Neural Network Training With Gradient Descent

Course 3, Module 3, Lesson 3

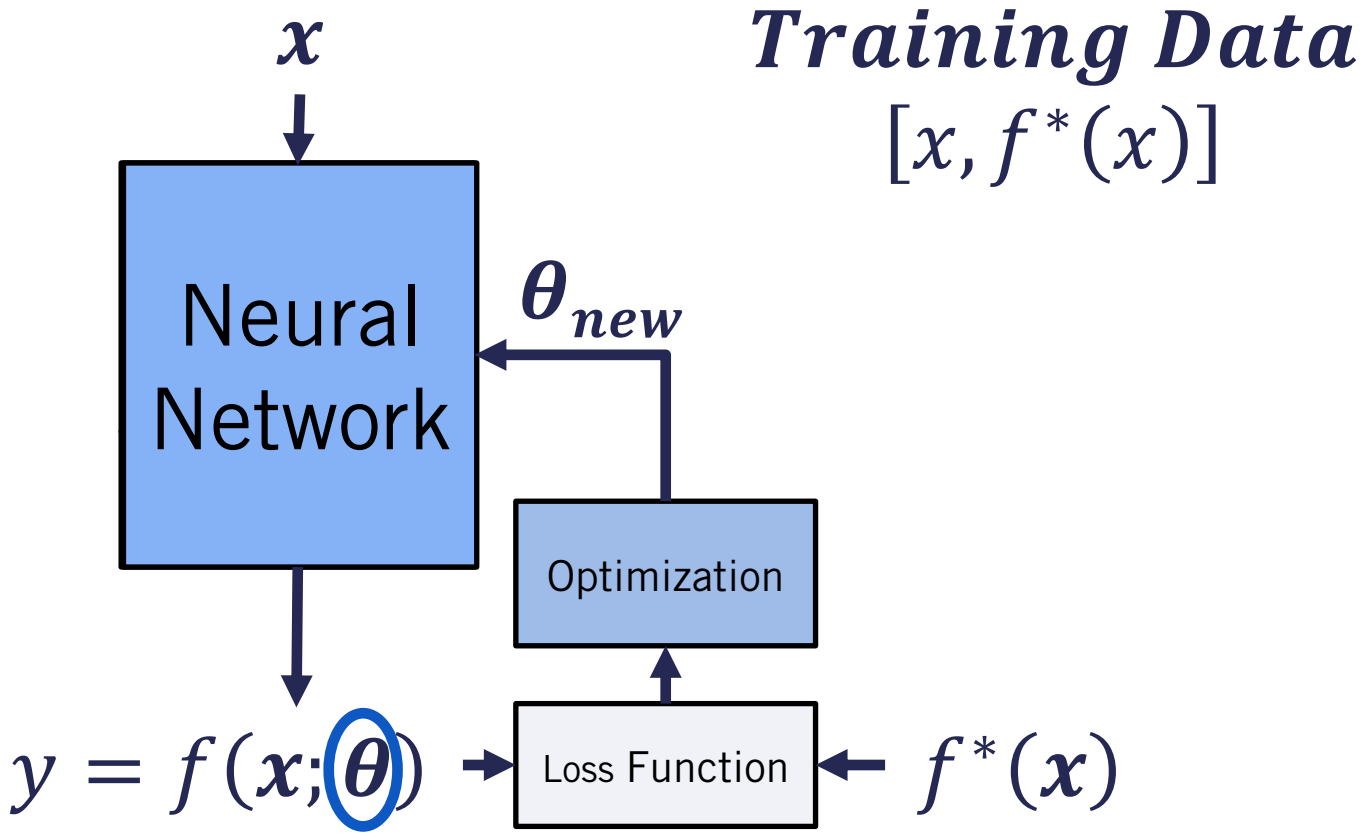
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Learning Objectives

- Learn how to train a neural network using the iterative optimization algorithm: gradient descent

- Learn how to initialize parameters at the start of the optimization process

Artificial Neural Networks



Neural Network Loss Functions

- **Thousands** of training example pairs $[\boldsymbol{x}, f^*(\boldsymbol{x})]$
- The **Loss function** computed over all **N** training examples is termed the **Training Loss** and can be written as:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L[f(x_i, \theta), f^*(x_i)]$$

