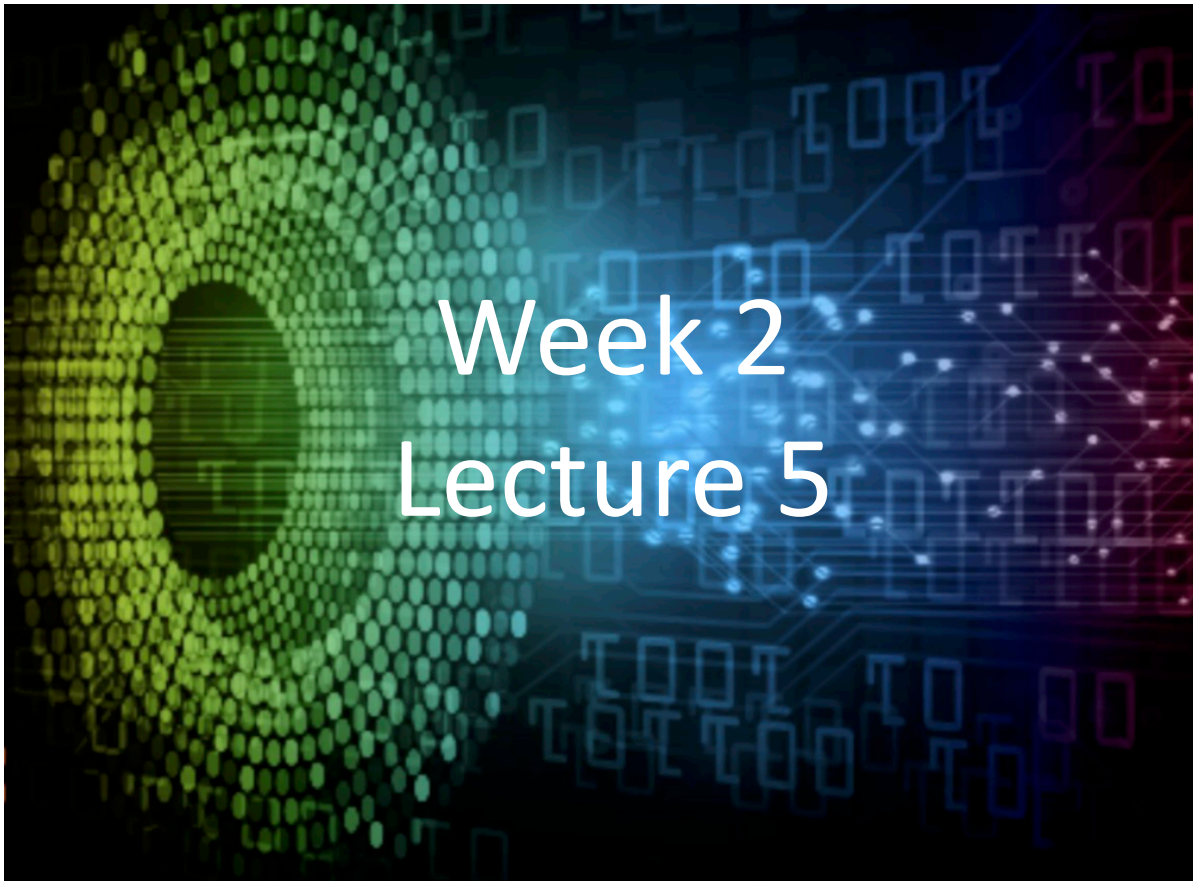
Introduction to Deep Learning
Applications and Theory



