BOSTON UNIVERSITY

Deep Learning for Data Science
DS 542

https://dl4ds.github.io/fa2025/

Gradient Descent

Original slides by Tom Gardos,
other content from Understanding Deep Learning unless otherwise cited

Announcements

- Re: discussion deadlines are moving to 11:59pm on the day of discussion.
  - Why? The practice is more important than the timing.
  - Still targeting ≤ 30 minutes to do, but more time if you need/want it.

- Shared Compute Cluster (SCC) Tutorial next Monday.
  - Please bring your laptop to class.
  - No graded exercise, but will be walking through account setup.

Plan for Today

- Loss functions for multiclass classification (spillover)
- Example of gradient descent
- Basics of gradient descent
- Gradient descent as a statistical process
- Challenges with gradient descent

Loss Function for Regression

If you recast regression as

1. Predicting the mean of a normal distribution with a fixed variance and
2. Optimize output for maximum likelihood,

Then the optimization is equivalent to optimizing with least squared errors (L_2) as your loss function.

Loss Function for Binary Classification

If you are modeling a binary classification problem,

1. The sigmoid function is handy to map arbitrary "scores" into probabilities, so
2. Your loss function is equivalent to

$$L[\phi] = \sum_i -(1 - y_i)\log[1 \; - \mathrm{sig}[f[x_i|\phi]] - y_i \log[\mathrm{sig}[f[x_i|\phi]]$$

Conceptualizing the Binary Loss Function

last time

$$L[\phi] = \sum_i -(1-y_i)\log[1 - \underbrace{\mathrm{sig}[f[x_i|\phi]]}_{\mathrm{Pr}[y=1\cdots]}] - y_i \log[\underbrace{\mathrm{sig}[f[x_i|\phi]]}]$$

1 − Pr[y=1...]

probability y=0

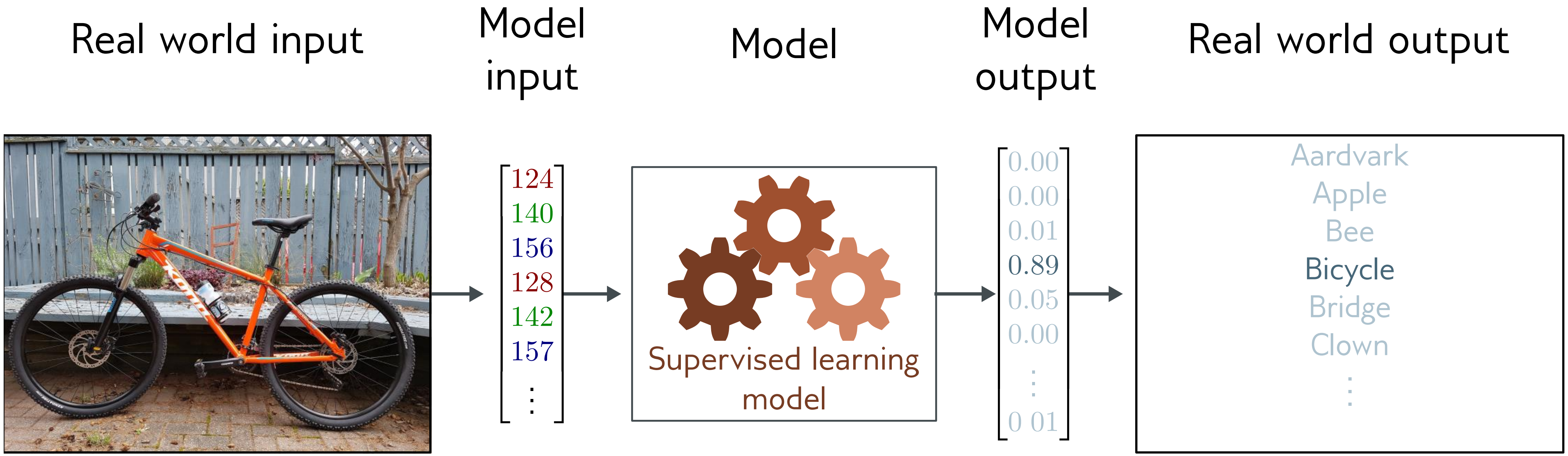
"probability $y_i=1$"

$$L[\phi] = \sum_i \underbrace{-(1-y_i)\log \overbrace{\mathrm{Pr}[y_i = 0|x_i]}^{\text{probability } y=0}}_{\text{keep if } y_i=0} \underbrace{- y_i \log \overbrace{\mathrm{Pr}[y_i = 1|x_i]}^{\text{probability } y_i=1}}_{\text{keep if } y_i=1}$$

keep if $y_i=0$ keep if $y_i=1$

$$L[\phi] = \sum_i -\log \underbrace{\mathrm{Pr}[y_i|\, x_i]}$$

Example 3: multiclass classification



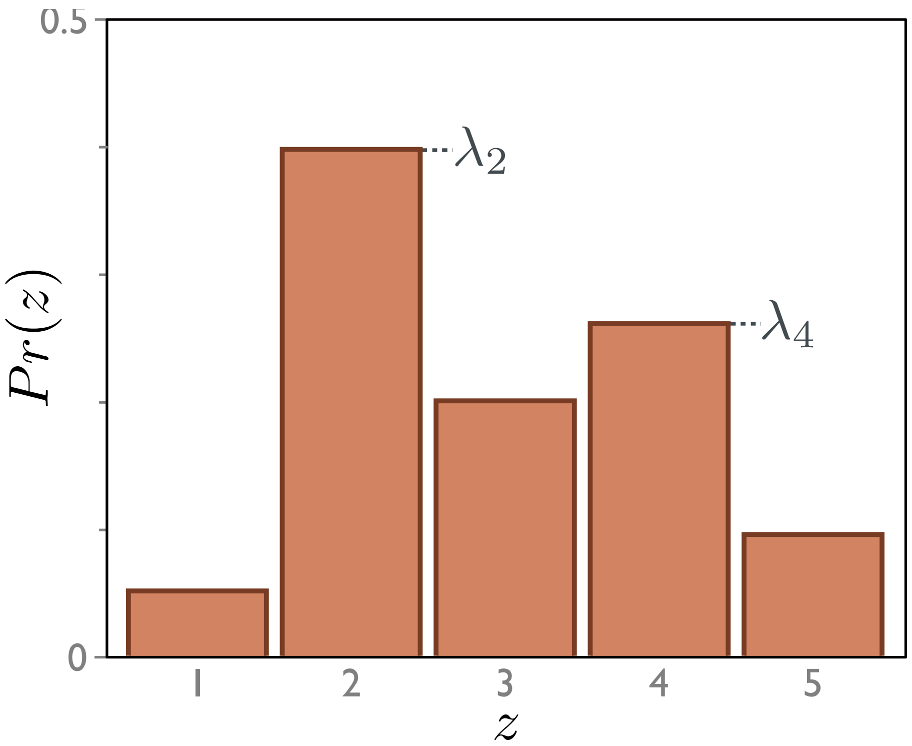
Goal: predict which of K classes $y \in \{1, 2, \ldots, K\}$ the input x belongs to.

