Regularization & Dropout

① Regularization

$\tilde{L}(\theta) = L(\theta) + \lambda \|\theta\|_2^2$

② Bagging: Bootstrap Aggregating (Dropout)

Reduce Variance

In FNN? : Dropout!

At each training step

each hidden unit or input

has probability p of being dropped out

implementation: $\hat{h}_i = \mu_{h_i} \cdot h_i$

add a binary mask variable: $\mu_{h_i} = \begin{cases} 0, (\text{probability } p) \\ 1, (p-1) \end{cases}$

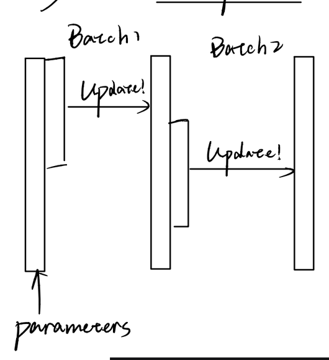
For each minibatch:

GD: $\mu = 1 \Rightarrow$ update weight; $\mu = 0 \Rightarrow$ no update

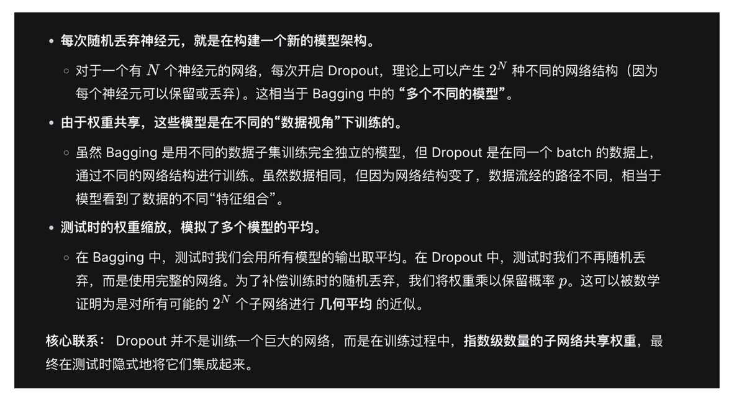
$\Rightarrow$ training is conducted in a part of network

Meanwhile: still only one network!

Dropout



Why "Dropout" is like "Bagging"



- 每次随机丢弃神经元，就是在构建一个新的模型架构。
  - 对于一个有 N 个神经元的网络，每次开启 Dropout，理论上可以产生 2^N 种不同的网络结构（因为每个神经元可以保留或丢弃）。这相当于 Bagging 中的“多个不同的模型”。
- 由于权重共享，这些模型是在不同的“数据视角”下训练的。
  - 虽然 Bagging 是用不同的数据子集训练完全独立的模型，但 Dropout 是在同一个 batch 的数据上，通过不同的网络结构进行训练。虽然数据相同，但因为网络结构变了，数据流经的路径不同，相当于模型看到了数据的不同“特征组合”。
- 测试时的权重缩放，模拟了多个模型的平均。
  - 在 Bagging 中，测试时我们会用所有模型的输出取平均。在 Dropout 中，测试时我们不再随机丢弃，而是使用完整的网络。为了补偿训练时的随机丢弃，我们将权重乘以保留概率 p。这可以被数学证明为是对所有可能的 2^N 个子网络进行 几何平均 的近似。

核心联系：Dropout 并不是训练一个巨大的网络，而是在训练过程中，指数级数量的子网络共享权重，最终在测试时隐式地将它们集成起来。

Optimized Algorithms

Task: $\hat{\theta} = \arg\min_{\theta} L(\theta)$ (always)

Steps: 1. init θ 2. dataset $\xrightarrow{\theta}$ output. 3. update θ — Then (back to 2.) till met criterion

Different Optimizer:

① Momentum:

$\bar{\theta}$ has speed $\bar{v}$ at each iteration,

1. current speed reduced, $\bar{v} \leftarrow \mu\bar{v}$
2. acceleration: $-\epsilon g$
3. $\bar{v} \leftarrow \mu\bar{v} - \epsilon \bar{g}$
4. $\theta \leftarrow \theta + \bar{v}$

Hence $\longrightarrow$ $V_t = -\eta(\nabla L_t + \mu \nabla L_{t-1} + \mu^2 \nabla L_{t-2} \cdots)$ (η: learning rate)

“既记录过去信息，又注重响应当前变化” (加权)

Momentum ① reduces the variation in overall gradient direction.

② speed up learning.