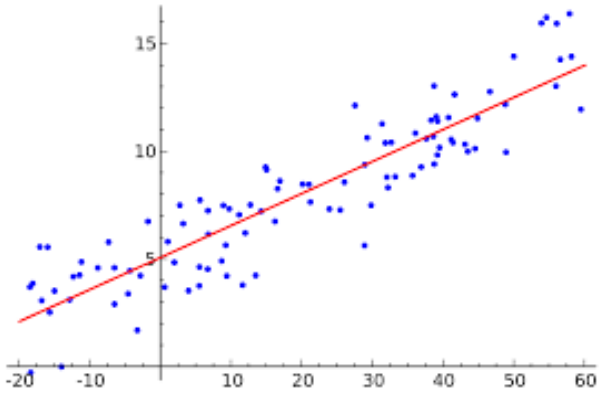
ECE 596 / AMATH 563

Previous Lecture: **Regression and Loss**

1. Regression /Classification

2. Loss and Cost Functions

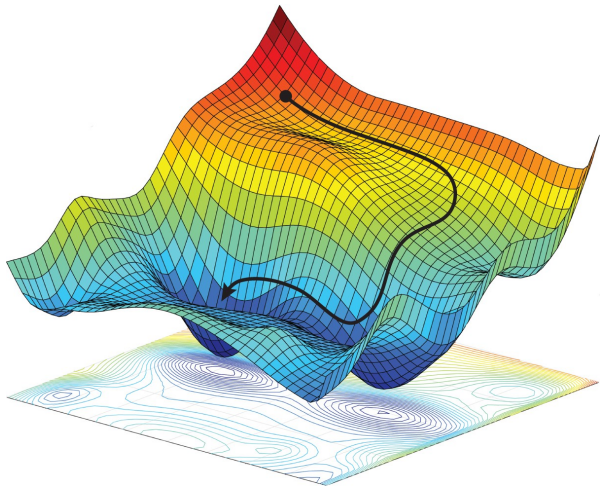
3. Logistic Regression example



Current Lecture: **Network Training/Optimization**

1. Gradient Descent

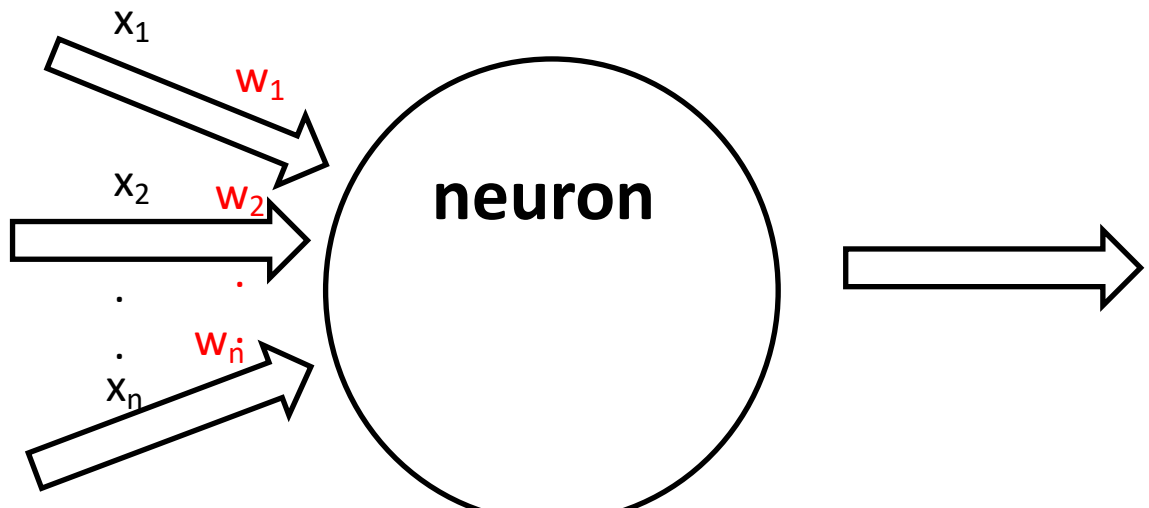
2. Training the Logistic Regression Loss



3. Classification/ Regression with NN

Logistic Regression

We'll start with single neuron logistic regression



input

$$\vec{x} \in \mathbb{R}^n$$

output

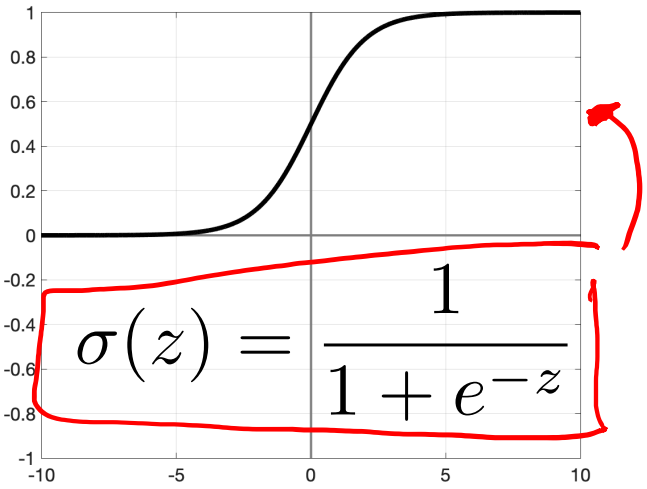
$$\hat{y} = P(y = 1|\vec{x})$$

$$0 \leq \hat{y} \leq 1$$

Activation

input

$\vec{x} \in \mathbb{R}^n$



Sigmoid

output

$\hat{y} = P(y = 1|\vec{x})$

$0 \leq \hat{y} \leq 1$

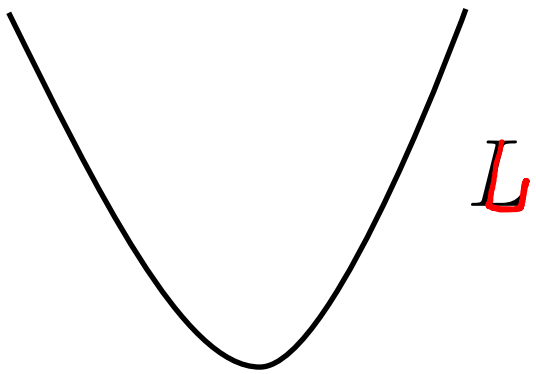
$\hat{y} = \sigma(\vec{w}^T \vec{x} + b)$