Example 3: multiclass classification

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

- Domain: $y \in \{1, 2, \ldots, K\}$
- **Categorical distribution**
- K parameters $\lambda_k \in [0, 1]$
- $\sum_k \lambda_k = 1$

$$Pr(y = k) = \lambda_k$$



Example 3: multiclass classification

2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

Problem:

- Output of neural network can be anything ← same problem as w/ binary case
- Parameters $\lambda_k \in [0,1]$, sum to one

z_k is "score" for class k

$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

Solution:

- Pass through function that maps "anything" to [0,1] and sums to one

$z_k \rightarrow \exp[z_k] = e^{z_k}$

now it is positive.

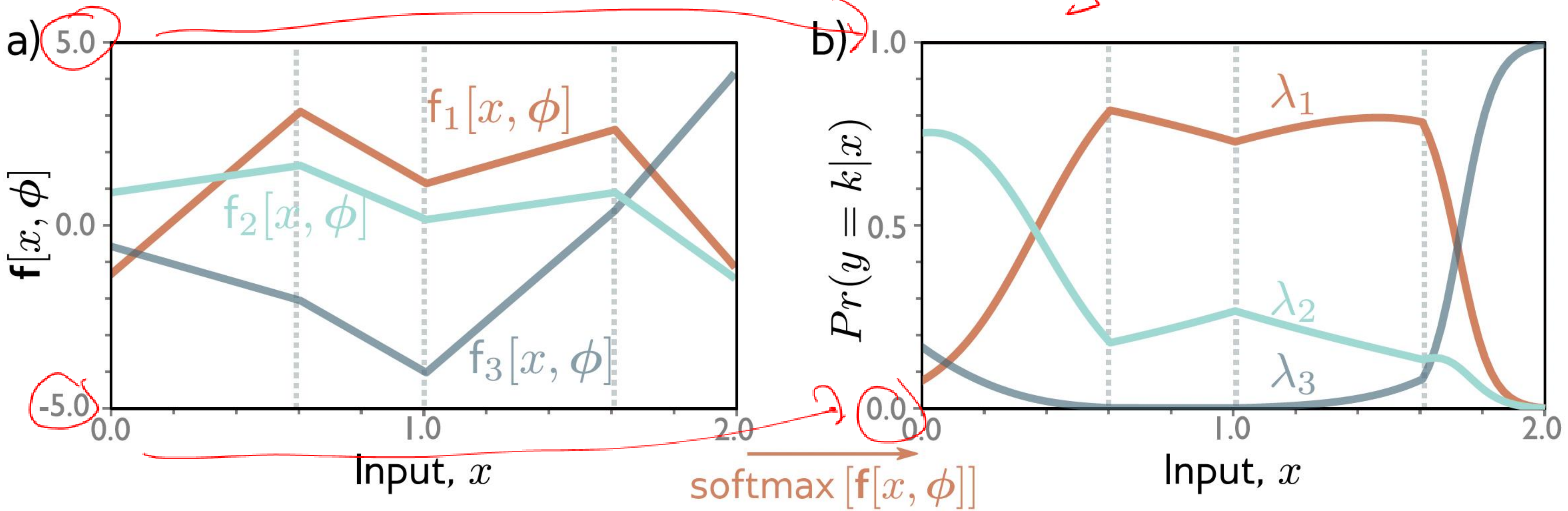
always [0,1]

$\text{sum}_k(\text{soft}\cdots) = 1$

$$\lambda_k \approx Pr(y = k|\mathbf{x}) \approx \text{softmax}_k[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]]$$

Example 3: multiclass classification

[-5,5] [0,1]



$$Pr(y = k|\mathbf{x}) = \text{softmax}_k[\mathbf{f}[\mathbf{x}, \phi]]$$

Example 3: multiclass classification

3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

template

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}\left[L[\phi]\right] = \underset{\phi}{\operatorname{argmin}}\left[-\sum_{i=1}^{I}\log\left[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\right]\right]. \tag{5.7}$$

$$L[\phi] = -\sum_{i=1}^{I}\log\left[\text{softmax}_{y_i}\left[\mathbf{f}\left[\mathbf{x}_i,\phi\right]\right]\right]$$

$$= -\sum_{i=1}^{I}\underbrace{\text{f}_{y_i}\left[\mathbf{x}_i,\phi\right]}_{\text{log softmax numerator}} - \underbrace{\log\left[\sum_{k=1}^{K}\exp\left[\text{ f}_k\left[\mathbf{x}_i,\phi\right]\right]\right]}_{\text{log softmax Denominator}}$$

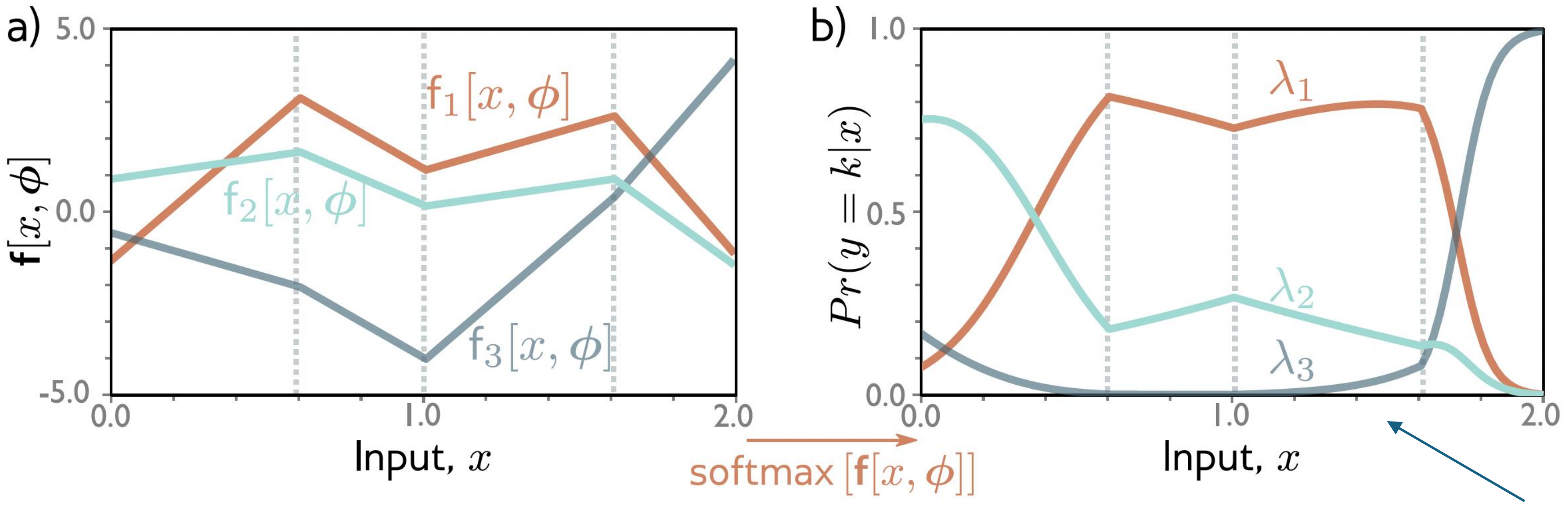
$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K}\exp[z_{k'}]}$$

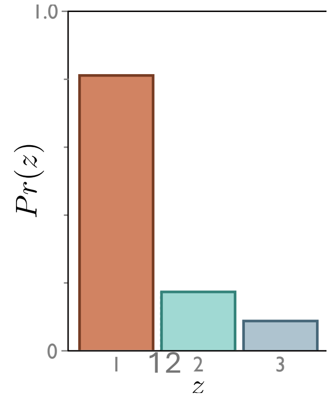
substitute + algebra

Multiclass cross-entropy loss

Example 3: multiclass classification

4. To perform inference for a new test example $\mathbf{x}$, return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\phi}])$ or the maximum of this distribution.





