonly used are **Learning Rate Decay** and **Warm Up**.

Nowadays we have various kinds of improvements to the gradient descent. One of the most commonly used is the comnine of momentum and learning rate scheduling:

$$\theta_j^{i+1} = \theta_j^i - \frac{\alpha^i}{\sigma_j^i} m_j^i$$

with m_j^i being the ith movement of the jth parameter. This is called **Adam** (Adagradient-Momentum).