Loss

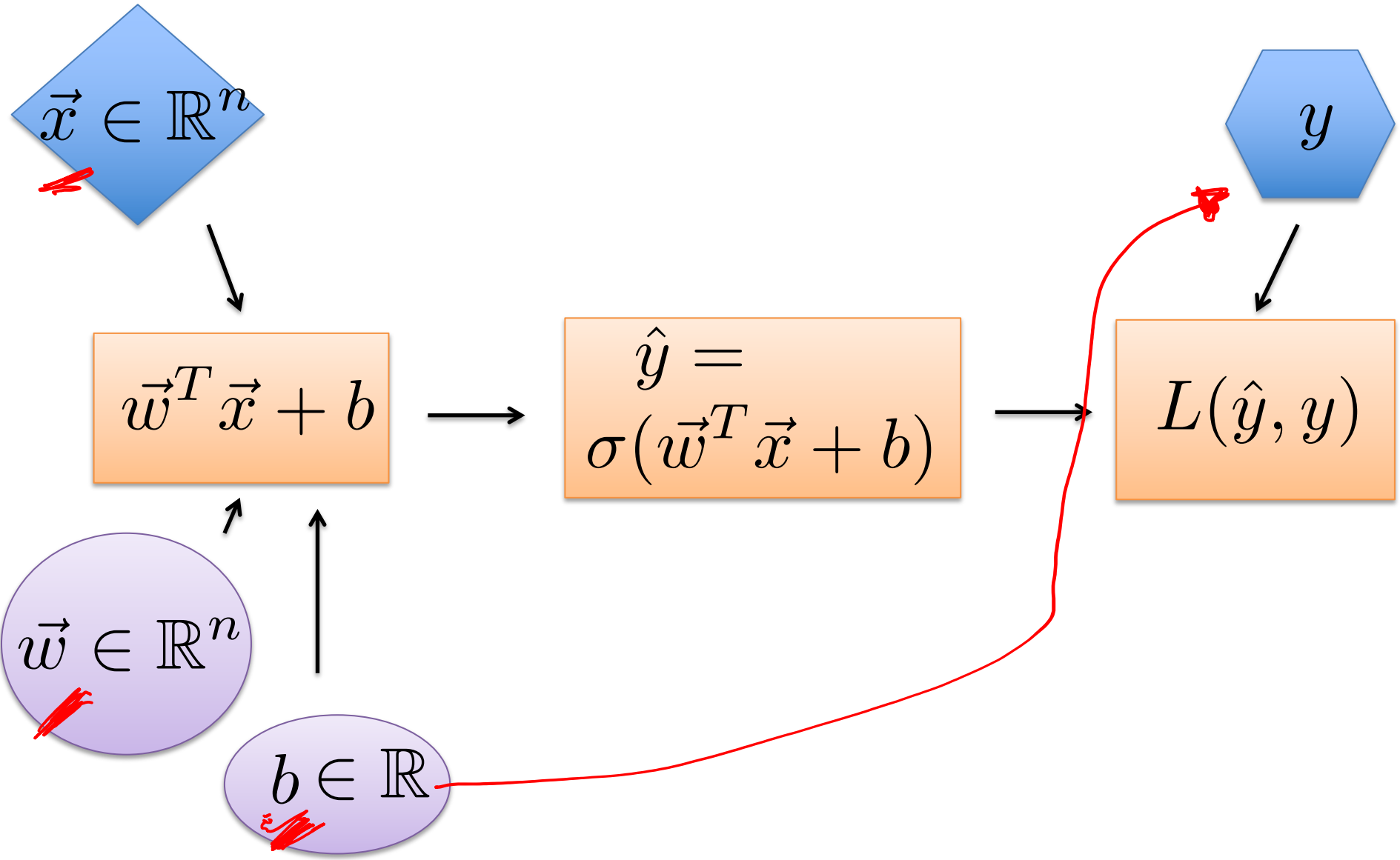
$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y}))$$

Cross Entropy

- Maximum Likelihood
- Convex Optimization



Computation Graph



Optimizing the parameters

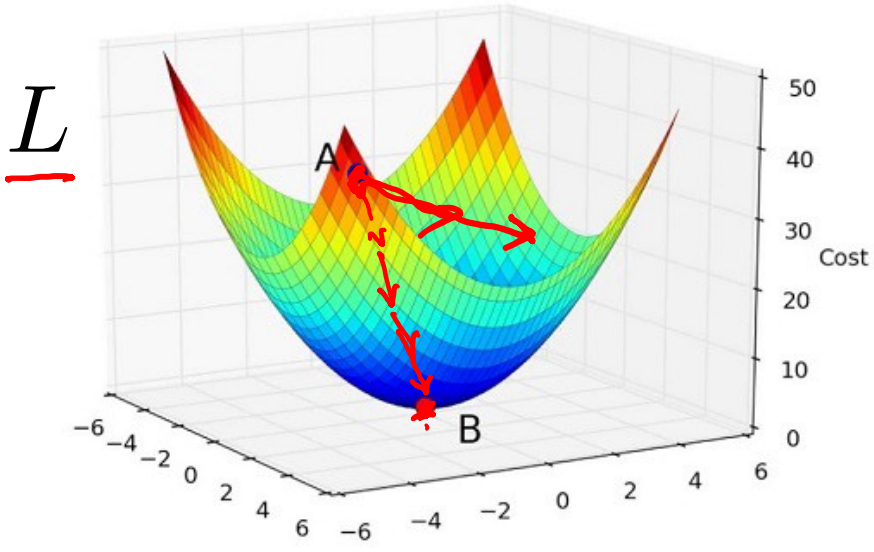
$$\vec{p}^{*} = \arg\min_{\vec{p}} L\left(\vec{p}; \hat{\vec{y}}(\vec{x})\right)$$

