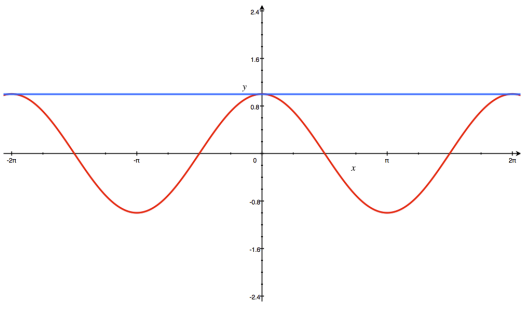
Ch.7 - Gradient Descent

Created	@January 16, 2024 2:53 PM
Book	Deep Learning: Foundations and Concepts

Basics

Quadratic Approximation



One Variate Functions

For a one-variate function $f(x)$, the best linear approximation around $x = a$ is given as follows, which is named '**Taylor Expansion**':

$$f(x) \approx \sum_{n=0}^{N} \frac{1}{n!} f'^{(n)}(x-a)^n$$

The larger `n`, the more accurate the approximation becomes.

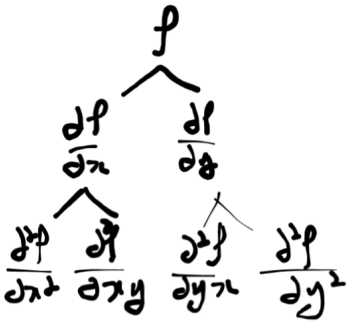
Multivariate Functions

The general form of Taylor Expansion, for multidimensional input $\mathbf{x}$, is written as quadratic approximation. It approximates the function $f(\mathbf{x})$ near $\mathbf{x} = a$ by `n=2` (second order)

$$f(\mathbf{x}) \approx f(a) + \nabla f(a)(\mathbf{x} - a) + \frac{1}{2}(\mathbf{x} - a)^T Hf(a)(x - a)$$

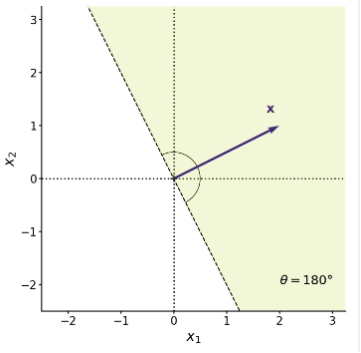
Where ∇f is gradient, Hf is Hessian matrix which contains all the second order partial derivatives. For instance of two dimensional input (x, y):

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix}, Hf = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial yx} \\ \\ \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$



Unlike one-variable input, the number of derivative terms of multivariate functions grows exponentially in n-ary tree structure by its depth. The form of Hessian matrix follows this pattern.

Positive Definite



Any Matrix

Let A be any square matrix of $n \times n$ size, then it is "positive definite" in the sense that:

$$\forall x \in R_n, x \neq 0 \implies x^T Ax = x \cdot (Ax) > 0$$

This implies that the linear transformation of `x` stays on the same side of the original `x`, since dot product($||A||\ ||B||\ cos\theta$) of two vectors is positive when the angle between them is $-90 < \theta < 90$.

Symmetric Matrix

📖 **Spectral Theorem**
Hermitian Matrix
A is always diagonalizable and has an orthogonal matrix U with n linearly independent eigenvectors as its column vectors (basis). $A = U\Sigma U^T$

Given that a symmetric matrix $A_{n\times n}$ is full rank, thus having n number of eigenvalues, diagonalization of A is then guaranteed. By the spectral theorem, there is an orthonormal matrix$U$
$U = [u_1\ u_2 \cdots\ u_n]$ and $u_i \perp u_j$, $A = U\Sigma U^T$.
If $y = U^T x \neq 0$ ($x \neq 0 \because \text{rank-nullity theorem}$):

$$x^T Ax = x^T(U\Sigma U^T)x = (U^T x)^T \Sigma (U^T x) = y^T \Sigma y = \sum_{i=1}^{n} \lambda_i y_i^2 > 0$$

Therefore, if all eigenvalues of a symmetric matrix $A_{n\times n}$ are positive, then A is positive definite and vice versa.

