IE 398/CS 398 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2018

Lecture 1

"Most of all, there is a shortage of [deep learning] talent, and the big companies are trying to land as much of it as they can. Solving tough A.I. problems is not like building the flavor-of-the-month smartphone app. In the entire world, fewer than 10,000 people have the skills necessary to tackle serious artificial intelligence research [...]"

– New York Times, October 2017.

Topics:

- Fully-connected networks
- Convolution networks
- Residual networks
- Recurrent networks (e.g., LSTM)
- Deep reinforcement learning
- Generative adversarial networks
- Optimization and training methods

PyTorch is a software library for training deep learning models.

- Seamlessly integrated with Python.
- **Define-by-run** framework allows for dynamic training of models.
- Capable of distributing training across multiple machines.

Computational resources:

- 25,000 GPU hours
- Unique opportunity to implement large-scale deep learning models!
- Training of deep learning models can be highly parallelized on GPUs (frequently $10\times$ faster than CPUs).

Homeworks

1. Implement a logistic regression model from scratch in Python for the MNIST dataset (no PyTorch).
2. Implement a neural network from scratch in Python for the MNIST dataset (no PyTorch).
3. Implement a convolution neural network from scratch in Python for the MNIST dataset (no PyTorch).
4. Deep convolution network for MNIST and CIFAR10.
5. Residual neural network for CIFAR100.
6. Image ranking with convolution networks.
7. Natural Language Processing
8. Generative adversarial networks (GANs)
9. Video recognition.

Extensive Code and Notes will be provided, including:

- PyTorch (and TensorFlow) Code
- Documentation/code on using PyTorch on Blue Waters
- Course notes

Grading:

35% Homeworks

35% Midterm

30% Final Project

Course website: https://courses.engr.illinois.edu/ie534/fa2018/

Submit homeworks via Compass.

Start homeworks early!

2 office hours per week.

- Machine learning estimates a statistical model for the relationship between an input X and an output Y.
- Formally, suppose there is data $(X, Y) \in \mathbb{R}^d \times \mathcal{Y}$ and a statistical model $f(x; \theta) : \mathbb{R}^d \to \mathbb{R}^K$.
- $\theta \in \Theta$ are the parameters in the model and must be estimated.
- We wish to find a model $f(x; \theta)$ such that $f(X; \theta)$ is "an accurate prediction" for Y.

$$\mathcal{L}(\theta) = \mathbb{E}_{(X,Y)}\big[\rho(f(X;\theta), Y)\big]. \tag{1}$$

- $\rho(z, y)$ measures the distance between the model prediction z and y.

- This distance is then averaged over the distribution $\mathbb{P}_{(X,Y)}$ of the data (X, Y).

The best model, within the class of models $\{f(x;\theta')\}_{\theta'\in\Theta}$, is the model $f(x;\theta)$ where θ satisfies

$$\theta = \arg\min_{\theta'\in\Theta} \mathcal{L}(\theta'). \tag{2}$$

- Typically, the distribution $\mathbb{P}_{(X,Y)}$ is unknown.
- Instead, i.i.d. data samples $(x^n, y^n)_{n=1}^N$ are available from the distribution $\mathbb{P}_{(X,Y)}$.
- Then, our objective function becomes

$$\mathcal{L}^N(\theta) = \frac{1}{N}\sum_{n=1}^{N} \rho(f(x^n;\theta), y^n). \tag{3}$$

- As the number of data samples $N \to \infty$, $\mathcal{L}^N(\theta) \to \mathcal{L}(\theta)$.

The best model is

$$\theta = \arg\min_{\theta' \in \Theta} \mathcal{L}^N(\theta'). \tag{4}$$

- For complicated models such as neural networks, these minimization problems cannot be exactly calculated.
- Instead, numerical methods are used to minimize the objective functions.
- Convexity versus Non-Convexity
- In the non-convex case, numerical methods are only guaranteed to converge to a point which satisfies certain optimization properties.
- **Stochastic gradient descent** is the method of choice for training deep learning models.