- The gradient of the training loss with respect to the parameters $\boldsymbol{\theta}$ can be written as:

$$\nabla_\theta J(\theta) = \nabla_\theta \left[\frac{1}{N}\sum_{i=1}^{N} L[f(x_i, \theta), f^*(x_i)]\right] = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L[f(x_i, \theta), f^*(x_i)]$$

Batch Gradient Descent

- Batch Gradient Descent is an **iterative first order** optimization procedure

first order derivative

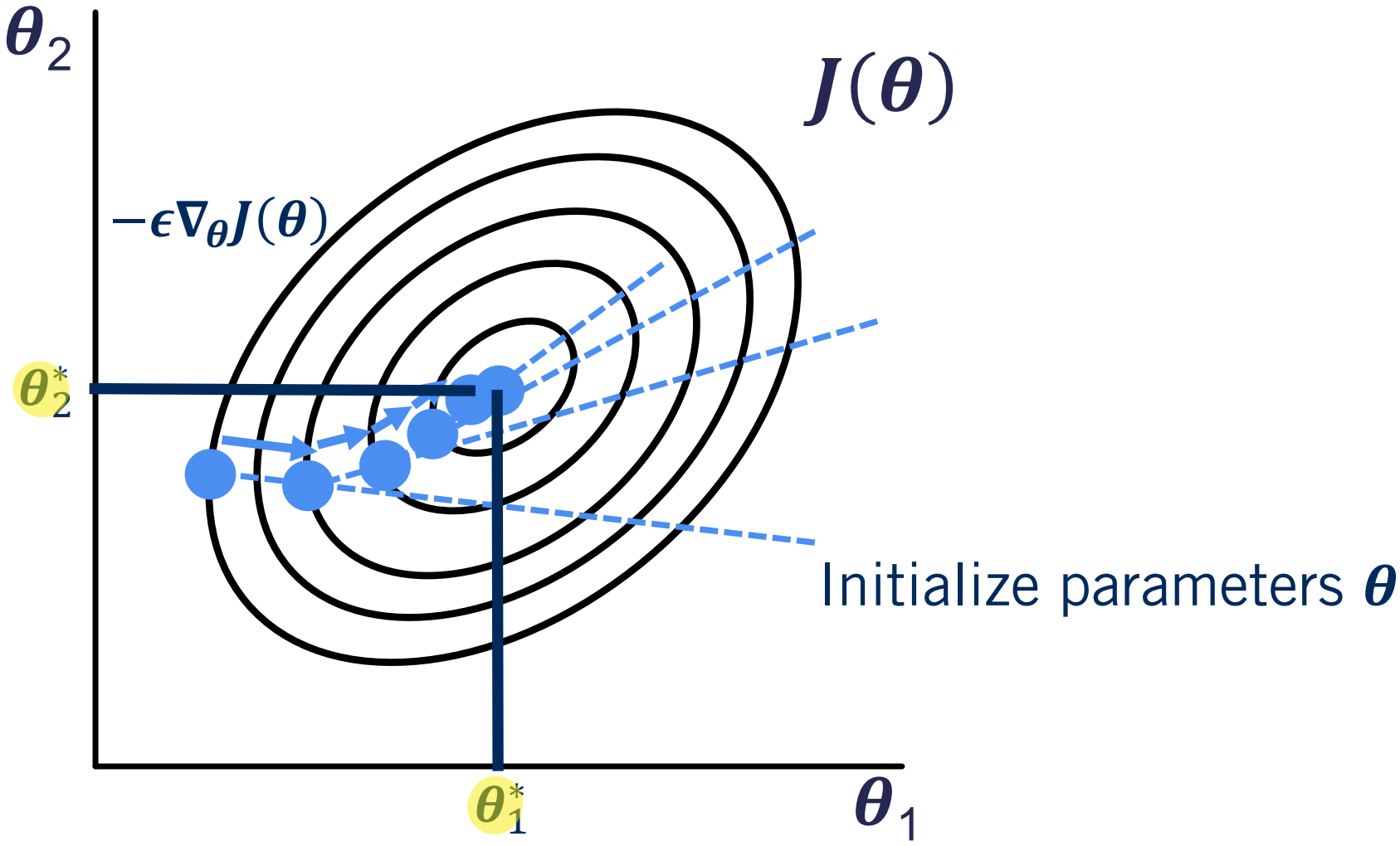
- **Batch Gradient Descent Algorithm:**
  - Initialize parameters $\boldsymbol{\theta}$
  - While **Stopping Condition** is **Not Met:**
    - Compute gradient of loss function over **all** training examples:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \, L[f(x_i, \theta), f^*(x_i)]$$