Choose the class with the largest probability
We also get probability or “confidence”

Any questions?

Multiple outputs

- Treat each output y_d as *independent*:

$$Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}_i,\phi]) = \prod_d Pr(y_d|\mathbf{f}_d[\mathbf{x}_i,\phi])$$

where $\mathbf{f}_d[\mathbf{x},\phi]$ is the d^{th} set of network outputs

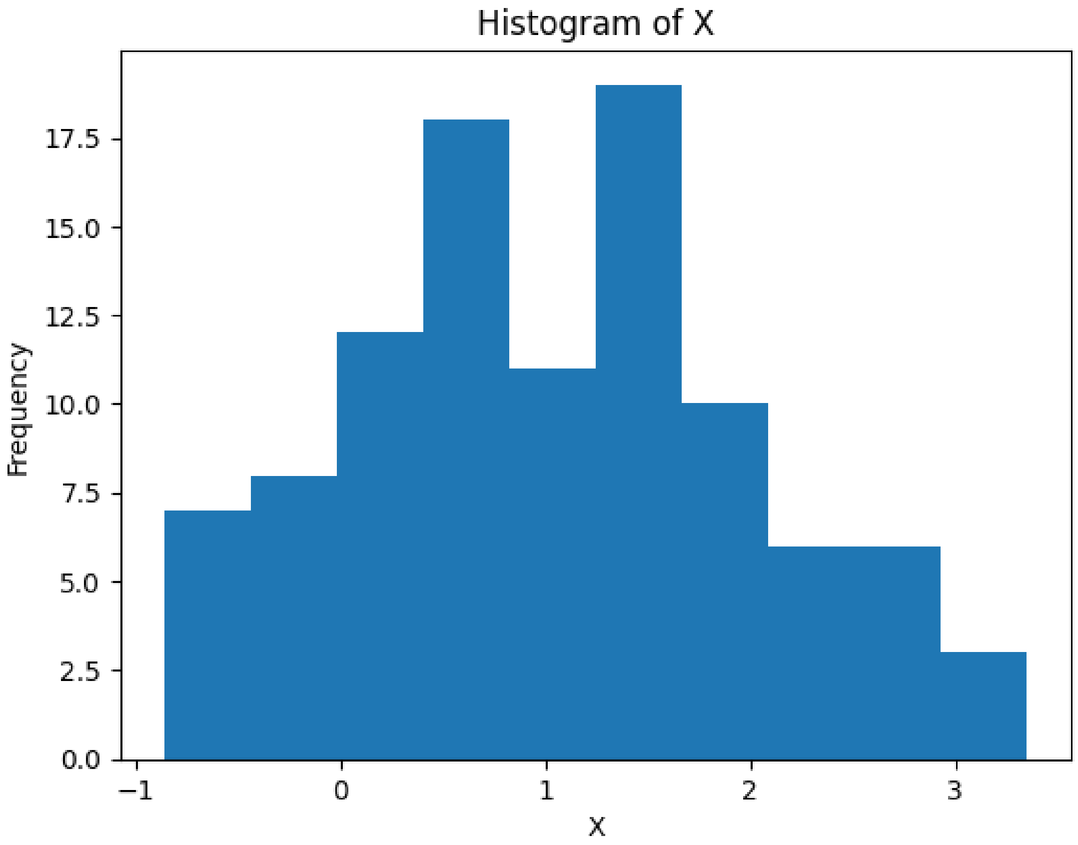
- Negative log likelihood becomes sum of terms:

$$L[\phi] = -\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}_i,\phi])\Big] = -\sum_{i=1}^{I}\sum_d \log\Big[Pr(\underbrace{y_{id}}_{d^{th}\text{output of the } i^{th} \text{ training example}}|\mathbf{f}_d[\mathbf{x}_i,\phi])\Big]$$

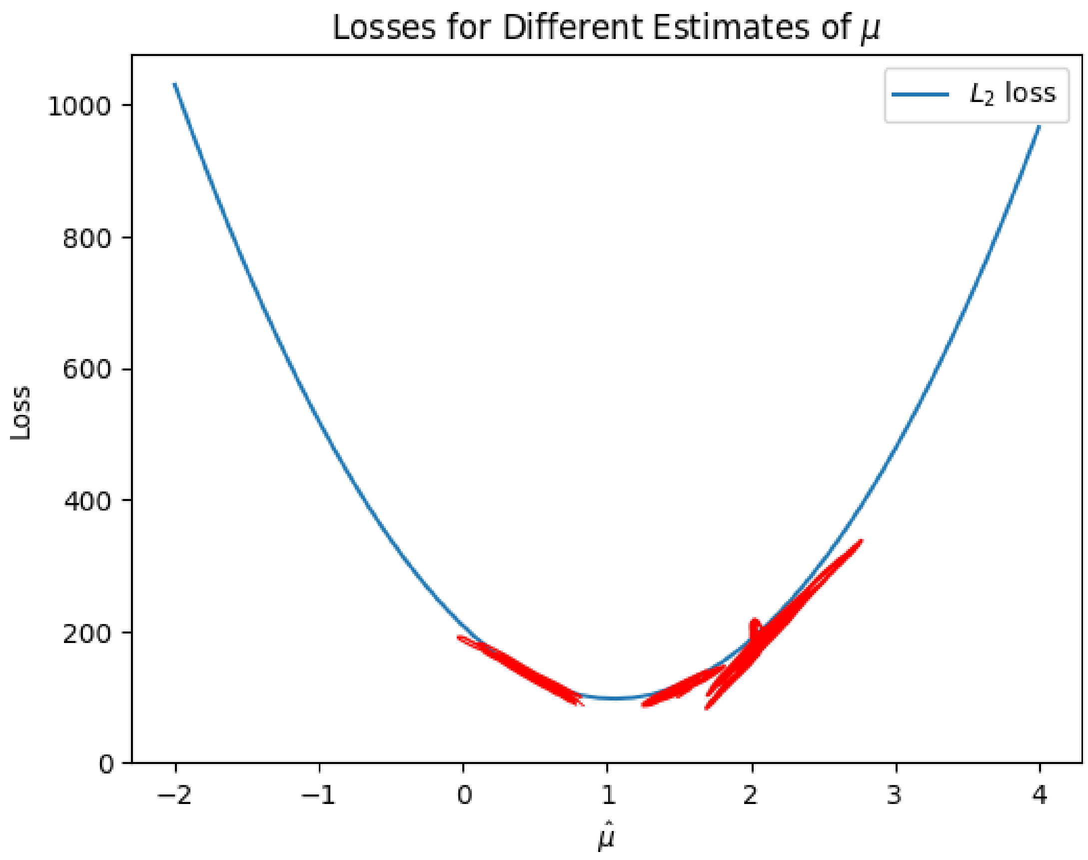
Any questions?

