

# Lecture 20: Back Propagation

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



# Outline

---

1. Back Propagation

2. Optimizers



# Outline

---

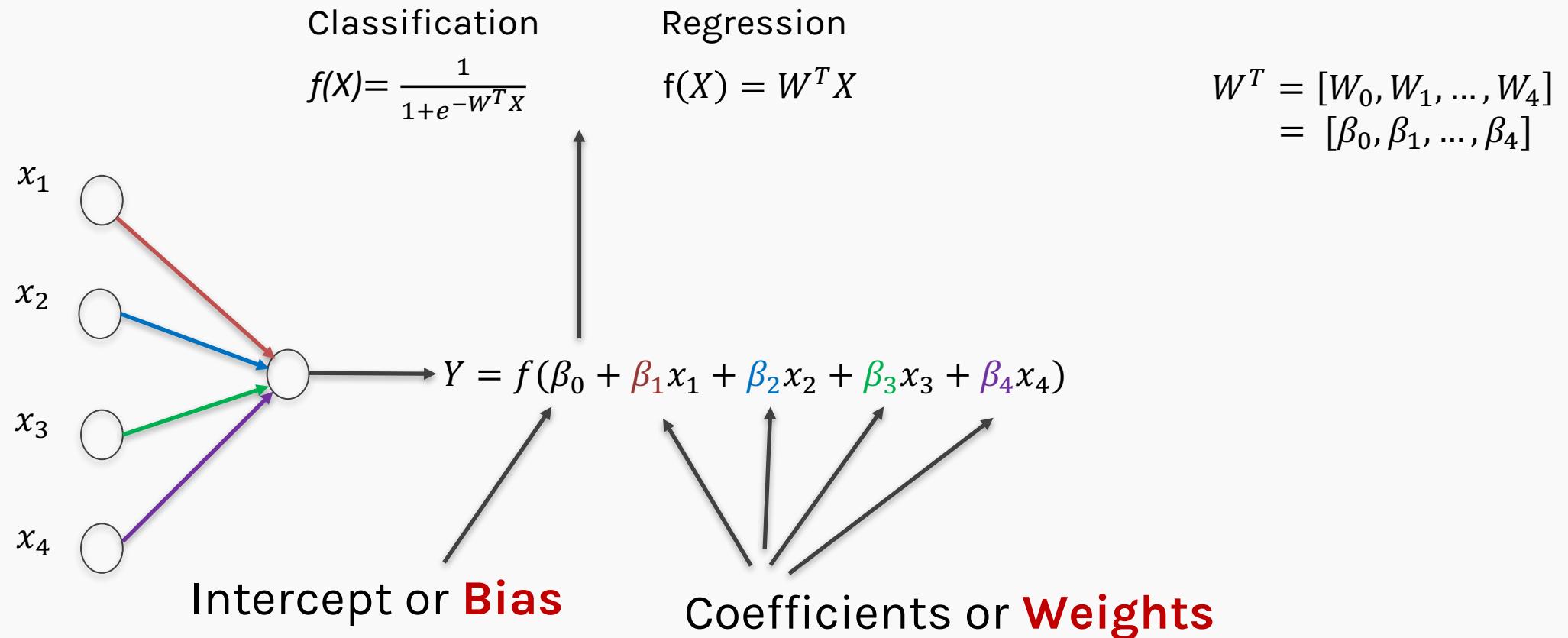
1. Back Propagation

2. Optimizers



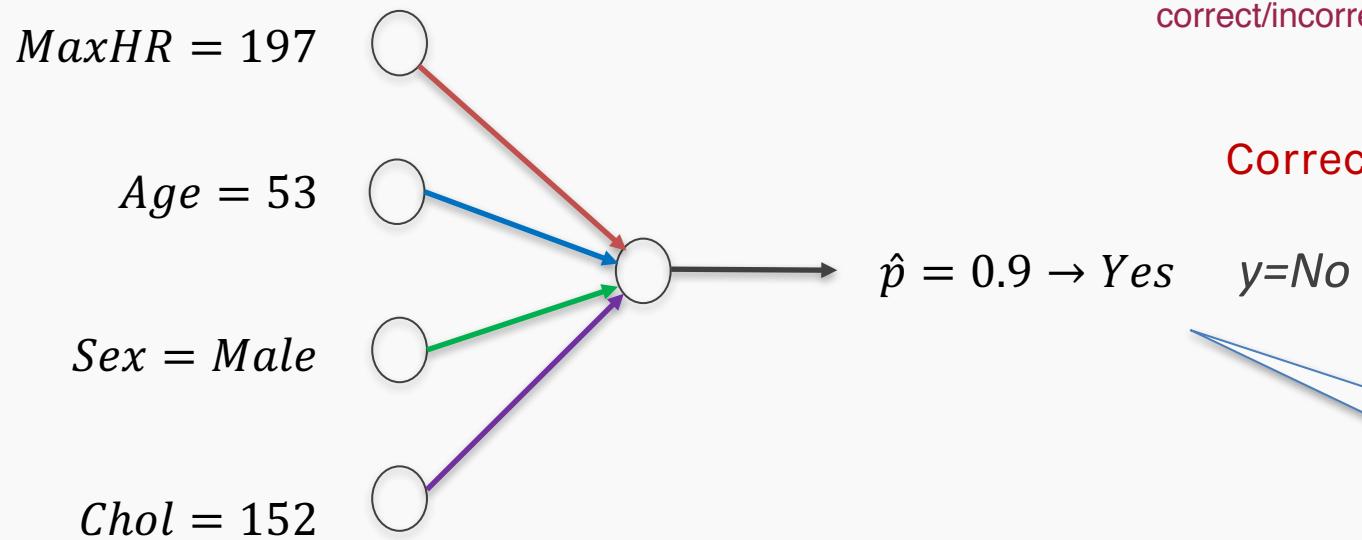
# Learning the coefficients

Start with Regression or Logistic Regression



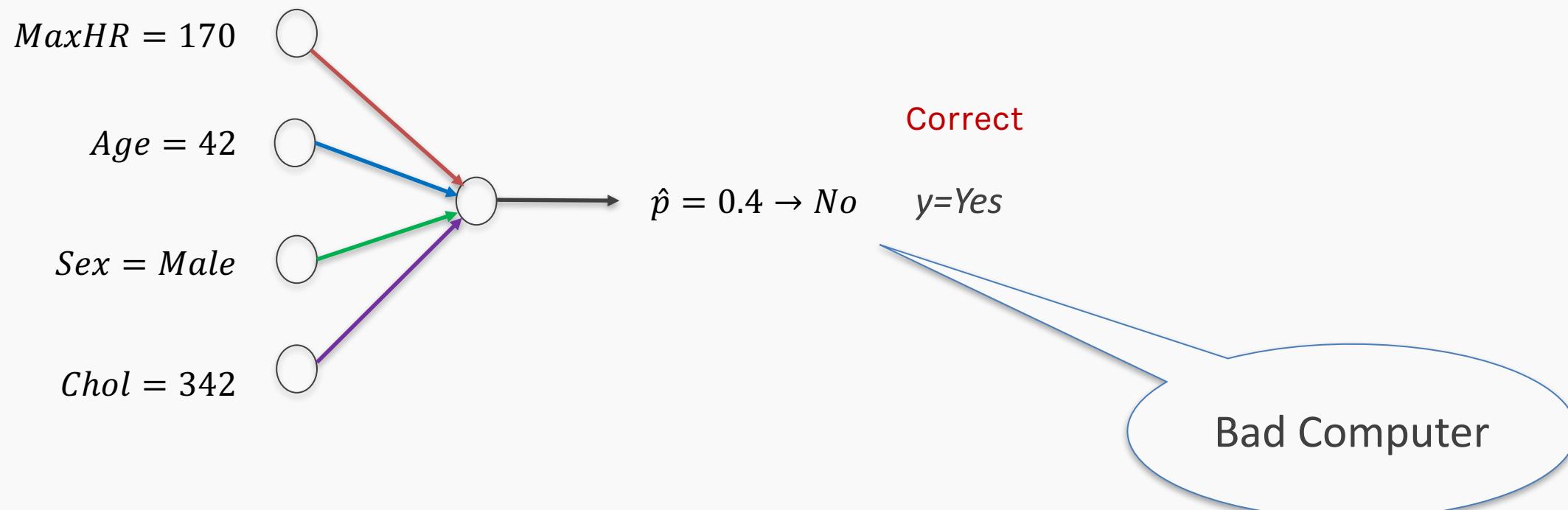
# But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly.  
For example, in our heart data, the model will be giving us the wrong answer.



# But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly. For example, in our heart data, the model will be giving us the wrong answer.



# But what is the idea?

---

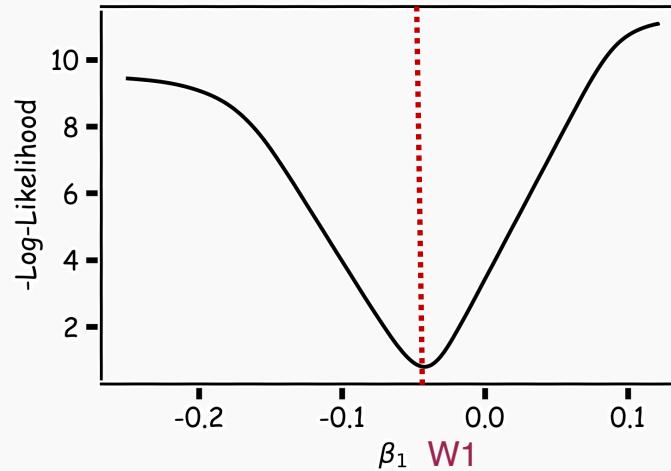
- **Loss Function:** Takes all of these results and averages them and tells us how bad or good the computer or those weights are.
- Telling the computer how **bad** or **good** is, does not help.
- You want to tell it how to change those weights so it gets better.

Loss function:  $\mathcal{L}(w_0, w_1, w_2, w_3, w_4)$

For now let's only consider one weight,  $\mathcal{L}(w_1)$

# Minimizing the Loss function

Ideally we want to know the value of  $w_1$  that gives the minimum  $\mathcal{L}(W)$

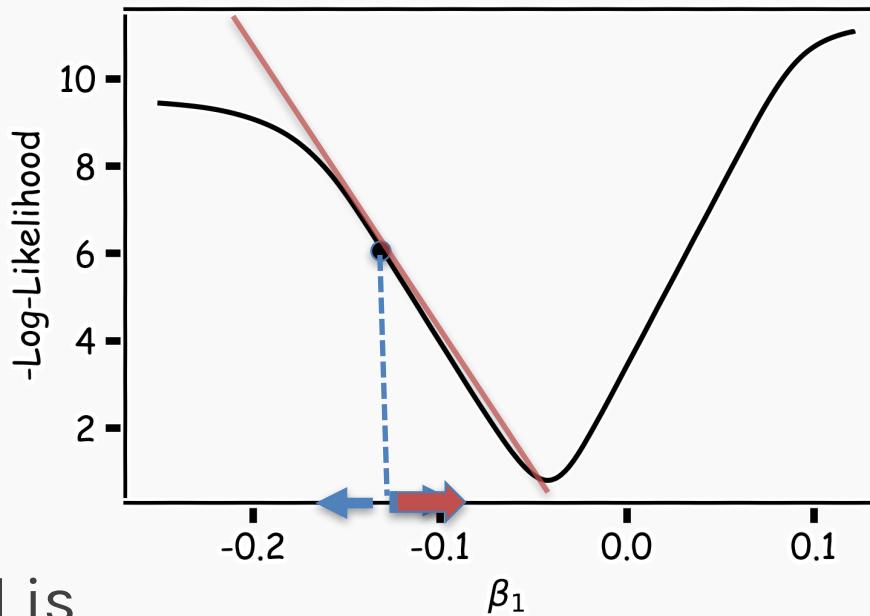


To find the optimal point of a function  $\mathcal{L}(W)$

$$\frac{d\mathcal{L}(W)}{dW} = 0 \quad \text{take the derivative of the loss fxn with respect to W}$$

And find the  $W$  that satisfies that equation. Sometimes there is no explicit solution for that.

# Estimate of the regression coefficients: gradient descent



let's start figuring things out with just one parameter

A more flexible method is

- Start from a random point
  1. Determine which direction to go to reduce the loss (left or right)
  2. Compute the slope of the function at this point and step to the right if slope is negative or step to the left if slope is positive
  3. Goto to #1

# Gradient Descent (cont.)

If the step is proportional to the slope then you avoid overshooting the minimum. How?



# Minimization of the Loss Function

**Question:** What is the mathematical function that describes the slope?

**Derivative**

**Question:** How do we generalize this to more than one predictor?

**Take the derivative with respect to each coefficient and do the same sequentially**

partial derivatives

**Question:** What do you think it is a good approach for telling the model how to change (what is the step size) to become better?

make steps proportional to the derivative - at minimum, step size is 0



# Let's play the Pavlos game

We know that we want to go in the opposite direction of the derivative and we know we want to be making a step proportionally to the derivative.

Making a step means:

$$w^{new} = w^{old} + step$$

Step size is  
proportion  
al to  
derivative

Opposite direction of the derivative means:

$$w^{new} = w^{old} - \lambda \frac{d\mathcal{L}}{dw}$$

Learning  
Rate

Change to more conventional notation:

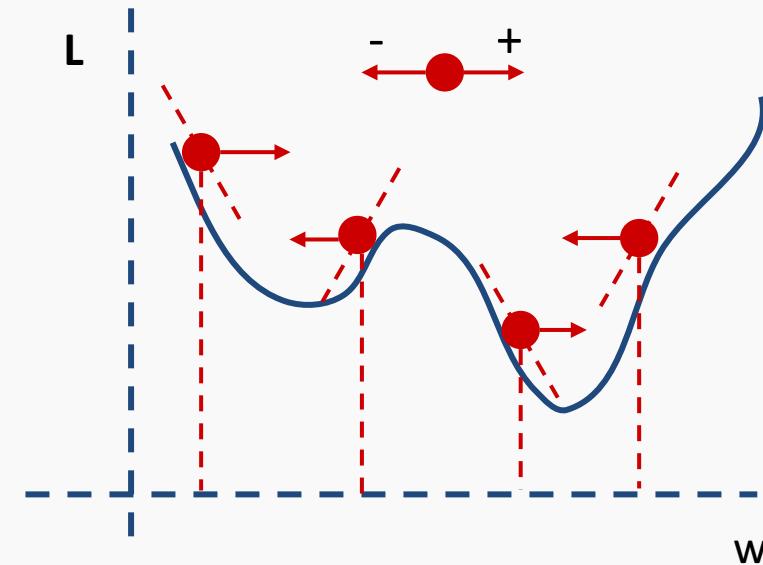
$$w^{(i+1)} = w^{(i)} - \lambda \frac{d\mathcal{L}}{dw}$$

what should lambda be? 1, 100, 0.01?  
lambda controls the scale so that W and  
 $d\mathcal{L}/dW$  are on the same scale

# Gradient Descent

- Algorithm for optimization of first order to finding a minimum of a function.
- It is an iterative method.
- $L$  is decreasing much faster in the direction of the negative derivative.
- The learning rate is controlled by the magnitude of  $\lambda$ .

$$w^{(i+1)} = w^{(i)} - \lambda \frac{d\mathcal{L}}{dw}$$



# Considerations

---

- We still need to calculate the derivatives.  
most often this is difficult!
- We need to know what is the learning rate or how to set it.
- Local vs global minima. how do you know you aren't stuck in the local minima?
- The full likelihood function includes summing up all individual ‘errors’. Unless you are a statistician, sometimes this includes hundreds of thousands of examples. expensive  
Loss fxn is sum of loss fxns over all training points - this can be a lot!!



# Considerations

---

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- Local vs global minima.
- The full likelihood function includes summing up all individual ‘errors’. Unless you are a statistician, sometimes this includes hundreds of thousands of examples.



# Calculate the Derivatives

---

## Example: Logistic Regression Derivatives

Can we do it?

Wolfram Alpha can do it for us!

We need a formalism to deal with these derivatives.