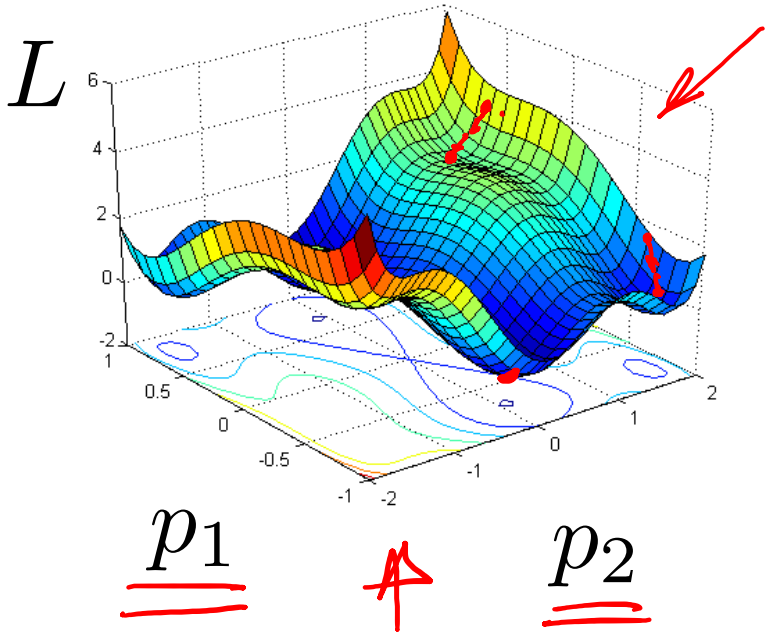
L

Optimizing the parameters

$$L\left(\vec{p}; \vec{\hat{y}}(\vec{x})\right)$$

$$\nabla_{\vec{p}} L\left(\vec{p}; \vec{\hat{y}}(\vec{x})\right)$$





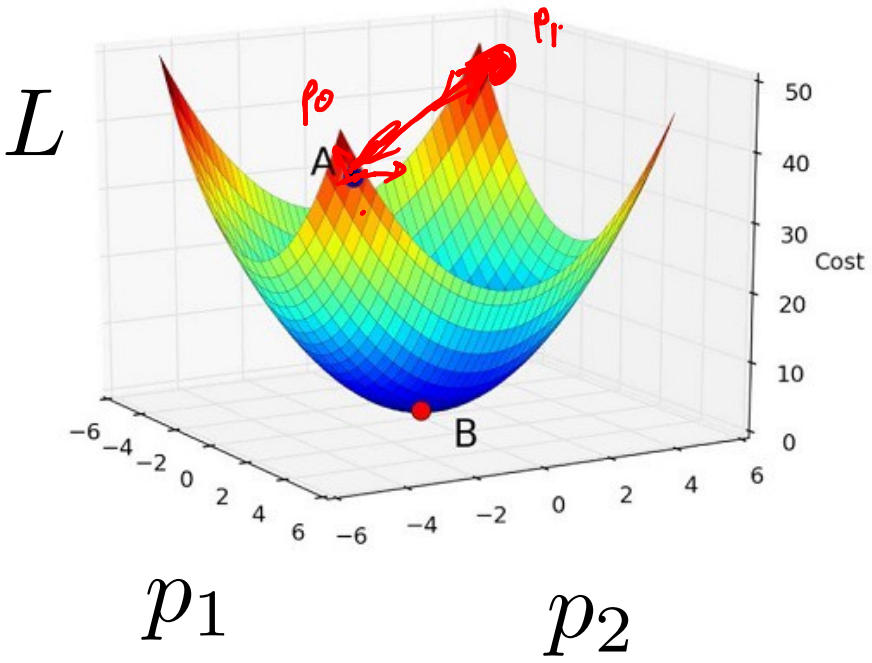
Gradient Descent (GD)

$$\vec{p}_{k+1} = \vec{p}_k - \alpha \nabla_{\vec{p}} L\left(\vec{p}; \hat{\vec{y}}(\vec{x})\right)$$

- **α** is the **learning** rate
- ***k*** is the stepping of updates for **p**

- Will converge to **global minimum** for convex function **J (under conditions)**
- Will converge to **local minimum** for non-convex function **J (under conditions)**

Gradient Descent



Learning rate

- The cost function is **convex** and **differentiable**
- The **gradient** is **Lipshitz** continuous with constant K then GD converges with O(1/K).
  Learning rate: **α < K**

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$$

- In practice:
  - Can optimize for **α** as well
  - Adaptive **α**
  - Alternating gradient descent

Q1. List several extensions to control the learning rate. Describe their pros and cons.

Gradient Descent

$$\vec{w}_{k+1} = \vec{w}_k - \alpha \nabla_{\vec{w}} J(\vec{w}_k; b)$$ GD $\vec{w}$

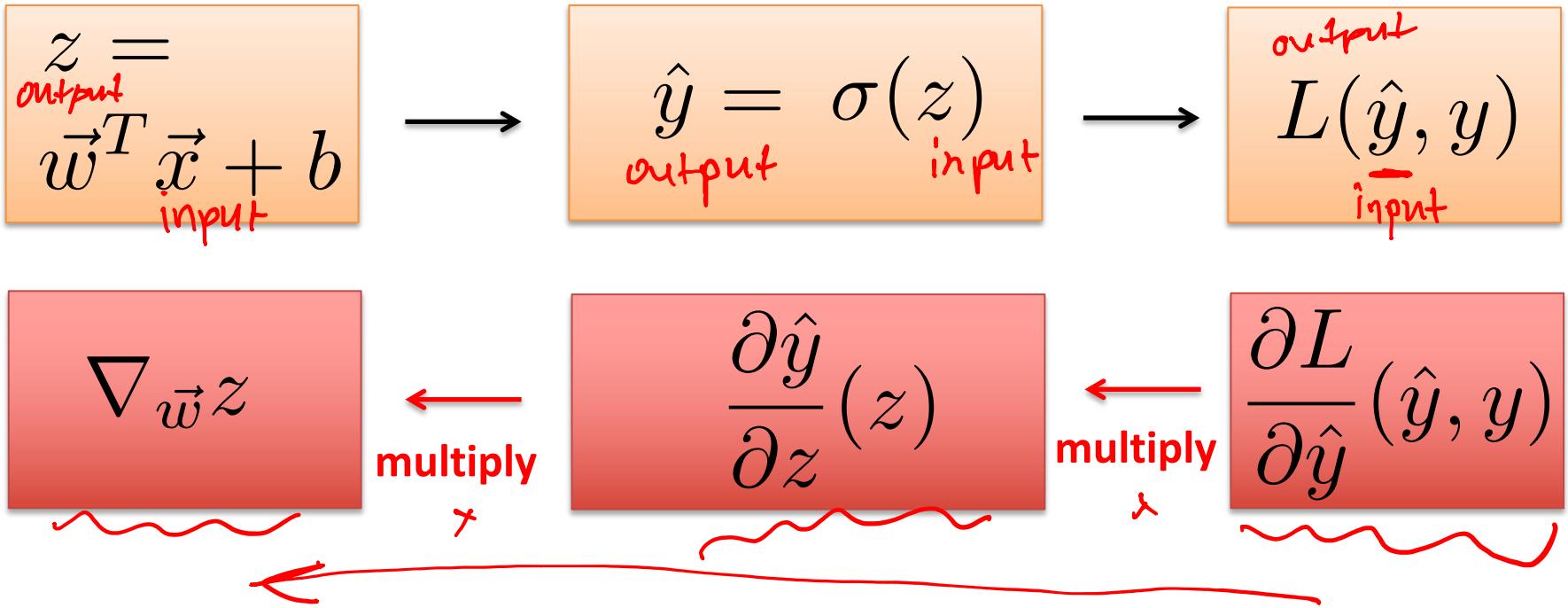
$$b_{k+1} = b_k - \alpha \frac{\partial}{\partial b} J(\vec{w}; b_k)$$ GD b

