CS7150 Deep Learning

Jiaji Huang

https://jiaji-huang.github.io

02/17/2024

Recap of Last Lecture

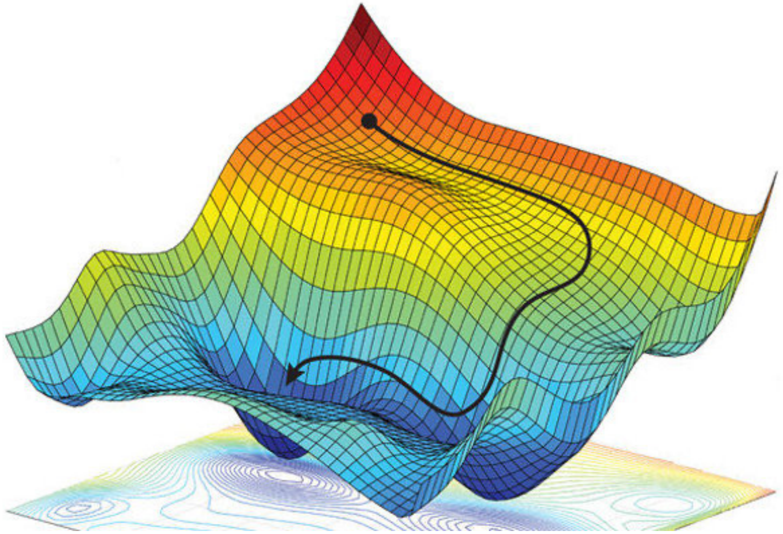
- Encoder: understanding, e.g., BERT
- Decoder: generation, e.g., GPT
- Encoder-Decoder: seq-to-seq tasks, e.g., translation, ASR

Recap of Gradient Descent (GD)

For step $s = 0, 1, \ldots$

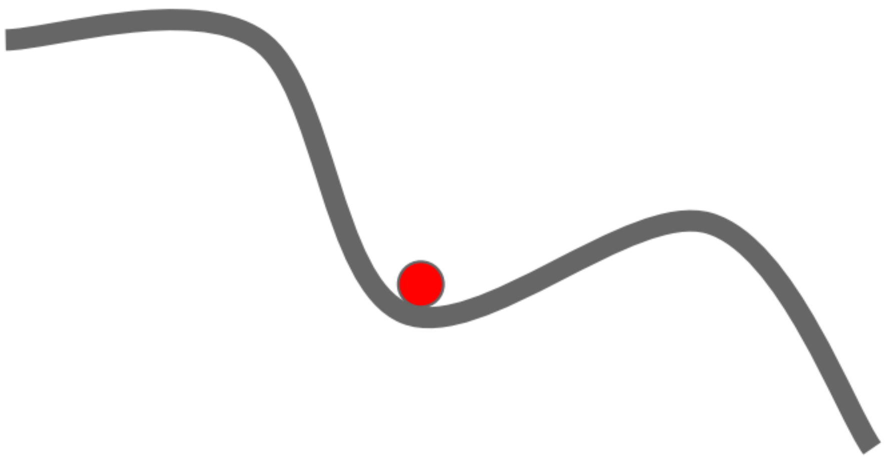
$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \eta \nabla L(\boldsymbol{w}_s),$$

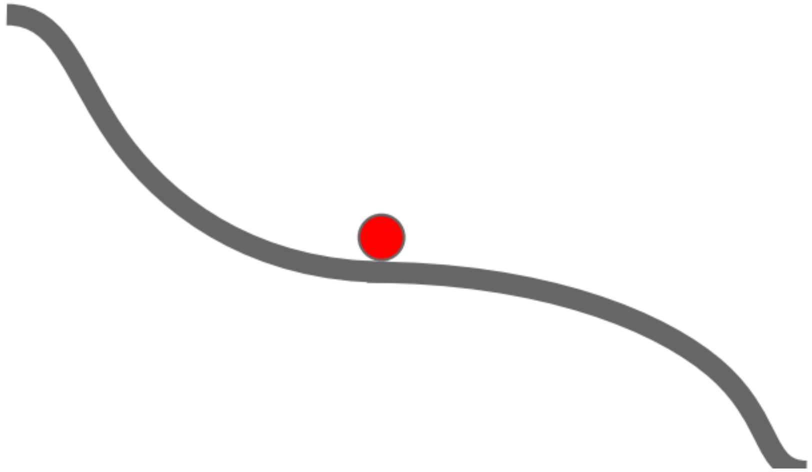
till $\nabla L(\boldsymbol{w}_s) \approx 0$ (stationary point reached)



Problem with GD

- Local minima and saddle point (both are stationary points)





- Convergence may be slow

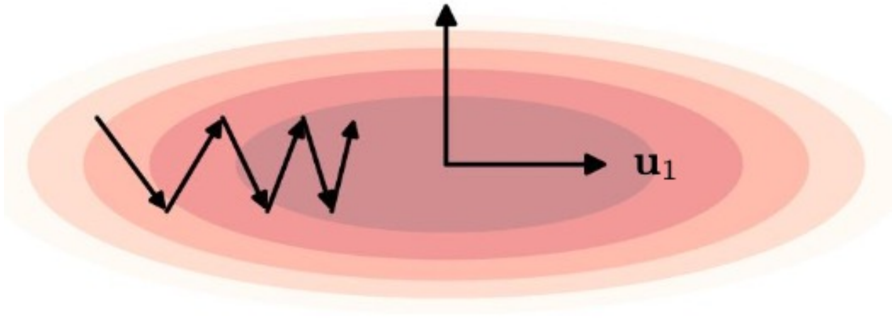


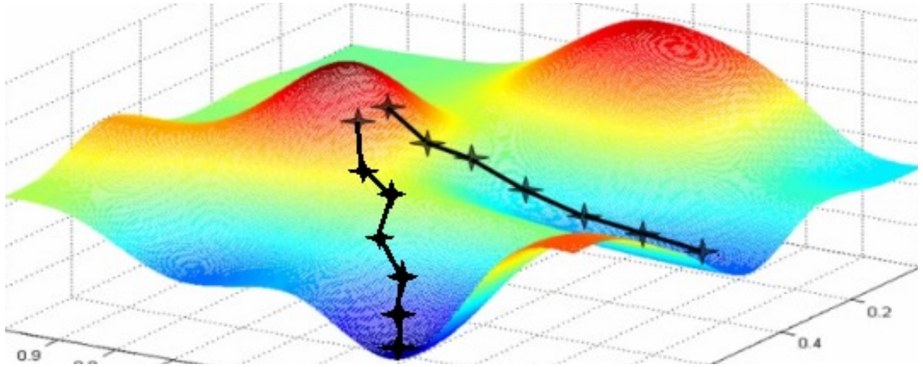
Illustration from CS231n and Bishop book chapter 7

Agenda

- Optimization
  - Initialization
  - Loss landscape and normalization
  - Momentum and higher order methods
- Generalization: old and new
- Parallelism

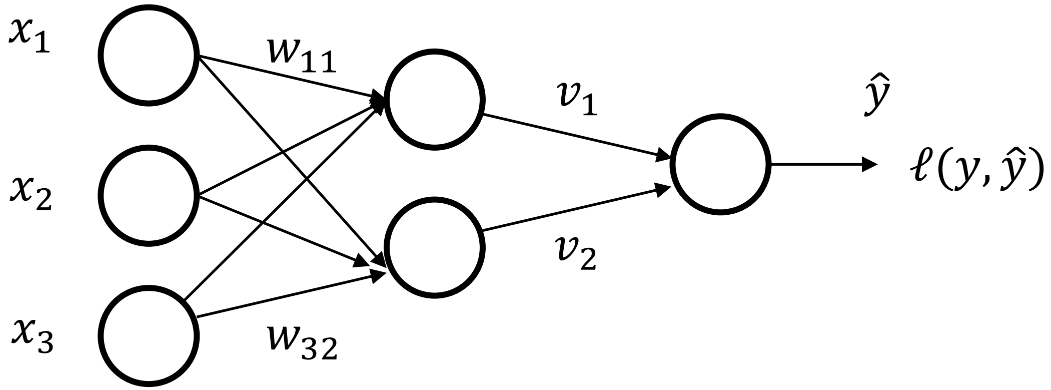
Initialization matters

- Different initializations land at different stationary points



A not so good idea

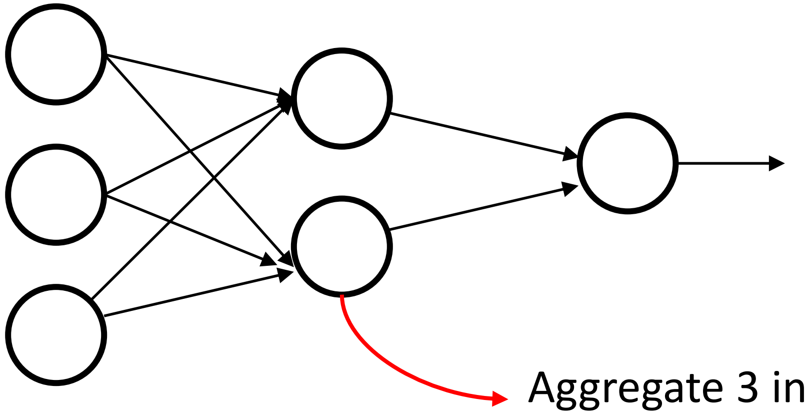
- Initialize all parameters as constant c?



- $\frac{\partial \ell}{\partial v_1} = \frac{\partial \ell}{\partial v_2}, \quad \frac{\partial \ell}{\partial w_{11}} = \cdots = \frac{\partial \ell}{\partial w_{32}}$
- Parameters in the same layer keep the same along gradient updates!