# Chain Rule

---

Chain rule for computing gradients:

- $y = g(x) \quad z = f(y) = f(g(x))$        $\mathbf{y} = g(\mathbf{x}) \quad z = f(\mathbf{y}) = f(g(\mathbf{x}))$   
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$
      
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$
  
if X, Y, Z are vectors - Jacobian (you sum over the points)
- For longer chains      
$$\frac{\partial z}{\partial x_i} = \sum_{j_1} \dots \sum_{j_m} \frac{\partial z}{\partial y_{j_1}} \dots \frac{\partial y_{j_m}}{\partial x_i}$$



# Logistic Regression derivatives

For logistic regression, the -ve log of the likelihood is:

$$\mathcal{L} = \sum_i \mathcal{L}_i = - \sum_i \log L_i = - \sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

rewrite p with respect to the weights

$$\mathcal{L}_i = -y_i \log \frac{1}{1 + e^{-W^T X}} - (1 - y_i) \log \left(1 - \frac{1}{1 + e^{-W^T X}}\right)$$

For a single point

To simplify the analysis let us split it into two parts,

$$\mathcal{L}_i = \mathcal{L}_i^A + \mathcal{L}_i^B$$

So the derivative with respect to  $W$  is:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_i \frac{\partial \mathcal{L}_i}{\partial W} = \sum_i \left( \frac{\partial \mathcal{L}_i^A}{\partial W} + \frac{\partial \mathcal{L}_i^B}{\partial W} \right)$$



deploy the chain rule for logistic regression

$$\mathcal{L}_i^A = -y_i \log \frac{1}{1 + e^{-W^T X}}$$

with respect to original parameters



Variables	Partial derivatives	Partial derivatives
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	$\frac{\partial \xi_1}{\partial W} = -X$
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{-W^T X}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{(1 + e^{-W^T X})^2}$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\frac{\partial \xi_5}{\partial \xi_4} = 1 + e^{-W^T X}$
$\mathcal{L}_i^A = -y \xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$		$\frac{\partial \mathcal{L}_i^A}{\partial W} = -y X e^{-W^T X} \frac{1}{(1 + e^{-W^T X})}$

multiply everything at the end



do the same thing for the other part of the cross entropy loss

$$\mathcal{L}_i^B = -(1 - y_i) \log[1 - \frac{1}{1 + e^{-W^T X}}]$$

Variables	derivatives	Partial derivatives wrt to X,W
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	$\frac{\partial \xi_1}{\partial W} = -X$
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{-W^T X}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$\frac{\partial \xi_3}{\partial 2} = 1$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{(1 + e^{-W^T X})^2}$
$\xi_5 = 1 - \xi_4 = 1 - \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = -1$	$\frac{\partial \xi_5}{\partial \xi_4} = -1$
$\xi_6 = \log \xi_5 = \log(1 - p) = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_6}{\partial \xi_5} = \frac{1}{\xi_5}$	$\frac{\partial \xi_6}{\partial \xi_5} = \frac{1 + e^{-W^T X}}{e^{-W^T X}}$
$\mathcal{L}_i^B = (1 - y)\xi_6$	$\frac{\partial \mathcal{L}}{\partial \xi_6} = 1 - y$	$\frac{\partial \mathcal{L}}{\partial \xi_6} = 1 - y$
$\frac{\partial \mathcal{L}_i^B}{\partial W} = \frac{\partial \mathcal{L}_i^B}{\partial \xi_6} \frac{\partial \xi_6}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$		$\frac{\partial \mathcal{L}_i^B}{\partial W} = (1 - y)X \frac{1}{(1 + e^{-W^T X})}$



# Considerations

---

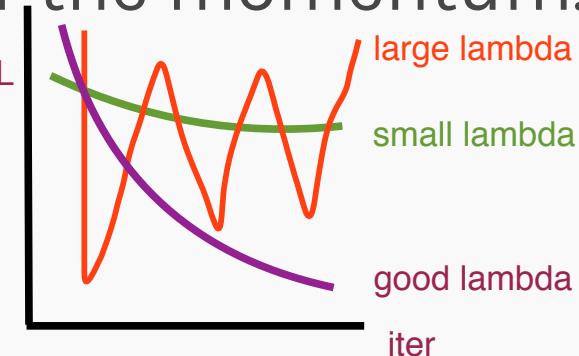
- We still need to calculate the derivatives.
- **We need to know what is the learning rate or how to set it.**
- Local vs global minima.
- The full likelihood function includes summing up all individual ‘errors’. Unless you are a statistician, sometimes this includes hundreds of thousands of examples.



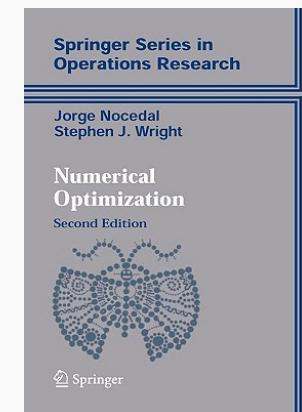
# Learning Rate

## Trial and Error.

There are many alternative methods which address how to set or adjust the learning rate, using the derivative or second derivatives and or the momentum. To be discussed in the next lectures on NN.



- \* J. Nocedal y S. Wright, “Numerical optimization”, Springer, 1999 
- \* *TLDR*: J. Bullinaria, “Learning with Momentum, Conjugate Gradient Learning”, 2015 

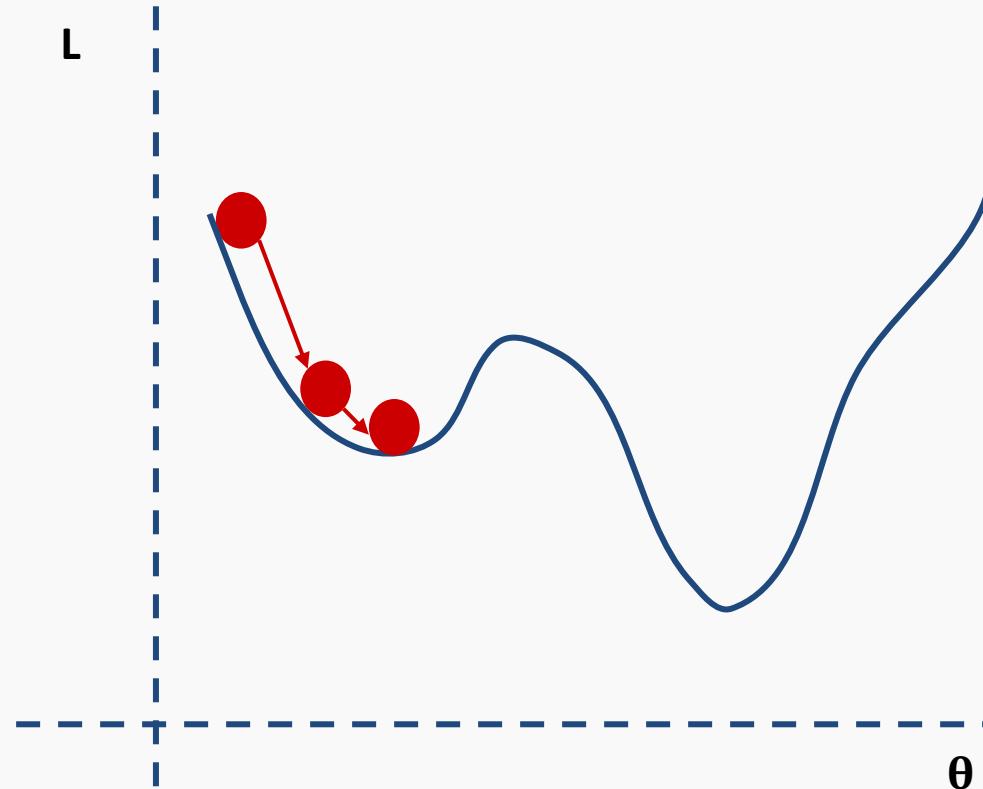


# Considerations

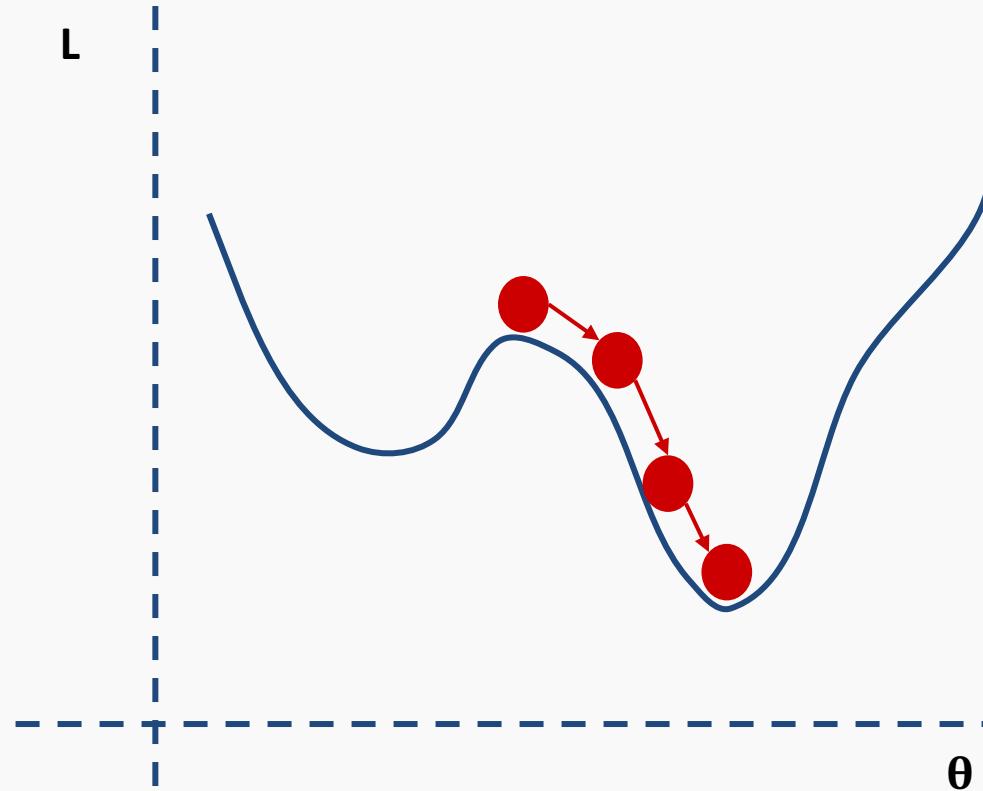
---

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- **Local vs global minima.**
- The full likelihood function includes summing up all individual ‘errors’. Unless you are a statistician, sometimes this includes hundreds of thousands of examples.

# Local vs Global Minima



# Local vs Global Minima



# Local vs Global Minima

---

No guarantee that we get the global minimum.

**Question:** What would be a good strategy?

random restarts and their derivatives - start at different places each time to try and find global minimum

no matter what you do, no guarantee you find global minimum (unless you do it for infinite time)...sometimes global minimum is hard to find

you can change learning rate to try and ‘jump’ out but still no guarantee

just ignore the bad feeling of maybe being stuck in local minima



# Considerations

---

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- Local vs global minima.
- **The full likelihood function includes summing up all individual ‘errors’. Unless you are a statistician, sometimes this includes hundreds of thousands of examples.**

# Batch and Stochastic Gradient Descent

$$\mathcal{L} = - \sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

full loss fxn is the sum of individual loss fxns for each training example

Instead of using all the examples for every step, use a subset of them (batch).

For each iteration  $k$ , use the following loss function to derive the derivatives:

if randomly select enough points, it should approx the full loss fxn decently (keep in mind that not exactly same as full one)

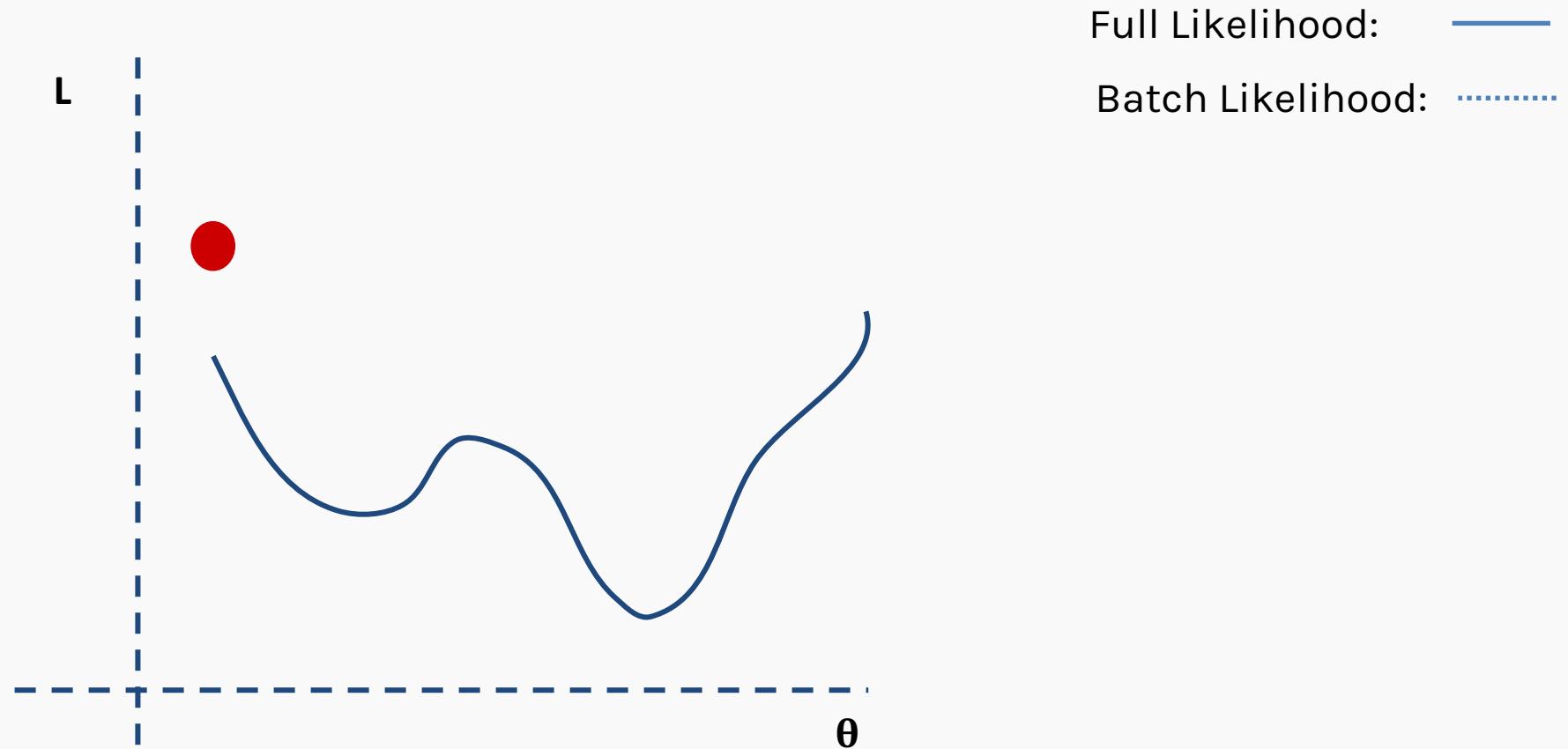
$$\mathcal{L}^k = - \sum_{i \in b^k} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

these subsampled L have error (deviation from full L), but errors should cancel out with enough samples

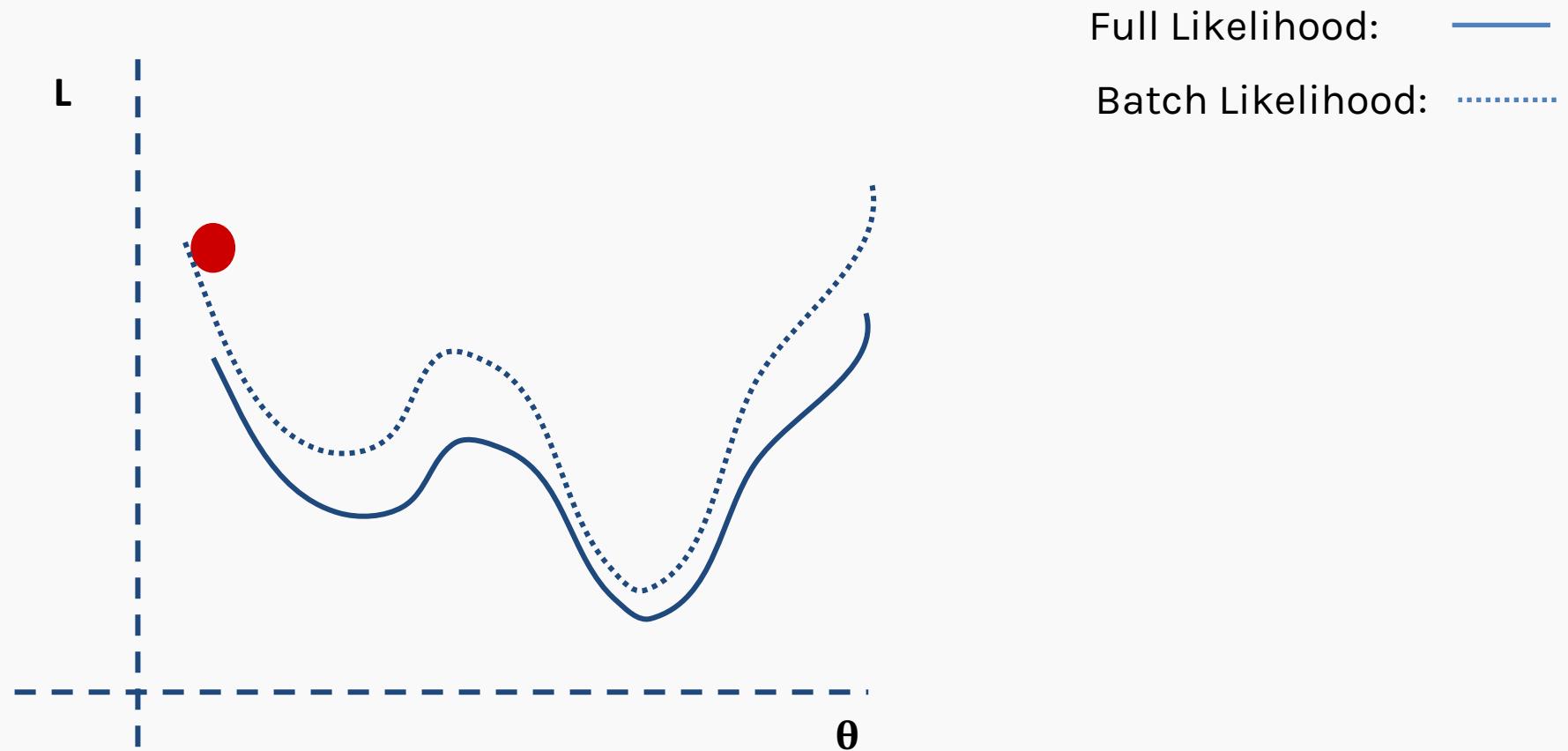
which is an **approximation** to the full loss function.