Algorithm with adaptive learning rate

"Parameter that have changed a lot should be allowed to change less in future"

- often changed $\longrightarrow$ maybe already close to optimized value
  - fixed rate $\to$ can not converge
  - adaptive rate $\to$ converge!
- seldom changed $\to$ if learning rate = rate(often changed) (i.e. fixed learning rate) then too slow! $\to$ needs to speed up.

Hadamard product / division

$\Rightarrow$ elementwise

$\odot$: product $\oslash$: division.

① AdaGrad:

① Initialize gradient accumulation variable $r = 0$

② While stop criterion not met do

1. Sample a minibatch of m samples from the training set $\{x^{(1)}, x^{(2)} \cdots x^{(n)}\}$ with corresponding targets $y^{(i)}$ ($x^{(1)} \to y^{(1)}$, $x^{(2)} \to y^{(2)}$, …, $x^{(n)} \to y^{(n)}$)
2. Compute Gradient: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$ Gradient
3. "$r \leftarrow r + g \odot g$" (Accumulate squared gradient)
4. $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$ (ϵ: "Learning rate"; $\sqrt{r}$: "Gradient" accumulation; δ: small constant, to make sure $\sqrt{r} + \delta \neq 0$)
5. $\theta \leftarrow \theta + \Delta\theta$

After t iterations: $\vec{r} = g_1^2 + g_2^2 + \cdots + g_t^2$

② RMSProp

$\bar{r} \leftarrow \rho \bar{r} + (1-\rho)\bar{g}^2$

After t iterations:

$\bar{r} = (1-\rho)(\rho^{t-1}\bar{g}_1^2 + \rho^{t-2}\bar{g}_2^2 + \cdots \rho \bar{g}_{t-1}^2 + \bar{g}_t^2)$ — "Exponentially Decaying weight" (Same → 既记录过去信息，又注重响应当前变化)

$\Delta\bar{\theta} = -\frac{\epsilon \bar{g}}{\delta + \sqrt{r}}$

AdaGrad: After t iterations: $\vec{r} = g_1^2 + g_2^2 + \cdots + g_t^2$ "No Decaying!" (Compare to)

③ Adam: Adaptive Moment

- RMSProp.
- Momentum — "Combination"
- Bias Correction

Key Procedure:

$s \leftarrow \rho_1 s + (1-\rho_1) g$ Momentum 1st Moment

$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$ RMSProp. 2st Moment.

Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$

Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

Where t is the count of iterations

$\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$

$\theta \leftarrow \theta + \Delta\theta$

Why doing This?

take "s" as example:

$s = (1-\rho_1)(\rho_1^t g_1 + \rho_1^{t-1} g_2 + \rho_1^{t-2} g_3 + \cdots + \rho_1 g_{t-1} + g_t)$

★ $1-\rho_1^t = (1-\rho_1)(\rho_1^t + \rho_1^{t-1} + \rho_2^{t-1} + \cdots + 1)$

↳ "Geometric Series! $\sum (\rho_1)^i = \frac{1-\rho_1^i}{1-\rho_1}$

$\therefore$ Bias Correction:

$\hat{s} = \frac{s}{1-\rho_1^t} = \frac{(1-\rho_1)(\rho_1^t g_1 + \rho_1^{t-1} g_2 + \rho_1^{t-2} g_3 + \cdots + \rho_1 g_{t-1} + g_t)}{(1-\rho_1)(\rho_1^t + \rho_1^{t-1} + \rho_2^{t-1} + \cdots + 1)}$

$= \frac{(\rho_1^t g_1 + \rho_1^{t-1} g_2 + \rho_1^{t-2} g_3 + \cdots + \rho_1 g_{t-1} + g_t)}{(\rho_1^t + \rho_1^{t-1} + \cdots + 1)}$

$= \frac{\rho_1^t}{z} g_1 + \frac{\rho_2^t}{z} g_2 + \cdots + \frac{1}{z} g_t$, where $z = \sum_{i=0}^{t} \rho_1^i$

Sum is 1! $\longrightarrow$ "归一化"

A Probability Distribution Over time!

↓ "Moment"

$E(x) = \sum_{i=1}^{k} x_i \cdot P(x_i)$

$s = \sum_{i=1}^{k} g_i \cdot P(g_i)$

Same!

Through Momentum & RMSProp,

We have $E(x)$ and $E(x^2)$

Then $Var(x) = E(x^2) - [E(x)]^2$!

- Moment / RMSProp: Create the distribution (with bias) "既记录过去信息，又注重响应当前变化"
- Bias correction: 归一化, Avoid:
  - 1st moment too small $\to$ slow update at beginning
  - 2st moment too small $\to$ Too large learning rate!