Break the Symmetry

- Initialize with random weights, draw from some distribution
- Control the variance



Aggregate 3 independent random weights, so variance may ×3

- Going deeper, activation’s variance (scale) can further increase

Xavier/Glorot Initialization

- Consider MLP network with $\sigma(\cdot)$ as sigmoid or tanh
- Key idea: activation and gradient with constant variance
- Key assumption:
  - Weights and inputs centered around 0, i.i.d distributed
  - Bias initialized as all zero
- Key property:
  - $\sigma(z) \approx z$ around 0

Glorot et. al, 2010

Xavier/Glorot Initialization

- For l-th layer:
  linear units $z_i^l = \sum_{j=1}^{n_{l-1}} w_{i,j} a_j^{l-1}$, activations $a_i^l = \sigma(z_i^l)$
- $Var\left[a_i^l\right] \approx Var\left[z_i^l\right]$ as $\sigma(z) \approx z$ around 0, so

$$Var\left[a_i^l\right] = Var\left[\sum_{j=1}^{n_{l-1}} w_{i,j} a_j^{l-1}\right] = \sum_{j=1}^{n_{l-1}} Var\left[w_{i,j} a_j^{l-1}\right]$$

$$= \sum_{j=1}^{n_{l-1}} Var\left[w_{i,j}\right] Var\left[a_j^{l-1}\right] = n_{l-1} Var\left[w_{i,j}\right] Var\left[a_j^{l-1}\right]$$

$$\Rightarrow Var\left[w_{i,j}\right] = \frac{1}{n_{l-1}}$$

Xavier/Glorot Initialization

- Consider backward

$$\frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1}^{n_l} \frac{\partial L}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial a_i^{l-1}} = \sum_{j=1}^{n_l} \frac{\partial L}{\partial a_j^l} \cdot w_{i,j}$$

$$Var\left[\frac{\partial L}{\partial a_i^{l-1}}\right] = n_l Var\left[\frac{\partial L}{\partial a_j^l}\right] Var[w_{i,j}]$$

$$\Rightarrow Var\left[w_{i,j}\right] = \frac{1}{n_l}$$

- Putting together, set

$$Var\left[w_{i,j}\right] = \frac{2}{n_{l-1} + n_l}$$

Xavier/Glorot Initialization

- If normal distribution

$$w_{i,j} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1} + n_l}\right)$$

- If uniform distribution

$$w_{i,j} \sim Uniform\left(-\frac{\sqrt{6}}{\sqrt{n_{l-1} + n_l}}, \frac{\sqrt{6}}{\sqrt{n_{l-1} + n_l}}\right)$$

He initialization

- Consider $a = ReLU(z)$
- $Var\left[a_i^l\right] \approx \frac{1}{2} Var\left[z_i^l\right]$
- Follow the same derivation, we can get
  - $Var\left[w_{i,j}\right] = \frac{2}{n_{l-1}}$ for forward variance preservation
  - $Var\left[w_{i,j}\right] = \frac{2}{n_l}$ for backward variance preservation

He et. al 2015

Question

- If the l-th layer is convolution, what would be the n_{l-1} and n_l?

(C_{out}, C_{in}, m, n) filter

$n_l = C_{out}$

$n_{l-1} = C_{in} \times m \times n$

Agenda

- Optimization
    - Initialization
    - Loss landscape and normalization
    - Momentum and higher order methods
- Generalization: old and new
- Parallelism

Loss landscape

- Consider Taylor's expansion near a stationary point $\boldsymbol{w}^*$
- $L(\boldsymbol{w}) = L(\boldsymbol{w}^*) + \nabla L(\boldsymbol{w}^*)(\boldsymbol{w}-\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*)$

  $= L(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*)$
- $\boldsymbol{H} = \left.\frac{\partial^2 L}{\partial \boldsymbol{w} \partial \boldsymbol{w}^T}\right|_{\boldsymbol{w}=\boldsymbol{w}^*}$ is Hessian
- Eigen-decomposition $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, and rewrite

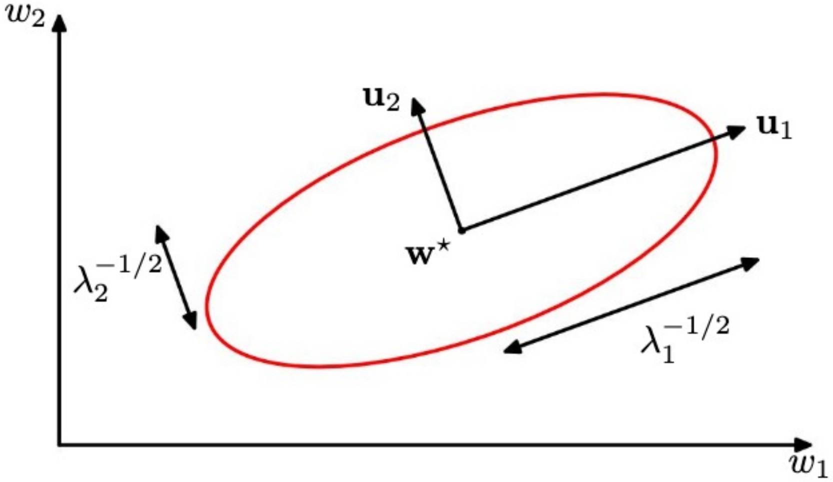
$$\frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*) = \frac{1}{2}\sum \lambda_i \left[\boldsymbol{u}_i^T(\boldsymbol{w}-\boldsymbol{w}^*)\right]^2$$

Loss landscape

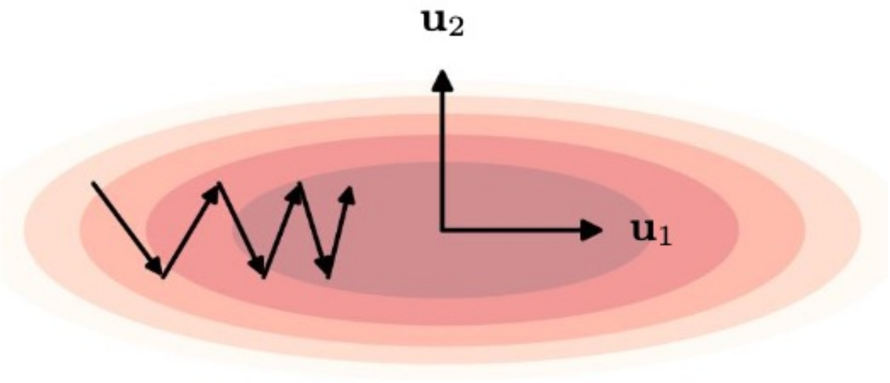
- So near local minimum $\boldsymbol{w}^*$, loss is

$$L(\boldsymbol{w}) = L(\boldsymbol{w}^*) + \frac{1}{2}\sum \lambda_i \left[\boldsymbol{u}_i^T(\boldsymbol{w} - \boldsymbol{w}^*)\right]^2$$

- If locally convex, all $\lambda_i \geq 0$



The contour of $L(\boldsymbol{w}) - L(\boldsymbol{w}^*) = 1$
(Bishop et. al, chapter 7)



Gradient steps through the contours
(Bishop et. al, chapter 7)

Loss Landscape

Local approximation

$$\boldsymbol{L}(\boldsymbol{w}) = L(\boldsymbol{w}^*) + \frac{1}{2}\sum \lambda_i \Big[\underbrace{\boldsymbol{u}_i^T(\boldsymbol{w} - \boldsymbol{w}^*)}_{\alpha_i}\Big]^2$$

- Gradient

$$\nabla L(\boldsymbol{w}) = \sum \lambda_i \boldsymbol{u}_i^T(\boldsymbol{w} - \boldsymbol{w}^*)\boldsymbol{u}_i$$

- Take a gradient step

$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \eta \nabla L(\boldsymbol{w}_s)$$

- $\alpha_{i,s+1} = (1 - \eta\lambda_i)\alpha_{i,s}$

Loss Landscape

Inspect the excessive loss term $\frac{1}{2}\sum \lambda_i \alpha_i^2$

- After s-th gradient step, $\alpha_{i,s+1} = (1 - \eta\lambda_i)\alpha_{i,s}$
- To make sure excessive loss decays, we want

$$|1 - \eta\lambda_i| \leq 1 \text{ for all } i$$

$$\Rightarrow \eta \leq \frac{2}{\lambda_{max}}$$

- The loss term that decays slowest has $1 - \eta\lambda_{min} \geq 1 - 2 \cdot \boxed{\frac{\lambda_{min}}{\lambda_{max}}}$

Inverse Condition number

The effect of unnormalized feature

- Consider linear regression

$$\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$$

- Hessian is $\boldsymbol{X}^T\boldsymbol{X}$
- e.g., 2D case, x_1: blood platelet count, ~10^5

  x_2: body height in meters, ~10^0
- $\boldsymbol{X}^T\boldsymbol{X}$ would $\sim \begin{bmatrix} 10^{10} & 10^5 \\ 10^5 & 1 \end{bmatrix}$, $\lambda_{max} \sim 10^{10}$, $\lambda_{min} \sim 0$
- Very skewed landscape, and $\frac{\lambda_{min}}{\lambda_{max}} \sim 0$, extremely slow convergence

Normalization

- Mean and standard deviation

$$\mu = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n,$$

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

- Normalization:

$$\overline{x_{n,i}} = \frac{x_{n,i} - \mu_i}{\sigma_i}$$

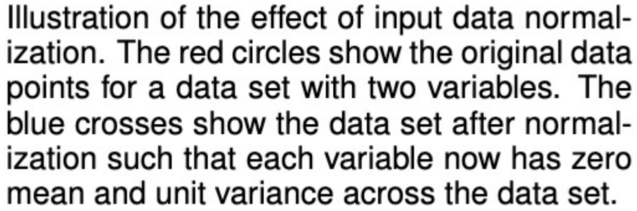


Illustration of the effect of input data normalization. The red circles show the original data points for a data set with two variables. The blue crosses show the data set after normalization such that each variable now has zero mean and unit variance across the data set.

