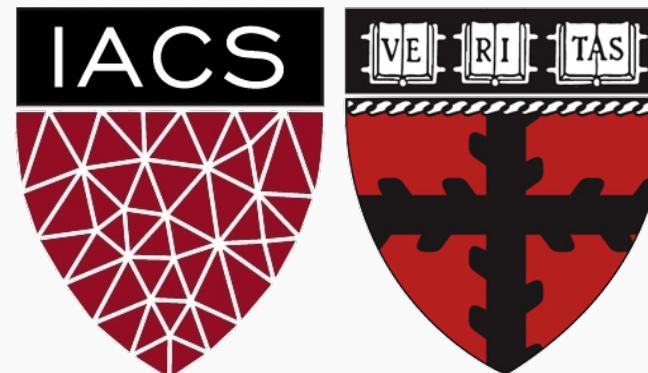


Lecture 21: Optimization and Regularization

CS109A Introduction to Data Science
Pavlos Protopapas, Kevin Rader and Chris Tanner



ANNOUNCEMENTS

- Homework 7 OH:
 - For conceptual questions: Kevin and Chris will continue their office hours.
 - If you have problems with TensorFlow please let us know on ED. We will arrange special OH to help if necessary.
- Project:
 - Milestone3 due on Wed. EDA and base model

Outline

Optimization Regularization of NN



Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout



Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Learning vs. Optimization

Goal of learning: minimize generalization error, or the loss function

$$\mathcal{L}(W) = \mathbb{E}_{(x,y) \sim p_{data}} [L(f(x, W), y)]$$

f is the neural network

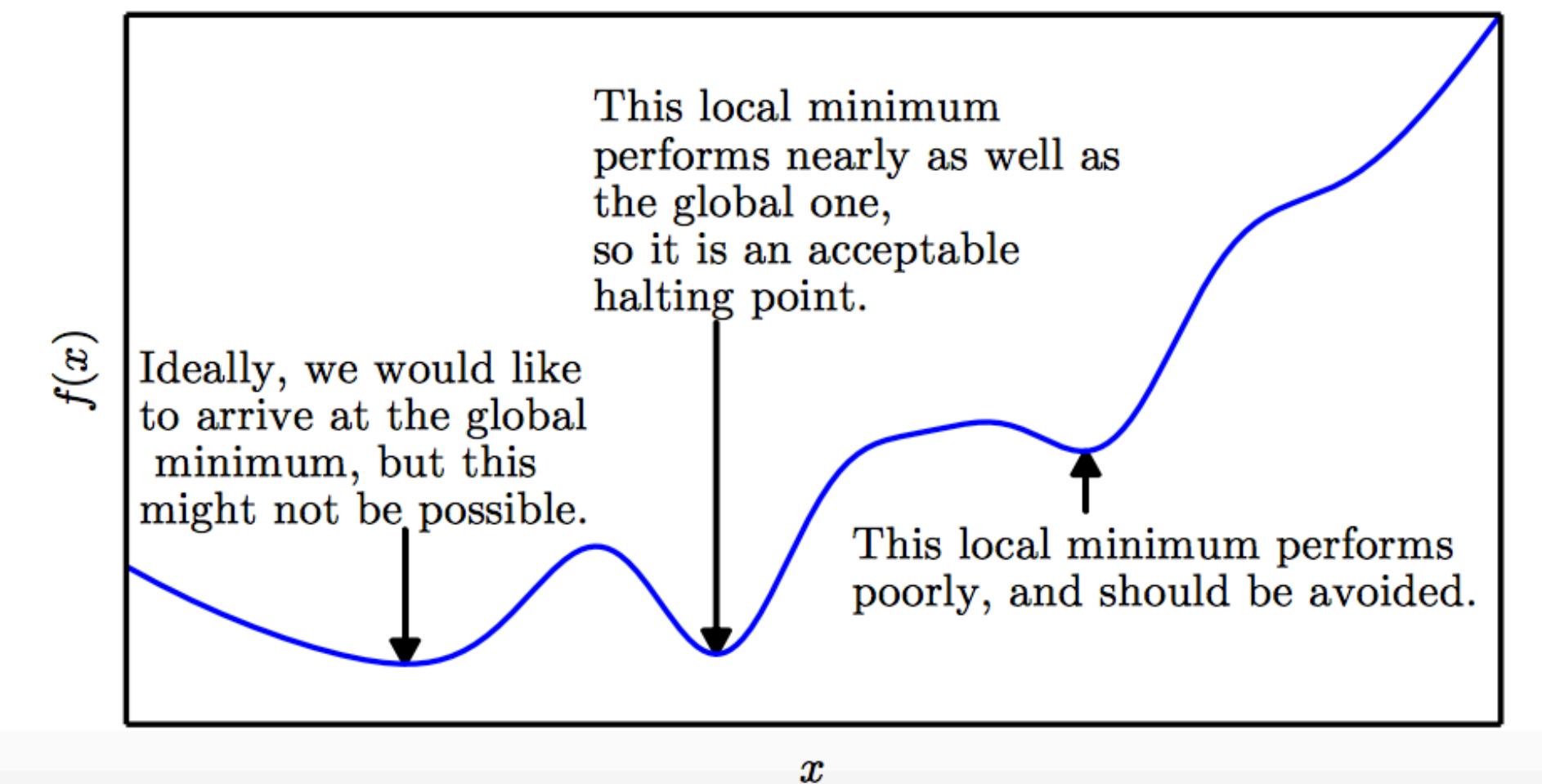
In practice, empirical risk minimization:

$$\mathcal{L}(W) = \sum_i [L(f(x_i; W), y_i)]$$

Quantity optimized
different from the quantity
we care about



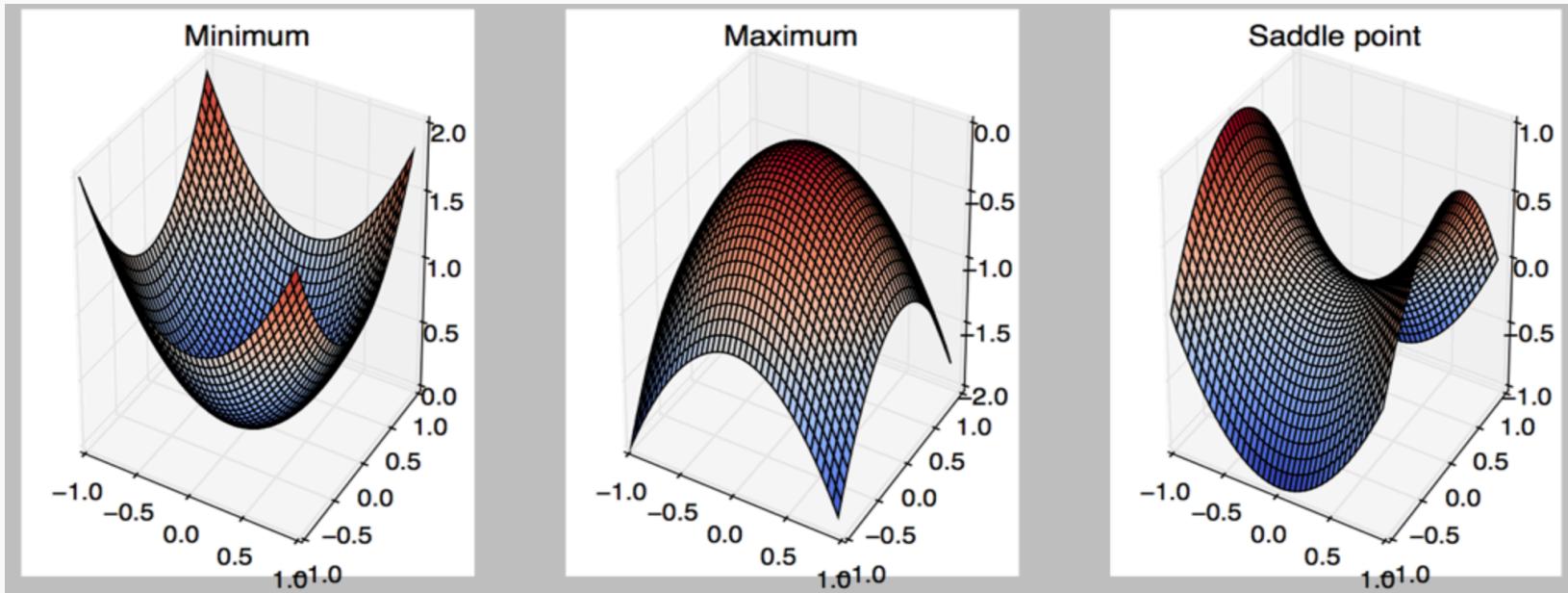
Local Minima



Critical Points

Points with **zero gradient**

2nd-derivate (Hessian) determines curvature



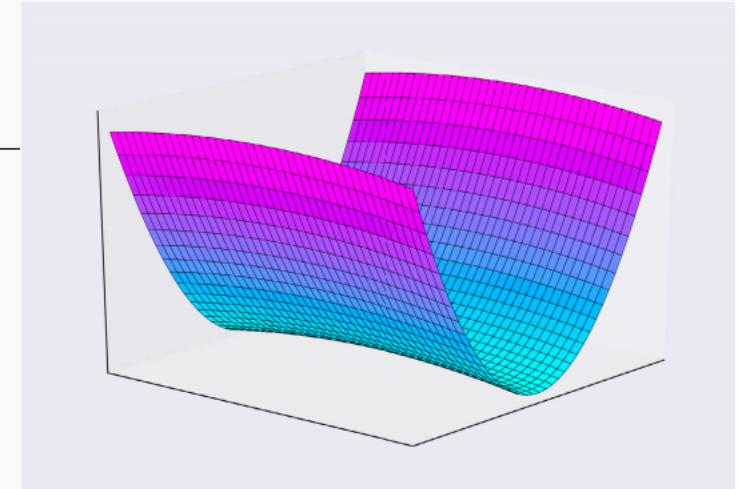
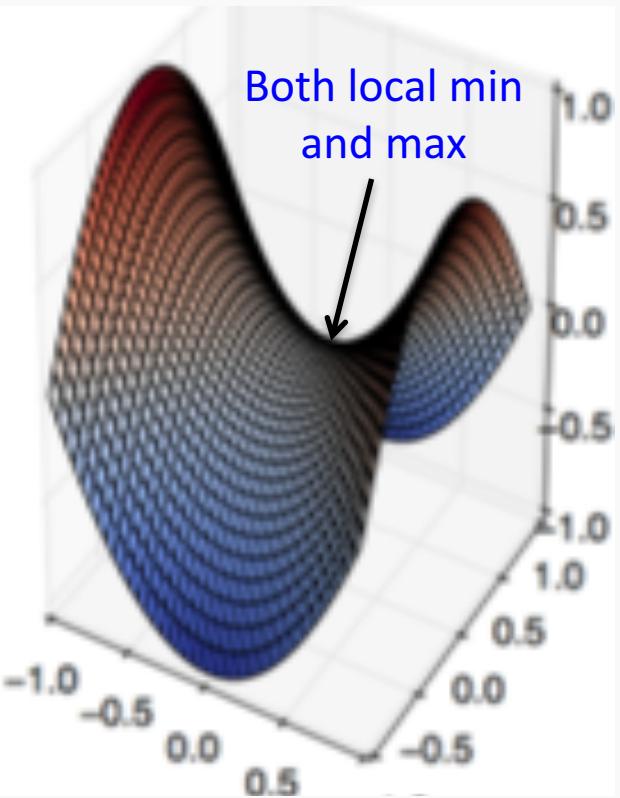
Local Minima

Old view: local minima is major problem in neural network training

Recent view:

- For sufficiently large neural networks, **most local minima incur low cost**
- Not important to find true global minimum

Saddle Points



Recent studies indicate that in high dim, saddle points are more likely than local min

Gradient can be very small near saddle points

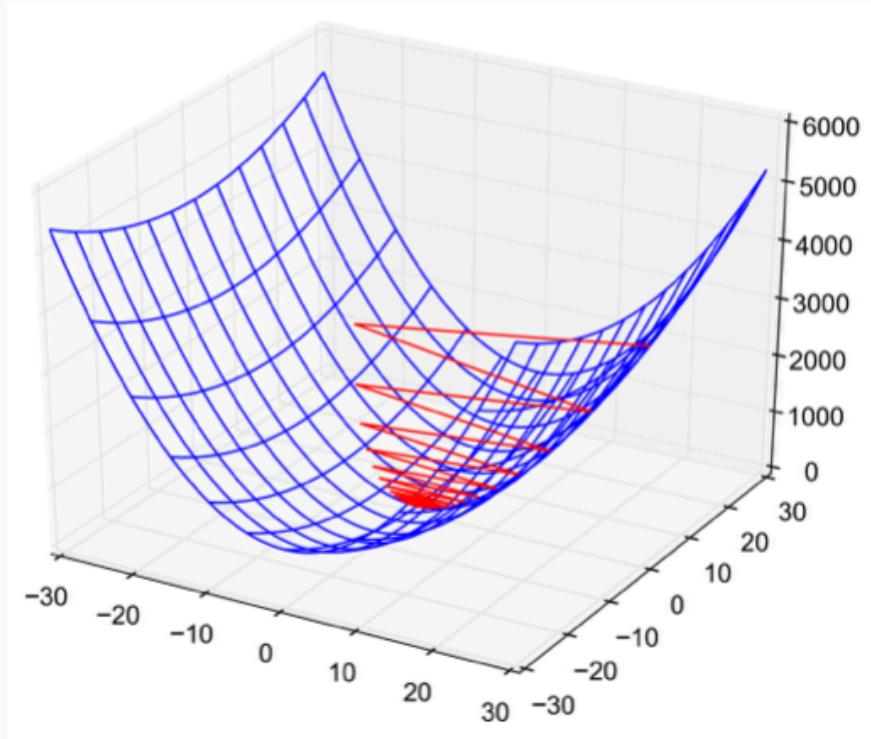
Poor Conditioning

Poorly conditioned Hessian matrix

- High curvature: small steps leads to huge increase

Learning is slow despite strong gradients

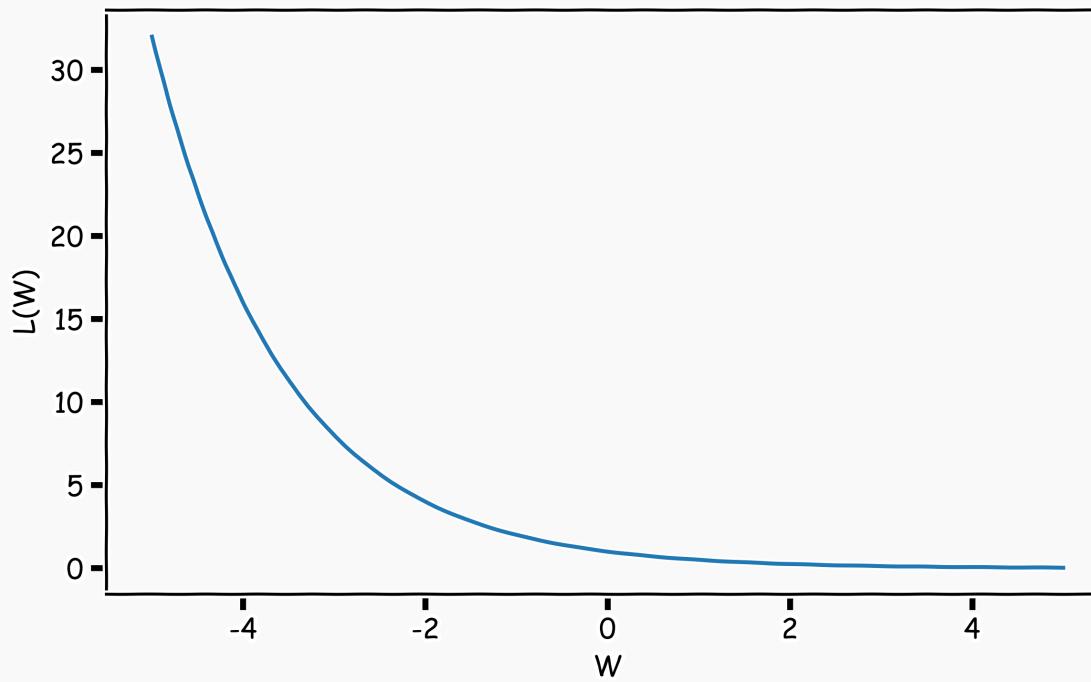
Oscillations slow down progress



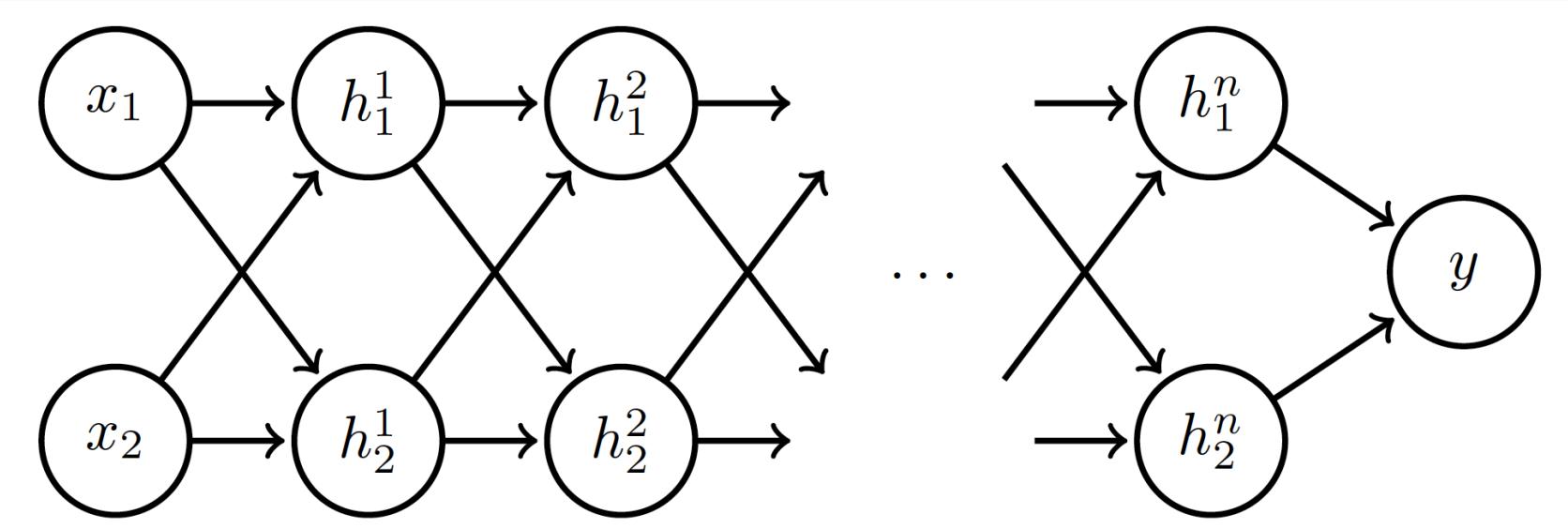
No Critical Points

Some cost functions do not have critical points. In particular classification.

