人工智能导论

Lecture 6: Training

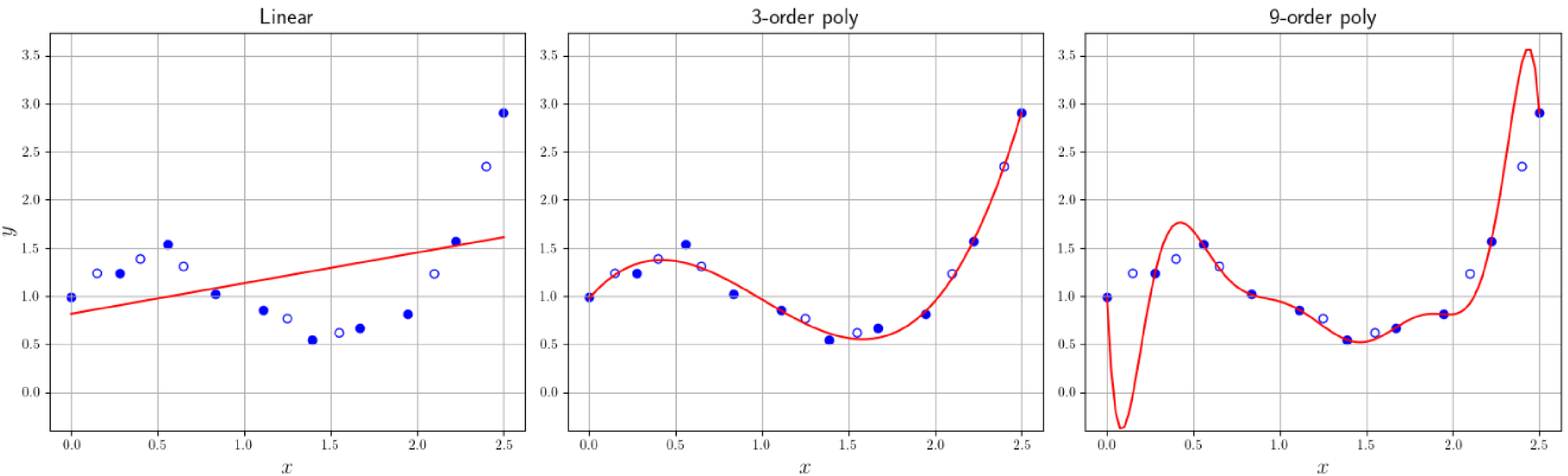
Supervised Learning - Review

- Given training data $\{(x_i, y_i)\}_{i=1}^{n}$ i.i.d. from distribution D

- Find $y = \overbrace{f}^{\text{model}}(x; \overbrace{\theta}^{\text{parameters}})$
  - which works well on test data i.i.d. from distribution D
  - θ can be trained by minimizing the empirical loss

$$\hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i; \theta), y_i)$$

Underfitting & Overfitting



underfitting well fitting overfitting

Underfitting & Overfitting

- Underfitting: the model cannot capture the underlying trend of the data
  - Large training error
  - Model is not complex enough

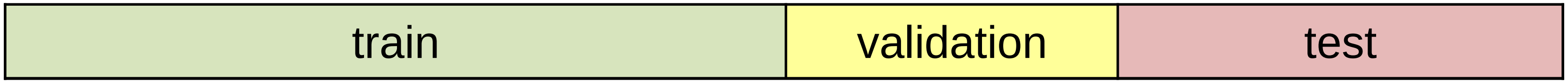
- Overfitting: the model describes random noise instead of the underlying relationship
  - Small training error but large test error
  - Model is too complex

Outline

- Model selection
- Optimization
- Not work well on training data
- Not work well on testing data
- Hyperparameter tuning

Model selection

- Given m models $f_1, f_2 \dots, f_m$, how to find the best model?



- Split data into train, validation, and test
- Choose hyperparameters on the validation data and evaluate on the test data

Cross validation

- Split data into folds, try each fold as validation and average the results

- Example: 5-fold cross validation

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Outline

- Model selection
- Optimization
- Not work well on training data
- Not work well on testing data
- Hyperparameter tuning

Optimization

- Minimize the empirical loss

$$\min_{\theta} L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i;\theta), y_i)$$

- Usually $\mathrm{L}(\theta)$ is continuous and differentiable (or subdifferentiable)

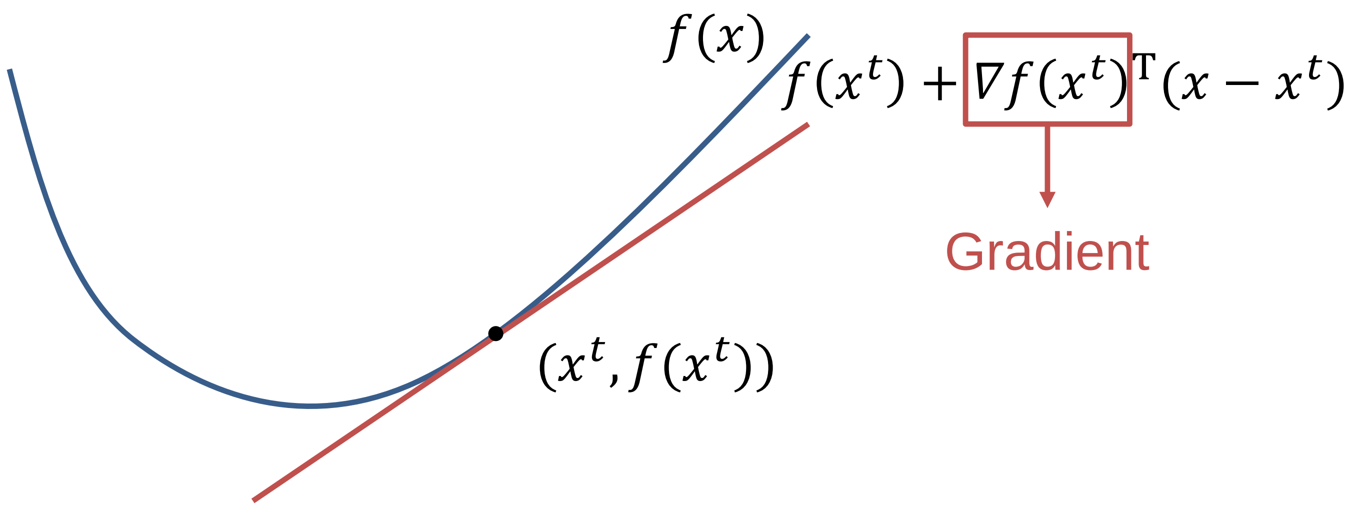
History of optimization

- 1847: Cauchy proposes gradient descent
- 1950s: Linear Programs, soon followed by non-linear, Stochastic Gradient Descent (SGD)
- 1980s: General optimization, convergence theory
- 2005-2015: Large scale optimization (mostly convex), convergence of SGD
- 2015-today: Improved understanding of SGD for deep learning

Gradient descent

- What is the steepest descent direction at x^t?