- Assume $i = 1$, and initial $\vec{w}_0, b_0$
- How do we compute $\vec{w}_1, b_1$?

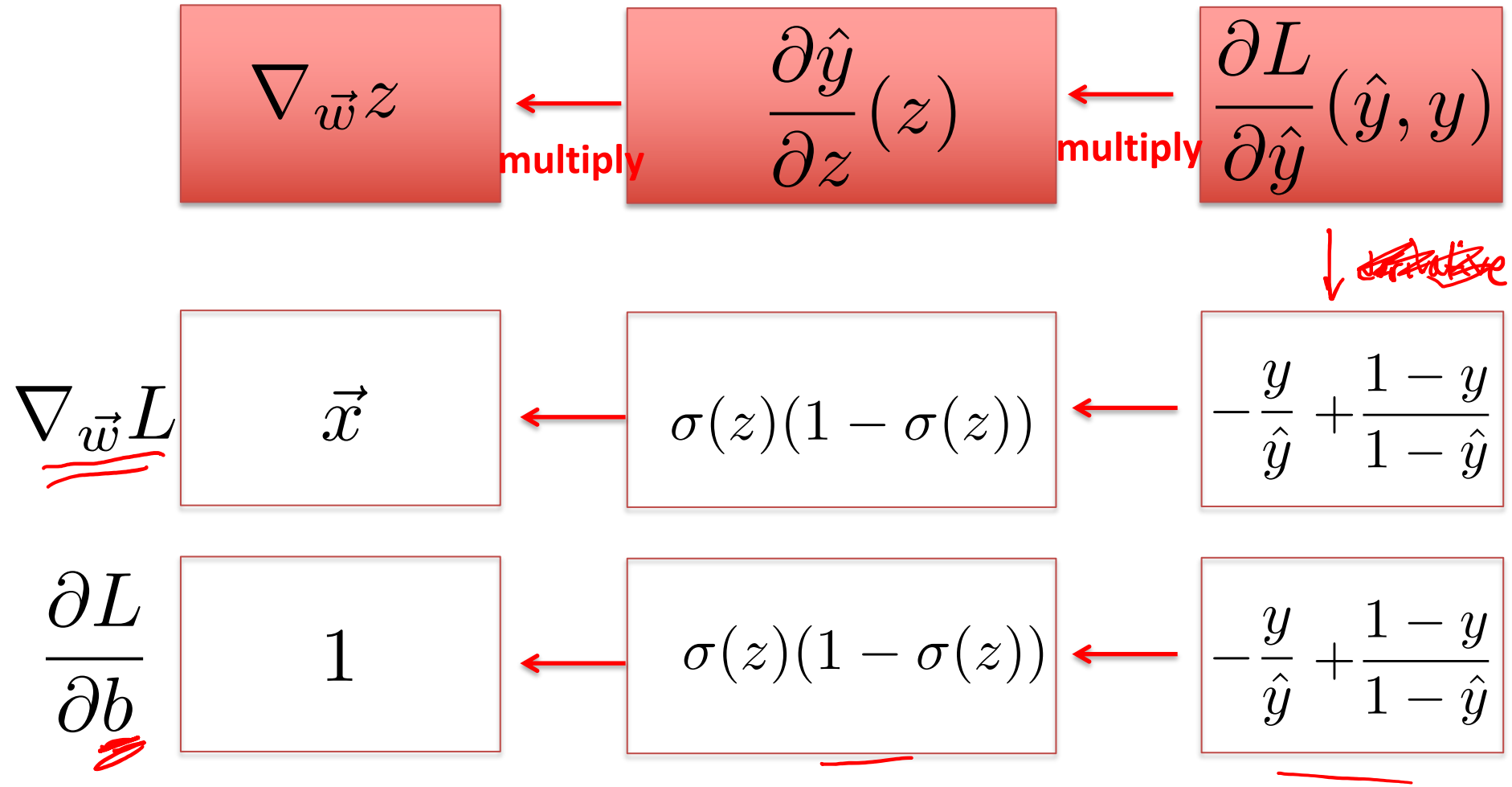
Our cost: $J = L(\hat{y}, y)$

Back Propagation – Chain Rule

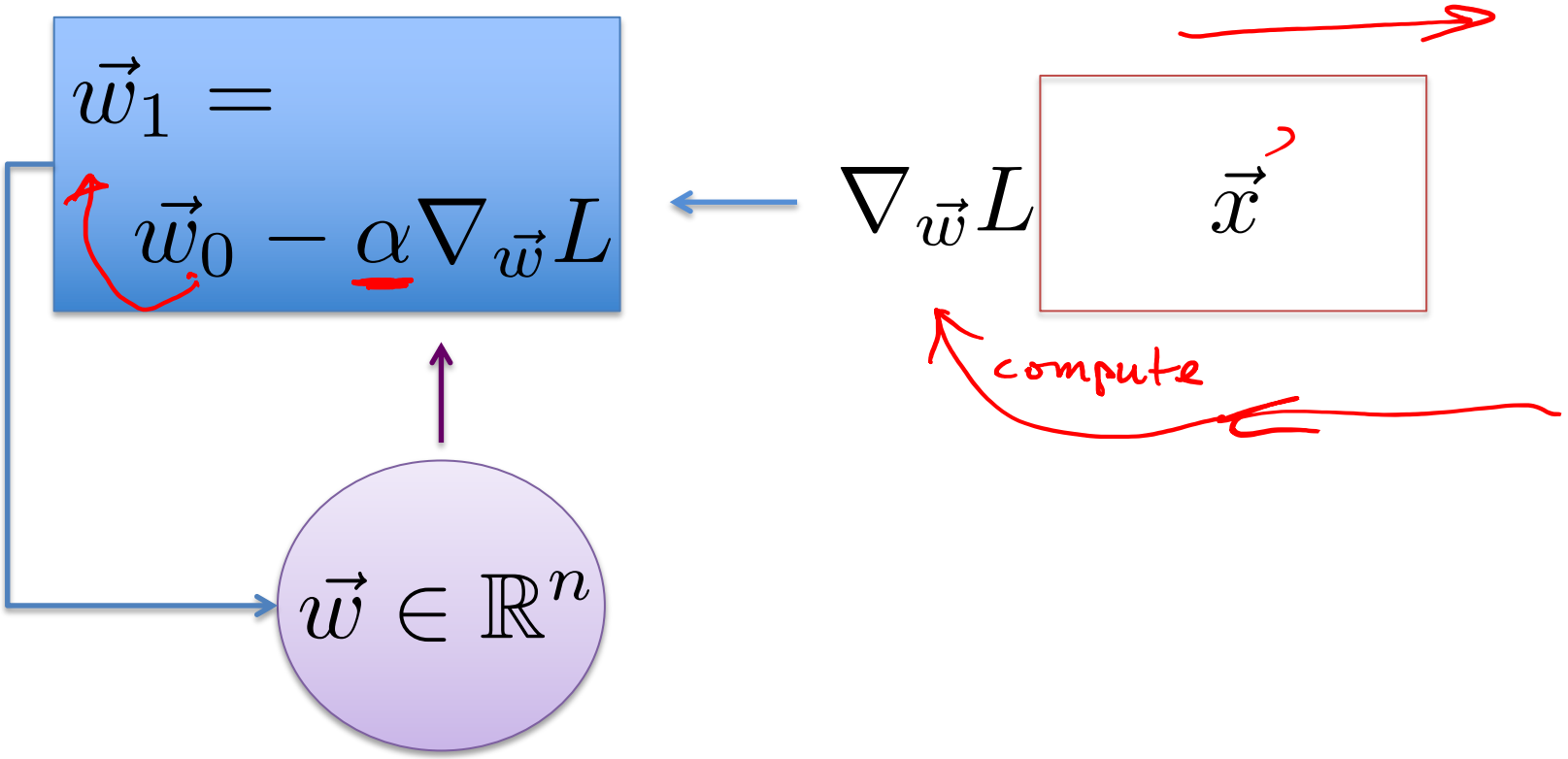
$$\nabla_{\vec{w}} L(\hat{y}, y) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \nabla_{\vec{w}} z$$



Back Propagation



Weights Update



- similarly for b_1

Multiple Examples Training set

- **Forward Propagation:** Computing the **loss through forward pass** for a single training example
- **Backward Propagation:** Computing **gradients of parameters** through backward pass for a single training example
- **Batch:** Training set could be divided into **smaller sets** called batches
- **Iteration:** When an **entire batch** is passed both **forward and backward**
- **Epoch:** When an **entire dataset** is passed both **forward and backward** through the NN once

Multiple Examples Training set

$$D : \{(\vec{x}^{(1)}, y^{(1)}), .., (\vec{x}^{(i)}, y^{(i)}), .., (\vec{x}^{(m)}, y^{(m)})\}$$

(Batch) GD:

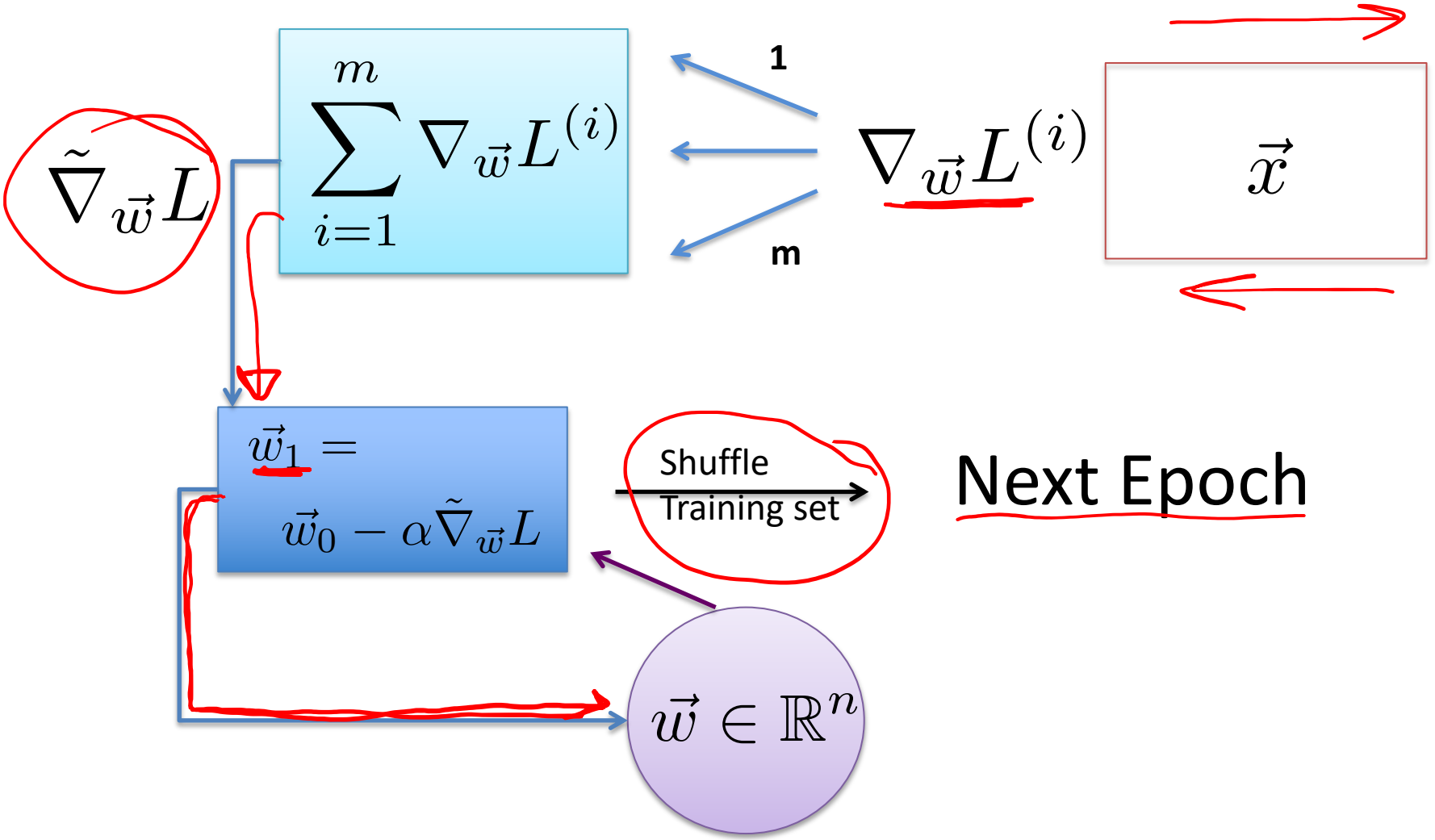
Since our cost function is

$$\nabla J(\{\hat{y}\}^m, \{y\}^m; \{\vec{x}\}^m) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

