ECE 449/590 – OOP and Machine Learning
Lecture 18 Stochastic Gradient Descent

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 26, 2022

Outline

Reading Assignment

- This lecture: Deep Learning 4.3, 5.9, 8.1.3, 8.3, 8.4
- Next lecture: Deep Learning 6

Outline

Unconstrained Optimization

- Minimize some function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$.
  - Use $-f(\boldsymbol{x})$ if you would like to maximize $f(\boldsymbol{x})$.
  - $f(\boldsymbol{x})$ is the objective function, a.k.a. criterion, cost function, loss function, and error function.
- The optimal solution $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$.
  - Does $\boldsymbol{x}^*$ exist?
  - How to find it? What do we know/can we compute about f?
  - What if we cannot find it?

Critical Points

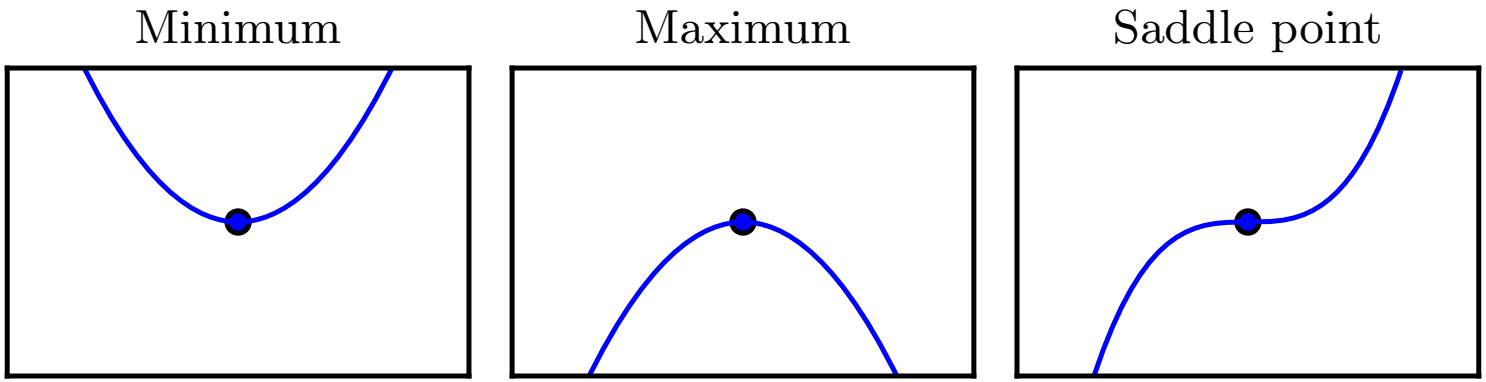


Figure 4.2

(Goodfellow 2017)

- Consider the 1-D case to minimize $y = f(x)$.
- Critical points $\frac{dy}{dx} = f'(x) = 0$.
  - Not necessary a minimum.

Gradient Descent (1-D)

- First order Taylor expansion: $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$
- If $f'(x) < 0$, increase x; if $f'(x) > 0$, decrease x.
  - For small enough ϵ, $f(x)$ will decrease.
- Iterative optimization: apply multiple steps
  - In each step, update x to $x + \epsilon$ by a small enough ϵ.
  - Eventually $f(x)$ is minimized.
- Challenges
  - What if $f'(x) = 0$ (or too small)?
  - How large should ϵ take?
  - How to compute $f'(x)$?
  - What if $f(x)$ has no minimum?