opposite direction of the gradient



Gradient descent

- Start with an initial point x^0
  - In each iteration, compute

$$x^{t+1} = x^t - \eta_t \nabla f(x^t)$$

- η_t is the learning rate

Gradient descent

- Objective function: $f(\boldsymbol{x}) = x_1^2 + 2x_2^2$, initial point: $\boldsymbol{x}^{(0)} = (1,1)^T$, learning rate $\eta = 0.2$

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1 \\ 4x_2 \end{bmatrix}$$

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - \eta \nabla f\left(\boldsymbol{x}^{(0)}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.2 \times \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}$$

$$\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)} - \eta \nabla f\left(\boldsymbol{x}^{(1)}\right) = \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix} - 0.2 \times \begin{bmatrix} 1.2 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 0.36 \\ 0.04 \end{bmatrix}$$

$$\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)} - \eta \nabla f\left(\boldsymbol{x}^{(2)}\right) = \begin{bmatrix} 0.36 \\ 0.04 \end{bmatrix} - 0.2 \times \begin{bmatrix} 0.72 \\ 0.16 \end{bmatrix} = \begin{bmatrix} 0.216 \\ 0.008 \end{bmatrix}$$

$$\cdots$$

Stochastic gradient descent

- Gradient descent:

$$\theta^{t+1} = \theta^t - \nabla_\theta \left(\frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; \theta^t), y_i) \right)$$

- Stochastic gradient descent:
  - Pick an data (x_i, y_i)

$$\theta^{t+1} = \theta^t - \nabla_\theta (\ell(f(x_i; \theta^t), y_i))$$

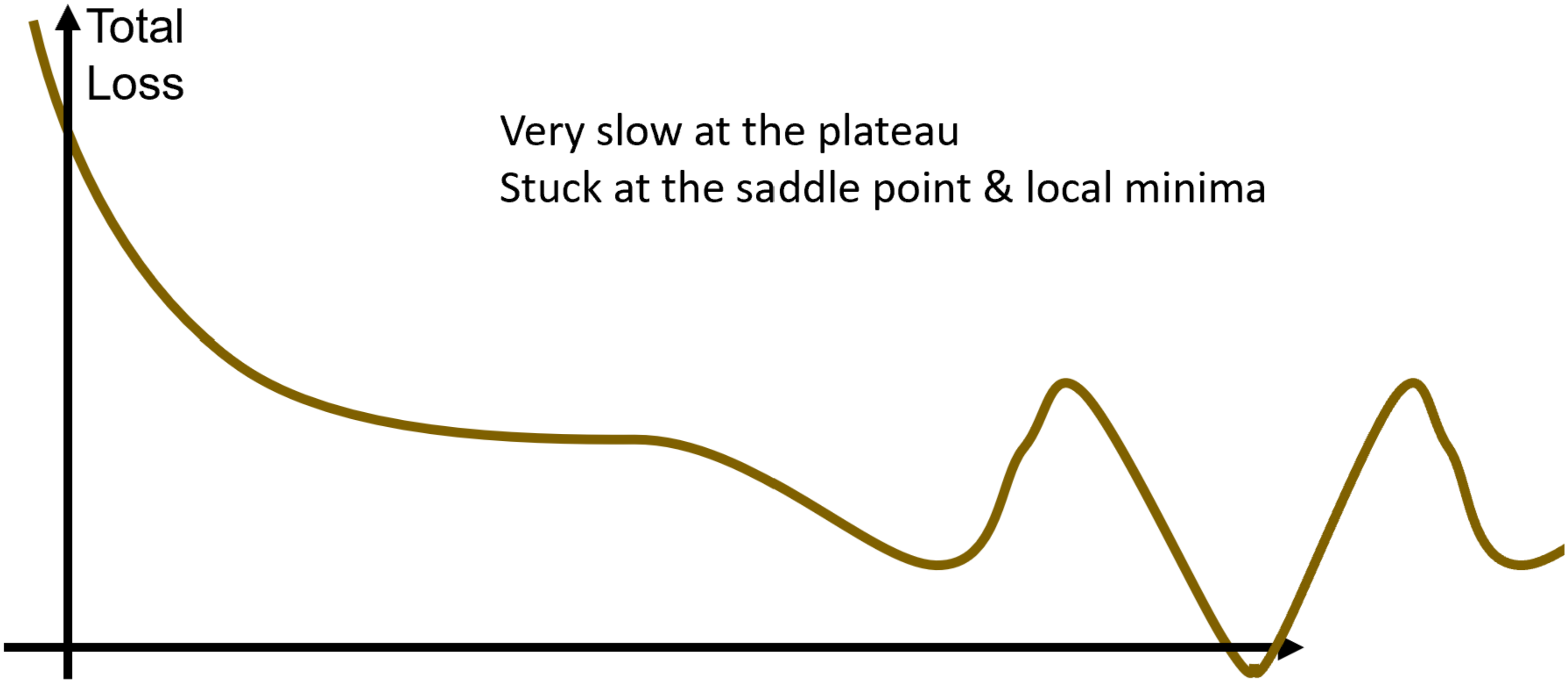
Mini-batch SGD

- In each iteration, randomly pick a mini-batch $S = \{(\mathrm{x}_{\mathrm{b}_1}, \mathrm{y}_{\mathrm{b}_1}), \ldots, (\mathrm{x}_{\mathrm{b}_k}, \mathrm{y}_{\mathrm{b}_k})\}$