WHY?



Exploding and Vanishing Gradients



Linear activation $h_i = Wx$
 $h_i = Wh_{i-1}, \quad i = 2, \dots, n$

Exploding and Vanishing Gradients

Suppose $\mathbf{W} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$:

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \dots \quad \begin{bmatrix} h_1^n \\ h_2^n \end{bmatrix} = \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Exploding and Vanishing Gradients

Suppose $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Case 1: $a = 1, b = 2$:

$$y \rightarrow 1, \quad \nabla y \rightarrow \begin{bmatrix} n \\ n2^{n-1} \end{bmatrix} \quad \text{Explodes!}$$

Case 2: $a = 0.5, b = 0.9$:

$$y \rightarrow 0, \quad \nabla y \rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{Vanishes!}$$



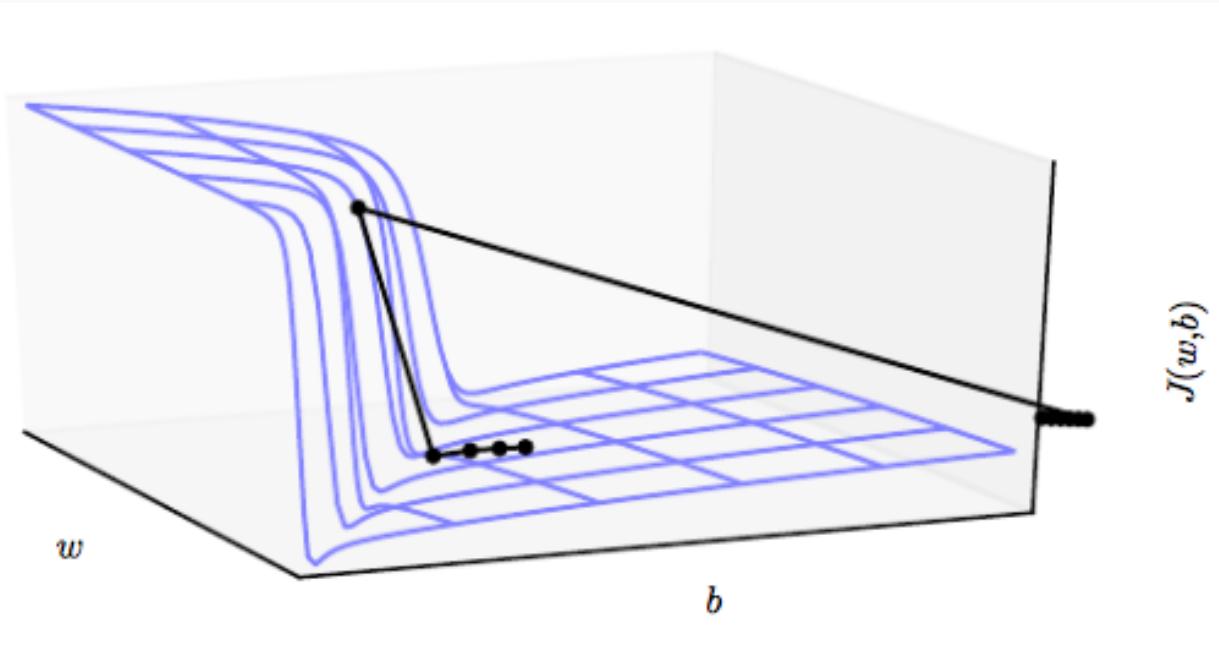
Exploding and Vanishing Gradients

Exploding gradients lead to cliffs

Can be mitigated using **gradient clipping**

if $\|g\| > u$

$$g \leftarrow \frac{gu}{\|g\|}$$



Outline

Optimization

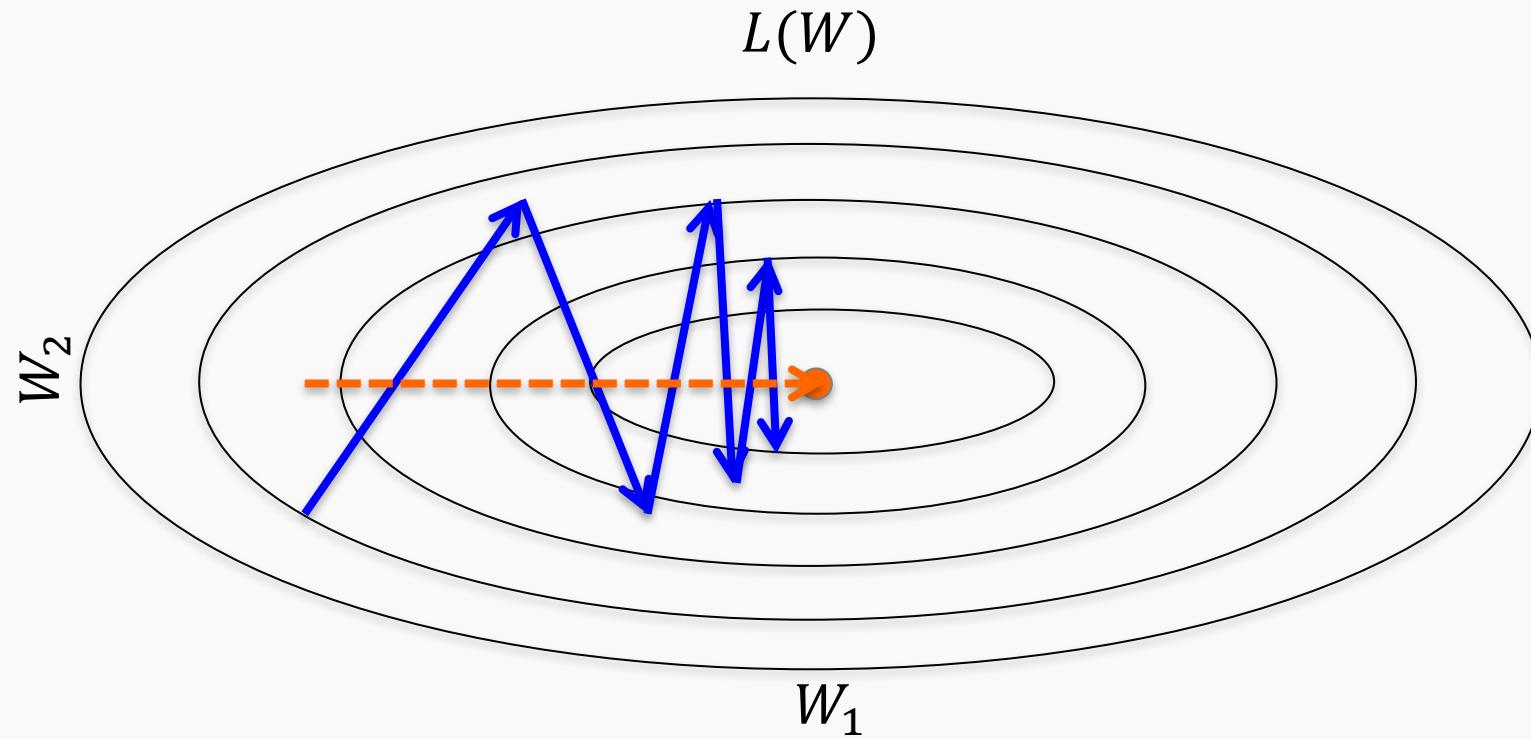
- Challenges in Optimization
- **Momentum**
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Momentum

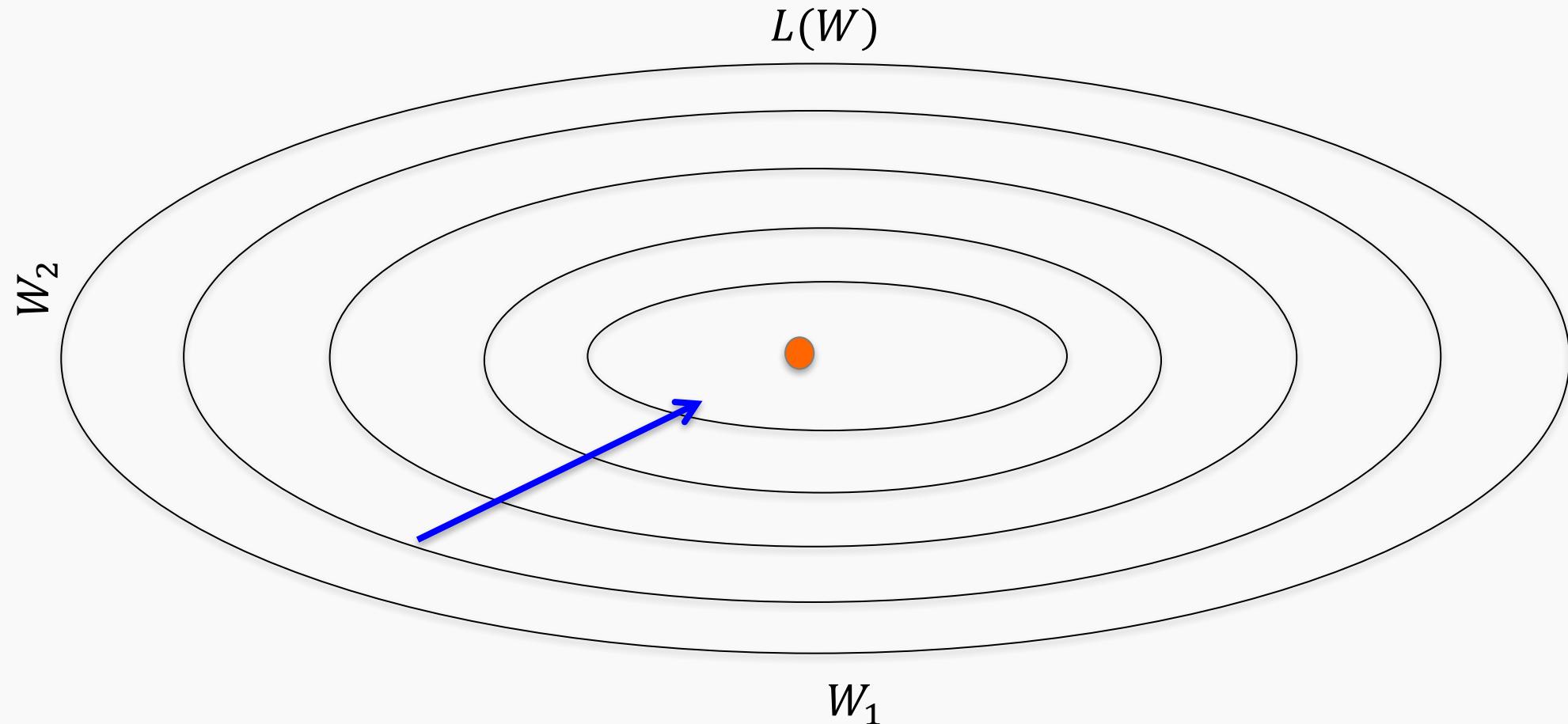
Oscillations because updates do not exploit curvature information



Average gradient presents faster path to optimal: vertical components cancel out

Momentum

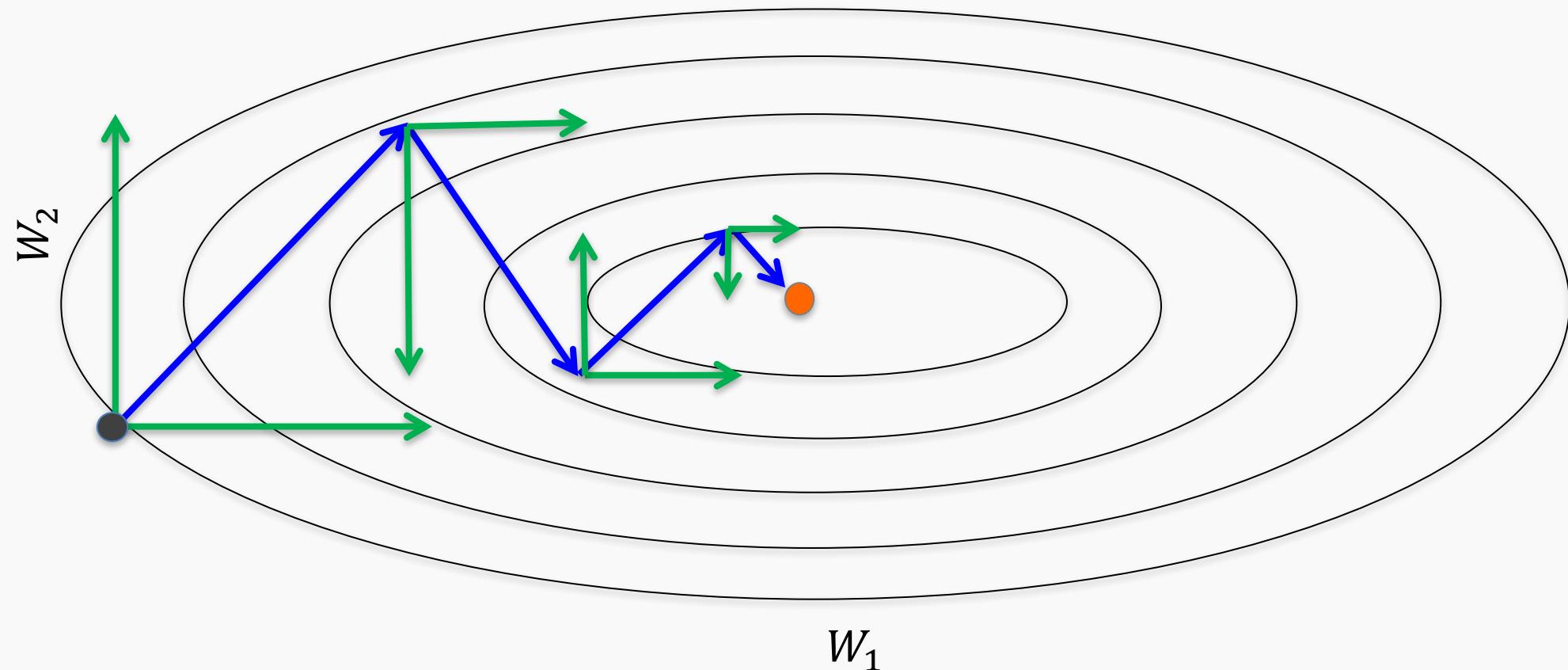
Question: Why not this?



Momentum

Let us figure out an algorithm which will lead us to the minimum faster.

