Deep Learning
lecture 3
Supervised Learning (2)

Yi Wu, IIIS

Spring 2025

Mar-3

Overview of Lecture 2

- Learning a multi-layer perceptron
  - Backpropagation for efficient gradient descent
    - Forward pass
    - Backward pass
    - Differentiable layers
    - Learning rate
  - Basic components
    - Linear layer
    - Softmax layer and cross-entropy loss
    - Activation
      - Gradient vanishing issue and activation function design
      - Subgradients

Overview of Lecture 2

- Convolutional Neural Network
  - Scanning for shift invariance
  - The convolution filter
    - Recursive scanning
    - Fewer parameters and larger receptive field
  - Pooling for jittering
  - Padding for retaining output size
  - 1D convolution
    - Time delay neural network or temporal CNN

Today’s Lecture

- Get your hands more dirty!

- Part 1: design a better learning algorithm
  - More tricks to play with gradients
- Part 2: more tricks for practical classification
  - Start to get professional in tuning!
- Part 3: advanced architectures
- Part 4: cloud computing tutorial

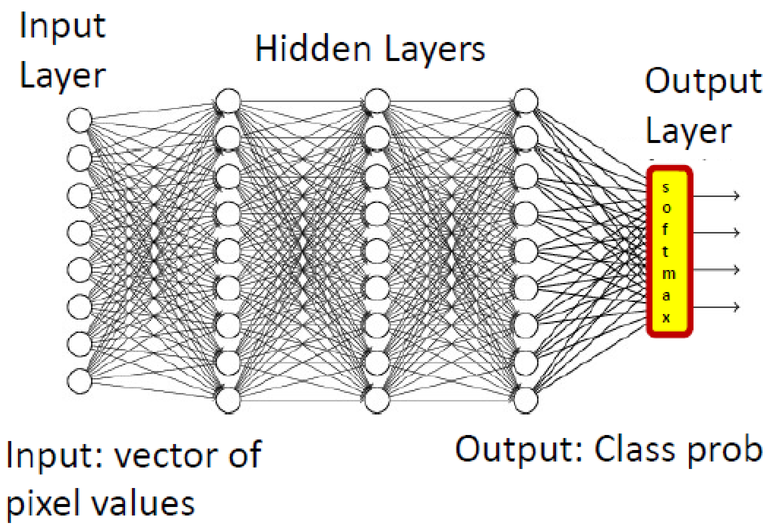
Today’s Lecture

- Get your hands more dirty!

- Part 1: design a better learning algorithm
  - More tricks to play with gradients
- Part 2: more tricks for practical classification
  - Start to get professional in tuning!
- Part 3: advanced architectures
- Part 4: cloud computing tutorial

Recap: Training a Neural Network

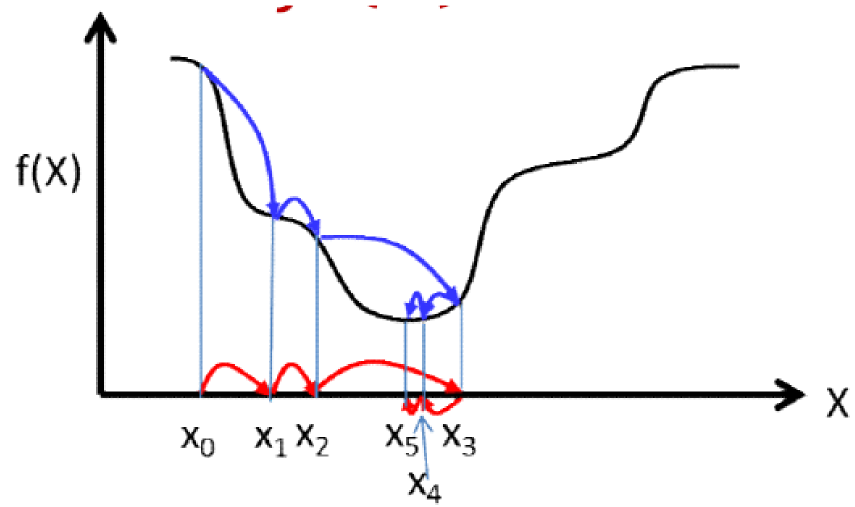
- Problem Statement
  - Given training data $X = \{(x^i, y^i)\}$
  - Design a neural network $y = f(x; \theta)$
  - Loss function $L(\theta) = \frac{1}{N}\sum_i err(f(x^i; \theta); y^i)$
  - Goal: minimize $L(\theta)$ w.r.t. θ



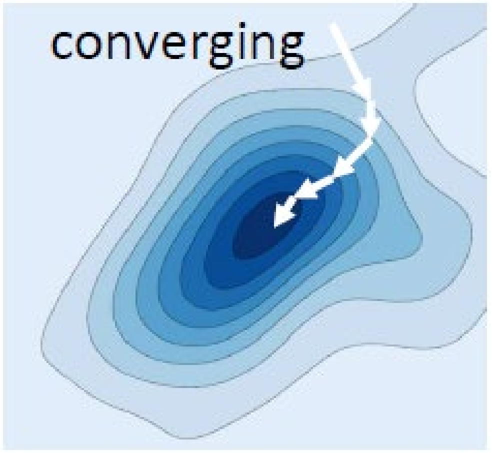
- Non-Convex Optimization!

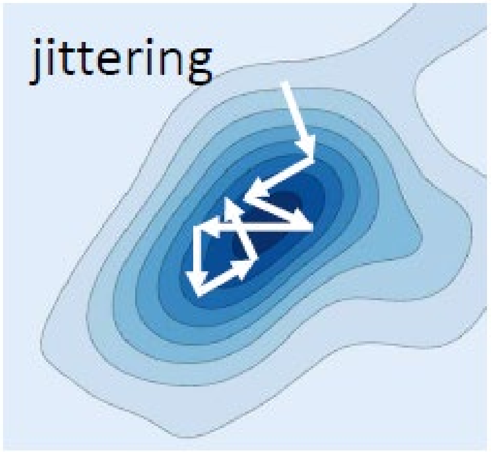
Recap: Training a Neural Network

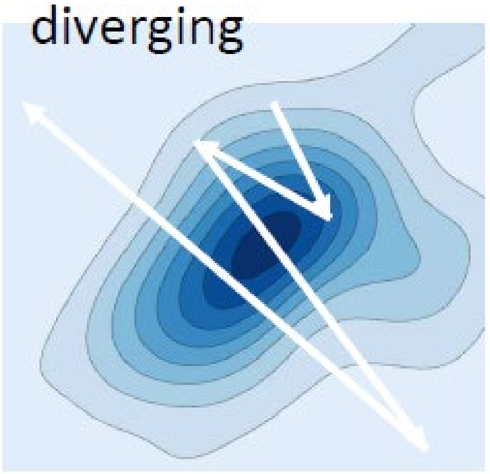
- The Gradient Descent Algorithm for $f(X)$
  - Choose X^0
  - $X^{k+1} = X^k - \eta^k \nabla_X f(X^k)$
  - Convergence: $\left|f(X^{k+1}) - f(X^k)\right| < \epsilon$



- Learning rate η^k is critical!





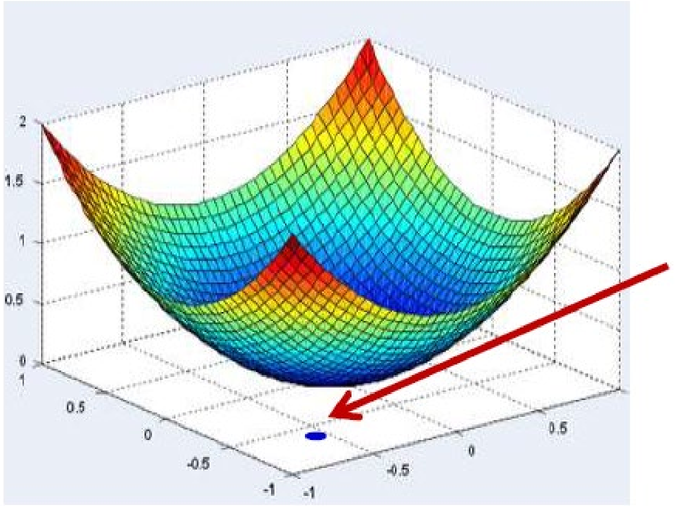


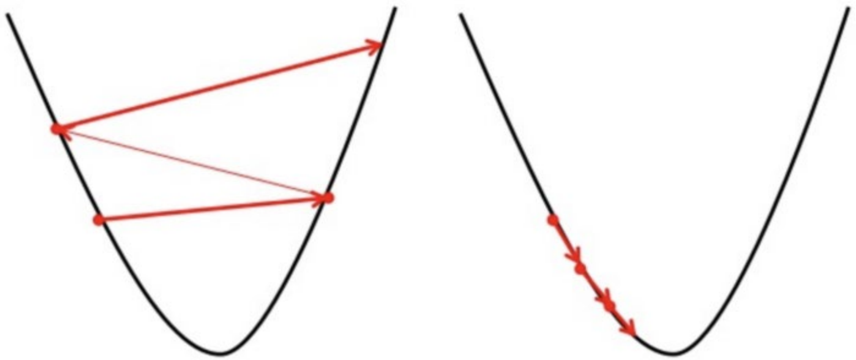
Convex Optimization

- How to set the learning rate η?
  - Get inspired from the convex setting
    - Global optimum <-> $\nabla f(x) = 0$

- Convex Function $f(x)$
  - $f(x+y) \leq \frac{1}{2}\left(f(x) + f(y)\right)$
  - $\nabla^2 f(x) \succcurlyeq 0$

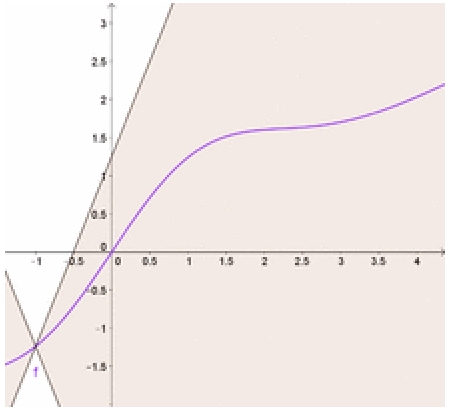
- Intuition: small learning rate
  - What's the threshold?





Convex Optimization

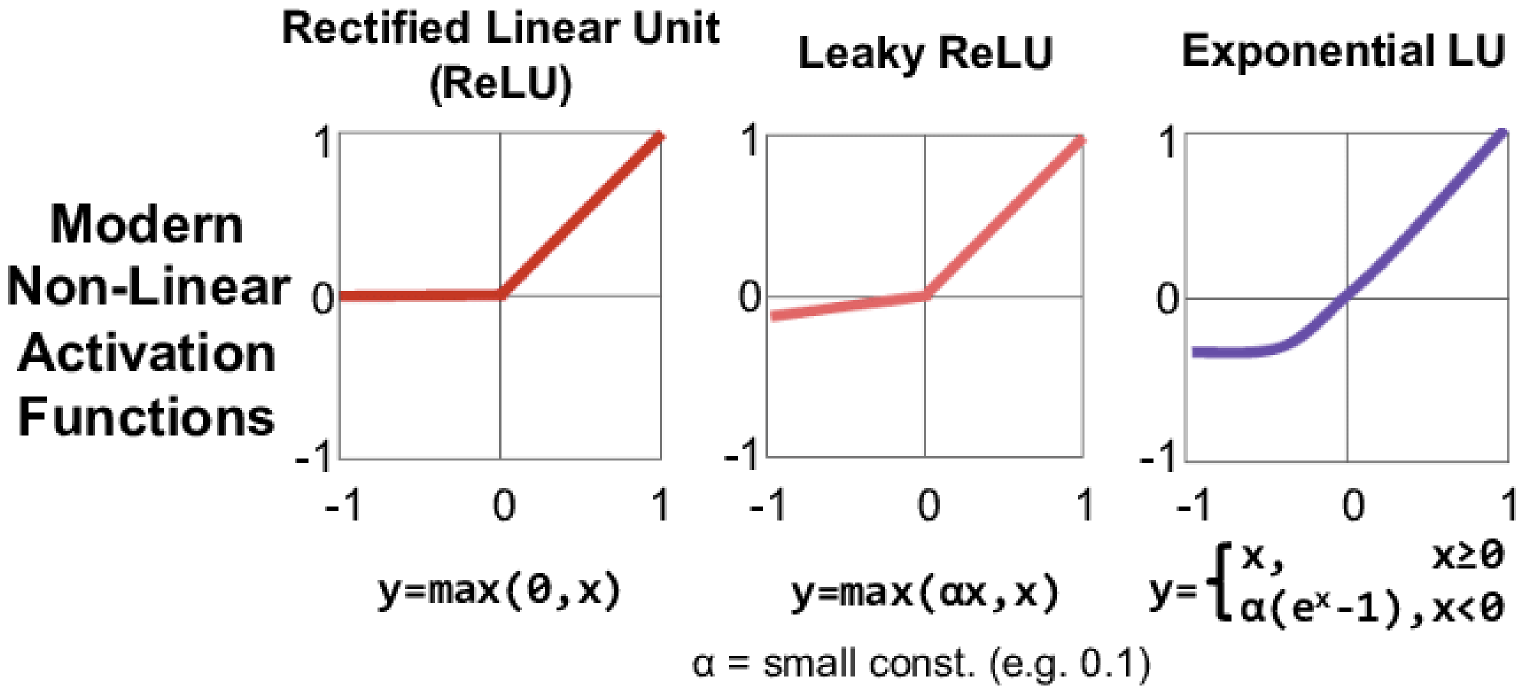
- Definition: *Lipschitz continuous*
  - *A function* $g(x)$ *is Lipschitz continuous:* $|g(x) - g(y)| \leq L|x - y|$



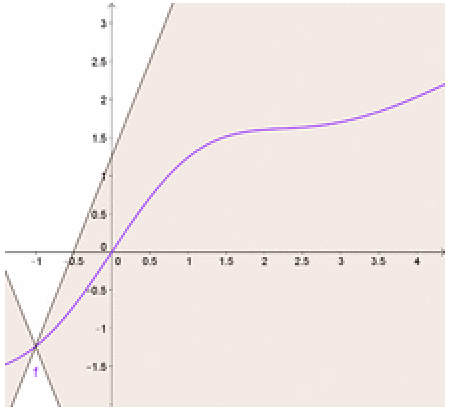
- Assumption: a "smooth" convex function $f(x)$
  - $f(x)$ is convex
  - Gradient of $f(x)$ is *Lipschitz continuous:* $|\nabla f(x) - \nabla f(y)| \leq L|x - y|$
  - "Gradient can't change arbitrarily fast"
    - $\nabla^2 f(x) \preccurlyeq LI$
  - A reasonably weak assumption
    - Machine learning, neural networks, etc

Convex Optimization

- [



$L|x-y|$

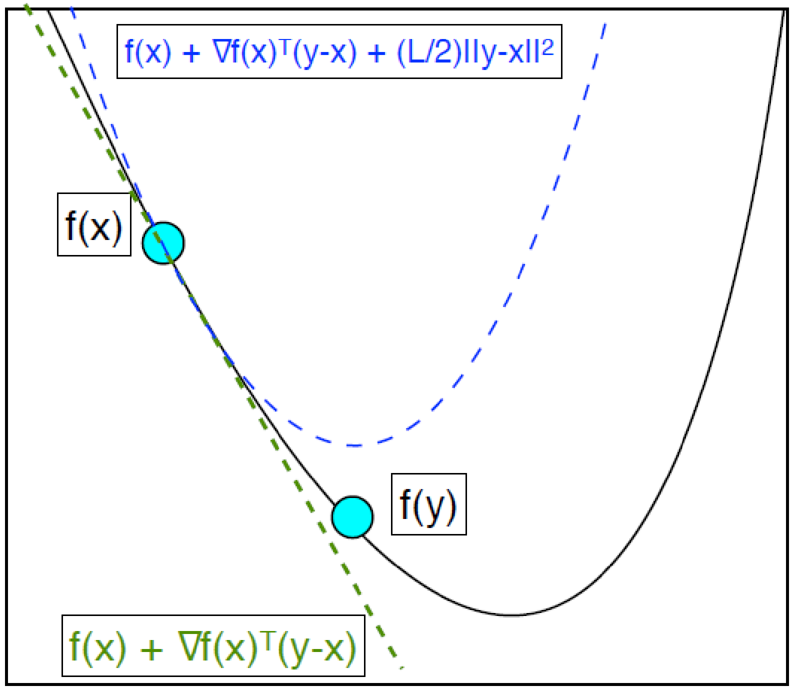


$| \leq L|f(x)-f(y)|$

- A reasonably weak assumption
  - Machine learning, neural networks, etc
  - Remark: ReLU v.s. ELU

Convex Optimization

- Descent Lemma
  - $f(y) \le f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} |x - y|^2$
    - Prove it in your homework ☺
- A convex quadratic upper bound on $f(x)$
  - Minimize the upper bound of y
  - $\eta = \frac{1}{L}$ (optimal)
  - Remark: any $0 < \eta < \frac{2}{L}$, decreases $f(x)$



Convex Optimization

- Convergence Rate with constant learning rate $\eta = \frac{1}{L}$
  - $X^{k+1} = X^k - \frac{1}{L}\nabla_X f\left(X^k\right)$
  - $f\left(X^{k+1}\right) \leq f\left(X^k\right) - \frac{1}{2L}\left|\nabla f\left(X^k\right)\right|^2$ (decent lemma)
  - $\left|\nabla f\left(X^k\right)\right|^2 \leq 2L\left(f\left(X^k\right) - f\left(X^{k+1}\right)\right)$ (progress bound)
  - $k \min_{i=0\ldots k}\left\{\left|\nabla f\left(X^i\right)\right|^2\right\} \leq 2L\left(f(X^0) - f(X^*)\right)$
- For X^k with $\left|\nabla f\left(X^k\right)\right|^2 \leq \epsilon$
  - $k = O\left(\frac{1}{\epsilon}\right)$, a sublinear convergence rate
- Also hold for non-convex function
  - Saddle point or local optimum (Refer to your ML course slides!)

Convex Optimization

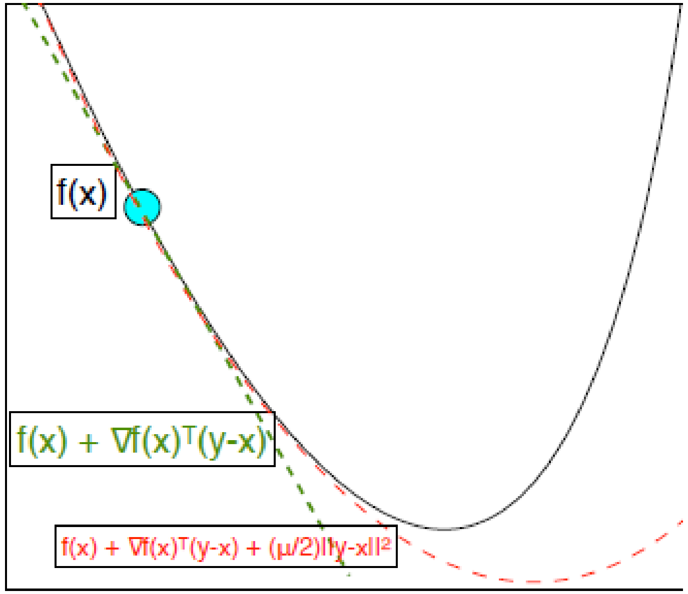
- How to estimate L ?
- Adaptive Learning Rate with Line Search
  - Start with a large learning rate η
  - Decrease if some condition unsatisfied
    - Naïve solution: ensure function value is decrease
    - Armijo condition: $f\left(w - \eta \nabla f\left(w^k\right)\right) \leq f\left(w^k\right) - \eta \cdot \gamma \left|\nabla f\left(w^k\right)\right|^2$ for $\gamma \in (0, 1/2]$

      (backtracking line-search)
    - And more (e.g., Wolfe conditions, etc, take a convex optimization course!)

- Practical Solution
  - If performance is not decreasing on the validation set, decrease learning rate by $\eta \leftarrow \alpha \cdot \eta$

Strongly Convex Functions

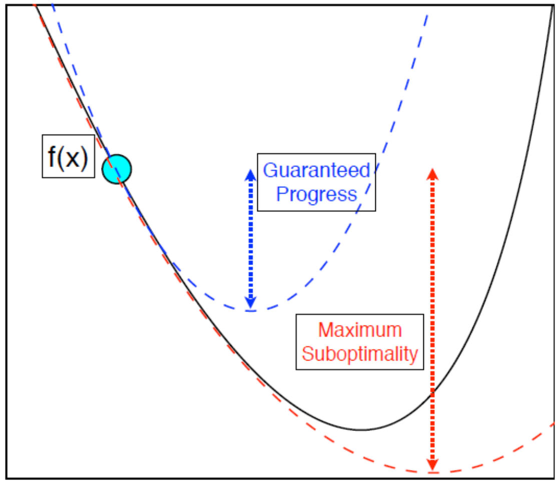
- Better results for convex functions?
  - Strongly convexity (for some $\mu > 0$)
    $$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} |y - x|^2$$
  - A quadratic lower bound!