❓ Shouldn't it be **full rank** in order to be diagonalizable ?
Why all symmetric matrices are diagonalizable ?

Second Partial Derivative Test

One Variable Function

If at a point $x = a$ where the slope $f'(x)$ is zero, the function is local minima when $f''(a) > 0$, as Taylor Expansion makes it clear:

$$f(a + \Delta x) \approx f(a) + \frac{1}{2} f''(a)\Delta x^2$$

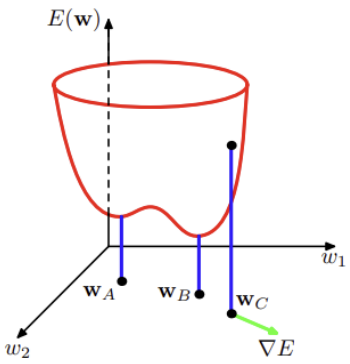
Multivariate Functions

Looking into Taylor Expansion near the critical point $\nabla f = 0$ where $f(a + \Delta x) \approx f(a) + \frac{1}{2}\Delta x^T Hf(a)\Delta x$, it becomes obvious that Hessian Matrix $Hf(a)$ should be positive definite in order to be local minima. By the spectral theorem that specifies relationship between positive definite and eigenvalues of a symmetric matrix:

- If $Hf(a)$ has all positive eigenvalues, $x = a$ is **local minima**
- If $Hf(a)$ has all negative eigenvalues, $x = a$ is **local maxima**
- If $Hf(a)$ has eigenvalues of mixed signs, $x = a$ is a **saddle point**

Error Surfaces

🎯 Find a local minima of the error function $E(w)$



Our goal is to find whether it's **local minima** or not near some critical point $x = a$. From the quadratic approximation using Taylor Expansion, a critical point where $\nabla E(a) = 0$ is given as:

$$E(a + \Delta x) \approx E(a) + \frac{1}{2}\Delta x^T H E(a) \Delta x$$

where $HE(a) = \nabla\nabla E_{|x=a}$ is the Hessian Matrix at the point a. By the spectral theorem, Hessian Matrix is diagonalizable with a unitary matrix U as $HE(a) = U^T \Sigma U$, thus if we define arbitrary $y = U\Delta x$:

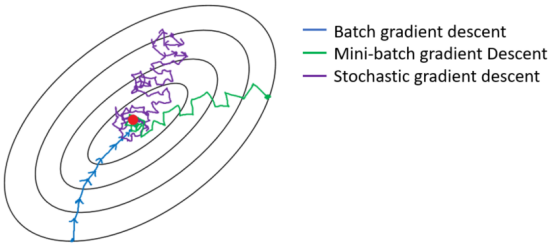
$$E(a + \Delta x) \approx E(a) + \frac{1}{2}\Delta x^T (U^T \Sigma U) \Delta x = E(a) + \frac{1}{2}\sum_{i=1}^{n} \lambda_i y_i^2$$

Therefore, we found a local minima near a critical point a if and only if $HE(a)$ is **positive definite**.

Gradient Descent Optimization

In evaluating error function to get the optimal set of w, it's often sufficient to use gradient information ∇E without the need of Hessian H. It will reduce the time complexity of $O(W^3)$ to $O(W^2)$ since the length of ∇E is only W, the number of parameters.

Batch vs Stochastic



✅ Batch Gradient Descent is slow in a large dataset

⇒

`n` samples * `w` parameters calculations for each step

✅ Stochastic Gradient Descent takes only 1 sample per step

⇒

`1` sample * `w` parameters calculations

✅ But SGD's gradient is merely an estimate, hence noisy

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}(w^T x_i - y_i)x_i \quad \cdots \text{ where m=}$$

✅ SGD with **mini-batch** would complement BGD and SGD

⇒ mini-batch would be taken randomly from the population

Parameter Initialization

> ⚠️ *Initializing all the weights with zeros leads the neurons to learn the same features during training.*

If the weights are all initialized to zero:

✅ Linear transformations, activation functions are the same

- $z_i = w^T x = 0,\ z_1 = z_2 = z_3$
- $a_i = \sigma(z_i),\ a_1 = a_2 = a_3$

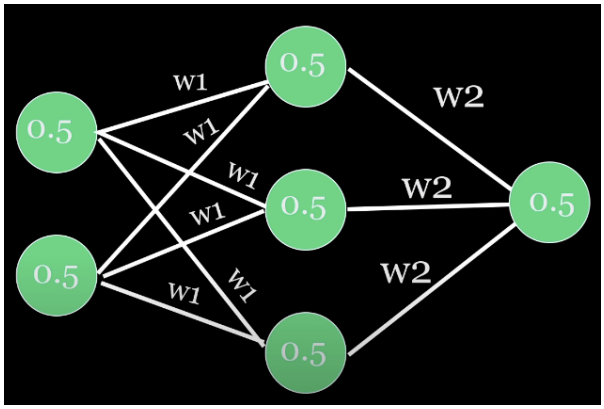
✅ Output layer's weights are the same

- $v_1 = v_2 = v_3$
- $y_{out} = v_1 a_1 + v_2 a_2 + v_3 a_3 = 3va$

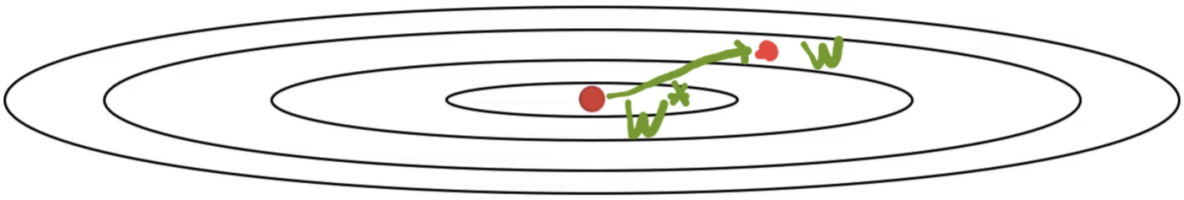
✅ Gradients with respect to each weight are the same

- $\partial E_{v_1} = \partial E_{v_2} = \partial E_{v_3}$

Thus all the units will learn the same features. This problem can be addressed by initializing parameters randomly from some distribution, e.g. $N(0, \epsilon^2)$ or Uniform distribution.



Convergence



Learning Rate η

Let $w \in R^n$ is a weights point near a local minima w^*, and there is a complete set of orthonormal basis vectors $U = [u_1\ u_2\ \cdots\ u_n]$ from $H = U\Sigma U^T$, where $HU = U\Sigma$. There exists any $\alpha \in R^n$ that:

$$w - w^* = U\alpha = \alpha_1 u_1 + \alpha_2 u_u + \cdots + a_n u_n$$

This is marginal movement vector from local minima w^* expressed in $[u_1\ u_2\ \cdots\ u_n]$ system. In order for gradient descent, $\nabla E(w)$ is given as:

$$\begin{aligned} E(w) &= E(w*) + \frac{1}{2}(w - w^*)^T H_{|w^*}(w - w^*) \\ &= E(w^*) + \frac{1}{2}(U\alpha)^T H_{|w*}(U\alpha) \\ &= E(w^*) + \frac{1}{2}\alpha^T U^T U\Sigma\alpha\ (\because HU = U\Sigma) \\ &= E(w^*) + \frac{1}{2}\alpha^T \Sigma\alpha\ (\because U^T = U^{-1}) \\ &= E(w^*) + \frac{1}{2}\sum_i \lambda_i a_i^2 \end{aligned}$$

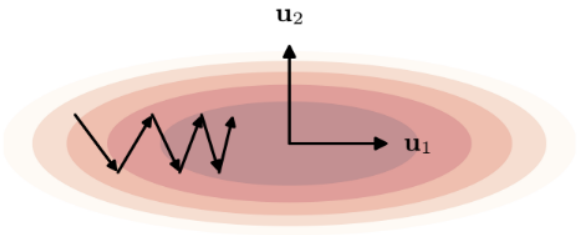
$$\implies \frac{\partial E(w)}{\partial a_i} = \lambda_i \alpha_i$$