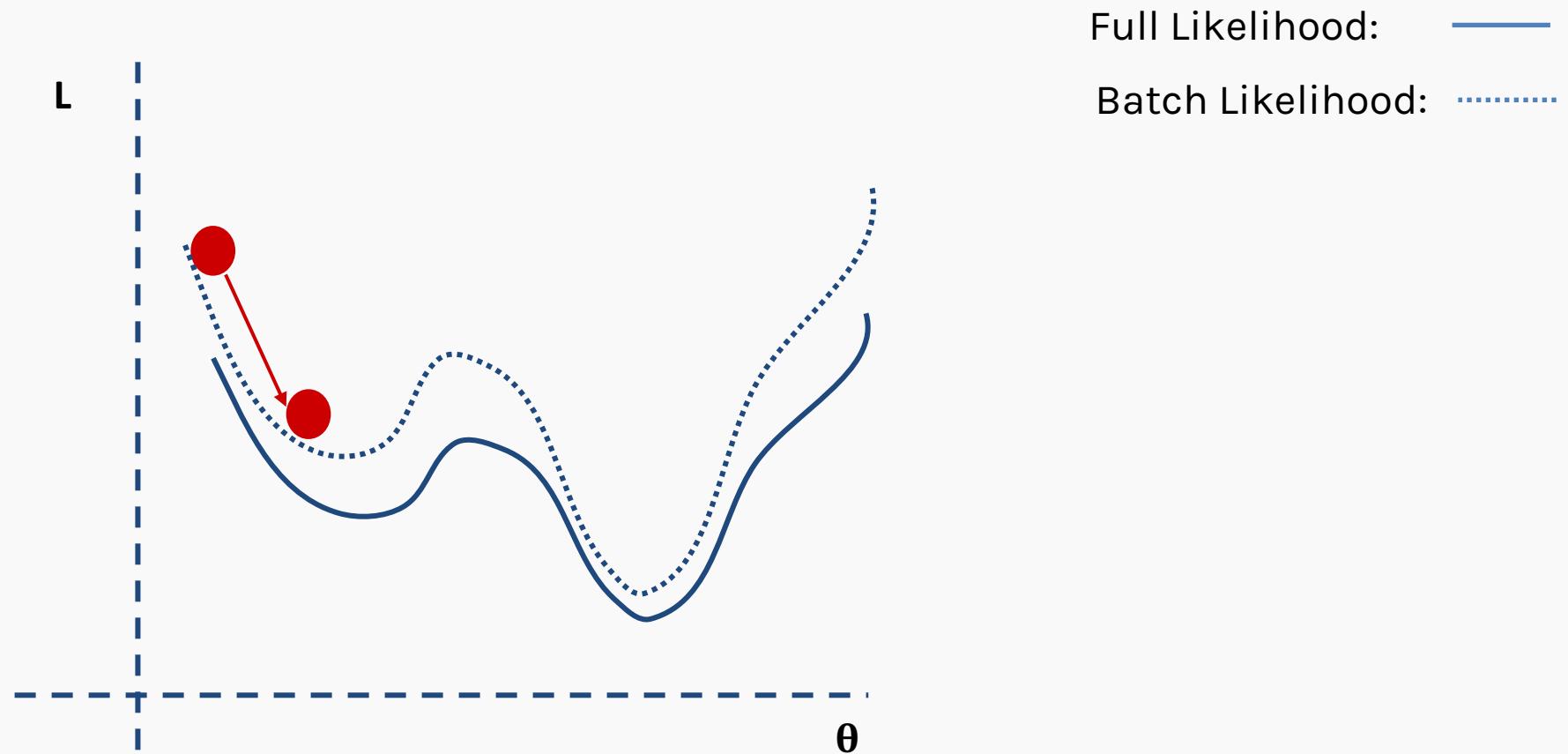
# Batch and Stochastic Gradient Descent



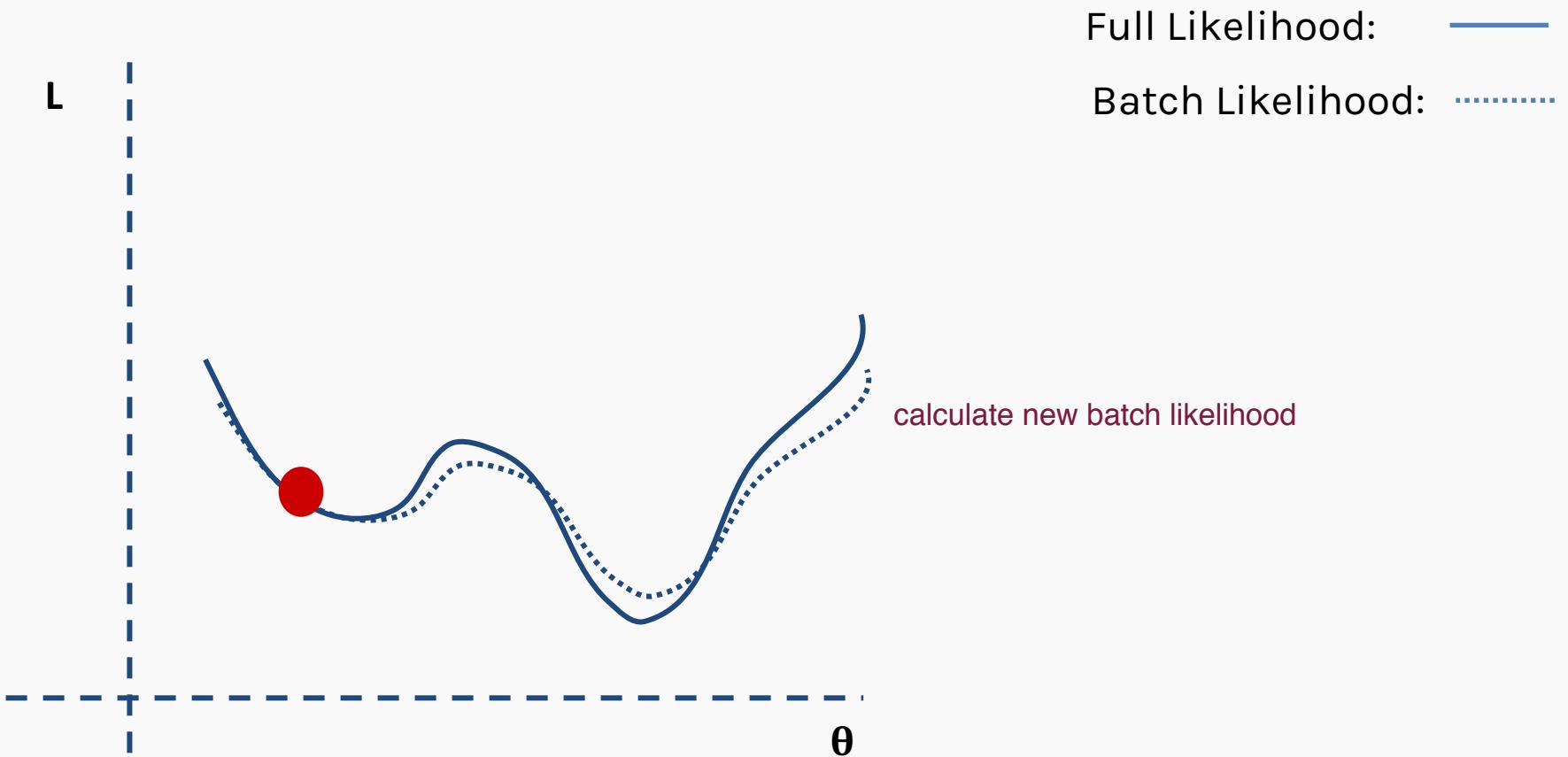
# Batch and Stochastic Gradient Descent



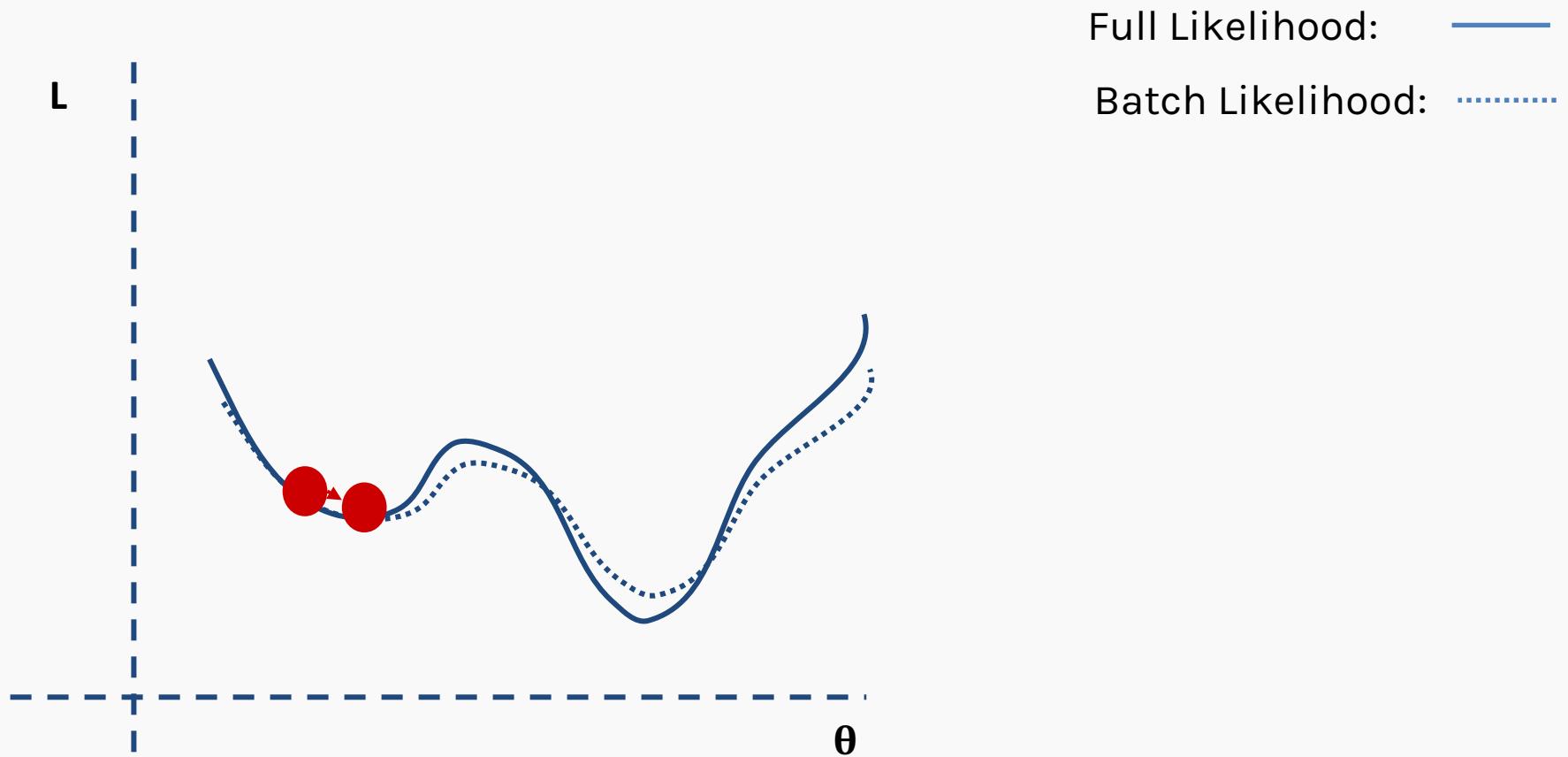
# Batch and Stochastic Gradient Descent



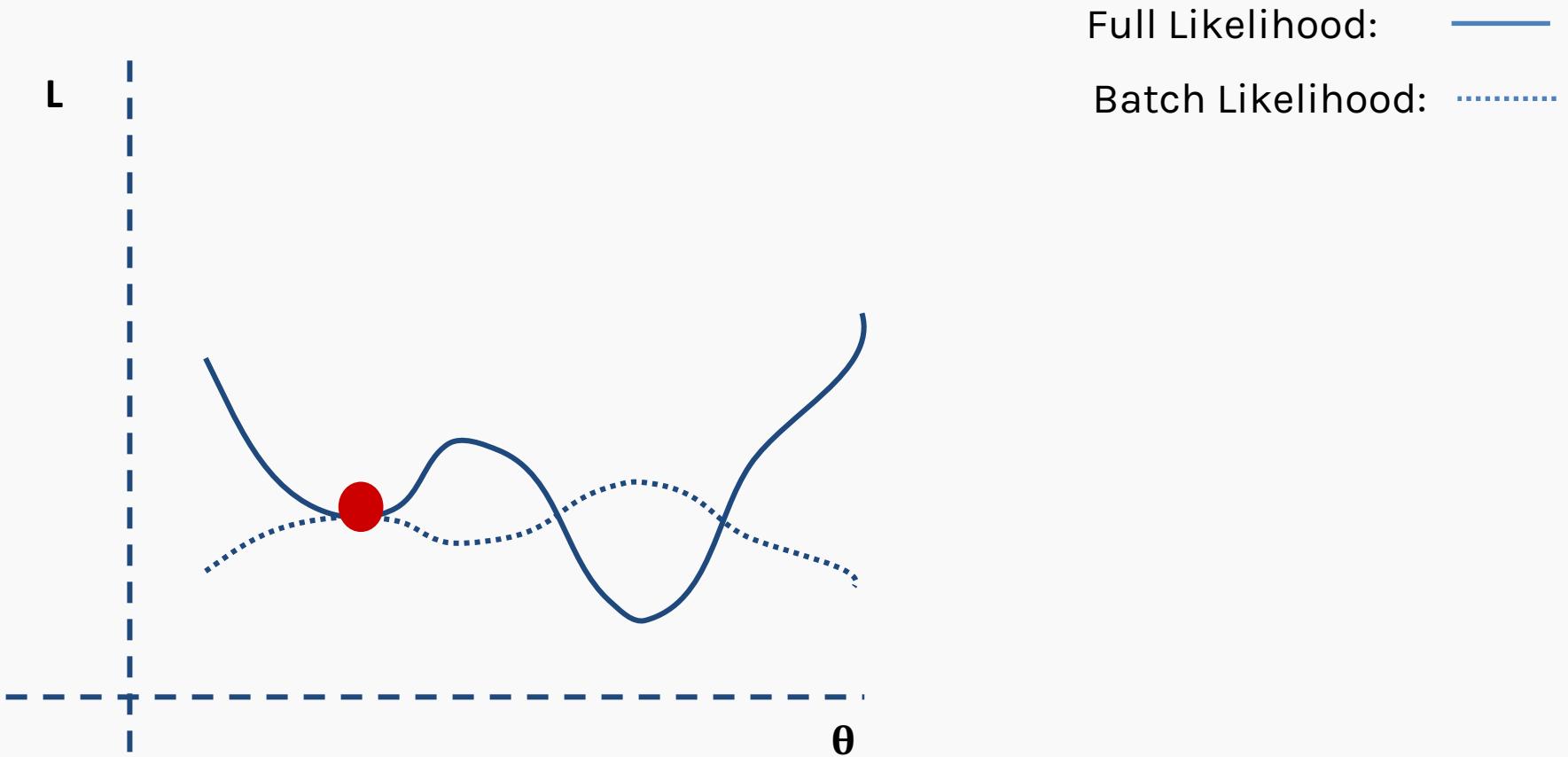
# Batch and Stochastic Gradient Descent



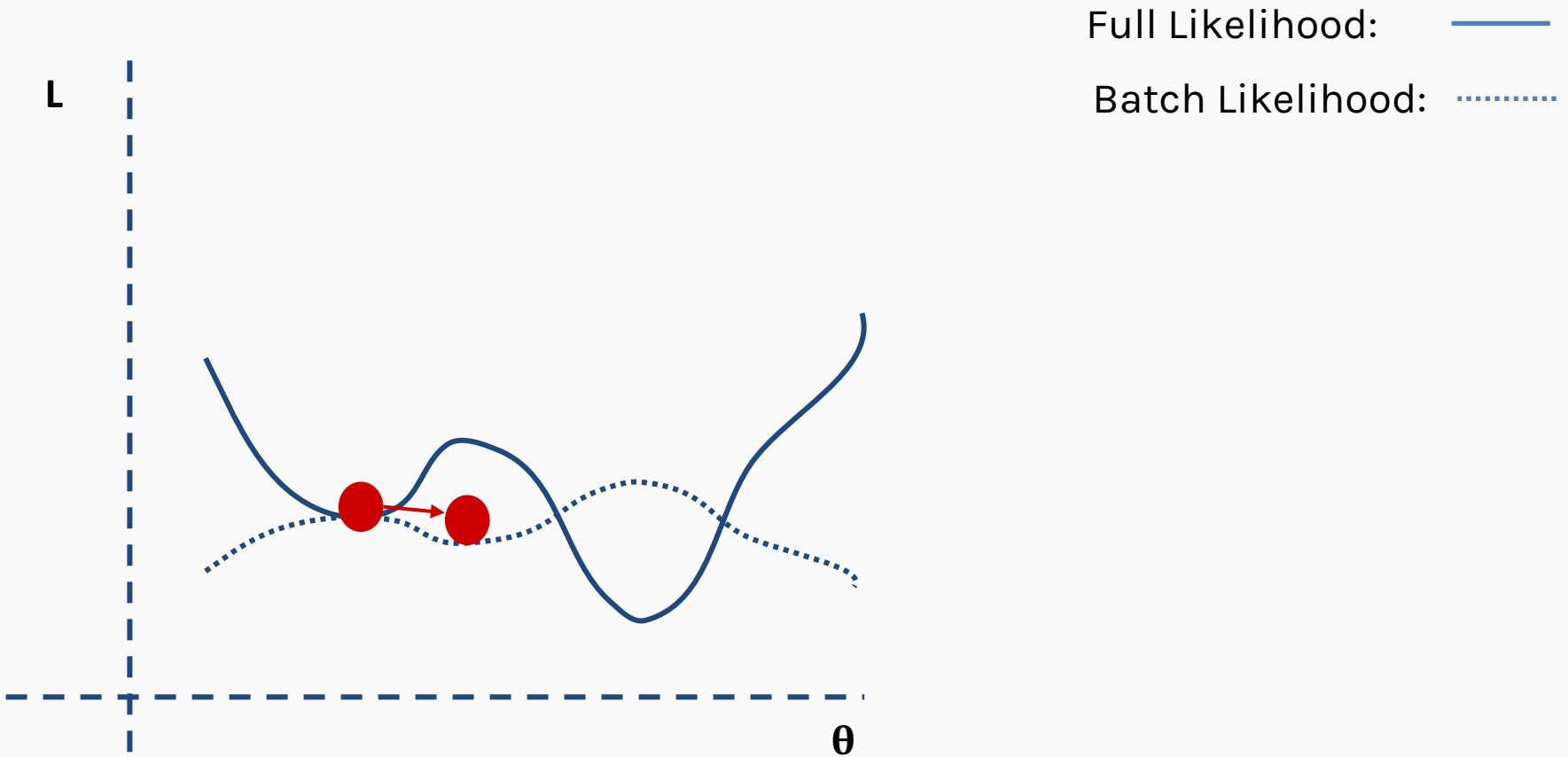
# Batch and Stochastic Gradient Descent



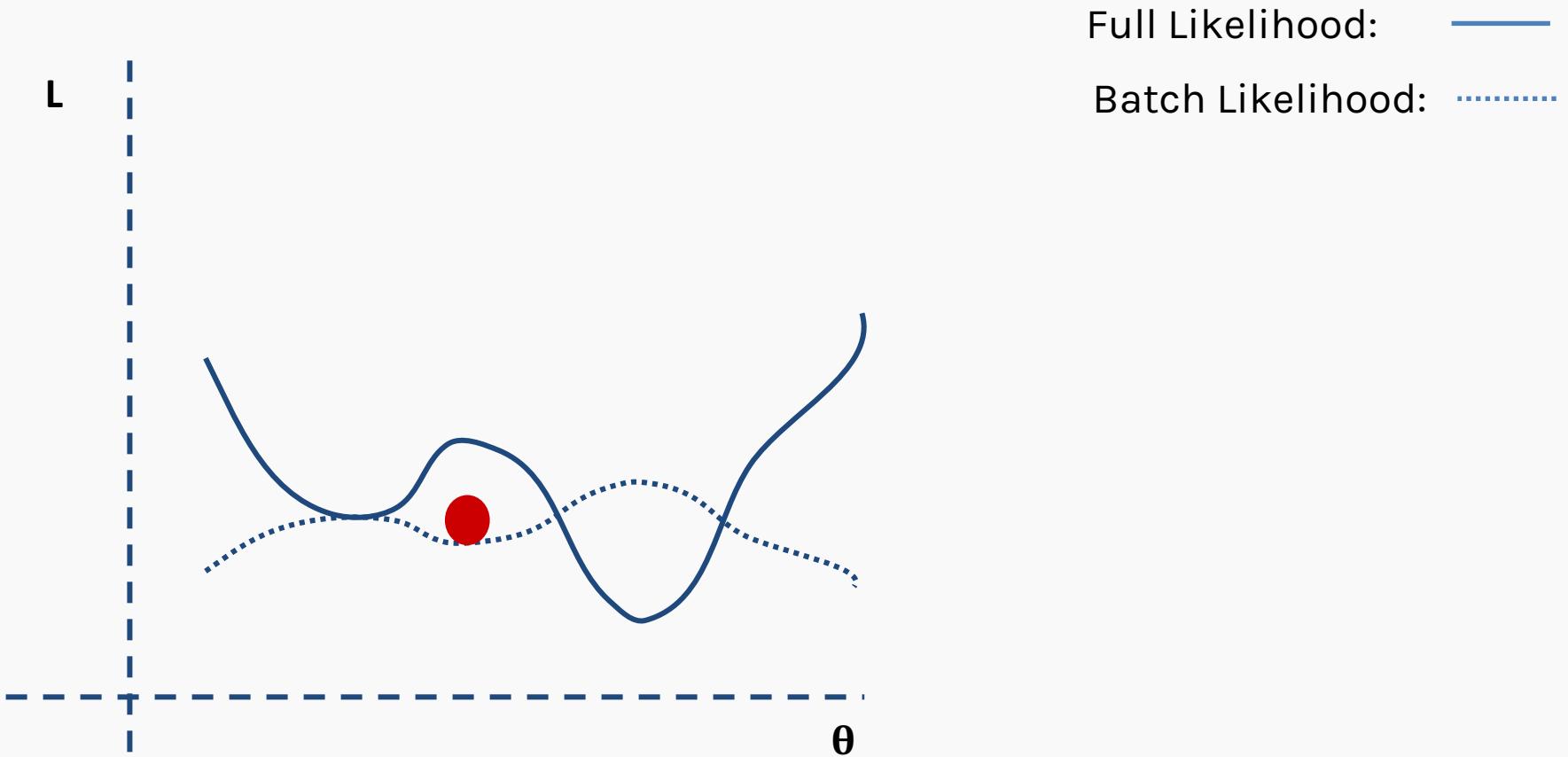
# Batch and Stochastic Gradient Descent



# Batch and Stochastic Gradient Descent



# Batch and Stochastic Gradient Descent



# Batch and Stochastic Gradient Descent

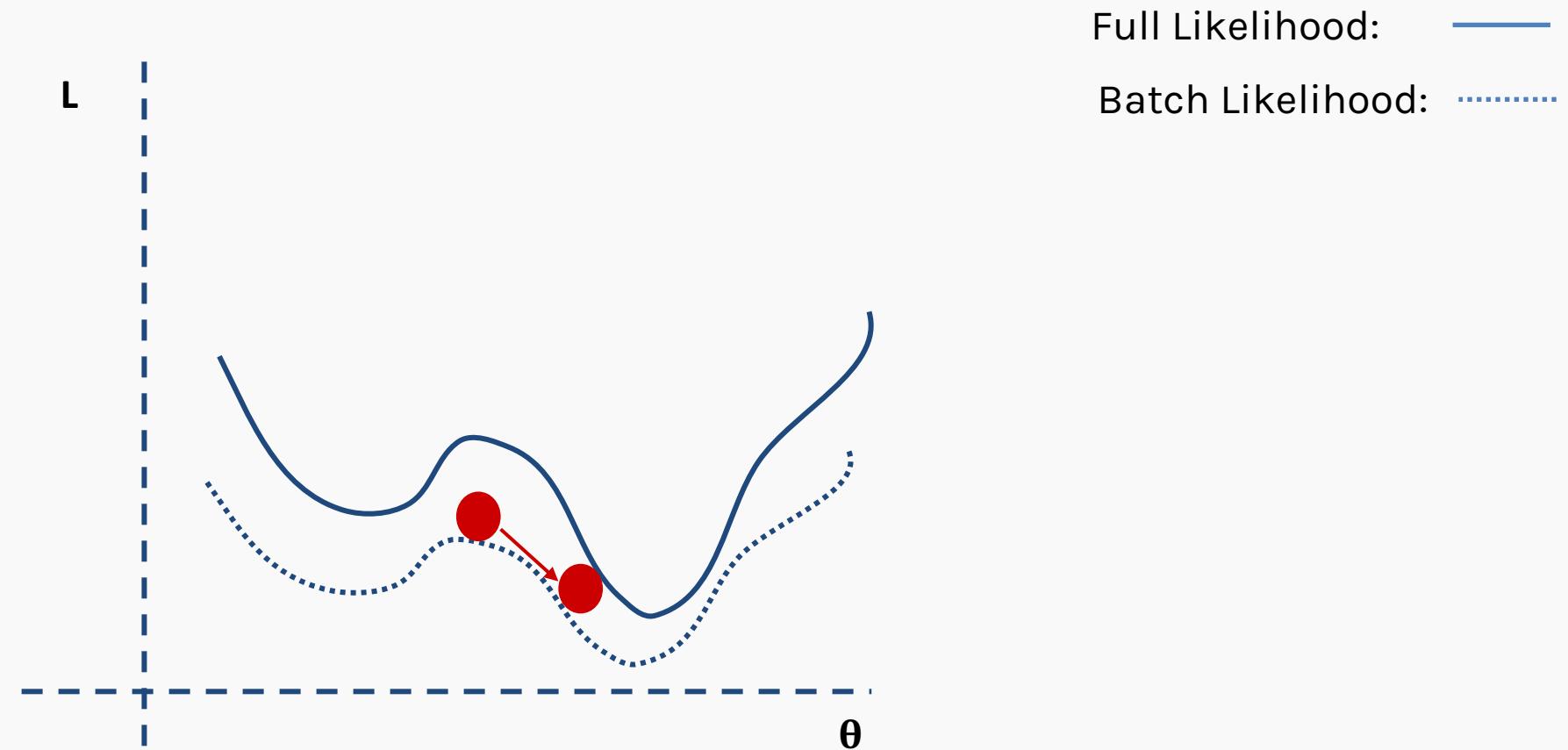
you got to the global minimum! by doing small batches, you reduce complexity, but also a chance to get out of the local minimum

This is stochastic gradient descent

Choice of batch size

small: too much ‘bounce’  
large: too little ‘bounce’ and get stuck in minima

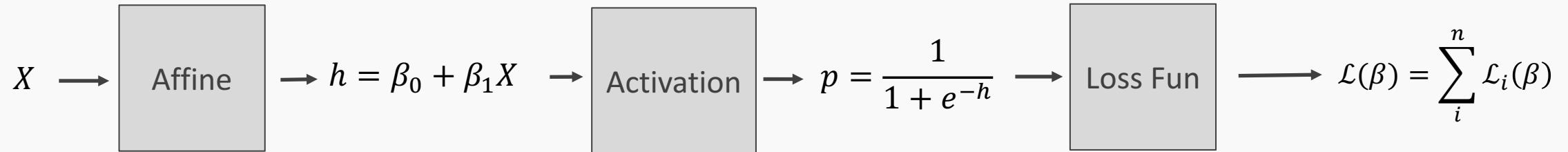
start with small batch size and the increase as you get closer to global (maybe) minimum



# Backpropagation



# Backpropagation: Logistic Regression Revisited



$$\frac{\partial \mathcal{L}}{\partial p} \frac{\partial p}{\partial h} \frac{\partial h}{\partial \beta} \quad \leftarrow \quad \frac{\partial \mathcal{L}}{\partial p} \frac{\partial p}{\partial h} \quad \leftarrow \quad \frac{\partial \mathcal{L}}{\partial p}$$

$$\frac{\partial h}{\partial \beta_1} = X, \frac{d\mathcal{L}}{d\beta_0} = 1$$

