How learning differs from pure optimization

$$J(\theta) = \mathbb{E}_{(x,y)\sim \hat{p}_{data}} L(f(x;\theta), y)$$

- Emperical Risk minimization

$$\mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y) = \frac{1}{m}\sum_{i=1}^{m} L(f(x_i;\theta), y_i)$$

- Surrogate Loss functions and early stopping

Pure optimization: Minimize/zero out the gradient.

Learning: Aim to minimize the generalization/validation error.

- Batch and minibatch algorithms.
  - Compute overall training data sometimes hard to accomplish, and some training data may be related.
  - Stochastic methods: Randomly sample batches.

    Online methods: Data come one-by-one sequentially from on-line.
  - When we get endless data source, how we arrange and utilize is more important than solving over-fitting.

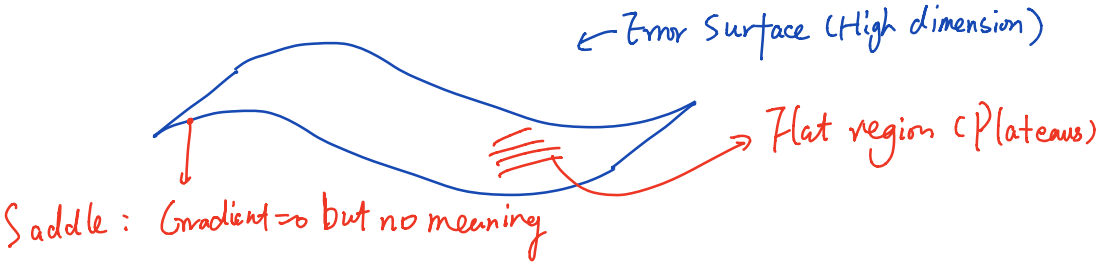
Challenges in Neural Network Optimization

- Ill Conditioning
  - Hessian matrix may be ill-conditioned/poorly-conditioned so that the error surface has more traps and too complicated.

$$\frac{1}{2}\epsilon^2 g^T H g - \epsilon g^T g \text{ when } g^T H g \text{ exceeds } g^T g$$

  - Ill-conditioning $\approx$ Gradient norm $\nearrow$ $\neq$ Tratning failed
- Local minima $\Rightarrow$ we can plot the grdient over time.
  - Model Identifiability Problem $\Rightarrow$ Sometimes caused by weight space symmetry

- Local minimum + High cost ⇒ Problem
- Plateaus, Saddle Points and other flat region



Learning algorithms should escape or avoid points of these kinds.

- Cliffs and Exploding Gradients
- Long terms Dependencies

$$W^t = (V \operatorname{diag}(\lambda) V^{-1})^t = V \operatorname{diag}(\lambda^t) V^{-1}$$

we get vanishing and exploding gradients problem.

- Inexact Gradients
- Poor Correspondence between local and global Structure
  - During learning process, the direction of local optimization may be useless or opposite to global optimal.
  - Initial weighs thus are important.
- Theoretical Limits for Optimization

Back to the conclusion that we really need focus on validation error.

Basic Algorithms

- Stochastic (batch selected) gradient descent.

When we talking about SGD, we are talking on:

$$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$ m is the size of a mini batch

$$\theta \leftarrow \theta - \epsilon_k \hat{g}$$ $G_k = (1-\alpha)G_{k-1} + \alpha G_{k-1}$, which means that we first go quick then go gentlely.

Though Stochastic minibatch may sometimes mis-leading, but it has many other benefits, say rapid progress when beginning

- Momentum

  Solving two aspects of problem:

  ① Poor-Conditioning Hessian.

  ② Variance in stochastic gradient.

  $v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_i L(f(x_i; \theta), y_i)\right)$

  $\theta \leftarrow \theta + v$

  Update rules with momentum is actually considering a sequence of gradients. Similar with a physical system with two force affecting a particle: Loss $\nabla_\theta L$ and viscous drag $-v(t)$

- Nesterov Momentum

  Calculating gradients with current θ adding by current momentum.

Parameter Initializating Strategies

- Initialization often breaks symmetry.
- Bigger value can keep stability but always failed with vanishing or exploding problem.
- We prefer small weights cause it gives more optimization space.
- Many initialization tricks: Sparse initialization, try with a mini-batch. etc.

Algorithms with adaptive learning rates

- AdaGrad

  Dividing all the historical gradients summation as the learning rate

- RMSprop ⇒ Adam algorithms

- RMSprop is a way of combining momentum with AdaGrad. Adam does more:

For each minibatch with size M.