Combining with the gradient formula $w^{new} = w^{old} - \eta\nabla_w E(w)$:

$$\begin{aligned} w^{new} - w^{old} &= U^T(a^{new} - a^{old}) = -\eta\sum_i \lambda_i \alpha_i^{old} u_i \\ \implies a_i^{new} &= (1 - \eta\lambda_i)a_i^{old} \end{aligned}$$

Note that a_i can be interpreted as the distance of w in the direction of eigenvector u_i. After successive terns of gradient descent:

$$a_i^T = (1 - \eta\lambda_i)^T a_i^0$$
$$\therefore T \to \infty,\ a_i^T \to 0 \ (\text{if } |1 - \eta\lambda_i| < 1)$$



Therefore, Gradient Descent **converges to the local minima** w^* as long as $\eta < \frac{2}{\lambda_{max}}$.

> ⚠️ **The trap of λ_{min}**
>
> However, the rate of convergence is governed by the smallest eigenvalue. Even though η is set to the maximum, the convergence to the direction u_{min} will be $(1 - \frac{2}{\lambda_{max}}\lambda_{min})$. So it's important to ensure that **the ratio $\frac{\lambda_{min}}{\lambda_{max}}$ is large enough** for fast convergence.

❓ What's the relationship between curvature and eigenvalues of H ? $a_i^{new} = (1 - \eta\lambda_i)a_i^{old}$?

Momentum $\mu\Delta w$

The problem of widely differing eigenvalues can be dealt with introducing momentum, which applies a certain degree of the precious Δw^{old} to the current Δw^{new}.

$$\Delta w^{new} = -\eta\nabla E(w^{old}) + \mu\Delta w^{(}$$
$$w^{new} = w^{old} - \eta\nabla E(w^{old}) + \mu$$

▼ In a region of **low curvature** (one-way): faster effective learning rate

Assume Δw is always positive and ∇E is constant:

$$\begin{aligned}
\Delta w^0 &= -\eta\nabla E \\
\Delta w^1 &= -\eta\nabla E + \mu\Delta w^0 = -\eta\nabla E(1 + \mu) \\
&\cdots \\
\Delta w &= -\eta\nabla E\{1 + \mu + \mu^2 + \cdots\} \\
&= -\frac{\eta}{1 - \mu}\nabla E
\end{aligned}$$

This indicates that the momentum term increases the learning rate from η to $\frac{\eta}{1-\mu}$

▼ In a region of **high curvature** (oscillate): slower effective learning rate

$$\begin{aligned}\Delta w &= \eta\nabla E\{1-\mu+\mu^2-\mu^3+\cdots\} \\ &= -\frac{\eta}{1+\mu}\nabla E\end{aligned}$$

> 💡 **Learning Rate Schedule**
> In practice, the best results are obtained by reducing the η from a larger value. If a parameter such as η needs to be found empirically, while monitoring the value over steps, it's said to be '**hyperparameter**'.

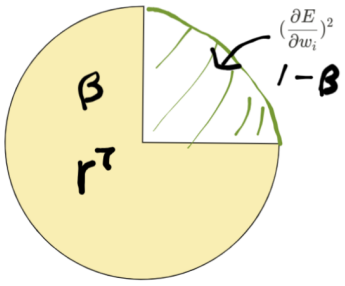
AdaGrad

Adaptive Gradient(AdaGrad) achieves learning rate optimization by accumulating the sum of squared derivatives $\frac{\partial E}{\partial w}^2$:

$$r_i^\tau = r_i^{\tau-1} + (\frac{\partial E}{\partial w_i})^2 \qquad\qquad w_i^\tau = w_i^{\tau-1} - \frac{\eta}{\sqrt{r_i^\tau}+\epsilon}(\frac{\partial E(w)}{\partial w_i})$$

where $\epsilon \approx e^{-6}$ in case that r^τ is close to zero. AdaGrad is only effective in short term of training as it accumulates exponentially, resulting in small learning rate to all parameters if the training takes long.