Gradient Descent

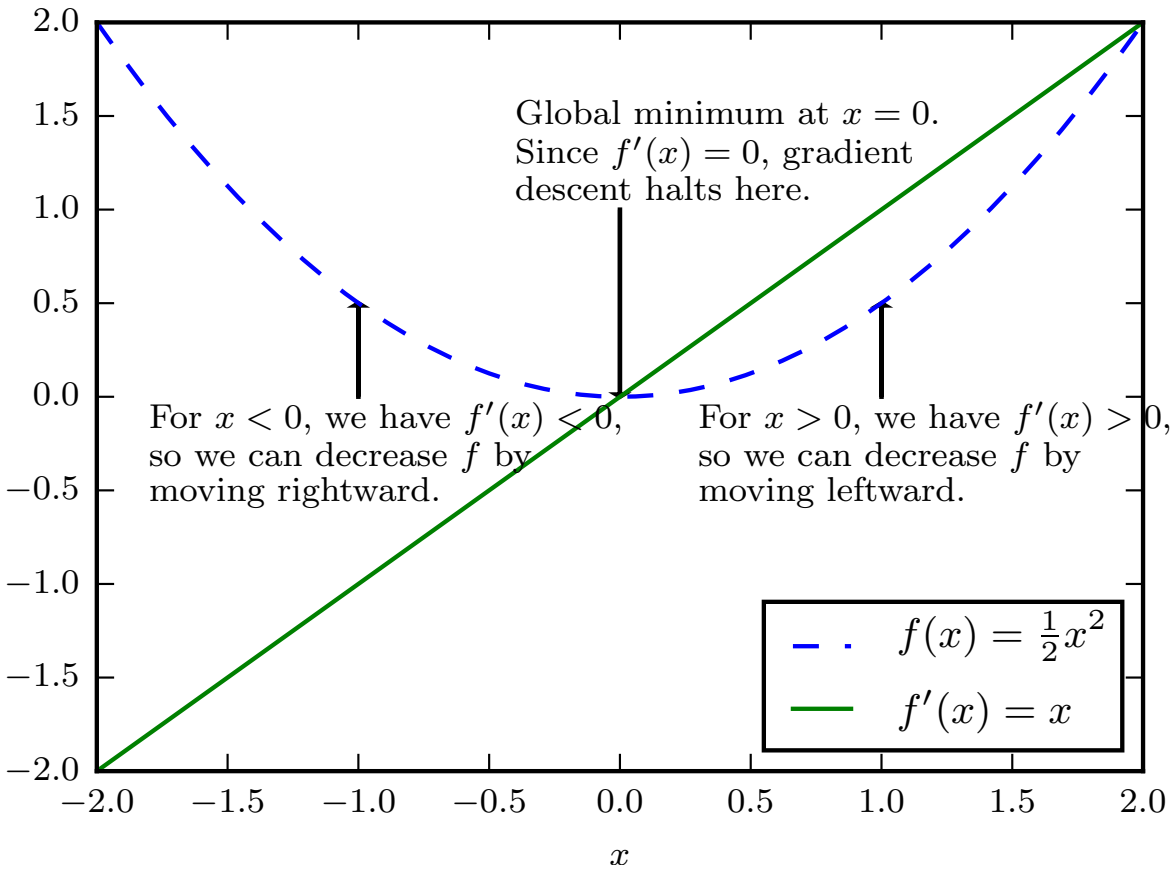


Figure 4.1

Approximate Optimization

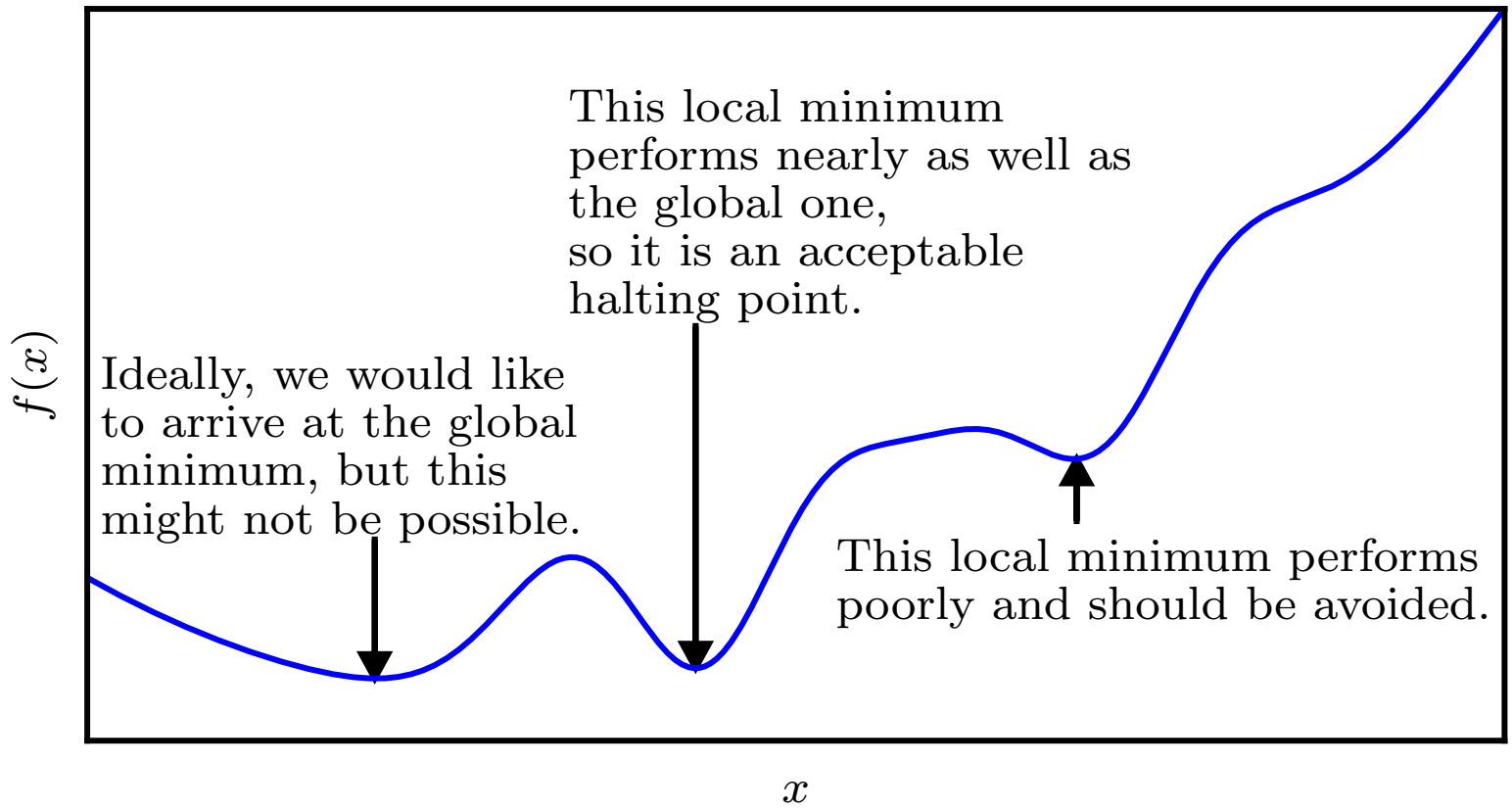


Figure 4.3

Gradient Descent

- First order Taylor expansion:
  $f(\boldsymbol{x} + \delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \delta\boldsymbol{x}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - $f(\boldsymbol{x})$ will decrease for small enough $\delta\boldsymbol{x}$ with $\delta\boldsymbol{x}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) < 0$.
- Let $\delta\boldsymbol{x} = \alpha\boldsymbol{u}$ for scalar α and unit vector $\boldsymbol{u}$.
- Can we minimize $\boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ as a function of $\boldsymbol{u}$?
  - For a small enough fixed α, make $\delta\boldsymbol{x}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ as small as possible in order to make $f(\boldsymbol{x} + \delta\boldsymbol{x})$ as small as possible.
- Gradient descent: $\boldsymbol{u}^* = -\frac{\nabla_{\boldsymbol{x}} f(\boldsymbol{x})}{||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||}$
  - A.k.a. method of steepest descent.
- For machine learning, usually we skip to compute the norm for $\boldsymbol{u}^*$ and choose $\delta\boldsymbol{x} = -\epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.
  - ϵ: learning rate
  - Do we need to find the best ϵ for each step?