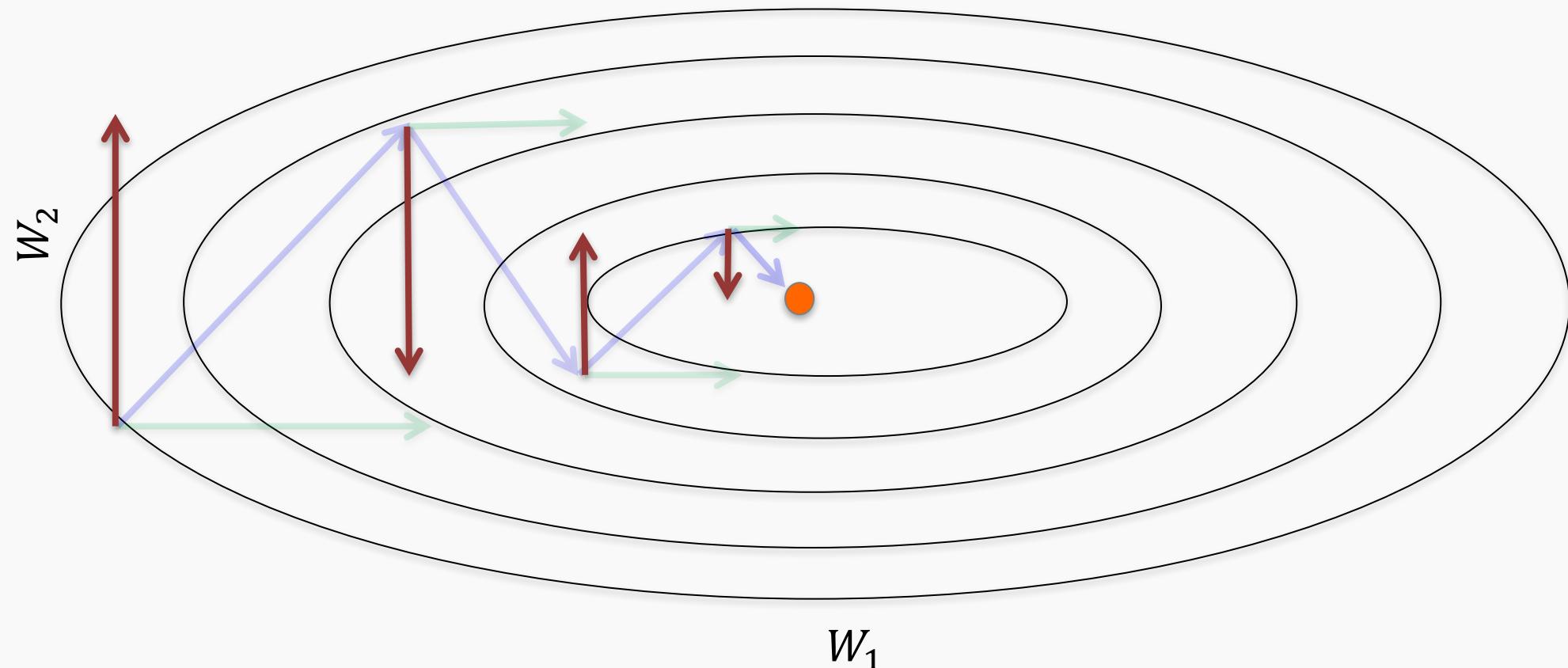
$$L(W)$$



Momentum

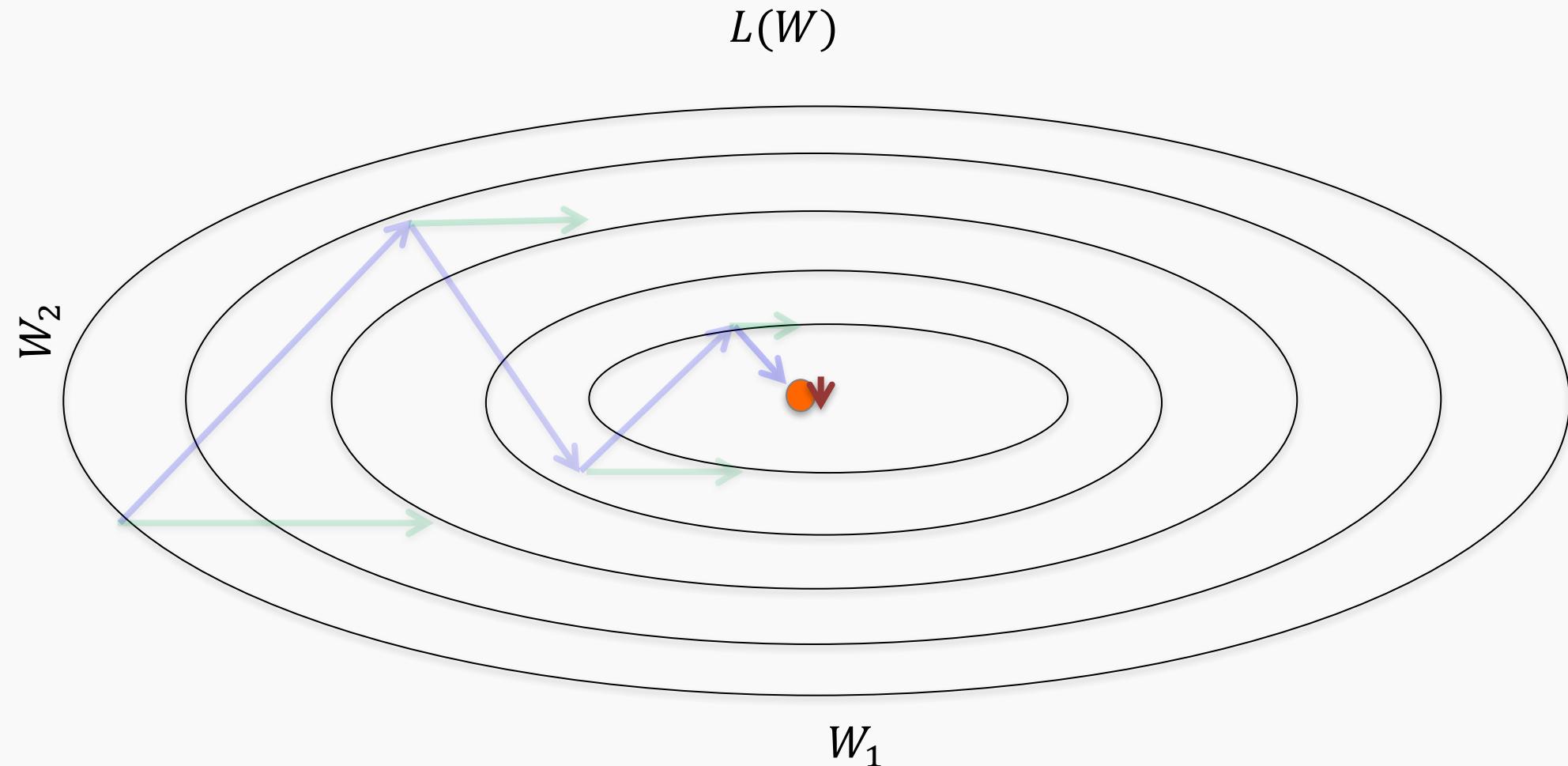
Look each component at a time

$$L(W)$$



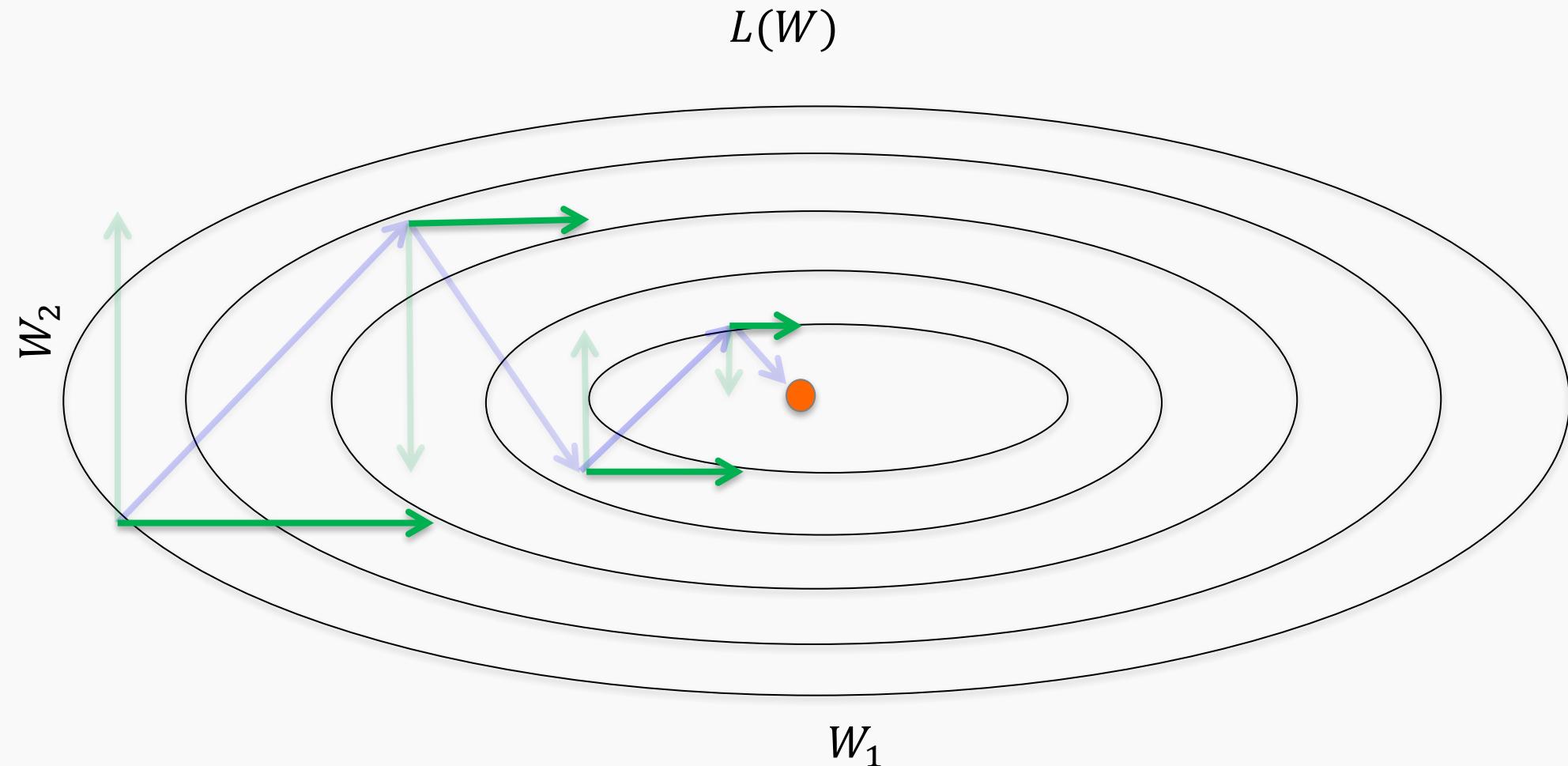
Momentum

Let us figure out an algorithm



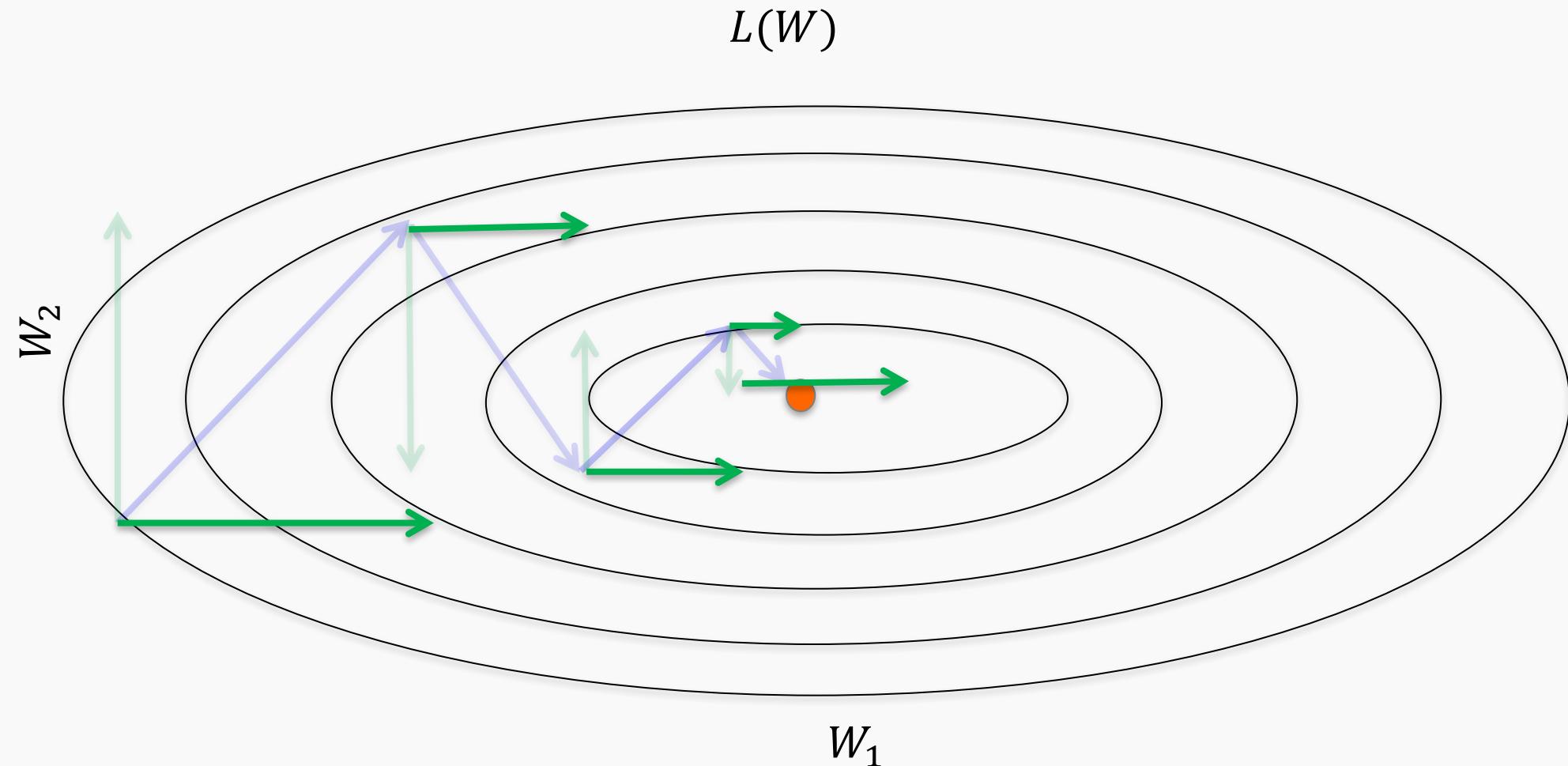
Momentum

Let us figure out an algorithm



Momentum

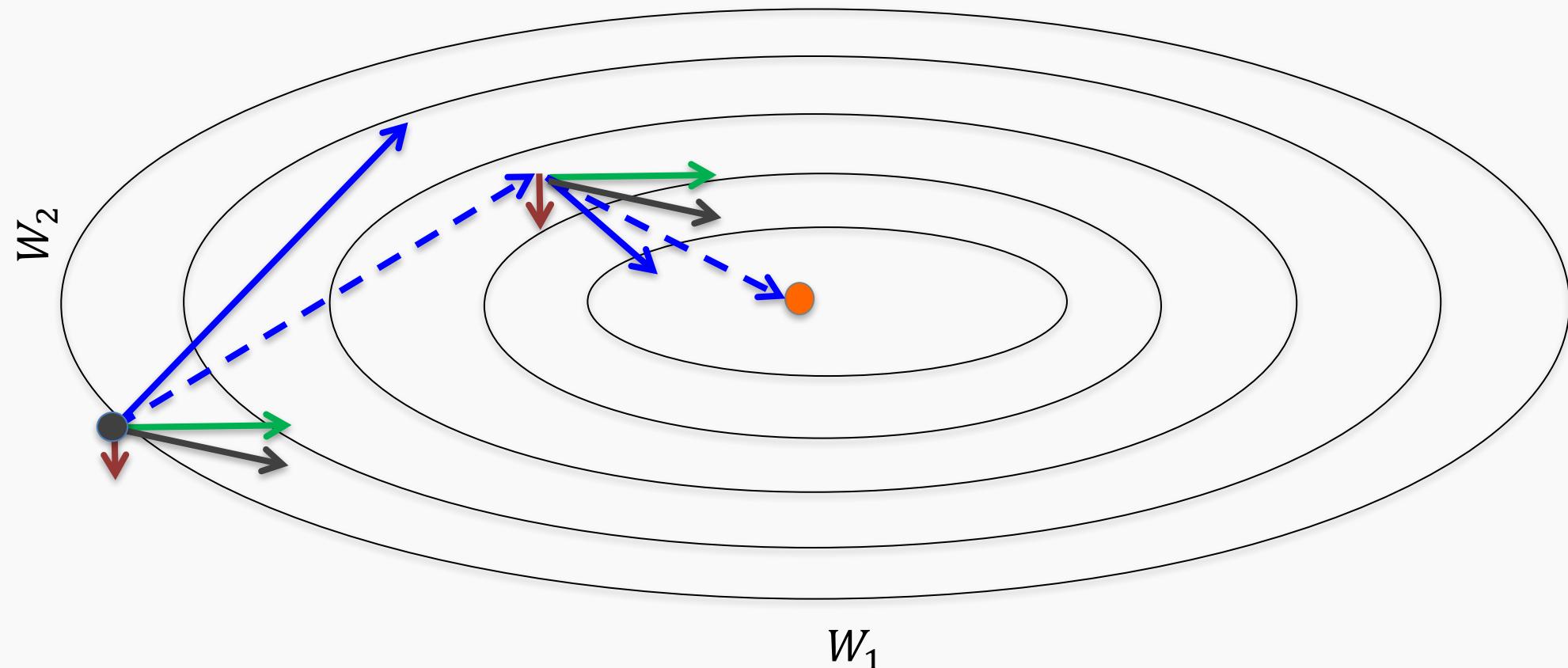
Let us figure out an algorithm



Momentum

Let us figure out an algorithm

$$L(W)$$



Momentum

f is the Neural Network

Old gradient descent:

$$g = \frac{1}{m} \sum_i \nabla_W L(f(x_i; W), y_i)$$

$$W^* = W - \lambda g$$

$g = g + average\ g\ from\ before$

New gradient descent with momentum:

$$\nu = \alpha \nu + (1 - \alpha) g$$

$$W^* = W - \lambda \nu$$

$\alpha \in [0,1)$ controls how quickly
effect of past gradients decay

Nesterov Momentum

Apply an **interim** update:

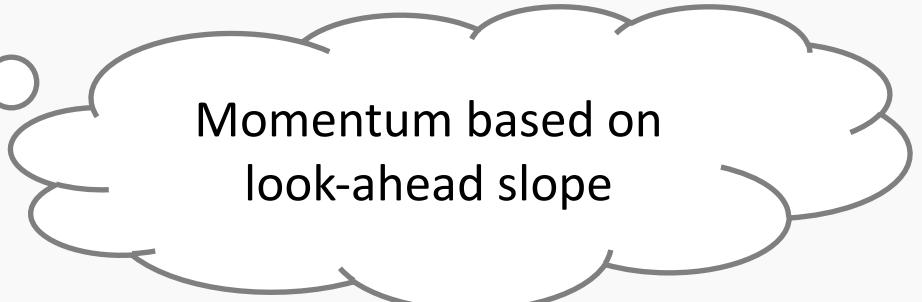
$$\tilde{W} = W + \nu$$

Perform a correction based on gradient at the interim point:

$$g = \frac{1}{m} \sum_i \nabla_W L(f(x_i; \tilde{W}), y_i)$$

$$\nu = \alpha \nu - \varepsilon g$$

$$W = W + \nu$$



Momentum based on
look-ahead slope



► LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

Outline

Optimization

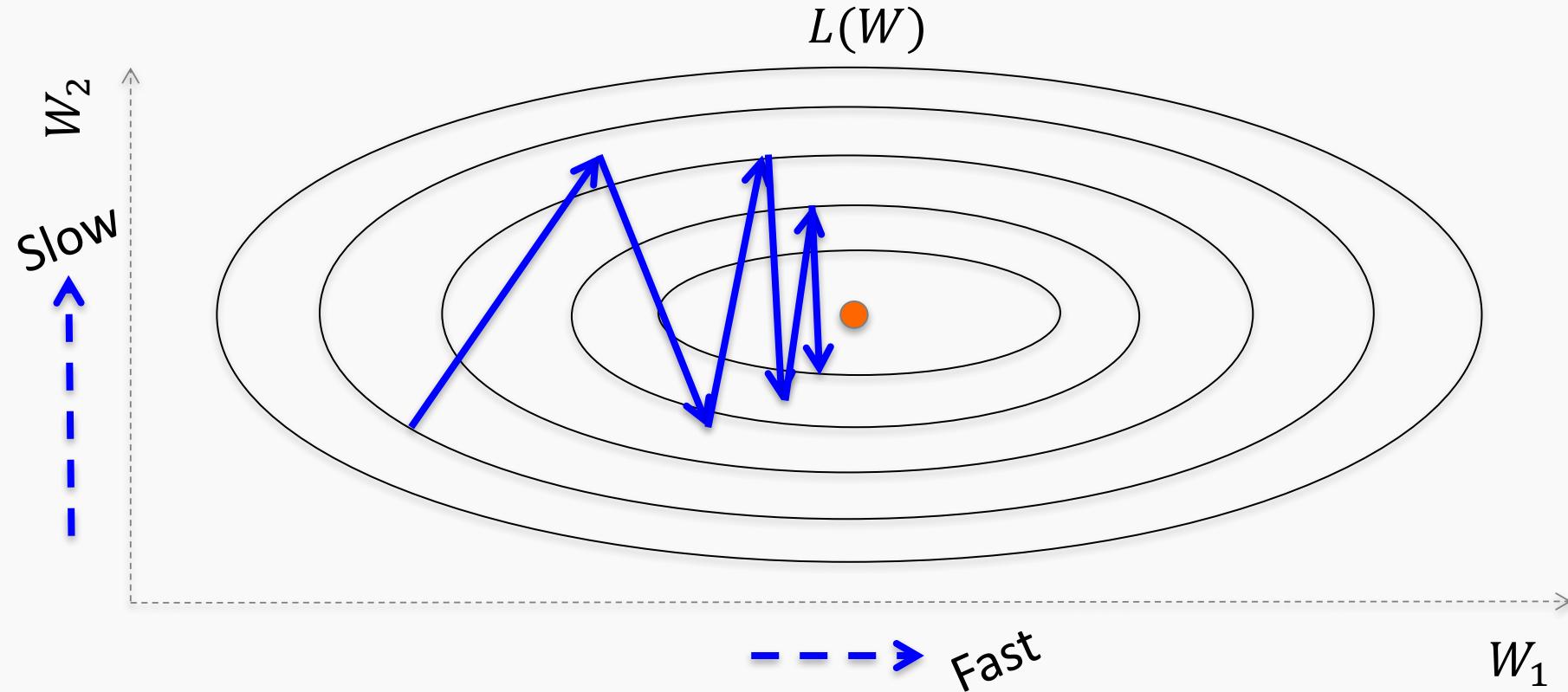
- Challenges in Optimization
- Momentum
- **Adaptive Learning Rate**
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout



Adaptive Learning Rates

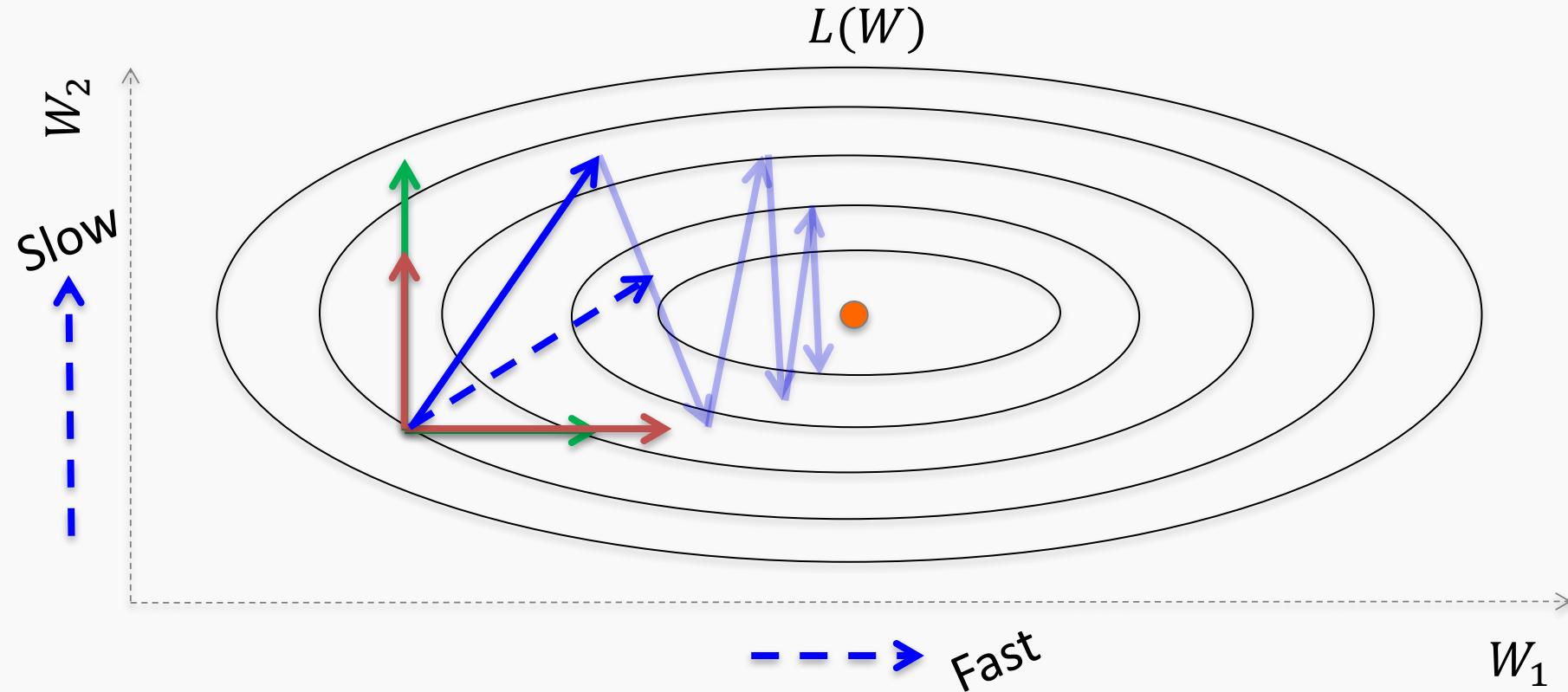


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

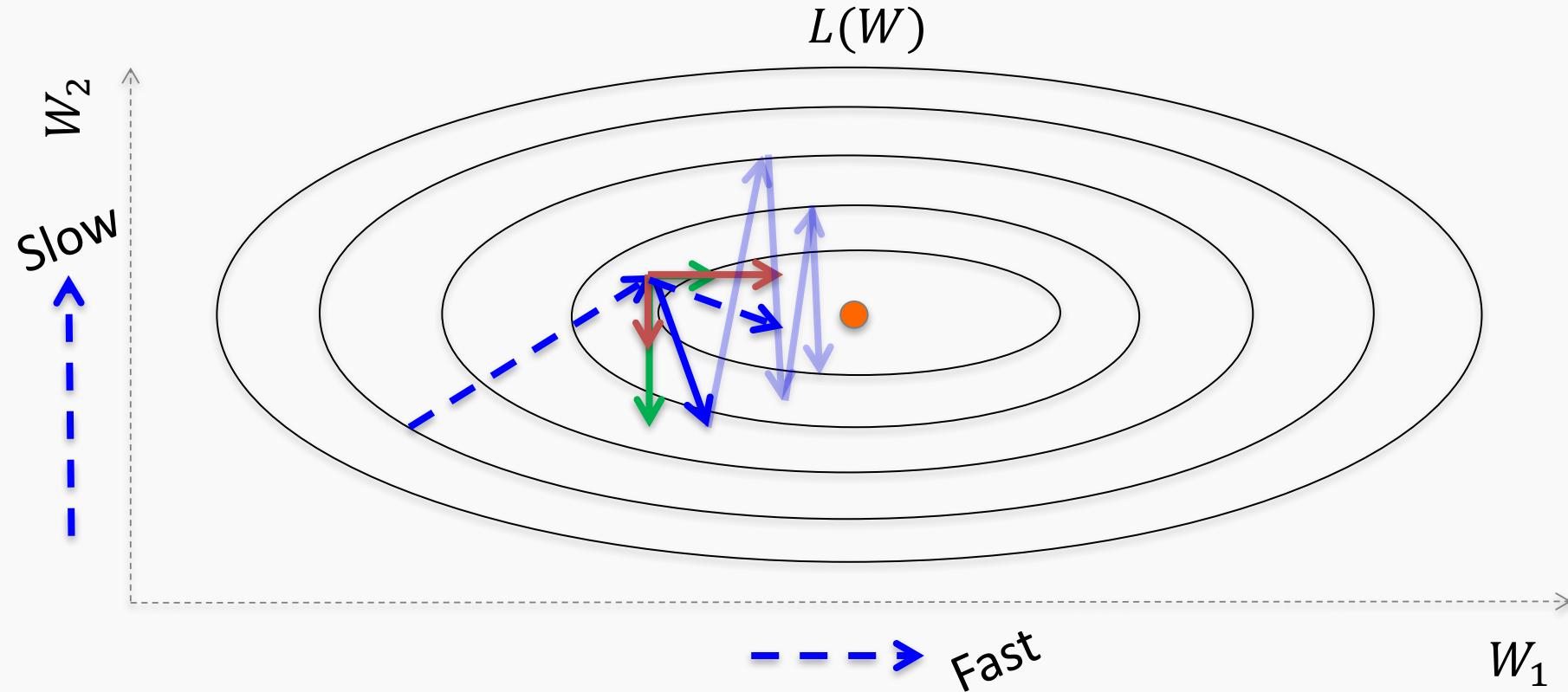


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

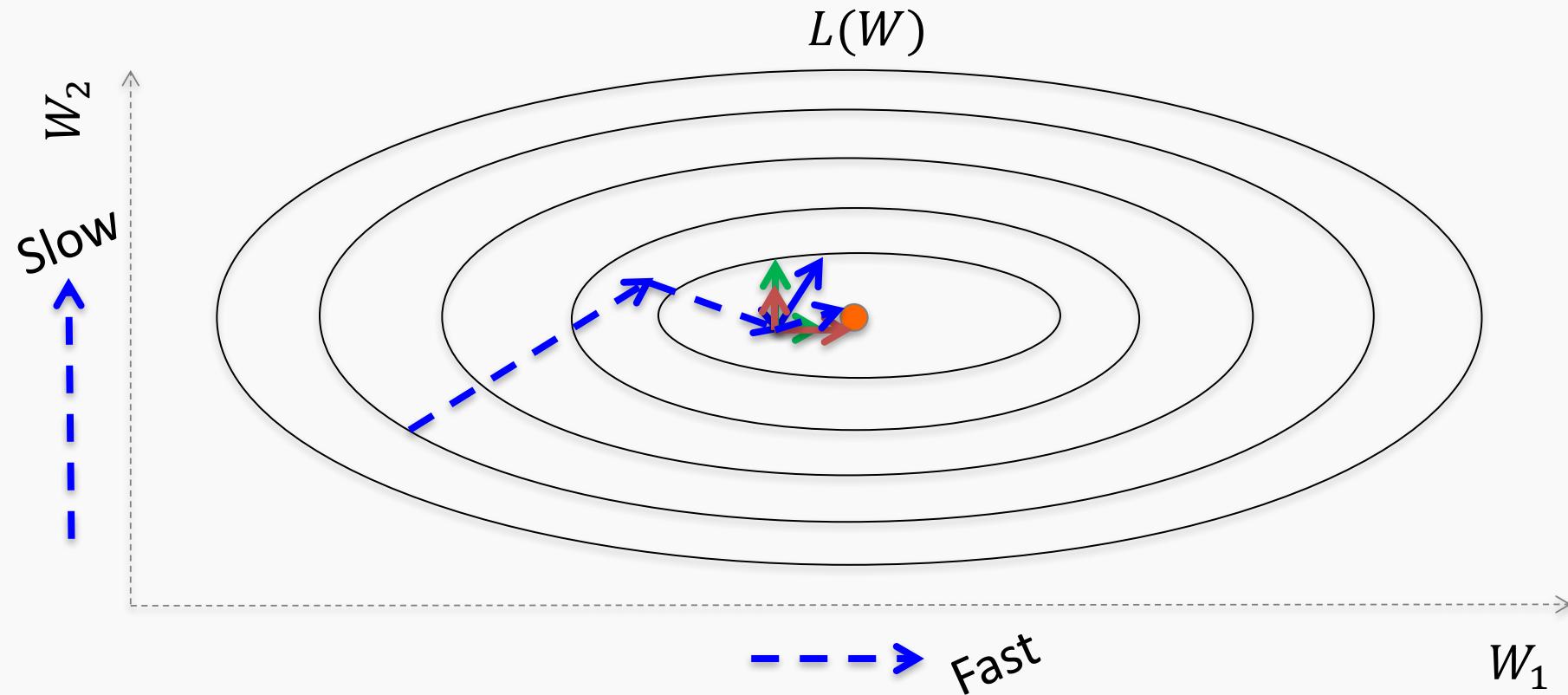


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

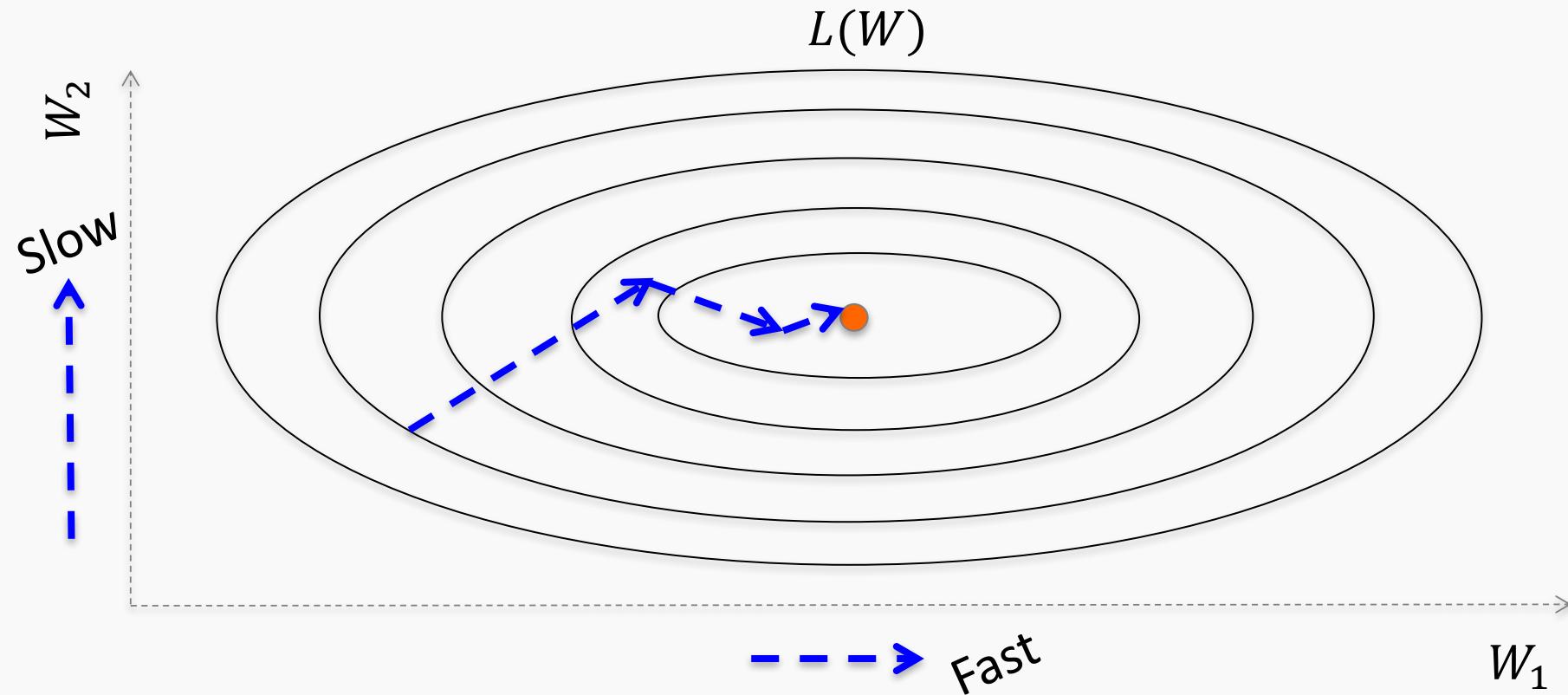


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates



Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

AdaGrad

- Accumulate squared gradients:

$$r_i = r_i + g_i^2$$

g is the gradient

- Update each parameter:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

- Greater progress along gently sloped directions

Inversely proportional to
cumulative squared gradient

AdaGrad

δ is a small number, making sure this does not become too large

Old gradient descent:

$$g = \frac{1}{m} \sum_i \nabla_W L(f(x_i; W), y_i)$$

$$W^*$$

$$W - \lambda g$$

We would like λ 's not to be the same and inversely proportional to the $|g_i|$

$$W_i^* = W_i - \lambda_i g_i$$

$$\lambda_i \propto \frac{1}{|g_i|} = \frac{1}{\delta + |g_i|}$$

New gradient descent with adaptive learning rate:

$$r_i^* = r_i + g_i^2$$

$$W_i^* = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

RMSProp

- For non-convex problems, AdaGrad can prematurely decrease learning rate
- Use **exponentially weighted average** for gradient accumulation

$$r_i = \rho r_i + (1 - \rho) g_i^2$$

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

Adam

- RMSProp + Momentum
- Estimate first moment:

$$v_i = \rho_1 v_i + (1 - \rho_1) g_i$$

Also applies
bias correction
to v and r

- Estimate second moment:

$$r_i = \rho_2 r_i + (1 - \rho_2) g_i^2$$

- Update parameters:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} v_i$$

Works well in practice,
is fairly robust to
hyper-parameters

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- **Parameter Initialization**
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Parameter Initialization

- Goal: **break symmetry** between units
 - so that each unit computes a different function
- Initialize all weights (not biases) **randomly**
 - Gaussian or uniform distribution
- **Scale of initialization?**
 - Large \rightarrow grad explosion, Small \rightarrow grad vanishing



Xavier Initialization

- Heuristic for all outputs to have **unit variance**
- For a fully-connected layer with m inputs:

$$W_{ij} \sim N\left(0, \frac{1}{m}\right)$$

- For ReLU units, it is recommended:

$$W_{ij} \sim N\left(0, \frac{2}{m}\right)$$

Normalized Initialization

- Fully-connected layer with m inputs, n outputs:

$$W_{ij} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

- Heuristic trades off between initialize all layers have same activation and gradient variance
- **Sparse** variant when m is large
 - Initialize k nonzero weights in each unit



► LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout



Feature Normalization

Good practice to normalize features before applying learning algorithm:

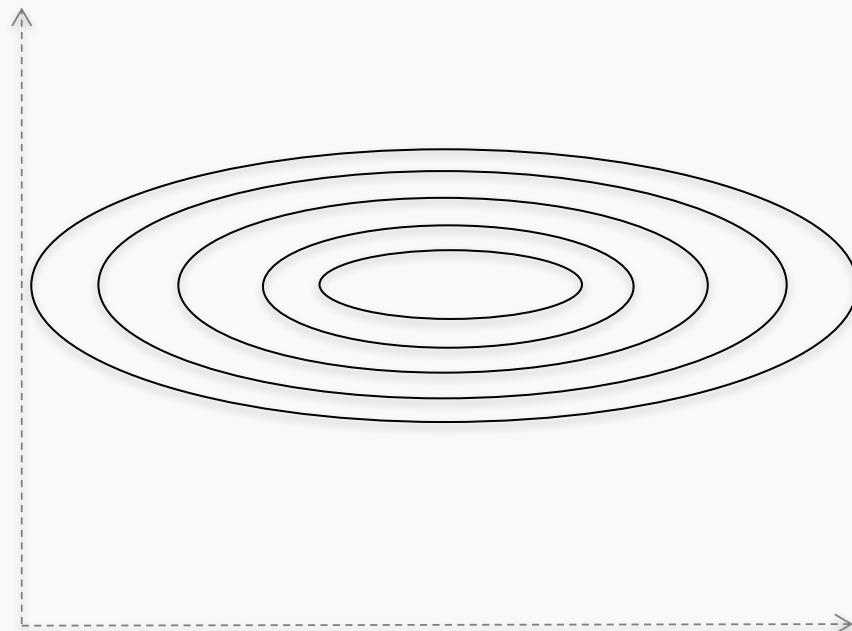
$$x' = \frac{x - \mu}{\sigma}$$

Feature vector x Vector of mean feature values
 μ
 σ Vector of SD of feature values

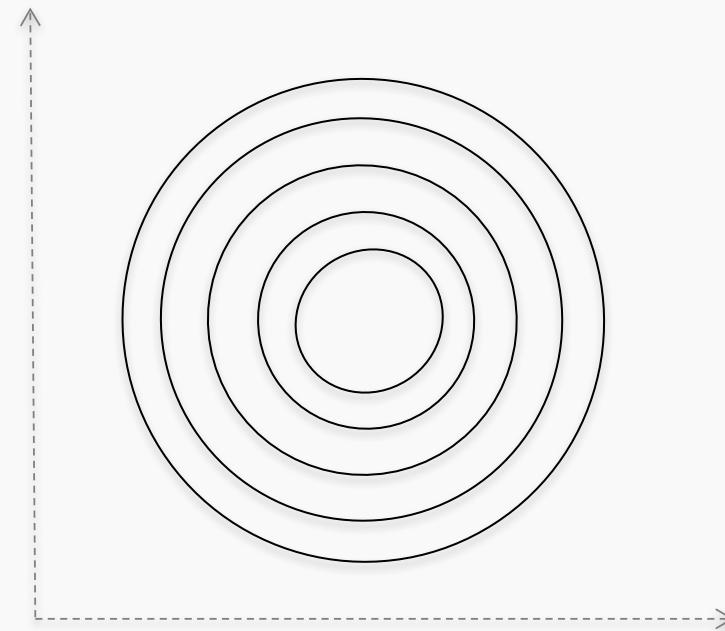
Features in **same scale**: mean 0 and variance 1

- Speeds up learning

Feature Normalization



Before normalization

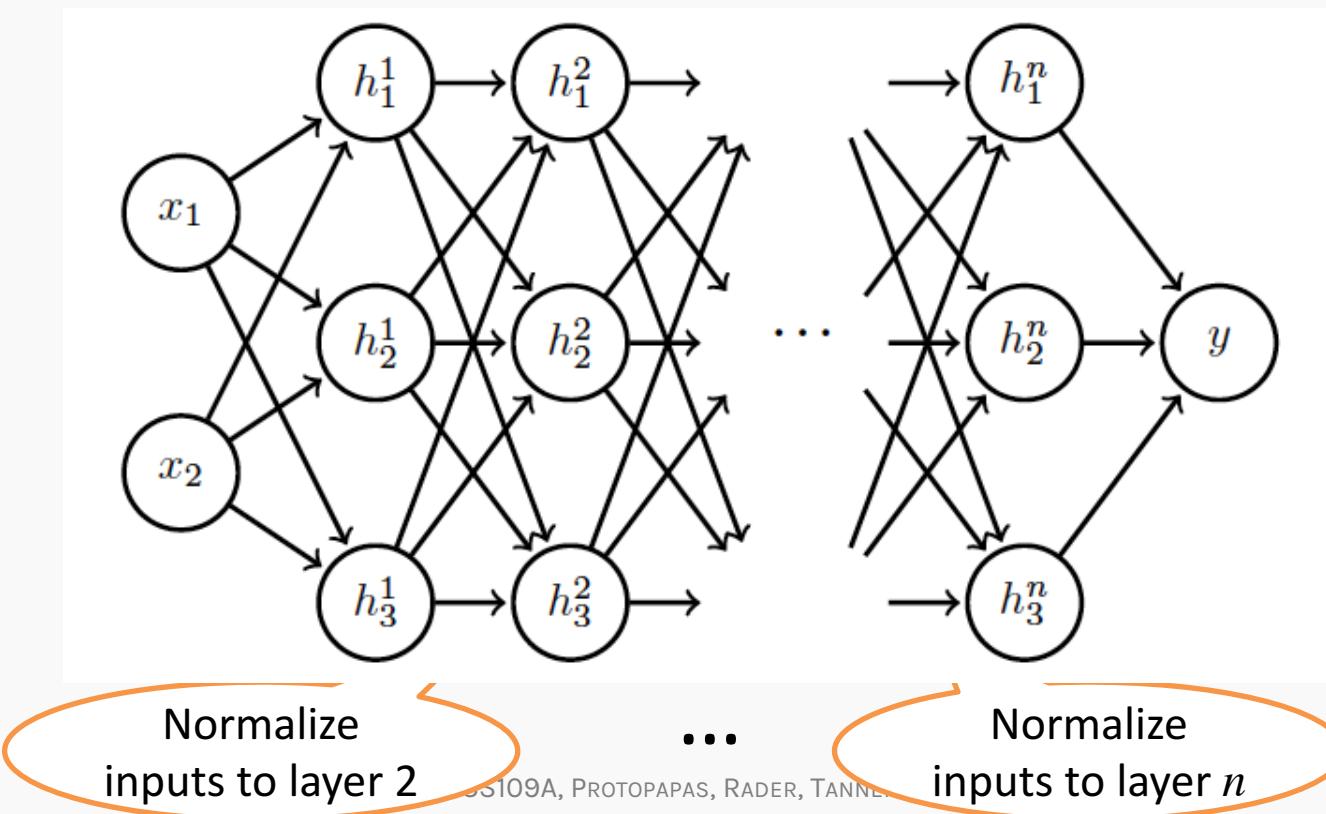


After normalization

$$L(W)$$

Internal Covariance Shift

Each hidden layer changes distribution of inputs to next layer: slows down learning



Batch Normalization

Training time:

- Mini-batch of activations for layer to normalize

$$H = \begin{bmatrix} H_{11} & \cdots & H_{1K} \\ \vdots & \ddots & \vdots \\ H_{N1} & \cdots & H_{NK} \end{bmatrix}$$

K hidden layer activations

N data points in mini-batch



Batch Normalization

Training time:

- Mini-batch of activations for layer to normalize

where

$$H' = \frac{H - \mu}{\sigma}$$

$$\mu = \frac{1}{m} \sum_i H_{i,:}$$

Vector of mean activations
across mini-batch

$$\sigma = \sqrt{\frac{1}{m} \sum_i (H - \mu)_i^2 + \delta}$$

Vector of SD of each unit
across mini-batch



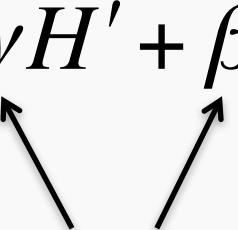
Batch Normalization

Training time:

- Normalization can reduce expressive power
- Instead use:

$$\gamma H' + \beta$$

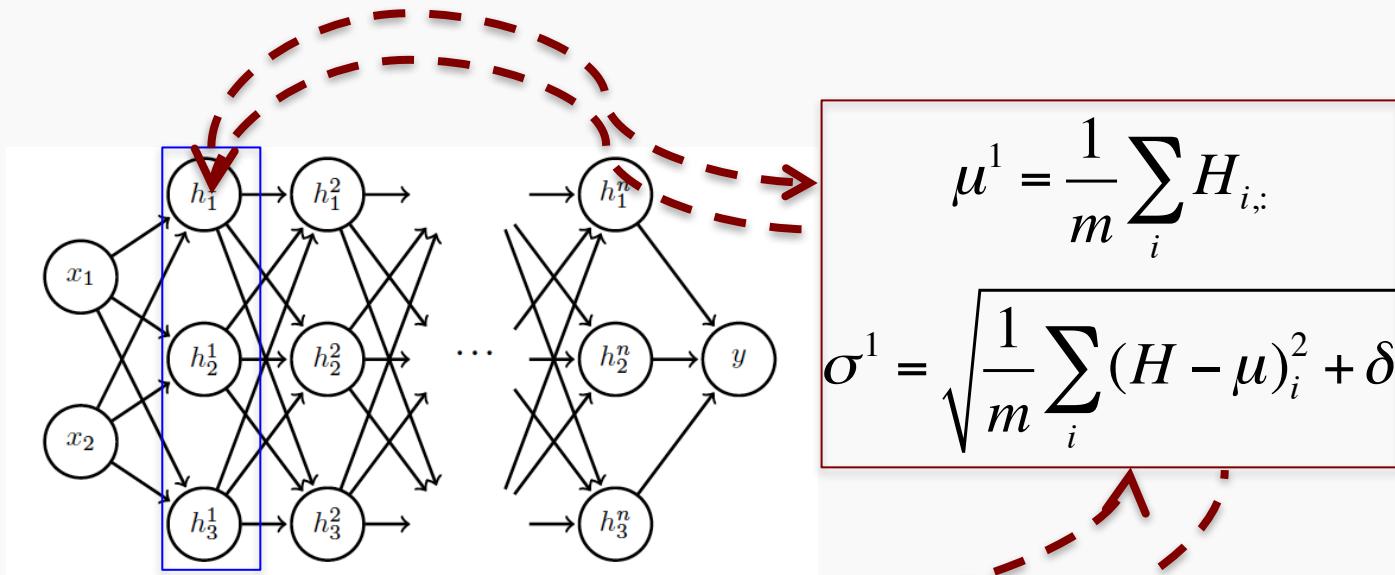
Learnable parameters



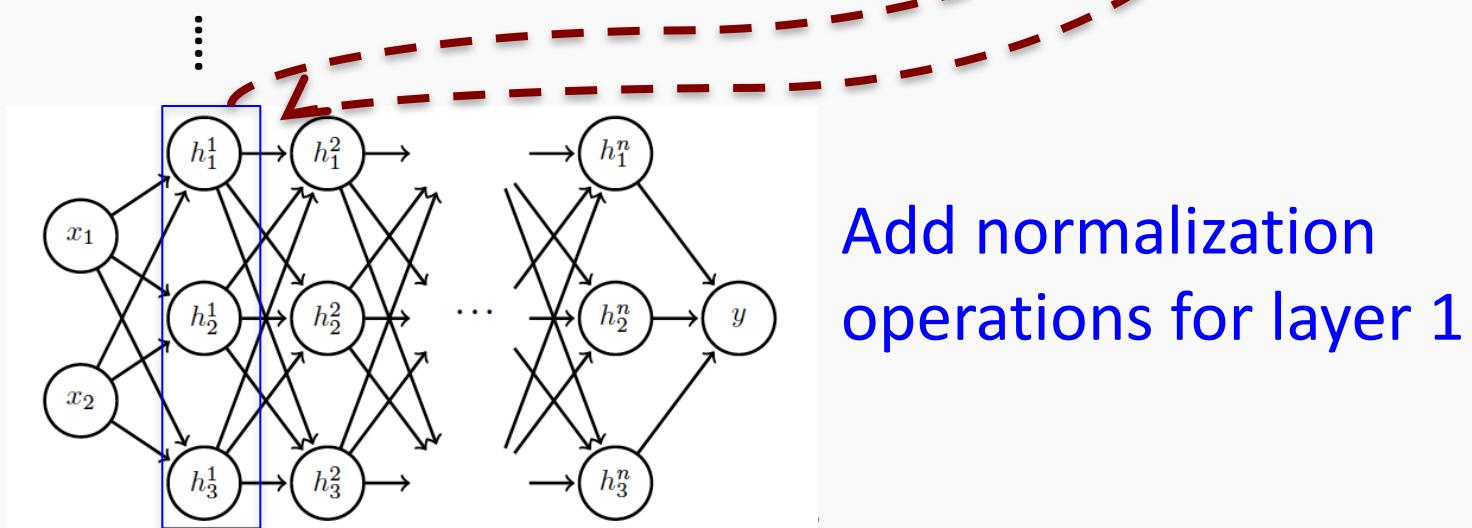
- Allows network to **control range of normalization**