An Example of Gradient Descent

- X is 100 samples from a normal distribution.
  - What were the parameters of that normal distribution? μ, σ^2
  - What was the mean of that normal distribution?



Visualizing Gradient Descent



Basics of Gradient Descent

t for current time

- Given current set of parameters ϕ_t,

derivative w/ ϕ_i pretending all other parameter are constant

  - Calculate all partial derivatives $\frac{\partial L[\phi]}{\partial \phi_i}$ based on current parameters ϕ_t.
  - The vector of these $\frac{\partial L[\phi]}{\partial \phi_i}$ is the gradient of the loss function $\nabla L[\phi]$.

vector of partial derivatives in same order as ϕ

  - Update $\phi_{t+1} = \phi_t - \underbrace{\alpha \, \nabla L[\phi]}$

    where α is the learning rate.

$\alpha > 0$

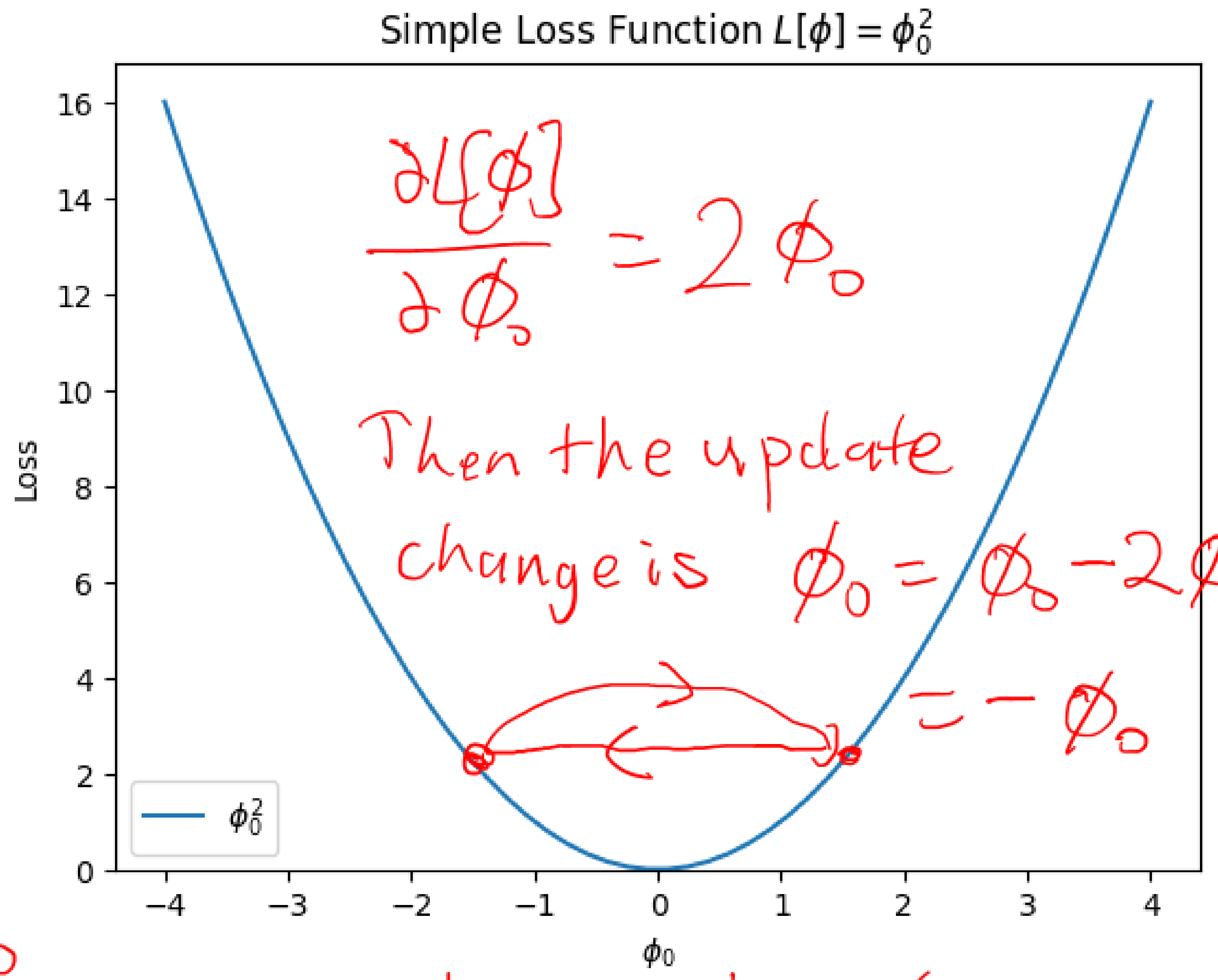
move in opposite direction, and proportionally to gradient

Uninformed α choice $= 0.001$

What should the learning rate be?

- Try $\alpha = 1.$

- Too small, and it takes many steps to get close.
- Too big, and it overshoots.