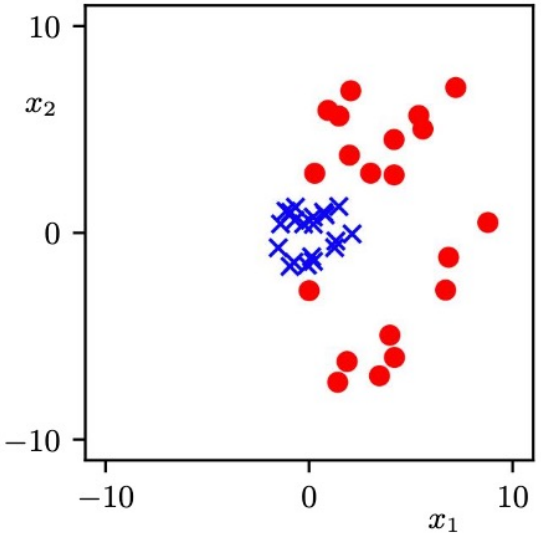
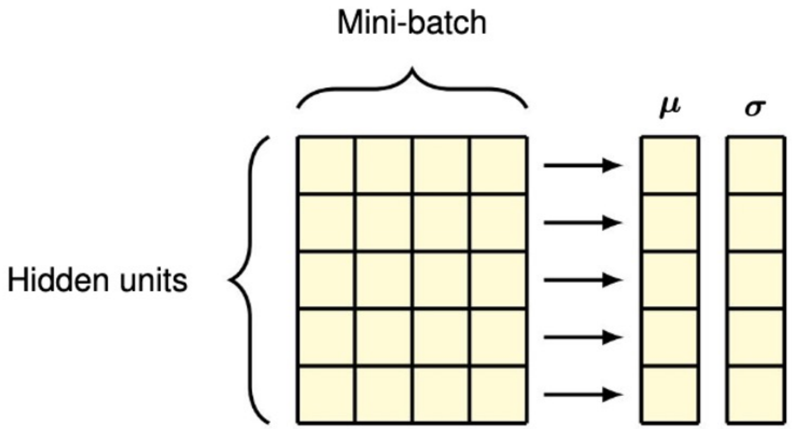


Illustration from Bishop book Chapter 7

Batch Normalization

- For the activations $\boldsymbol{x}$ at a layer
- Training phase:
  - Estimate μ and σ_i's for a minibatch, update moving average estimate
  - Normalize $\boldsymbol{x}$ to $\overline{\boldsymbol{x}}$
  - Update to $\widetilde{\boldsymbol{x}}_i = \gamma_i \overline{\boldsymbol{x}}_i + \beta_i$
- Inference phase
  - Use the moving average estimated μ and σ_i's



Ioffe et. al, 2015

Illustration: Bishop book Chapter 7

Discussion

- Why we set the learnable multipliers γ_i and β_i?
- Drawback of batch normalization?
    - Batch size = 1?
    - Determinism

Layer Norm

- Do this for each sample $\boldsymbol{x}$ in the minibatch
- Compute μ by averaging all dimensions of $\boldsymbol{x}$
- Compute σ as the norm of $\boldsymbol{x} - \mu$
- Normalize by $\widetilde{\boldsymbol{x}} = (\boldsymbol{x} - \mu)/\sigma$
- Compute $\widetilde{\boldsymbol{x}}_i = \gamma_i \overline{\boldsymbol{x}}_i + \beta_i$
- No difference between training and testing phase

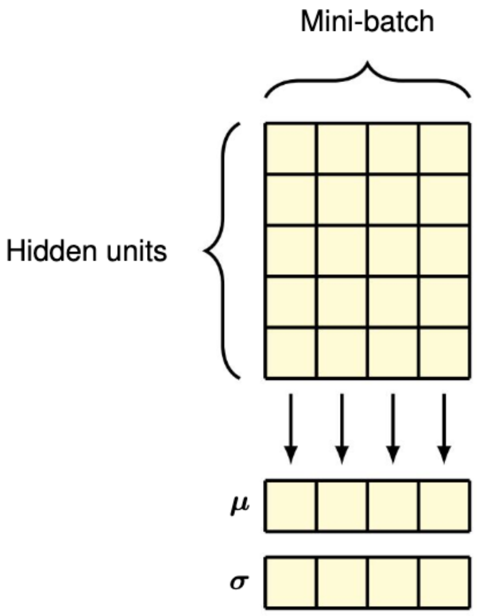


Illustration: Bishop book Chapter 7

Illustration from Bishop book Chapter 7

Agenda

- Optimization
  - Initialization
  - Loss landscape and normalization
  - Momentum and higher order methods
- Generalization: old and new
- Parallelism

Recap of Stochastic Gradient Descent (SGD)

- $\min_{\boldsymbol{w}}\{L = \frac{1}{N}\sum_{n=1}^{N} \ell(\boldsymbol{w}, \boldsymbol{x}_n)\}$
- If N is big, Compute $\nabla \ell(\boldsymbol{w}, \boldsymbol{x}_n)$ for all n's can be costly
- So we sample a mini-batch $\mathcal{B}$ of n's each step

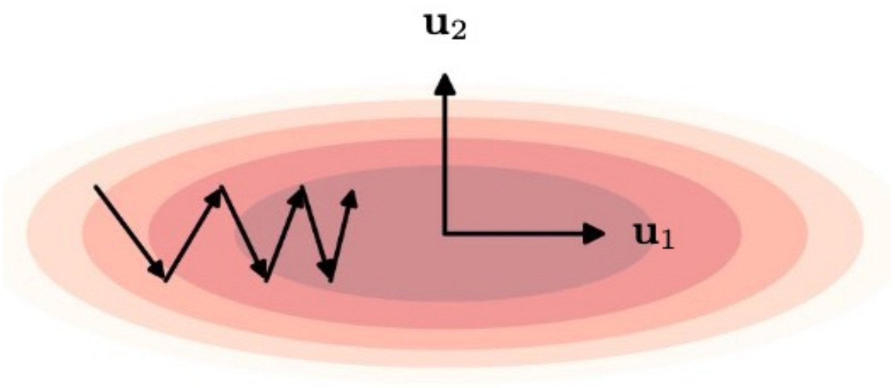
For step $s = 0, 1, \ldots$

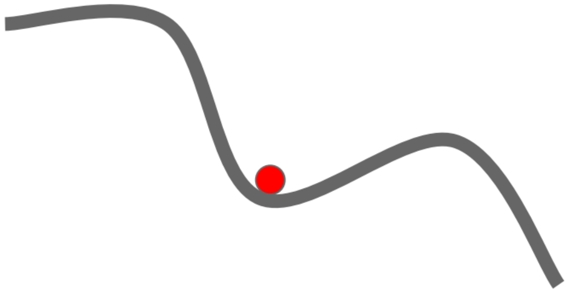
$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \eta \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \nabla \ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$$

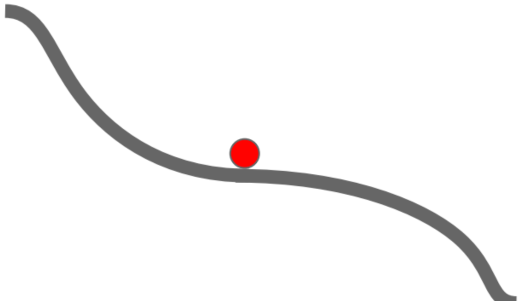
till stopping criteria met

Problems with SGD

- Same as GD
  - big condition number hamper convergence
  - Local minima and saddle point







- In addition: noisy steps

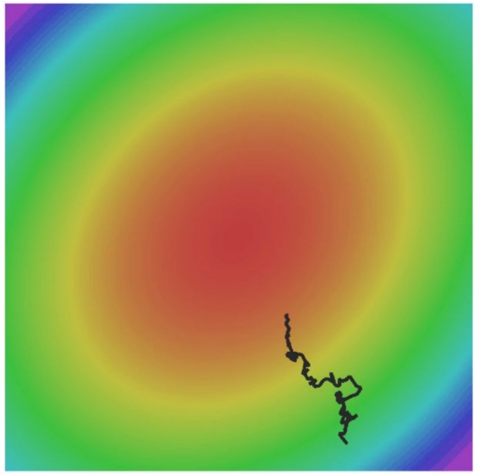


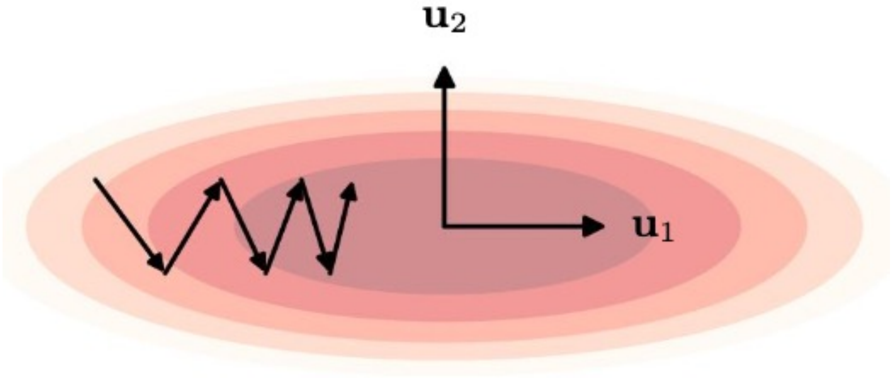
Figure from CS231n

Momentum

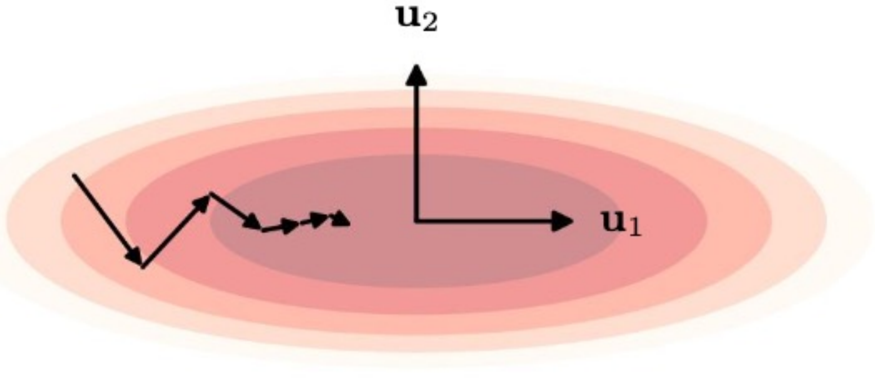
- Smooth the steps by moving average of historical gradients

$$V_s = \alpha V_{s-1} - \eta \nabla L(\boldsymbol{w}_s)$$
$$\boldsymbol{w}_s = \boldsymbol{w}_{s-1} + V_s$$