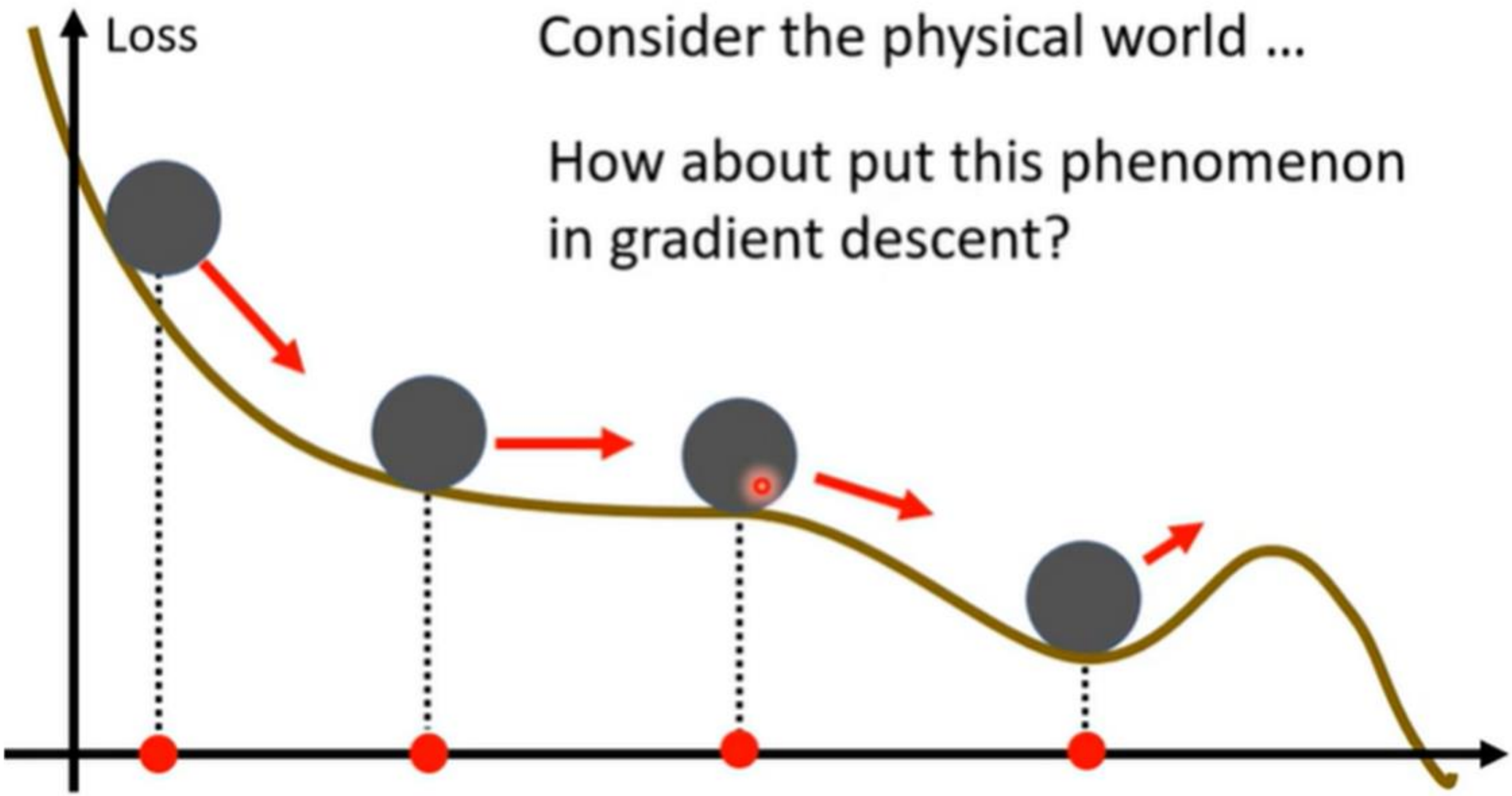
$$\theta^{t+1} = \theta^t - \nabla_\theta \left(\frac{1}{|S|} \sum_{(x_i, y_i) \in S} \ell(f(x_i; \theta^t), y_i) \right)$$

- What are the parameters?
  - Learning rate
  - Batch size
  - When to stop

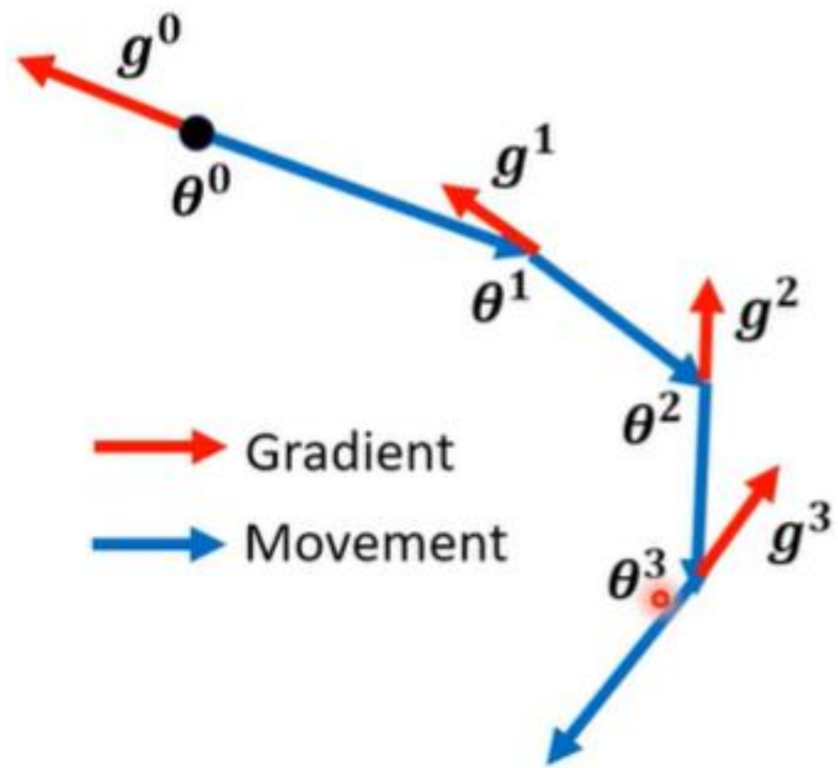
Challenges of nonconvex optimization



Momentum


Loss
Consider the physical world ...
How about put this phenomenon in gradient descent?

Vanilla Gradient Descent



Starting at θ^0

Compute gradient g^0

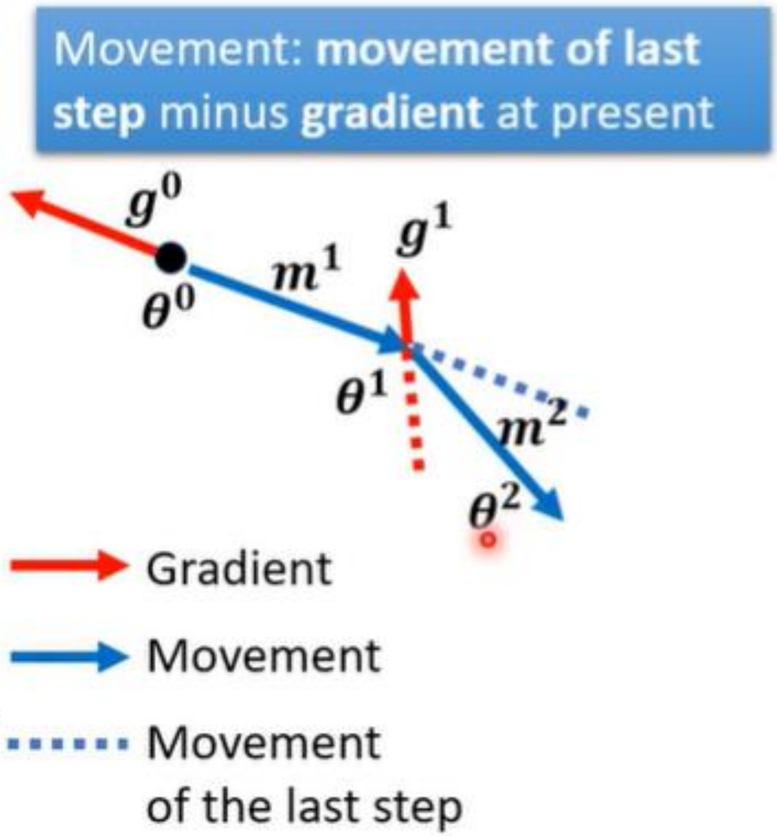
Move to $\theta^1 = \theta^0 - \eta g^0$