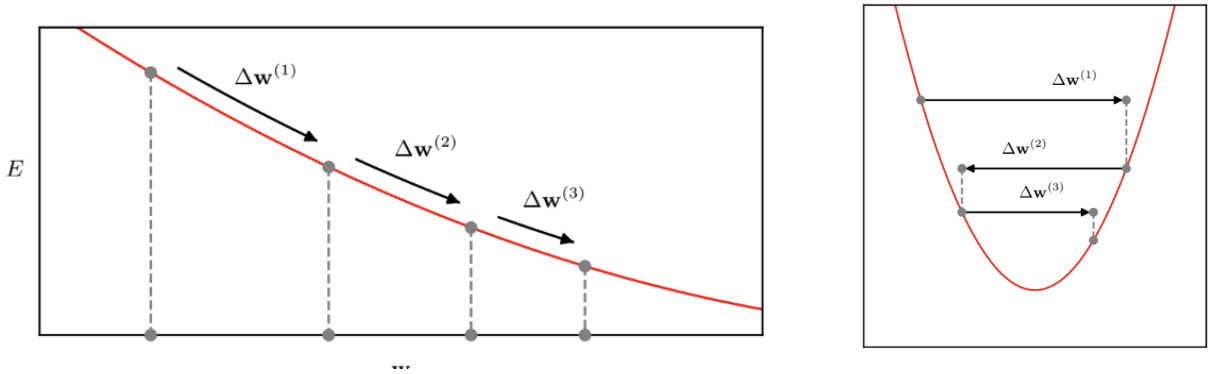
RMSProp



$$r_i^\tau = \beta r_i^{\tau-1} + (1-\beta)(\frac{\partial E}{\partial w_i})^2$$
$$w_i^\tau = w_i^{\tau-1} - \frac{\eta}{\sqrt{r_i^\tau}+\epsilon}(\frac{\partial E(w)}{\partial w_i})$$

Root Mean Squared Propagation(RMSProp) is to solve AdaGrad's continuous increase by applying decay rate to allow the sum to decrease. The sum r^τ evolves around the initial gradient over time with a small deviation.

Adam



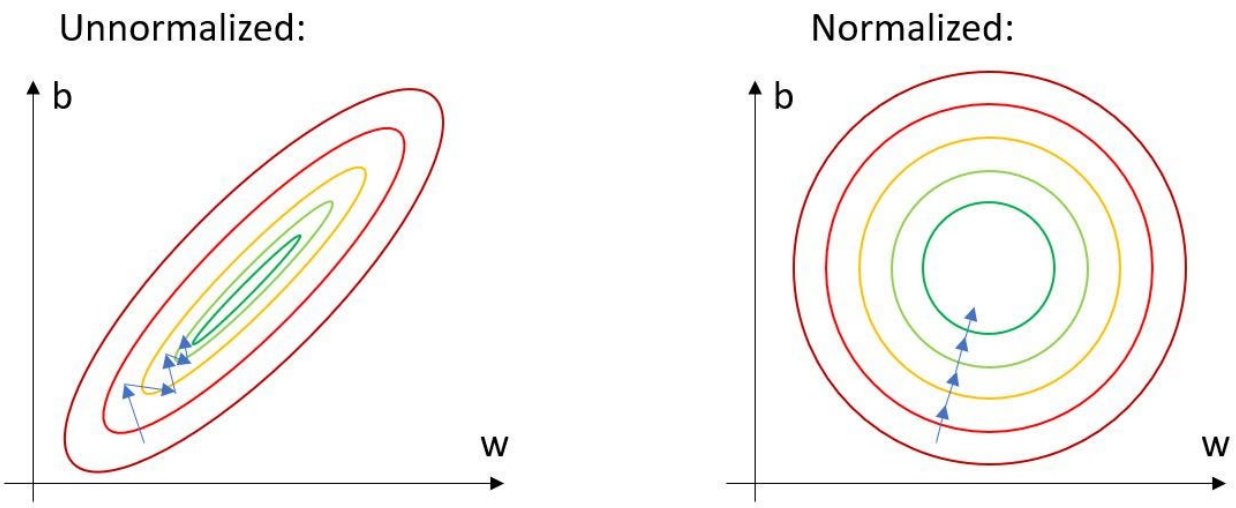
ADAM is a combination of momentum and RMSProp. It consists of three equations:

$$s_i^\tau = \beta_1 s_i^{\tau-1} + (1-\beta_1)(\frac{\partial E(w)}{\partial w_i})$$
$$r_i^\tau = \beta_2 r_i^{\tau-1} + (1-\beta_2)(\frac{\partial E}{\partial w_i})^2$$
$$w_i^\tau = w_i^{\tau-1} - \eta \frac{s_i^\tau}{\sqrt{r_i^\tau} + \epsilon}$$

- s_i^τ: the first momentum that stores **gradient**
- r_i^τ: the second momentum that stores **the sum of squared gradient**

Note that gradient is not directly used on updating rule. Instead, it uses the first equation as the momentum of gradient which has accumulated information of gradient over time. The second momentum acts as an amplifier that magnifies if ∂E_{w_i} > 1 or shrinks if ∂E_{w_i} < 1.

Normalization



< Input Normalization >

Different scales by features affect curvatures of its error function and the size of a gradient step as it can be seen from MSE function:

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j$$

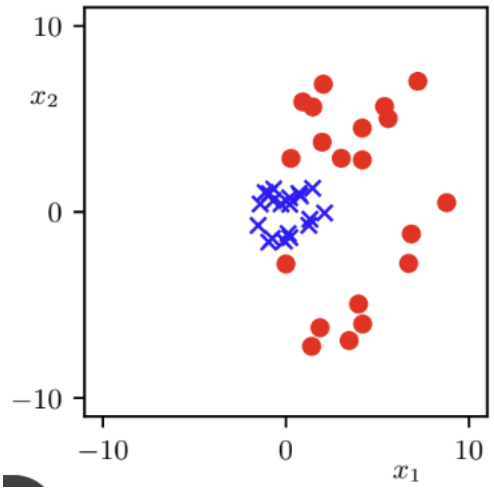
If we can flatten the curvature as wide as possible, it would be much beneficial to improving convergence and speeding up GD training.

Input Normalization

$$\mu_i = \frac{1}{N} \sum_{n=1}^{N} x_{ni}$$

$$\sigma_i^2 = \frac{1}{N} \sum_{n=1}^{N} (x_{ni} - \mu_i)^2,$$

$$\widetilde{x}_{ni} = \frac{x_{ni} - \mu_i}{\sigma_i}$$



✅ Normalizes input space to $N(0, 1)$

Mini-batch Normalization

$$\frac{\partial E}{\partial w_i} = \sum_m \cdots \sum_l \sum_j \frac{\partial z_m^{(1)}}{\partial w_i} \cdots \frac{\partial z_j^{(K)}}{\partial z_l^{(K-1)}} \frac{\partial E}{\partial z_j^{(K)}}$$

< Vanishing / Exploding gradients >

- Need to be repeated every batch per layer during training
- Inner normalization is need to avoid vanishing / exploding gradients

$$\mu_i = \frac{1}{K} \sum_{n=1}^{K} a_{ni}$$

$$\sigma_i^2 = \frac{1}{K} \sum_{n=1}^{K} (a_{ni} - \mu_i)^2$$

$$\widehat{a}_{ni} = \frac{a_{ni} - \mu_i}{\sqrt{\sigma_i^2 + \delta}}$$

< Mini Batch Normalization on pre activation >

- Normalizing pre-activation a_i or activation z_i $z_i = h(a_i)$
  ⇒ either is fine
- Normalize → Re-scale (deviation γ, mean β)
  ⇒
  $\hat{a} = \gamma a + \beta$
- How to deal with a new data for prediction ?
  ⇒ compute moving averages throughout the training phase

$$\overline{\mu}_i^{(\tau)} = \alpha \overline{\mu}_i^{(\tau-1)} + (1 - \alpha)\mu_i$$

$$\overline{\sigma}_i^{(\tau)} = \alpha \overline{\sigma}_i^{(\tau-1)} + (1 - \alpha)\sigma_i$$

< moving averages of each layer >