- α: momentum parameter



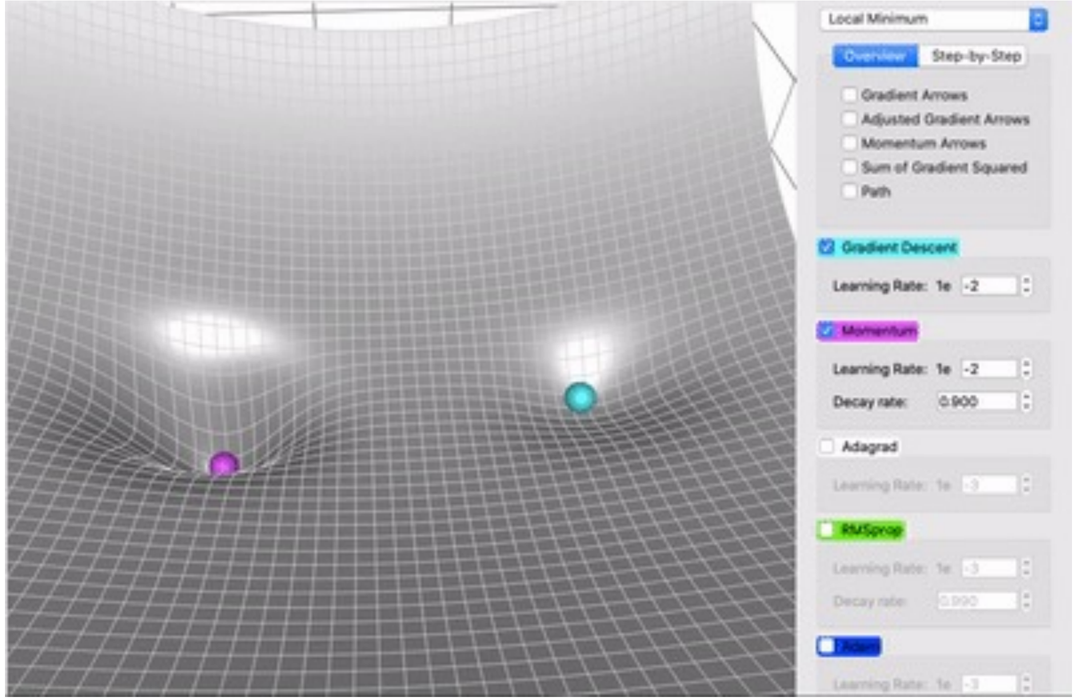
Gradient steps without momentum



Gradient steps with momentum

Skip local minima

- The inertia helps



□ Gradient Arrows
□ Adjusted Gradient Arrows
□ Momentum Arrows
□ Sum of Gradient Squared
□ Path

☑ Gradient Descent
Learning Rate: 1e -2

☑ Momentum
Learning Rate: 1e -2
Decay rate: 0.900

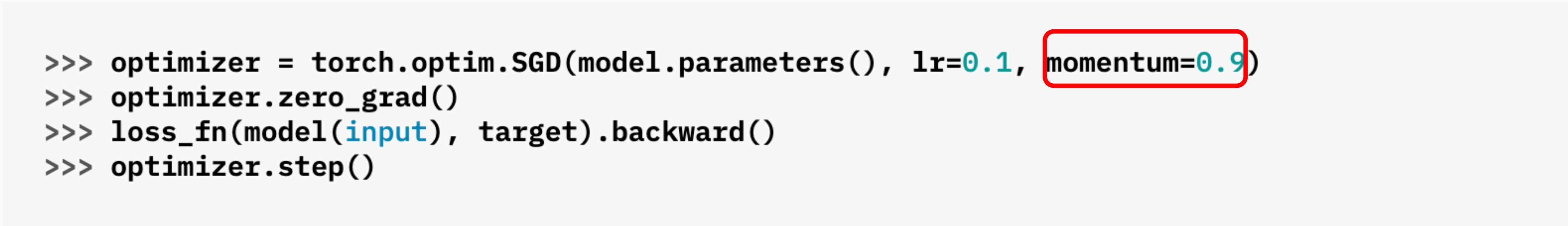
□ Adagrad
Learning Rate: 1e -3

□ RMSprop
Learning Rate: 1e -3
Decay rate: 0.990

□ Adam
Learning Rate: 1e -3

Illustration from [paperwithcode]()

In pyTorch



```
>>> optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
>>> optimizer.zero_grad()
>>> loss_fn(model(input), target).backward()
>>> optimizer.step()
```

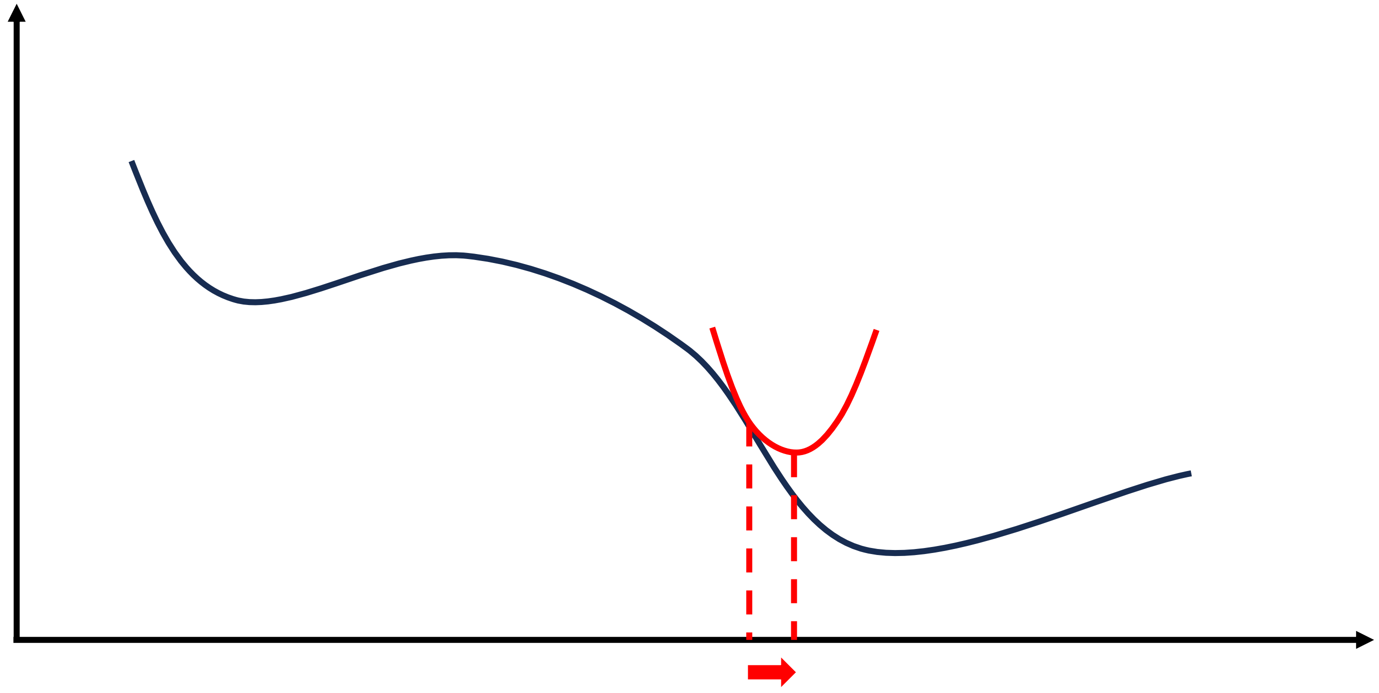
- Usually use momentum > 0.9

Agenda

- Optimization
  - Initialization
  - Loss landscape and normalization
  - Momentum
  - Higher order methods
- Generalization: old and new
- Parallelism

2nd order Optimization

- Approximate with quadratic function, and move to its minimum



Newton’s method

- $L(\boldsymbol{w}) \approx L(\boldsymbol{w}_s) + (\boldsymbol{w} - \boldsymbol{w}_s)^T \nabla L(\boldsymbol{w}_s) + (\boldsymbol{w} - \boldsymbol{w}_s)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}_s)$
- RHS has minimum at

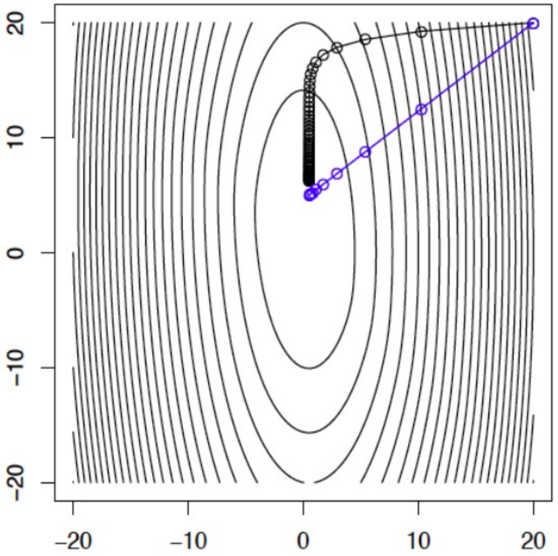
$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \boldsymbol{H}^{-1} \nabla L(\boldsymbol{w}_s)$$

- No learning rate required, faster convergence than GD
- Damped Newton’s method:

$$\boldsymbol{w}_{s+1} = \boldsymbol{w}_s - \eta \cdot \boldsymbol{H}^{-1} \nabla L(\boldsymbol{w}_s)$$

Where $0 < \eta < 1$

- Inverse Hessian is costly, $O(M^3)$, where M is number of model parameters



Black: Gradient Descent
Blue: Newton’s method

Image source: notes here

Stochastic Version

- Approximate the $\boldsymbol{H}^{-1}\nabla L(\boldsymbol{w}_s)$, where $L(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\boldsymbol{w}, \boldsymbol{x}_i)$
- Sample a batch $\mathcal{B}$

$$\nabla \widehat{L(\boldsymbol{w}_s)} = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla\ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$$

$$\widehat{\boldsymbol{H}} = \frac{1}{|\mathcal{B}|}\sum\nolimits_{n\in\mathcal{B}} \nabla^2\ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$$

- Still not practical for deep learning, because of $\widehat{\boldsymbol{H}}^{-1}$

Scale the Gradient: Adagrad

- Normalize each dimension of the gradient by accumulated magnitude

```
$G = 0$
For s=1,2,…
    sample batch $\mathcal{B}$
    compute stochastic gradient $G = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla\ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$
    $\boxed{\text{update } G \mathrel{+}= g * g}$
    $w_{s+1} = w_s - \eta \cdot \frac{g}{\sqrt{G}}$
```