- Better rate for L-smooth strongly convex function
  - $LI \succcurlyeq \nabla^2 f(x) \succ 0$

Prove it in your homework ☺

Strongly Convex Functions

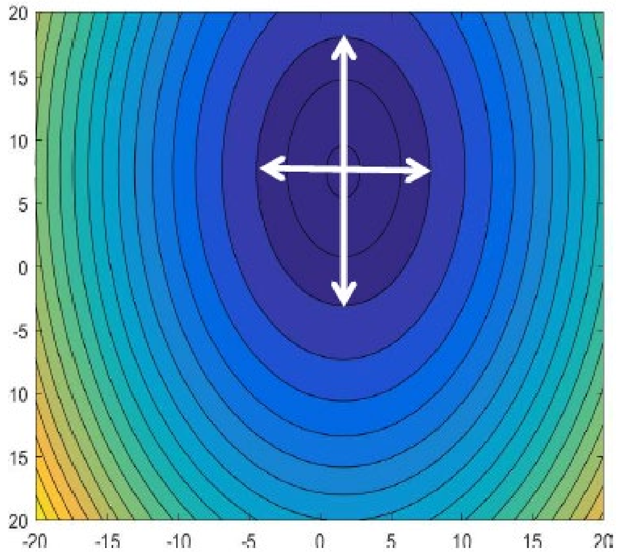


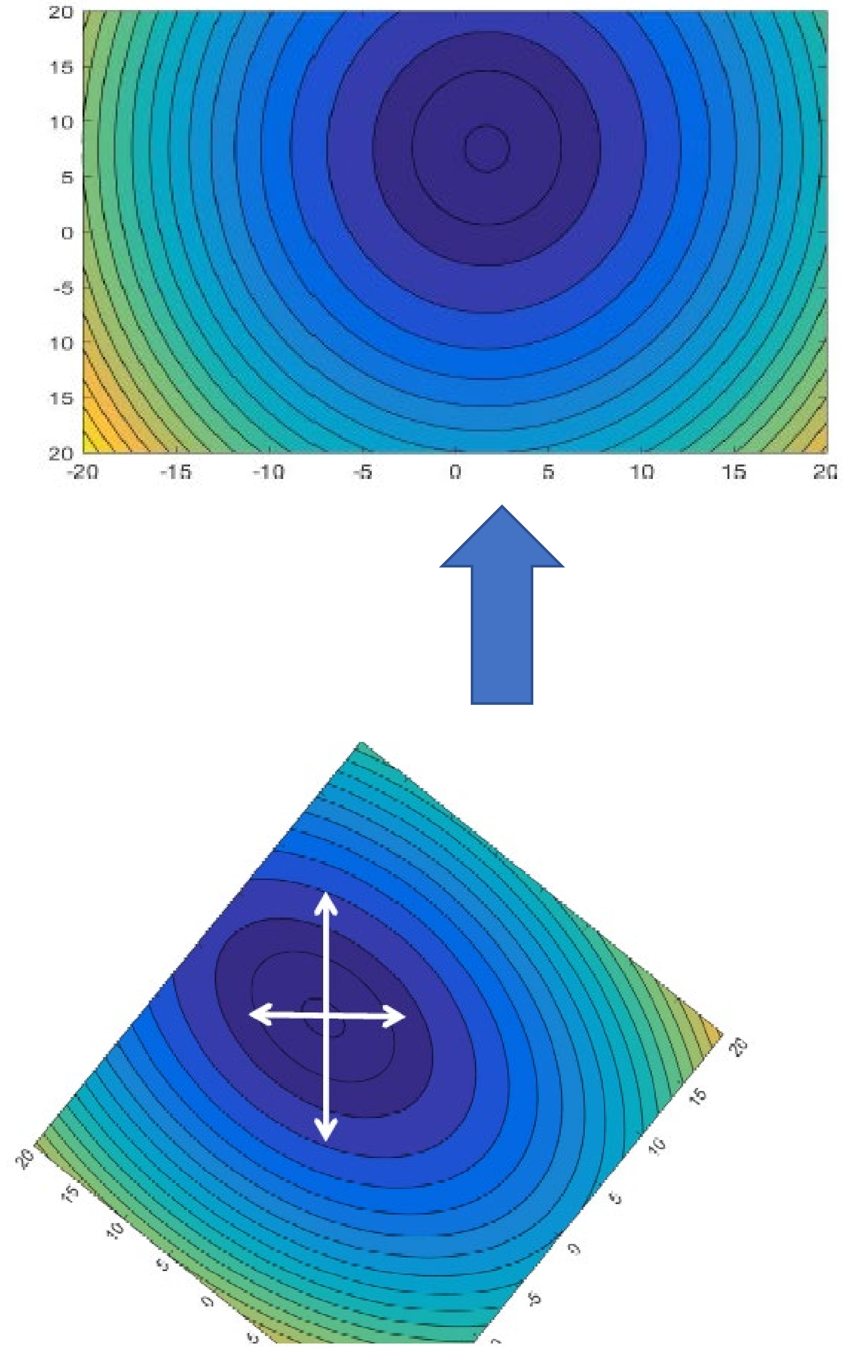
- Linear convergence on strongly convex L-smooth function
  - For $f(X^k) - f^* \leq \epsilon$, we have $k = O\left(\log\frac{1}{\epsilon}\right)$
- Strong convexity is good!
  - A quadratic lower-bound
  - Bounded sub-optimality: progress is at least a fraction of max sub-optimality

- (very unofficial) Justification for weight decay (l2-norm)
  - $f(x) \rightarrow f(x) + \alpha|x|^2$ convex → strongly convex
  - Remark: adding regularizations often gives you better analytical properties

Second-Order Optimization

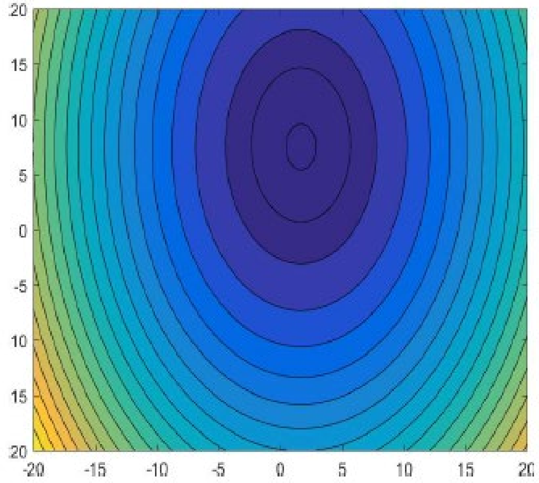
- Can we do better for strongly convex functions?
    - Recap: $X^{k+1} = X^k - \frac{1}{L}\nabla_X f\left(X^k\right)$
    - We use a fixed learning rate for all the coordinates
    - Is this optimal?
- Imagine a 2-dimensional quadratic function
    - $y_1 = ax_1^2 + bx_2^2$
        - Different optimal learning rate for each axis
    - $y_2 = ax_1^2 + bx_2^2 + cx_1x_2$
    - $y = \frac{1}{2}X^TAX + bX + c$
    - $y^* = \frac{1}{2}\hat{X}^T\hat{X} + b\hat{X} + c$





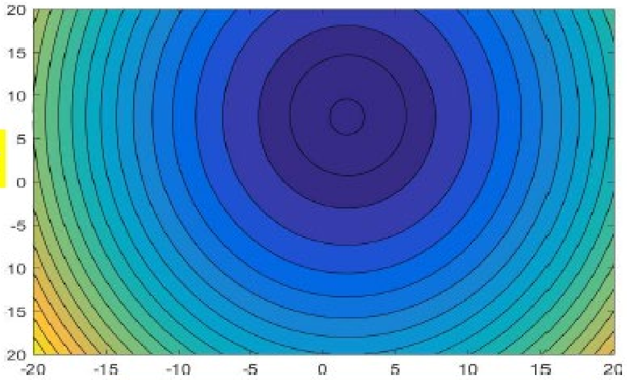
Second-Order Optimization

- Scaling the axis (quadratic function case)
  - $f(X) = \frac{1}{2}X^T AX + bX + c$ with $A \succ 0$
  - $\hat{X} = A^{0.5}X$
- Gradient descent on $f(\hat{X})$
  - $\hat{X}^{k+1} = \hat{X}^k - \eta\nabla f(\hat{X}^k)$
- Modified update rule
  - $X^{k+1} = X^k - \eta A^{-1}\nabla f(X^k)$
  - Newton's method



$$E = \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} + \mathbf{b}^T\mathbf{w} + c$$

$$\hat{\mathbf{w}} = \mathbf{A}^{0.5}\mathbf{w}$$

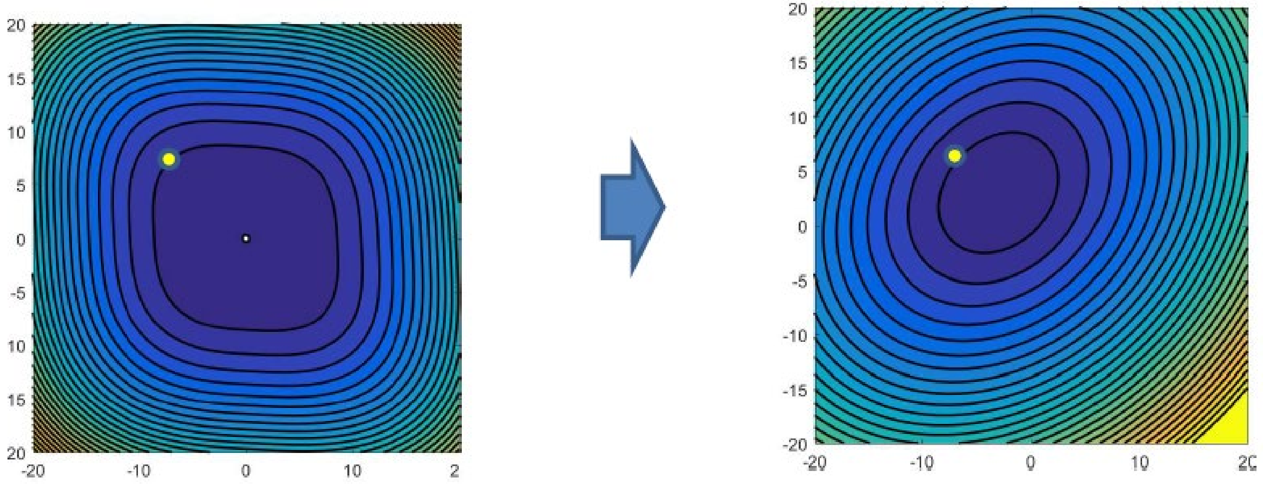


$$E = \frac{1}{2}\hat{\mathbf{w}}^T\hat{\mathbf{w}} + \hat{\mathbf{b}}^T\hat{\mathbf{w}} + c$$

Second-Order Optimization

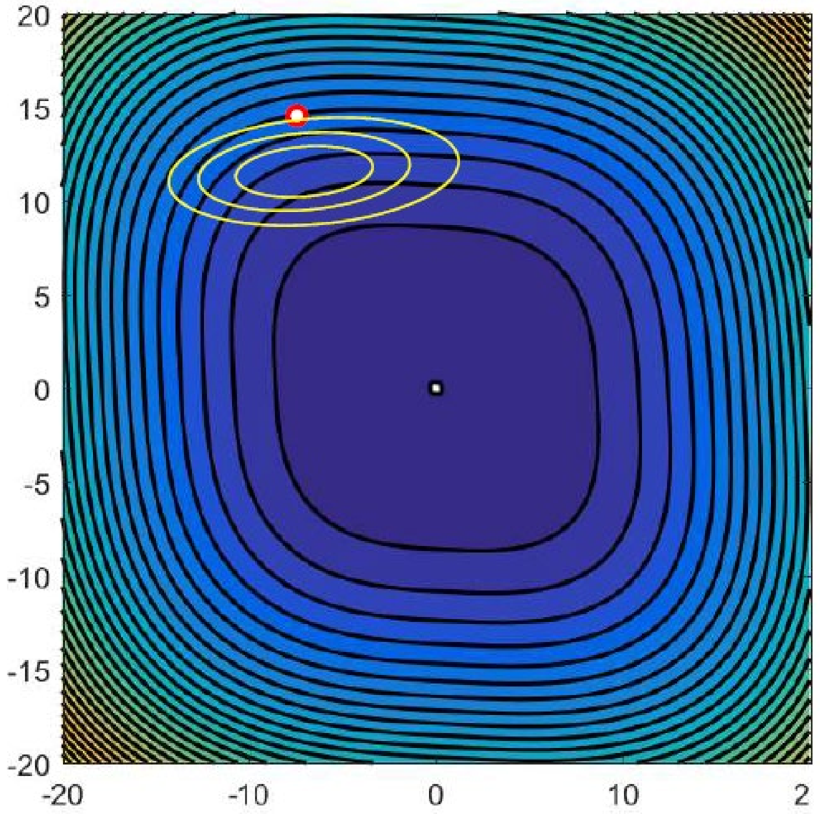
- General L-smooth strongly convex functions?
  - Generalized Newton's method
  - Taylor's expansion (progress bound)
    $$f(y) \leq f(x) + \nabla f(x)^T(y-x) + (y-x)^T \nabla^2 f(x)(y-x)$$
  - Newton's method to optimize this upper-bound
  - L-smoothness gives you an improvement guarantee!

- Second-order optimization
  - $X^{k+1} = X^k - \eta^k \nabla^2 f\left(X^k\right)^{-1} \nabla f(X^k)$



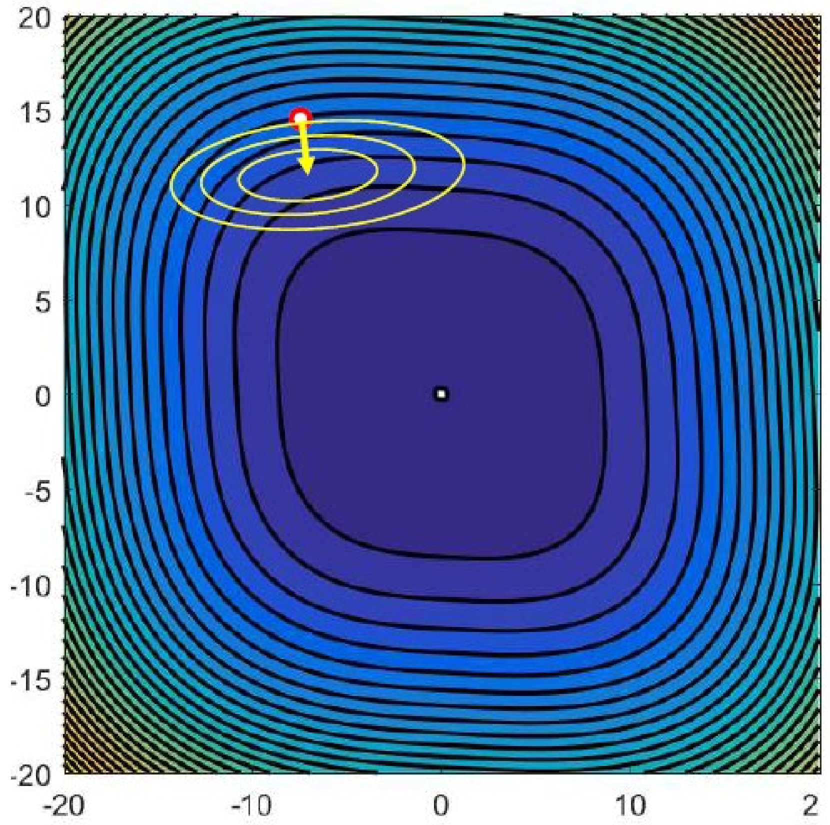
Second-Order Optimization

- Convergence (homework ☺)
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



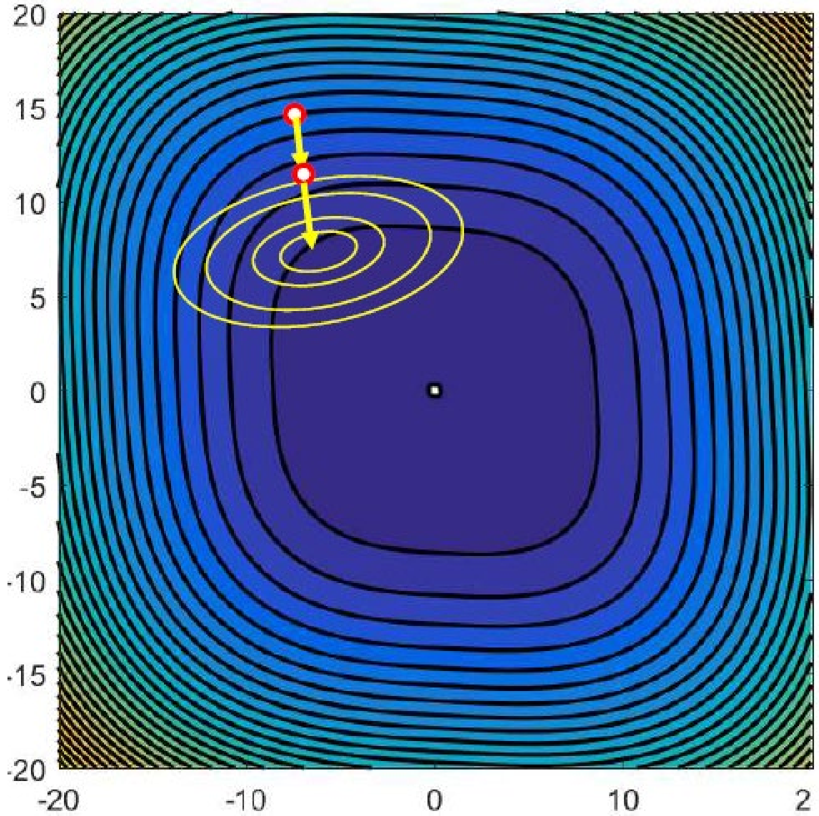
Second-Order Optimization

- Convergence
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



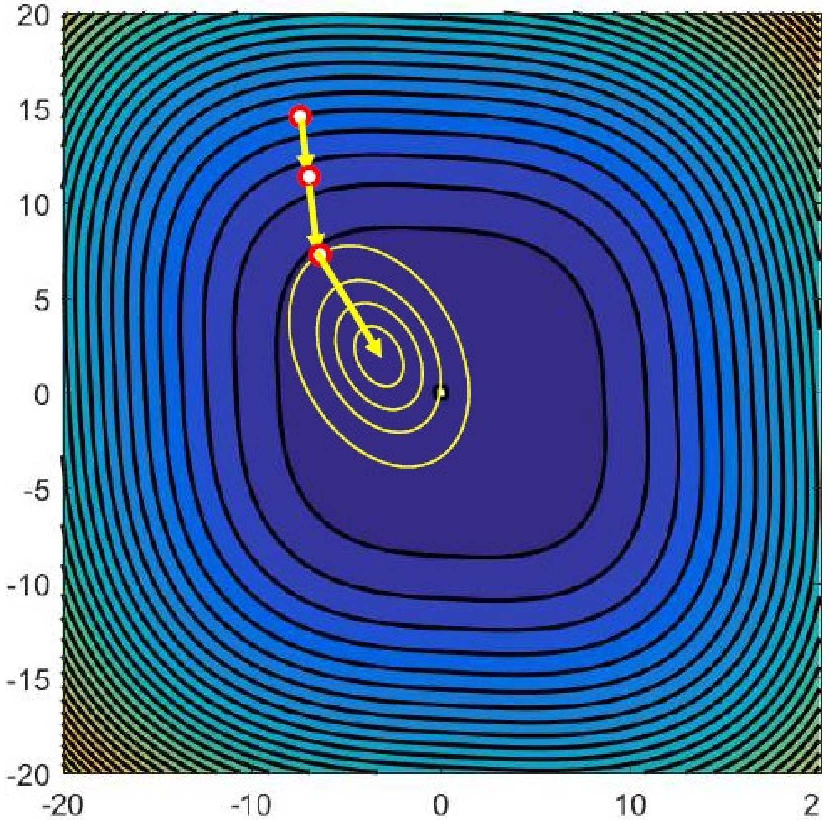
Second-Order Optimization

- Convergence
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



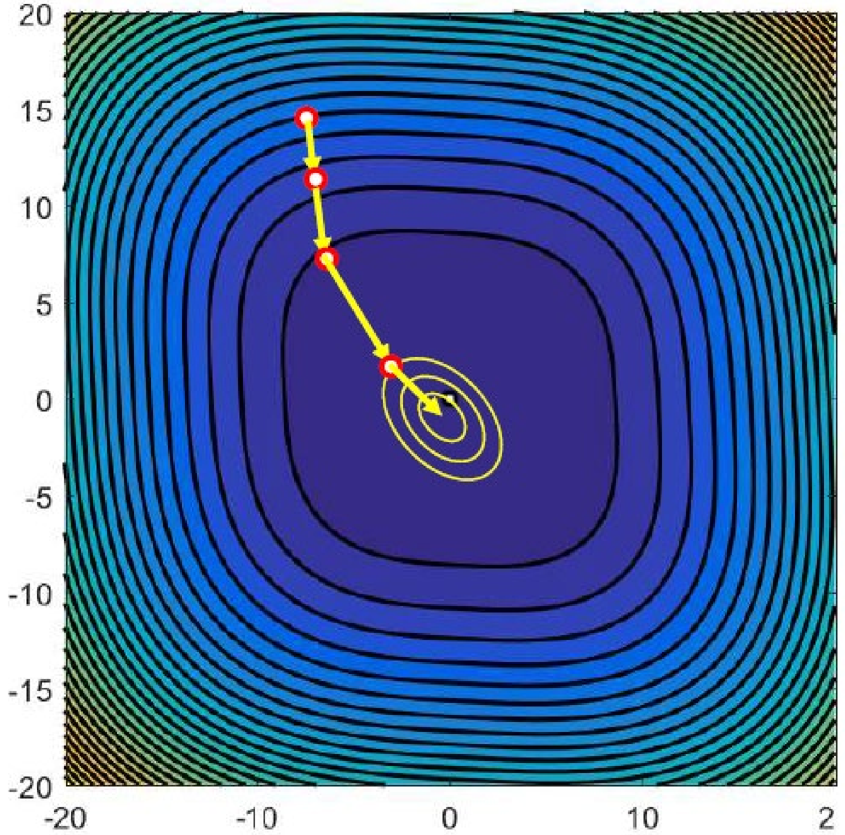
Second-Order Optimization

- Convergence
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



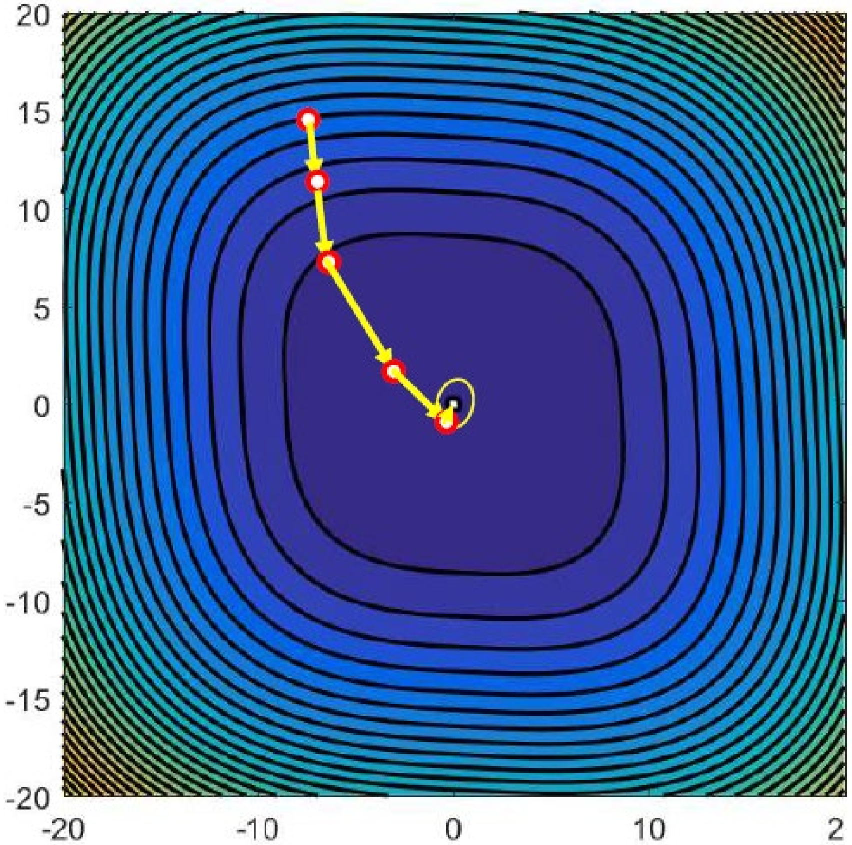
Second-Order Optimization

- Convergence
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



Second-Order Optimization

- Convergence
  - Strongly convex $f(X)$ with Lipschitz Hessian, then we have quadratic convergence
- Example:



Second-Order Optimization

- Issues with Hessian
  - Extremely expensive to compute for neural networks (quadratic number of parameters to compute)
  - Even harder to invert it
  - It can diverge for non-convex functions due to negative eigenvalues

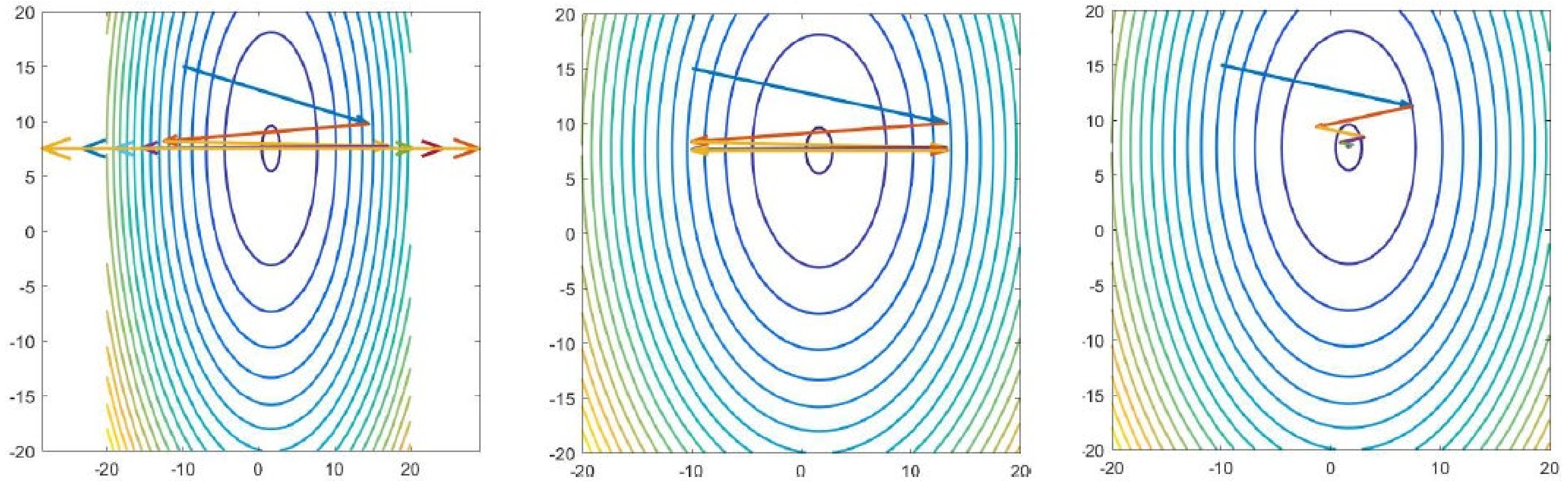
Second-Order Optimization

- Issues with Hessian
  - Extremely expensive to compute for neural networks (quadratic number of parameters to compute)
  - Even harder to invert it
  - It can diverge for non-convex functions due to negative eigenvalues

- But second-order method normalizes the axis!
  - *Many of the convergence issues arise because we force the same learning rate on all parameters*

Accelerated Methods

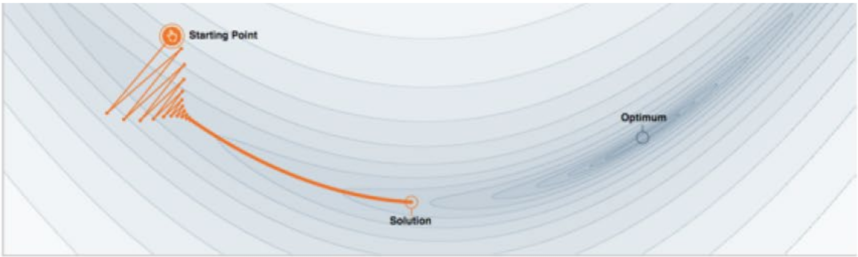
- Can we do better with first-order methods?
  - The issue of dimension-independent learning rate
  - Some dimension will be converging but some other dimensions will oscillate (and even diverge)

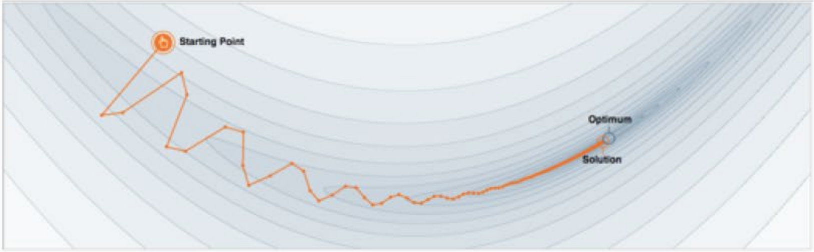


  - Goal: encouraging converging dimensions while reduce step size on the oscillating dimensions

Accelerated Methods

- The heavy-ball method
  - Maintain an running average of past gradients (typically $\beta = 0.9$)
    $$\Delta X^k = \beta \Delta X^{k-1} - \eta \nabla f(X^k)$$
  - Update with the averaged gradients instead of the current one
    $$X^{k+1} = X^k + \Delta X^k$$
  - Intuition:
    - Larger steps when gradients keeps the same direction
    - Smaller steps when gradients flip directions
  - Gradient Descent with Momentum
    $$X^{k+1} = X^k - \eta \nabla f(X^k) + \underbrace{\boxed{\beta(X^k - X^{k-1})}}_{\textbf{momentum}}$$



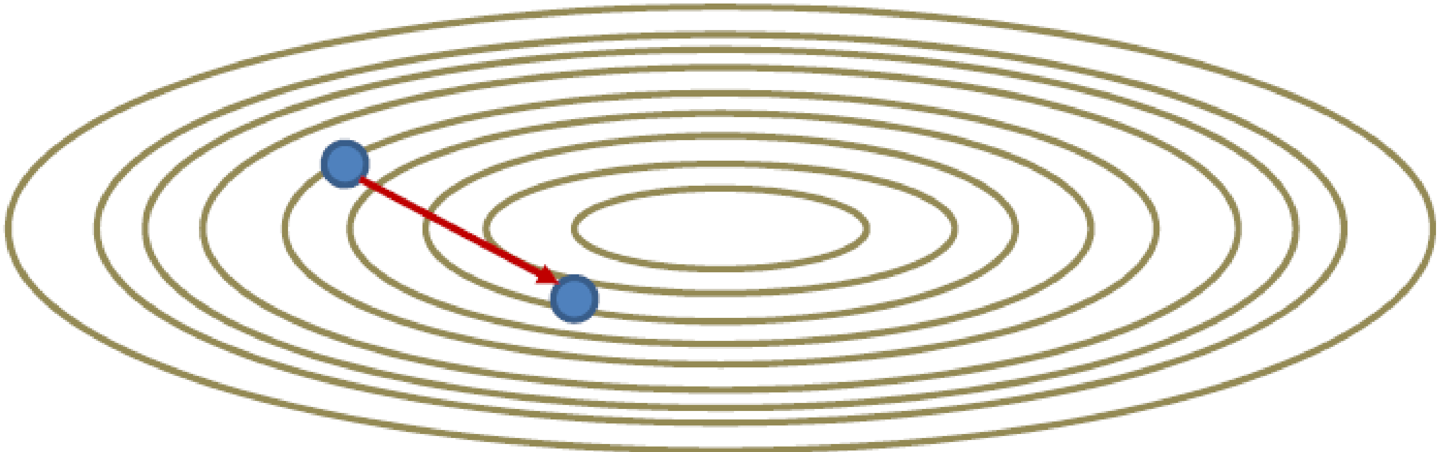


Accelerated Methods

- The momentum method

$$X^{k+1} = X^k - \eta \nabla f\left(X^k\right) + \beta\left(X^k - X^{k-1}\right)$$

  - For each iteration

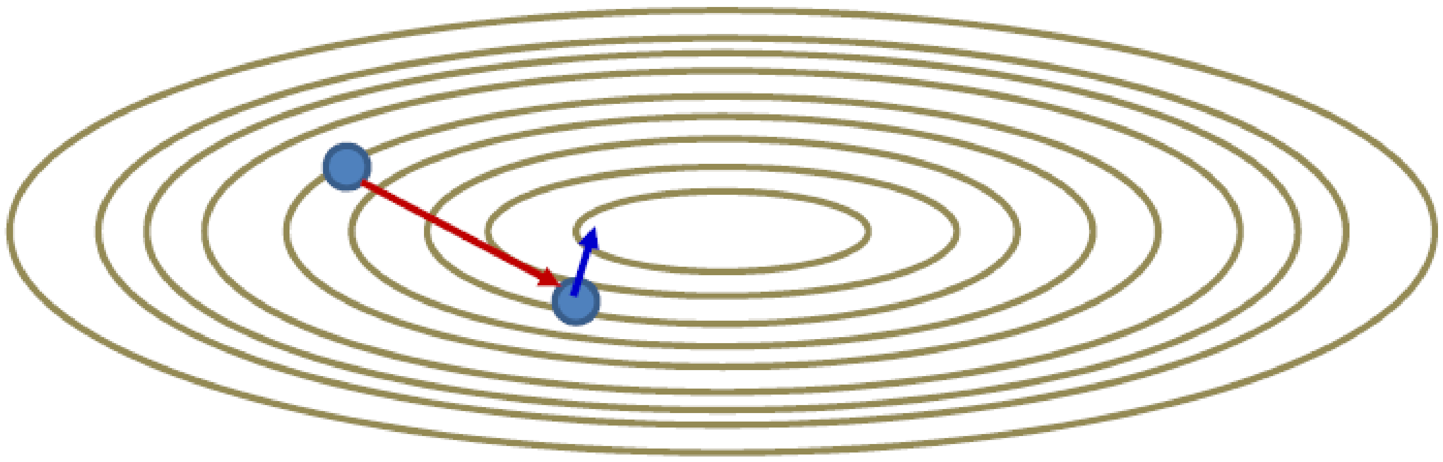


Accelerated Methods

- The momentum method

$$X^{k+1} = X^k - \eta \nabla f(X^k) + \beta(X^k - X^{k-1})$$

  - For each iteration
    - Compute the current gradient

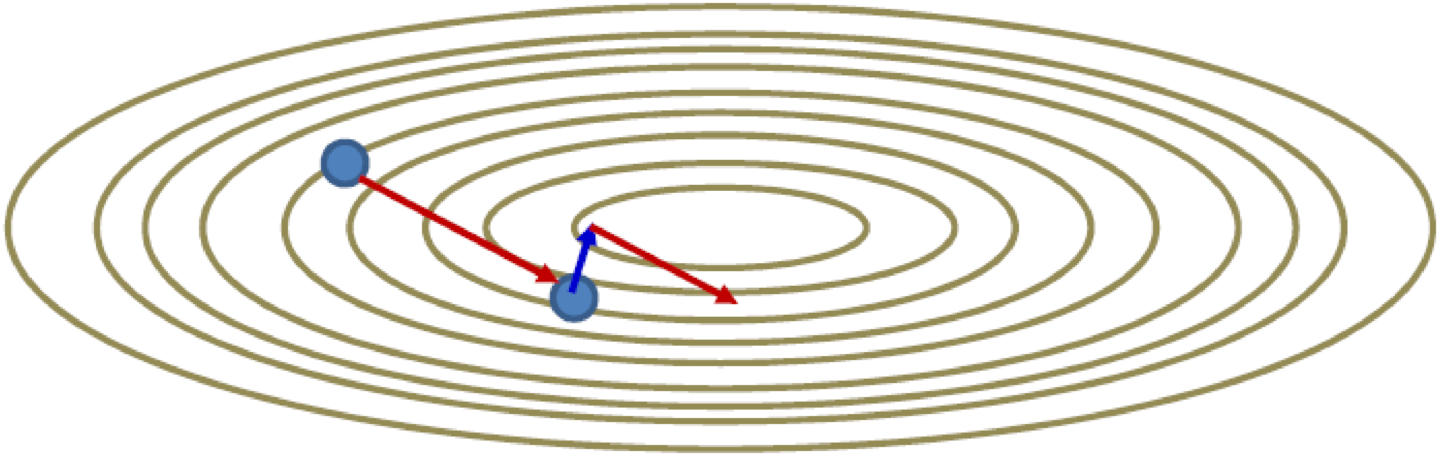


Accelerated Methods

- The momentum method

$$X^{k+1} = X^k - \eta \nabla f\left(X^k\right) + \beta\left(X^k - X^{k-1}\right)$$

  - For each iteration
    - Compute the current gradient
    - Add β-scaled previous step
      - Actually the running average from the previous iteration

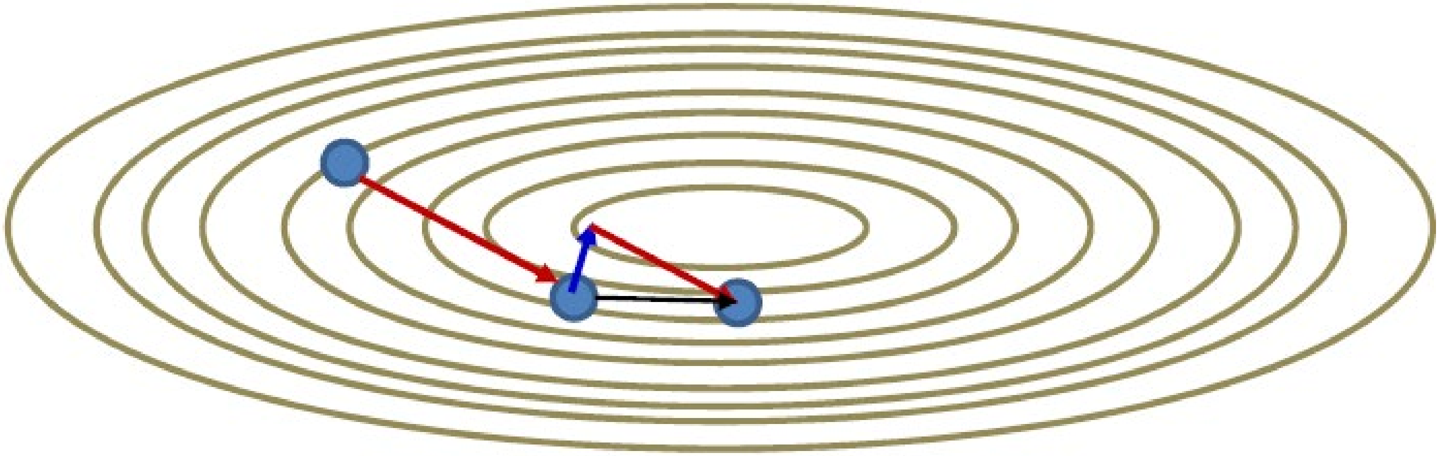


Accelerated Methods

- The momentum method

$$X^{k+1} = X^k - \eta\nabla f\left(X^k\right) + \beta\left(X^k - X^{k-1}\right)$$

  - For each iteration
    - Compute the current gradient
    - Add β-scaled previous step
    - Get the final direction



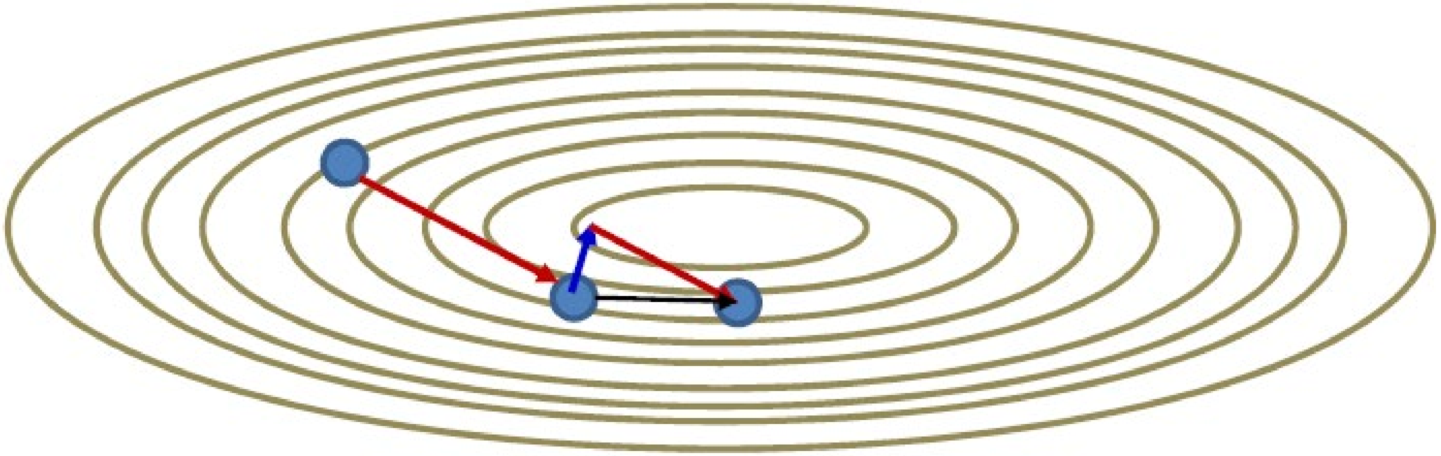
Accelerated Methods

- The momentum method

$$X^{k+1} = X^k - \eta \nabla f(X^k) + \beta (X^k - X^{k-1})$$

  - For each iteration
    - Compute the current gradient
    - Add β-scaled previous step
    - Get the final direction
- Convergence Rate?