Gradient Descent and Poor Conditioning

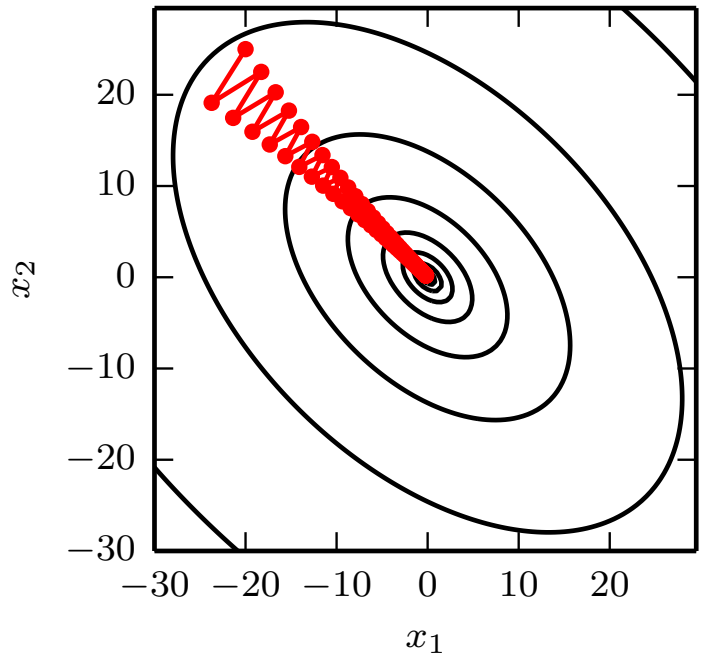


Figure 4.6

(Goodfellow 2017)

- If for each step we could find ϵ^* to minimize $f(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$, it may still take a lot of steps to reach the minimal f.

Beyond Gradient Descent

- Second order Taylor expansion:

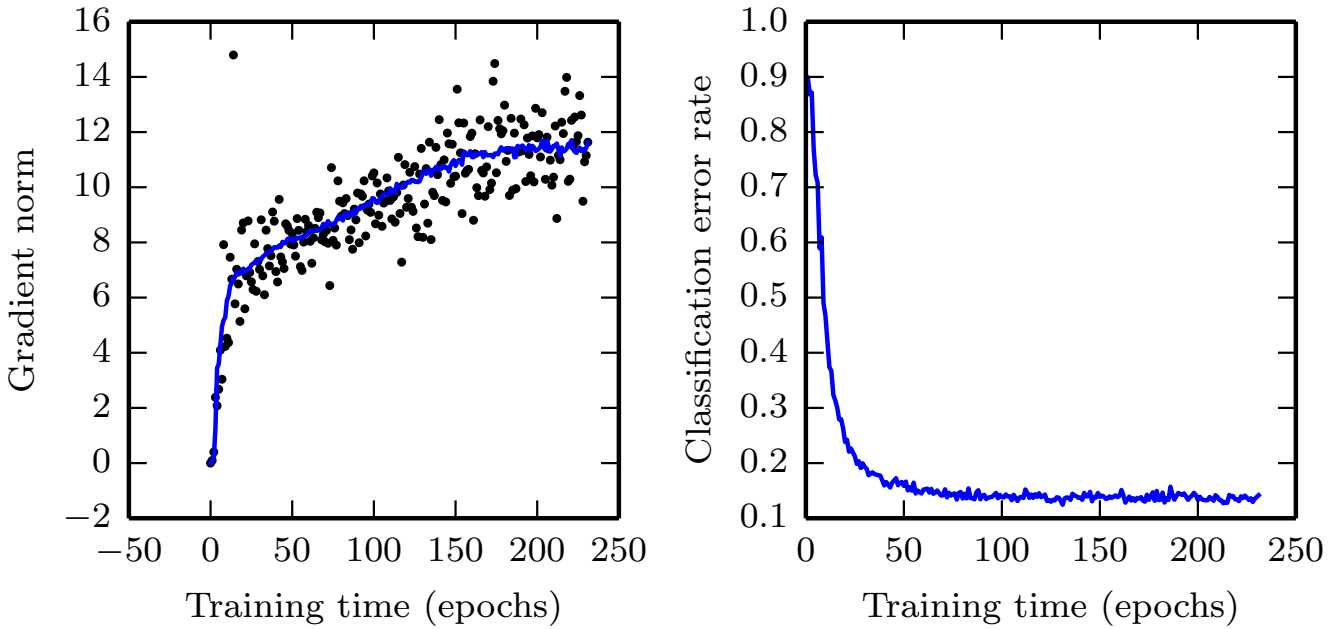
$$f(\boldsymbol{x} + \delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \delta\boldsymbol{x}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \frac{1}{2}\delta\boldsymbol{x}^\top \boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})\delta\boldsymbol{x}$$