$$\frac{\partial p}{\partial h} = \sigma(h)(1 - \sigma(h))$$

$$\frac{\partial \mathcal{L}}{\partial p} = -y \frac{1}{p} - (1 - y) \frac{1}{1 - p}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = -X\sigma(h)(1 - \sigma(h))\left[y \frac{1}{p} + (1 - y) \frac{1}{1 - p}\right]$$

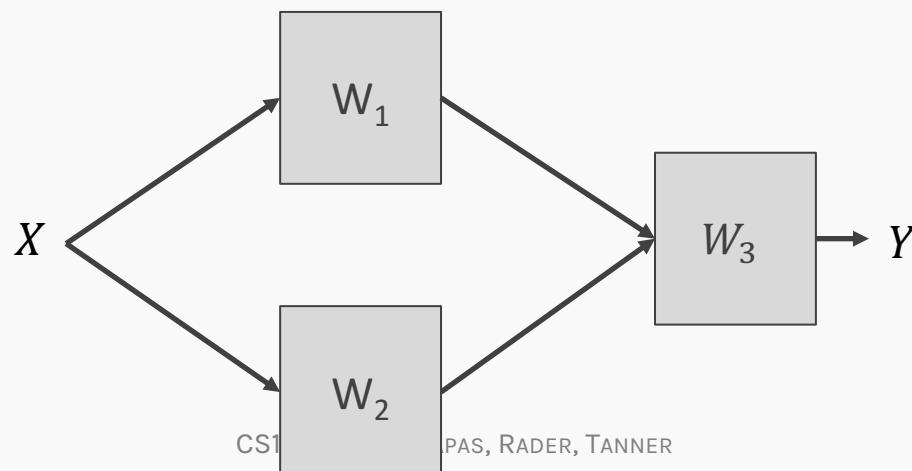
$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\sigma(h)(1 - \sigma(h))\left[y \frac{1}{p} + (1 - y) \frac{1}{1 - p}\right]$$

calculate the derivatives using the chain rule

# Backpropagation

1. Derivatives need to be evaluated at some values of  $X, y$  and  $W$ .
2. But since we have an expression, we can build a function that takes as input  $X, y, W$  and returns the derivatives and then we can use gradient descent to update.
3. This approach works well but it does not generalize. For example if the network is changed, we need to write a new function to evaluate the derivatives.

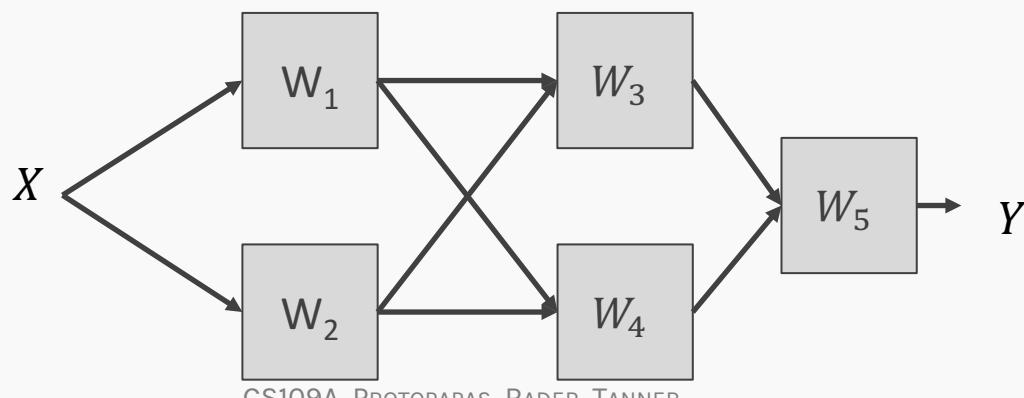
For example this network will need a different function for the derivatives



# Backpropagation

1. Derivatives need to be evaluated at some values of X,y and W.
2. But since we have an expression, we can build a function that takes as input X,y,W and returns the derivatives and then we can use gradient descent to update.
3. This approach works well but it does not generalize. For example if the network is changed, we need to write a new function to evaluate the derivatives.