Duchi et. al, 2011

Scale the Gradient: RMSProp

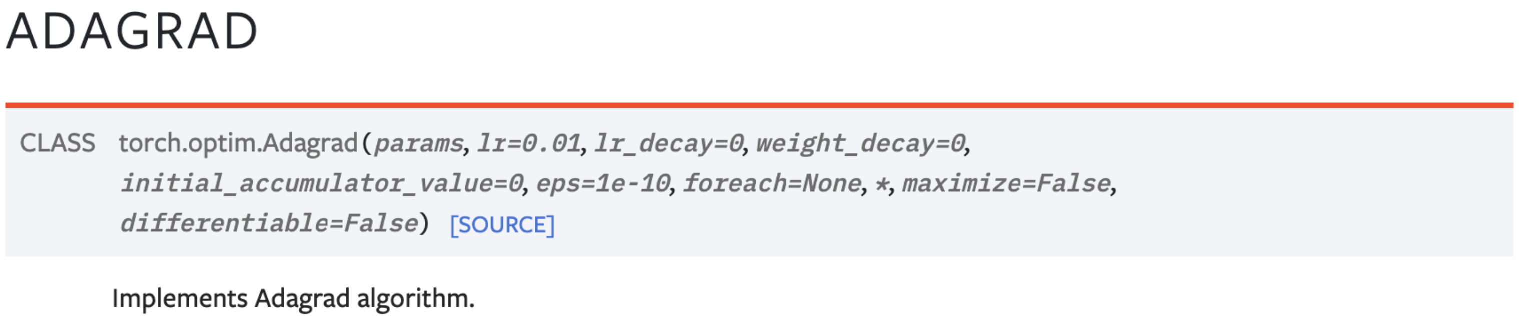
```
$G = 0$
For s=1,2,…
    sample batch $\mathcal{B}$
    compute stochastic gradient $g = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla\ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$
    update $G = \gamma G + (1-\gamma) g * g$
    $w_{s+1} = w_s - \eta \cdot \frac{g}{\sqrt{G}}$
```

Hinton et. al, 2012

Scale the Gradient: Adam

```
$G_1 = 0, G_2 = 0$
For s=1,2,...
    sample batch $\mathcal{B}$
    compute stochastic gradient $g = \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \nabla \ell(\boldsymbol{w}_s, \boldsymbol{x}_n)$
    update $G_1 = \beta_1 G_1 + (1 - \beta_1) g$
    update $G_2 = \beta_2 G_2 + (1 - \beta_2) g * g$
    bias correction: $G_1 = G_1 / (1 - \beta_1^t), G_2 = G_2 / (1 - \beta_2^t)$
    $w_{s+1} = w_s - \eta \cdot \frac{G_1}{\sqrt{G_2}}$
```

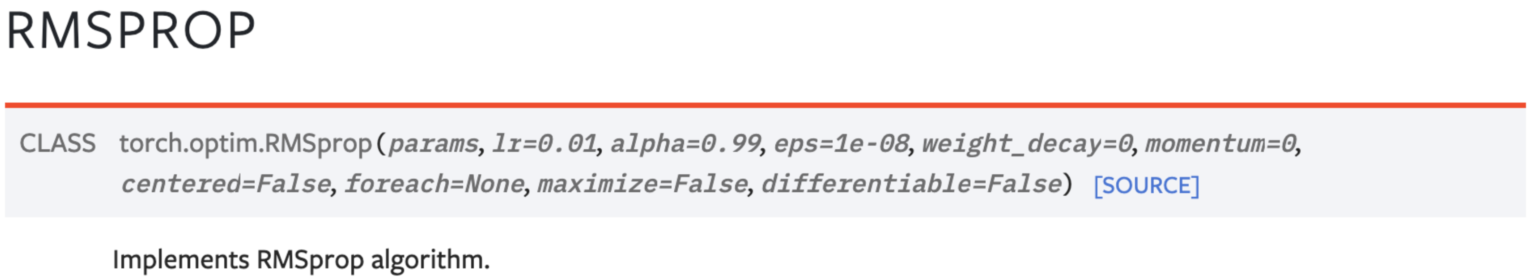
In pytorch



ADAGRAD

CLASS torch.optim.Adagrad(*params*, *lr=0.01*, *lr_decay=0*, *weight_decay=0*, *initial_accumulator_value=0*, *eps=1e-10*, *foreach=None*, ***, *maximize=False*, *differentiable=False*) [SOURCE]

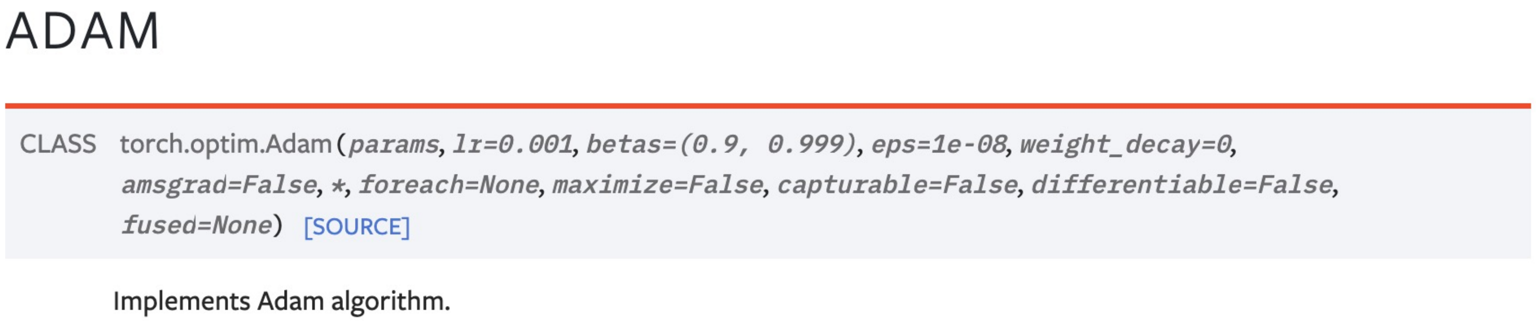
Implements Adagrad algorithm.



RMSPROP

CLASS torch.optim.RMSprop(*params*, *lr=0.01*, *alpha=0.99*, *eps=1e-08*, *weight_decay=0*, *momentum=0*, *centered=False*, *foreach=None*, *maximize=False*, *differentiable=False*) [SOURCE]

Implements RMSprop algorithm.



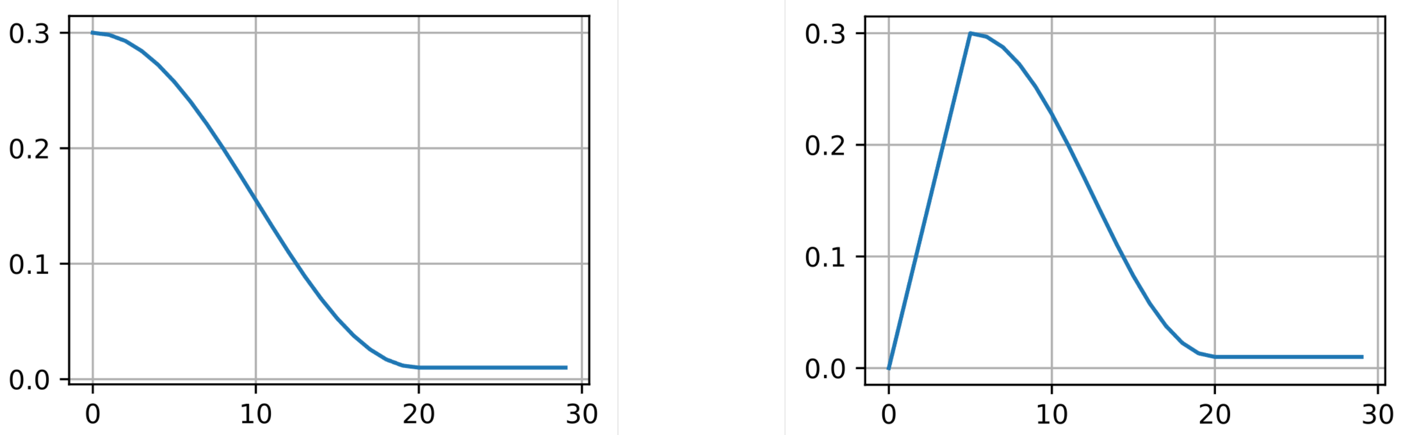
ADAM

CLASS torch.optim.Adam(*params*, *lr=0.001*, *betas=(0.9, 0.999)*, *eps=1e-08*, *weight_decay=0*, *amsgrad=False*, ***, *foreach=None*, *maximize=False*, *capturable=False*, *differentiable=False*, *fused=None*) [SOURCE]

Implements Adam algorithm.

Learning Rate Scheduler

- Infeasible to compute ideal learning rate
- Often decay the learning rate along training
- *Warmup* is sometimes used to skip local minima



cosine learning rate scheduler. Left: no warmup; right: with warmup

Check pytorch page, figures from this post

Discussion

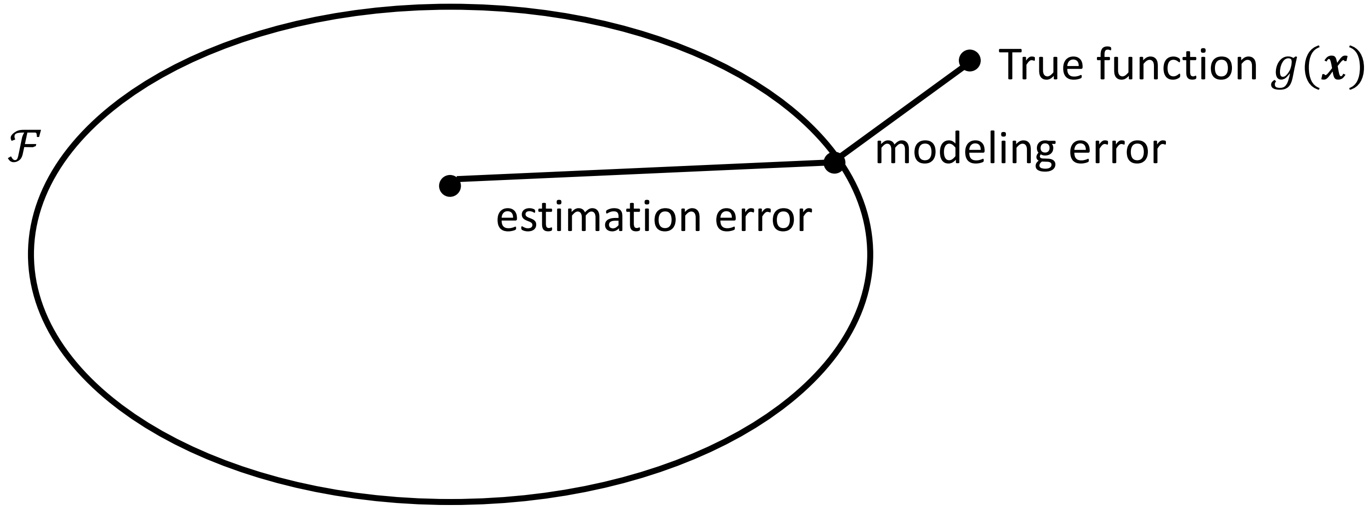
- How to choose batch size in practice?
- How do we scale learning rate w.r.t. batch size?