Unfortunately no… disprove it in your homework ☺

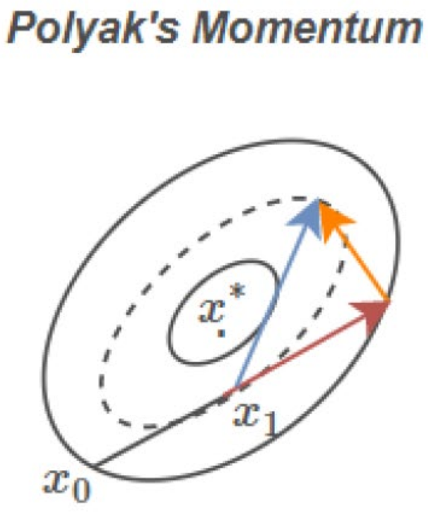


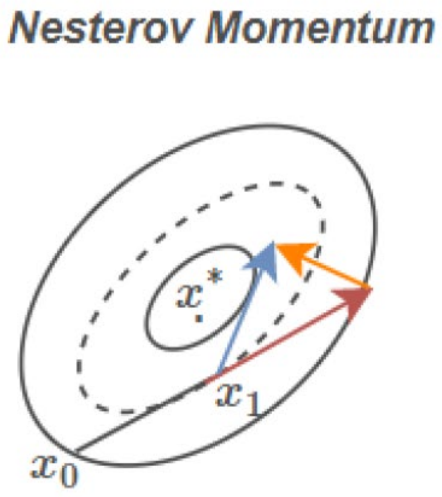
Accelerated Methods

- Nesterov's accelerated gradient descent

$$X^{k+1} = X^k - \eta \nabla f\left(X^k + \beta(X^k - X^{k-1})\right) + \beta(X^k - X^{k-1})$$

  - For each iteration
    - Compute the current gradient evaluated at the resultant location
    - Add β-scaled previous step
    - Get the final direction





Accelerated Methods

- Nesterov's accelerated gradient descent

$$X^{k+1} = X^k - \eta \nabla f\left(X^k + \beta(X^k - X^{k-1})\right) + \beta(X^k - X^{k-1})$$

  - For each iteration
    - Compute the current gradient evaluated at the resultant location
    - Add β-scaled previous step
    - Get the final direction

Class of Function	GD	NAG
Smooth	$O(1/T)$	$O(1/T^2)$
Smooth & Strongly-Convex	$O\left(exp\left(-\frac{T}{\kappa}\right)\right)$	$O\left(exp\left(-\frac{T}{\sqrt{\kappa}}\right)\right)$

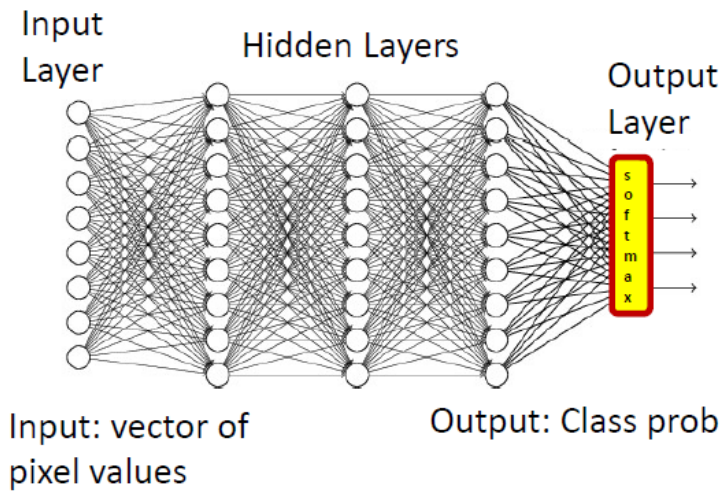
$\kappa = \frac{L}{\mu}$ $L \gg \mu$, (some direction has large gradient changes) the improvement becomes significant

Convex Optimization: Summary

- Take-Aways
  - Adaptive learning rates help convergence
  - Gradient descent maybe too slow or unstable due to inconsistency between dimensions
  - Second-order methods normalize dimensions but too expensive
  - Momentum methods emphasizes the directions of steady improvement and can significantly improve naïve GD

- Let’s switch to neural networks!

Optimization for Neural Networks



- Learning a neural classifier
  - Loss function $L(\theta) = \frac{1}{N}\sum_i err\left(f\left(x^i;\theta\right);y^i\right)$
  - Goal: minimize $L(\theta)$ w.r.t. θ
- Gradient descent
  - Backpropagation for each training sample $L_i(\theta) = \nabla_\theta err\left(f\left(x^i;\theta\right);y^i\right)$
  - Average gradients over N samples $\nabla L(\theta) = \frac{1}{N}\sum_i \nabla_\theta L_i(\theta)$
  - $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla L\left(\theta^k\right)$
- What if N is large?

Optimization for Neural Networks

- Stochastic Gradient Descent
  - Given training data $\mathcal{X} = \{(x^i, y^i)\}$
  - Random sample a data point $(x^j, y^j) \in \mathcal{X}$
  - Incremental update: $\theta^{k+1} \leftarrow \theta^k - \eta^k \nabla L_j(\theta^k)$

- Remark
  - An unbiased gradient estimate: $\nabla L(\theta) = \mathrm{E}_j[\nabla L_j(\theta)]$
  - Extremely efficient computation but may suffer from the variance
  - We can also apply a cyclic rule: $j = 1,2, \dots, N$
    - (refer to your machine learning course materials!)

Optimization for Neural Networks

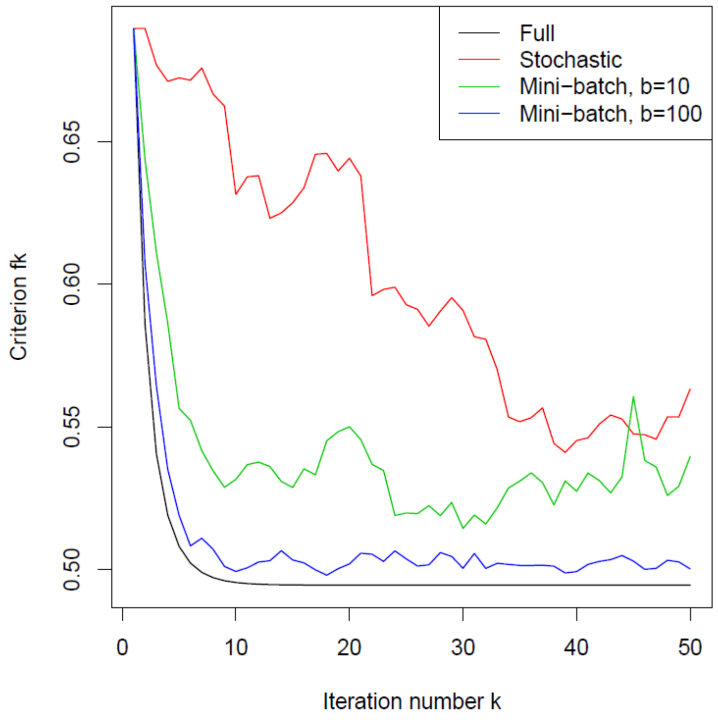
- Convergence Condition
  - A diminishing learning rate: $\sum_k \eta^k = \infty$ and $\sum_k {\eta^k}^2 < \infty$
  - Typical usage: $\eta^k \propto \frac{1}{k}$
  - Why wouldn't a constant learning rate work?
- Convergence Rate (ML Course Recap.)
  - Convex function: $\mathrm{E}\left[f\left(X^k\right)\right] - f^* = O\left(\frac{1}{\sqrt{k}}\right)$ (no improvement w. L-smooth)
  - Strongly convex function: $\mathrm{E}\left[f\left(X^k\right)\right] - f^* = O\left(\frac{1}{k}\right)$
  - Recap of GD: $O(1/k)$ for convex and L-smooth and linear for strongly convex
    - SGD is slightly worse (we will try to improve it later)

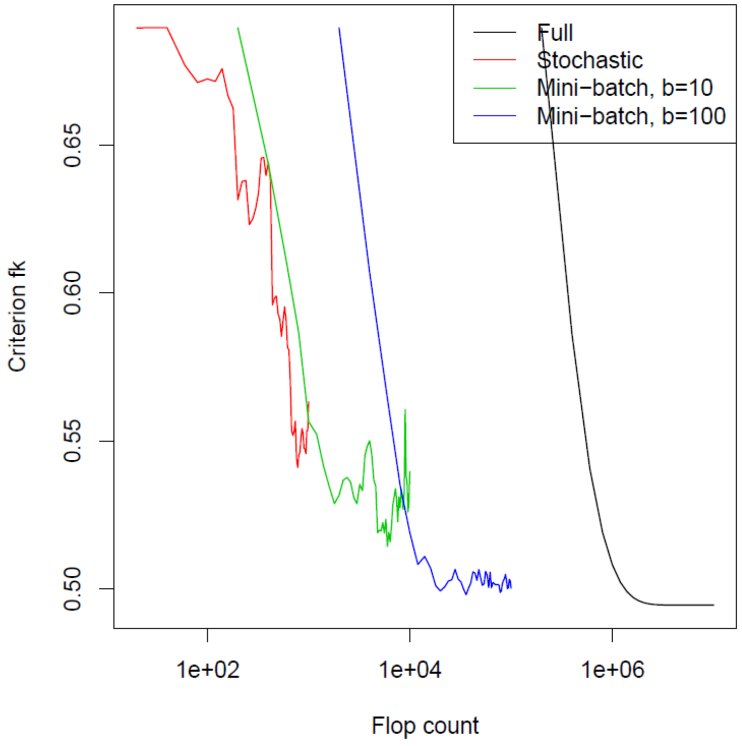
Optimization for Neural Networks

- Mini-batches!
  - We choose a random subset of indices $I_j \subset \{1, \dots, N\}$ and $|I_j| = b$ (***batch-size***)
  - $\mathcal{X} = \{(x^i, y^i): i \in I_j\}$: a mini-batch of training data
  - $L_{I_j}(\theta) = \frac{1}{b} \sum_{i \in I_j} L_i(\theta)$ estimate the gradient over a mini-batch
  - $\theta^{k+1} = \theta^k - \eta^k \nabla L_{I_j}(\theta^k)$
- Mini-batch gradient is still an unbiased estimate of the gradient
  - Reduce the variance by $\frac{1}{b}$
- Practical Use:
  - Randomly split $\mathcal{X}$ into $\frac{N}{b}$ mini-batches (an ***epoch***)
  - Run mini-batch gradient descent over these pre-split mini-batches

Optimization for Neural Networks

- Example
  - Logistic regression (strongly convex)
  - $d = 20$ dimensions, $n = 10^4$ data points, fixed learning rate





In practice, start with a large batch size when computation permitted

Optimization for Neural Networks

- Why do we use mini-batch for neural networks
  - Computation variance trade-off: GD too expensive; SGD too noisy
  - Noise can sometimes help escape local minimum / saddle point (with assumptions, refer to your ML course!)

On Nonconvex Optimization for Machine Learning: Gradients, Stochasticity, and Saddle Points

Chi Jin
University of California, Berkeley
chijin@cs.berkeley.edu

Praneeth Netrapalli
Microsoft Research, India
praneeth@microsoft.com

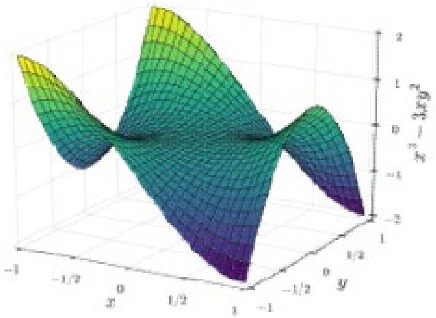
Rong Ge
Duke University
rongge@cs.duke.edu

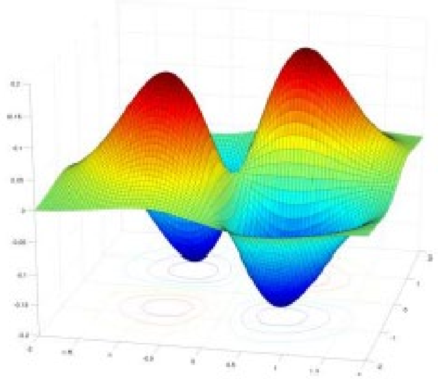
Sham M. Kakade
University of Washington, Seattle
sham@cs.washington.edu

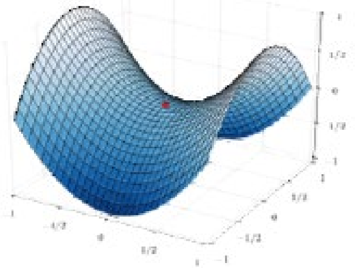
Michael I. Jordan
University of California, Berkeley
jordan@cs.berkeley.edu

September 5, 2019

Optimization for Neural Networks



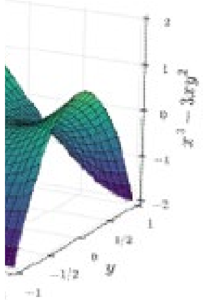




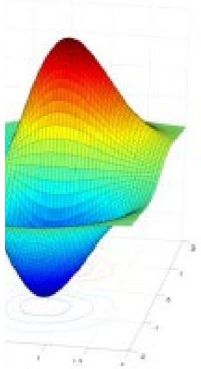
- Why do we use mini-batch for neural networks
  - Computation variance trade-off: GD too expensive; SGD too noisy
  - Noise can sometimes help escape local minimum / saddle point (with assumptions)

- Popular hypothesis in deep learning
  - Saddle points are far more common than local minima (exponential in network size)
    - Saddle points: Gradient is 0 and Hessian with both pos/neg eigenvalues
  - Most local minima are equivalent and close to global optimum
  - **NOT TRUE for small networks or in other domains like deep RL**

- **Baldi and Hornik (89), "***Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima*" : An MLP with a *single* hidden layer has only saddle points and no local Minima

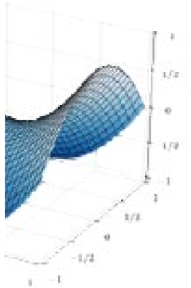


- **Dauphin et. al (2015), "***Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*" : An exponential number of saddle points in large networks



- **Chomoranksa et. al (2015)**, "*The loss surface of multilayer networks*" : For large networks, most local minima lie in a band and are equivalent
  - Based on analysis of spin glass models

- **Swirscz et. al. (2016)**, "Local minima in training of deep networks", In networks of finite size, trained on finite data, you *can* have horrible local minima



Optimization for Neural Networks

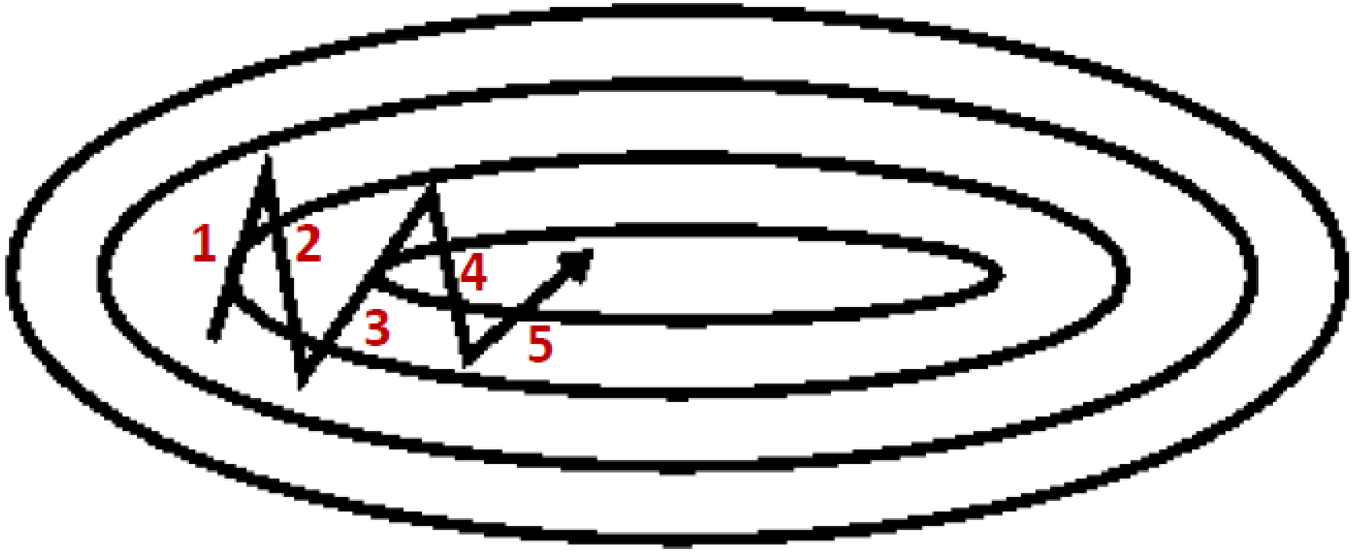
- Can we improve Mini-Batch GD?

- Recap: Inspirations from Convex Optimization
  - Adaptive learning rate
  - Decouple learning rate for dimensions
  - Second-order approximation can help (scaling axis)
  - Momentum helps

- SGD Variants for deep learning
  - AdaGrad; RMSProp; Adam; …

Optimization for Neural Networks

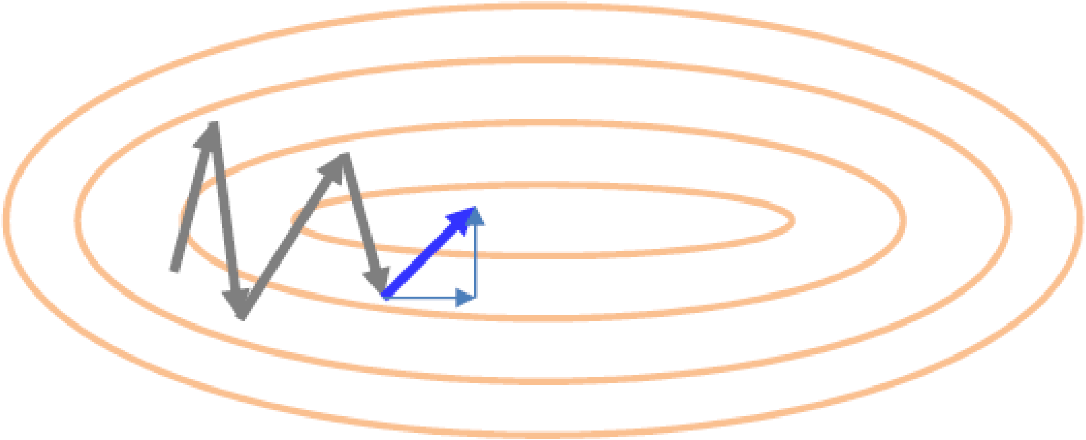
- A closer look at gradient descent
  - Standard accelerated methods still have oscillations
  - Observation: Steps in “oscillatory” directions show large total movement
  - Idea: slow down learning rate in directions with high motion
    - **Still high-order methods**

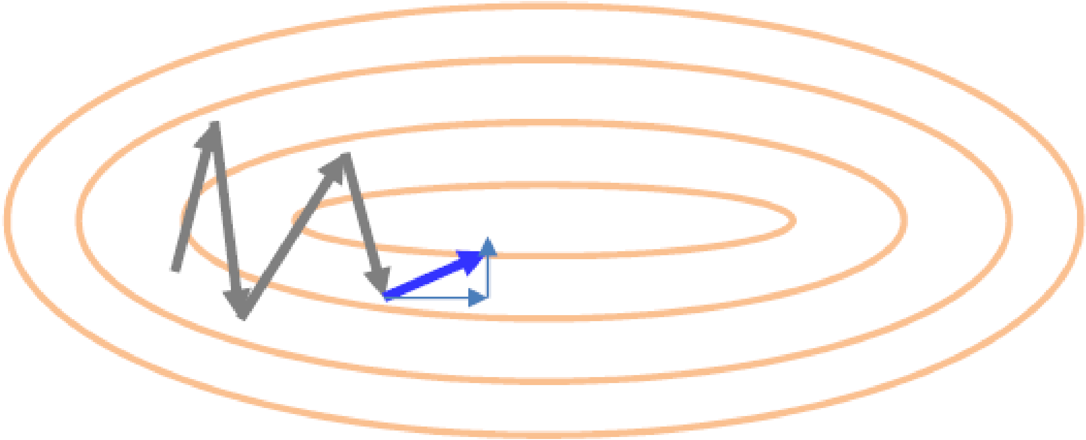


Step	X component	Y component
1	1	+2.5
2	1	-3
3	3	+2.5
4	1	-2
5	2	1.5

Optimization for Neural Networks

- A closer look at gradient descent
  - Standard accelerated methods still have oscillations
  - Observation: Steps in “oscillatory” directions show large total movement
  - Idea: slow down learning rate in directions with high motion
    - **Still high-order methods**





AdaGrad

- AdaGrad Algorithm (Duchi et al, 2011)
  - $g^k = \nabla_\theta f\left(\theta^k\right)$
  - $G_i^k = G_i^{k-1} + \left|g_i^k\right|^2$ (accumulated gradient square for each weight)
  - $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{G_i^k + \epsilon}} g_i^k$ (annealing learning rate for each weight)
- Remark
  - No need to tune learning rate (in practice fix $\eta = 0.01$, $\epsilon = 10^{-8}$)
  - Decoupled learning rate for each dimension
    - GLoVe use AdaGrad for word embeddings, so rare words have higher learning rate
  - Weakness: learning rate decays too quickly

RMSProp

- RMSProp Algorithm (by Hinton, in his lecture notes, ~ 2012)
  - $g^k = \nabla_\theta f(\theta^k)$
  - $G_i^k = \gamma G_i^{k-1} + (1-\gamma)\left|g_i^k\right|^2$ (moving average of square gradient)
  - $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{G_i^k + \epsilon}} g_i^k$
- Remark
  - Address the vanishing learning rate for AdaGrad
  - Works particularly well for RNNs