For example this network will need a different function for the derivatives



# Backpropagation. Pavlos game #456

Need to find a formalism to calculate the derivatives of the loss wrt to weights that is:

1. Flexible enough that adding a node or a layer or changing something in the network won't require to re-derive the functional form from scratch.
2. It is exact.
3. It is computationally efficient.

Hints:

1. Remember we only need to evaluate the derivatives at  $X_i, y_i$  and  $W^{(k)}$ .
2. We should take advantage of the chain rule we learned before

# Idea 1: Evaluate the derivative at: $X=\{3\}$ , $y=1$ , $W=3$

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\xi_n}{dW}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$e^{-9}$	$e^{-9}$	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$	$-3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$	$-3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)$
$\mathcal{L}_i^A = -y \xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1	$3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$			-3	0.00037018372

# Basic functions

We still need to derive derivatives ☹

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\xi_n}{dW}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$e^{-9}$	$e^{-9}$	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$	$-3e^{-9} \left(\frac{1}{1+e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$	$-3e^{-9} \left(\frac{1}{1+e^{-9}}\right)$
$\mathcal{L}_i^A = -y \xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1	$3e^{-9} \left(\frac{1}{1+e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$			-3	0.00037018372



# Basic functions

elemental functions

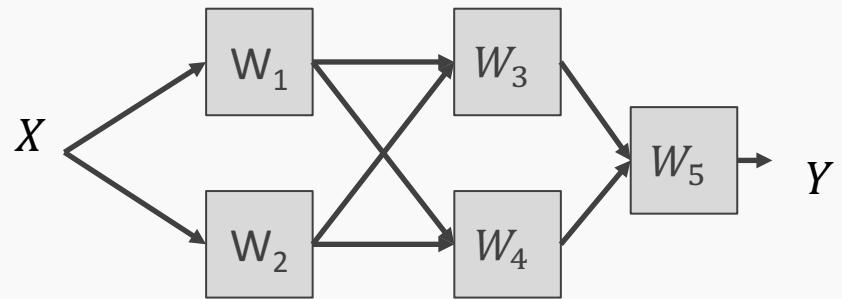
Notice though those are basic functions that my grandparent can do  
variable derivative

$\xi_0 = X$	$\frac{\partial \xi_0}{\partial X} = 1$	def x0(x): return X	def derx0(): return 1
$\xi_1 = -W^T \xi_0$	$\frac{\partial \xi_1}{\partial W} = -X$	def x1(a, x): return -a*x	def derx1(a, x): return -a
$\xi_2 = e^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	def x2(x): return np.exp(x)	def derx2(x): return np.exp(x)
$\xi_3 = 1 + \xi_2$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	def x3(x): return 1+x	def derx3(x): return 1
$\xi_4 = \frac{1}{\xi_3}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	def der1(x): return 1/(x)	def derx4(x): return -(1/x) ** (2)
$\xi_5 = \log \xi_4$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	def der1(x): return np.log(x)	def derx5(x): return 1/x
$\mathcal{L}_i^A = -y \xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	def der1(y, x): return -y*x	def derL(y): return -y



# Putting it altogether

1. We specify the network structure

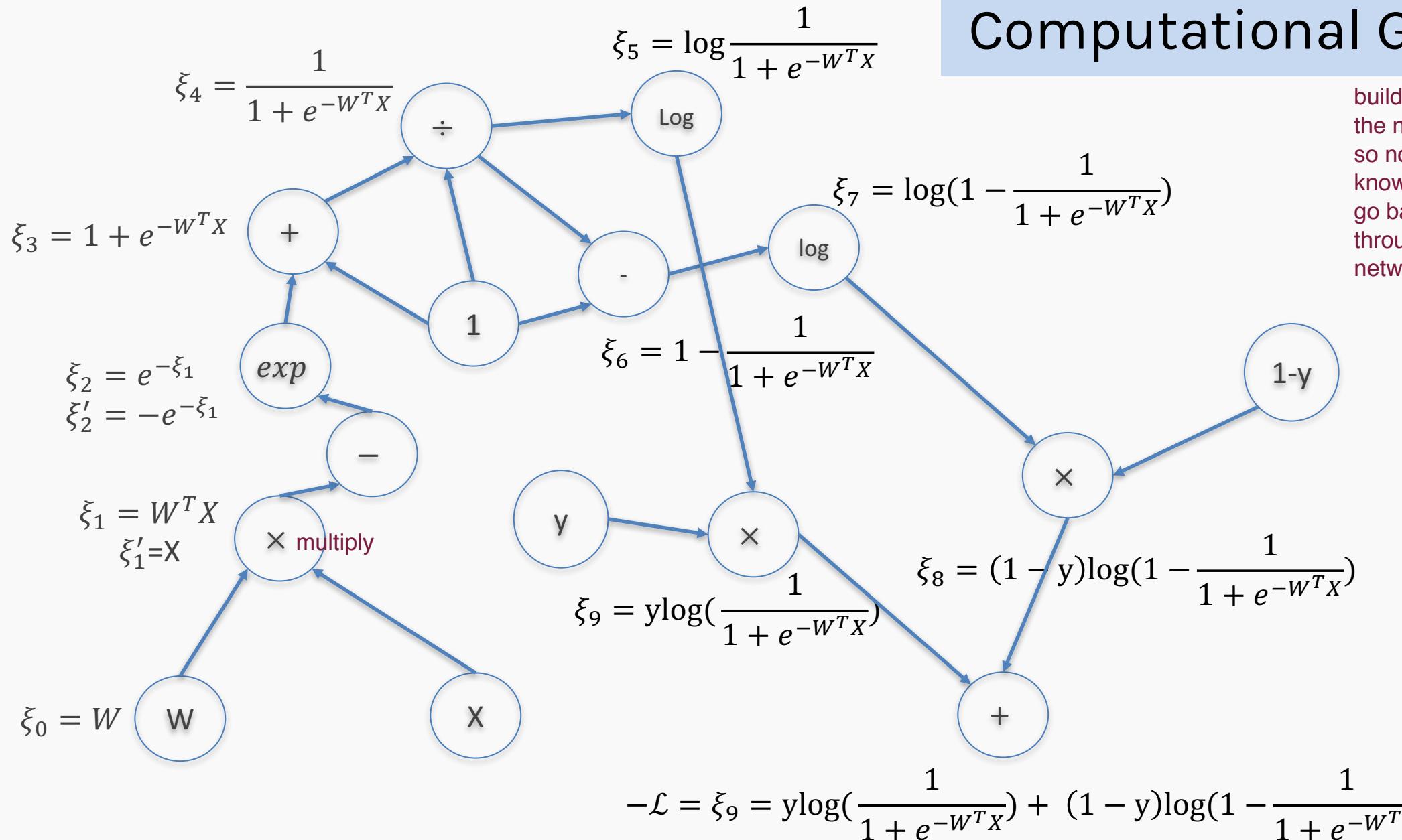


2. We create the computational graph ...

What is computational graph?

# Computational Graph

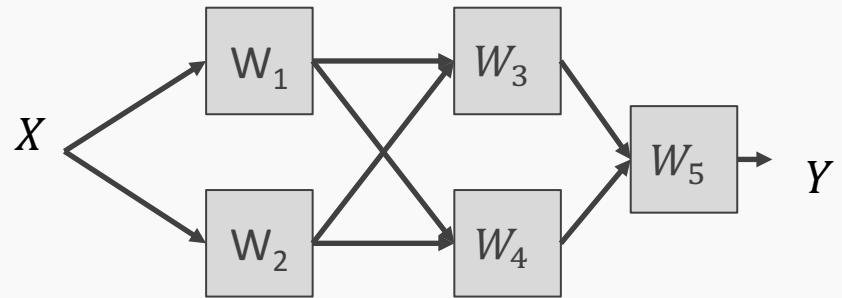
build it with the network, so now you know how to go back through the network



$$-\mathcal{L} = \xi_9 = y \log\left(\frac{1}{1 + e^{-W^T X}}\right) + (1 - y) \log\left(1 - \frac{1}{1 + e^{-W^T X}}\right)$$

# Putting it altogether

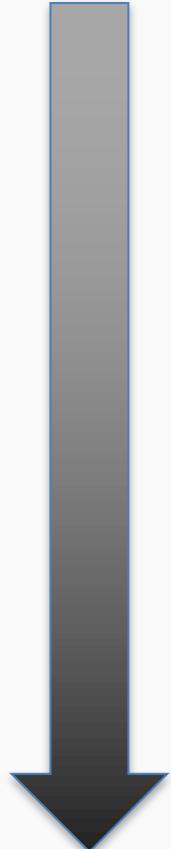
## 1. We specify the network structure



- We create the computational graph.
- At each node of the graph we build two functions: the evaluation of the variable and its partial derivative with respect to the previous variable (as shown in the table 3 slides back)
- Now we can either go forward or backward depending on the situation. In general, forward is easier to implement and to understand. The difference is clearer when there are multiple nodes per layer.

# Forward mode: Evaluate the derivative at: $X=\{3\}$ , $y=1$ , $W=3$

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\mathcal{L}}{d\xi_n}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$e^{-9}$	$e^{-9}$	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$	$-3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$	$-3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)$
$\mathcal{L}_i^A = -y \xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1	$3e^{-9} \left(\frac{1}{1 + e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$			-3	0.00037018372



# Backward mode: Evaluate the derivative at: $X=\{3\}$ , $y=1$ , $W=3$

Variables	derivatives	Value of the variable	Value of the partial derivative
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$e^{-9}$	$e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$			Type equation here.

Store all these values

more efficient in reverse mode for large networks

keras fit method:

calculate derivatives

use the derivatives with stochastic gradient descent to optimize loss fxn

everything we just did by hand!

# Optimizers



# Learning vs. Optimization

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

for the whole population of data

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

~~[ $L(f(x, W)), y]$~~

where  $f$  is the  
neural network

In practice, empirical risk minimization: from some training points

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

stochastic GD: you subsample your training points  
(one more level down)

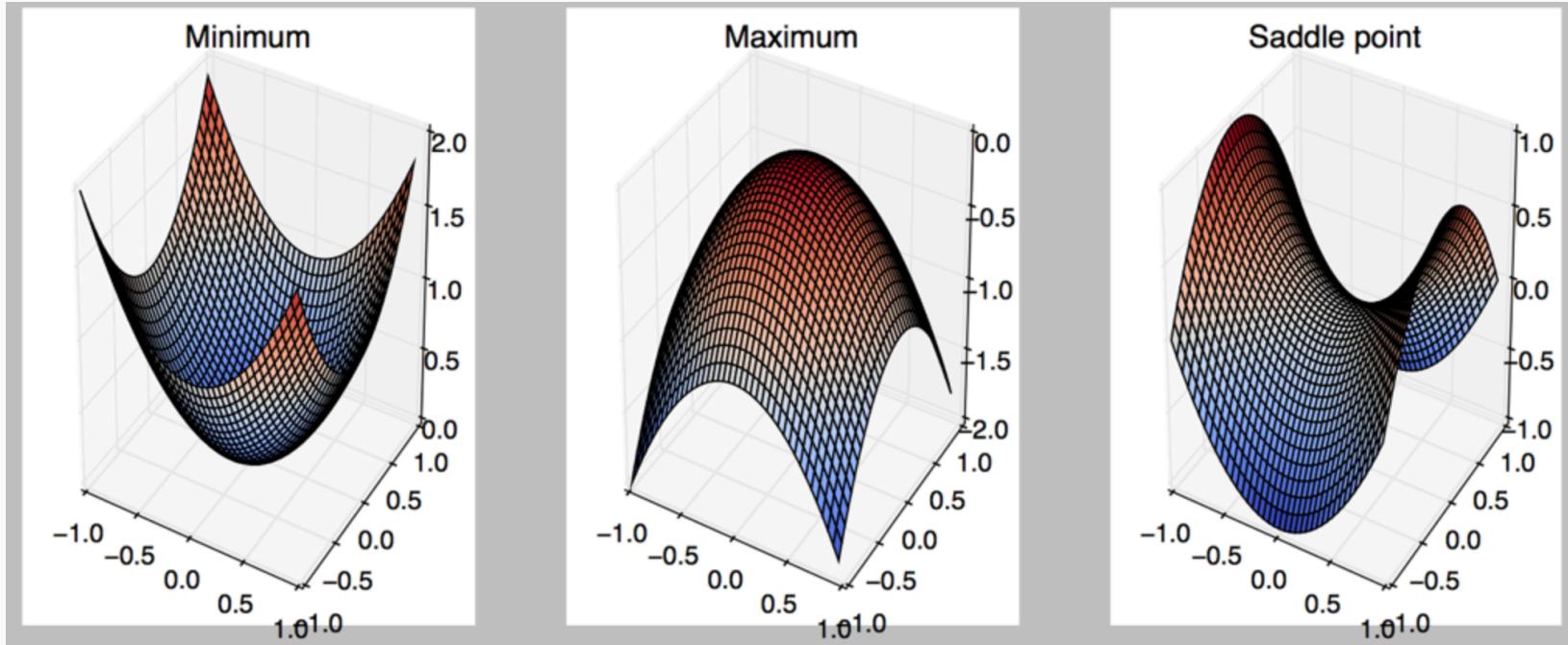
Quantity optimized  
different from the quantity  
we care about



# Critical Points

Points with **zero gradient**

2<sup>nd</sup>-derivate (Hessian) determines curvature



# Outline

---

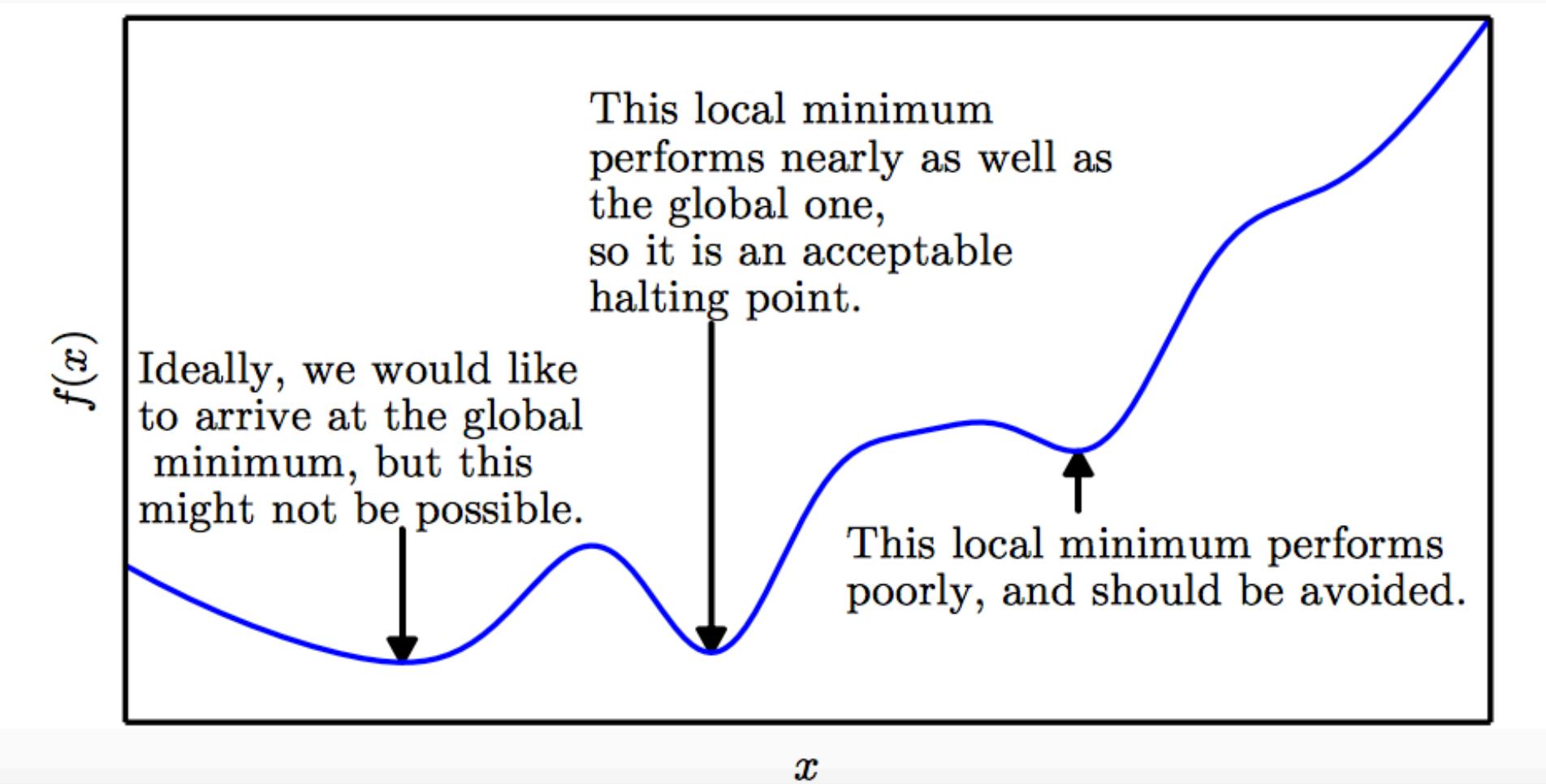
## Optimization

- Challenges in Optimization    changing the standard SGD (stochastic GD) to improve it
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization



# Local Minima

neural networks are difficult to optimize - you don't guarantee global minimum, so every time you do your neural network, you can end up somewhere else



# Local Minima

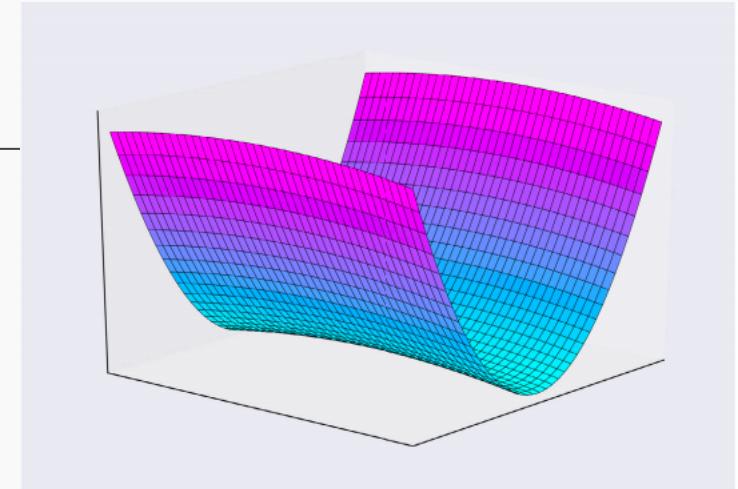
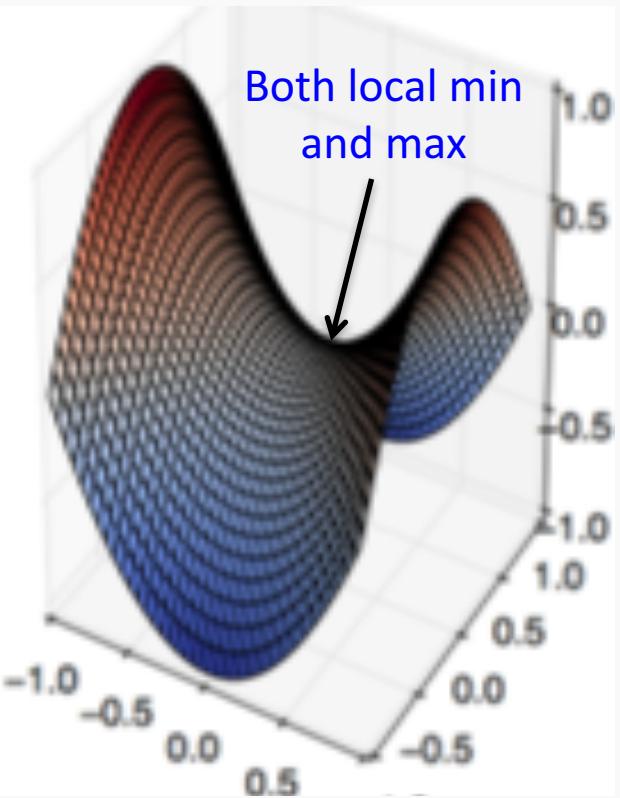
---

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

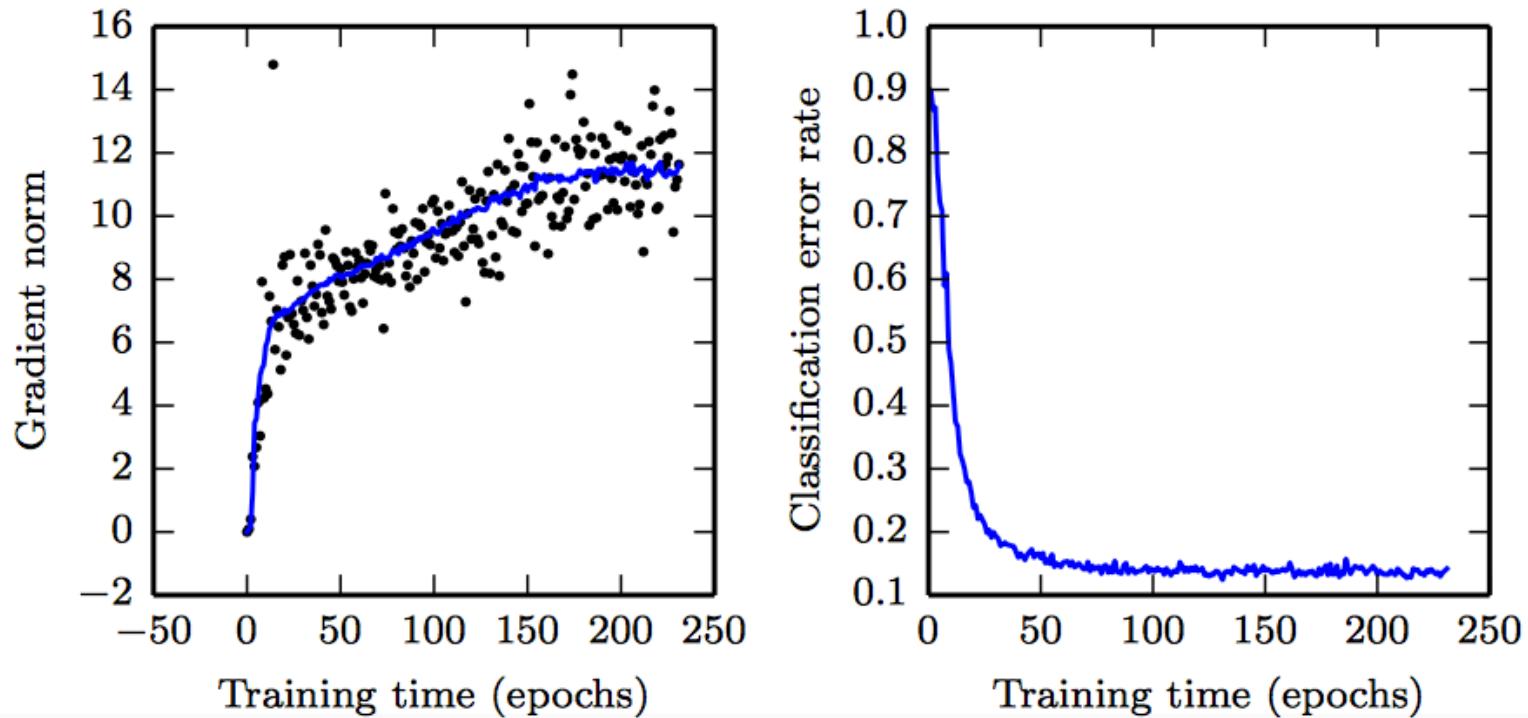


Recent studies indicate that in high dim, saddle points are more likely than local min

Gradient can be very small near saddle points

# No Critical Points

Gradient norm increases, but validation error decreases



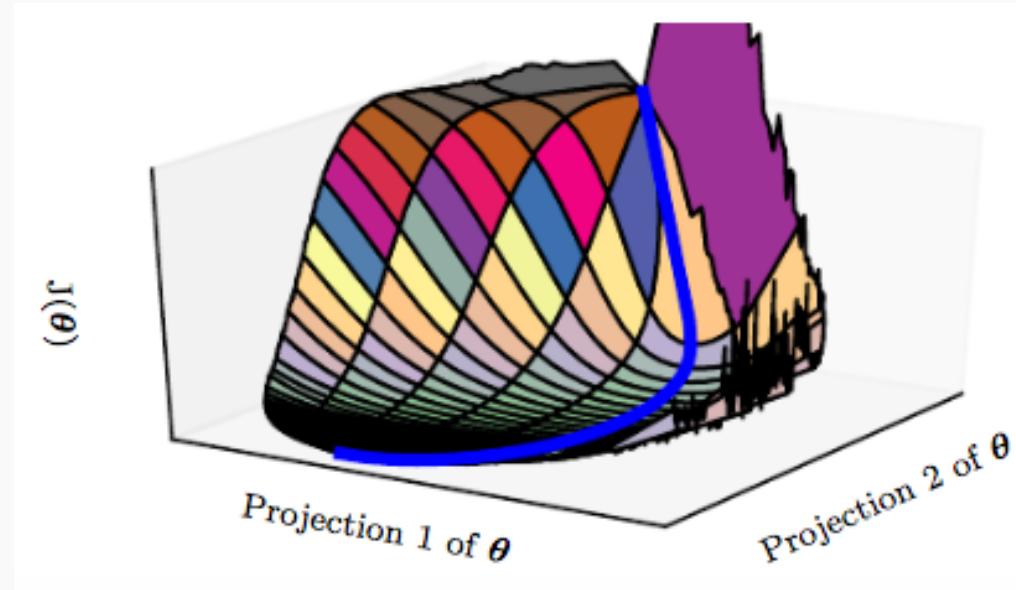
*Convolution Nets for Object Detection*



# Saddle Points

SGD is seen to escape saddle points

- Moves down-hill, uses noisy gradients



Second-order methods get stuck

- solves for a point with zero gradient

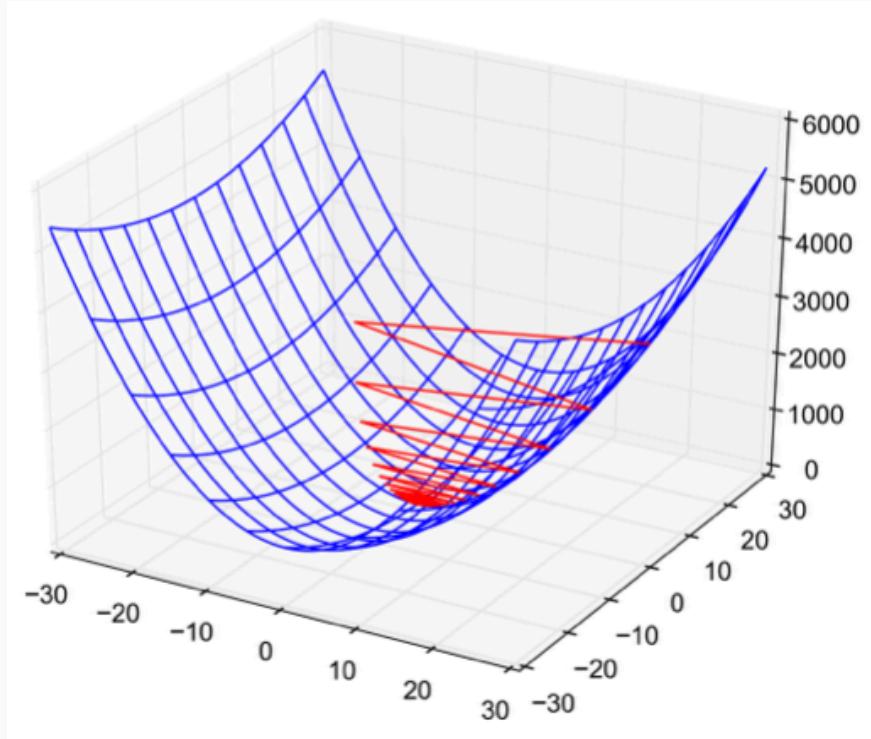
# Poor Conditioning

Poorly conditioned Hessian matrix

- High curvature: small steps leads to huge increase

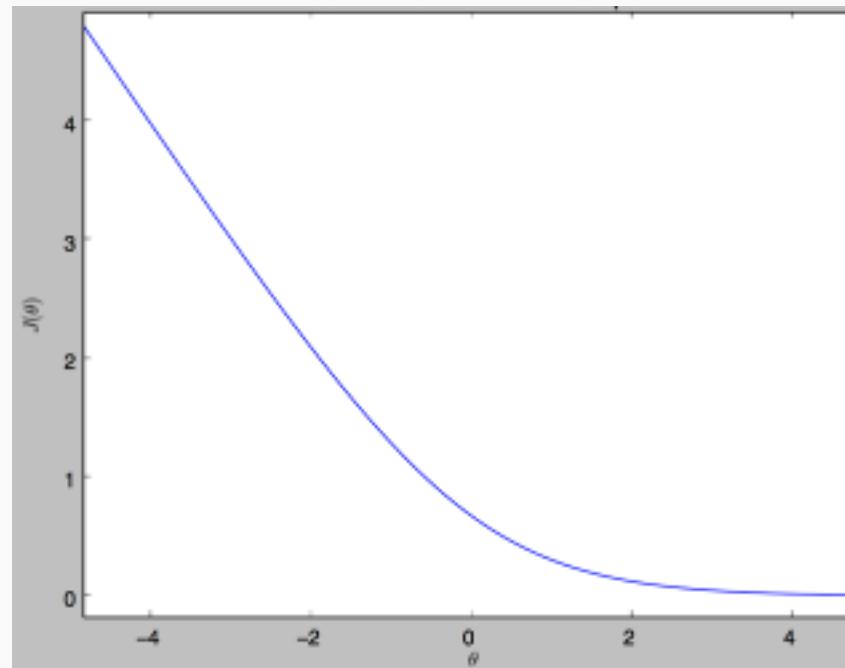
Learning is slow despite strong gradients