Agenda

- Optimization
    - Initialization
    - Loss landscape and normalization
    - Momentum and higher order methods
- Generalization: old and new
- Parallelism

Recap: Generalization

- The ability to predict well on unseen samples
- Model family $\mathcal{F} = \{f_{\boldsymbol{\theta}}(\boldsymbol{x})\}$
- Generalization error: expected error/loss on a test input
- ≈ modeling error + estimation error



Cartoon adapted from [OMCS lecture slides]

Reduce the Variance

- Inject inductive bias (encodes our prior knowledge)
- e.g., small variation doesn’t change object category

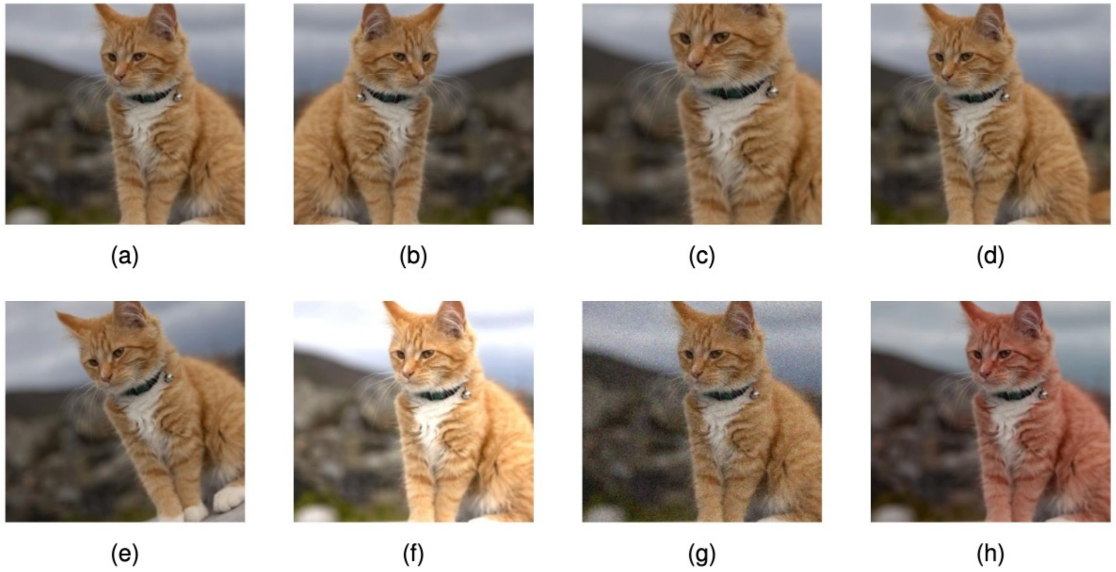


Figure 9.1 Illustration of data set augmentation, showing (a) the original image, (b) horizontal inversion, (c) scaling, (d) translation, (e) rotation, (f) brightness and contrast change, (g) additive noise, and (h) colour shift.

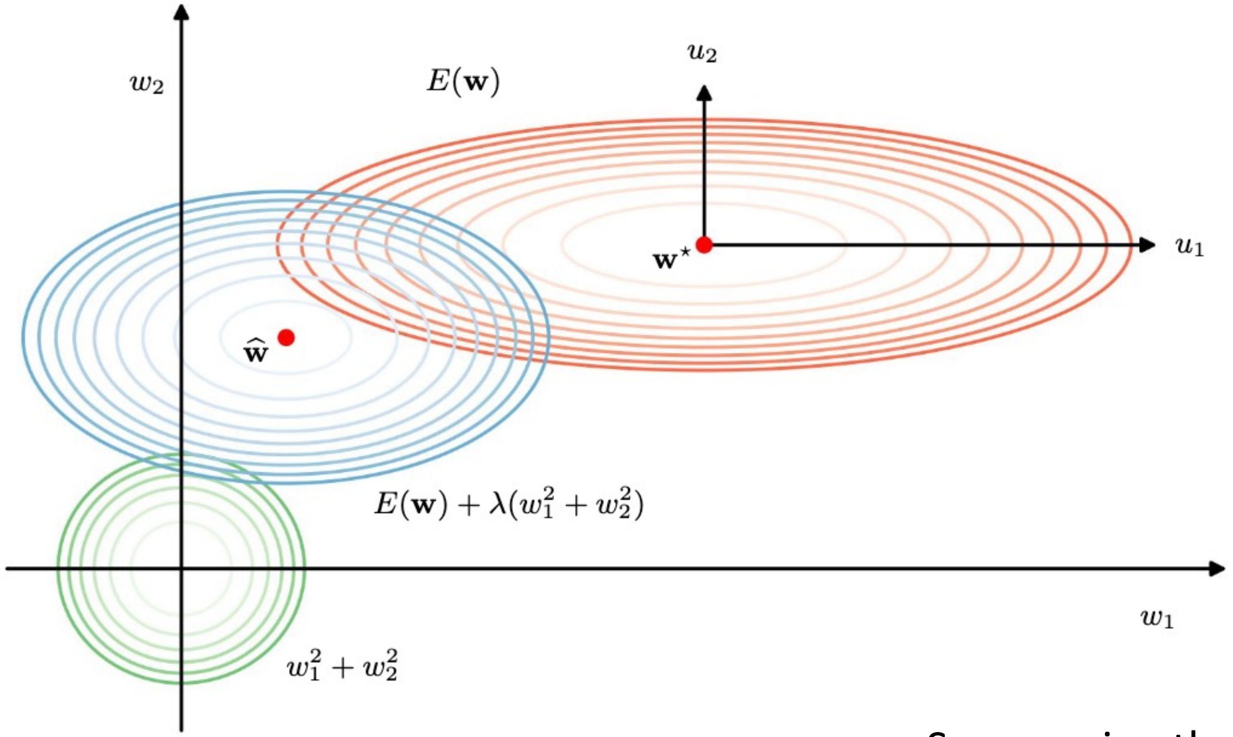
Illustration from Bishop book, chapter 9

Regularization

- Constrain the model family $\mathcal{F}$
- e.g., weight decay $\min_{\boldsymbol{w}} L(\boldsymbol{w}) + \frac{\lambda}{2} ||\boldsymbol{w}||^2$
- e.g., sparsity $\min\limits_{\boldsymbol{w}} L(\boldsymbol{w}) + \frac{\lambda}{2} ||\boldsymbol{w}||_1$

Question: what are the gradients for these two cases?

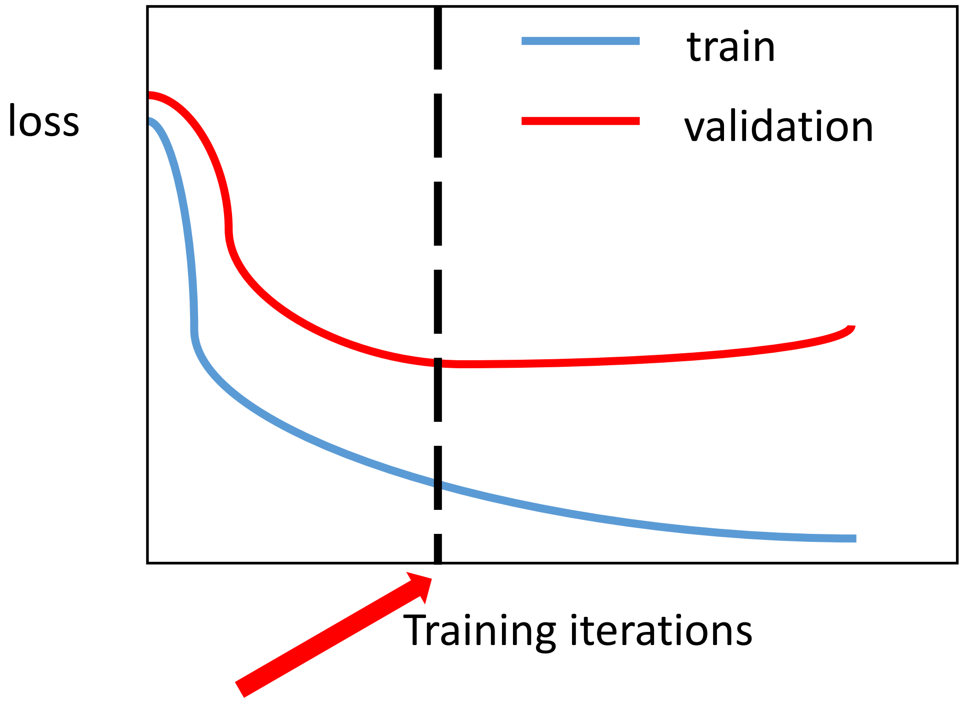
Weight Decay: Loss Landscape View



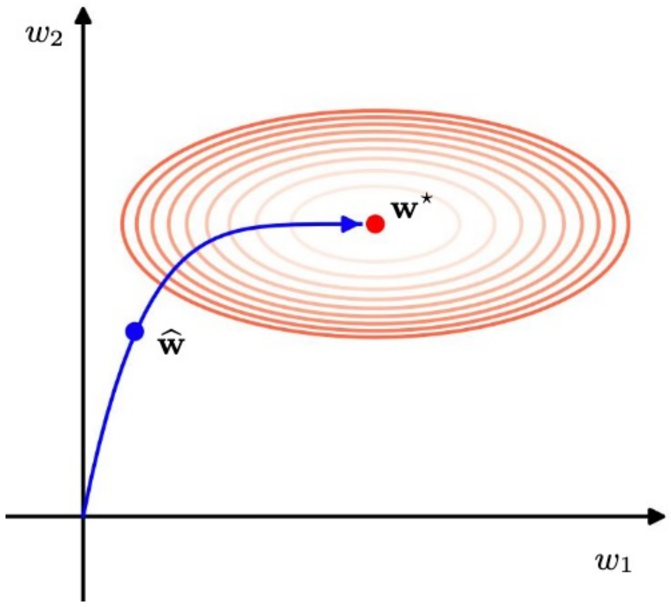
- Suppressing the magnitude of weights
- Especially those to which loss is less sensitive