We will sum the gradients for **all m examples (Epoch)**

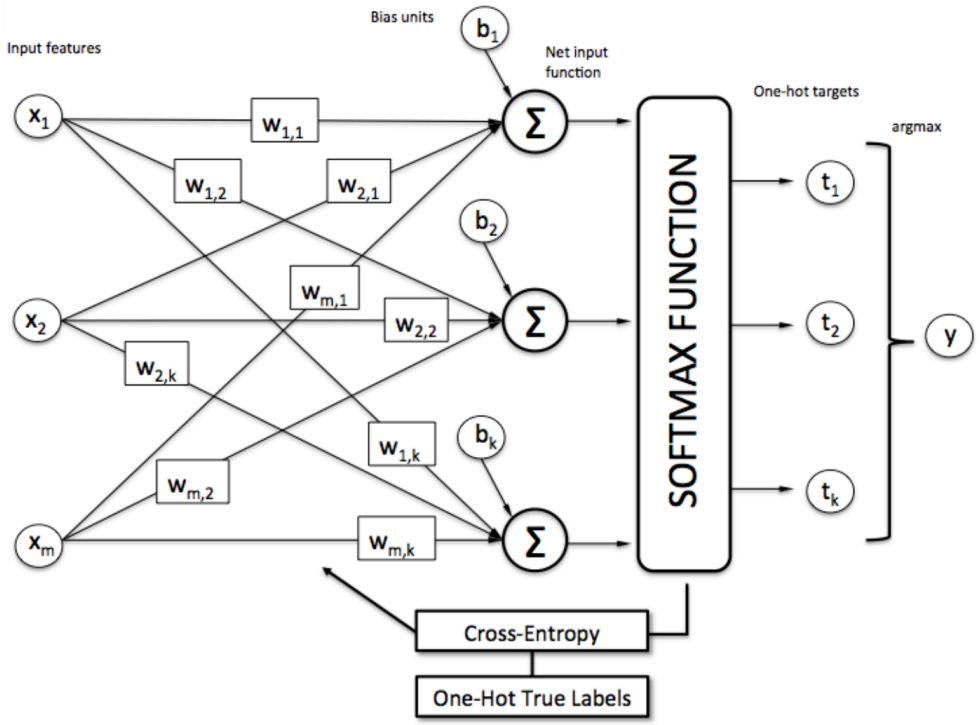
Graph

Batch GD



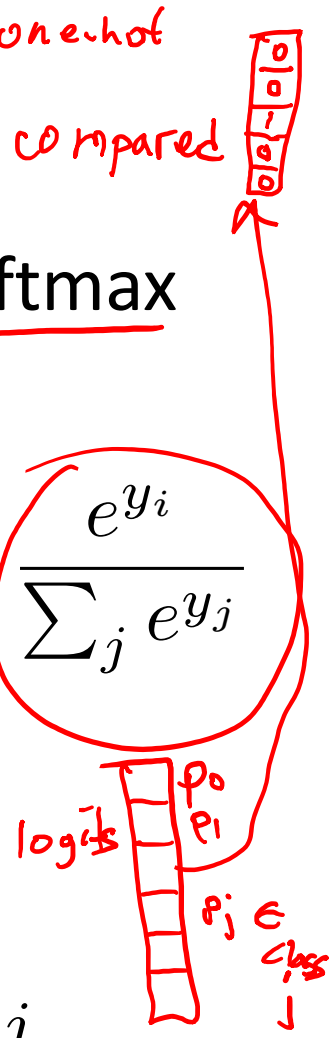
Next Epoch

Multiple Outputs



Sigmoid -> Softmax

$$softmax(\hat{y})_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$



$$\nabla_{\vec{w}_j} L_j(\hat{y}_j, y_j) = \frac{\partial L_j}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_j} \nabla_{\vec{w}_j} z_j$$

Q2. Describe the softmax function purpose and its relation to one-hot encoding and logits.

Other Loss/ Activations

- Changing **activation** will change the gradients

- Changing the **loss** will change the gradients and could make the composition unseparable

- Most activations we discussed have analytic gradients. It is possible that there is no analytic expression.

Gradient Check

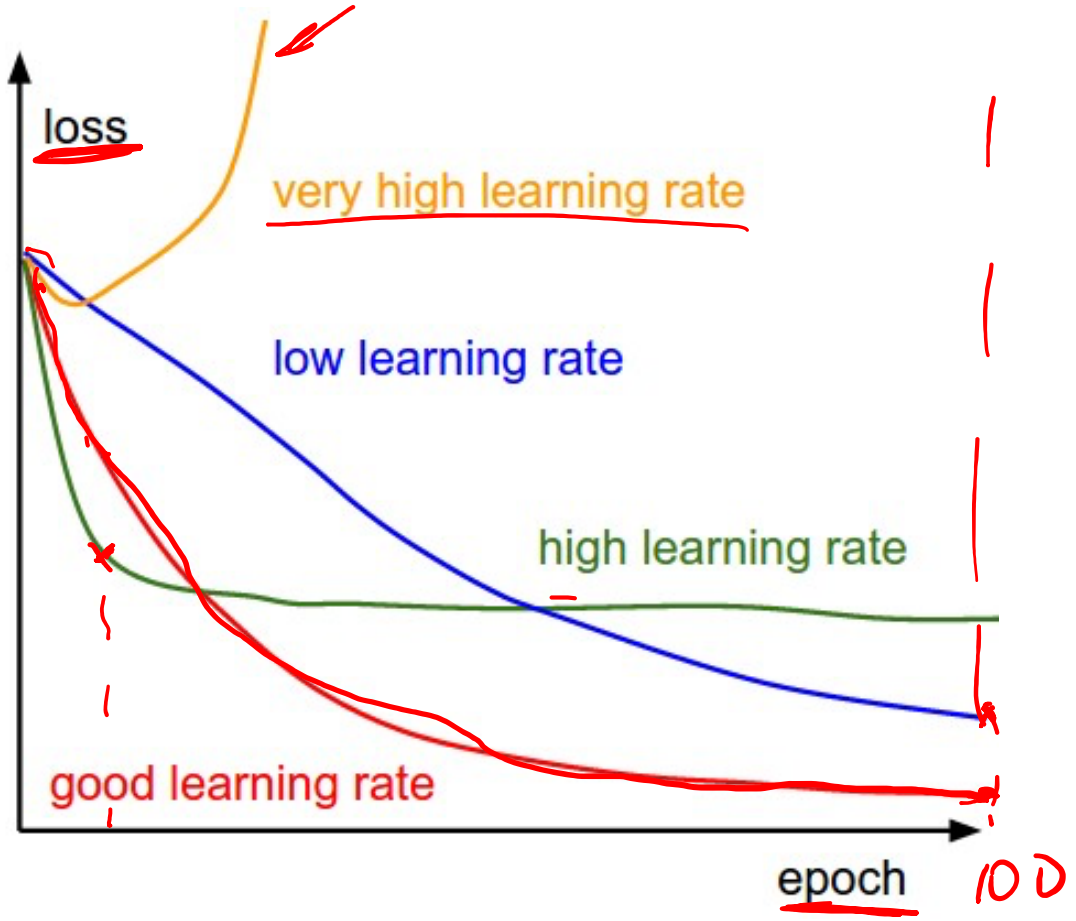
- Gradients could be evaluated numerically
  - When **analytic** expression is not available
  - When it is **faster** to evaluate them numerically
- Use central difference formula

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h}$$

- Check against analytic gradient for several examples

$$\frac{|\, f'_a - f'_n \,|}{\max(|\, f'_a \,|, |\, f'_n \,|)}$$

Convergence



Curriculum Learning

- Training machine learning models with particular order. Starting with easier subtasks and gradually increase the difficulty level of the tasks.
- Both training set and cost functions are updated accordingly

Examples:

- Easy shapes -> harder shapes
- Simple words -> sentences

Curriculum Learning