$$\frac{\partial L[\phi]}{\partial \phi_0} = 2\phi_0$$

Then the update change is $\phi_0 = \phi_0 - 2\phi_0$

$= -\phi_0$

$\alpha = 2 \Rightarrow \phi_0 = \phi_0 - 4\phi_0$

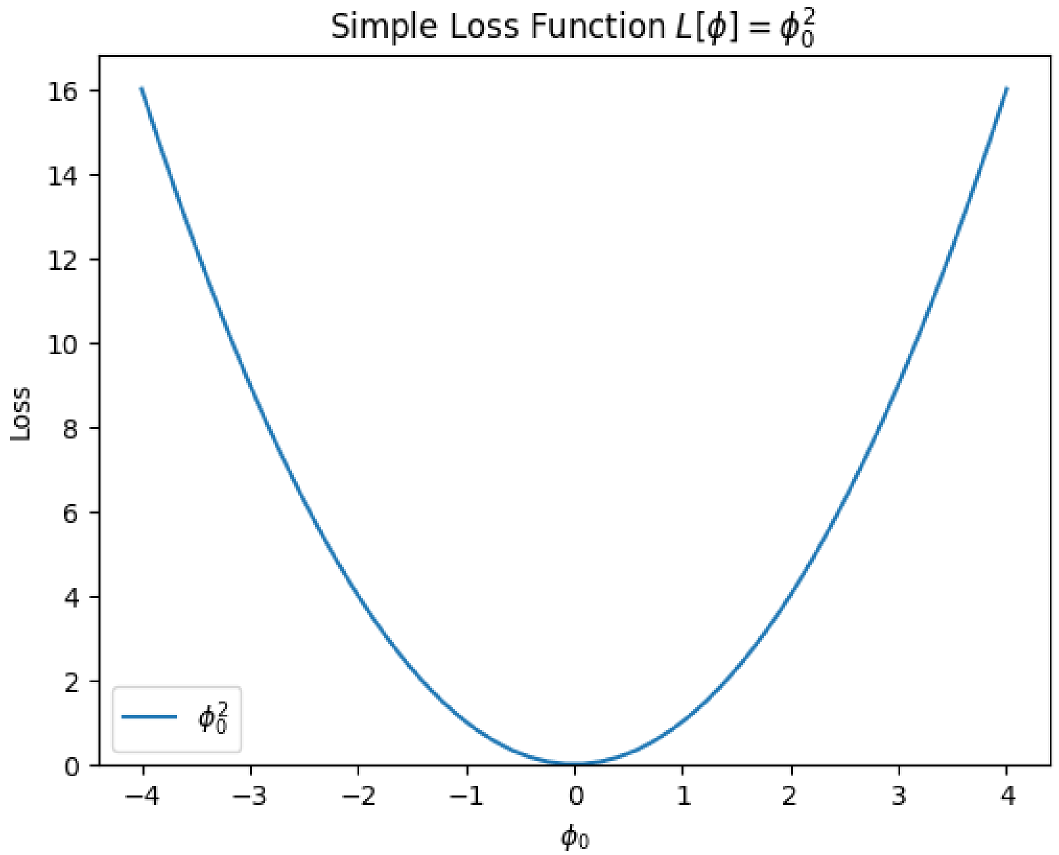
$= -3\phi_0$

$\alpha = \frac{1}{2} \Rightarrow \phi_0 = \phi_0 - \frac{1}{2} \cdot 2 \cdot \phi_0 = 0$

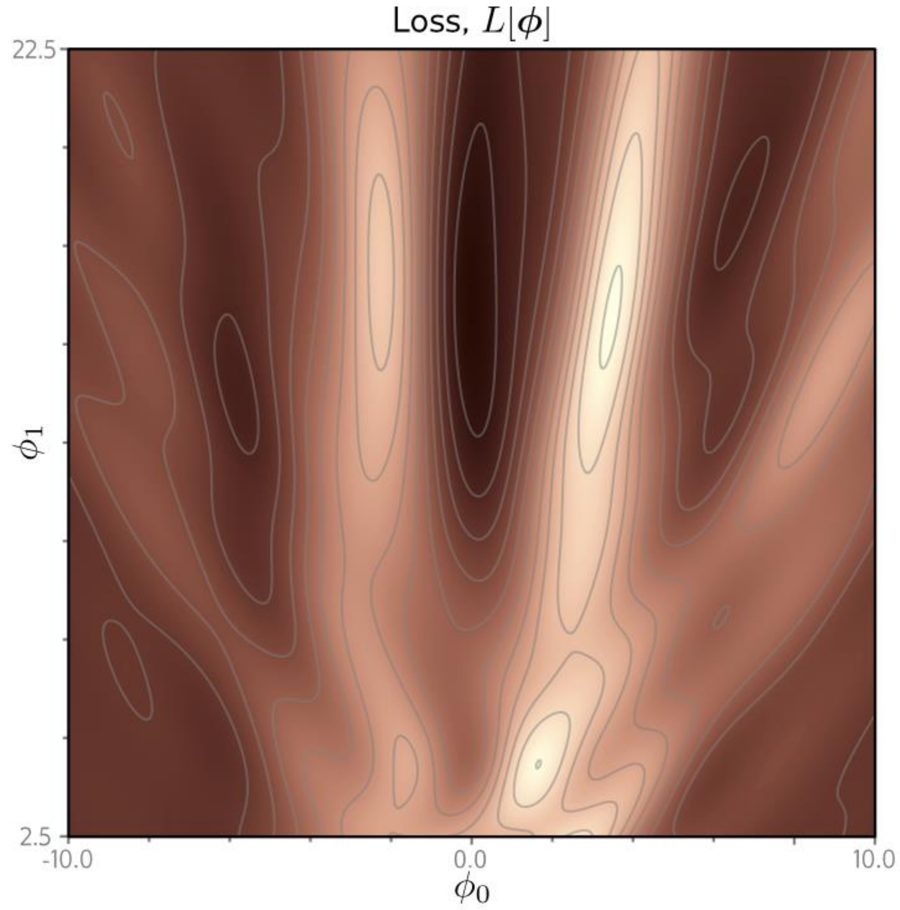
$\alpha = 1/10 \Rightarrow \phi_0 = \phi_0 - \frac{1}{5}\phi_0 = 4/5\,\phi_0$

Convex Loss Functions

- Generally, a lot easier to optimize…
- With gradient descent, main issue is not overshooting minimum too much.