Adam

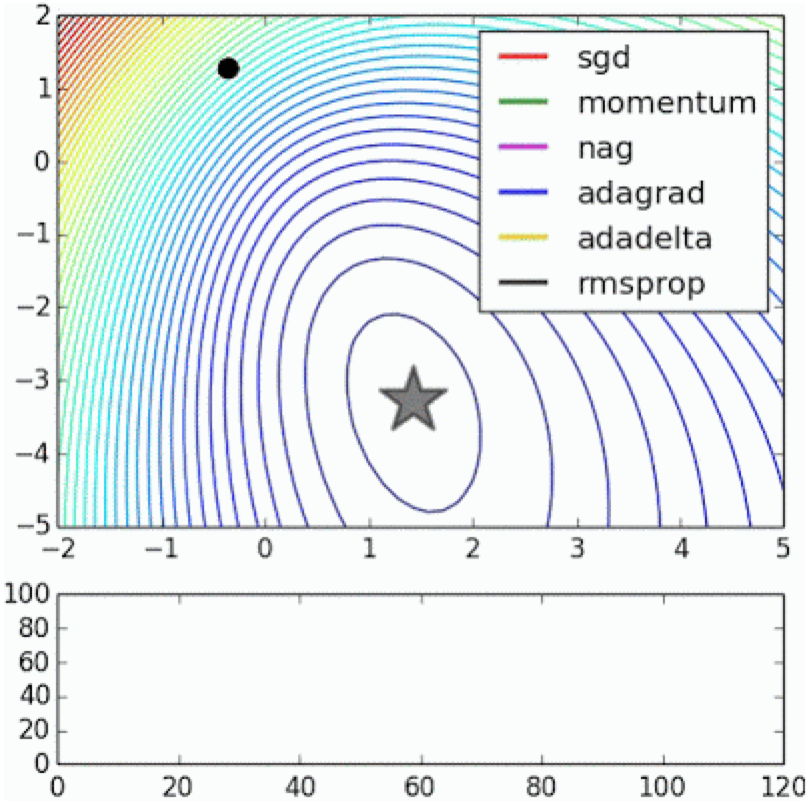
- Adam algorithm (Diederik P. Kingma & Jimmy Ba, 2014, arxiv)
  - RMSProp with Momentum, the most popular optimizer, >200k citations
  - $g^k = \nabla_\theta f(\theta^k)$
  - $M_i^k = \delta M_i^{k-1} + (1-\delta) g_i^k$ (momentum)
  - $G_i^k = \gamma G_i^{k-1} + (1-\gamma) \left| g_i^k \right|^2$ (RMS of square gradient)
  - $\widehat{M}^k = \frac{M^k}{1-\delta^k}, \quad \hat{G}^k = \frac{G^k}{1-\gamma^k}$ (ensure γ and δ terms do not dominate in early iters)
  - $\theta_i^{k+1} = \theta_i^k - \frac{\eta}{\sqrt{\hat{G}_i^k + \epsilon}} \widehat{M}_i^k$
- Remark
  - Particularly effective for RNN, generative models, RL

Optimizers for Deep Learning

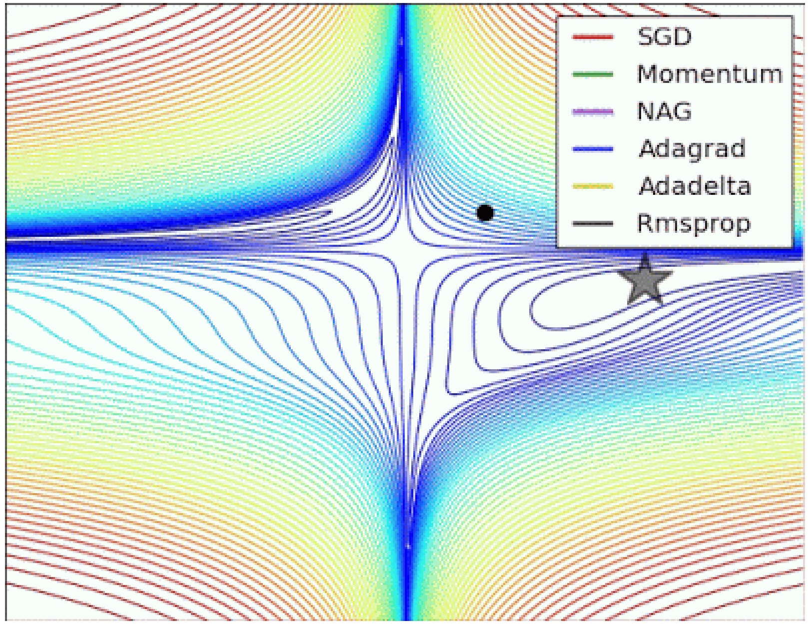
- A visualization by (Alec Radford, recently retired from OpenAI)

Optimizers for Deep Learning

- A visualization by (Alec Radford, recently retired from OpenAI)



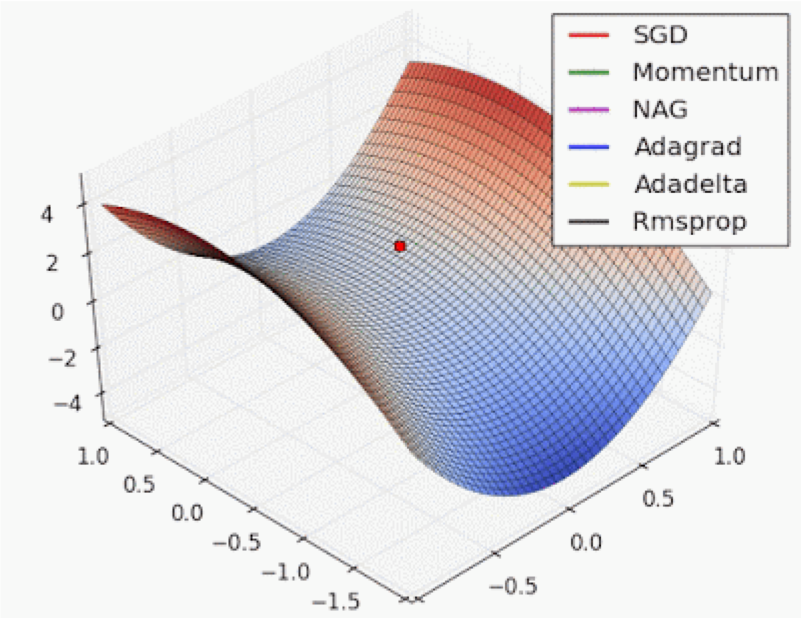
Noisy Moon



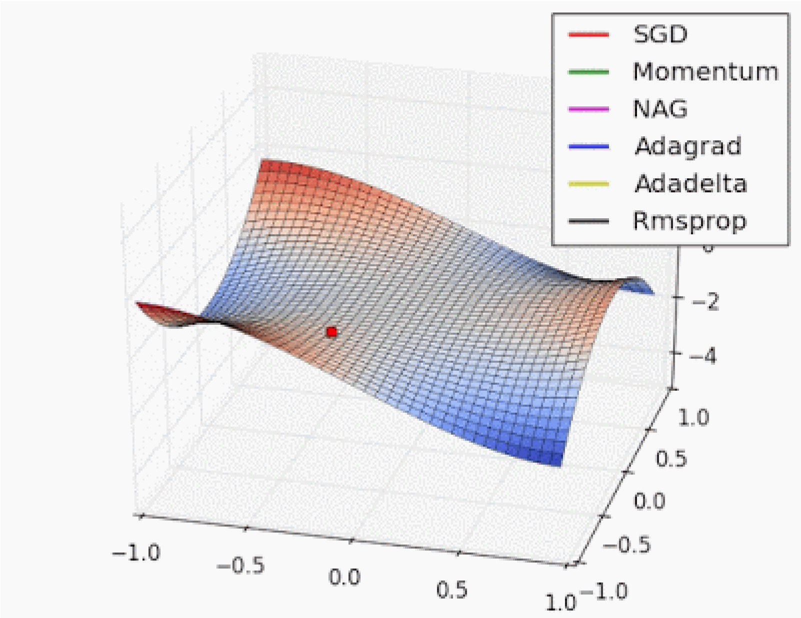
Beale’s Function

Optimizers for Deep Learning

- A visualization by (Alec Radford, recently retired from OpenAI)



Long Valley



Saddle Point

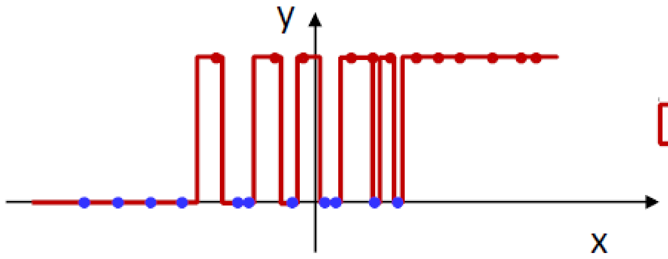
Today’s Lecture

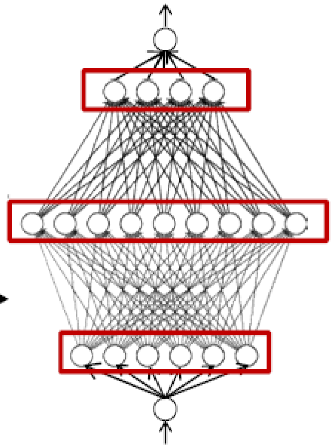
- Get your hand more dirty!

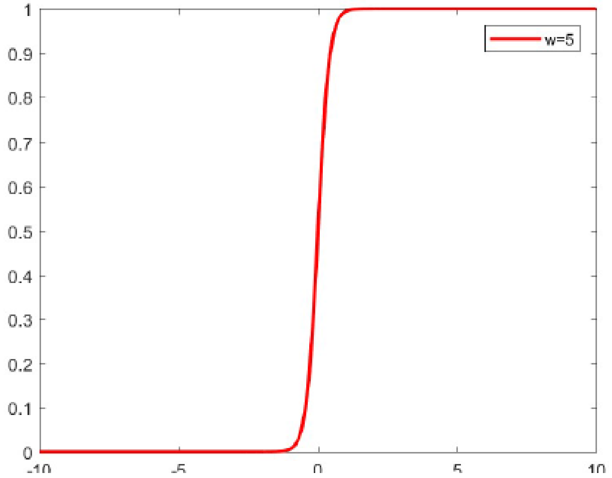
- Part 1: design a better learning algorithm
  - More tricks to play with gradients
- Part 2: more tricks for practical classification
  - Start to get professional in tuning!
- Part 3: advanced architectures
- Part 4: cloud computing tutorial

More Tricks Approaching!

- So Far
  - Common optimizers to minimize the loss
  - SGD, Momentum, AdaGrad, RMSProp, Adam
- Overfitting!
  - Neural networks are universal functions
  - Overfitted responses facilitated by large weights

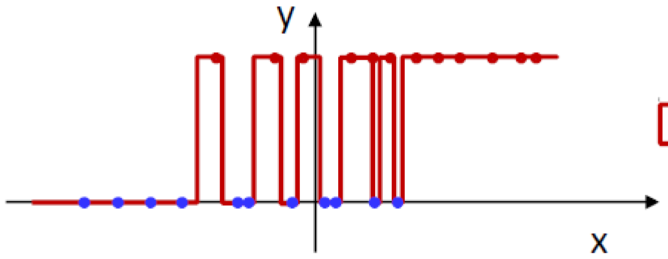


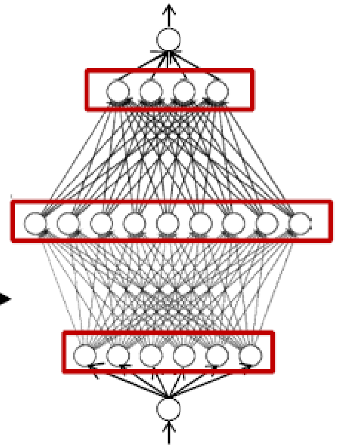


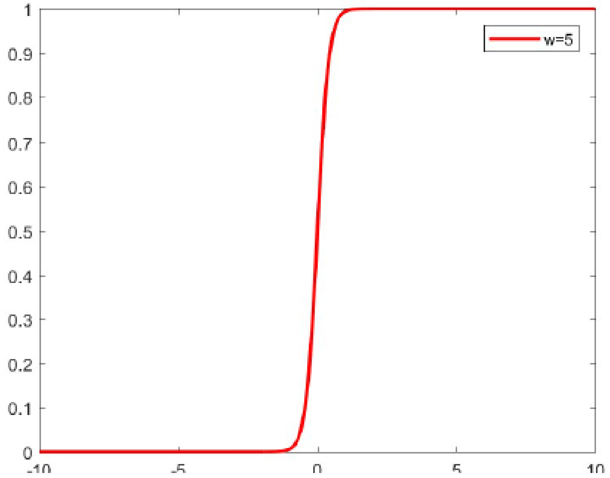


More Tricks Approaching!

- So Far
  - Common optimizers to minimize the loss
  - SGD, Momentum, AdaGrad, RMSProp, Adam
- Overfitting!
  - Neural networks are universal functions
  - Overfitted responses facilitated by large weights → Weight Decay (L2-norm)



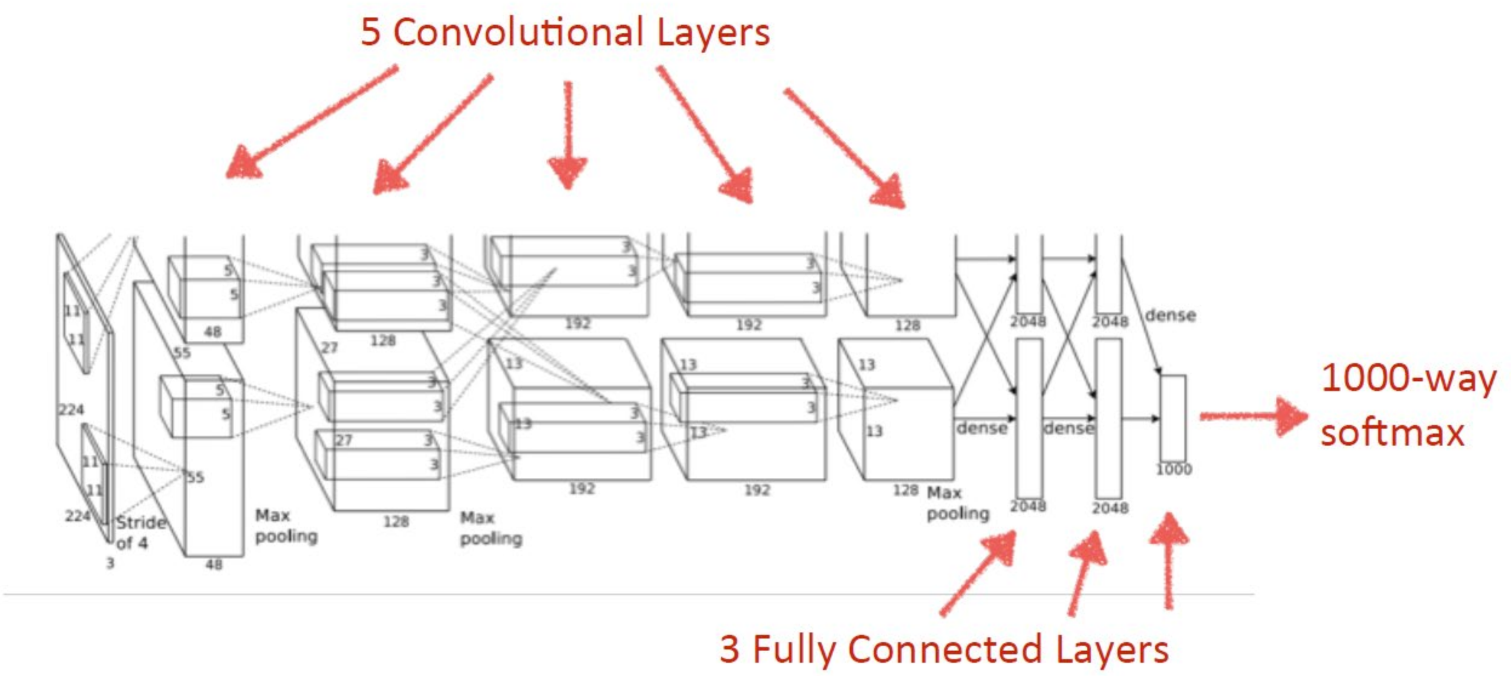


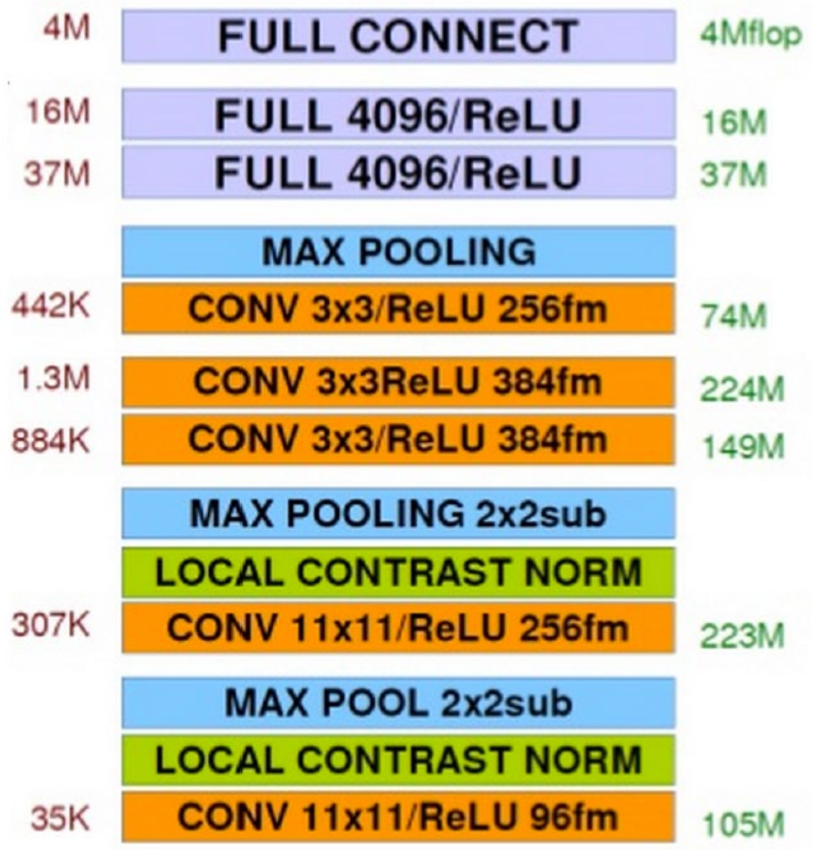


More?

Tricks in AlexNet

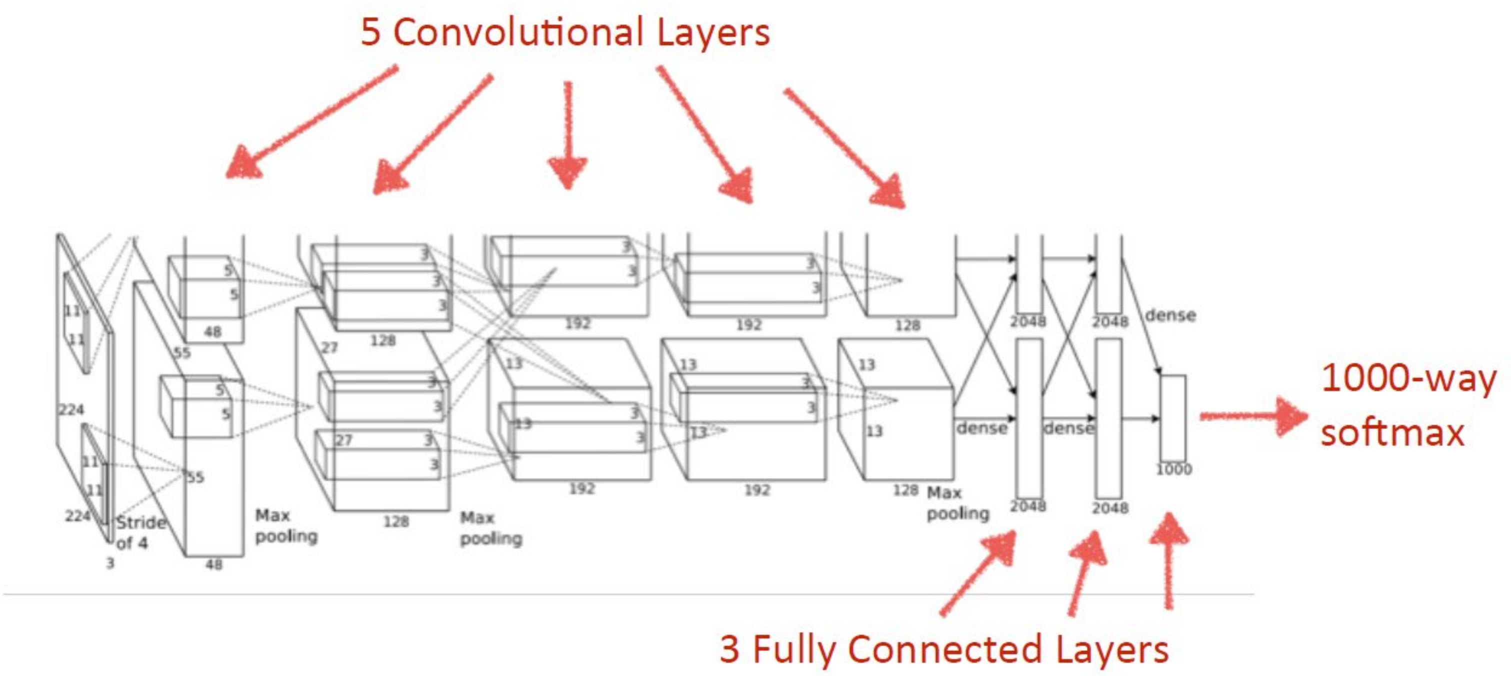
- AlexNet (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012)
  - First deep learning breakthrough in image classification