Illustration from Bishop book, chapter 9

Earlier Stopping



Stop training here



From the perspective of weight decay

Bishop book Chapter 9

Drop out

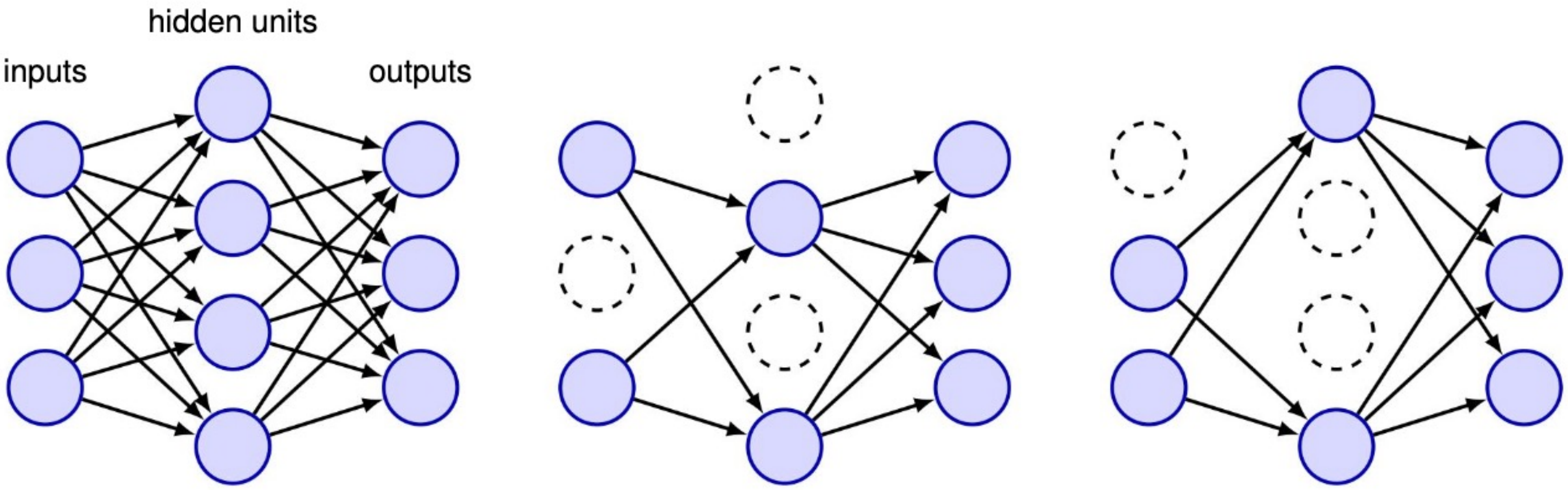
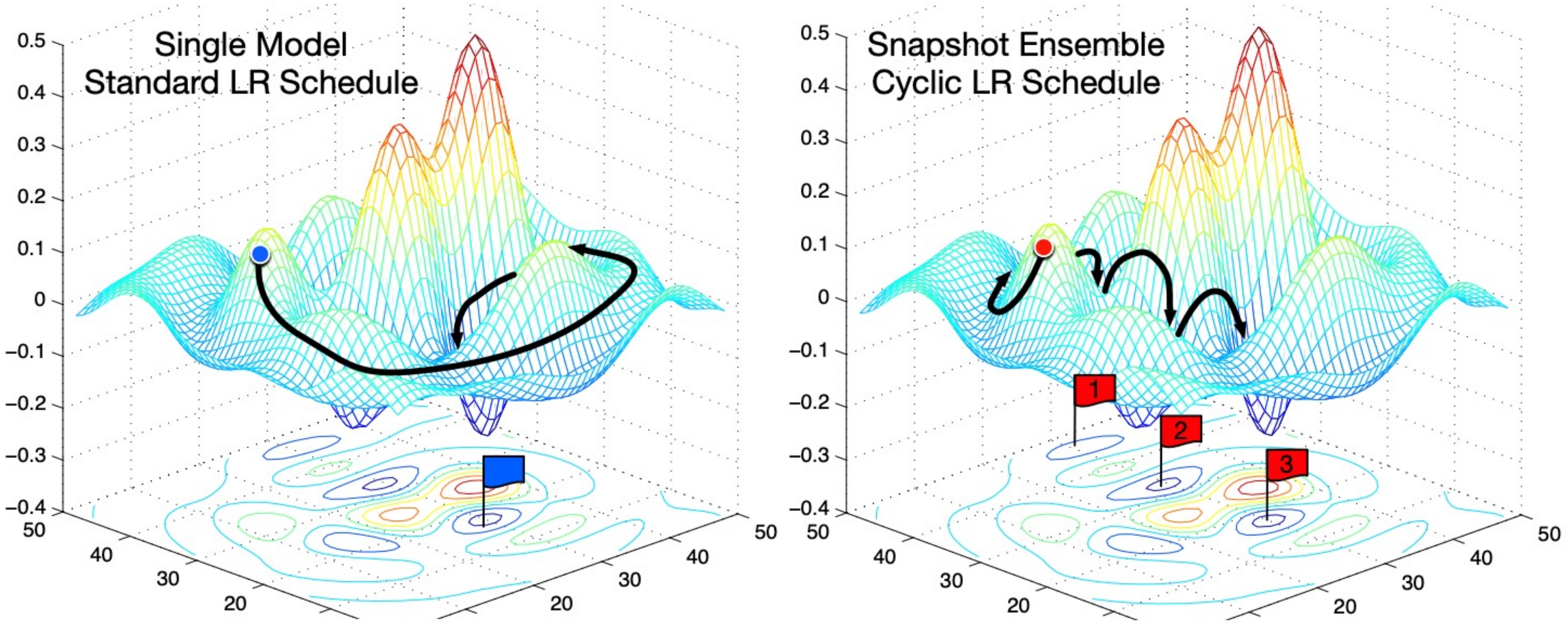


Figure 9.17 A neural network on the left along with two examples of pruned networks in which a random subset of nodes have been omitted.

Bishop book Chapter 9

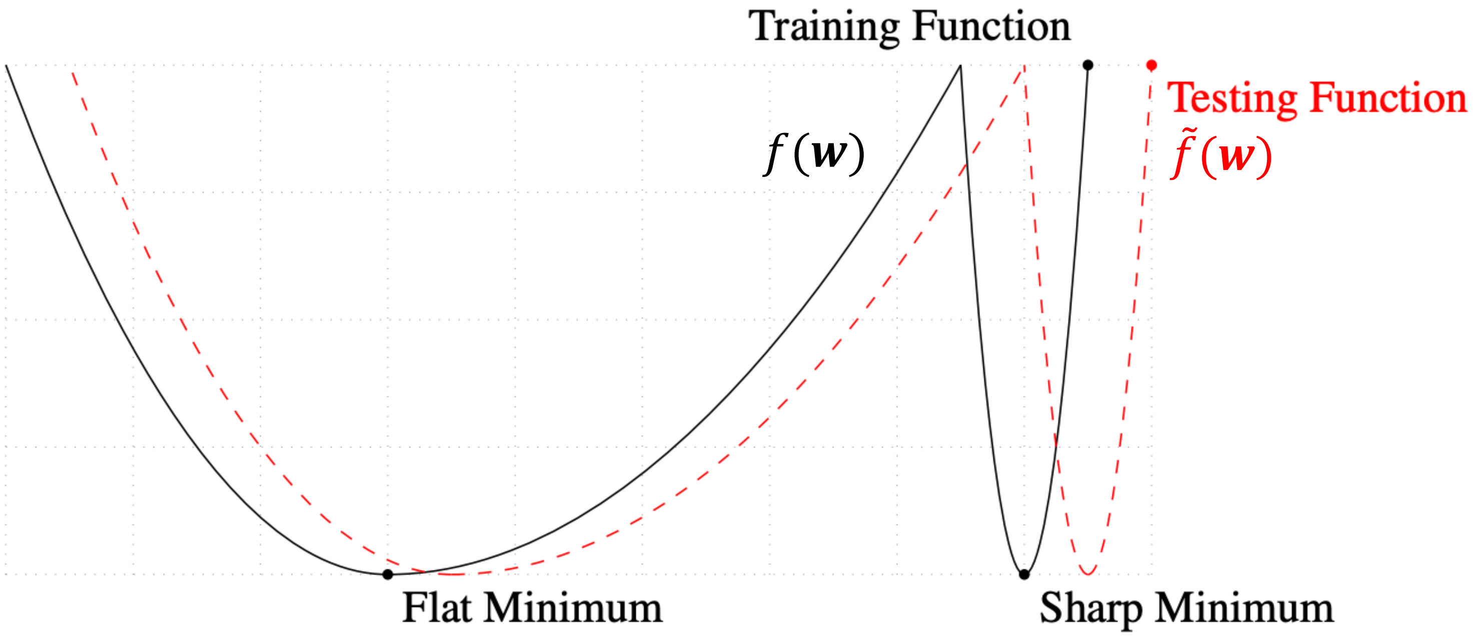
Model Averaging



Huang et. al, 2017

Sharpness

- Sharp local minima generalizes poorly
- Question: How to measure sharpness?