  - The Hessian matrix: $\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$
- Best step size for gradient descent $\delta\boldsymbol{x} = -\epsilon^* \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$:

$$\epsilon^* = \frac{\nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})}{\nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top \boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x}) \nabla_{\boldsymbol{x}} f(\boldsymbol{x})}$$

- Newton's method: $\delta\boldsymbol{x}^* = -(\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x}))^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - Allow to move in a different direction than gradient descent.
  - Still, there are issues related with non-minimum critical points and $\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})$ being too close to 0.
- Practical challenges for machine learning
  - How to calculate $\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})$?
  - How to store $\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})$?

We usually don't even reach a local minimum



(Goodfellow 2017)

- For many machine learning problems, we don't even need $f'(x)$ to approach 0.
  - For simplicity, just think about to minimize $f(x) = \log(|x - c|)$

Outline

Loss Function Revisited

- For training, loss function is the average of individual ones.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}} L(\boldsymbol{x}, y; \theta) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

- The gradient can be computed as the average too.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

- But this will be too time consuming for gradient descent.
  - Need to visit every training exmaple for each step.

Stochastic Gradient Descent (SGD)

- Instead of computing the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ from all training examples, we may approximate it by sampling.
  - Indeed, $\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$ is an approximation of the actual loss function $\mathbb{E}_{\boldsymbol{x},y\sim p_{data}} L(\boldsymbol{x}, y; \theta)$.
  - For machine learning, finding the actual minimum is not as important as reducing the loss function.
- Stochastic Gradient Descent (SGD) and Minibatch
  - Sample m' examples $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$ with a fixed $m' << m$ no matter how large m is.
  - $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$
- In practice, training of neural network models are organized into epochs.
  - The epoch begins by randomly shuffling training examples.
  - Then each step consumes m' examples.
  - The epoch ends after $\frac{m}{m'}$ steps when all examples are consumed once.

Discussions on Minibatch Algorithms

- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns.
- Multicore architectures are usually underutilized by extremely small batches.
- Typically, all examples need to be available from the memory so that they could be processed in parallel. This may limit batch size in certain hardware.
- Some kinds of hardware, especially GPUs, achieve better runtime with specific sizes of arrays, e.g. powers of 2.
- Small batches introducing noises to the training process may benefit learning as a whole because they may work as regularization to reduce generalization errors.

The Learning Rate

- It is common for SGD to use different learning rates ϵ_k for different steps k.
- In theory, for SGD to converge, it is sufficient that

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

- In practice, reduce learning rates as training progresses.
  - Larger rates help to update weights faster in the beginning.
  - Smaller rates help to keep what have already been learned.
  - E.g. linear decay until step τ: $\epsilon_k = (1 - \frac{k}{\tau})\epsilon_0 + \frac{k}{\tau}\epsilon_\tau$

Momentum

- Noisy gradients due to minibatch or poorly conditioned Hessian may cause gradient descent to to follow a zig-zaging path.
  - Lead to slow convergence as steps are cancelling each other.
- Momentum: use an exponentially decaying moving average of past gradients to update weights.
  - Velocity updates: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{m'} \sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}))$
  - Weights updates: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
- The actual step sizes are large to accelerate learning if past gradients are aligned.

Parameter Initialization Strategies

- While SGD takes multiple steps to reduce $J(\boldsymbol{\theta})$, if a good $\boldsymbol{\theta}$ is chosen for the first step, the iterative process may take less steps to converge.
  - In extreme cases, bad initial $\boldsymbol{\theta}$'s may prevent SGD to converge to a reasonable minimum.
- Symmetry in neural network models
  - Many nodes in neural network models have identical inputs and drive the same output.
  - The training process is expected to assign different weights to such nodes so they would learn different features.
  - However, if their weights are the same in the beginning, gradient descent will compute the same gradient for them and they will remain the same through the learning process.
- "Break symmetry": initialize parameters randomly

Summary

- Machine learning introduces many unique challenges to optimization.
- SGD with minibatch works well when training neural network models.
  - Choices of learning rate, momentum, and initialization may still affect the efficiency of the learning process.