Batch Normalization

Batch 1

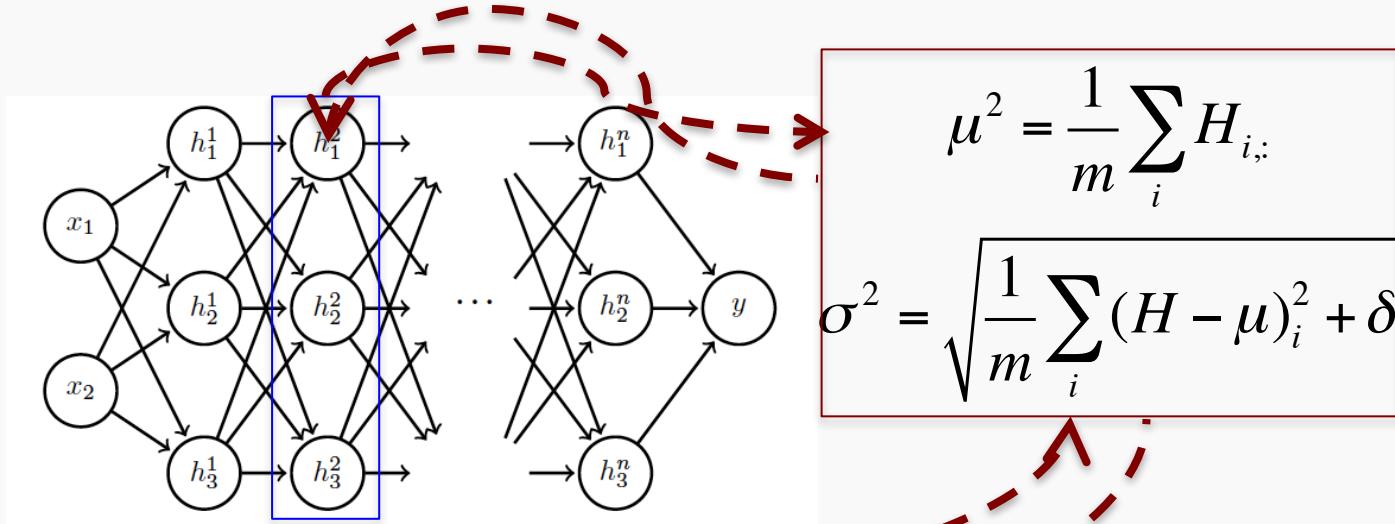


Batch N

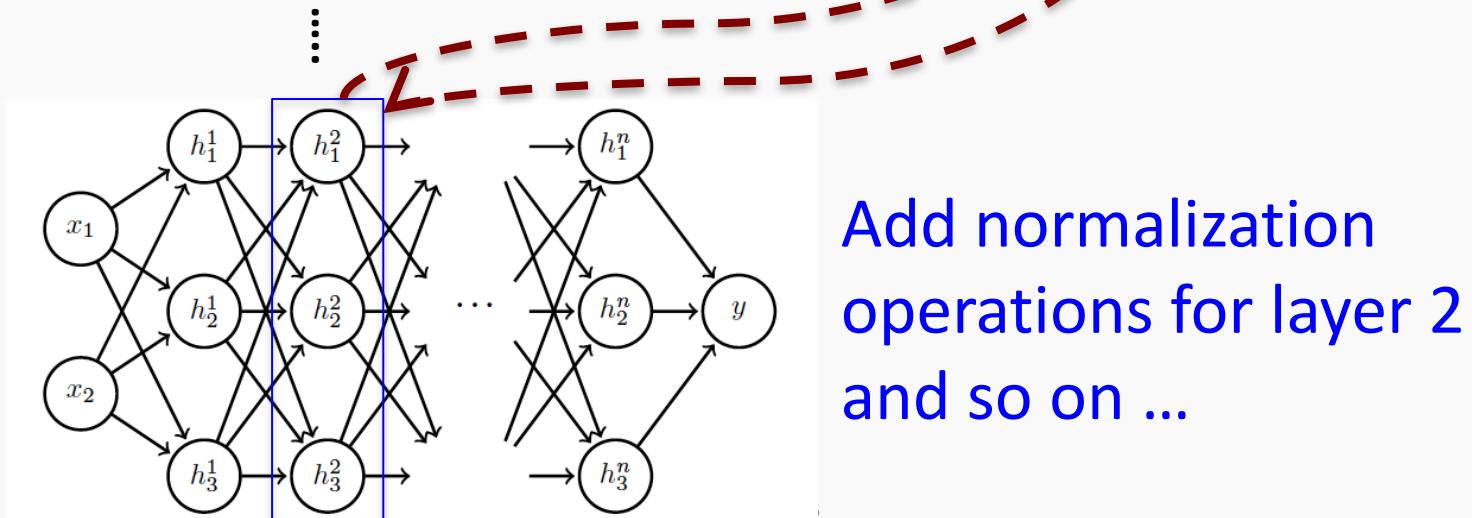


Batch Normalization

Batch 1



Batch N



Batch Normalization

Differentiate the **joint loss** for N mini-batches

Back-propagate through the norm operations

Test time:

- Model needs to be evaluated on a *single* example
- Replace μ and σ with **running averages** collected during training



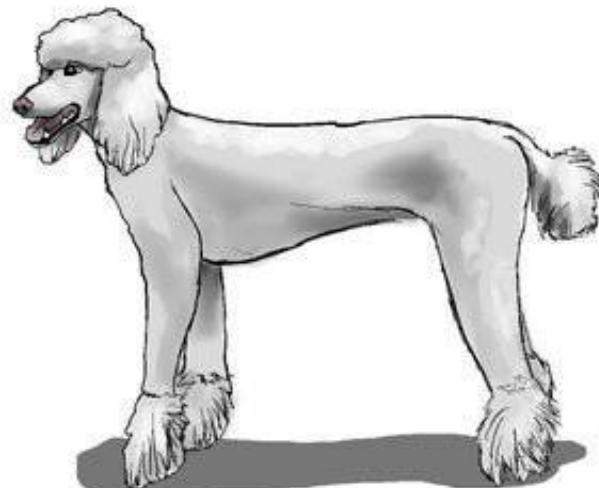
Scalar



Vector



Matrix



Tensor



Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Regularization

Regularization is any modification we make to a learning algorithm that is intended to **reduce its generalization error** but not its training error.

Outline

Optimization

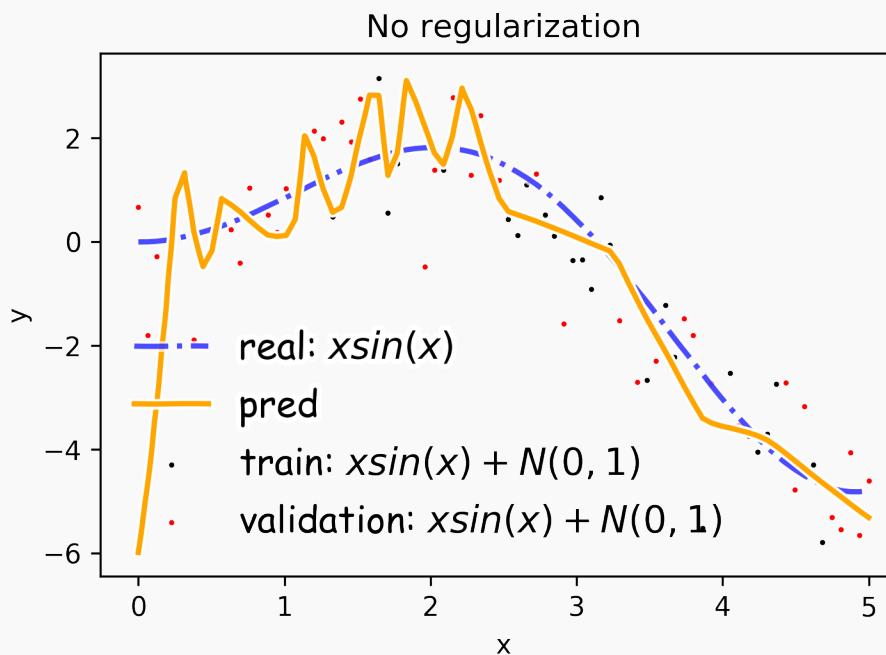
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Overfitting

Fitting a deep neural network with 5 layers and 100 neurons per layer can lead to a very good prediction on the training set but poor prediction on validations set.



Norm Penalties

We used to optimize:

$$L(W; X, y)$$

Change to ...

$$L_R(W; X, y) = L(W; X, y) + \alpha \Omega(W)$$



L_2 regularization:

- Weights decay
- MAP estimation with Gaussian prior

$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$

L_1 regularization:

- encourages sparsity
- MAP estimation with Laplacian prior

$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Norm Penalties

We used to optimize:

Change to ...

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial L}{\partial W}$$

$$L_R(W; X, y) = L(W; X, y) + \frac{1}{2} \alpha W^2$$

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial L}{\partial W} \boxed{- \lambda \alpha W}$$

Weights decay
in proportion
to size

Biases not
penalized

L_2 regularization:

- Decay of weights
- MAP estimation with Gaussian prior

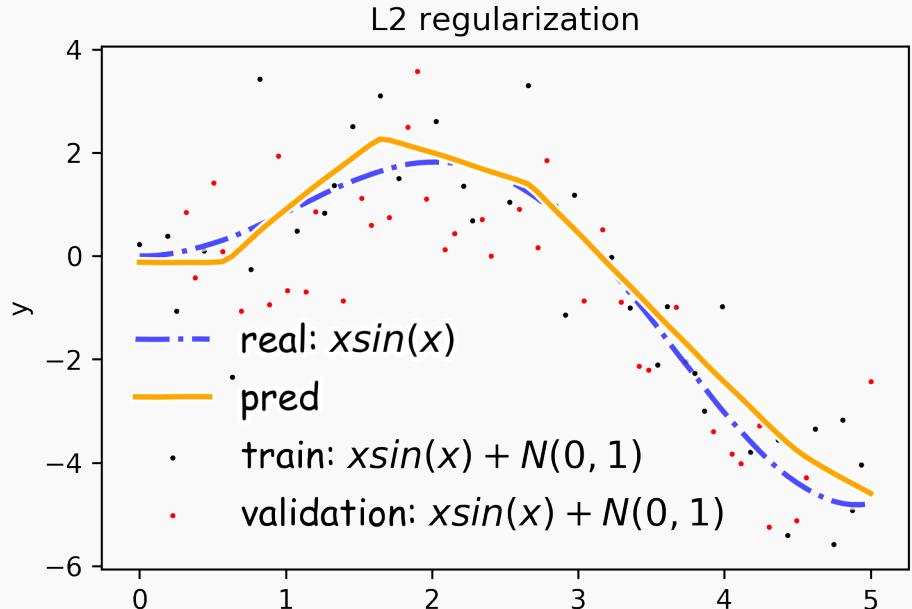
L_1 regularization:

- encourages sparsity
- MAP estimation with Laplacian prior

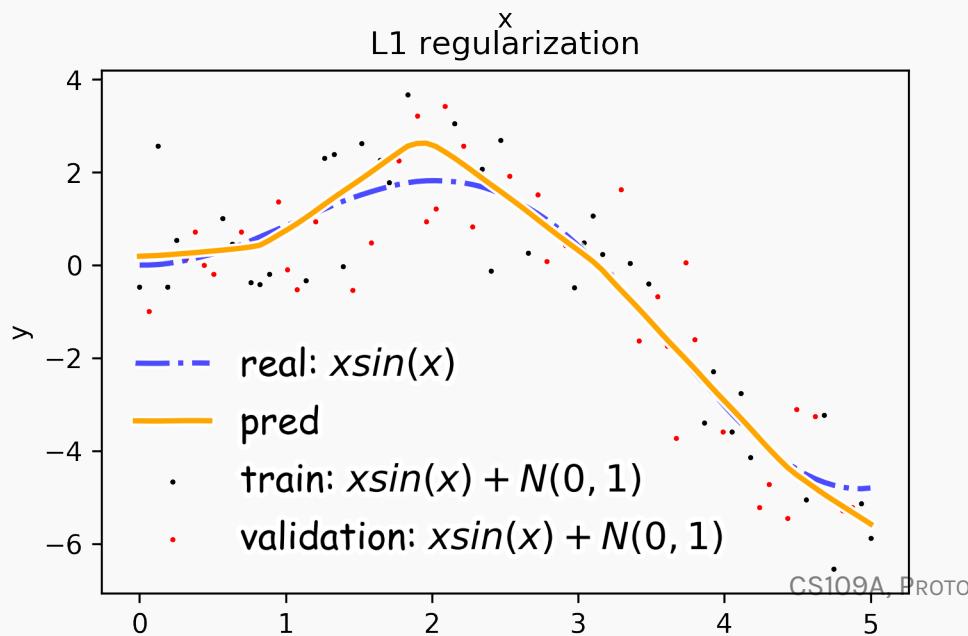
$$\Omega(W) = \frac{1}{2} \|W\|_2^2$$

$$\Omega(W) = \frac{1}{2} \|W\|_1$$

Norm Penalties



$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$



$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Norm Penalties as Constraints

$$\min_{\Omega(W) \leq K} J(W; X, y)$$

Useful if K is known in advance

Optimization:

- Construct Lagrangian and apply gradient descent
- Projected gradient descent



Outline

Optimization

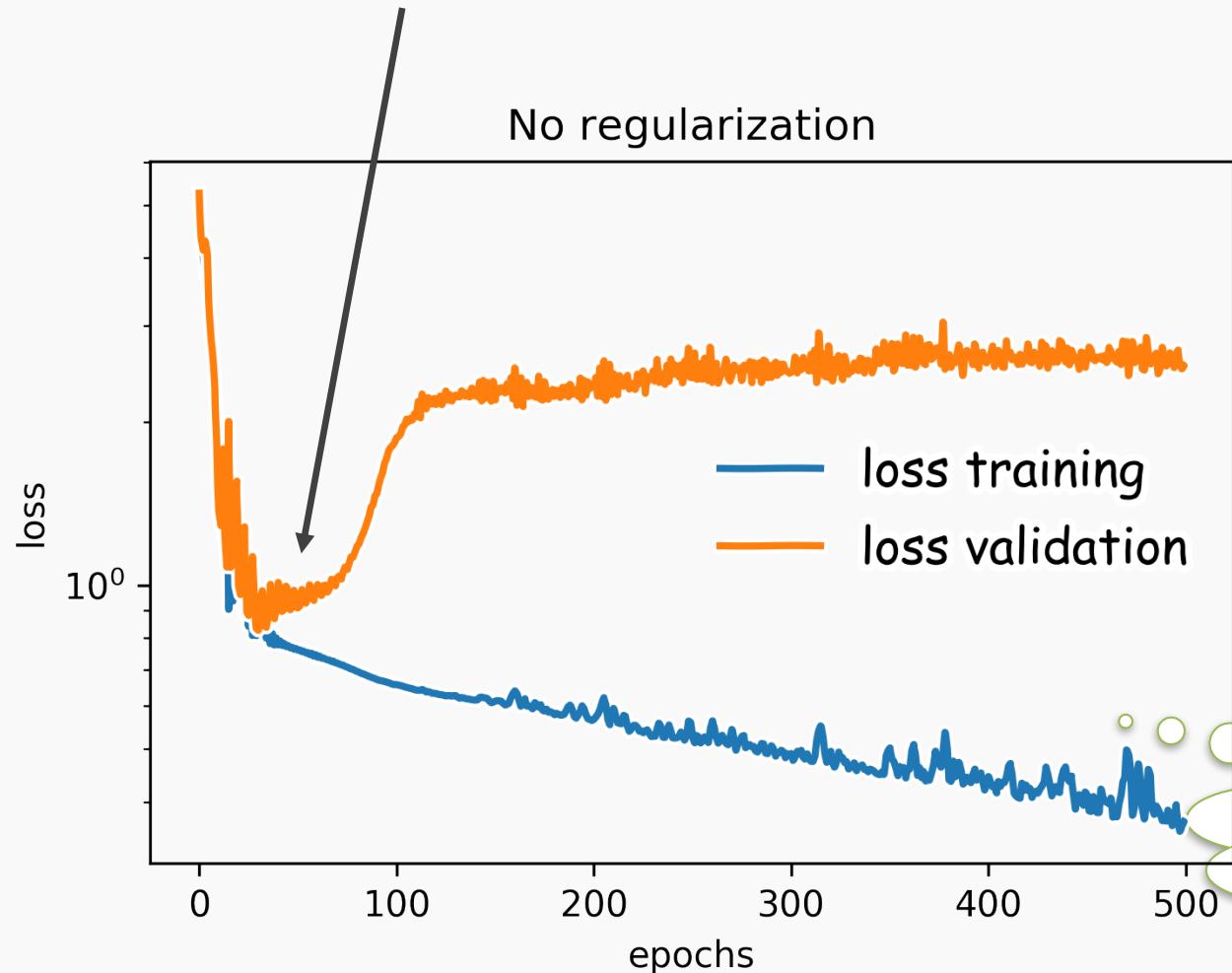
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- **Early Stopping**
- Data Augmentation
- Sparse Representation
- Dropout

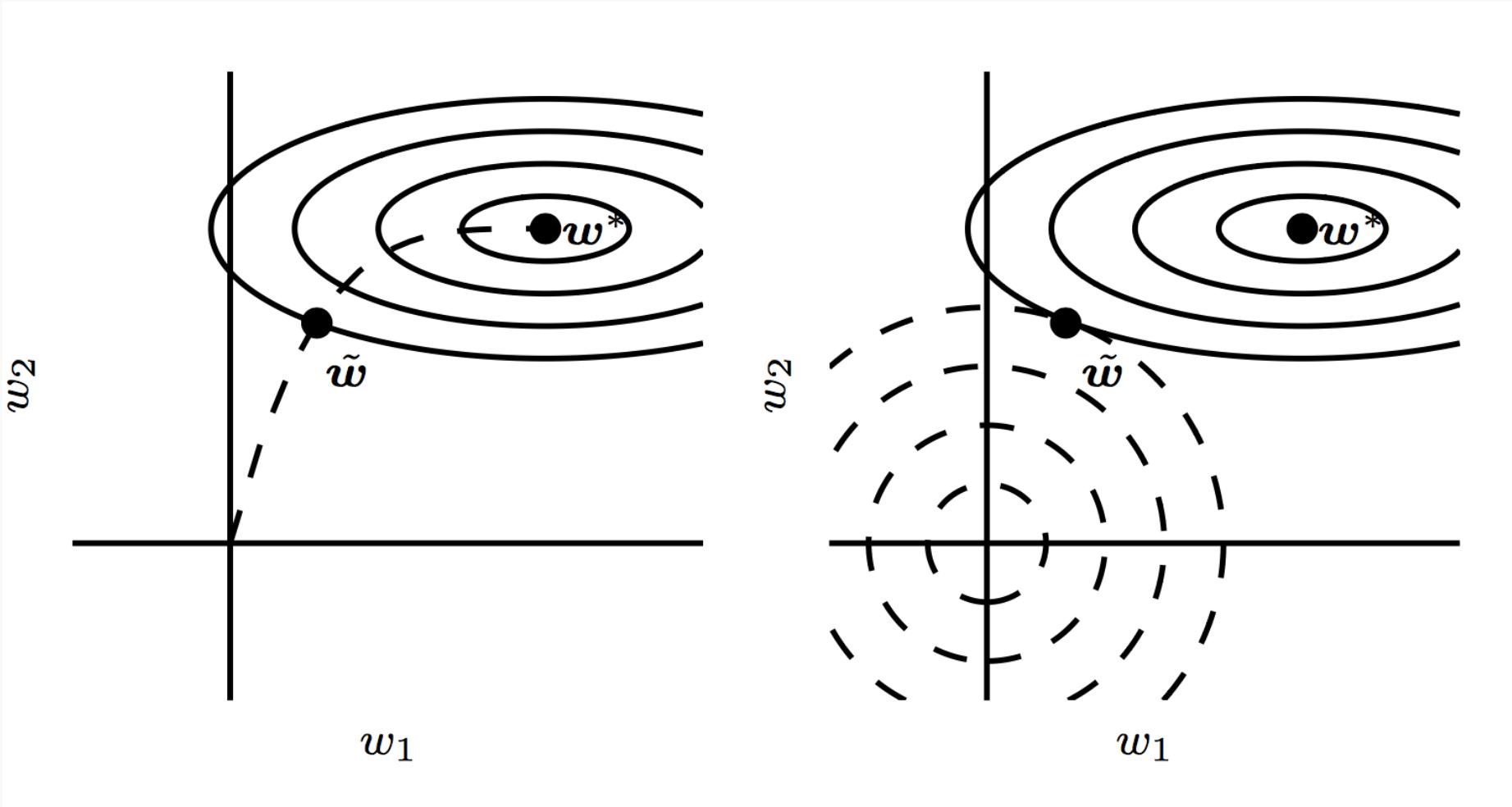
Early Stopping

Early stopping: terminate while validation set performance is better



Training time can be
treated as a
hyperparameter

Early Stopping



Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

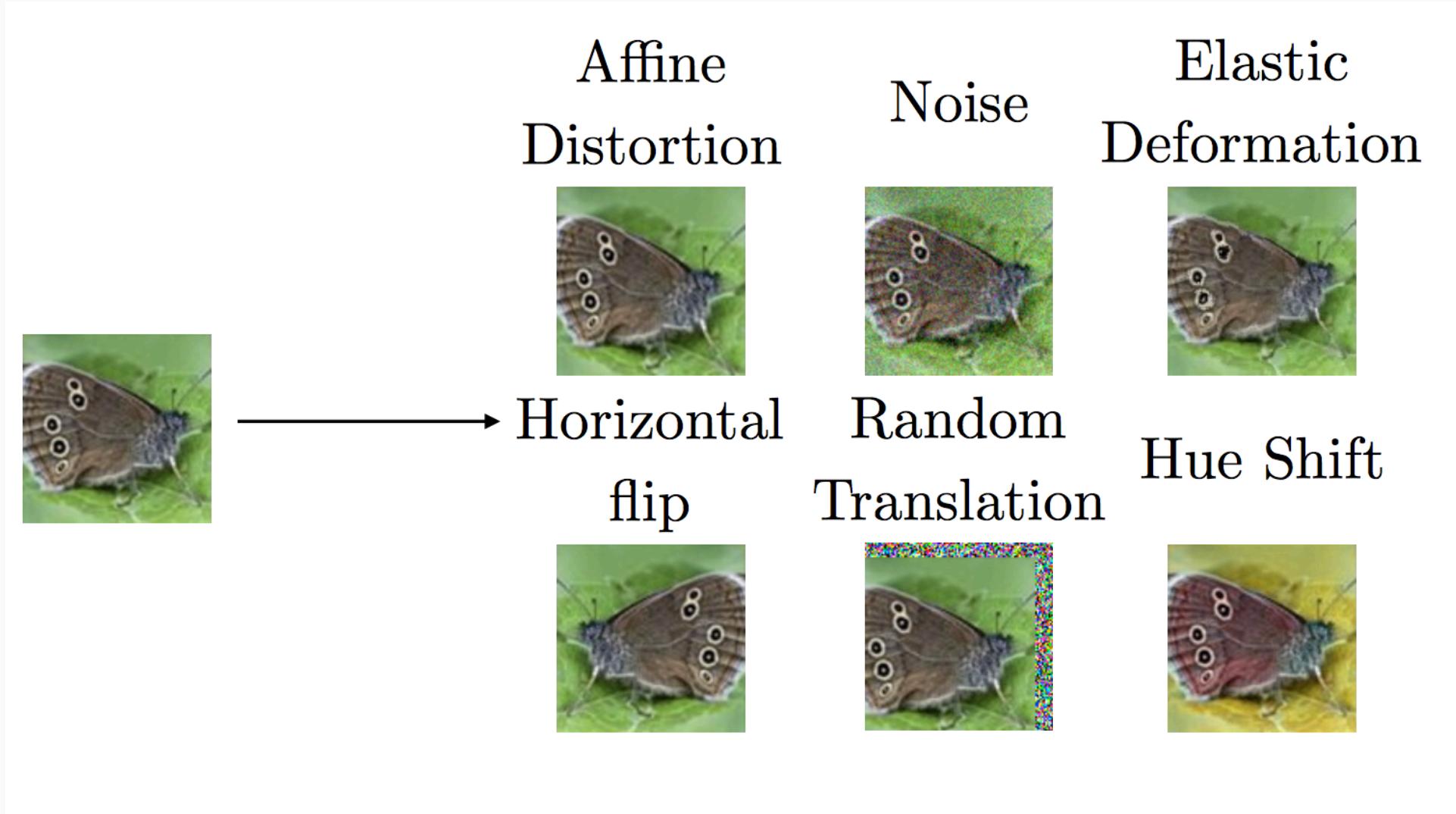
Regularization of NN

- Norm Penalties
- Early Stopping
- **Data Augmentation**
- Sparse Representation
- Dropout

Data Augmentation



Data Augmentation



Outline

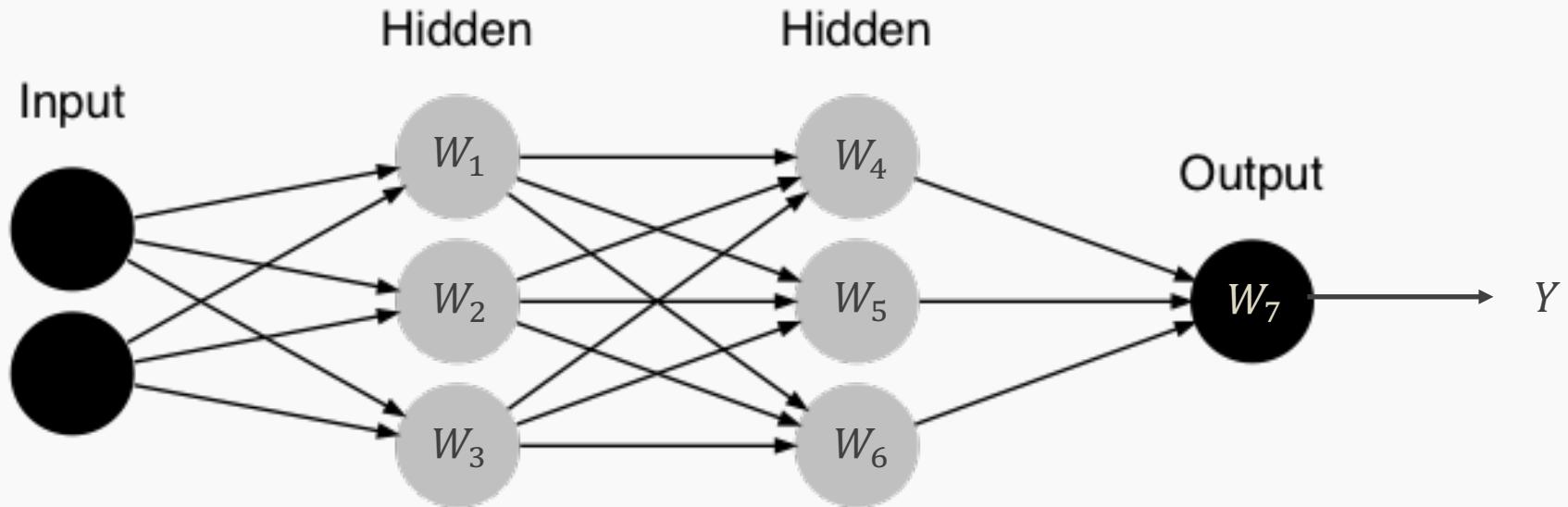
Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- **Sparse Representation**
- Dropout

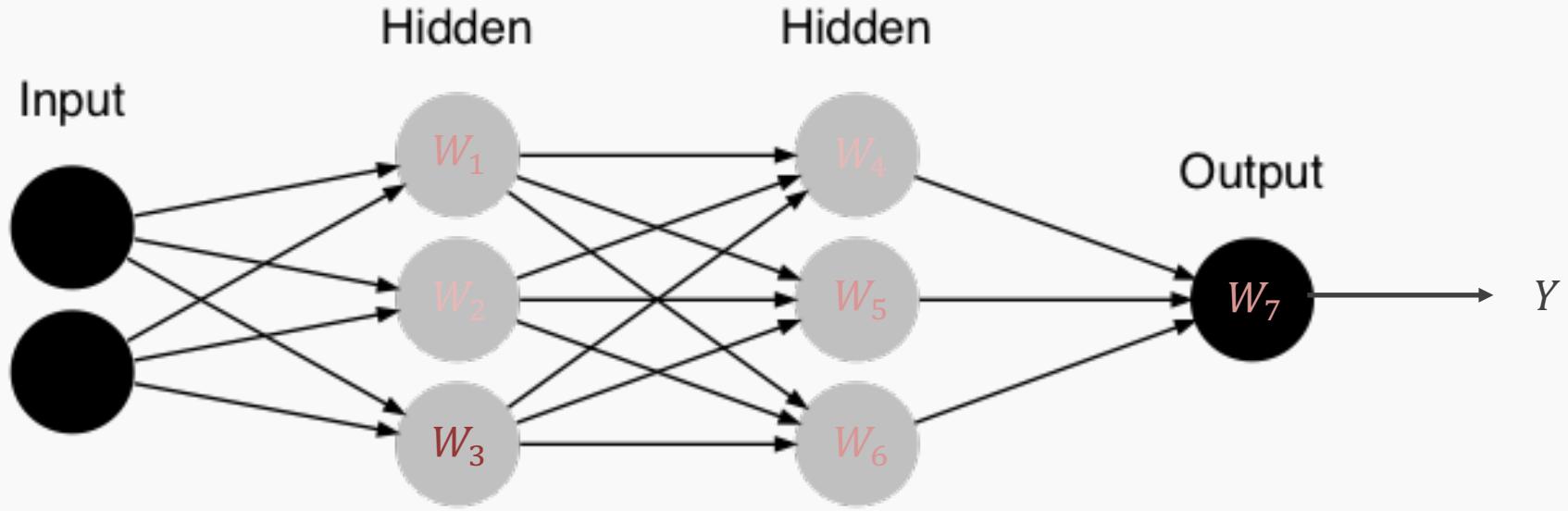
Sparse Representation



$$L(\theta; X, y)$$

$$[4.34] = [3.2 \quad 2.0 \quad 1.8] \underbrace{\begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix}}_{W_7}$$