Keskar et. al, 2016

Agenda

- Optimization
  - Initialization
  - Loss landscape and normalization
  - Momentum and higher order methods
- Generalization: old and new
- Parallelism

Data Parallelism

- Want to use large batch size, but single GPU doesn't have enough memory

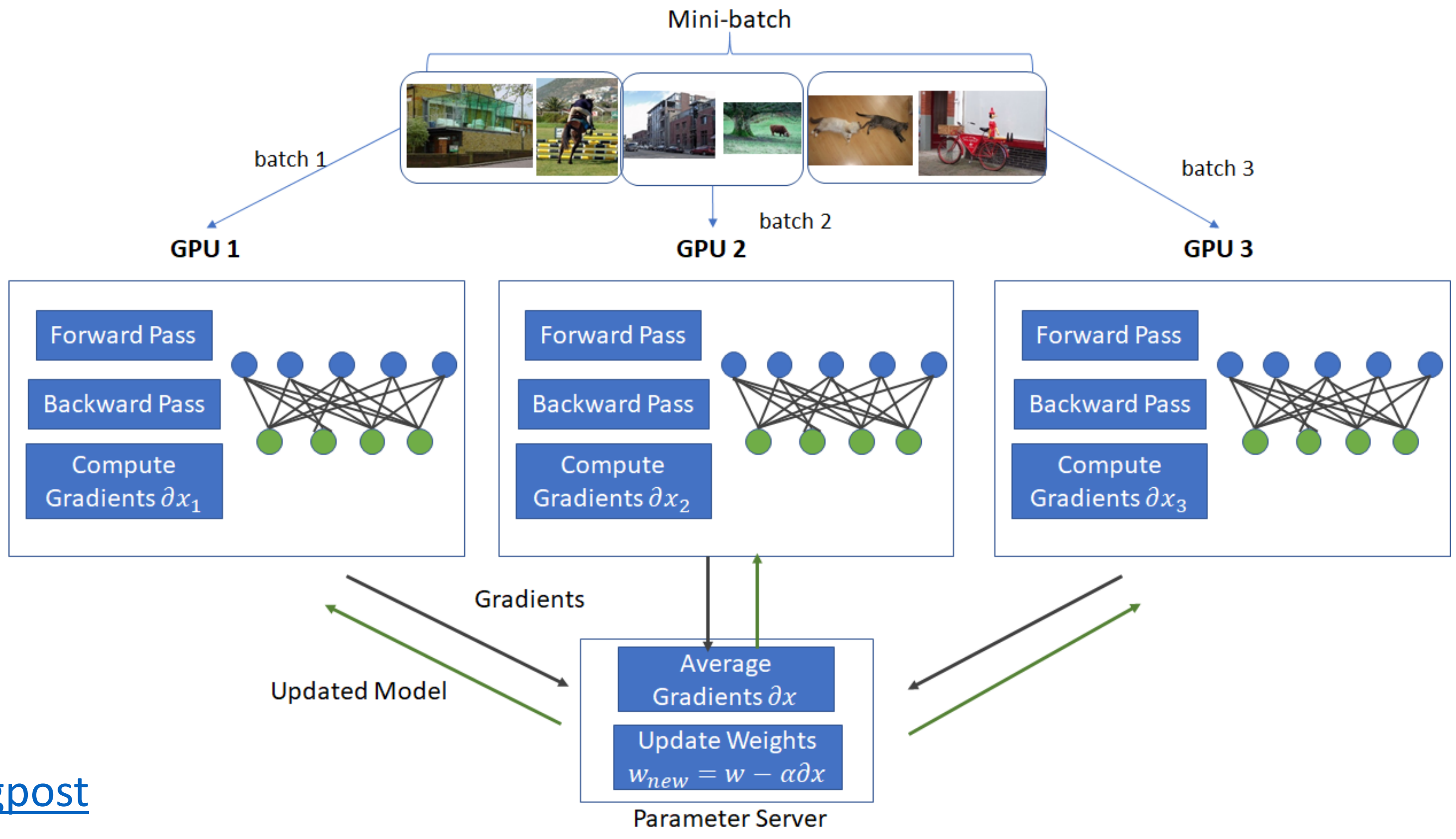

Mini-batch
batch 1
batch 2
batch 3
GPU 1
GPU 2
GPU 3
Forward Pass
Backward Pass
Compute Gradients ∂x_1
Forward Pass
Backward Pass
Compute Gradients ∂x_2
Forward Pass
Backward Pass
Compute Gradients ∂x_3
Gradients
Updated Model
Average Gradients ∂x
Update Weights $w_{new} = w - \alpha \partial x$
Parameter Server

Illustration from this blogpost

SGD with multiple workers

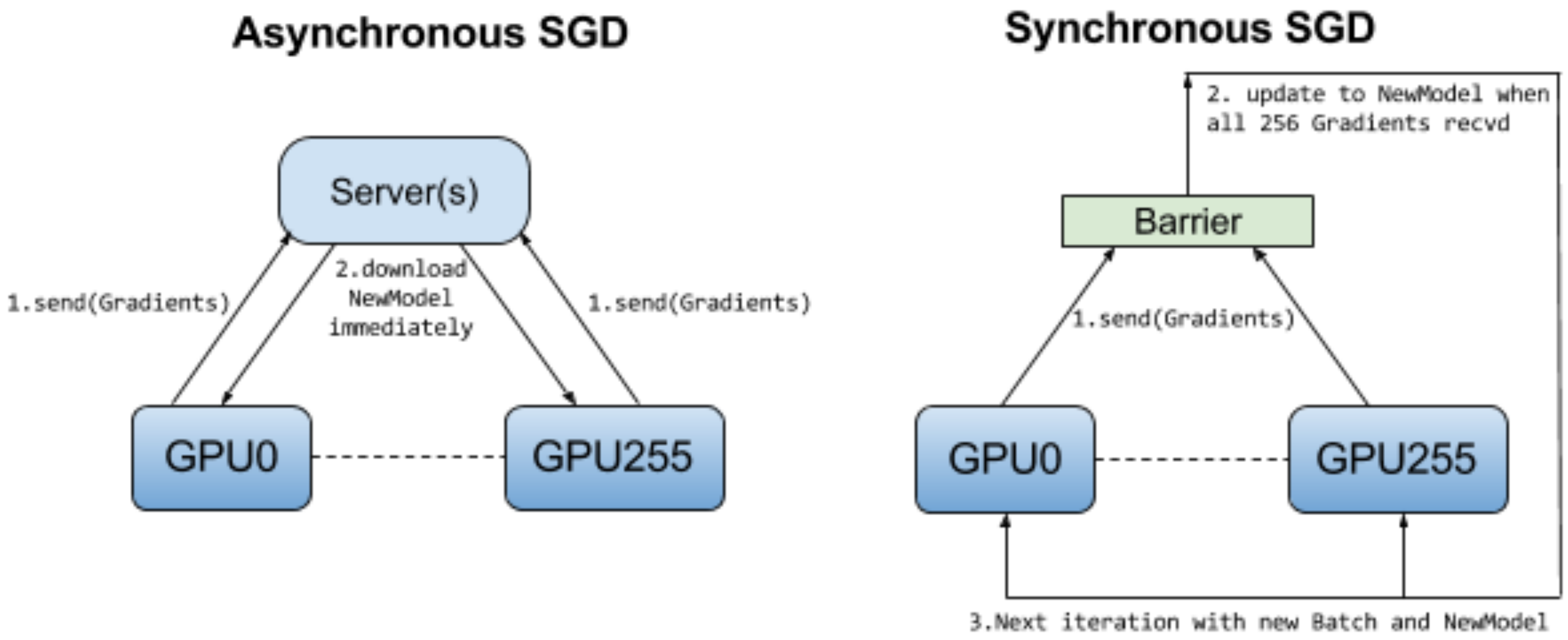
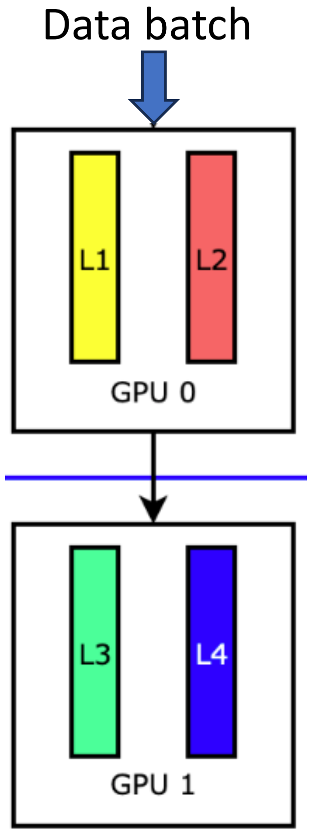

Asynchronous SGD
Server(s)
1.send(Gradients)
2.download NewModel immediately
1.send(Gradients)
GPU0
GPU255
Synchronous SGD
2. update to NewModel when all 256 Gradients recvd
Barrier
1.send(Gradients)
GPU0
GPU255
3.Next iteration with new Batch and NewModel

Illustration from this blogpost

Model Parallelism

- Pipeline parallel



- Tensor parallel

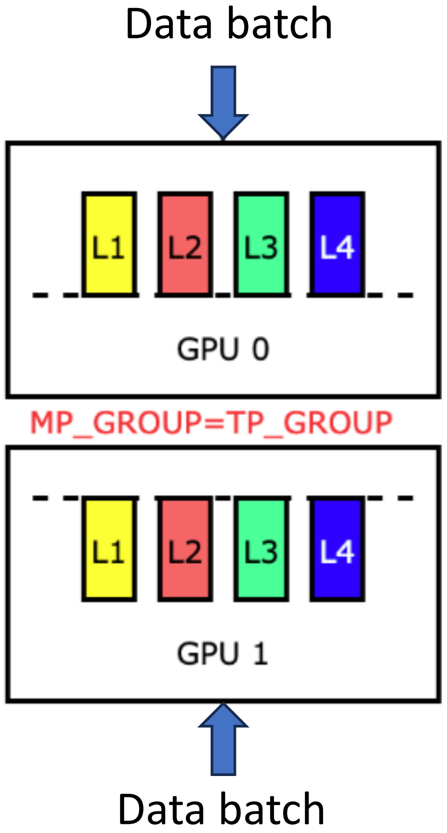


Illustration adapted from aws sagemaker