Non-Convex Loss Functions



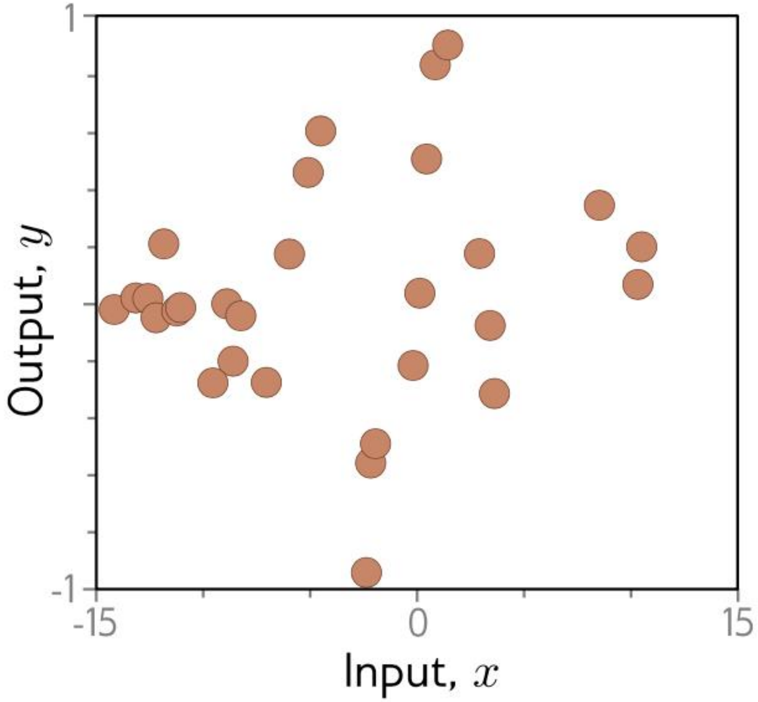
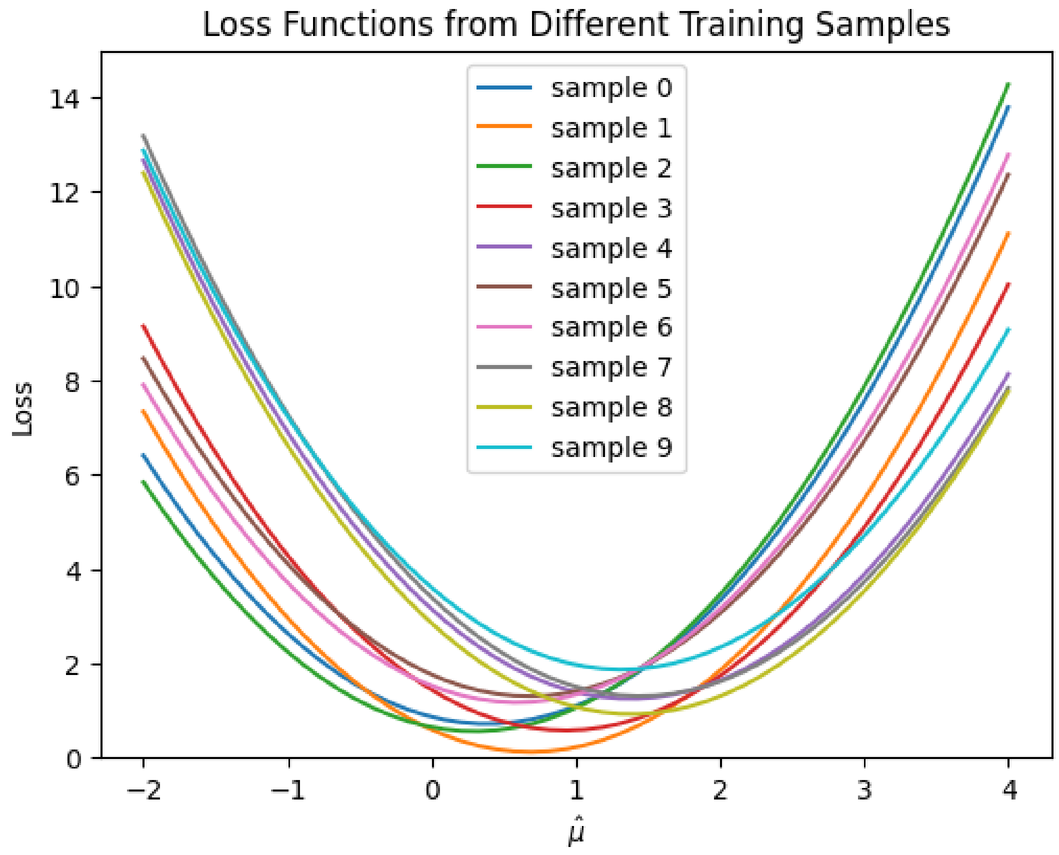


Image Source: Understanding Deep Learning, via https://udlbook.github.io/udlfigures/

Any questions?

Gradient Descent as a Statistical Process

- Our training data is a sample of the whole population.
  - Different training samples yield different training loss functions.



Loss Functions for Different Training Samples

- If we collect different training data sets, will we get different models?

Different training data $\Rightarrow$ different loss function

$\Rightarrow$ different model

Loss Functions for Samples of the Training Set

- If we sample the training data, will we get different models?

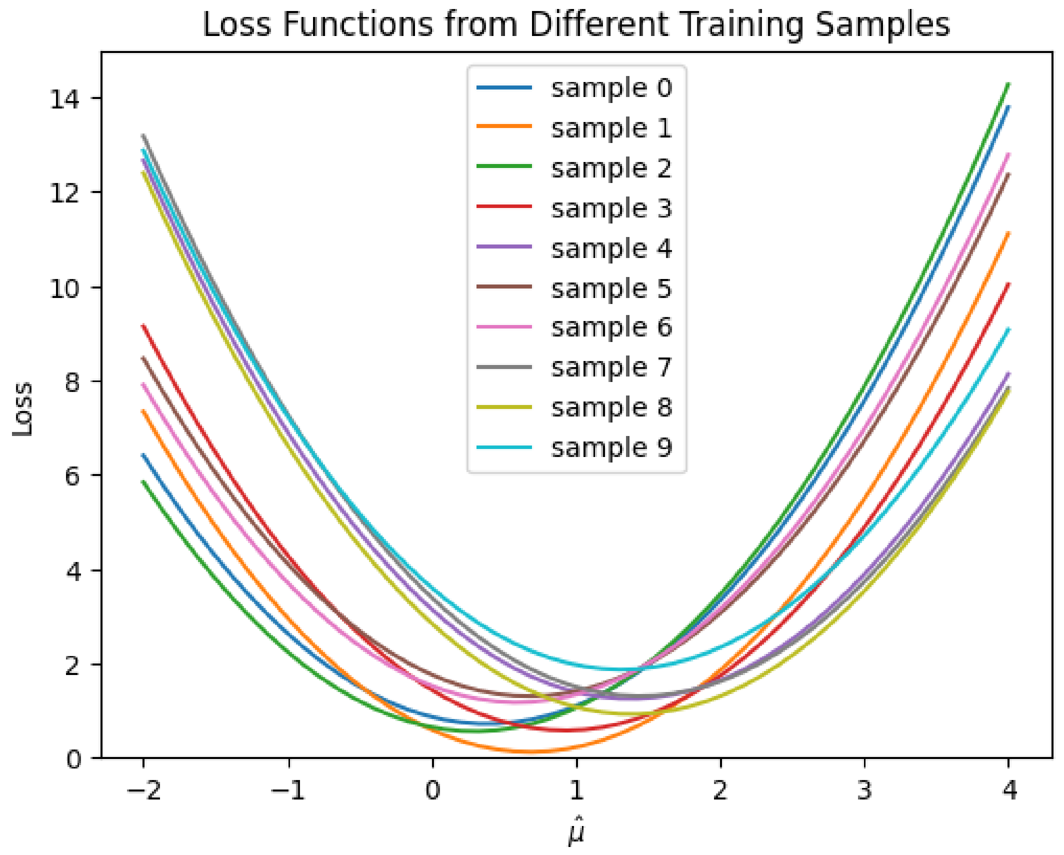
different samples of training data

$\Rightarrow$ different lossfunction

Comparing Models with Different Training Samples

- How far apart are the models of these samples?

- Where do they agree and disagree?