    - Update parameters according to:

$$\theta \leftarrow \theta - \epsilon \nabla_\theta J(\theta)$$

Batch Gradient Descent



Parameter Initialization and Stopping Conditions

- **Parameter Initialization:**
  - **Weights:** initialized by randomly sampling from a standard normal distribution
  - **Biases**: initialized to 0
  - Other **heuristics** exist

- **Stopping Conditions:**
  - **Number of iterations:** How many training iterations the neural network has performed
  - **Change In $\boldsymbol{\theta}$ value:** Stop if $\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old} <$ Threshold
  - **Change In $\boldsymbol{J(\theta)}$ value:** Stop if $J(\boldsymbol{\theta}_{new}) - J(\boldsymbol{\theta}_{old}) <$ Threshold

Batch Gradient Descent

- **Backpropagation** used to compute $\nabla_{\theta} J(\theta)$ is very expensive to compute over the whole training dataset.

  $\nabla_{\theta} J(\theta) = \frac{1}{N} \sum_{i}^{N} \nabla_{\theta} L\left[f_i(x, \theta), f_i^*(x)\right]$ is a mean!
- Standard error of the mean estimated from N samples is $\frac{\sigma}{\sqrt{N}}$, where σ is the standard deviation of the value of the samples
- Using all samples to estimate the gradient results in **less than linear** return in accuracy of this estimate
- Use a **small subsample (Minibatch)** of the training data to estimate the gradient!

rate of decrease in error is less than linear in the # of samples

Stochastic (minibatch) Gradient Descent

- **Stochastic (minibatch) Gradient Descent Algorithm:**
  - Initialize parameters $\boldsymbol{\theta}$
  - While **Stopping Condition** is **False**:
    1. Sample a preset number N' of examples (minibatch) from the training data
    2. Compute gradient of Loss Function Over **all** training examples:

$$\nabla_\theta J(\theta) = \frac{1}{N'} \sum_{i}^{N'} \nabla_\theta L\left[f_i(x,\theta), f_i^*(x)\right]$$

    3. Update parameters according to:

$$\theta \leftarrow \theta - \epsilon \nabla_\theta J(\theta)$$

What Minibatch Size To Use ?

- GPUs work better with **powers of 2** batch sizes
- **Large batch sizes > 256:** 32~250
  - Hardware underutilized with very small batch sizes.
  - More accurate estimate of the gradient, but with less than linear returns
- **Small batch size < 64**
  - Small batches can offer a **regularizing effect**. The best generalization error is often achieved with batch size of 1.
  - Small batch sizes allow for faster convergence, as the algorithm can compute the parameter updates rapidly

- Always make sure dataset is **shuffled** before sampling minibatch

SGD Variations

- Many variations of SGD exist
  - Momentum SGD, Nestrove Momentum SGD
  - Ada-Grad, RMS-Prop
  - ADAM (Adaptive Moment Estimation)

- Which one to use ?
  - ADAM: Implemented in most deep neural network libraries, fairly robust to the choice of the learning rate and other hyperparameters

- http://deeplearningbook.org

Summary

- You can optimize for neural network parameters using batch gradient descent
- When the number of training examples is extremely large, you can use minibatch (stochastic) gradient descent
- State-of-the-art optimization algorithms built on top of SGD are implemented in most neural network libraries

- **Next: Dataset Splits and Neural Network Performance Evaluation**