Compute gradient g^1

Move to $\theta^2 = \theta^1 - \eta g^1$

⋮

Momentum



Starting at $\boldsymbol{\theta}^0$

Movement $\boldsymbol{m}^0 = \mathbf{0}$

Compute gradient $\boldsymbol{g}^0$

Movement $\boldsymbol{m}^1 = \lambda \boldsymbol{m}^0 - \eta \boldsymbol{g}^0$

Move to $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 + \boldsymbol{m}^1$

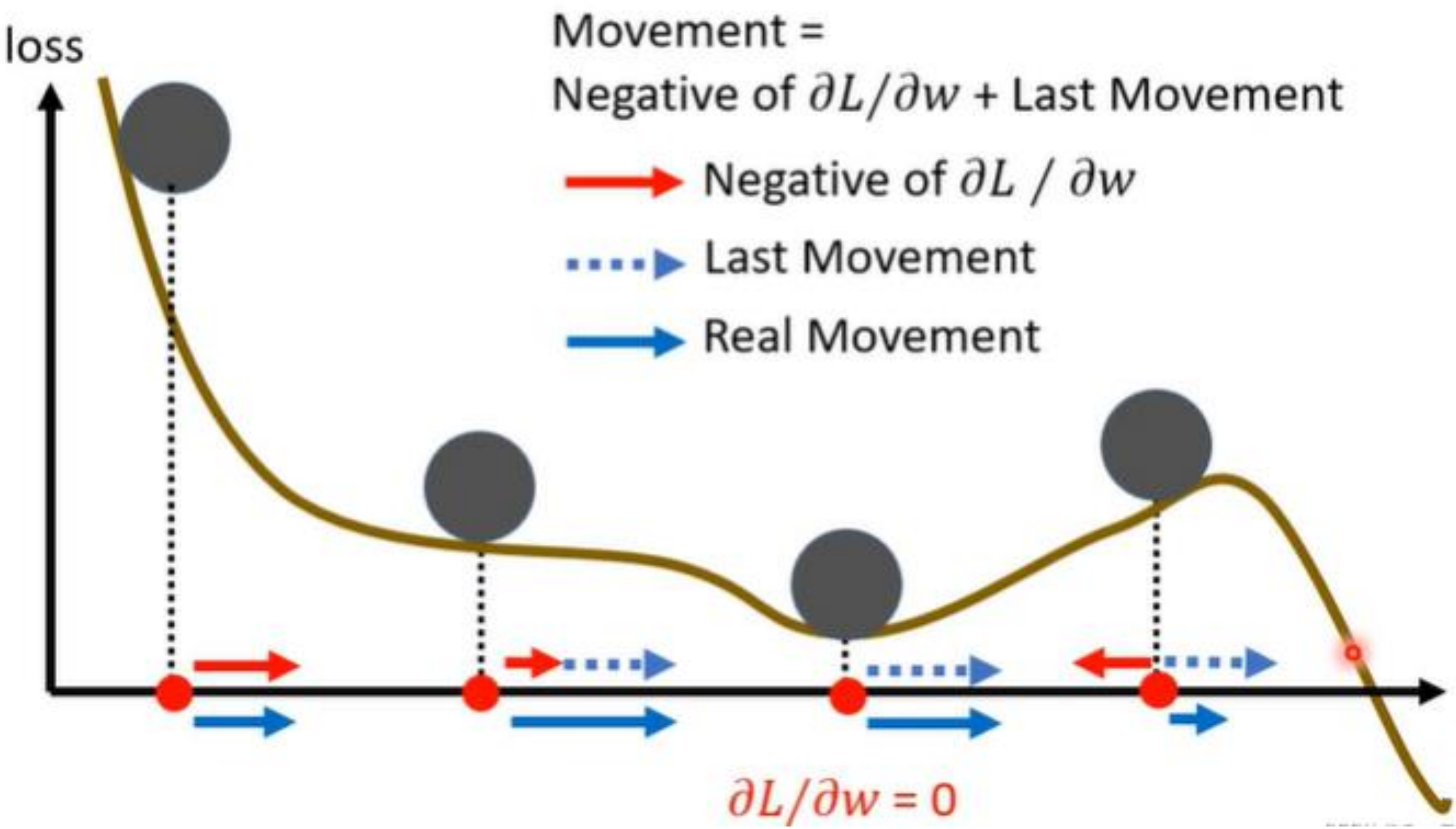
Compute gradient $\boldsymbol{g}^1$

Movement $\boldsymbol{m}^2 = \lambda \boldsymbol{m}^1 - \eta \boldsymbol{g}^1$

Move to $\boldsymbol{\theta}^2 = \boldsymbol{\theta}^1 + \boldsymbol{m}^2$

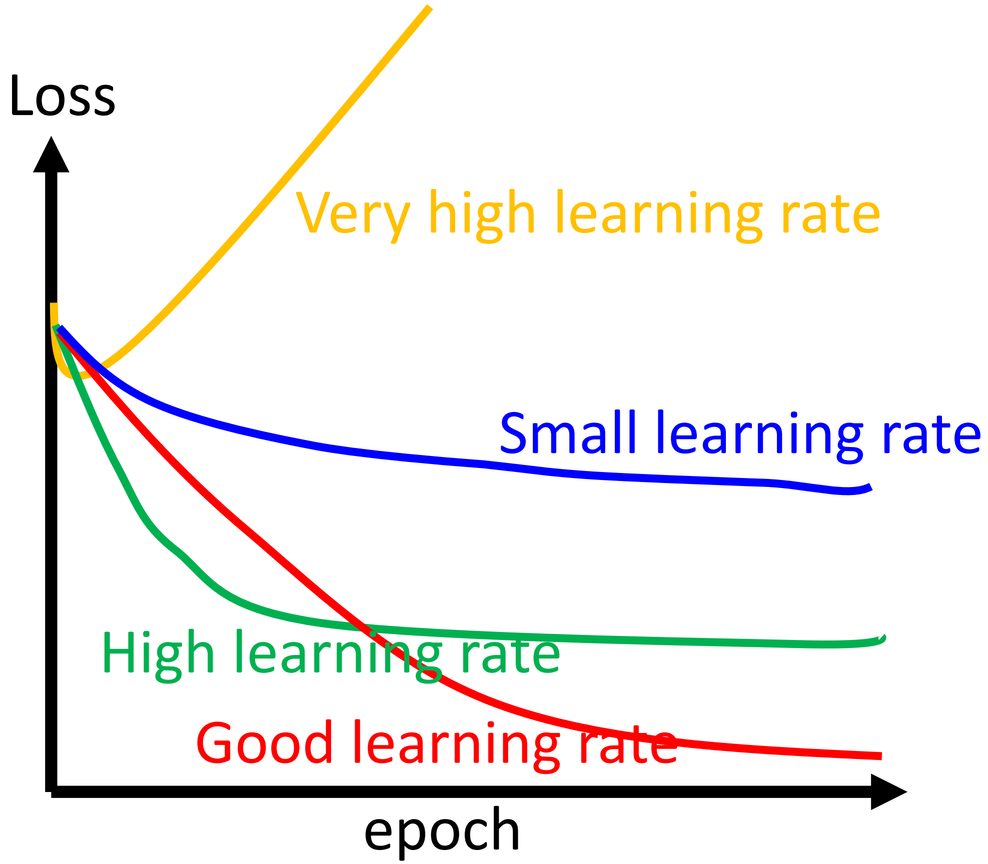
Movement not just based on gradient, but previous movement.