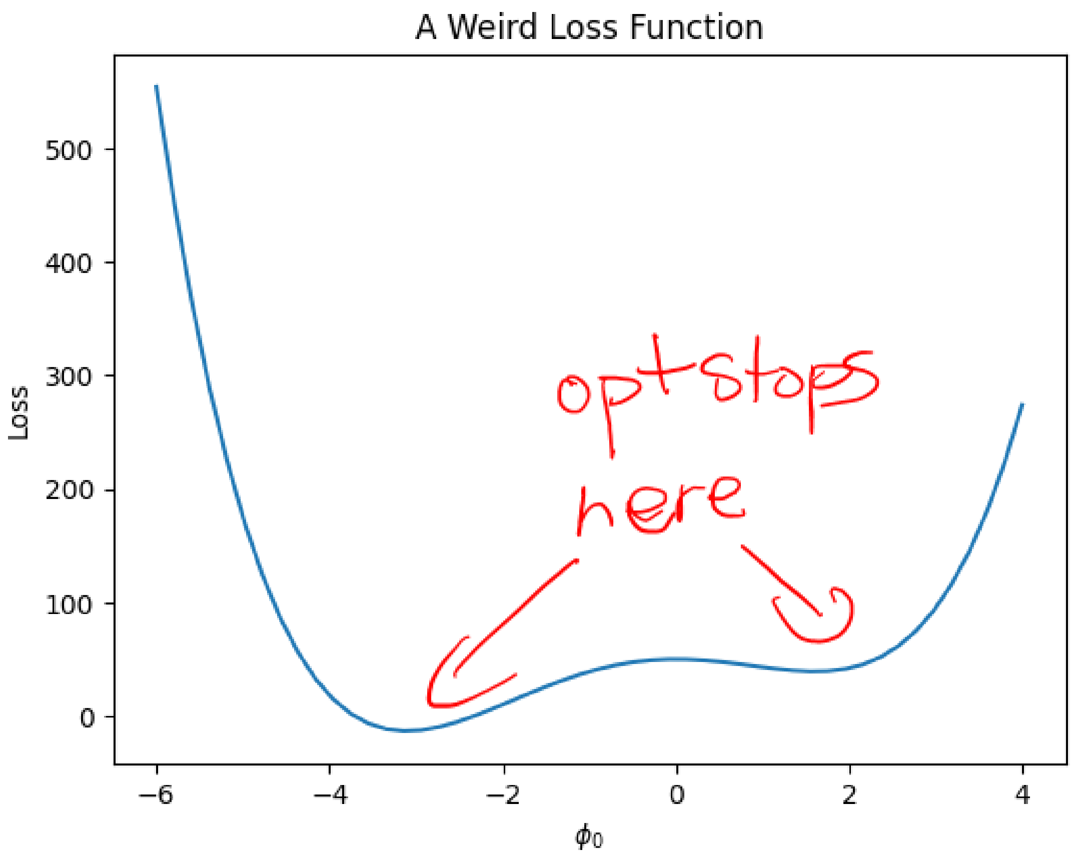


agree →

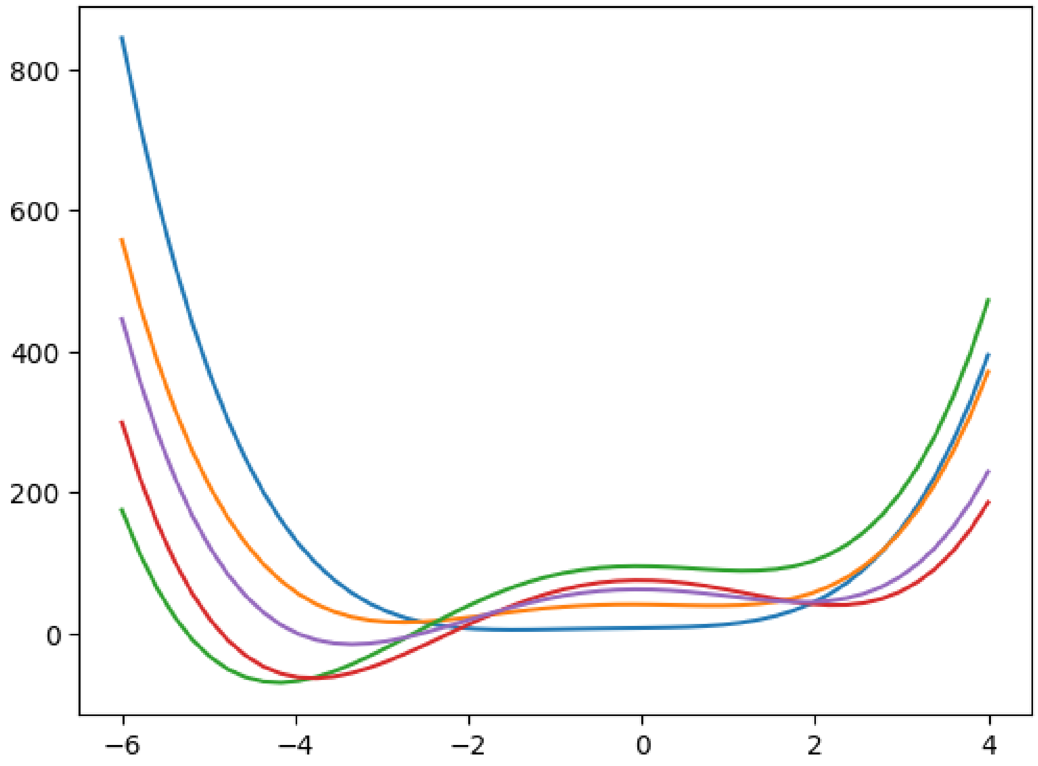
disagree on exact mean

agree ←

A Weird Loss Function



Local Minima vs Samples of the Training Set

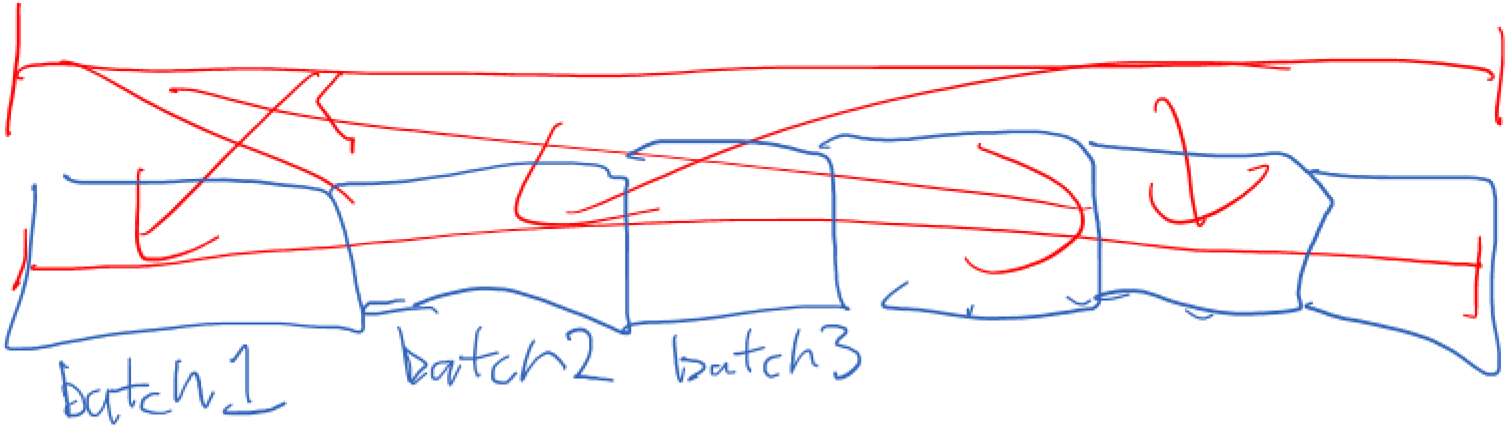


Stochastic Gradient Descent

Idea: Run gradient descent with "mini batches" instead of the full training set.

- E.g. pick a random partition of data into 10 equal-sized batches.
- One epoch = running through all the data once.
  - Vanilla gradient → one parameter update.
  - Stochastic gradient descent → one parameter update per mini batch.

training data



random permutation

For each batch, compute loss + update once.