Oscillations slow down progress

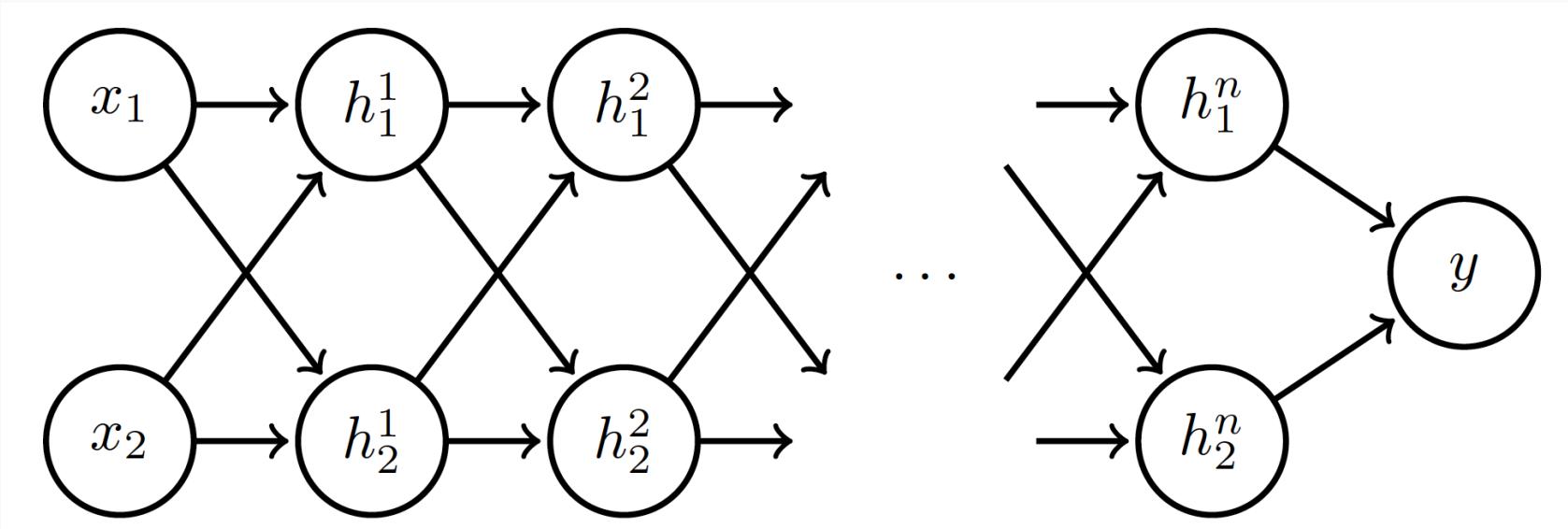


# No Critical Points

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



# 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:  $a = 0.5, b = 0.9$ :

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



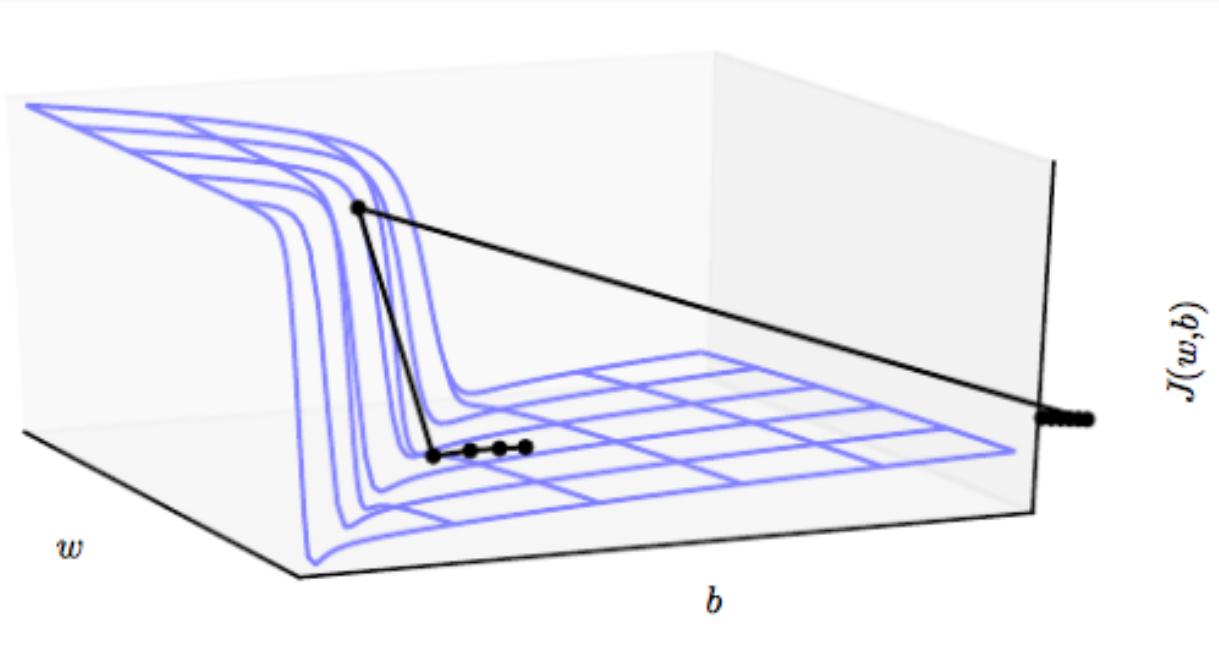
# Exploding and Vanishing Gradients

Exploding gradients lead to cliffs

Can be mitigated using **gradient clipping**

if  $\|g\| > u$

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



# Outline

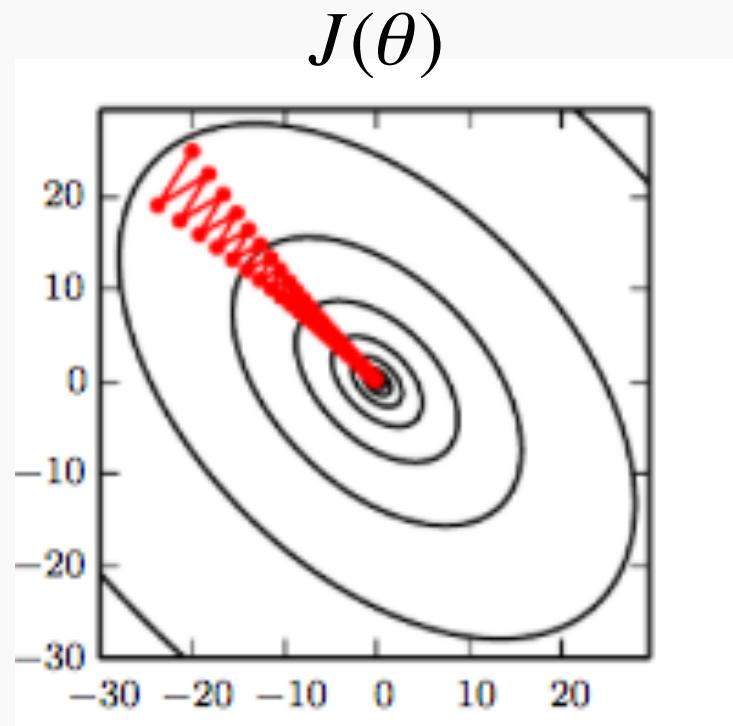
---

## Optimization

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



# Stochastic Gradient Descent

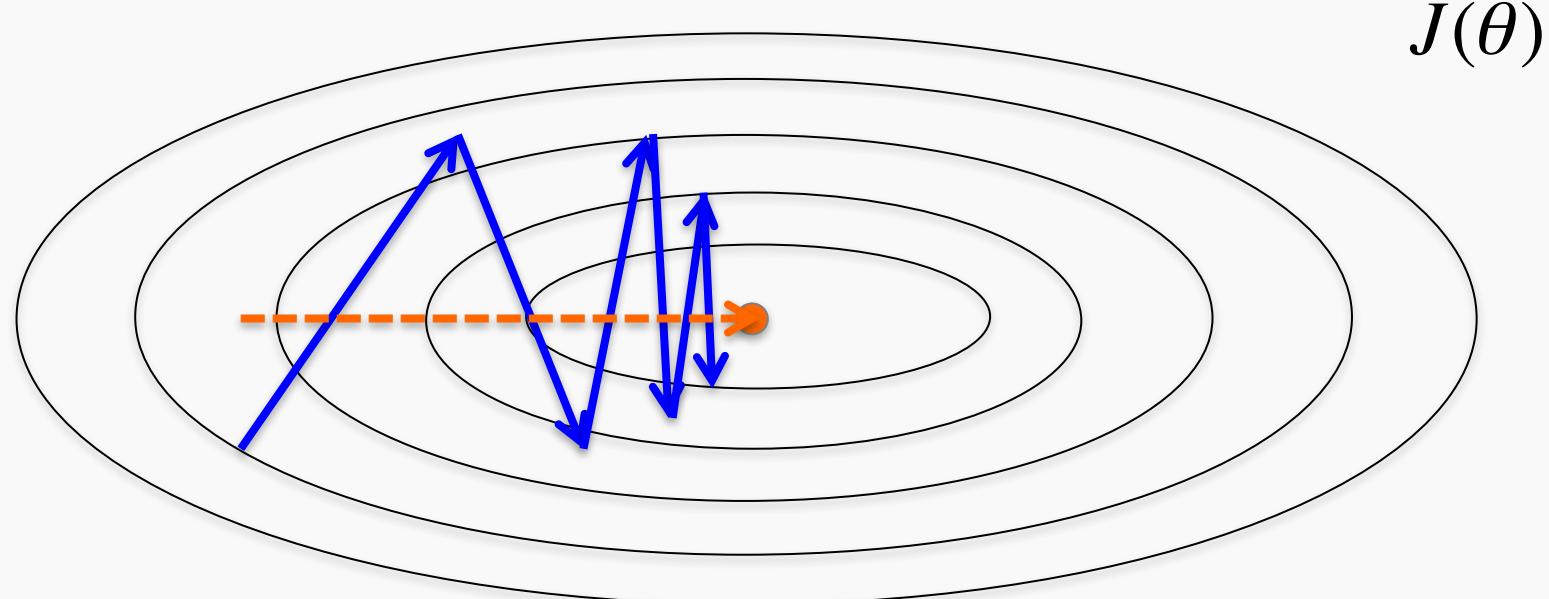


Oscillations because  
updates do not exploit  
curvature information

Goodfellow et al. (2016)

# Momentum

SGD is slow when there is **high curvature**



Average gradient presents faster path to opt:

- vertical components cancel out

# Momentum

Uses **past gradients** for update

Maintains a new quantity: ‘**velocity**’

Exponentially decaying average of gradients:

$$g = \frac{1}{m} \sum_i \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$$

Current gradient update

$$v = \alpha v + (-\varepsilon \overrightarrow{g})$$

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



# Momentum

---

Compute gradient estimate:

$$g = \frac{1}{m} \sum_i \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$$

Update velocity:

$$\nu = \alpha \nu - \epsilon g$$

Update parameters:

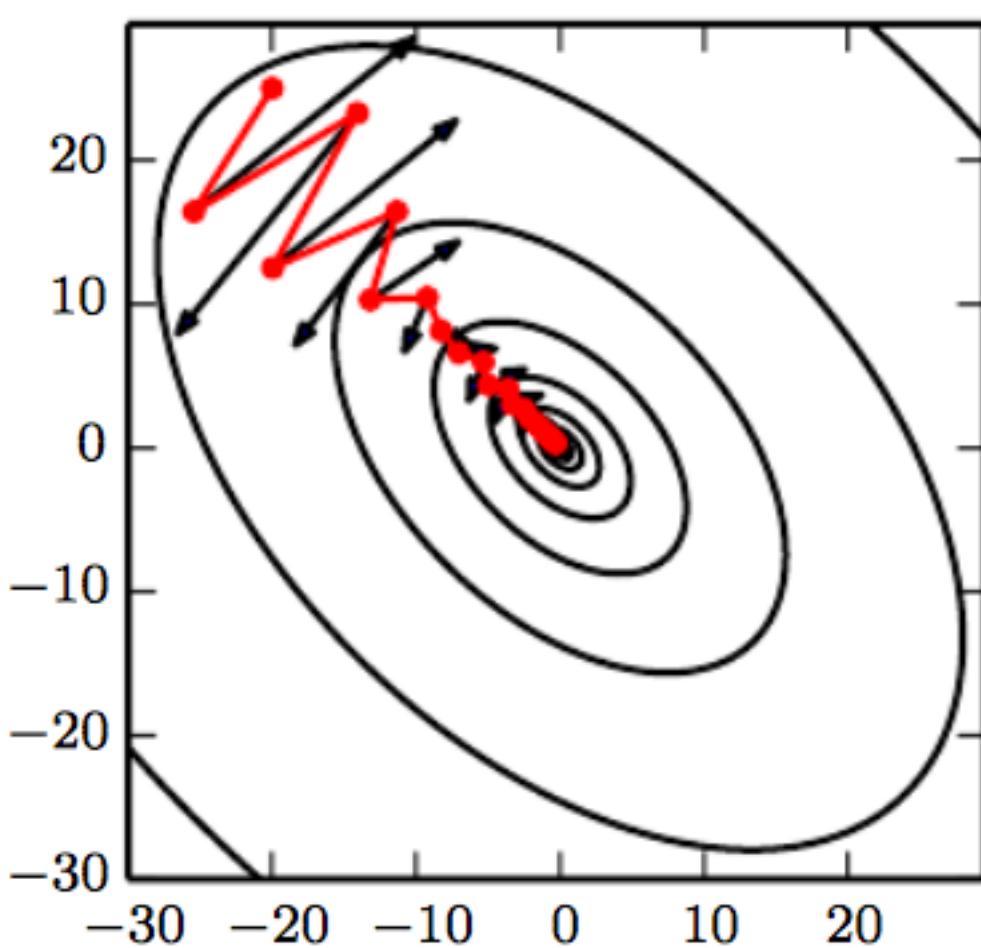
$$\theta = \theta + \nu$$



# Momentum

*Damped oscillations:*  
gradients in opposite  
directions get  
cancelled out

$$J(\theta)$$



# Nesterov Momentum

Apply an **interim** update:

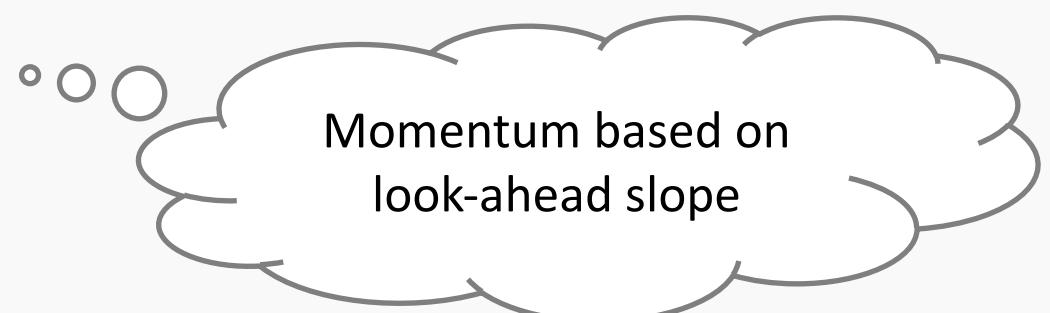
$$\tilde{\theta} = \theta + v$$

Perform a correction based on gradient at the interim point:

$$g = \frac{1}{m} \sum_i \nabla_{\theta} L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$$

$$v = \alpha v - \varepsilon g$$

$$\theta = \theta + v$$





► LiveSlides web content

To view

**Download the add-in.**

[liveslides.com/download](http://liveslides.com/download)

**Start the presentation.**

# Outline

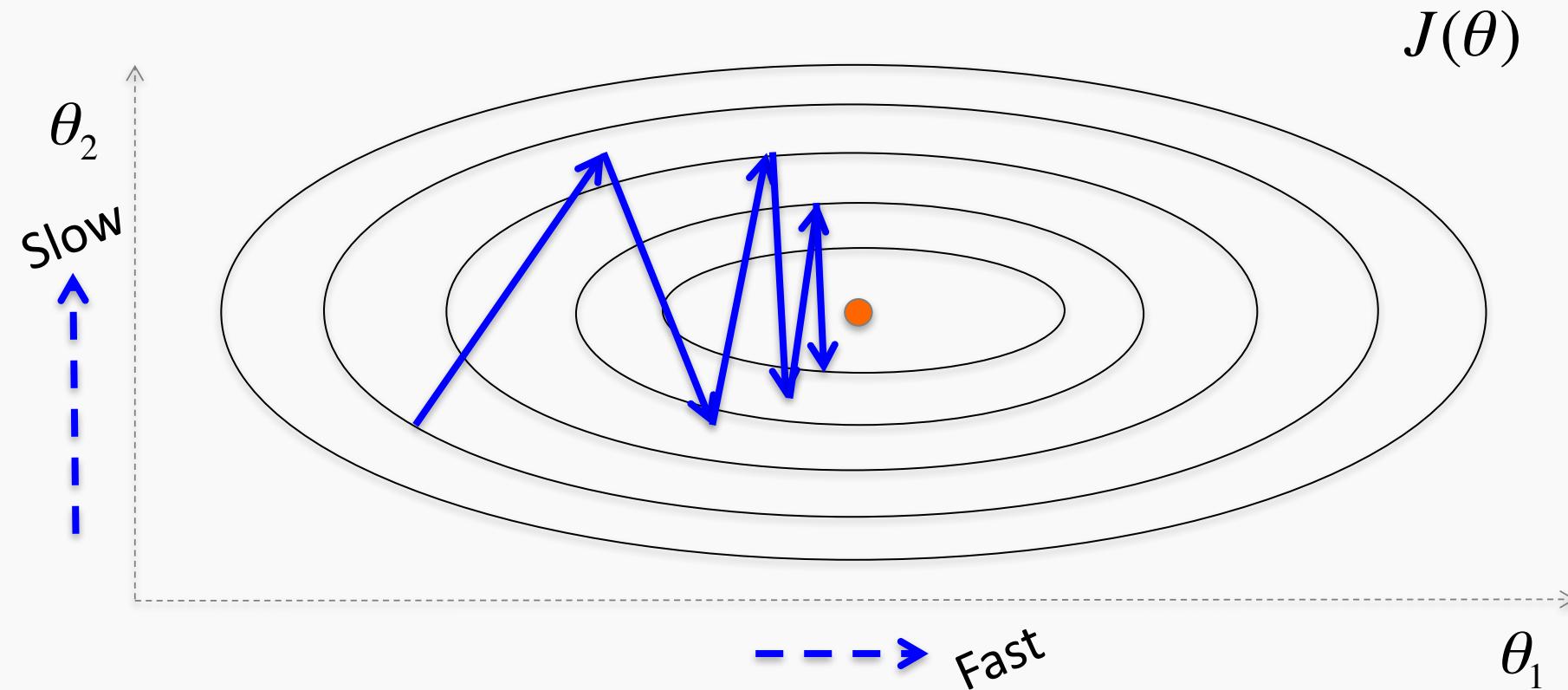
---

## Optimization

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



# 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:

$$r_i = r_i + g_i^2$$

- Update each parameter:

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

Inversely  
proportional to  
cumulative  
squared gradient

- Greater progress along gently sloped directions

# RMSProp

---

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

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

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

# 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:

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

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

# Outline

---

## Optimization

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





# 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

# Bias Initialization

---

- Output unit bias
  - Marginal statistics of the output in the training set
- Hidden unit bias
  - Avoid saturation at initialization
  - E.g. in ReLU, initialize bias to 0.1 instead of 0
- Units controlling participation of other units
  - Set bias to allow participation at initialization





► LiveSlides web content

To view

**Download the add-in.**

[liveslides.com/download](http://liveslides.com/download)

**Start the presentation.**

# Outline

---

Challenges in Optimization

Momentum

Adaptive Learning Rate

Parameter Initialization

**Batch Normalization**



# Feature Normalization

Good practice to normalize features before applying learning algorithm:

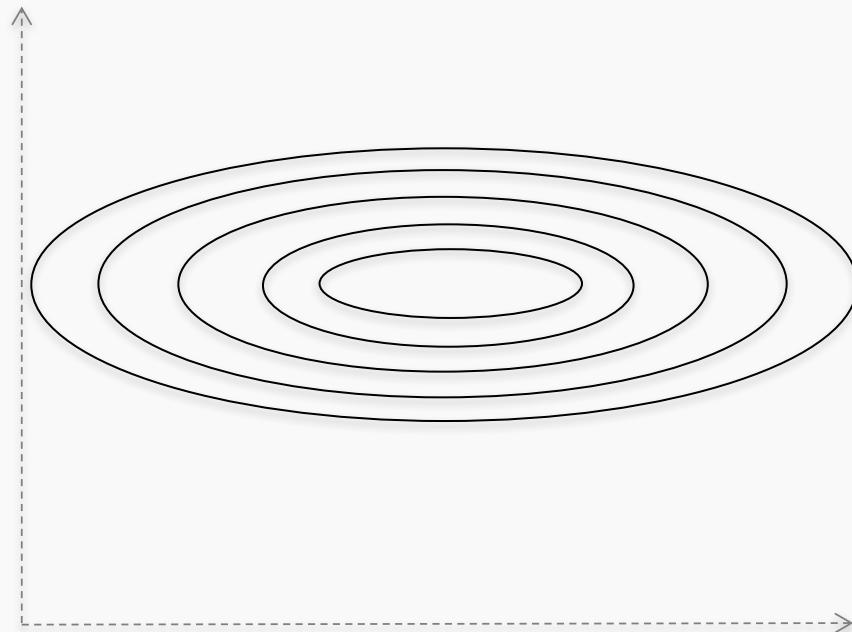
$$x' = \frac{x - \mu}{\sigma}$$

Feature vector       $x$       Vector of mean feature values  
                         $\mu$   
                         $\sigma$       Vector of SD of feature values

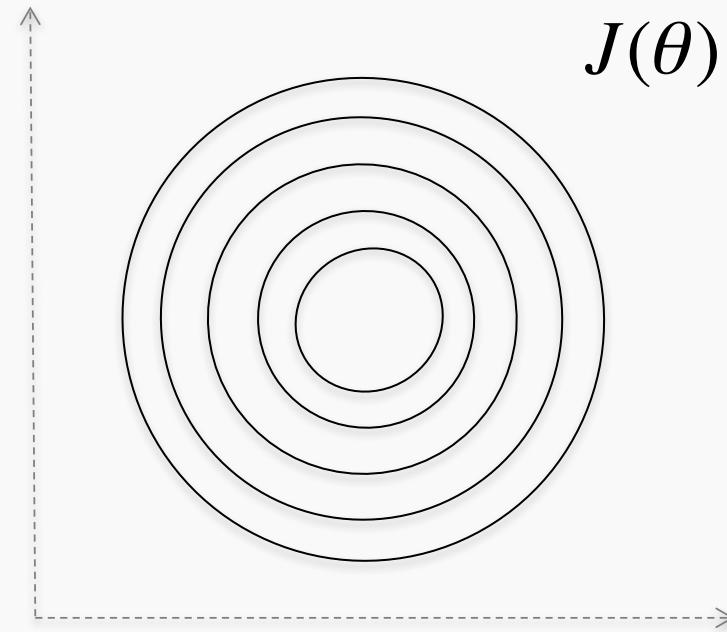
Features in **same scale**: mean 0 and variance 1

- Speeds up learning

# Feature Normalization



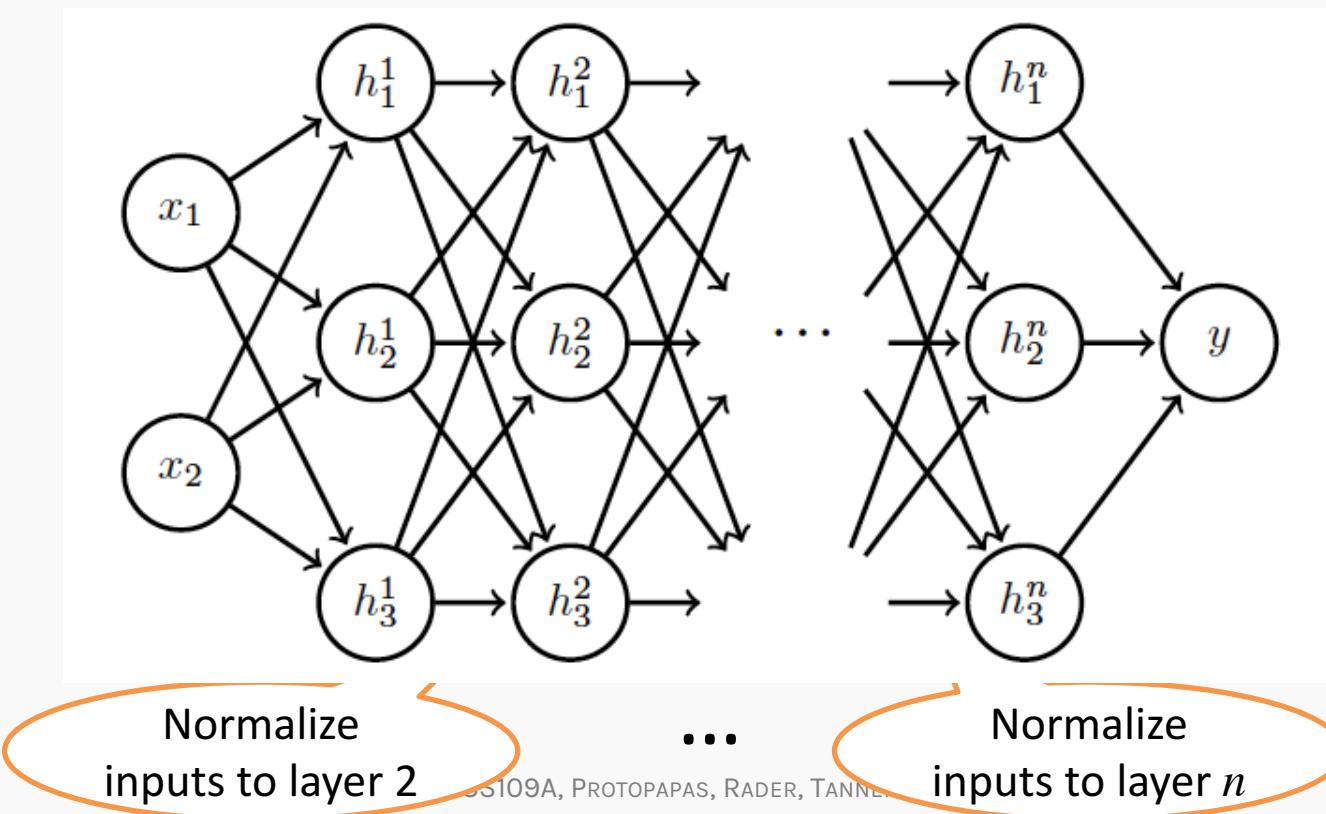
Before normalization



After normalization

# 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



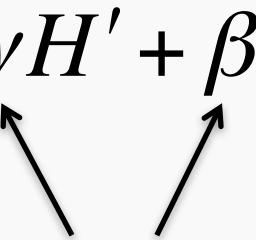
# Batch Normalization

Training time:

- Normalization can reduce expressive power
- Instead use:

$$\gamma H' + \beta$$

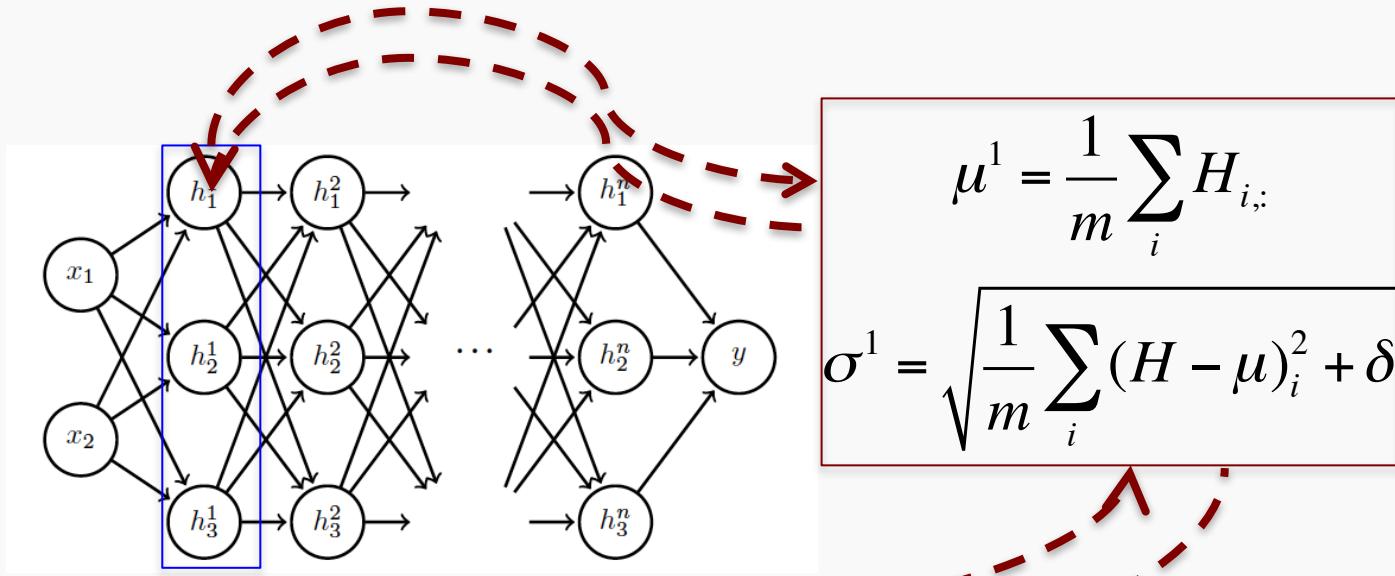
Learnable parameters



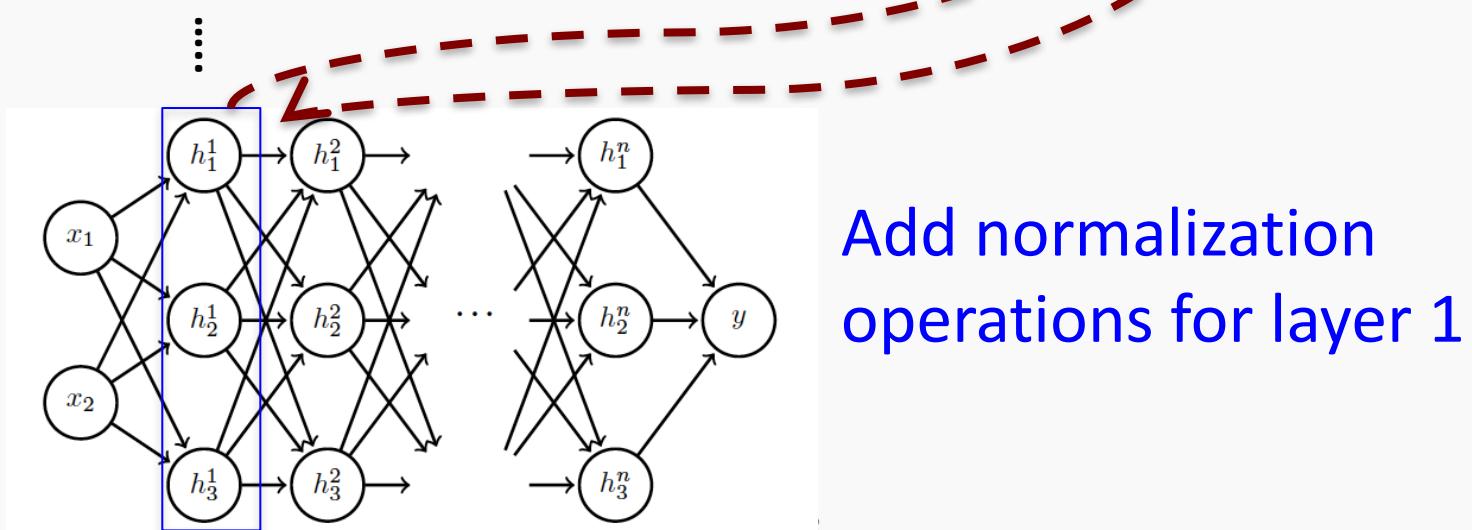
- Allows network to **control range of normalization**

# Batch Normalization

Batch 1

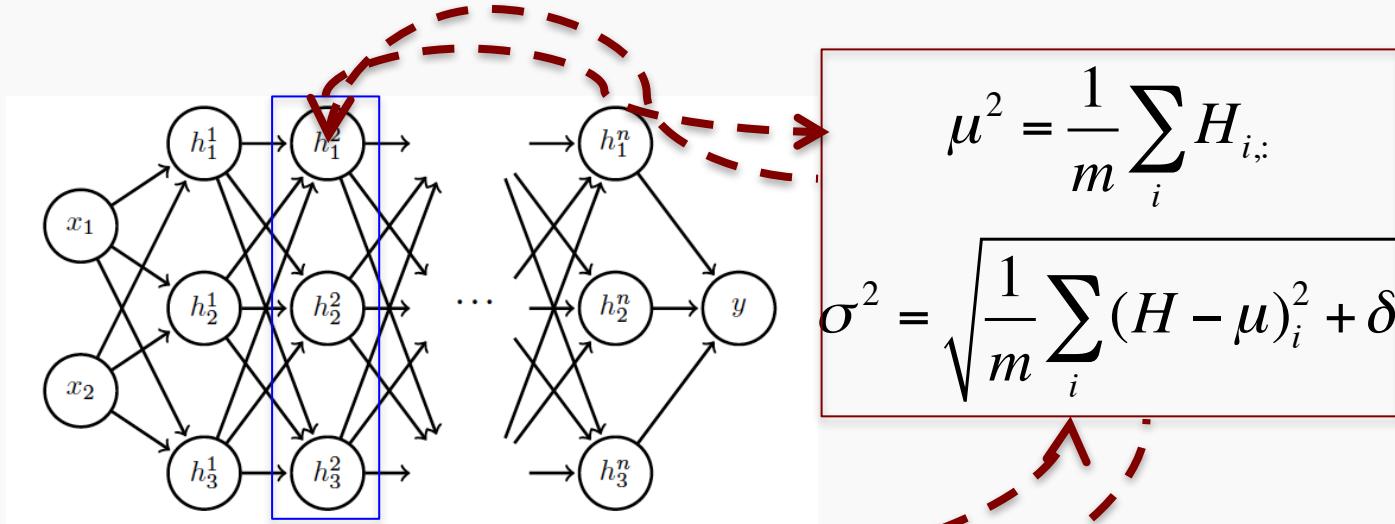


Batch  $N$

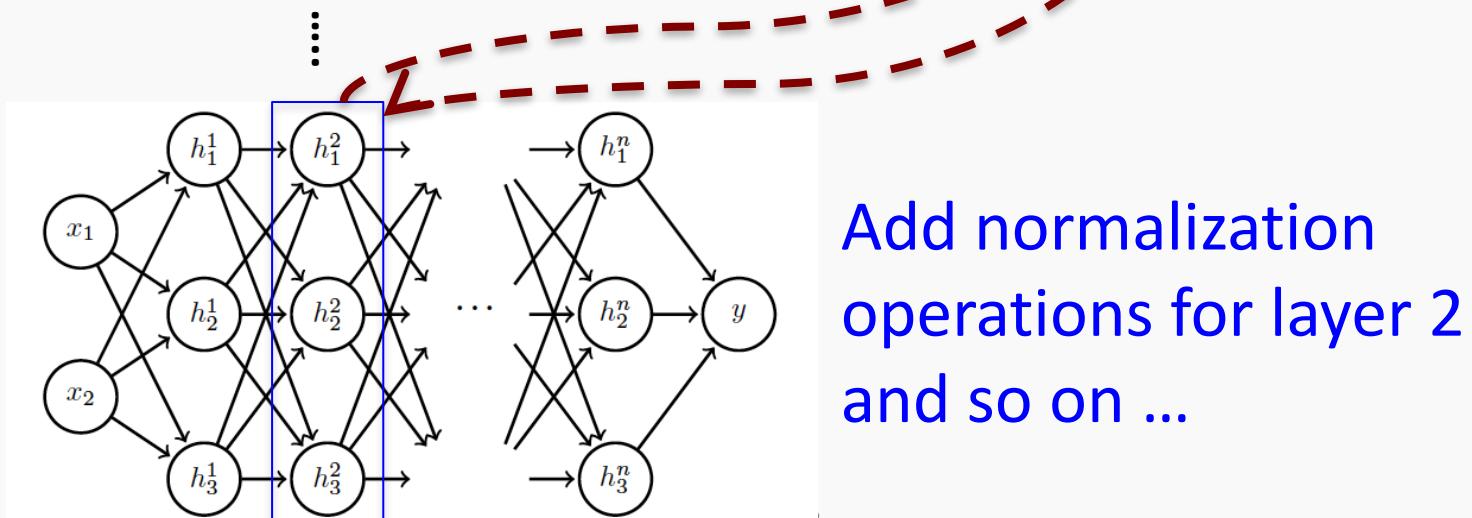


# Batch Normalization

Batch 1



Batch  $N$



Add normalization  
operations for layer 2  
and so on ...

# 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  $\mu$  and  $\sigma$  with **running averages** collected during training