Momentum



loss
Movement =
Negative of $\partial L/\partial w$ + Last Movement
Negative of $\partial L / \partial w$
Last Movement
Real Movement
$\partial L/\partial w = 0$

The effects of different learning rates



- SGD usually require decaying learning rate:

$$\eta_t = \frac{1}{T}$$

Adagrad

- Divide the learning rate of each parameter by the ***root mean square of its previous deviation***

  - ***Vanilla Gradient descent***
    - $g^t = \nabla_\theta \left(\frac{1}{|S|} \sum_{(x_i, y_i) \in S} \ell(f(x_i; \theta^t), y_i) \right)$
    - $\theta^{t+1} = \theta^t - \eta_t g^t$
  - ***Adagrad***
    - $r = r + g^t \odot g^t$
    - $\theta^{t+1} = \theta^t - \eta_t \frac{1}{\delta + \sqrt{r}} \odot g^t$

RMSProp

Adagrad

$$r = r + g^t \odot g^t$$

$$\theta^{t+1} = \theta^t - \eta_t \frac{1}{\delta + \sqrt{r}} \odot g^t$$

RMSProp

$$r = \alpha r + (1 - \alpha) g^t \odot g^t$$

$$\theta^{t+1} = \theta^t - \eta_t \frac{1}{\delta + \sqrt{r}} \odot g^t$$

Adam

RMSProp

$$r = \alpha r + (1 - \alpha) g^t \odot g^t$$

$$\theta^{t+1} = \theta^t - \eta_t \frac{1}{\delta + \sqrt{r}} \odot g^t$$

Adam: RMSProp+Momuntem

$$r = \alpha r + (1 - \alpha) g^t \odot g^t$$

$$v = \rho v - \eta_t \frac{1}{\delta + \sqrt{r}} \odot g^t$$

$$\theta^{t+1} = \theta^t + v$$

Convergence of Adam

- Adam may not convergent in some special cases!

- Many provable variants of Adam:
  - Amsgrad
  - Adashift
  - …

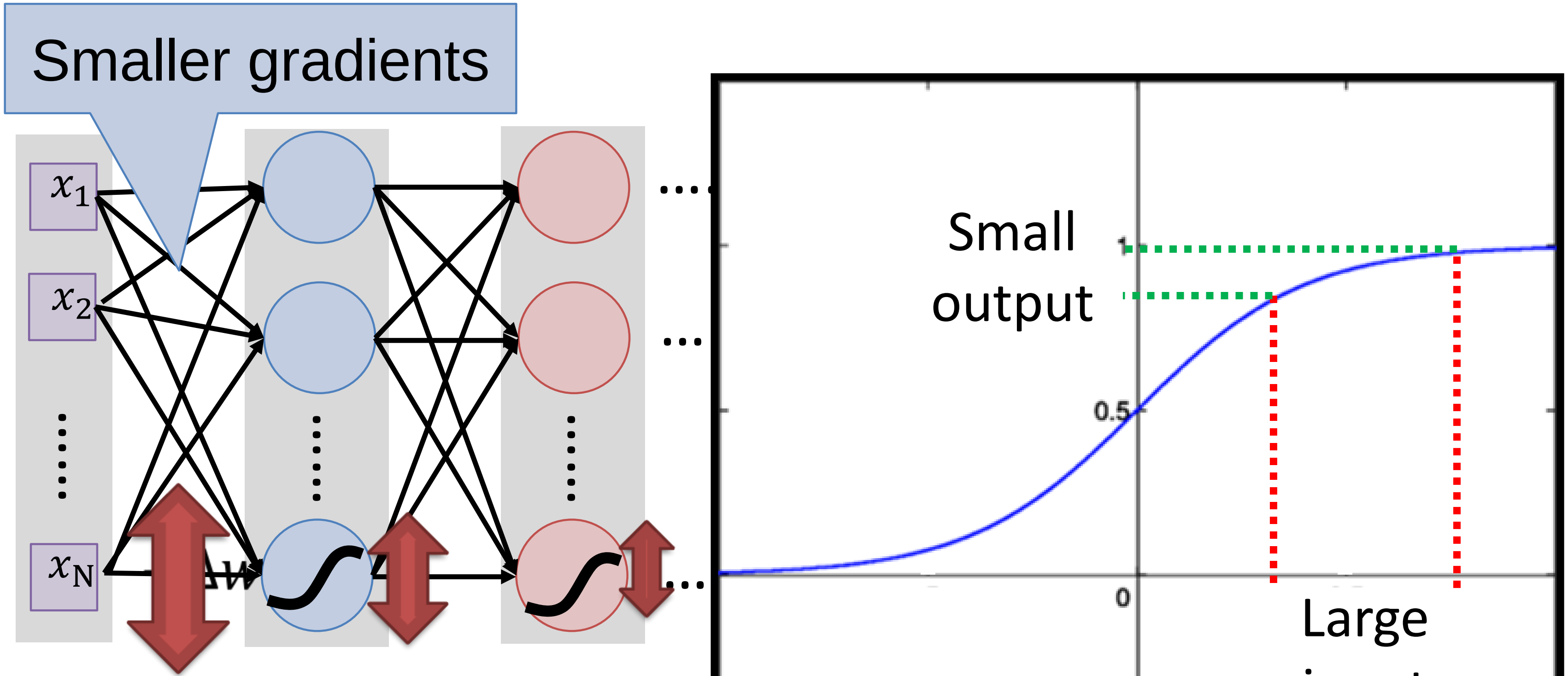
Adam vs SGD

- Adam
  - Faster convergence in practice
  - Not sensitive to the learning rate
  - Does not perform well on image classification tasks
- SGD
  - Usually slower than Adam
  - Require fine tune of learning rate
  - Has better generalization performance

Outline

- Model selection
- Optimization
- Not work well on training data
- Not work well on testing data
- Hyperparameter tuning

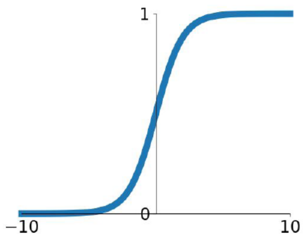
Vanishing Gradient Problem



Activation Functions

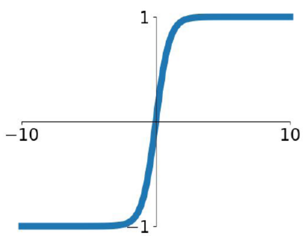
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