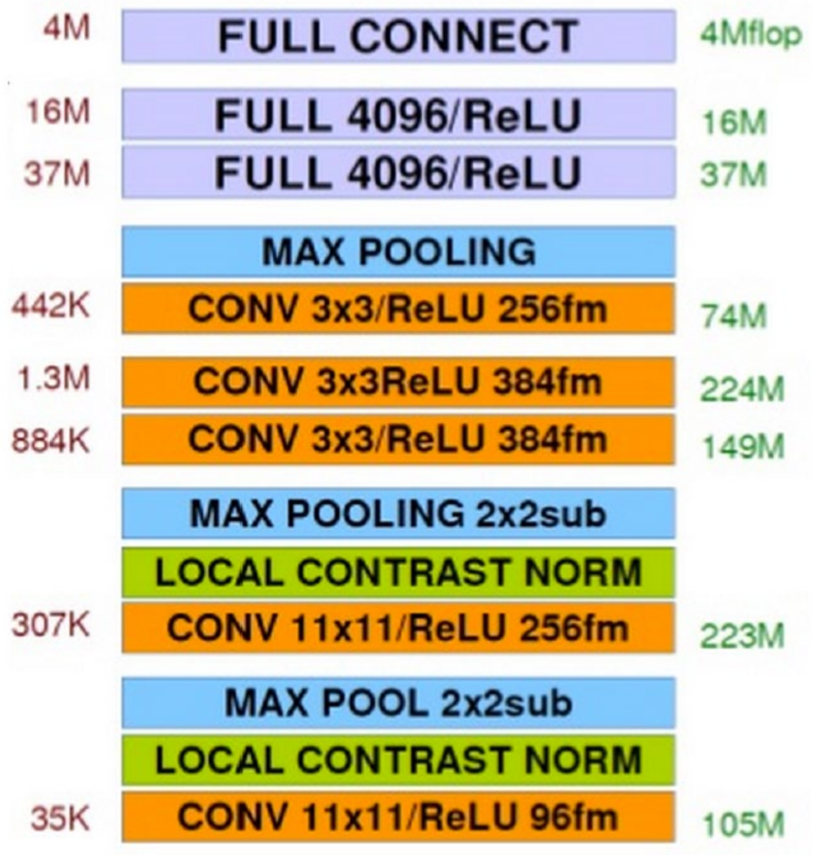




Tricks in AlexNet

- AlexNet (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012)
  - First deep learning breakthrough in image classification
  - ReLU activation and overlapping pooling

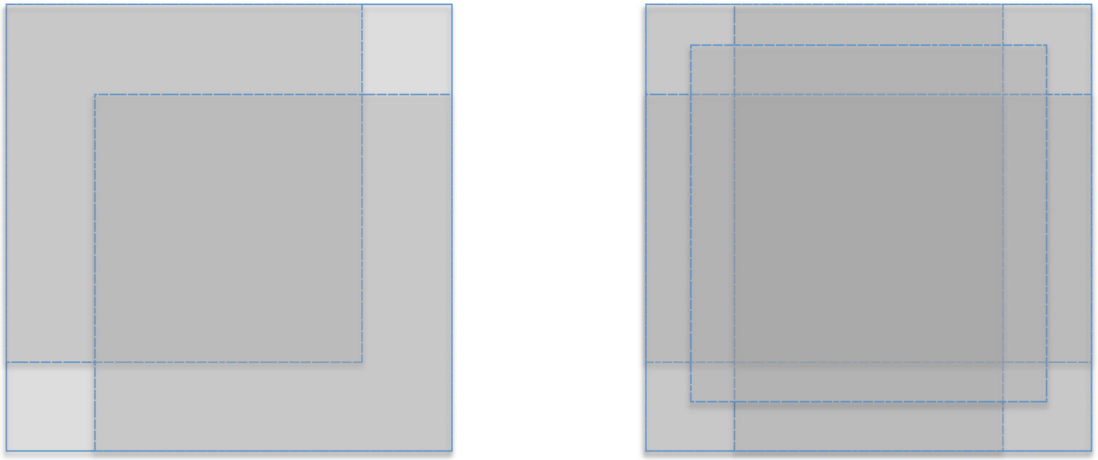




Tricks in AlexNet

- Data Preparation and Augmentation
  - Subtract the mean activity over the training set from each pixel
  - Crop 224x224 patches (and their horizontal reflections.)
  - At test time, average the predictions on the 10 patches (ensemble)
  - Change the intensity of RGB channels, add PCA components

$$[I_R, I_G, I_B] \mathrel{+}= \alpha[P_R, P_G, P_B][\lambda_R, \lambda_G, \lambda_B]^T, \alpha \sim N(0,0.1)$$

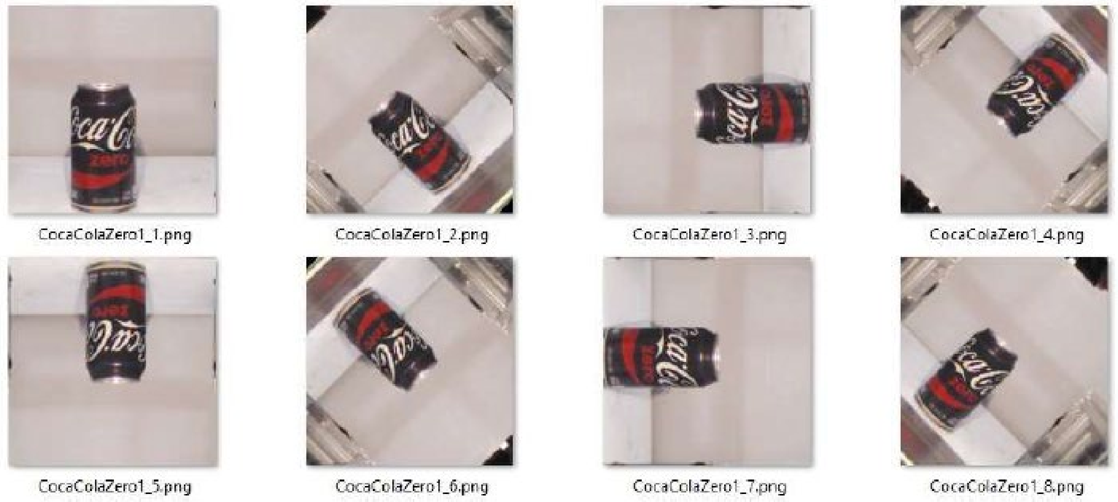


Tricks in AlexNet

- Data Preparation and Augmentation
  - Subtract the mean activity over the training set from each pixel
  - Crop 224x224 patches (and their horizontal reflections.)
  - At test time, average the predictions on the 10 patches (ensemble)
  - Change the intensity of RGB channels, add PCA components

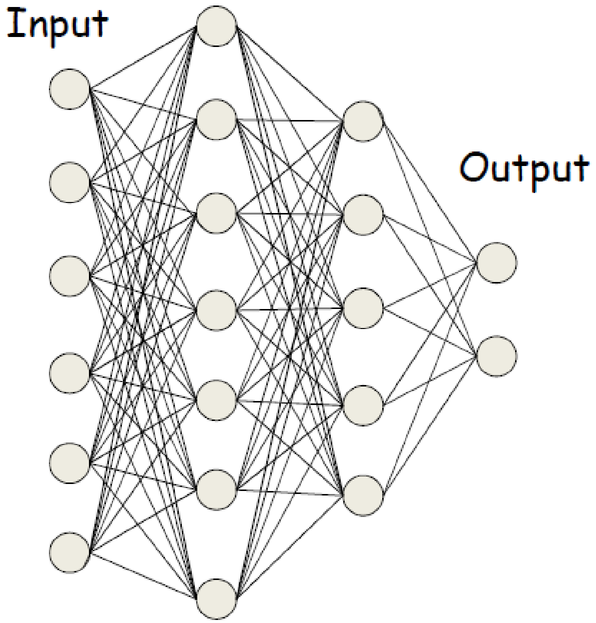
$$[I_R, I_G, I_B] \mathrel{+}= \alpha[P_R, P_G, P_B][\lambda_R, \lambda_G, \lambda_B]^T, \alpha \sim N(0,0.1)$$

  - More data augmentation tricks
    - Rotation, stretching, flipping, etc



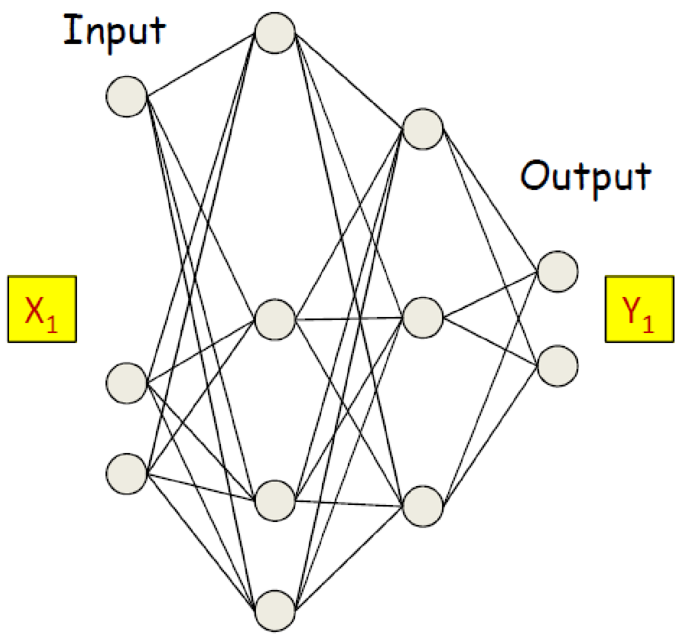
Tricks in AlexNet

- Dropout
  - **Training:** for each input, at each iteration, randomly “turn off” each neuron with a probability $1 - \alpha$



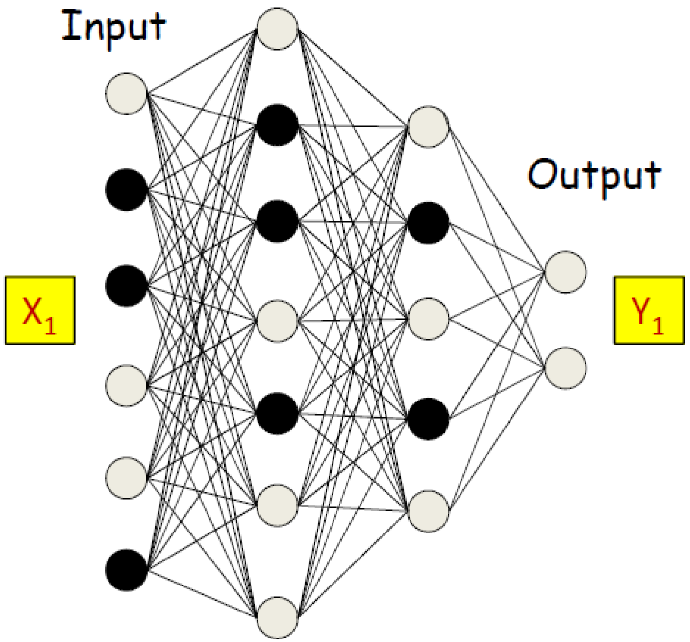
Tricks in AlexNet

- Dropout
  - **Training:** for each input, at each iteration, randomly “turn off” each neuron with a probability $1 - \alpha$
    - Intuition: randomly cut off some connections and neurons



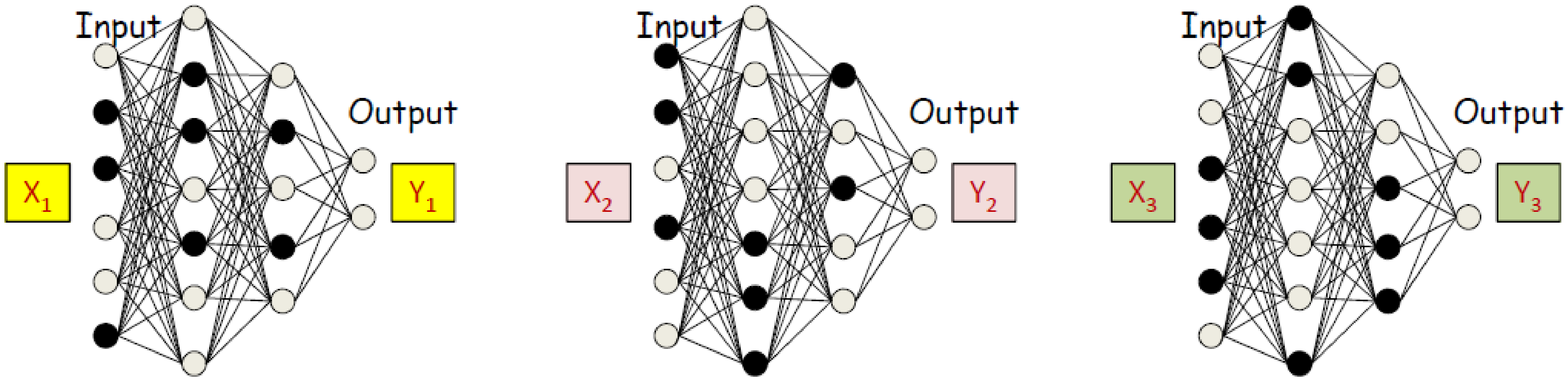
Tricks in AlexNet

- Dropout
  - **Training:** for each input, at each iteration, randomly "turn off" each neuron with a probability $1 - \alpha$
    - In practice, we change a neuron to 0 by sampling a Bernoulli variable with prob. $1 - \alpha$



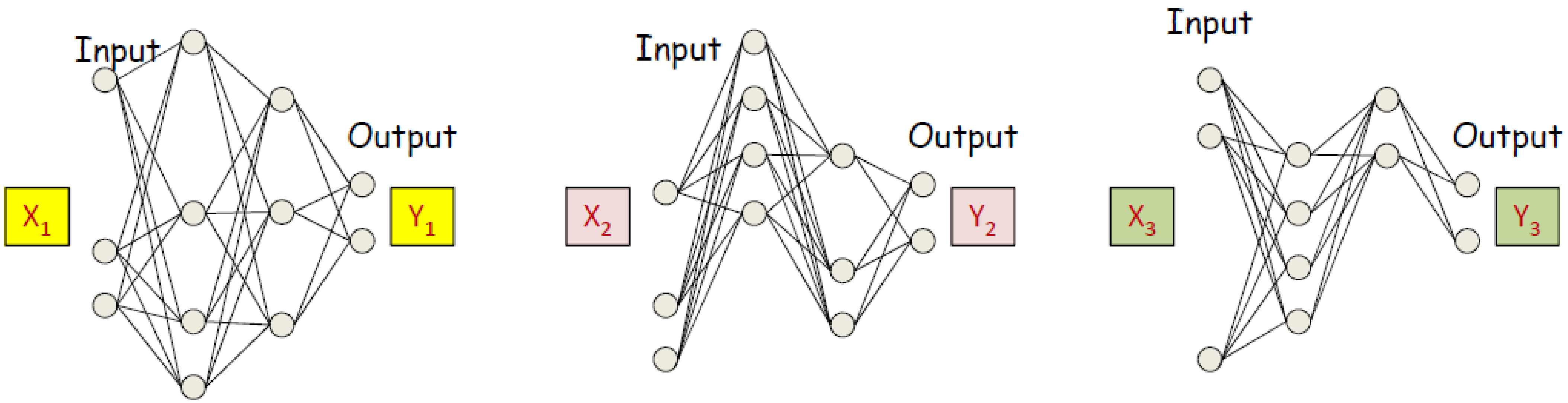
Tricks in AlexNet

- Dropout
  - **Training:** for each input, at each iteration, randomly "turn off" each neuron with a probability $1 - \alpha$
    - In practice, we change a neuron to 0 by sampling a Bernoulli variable with prob. $1 - \alpha$
    - Random "turn-off" prevent overfitting to particular neurons or weights



Tricks in AlexNet

- Dropout
  - **Training:** for each input, at each iteration, randomly "turn off" each neuron with a probability $1-\alpha$
    - In practice, we change a neuron to 0 by sampling a Bernoulli variable with prob. $1-\alpha$
    - Random "turn-off" prevent overfitting to particular neurons or weights
    - Gradient only propagated from non-zero neurons



Tricks in AlexNet

- Understanding Dropout
  - Dropout forces the neural network to learn redundant patterns
  - Dropout can be viewed as an implicit L2 regularizer

Dropout Training as Adaptive Regularization

Stefan Wager*, **Sida Wang**†, and **Percy Liang**†
Departments of Statistics* and Computer Science†
Stanford University, Stanford, CA-94305
swager@stanford.edu, {sidaw, pliang}@cs.stanford.edu

Abstract

Dropout and other feature noising schemes control overfitting by artificially cor-

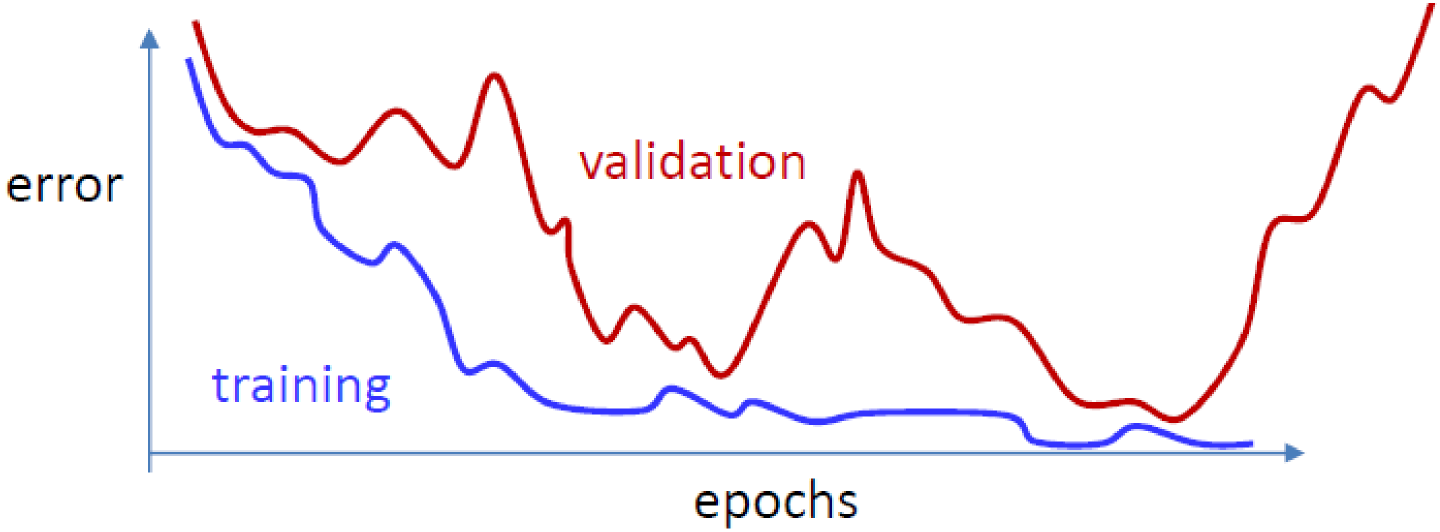
Tricks in AlexNet

- Dropout changes the scale of the output neuron
  - $y = Dropout(\sigma(\sum_i w_i x_i + b))$
  - $E[y] = \alpha E[\sigma(\sum_i w_i x_i + b)]$

- Dropout at Inference Time
  - $y = \alpha\sigma(\sum_i w_i x_i + b)$ expected output of the neuron

More Tricks

- Early Stopping
  - Continue training may lead to training data overfitting
  - Track performance on a held-out validation set



More Tricks

- Initialization
  - Zero initialization makes all neurons learn the same pattern

More Tricks

- Initialization
  - Zero initialization makes all neurons learn the same pattern → random init

More Tricks

- Initialization
  - Zero initialization makes all neurons learn the same pattern → random init
  - A too-large initialization leads to exploding gradients

More Tricks

- Initialization
  - Zero initialization makes all neurons learn the same pattern → random init
  - A too-large initialization leads to exploding gradients → small init

More Tricks

- Initialization
  - Zero initialization makes all neurons learn the same pattern → random init
  - A too-large initialization leads to exploding gradients → small init

- Design Principle
  - Zero activation mean
  - Activation variance remains same across layers

More Tricks

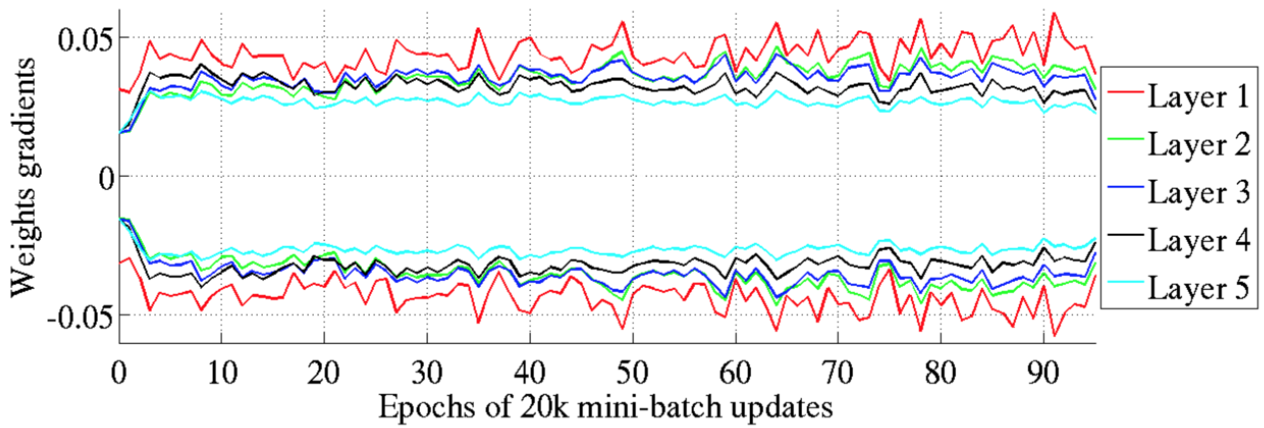
- Initialization
  - Zero initialization makes all neurons learn the same pattern → random init
  - A too-large initialization leads to exploding gradients → small init

- Design Principle
  - Zero activation mean → value at 0 (softsign, tanh) or large gradient (sigmoid)
  - Activation variance remains same across layers