"$\Delta\theta \leftarrow -\frac{\epsilon \hat{s}}{\sqrt{\hat{r}} + \delta}$"

Weight Decay & L_2 regularization

L_2 regularization: $Loss = Loss_{original} + \frac{\lambda}{2}\|\theta\|^2$ (weight penalty)

Weight penalty $\longrightarrow$ smaller model capacity $\to$ avoid overfitting

Weight update:

$\nabla_\theta L = \nabla_\theta L + \lambda\theta$

$\theta_{t+1} = \theta_t - \eta(\nabla_\theta L + \lambda\theta_t)$

$= (1-\eta\lambda)\theta_t - \eta\nabla_\theta L$

Weight decay:

$\theta_{t+1} = (1-\eta\lambda)\theta_t - \eta\nabla_\theta L$

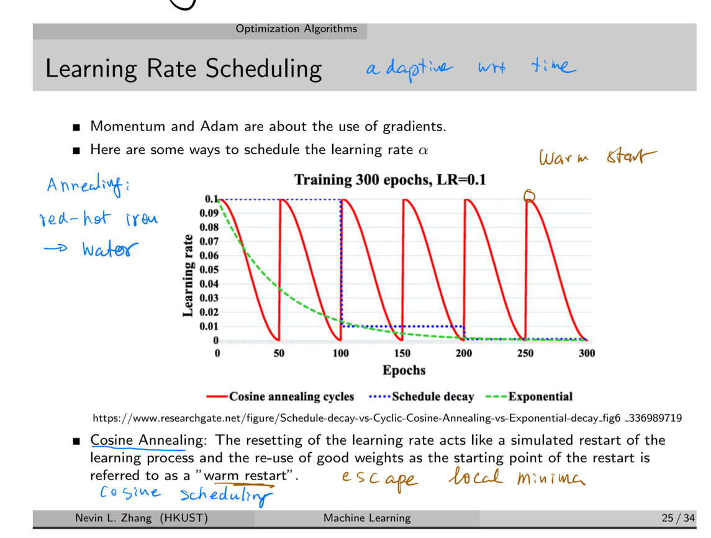
L_2 regularization and Weight Decay are equivalent for SGD

But They are not equivalent for Adam!

AdamW: Adam with decoupled Weight decay

"$g_t = \nabla L + \lambda\theta_{t-1}$"

Learning Rate rescheduling



Batch normalization

if $x_1 \in [0, 1]$, $x_2 \in [10, 100]$

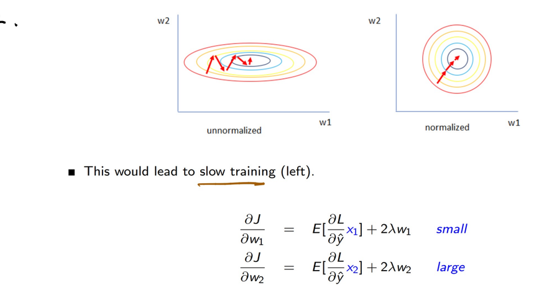
$\hat{y} = w_0 + x_1 w_1 + x_2 w_2$

$J(w) = E[L(y, \hat{y})] + \lambda(w_1^2 + w_2^2)$

"Loss Function with L_2 regularization"

Problem: 1. regularization affect w_1 more than w_2

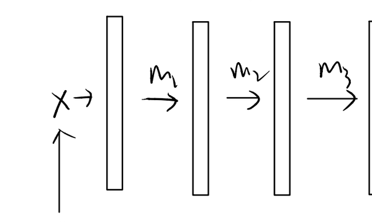
2.



Data Normalization

$\hat{x}_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}$

In deep model



if we only normalize x.

$x \xrightarrow{\text{layer 1}} m_1$, m_1's distribution cannot be guaranteed!

different layers becomes dependent / Covariate shift

$x \sim N(\mu, \sigma)$ $\to$ after n layers $\to$ "$m \sim$?" (layer n+1) $\Rightarrow$ Unstable learning; Lower learning rate; cannot converge; ⋮

We're going to normalize each layer for each mini-batch

layer n $\to x \to \hat{x} = \frac{x-\mu}{\sigma}$ (Normalization) $\longrightarrow y = \gamma\hat{x} + \beta$ (learnable variables) $\longrightarrow$ layer n+1

$x \longrightarrow \hat{x} \sim N(0, 1)$ is that enough? $\to$ lack of expression!

Scaling & shifting: make data more expressive!

Remarks:

1. Batch normalization can reduce "Vanishing Gradient" Problem
2. Batch normalization has different computation result in training and prediction (i.e. testing) modes. (Since dataset are not the same!)