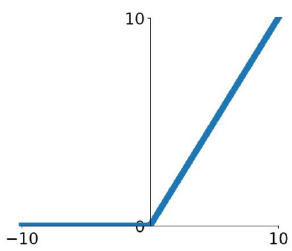
tanh

$\tanh(x)$



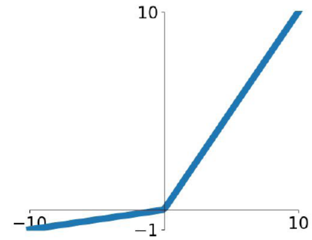
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

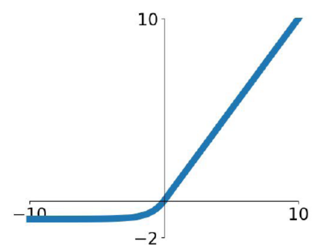


Maxout

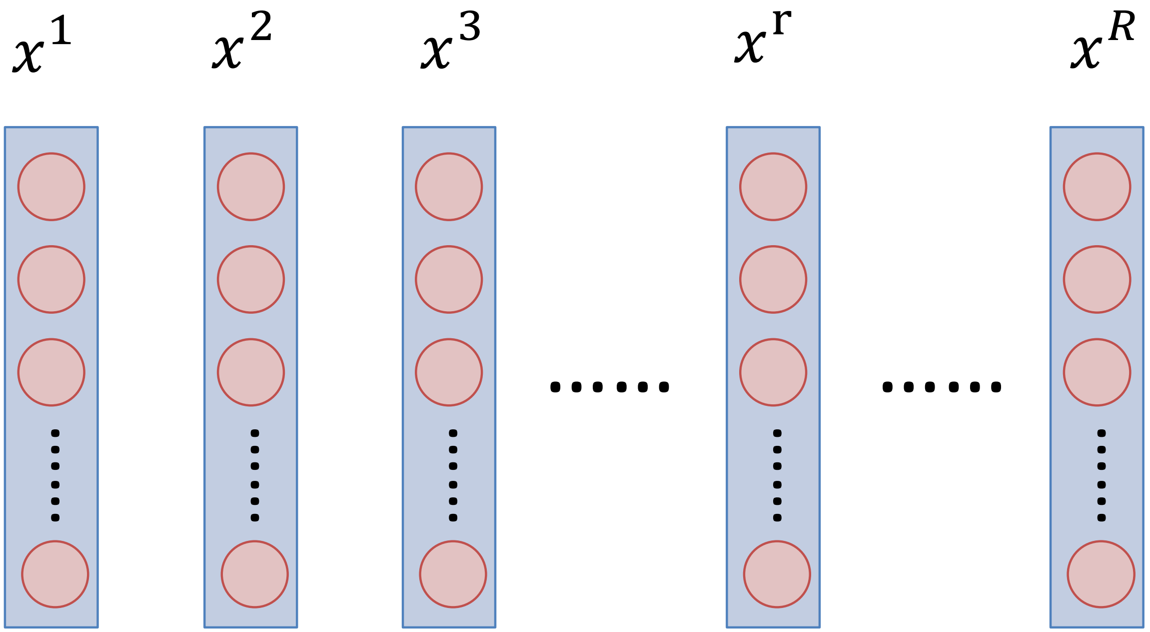
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Feature Scaling



Mean: m_i
Standard deviation: σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

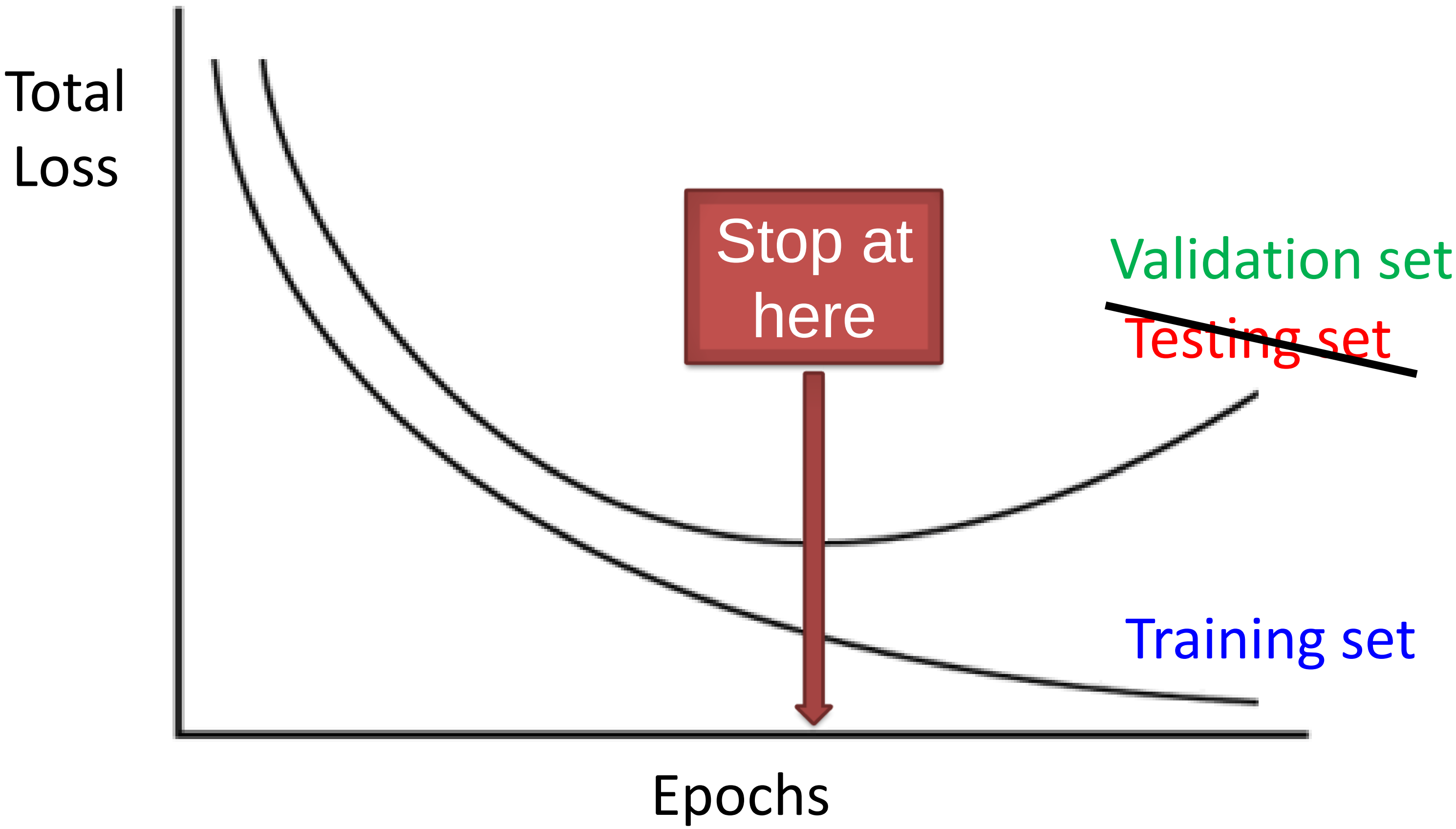
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe and Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015