    → prevent gradient vanishing/exploision

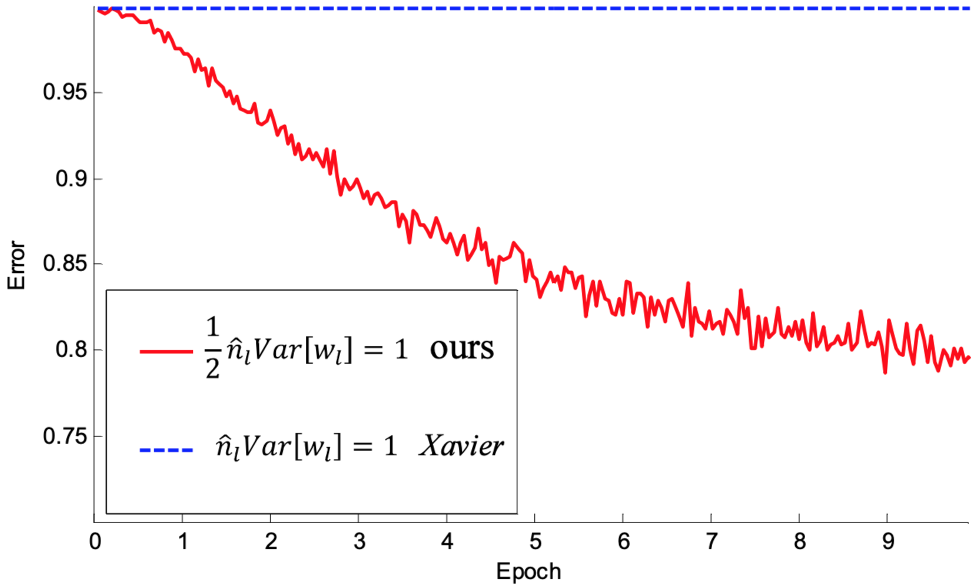
More Tricks

- Xavier Initialization (Xavier Glorot & Yoshua Bengio, AISTATS10)
  - $b^{(k)} \leftarrow 0$
  - $W^{(k)} \sim unif \pm \frac{\sqrt{6}}{\sqrt{n_k + n_{k+1}}}$
  - n_k hidden size of layer k (fan-in); n_{k+1} (fan-out)
- Experiments from the paper (tanh activation)



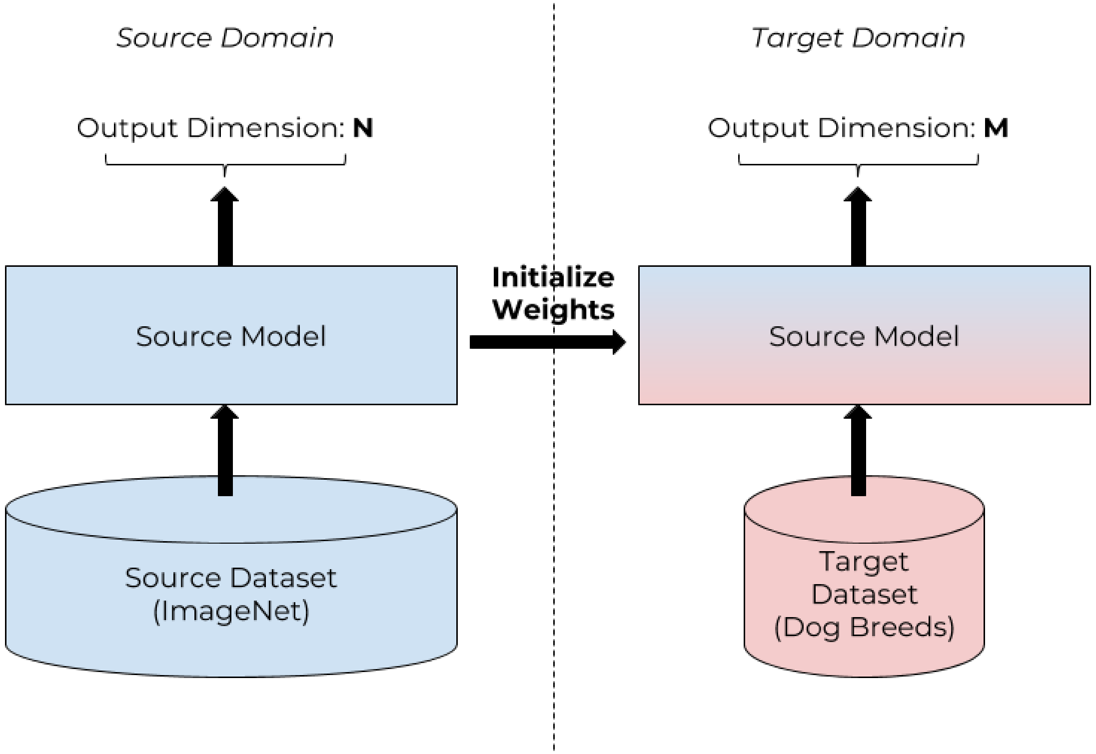
More Tricks

- Kaiming Initialization (Kaiming He et al., 2015)
  - $b^{(k)} \leftarrow 0$
  - $W^{(k)} \sim unif \pm \frac{\sqrt{2}}{\sqrt{n_k}}$
  - Only fan-in
- Remark
  - Designed for ReLU activation
  - Results for a 30-layer network



More Tricks

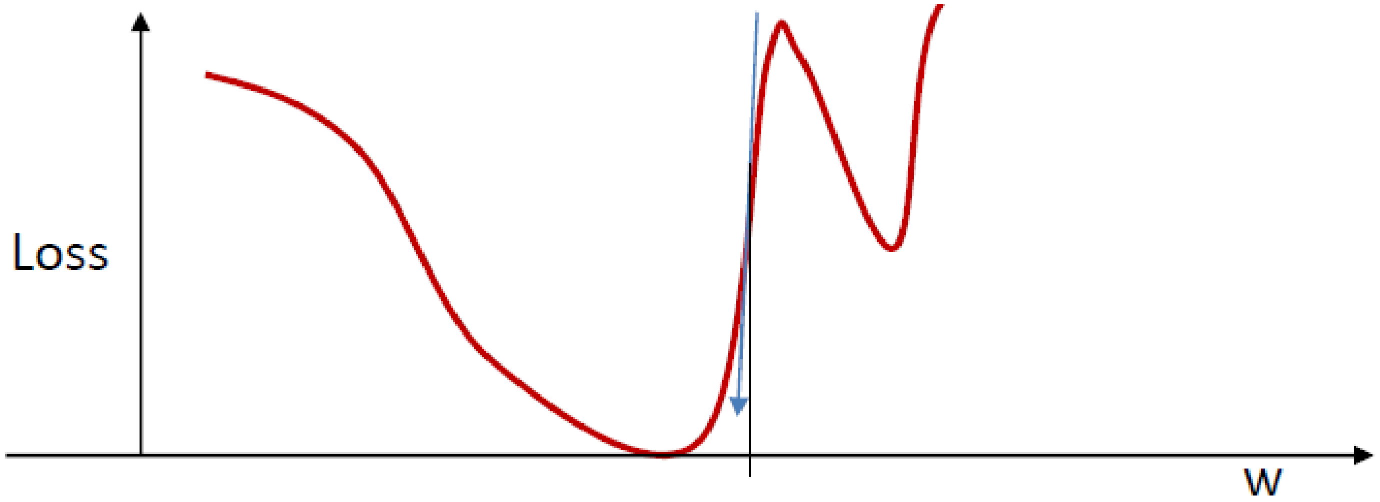
- Initialization by pretraining
  - Use a pretrained network as initialization
  - And then fine-tuning (a few layers or the whole network)



More Tricks

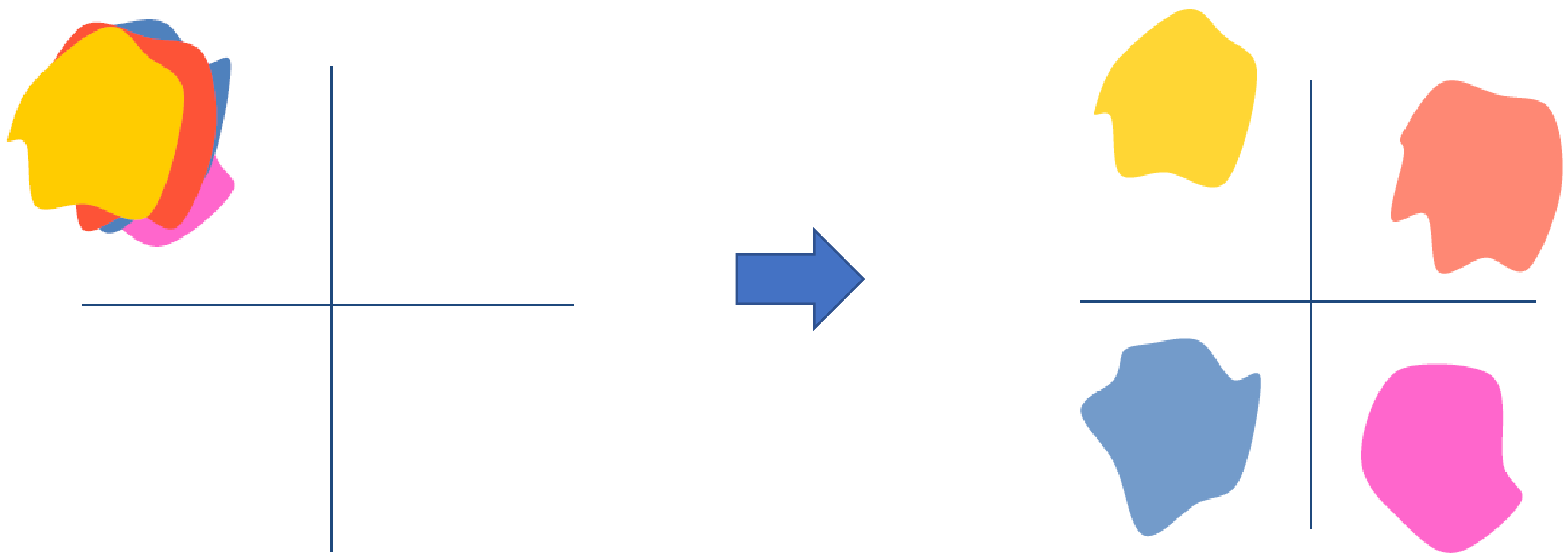
- Gradient Clipping
  - The loss can occasionally lead to a steep decent
  - This can result in immediate instability

  - If $\nabla\theta_i > 5$, then set $\nabla\theta_i$ to 5. (you can also scale the norm of $|\nabla\theta|$)



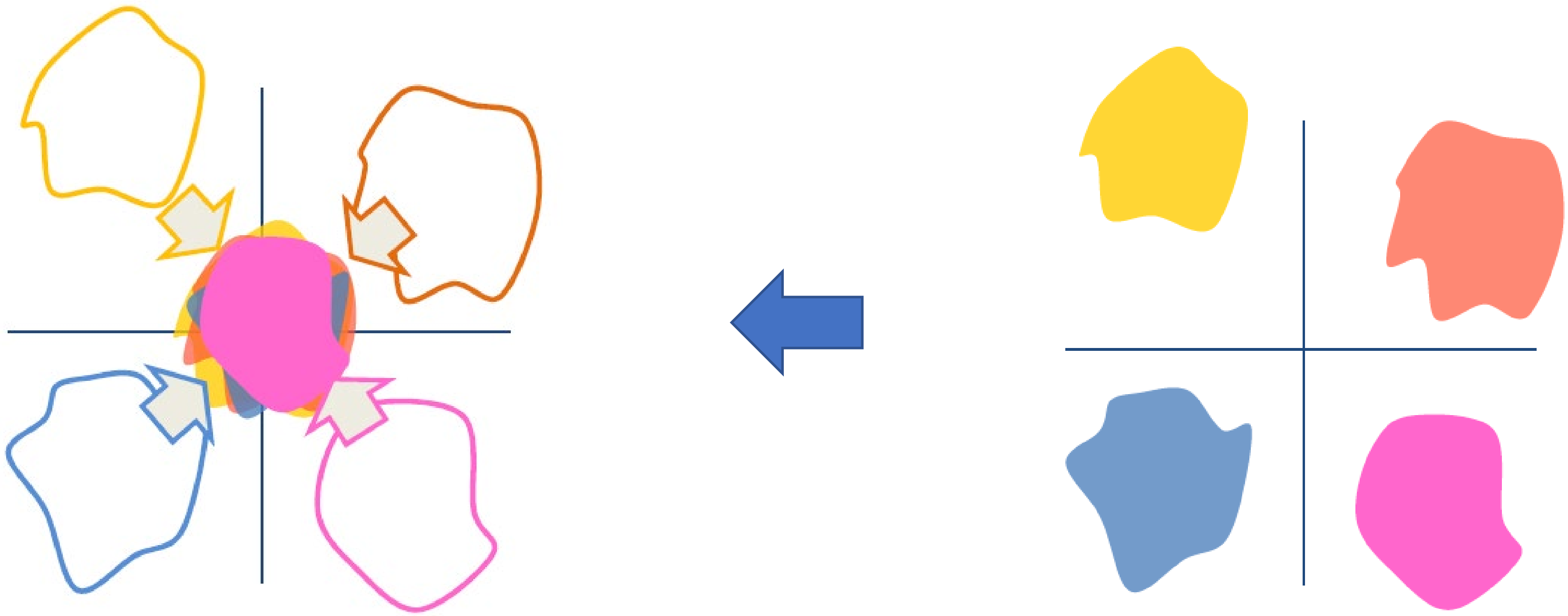
More Tricks: Covariance Shift

- The problem of covariance shift
  - Assumption: mini-batches share a similar data distribution
  - Reality: each minibatch may have a different distribution
    - Covariance shift
    - It can also cause covariance shift for different layers



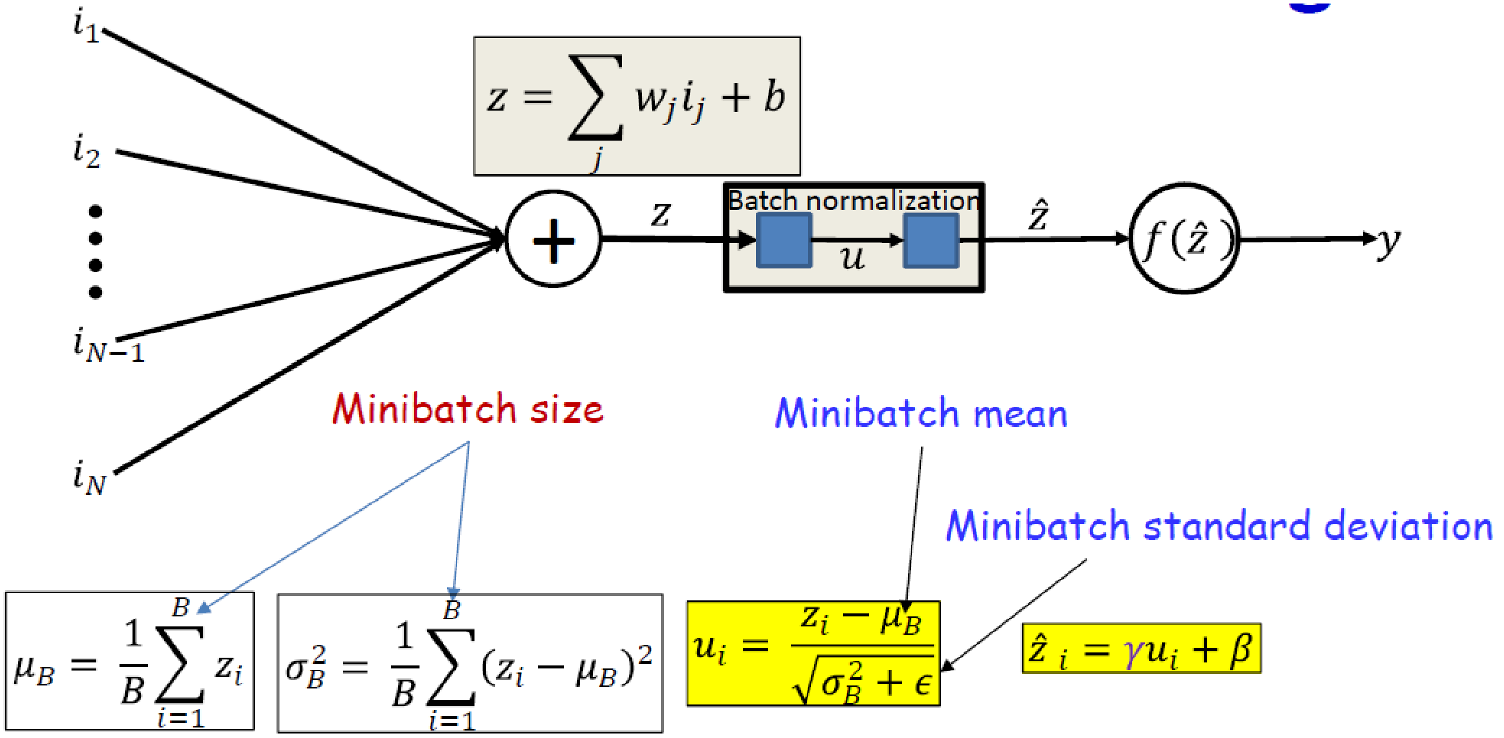
More Tricks: Covariance Shift

- The problem of covariance shift
  - Assumption: mini-batches share a similar data distribution
  - Reality: each minibatch may have a different distribution
  - Solution: make each batch same mean and standard deviation for training



Batch Normalization

- BatchNorm Layer (Sergey Ioffe & Christian Szegedy, 2015, 60k cites)
  - u_i : scaled activations with zero-mean and unit std dev
  - $\hat{z}_i = \gamma u_i + \beta$: Then shift to a proper location, γ, β are parameters



Batch Normalization

- BatchNorm at Training Time
  - Standard Backprop performed for each single training data

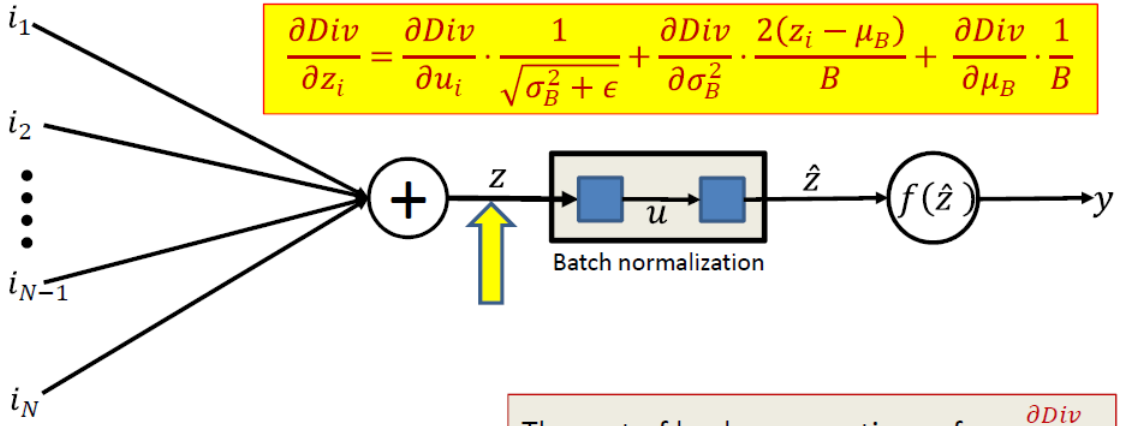
Batch Normalization

- BatchNorm at Training Time
  - Standard Backprop performed for each single training data
  - Now backprop is performed over the entire batch (derivation skipped)

$$\frac{\partial Div}{\partial \sigma_B^2} = \frac{-1}{2}\left(\sigma_B^2 + \epsilon\right)^{-3/2} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^{B} \frac{\partial Div}{\partial u_i}$$

$$\frac{\partial Div}{\partial z_i} = \frac{\partial Div}{\partial u_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial Div}{\partial \sigma_B^2} \cdot \frac{2(z_i - \mu_B)}{B} + \frac{\partial Div}{\partial \mu_B} \cdot \frac{1}{B}$$



The rest of backprop continues from $\frac{\partial Div}{\partial z_i}$

Batch Normalization

- BatchNorm at Training Time
  - Standard Backprop performed for each single training data
  - Now backprop is performed over the entire batch (derivation skipped)

- BatchNorm at Inference Time
  - We need to estimate μ_B and σ_B^2
  - (Running) Average of training mini-batches!
    - $\mu_B = \frac{1}{N\_batch} \sum_{batch} \mu_B(batch)$
    - $\sigma_B^2 = \frac{1}{N\_batch} \cdot \frac{B}{B-1} \sum_{batch} \sigma_B^2(batch)$