$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x_i;\theta), y_i)$ Compute gradient

$S \leftarrow \rho_1 S + (1-\rho_1) g$ Corresponding to momentum

$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$ Corresponding to Adagrad.

$\hat{S} \leftarrow \frac{S}{1-\rho_1^t}$

$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$ > Rescale them in case they sometimes being too small.

$\theta \leftarrow \theta - \epsilon \frac{\hat{S}}{\sqrt{r}+\delta}$ → momentum; → Adagrad.

Approximate Seand-Orders Methods

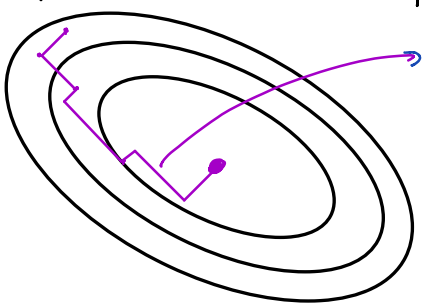
Note that for poorly conditioning Hessian, second orders methods may fail this disaussion here introducing Newton Method and some other optimization for saving computational loss.

- Newton's method

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

- Conjugate Gradients

An optimized version of steepest gradient descent



Steepest descent line search according to vertical direction of last time

$d_t \cdot d_{t-1} = 0$

$d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$ → Approximate H^{-1}

Conjugate Gradients define a direction of line search, which is more efficient than steepest descent.

- BFGS

  Saving computational cost of geting H using a part information ∇.

Optimization Strategies and Meta-Algorithm

- Batch Normalization
  - An important assumption of deep learning: Updating a layer with other layers stay stable.

    ↳ But we cannot achieve. ⟶ Vanishing/Exploding gradients.
  - Form of batch normalization

$$\mu = \frac{1}{m}\sum_i H_i \quad \text{(Here } H \text{ is design matrix)}$$

$$\delta = \sqrt{\delta + \frac{1}{m}\sum_i (H-\mu)_i^2} \quad (\delta \text{ is a stable factor})$$

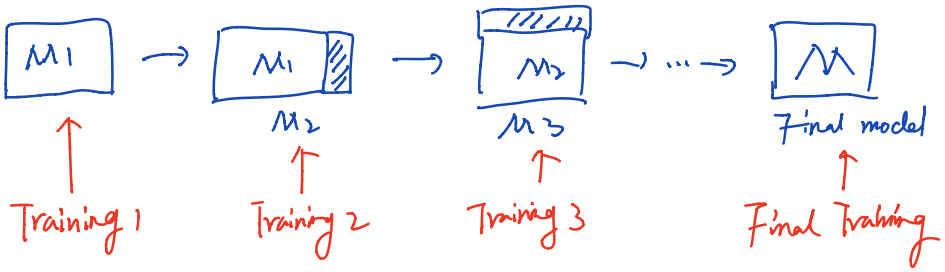
  - BN can degenerate linear accumulation of past units but preserve non-linear and other attributes.
  - Learnable parameters γ and β.

$$\gamma H' + \beta \longrightarrow \hat{H}$$

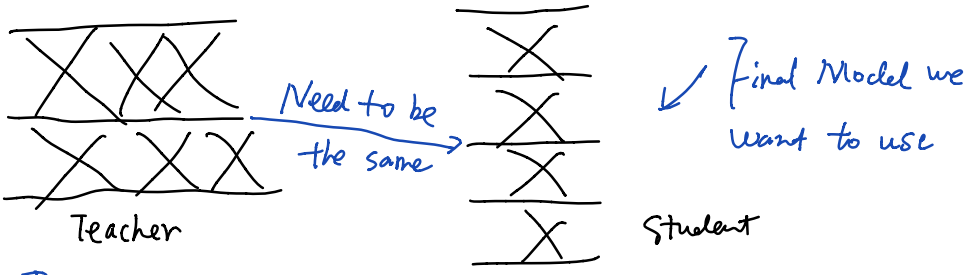
Here these two parameters can be learned by deep neural networks easily and increasing the generalization ability.

- Coordinate Descent : learn partly the parameters each time.
- Polyak Averaging : Average gradients from a period, then update.
- Designing models to aid optimization: Auxiliary link / structure
- Continuation methods and curriculum learning : Leaving from J_0 (simple) to J (hard)

- Supervised pretraining:
  - Greedy supervised pretraining:



  - FitNet:



Train first the same task