Sparse Representation

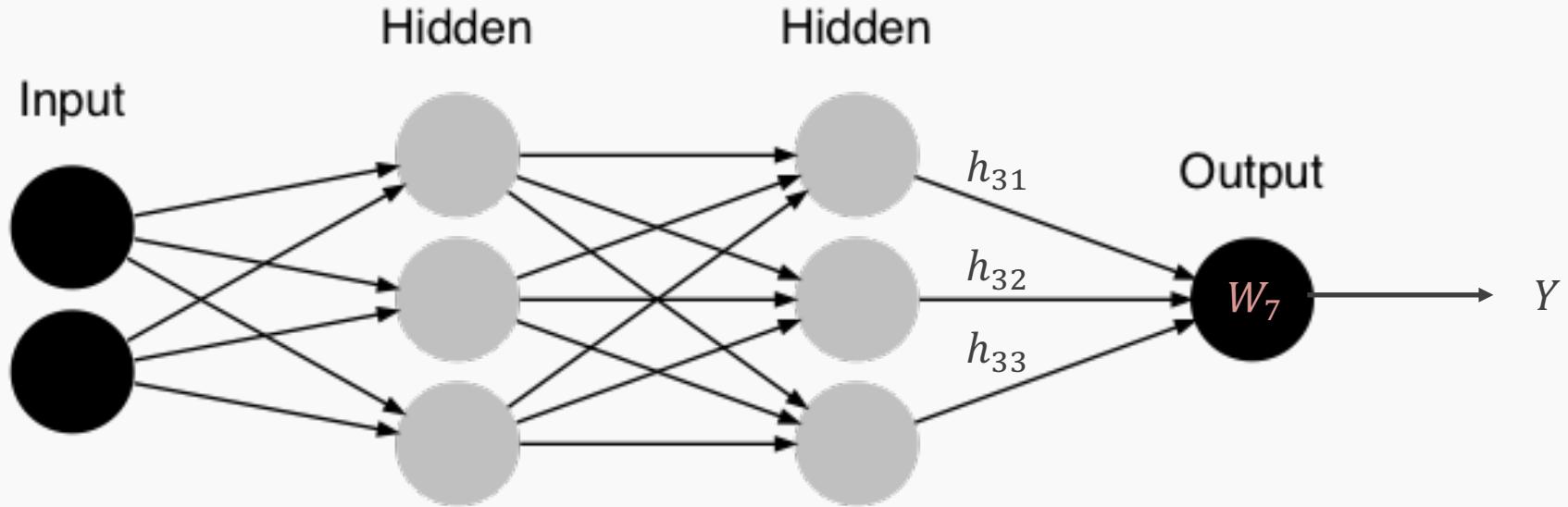


$$L_R(W; X, y) = L(\theta; X, y) + \alpha\Omega(W)$$

$$[0.69] = [0.5 \quad .2 \quad 0.1] \underbrace{\begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix}}_{W_7}$$

Weights in output layer

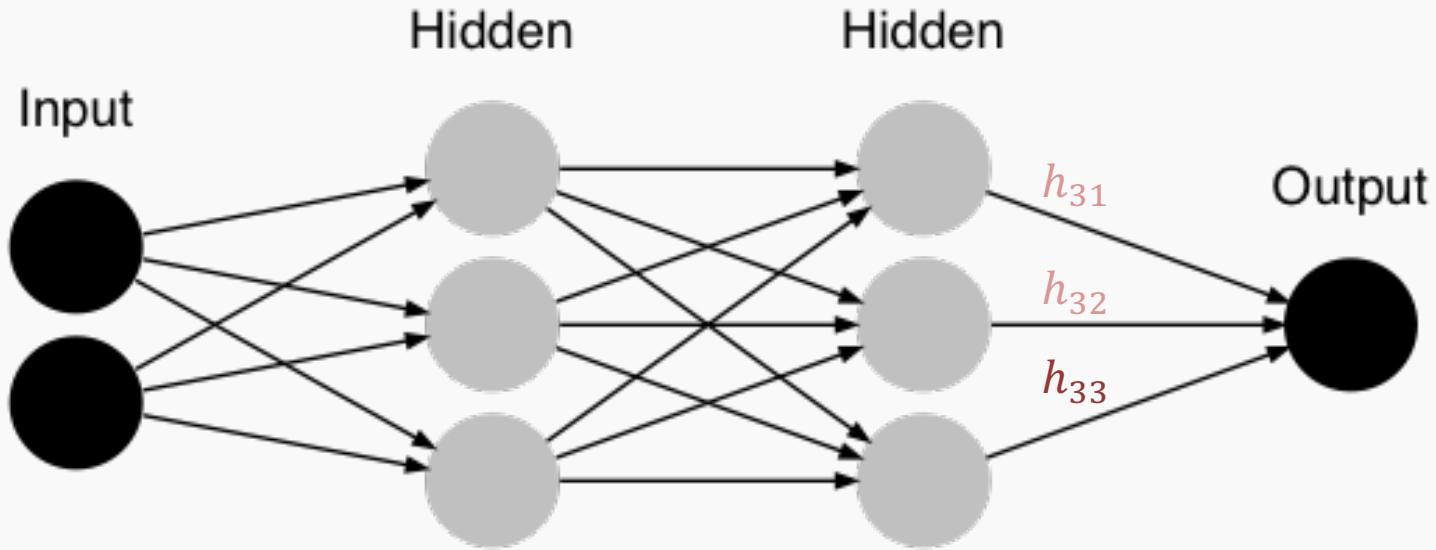
Sparse Representation



$$L(\theta; X, y)$$

$$[4.34] = [3.2 \quad 2 \quad 1] \begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix} \brace{h_{31}, h_{32}, h_{33}}$$

Sparse Representation



$$L_R(W; X, y) = L(\theta; X, y) + \alpha \Omega(h)$$

$$[1.3] = [3.2 \quad 2 \quad 1] \begin{bmatrix} 0 \\ -0.2 \\ .9 \end{bmatrix} \quad h_{31}, h_{32}, h_{33}$$



Output of hidden layer

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- **Dropout**

Noise Robustness

Random perturbation of network weights

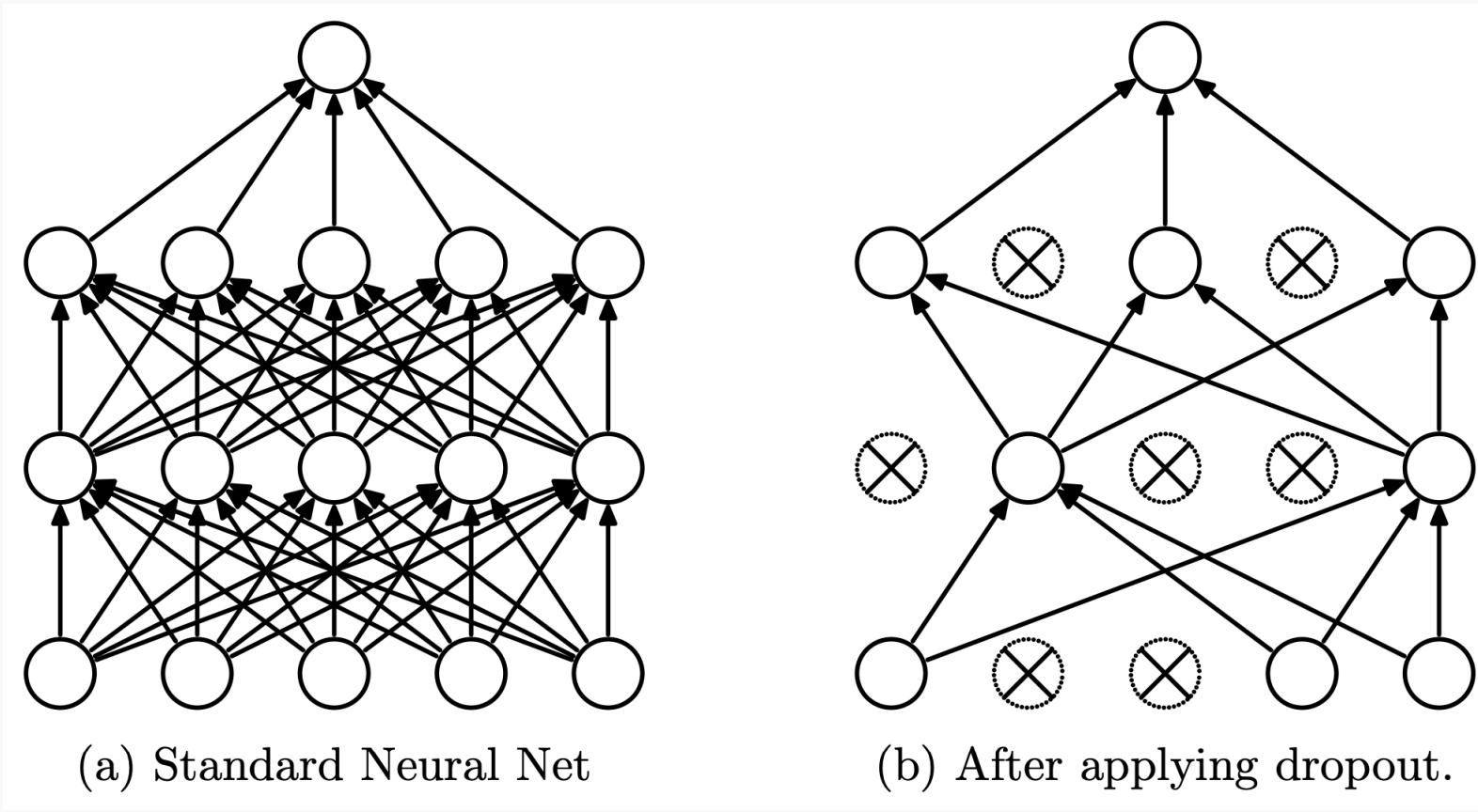
- Gaussian noise: Equivalent to minimizing loss with regularization term
- Encourages smooth function: small perturbation in weights leads to small changes in output

Injecting noise in output labels

- Better convergence: prevents pursuit of hard probabilities

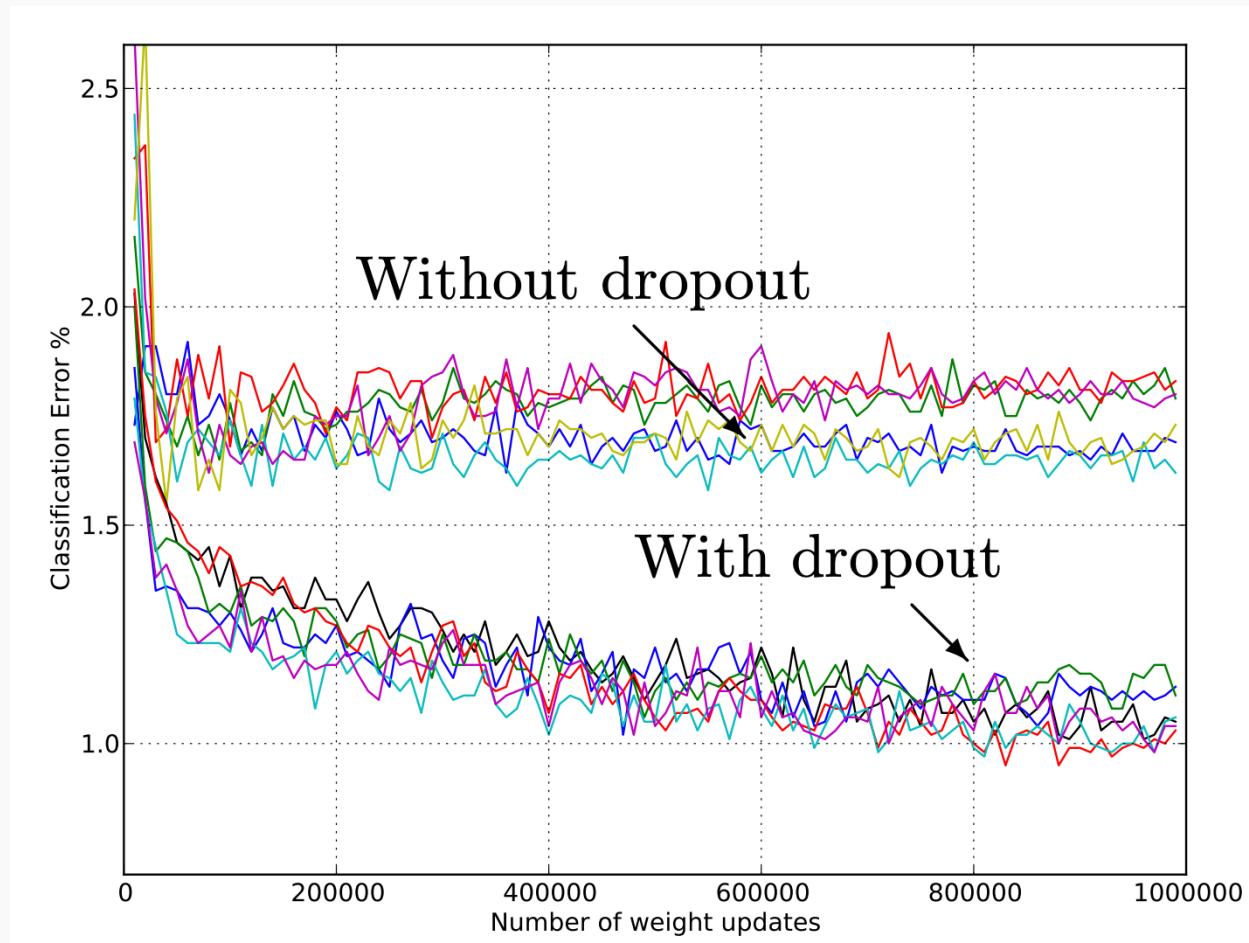
Dropout

- Randomly set some neurons and their connections to zero (i.e. “dropped”)
- Prevent overfitting by reducing co-adaptation of neurons
- Like training many random sub-networks



Dropout

- Widely used and highly effective
- Proposed as an alternative to ensembling, which is too expensive for neural nets



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Dropout: Stochastic GD

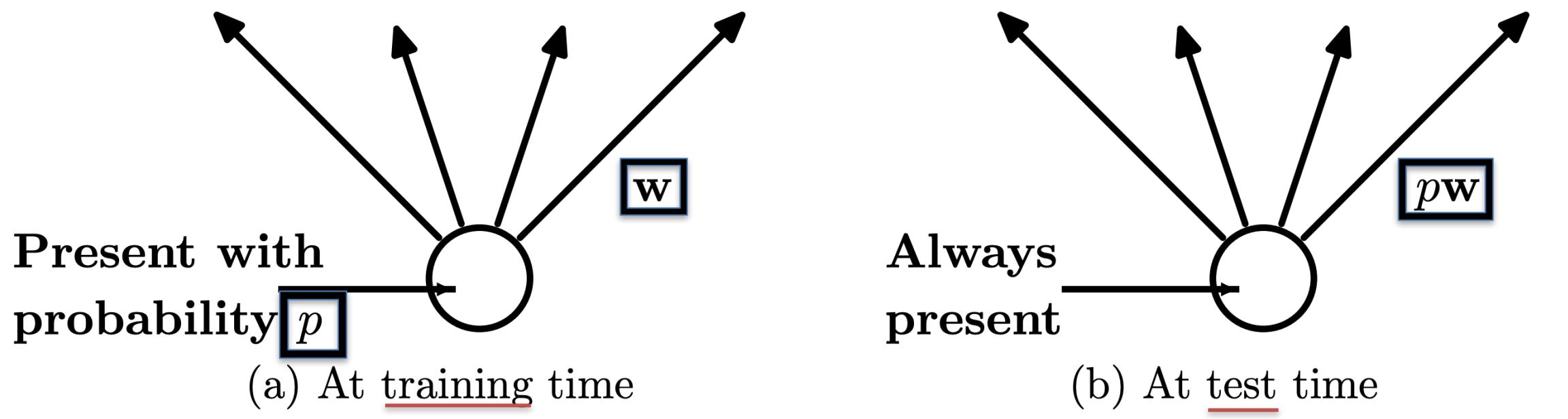
For each new example/mini-batch:

- Randomly sample a binary mask μ independently, where μ_i indicates if input/hidden node i is included
- Multiply output of node i with μ_i , and perform gradient update

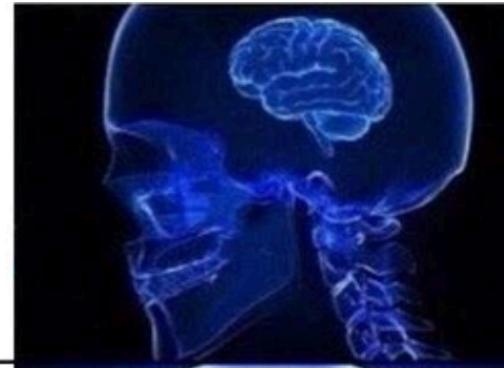
Typically, an input node is **included** with **prob=0.8**, hidden node with **prob=0.5**.

Dropout: Weight Scaling

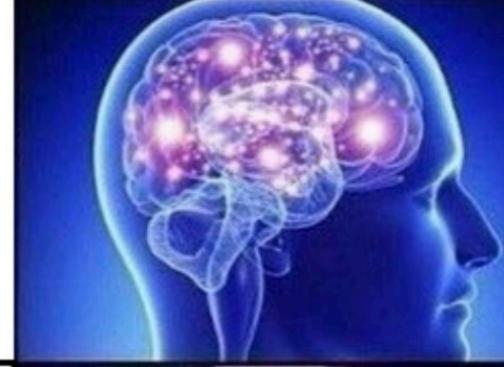
- We can think of dropout as training many of sub-networks
- At **test time**, we can “aggregate” over these sub-networks by **reducing connection weights in proportion to dropout probability, p**



**INCLUDING
DROPOUT LAYERS**



**SCHEDULING
THE
LEARNING RATE**



**EXPERIMENTING
WITH
ACTIVATION FUNCTIONS**



**OPTIMIZING
THE RANDOM SEED**