Outline

- Model selection
- Optimization
- Not work well on training data
- <u>Not work well on testing data</u>
- Hyperparameter tuning

Early stopping



Regularization

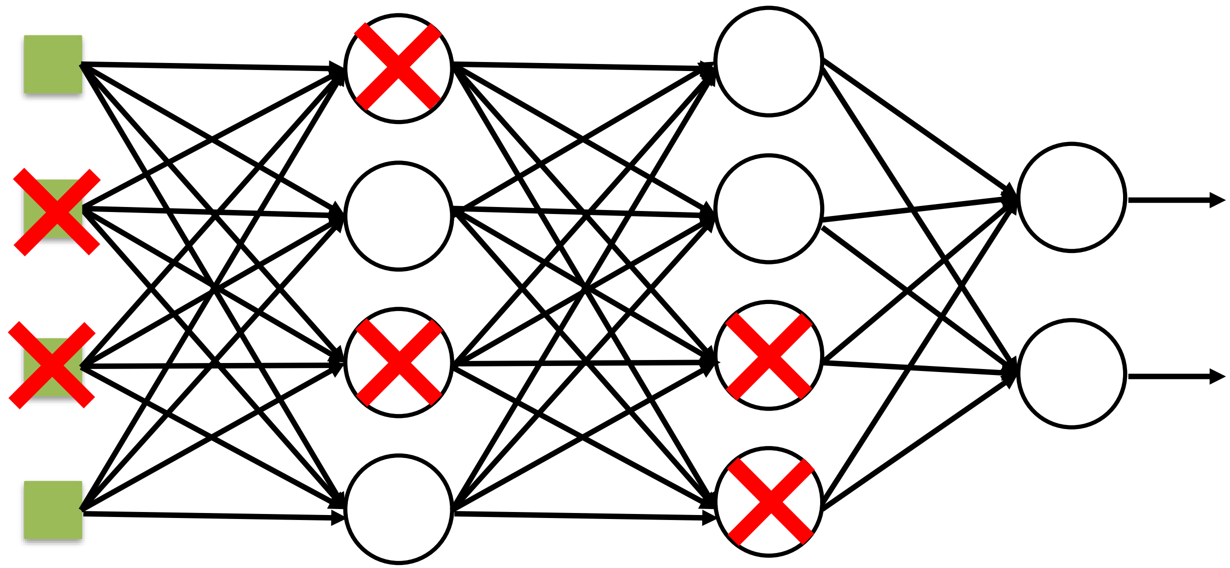
- New loss function to be minimized

- Find a set of weight not only minimizing the original cost but also close to zero

$$\mathrm{L}'(\theta) = \underbrace{L(\theta)}_{\text{Original loss}} + \lambda \underbrace{R(\theta)}_{\text{Regularization term}}$$

Dropout

Training:

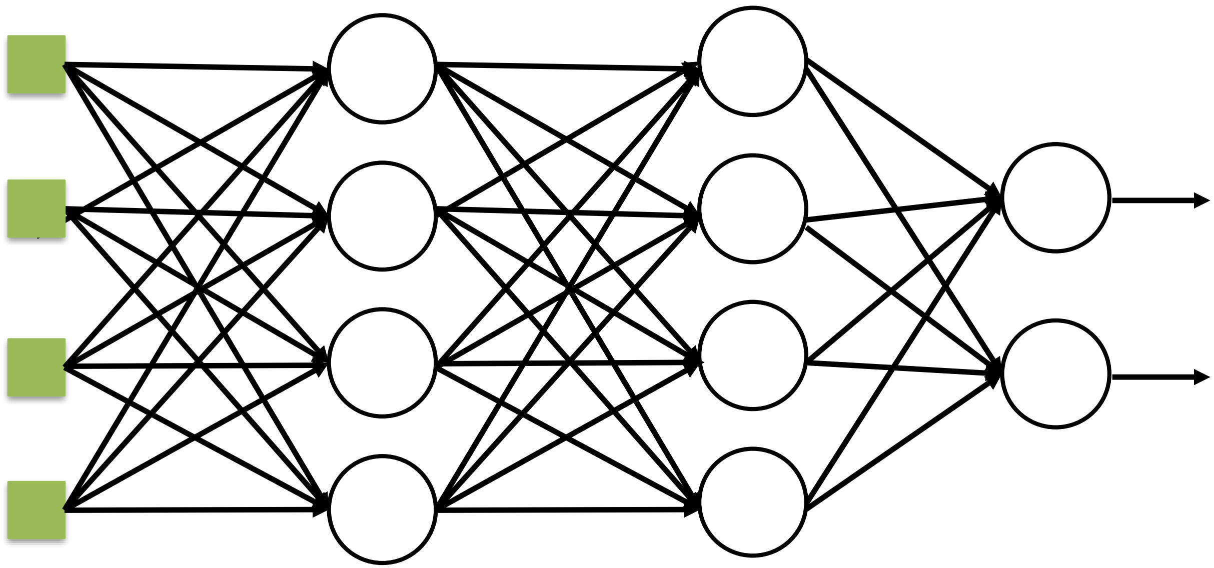


- Each time before updating the parameters
  - Each neuron has a probability of p to be dropped