Variation in Sampled Gradients

- Expected mini batch gradient = whole training set gradient.
  - On average, they agree.
  - But with noise from sampling. ← want batch big enough to reduce but not eliminate noise.

Practical advice: if too much data for 1 GPU, pick batchsize just fitting.

- But remember, just taking one step with each mini batch.
  - Not optimizing to mini batch minimum loss.

Local Minima vs Stochastic Gradient Descent

- When far from a local minima, mini batches tend to agree on gradient direction.

- When close to a local minima, mini batches disagree more.
  - Sampling noise.
  - Explore the flat area around the minima.

Speed of Stochastic Gradient Descent

- How fast is this compared to vanilla gradient descent?

10 mini batches $\Rightarrow$ 10 parameter updates
in time for 1 full batch

Any questions?

Gradient Descent as a Universal Algorithm

- What's the catch?

Local minima (SGD helps)

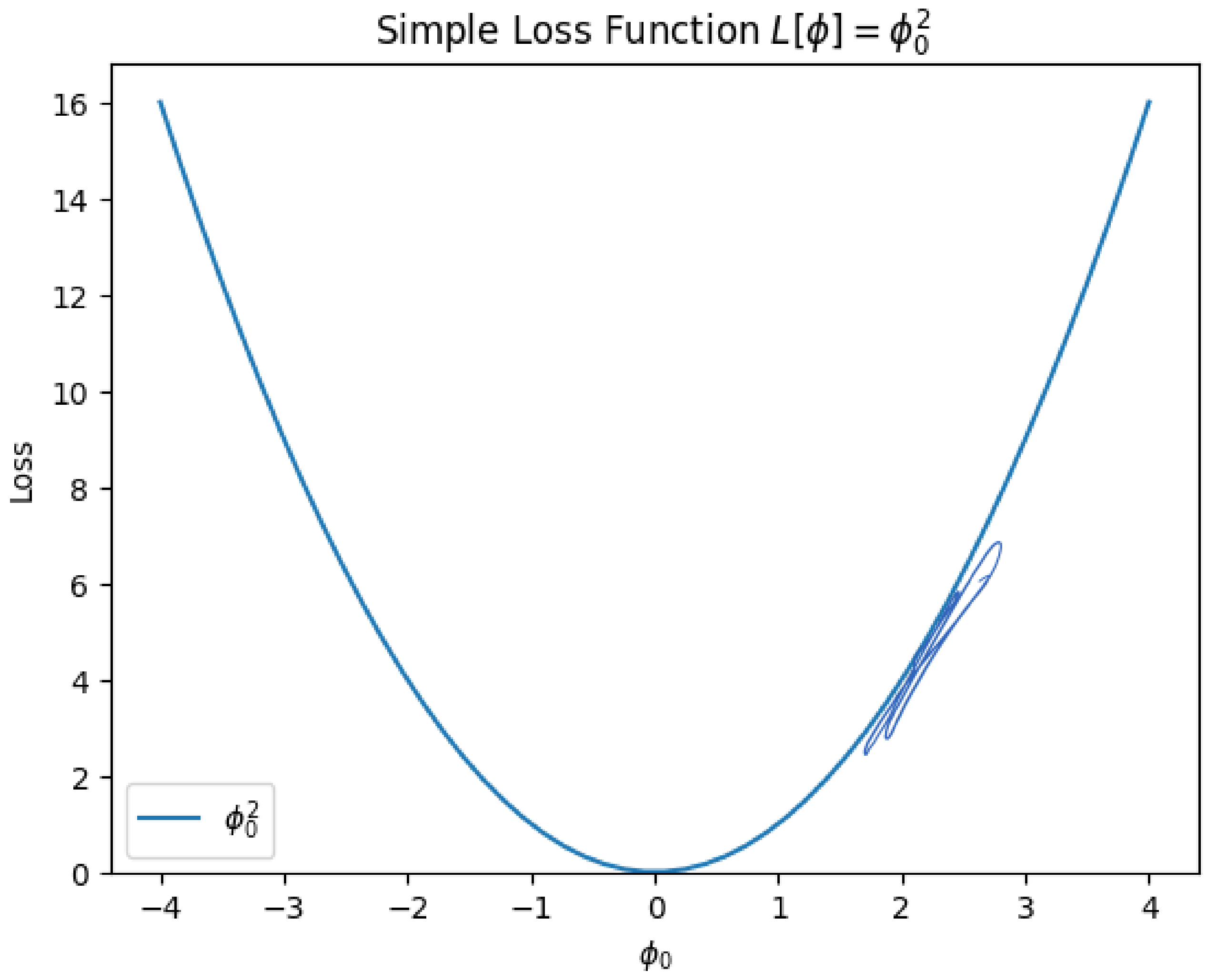
Loss must be differentiable.

Learning rate tuning /scheduling.

How do we pick Learning Rate?

- Remember, $\alpha = 1$ gives an infinite loop.

- Also, be impatient.



Really Bad Linear Regression

- $f(x) = f_1\left(f_2\left(f_3\left(f_4(x)\right)\right)\right)$

- $f_1(x) = a_1 x + b_1$
- $f_2(x) = a_2 x + b_2$
- $f_3(x) = a_3 x + b_3$
- $f_4(x) = a_4 x + b_4$

8 parameters, only need 2

- $f(x)$ is just a linear function? ✓

Really Bad Linear Regression (part 2)

- $f(x) = f_1\left(f_2\left(f_3(f_4(x))\right)\right)$

- $f_1(x) = a_1 x + b_1$
- $f_2(x) = a_2 x + b_2$
- $f_3(x) = a_3 x + b_3$
- $f_4(x) = a_4 x + b_4$

- $f(x)$ is just a linear function?

- Initialize all parameters to zero.

- What are the gradients?

all zero ~~s~~

except $\frac{\partial f(\cdots)}{\partial b_1}$

can only optimize b_1

Really Bad Linear Regression (part 3)

- $f(x) = f_1\Big(f_2\big(f_3(f_4(x))\big)\Big)$

- $f_1(x) = a_1 x + b_1$
- $f_2(x) = a_2 x + b_2$
- $f_3(x) = a_3 x + b_3$
- $f_4(x) = a_4 x + b_4$

- $f(x)$ is just a linear function?

- Initialize all parameters to 100.

- What are the gradients?

Probably big numbers

$f_4(x) = 100x + 100$

$f_3(f_4(x)) = 10000x + 10000\cancel{0} + 100$

$f_2(f_3(f_4(x))) = 1000000x + 1000000 + 100000 + 100$

Really Bad Linear Regression (part 4)

- $f(x) = f_1\left(f_2\left(f_3(f_4(x))\right)\right)$

- $f_1(x) = a_1 x + b_1$
- $f_2(x) = a_2 x + b_2$
- $f_3(x) = a_3 x + b_3$
- $f_4(x) = a_4 x + b_4$

- $f(x)$ is just a linear function?

- We will see both these problems with neural networks if we use the wrong initialization.

Any questions?

Shallow Neural Networks

SCC

Deep NN

Backprop + Init