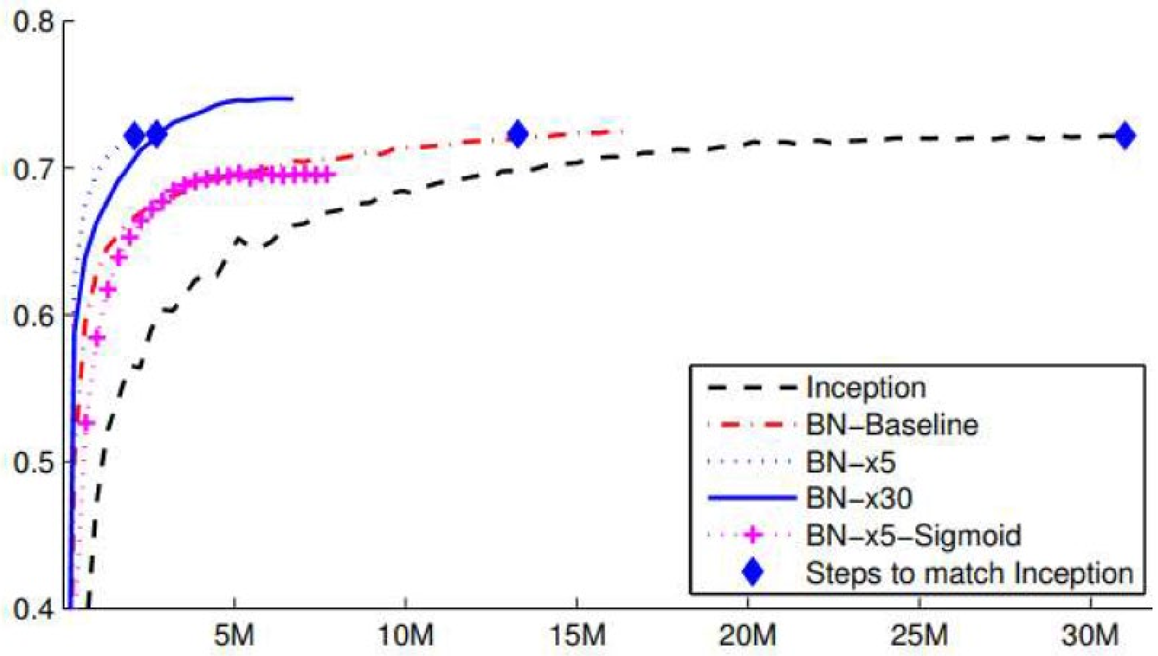
Batch Normalization

- BatchNorm at Training Time
  - Standard Backprop performed for each single training data
  - Now backprop is performed over the entire batch (derivation skipped)

- BatchNorm at Inference Time
  - We need to estimate μ_B and σ_B^2
  - (Running) Average of training mini-batches!
    - $\mu_B = \frac{1}{N\_batch} \sum_{batch} \mu_B(batch)$
    - $\sigma_B^2 = \frac{1}{N\_batch} \cdot \frac{B}{B-1} \sum_{batch} \sigma_B^2(batch)$ ← unbiased variance estimator

Batch Normalization

- Remarks
  - Evidently, no dropout necessary (or tiny dropout rate) with batch norm
  - Batch norm applies to specific layers (most popular in convolution layer)
  - Larger learning rate and faster decay (data always in high gradient region)



Layer Normalization

- LayerNorm layer (Jimmy Ba, Jamie Kiros, Hinton, 2016)
  - Scales the mean and std-dev of a hidden layer

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

- Remark:
  - Batch-independent
  - Particularly suitable for RNN
  - It also works extremely well for MLPs

More Regularizations

- WeightNorm
  - Suitable for meta-learning setting when high order of gradients are computed
- InstanceNorm
  - Batch-independent, suitable for generation tasks
- GroupNorm (by Yuxin Wu & Kaiming He)
  - Batch-independent, improve BatchNorm for small batch size

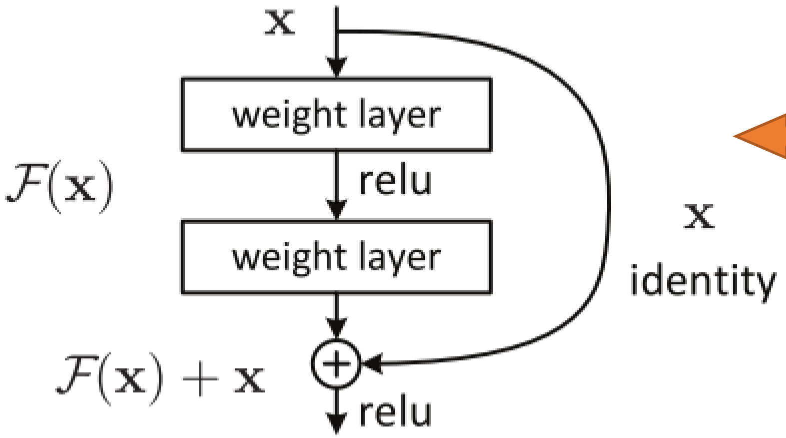
Today's Lecture

- Get your hand more dirty!

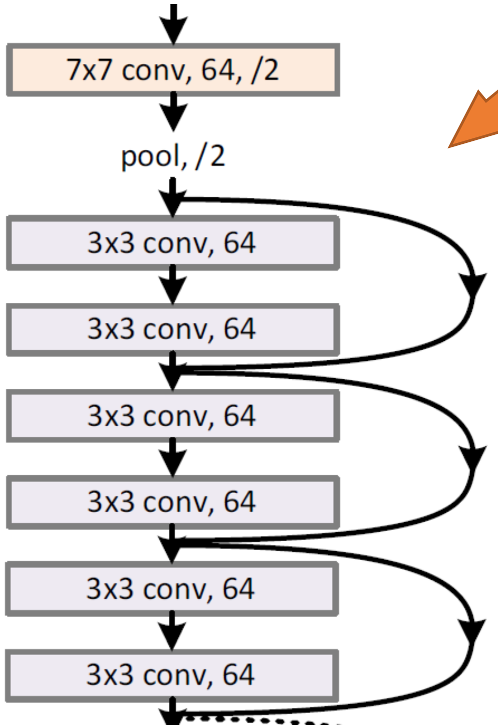
- Part 1: design a better learning algorithm
  - More tricks to play with gradients
- Part 2: more tricks for practical classification
  - Start to get professional in tuning!
- Part 3: advanced architectures
- Part 4: cloud computing tutorial

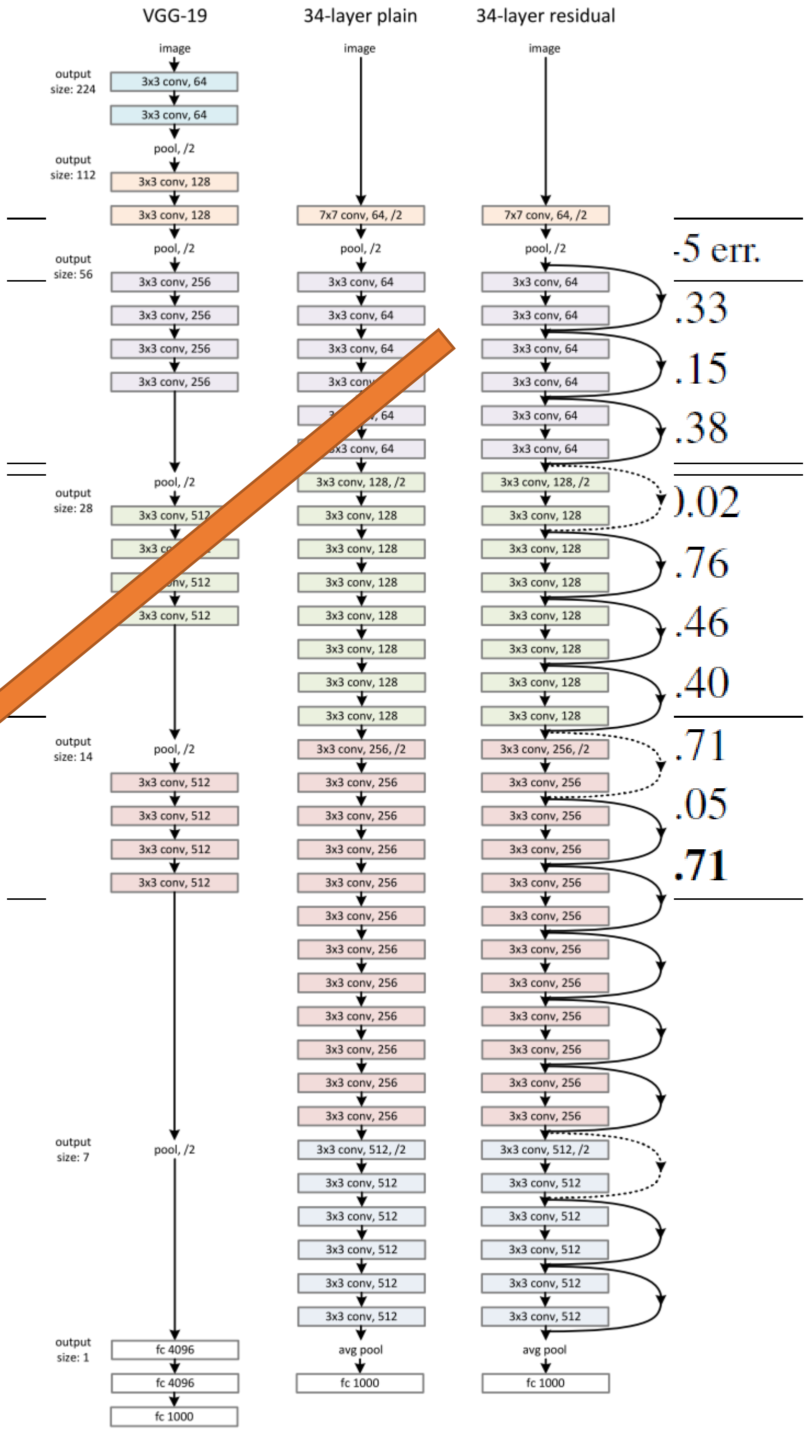
Residual Network

- ResNet (Kaiming He, et al., 2015)
  - ImageNet 2015 Champion
  - First “deep” network with >100 layers!



Residual Block





Residual Network

- Residual Connection

$$z = \sigma(f(x) + x)$$

- Justification from the paper
  - A trivial solution with good precondition for arbitrarily deep network

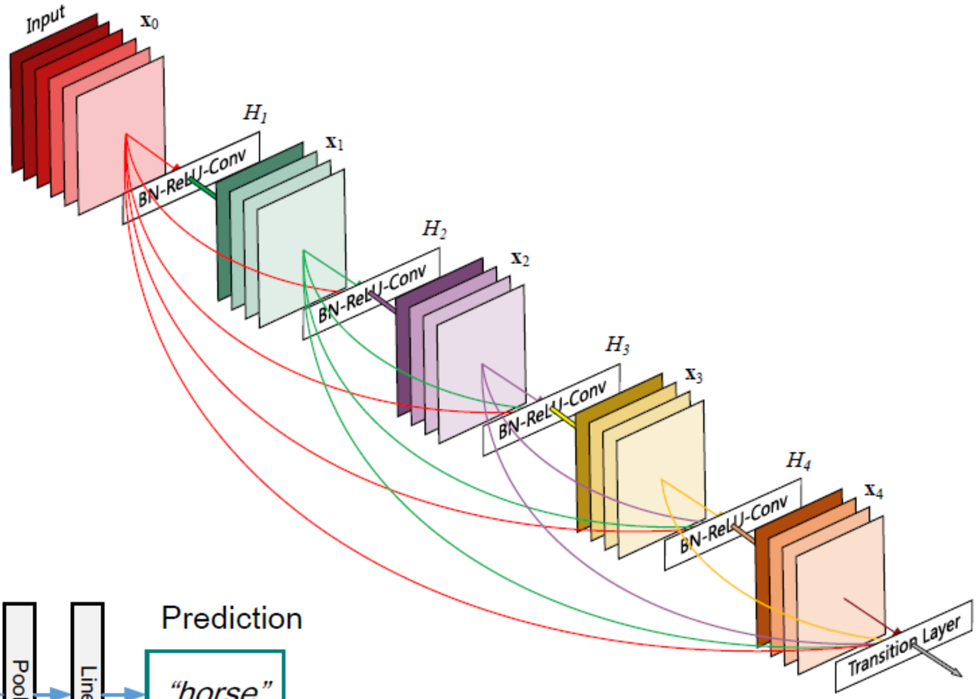
$$W^{(k)} = I$$

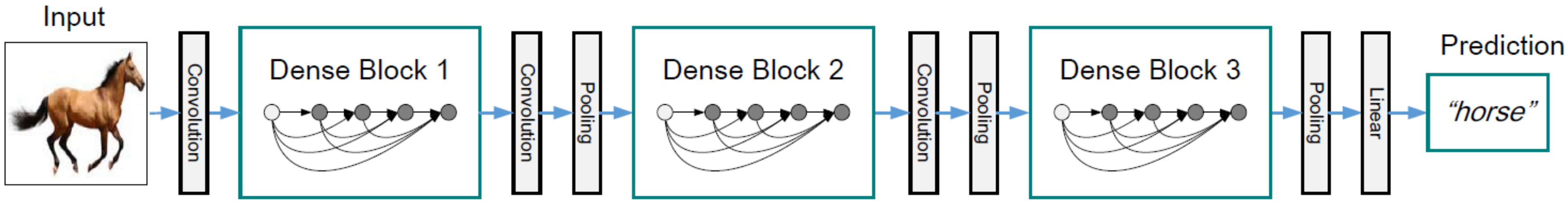
  - Hypothesis: hard to learn identity but easy to learn zero
  - Solution: fit the residual function $H(x) = f(x) - x$
- True story
  - One day a bug happened and you see extremely good valid error…
- Fun story about 何恺明
  - 2003清华基科班， first paper out at PhD 3rd year, CVPR 09 best paper
  - BP at CVPR 2016, ICCV 2017, BP honorable mention ECCV2018

Do solid research!

Densely Connected Network

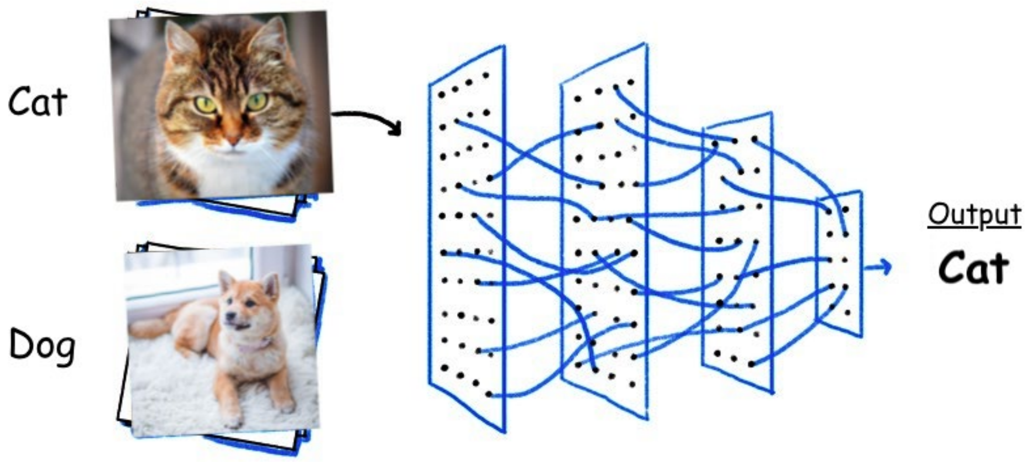
- Shallow networks to achieve the same performance of ResNet?
- DenseNet (by 黄高 & 刘壮, et al, 2016, CVPR17 best paper)
    - Take outputs of all previous layers
    - Directly get information flow from all layers
    - Issue:
        - Network maybe too wide
        - Need to be careful about memory consumption





Deconvolution

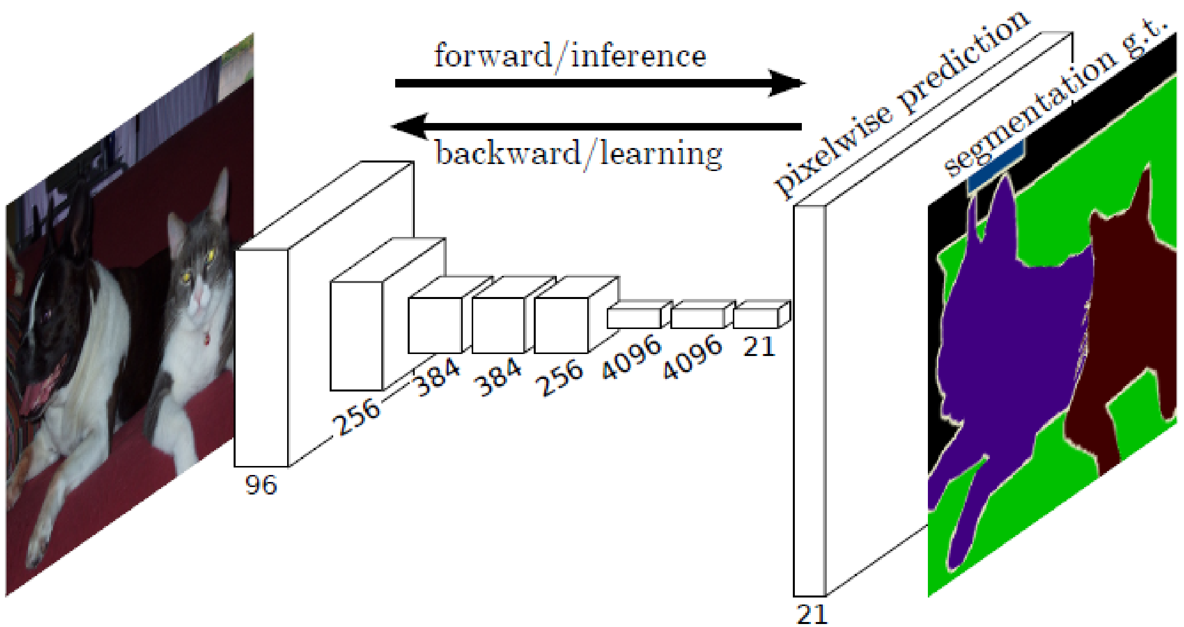
- Image Classification
  - From high-dimensional to a low-dimensional output
  - Convolution / pooling to keep down-sampling the image

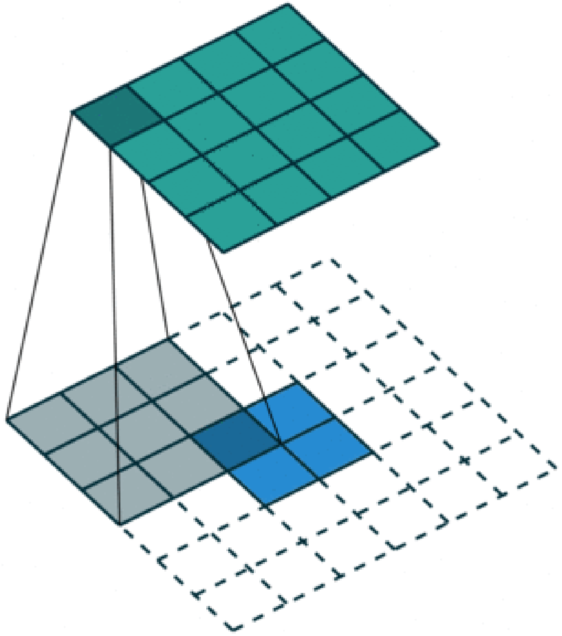


- The reverse order?
  - label → image?
  - image → image?

Fully Convolutional Network (Revisited)

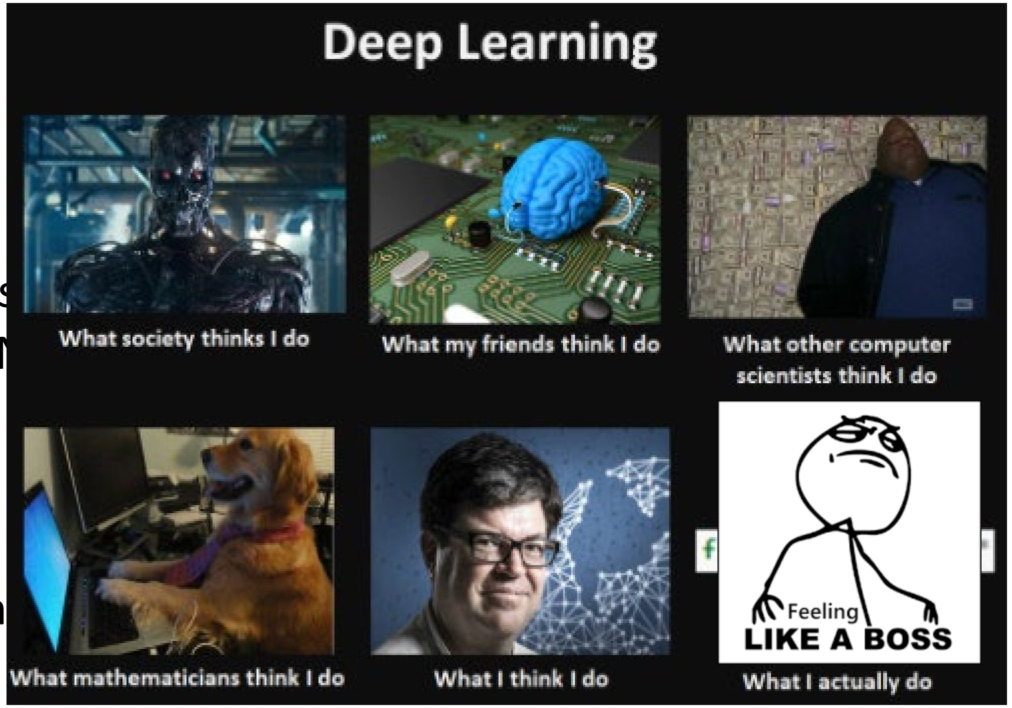
- FCN for Semantic Segmentation (Long et al, 2014)
  - First example of fully convolutional network
  - Image to segmentation mask
  - Use deconvolution layer to up sampling an image/map
  - More to use in generative models!





Summary

- The tricks today!
  - Optimizers
    - SGD, Momentum, RMSProp, Adam, etc
  - Regularization techniques
    - Initialization, clipping, early stopping, data process
    - Regularization layers (Dropout, BatchNorm, Layer
  - Architecture
    - Residual Connection
    - FCN
    - And more to come (later in this course and in com



- You are now ready for becoming a tuning professional!
  - General hints:
    - First overfit, then regularize; L-rate decay; Learn from well-tuned architectures

Today's Lecture

- Get your hand more dirty!

- Part 1: design a better learning algorithm
  - More tricks to play with gradients
- Part 2: more tricks for practical classification
  - Start to get professional in tuning!
- Part 3: advanced architectures
- Part 4: cloud computing tutorial