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

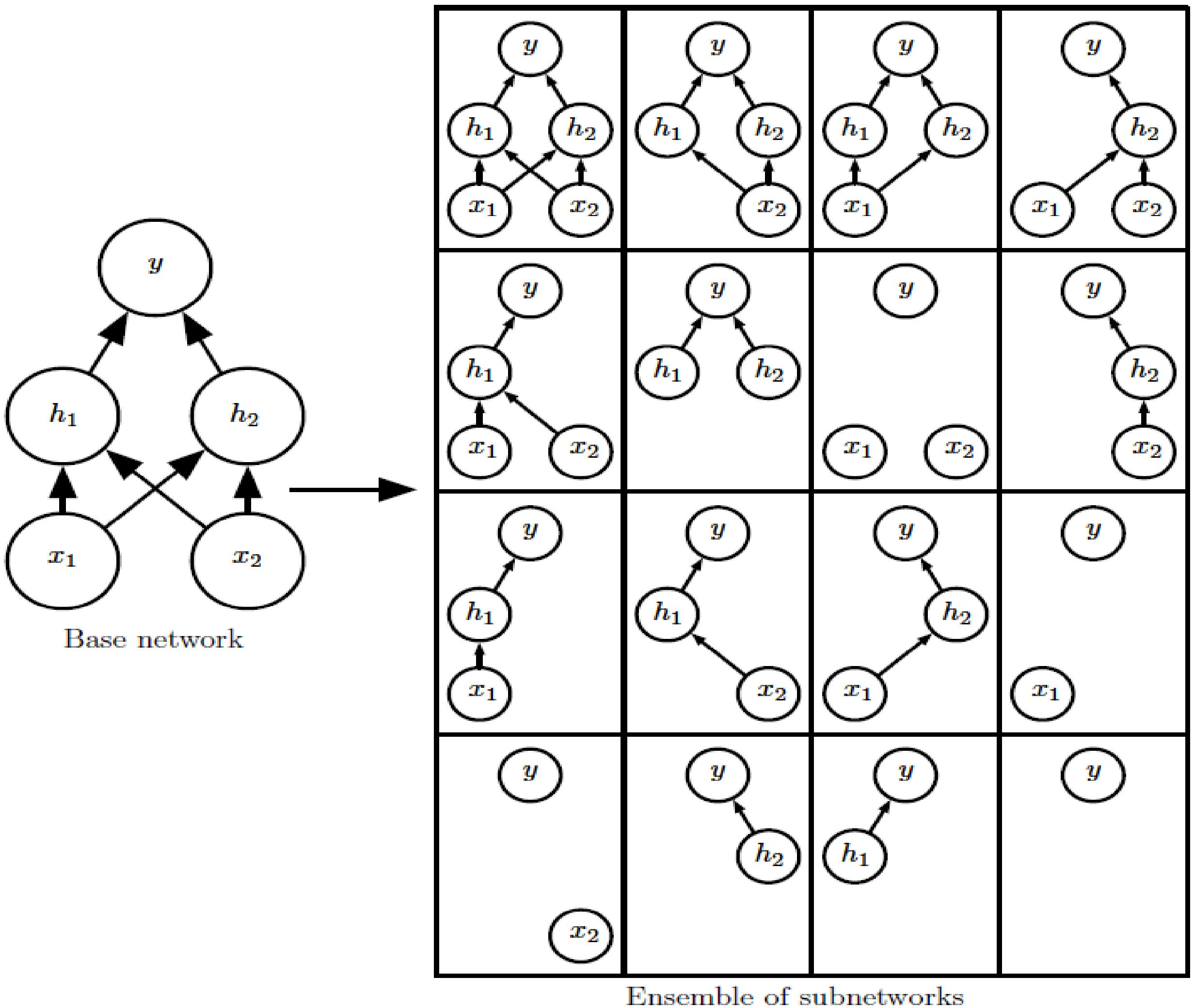
Dropout

Testing:



- No dropout during testing
  - If the dropout rate at training is p, all the weights times $1 - p$
  - Usually we choose $p = 0.5$

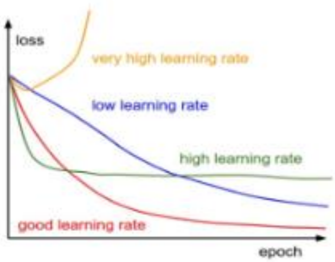
Why dropout performs well?



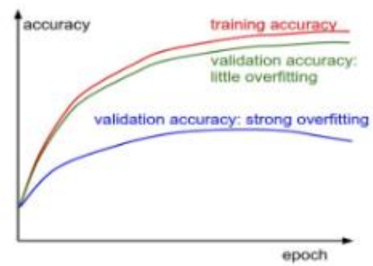
Hyperparameter tuning

- Baby sitting

Act/Grad/Filter

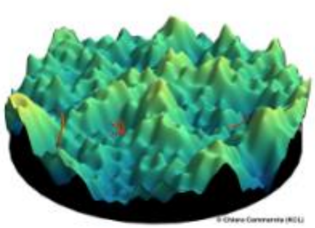


Loss



Metric

Data

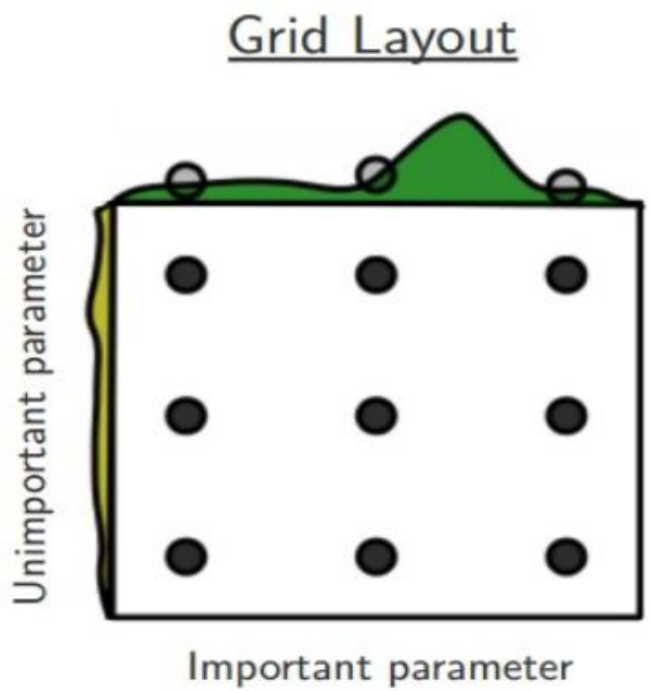


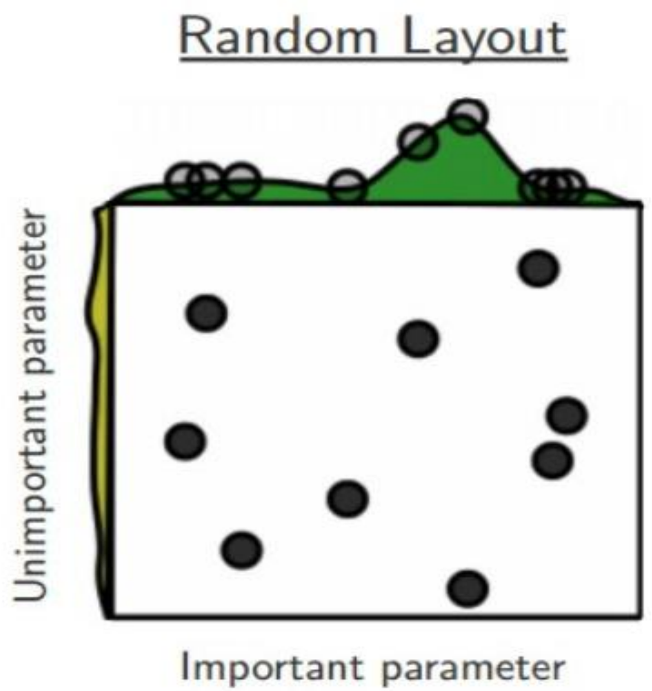
Space

Grid Search

- Workflow:
  - Define a grid on *n* dimensions, where each of these maps is for an hyperparameter.
    - e.g. *n* = (learning_rate, dropout_rate, batch_size)
  - For each dimension, define the range of possible values:
    - e.g. batch_size = [8, 16, 32, 64, 128, 256]
  - Search for all the possible configurations and wait for the results to establish the best one:
    - e.g. C1 = (0.1, 0.3, 8) -> acc = 92%, C2 = (0.1, 0.35, 8) -> acc = 92.3%, etc...

Grid Search vs Random Search





Bad on high dimensional spaces

It doesn’t guarantee to find the best hyperparameters

Good on high dimensional spaces

Give better results in less iterations

Summary

- Supervised Learning: underfitting & overfitting
- Model selection
- Optimization
- Not work well on training data
- Not work well on testing data
- Hyperparameter tuning

Questions