Example

- Consider a logistic regression model for classification where $\mathcal{Y} = \{0, 1, \ldots, K-1\}$ and $\Theta = \mathbb{R}^{K \times d}$.

- Given an input $x \in \mathbb{R}^d$, the model $f(x; \theta)$ produces a probability of each possible outcome in $\mathcal{Y}$:

$$\begin{aligned} f(x;\theta) &= F_{\text{softmax}}(\theta x), \\ F_{\text{softmax}}(z) &= \frac{1}{\sum_{k=0}^{K-1} e^{z_k}} \left(e^{z_0}, e^{z_1}, \ldots, e^{z_{K-1}} \right). \end{aligned} \tag{5}$$

- $F_{\text{softmax}}(z)$ takes a K-dimensional input and produces a probability distribution on $\mathcal{Y}$.

- The function $F_{\text{softmax}}(z) : \mathbb{R}^K \to \mathcal{P}(\mathcal{Y})$ is called the “softmax function” and is frequently used in deep learning.

The objective function is the negative log-likelihood (commonly referred to in machine learning as the “cross-entropy error”):

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(X,Y)}\big[\rho(f(X;\theta),Y)\big], \\ \rho(z,y) &= -\sum_{k=0}^{K-1} \mathbf{1}_{y=k} \log z_k, \end{aligned} \tag{6}$$

where z_k is the k-th element of the vector z and $\mathbf{1}_{y=k}$ is the indicator function

$$\mathbf{1}_{y=k} = \begin{cases} 1 & y = k \\ 0 & y \neq k \end{cases}$$

Gradient Descent (GD):

$$\theta^{(\ell+1)} = \theta^{(\ell)} - \alpha^{(\ell)} \nabla_\theta \mathcal{L}(\theta^{(\ell)}). \tag{7}$$

- Gradient descent repeatedly takes steps in the direction of *steepest descent*.
- The magnitude of these steps is governed by the "learning rate" $\alpha^{(\ell)}$, which is a positive scalar which may depend upon the iteration number ℓ.

We can show that if the learning rate $\alpha^{(\ell)}$ is sufficiently small, the ℓ-th step of the gradient descent algorithm (7) is guaranteed to decrease the objective function.

Using a Taylor expansion,

$$\begin{aligned} \mathcal{L}(\theta^{(\ell+1)}) - \mathcal{L}(\theta^{(\ell)}) &= \nabla_\theta \mathcal{L}(\theta^{(\ell)})(\theta^{(\ell+1)} - \theta^{(\ell)}) \\ + \quad & \frac{1}{2}(\theta^{(\ell+1)} - \theta^{(\ell)})^\top \nabla_{\theta\theta}\mathcal{L}(\bar{\theta})(\theta^{(\ell+1)} - \theta^{(\ell)}). \end{aligned} \tag{8}$$

Substitute for $\theta^{(\ell+1)} - \theta^{(\ell)}$ using the gradient descent update

equation:

$$\begin{aligned} \mathcal{L}(\theta^{(\ell+1)}) - \mathcal{L}(\theta^{(\ell)}) &= -\alpha^{(\ell)}\left(\nabla_\theta \mathcal{L}(\theta^{(\ell)})\right)^\top \nabla_\theta \mathcal{L}(\theta^{(\ell)}) \\ &+ \frac{1}{2}\left(\alpha^{(\ell)}\right)^2 \nabla_\theta \mathcal{L}(\theta^{(\ell)})^\top \nabla_{\theta\theta}\mathcal{L}(\bar{\theta}^{(\ell)}) \nabla_\theta \mathcal{L}(\theta^{(\ell)}), \end{aligned} \tag{9}$$

- It is also clear that if $\alpha^{(\ell)}$ is too large, the objective function may *increase* due to the second-order term.

- In practice, a careful choice of the learning rate is very important.

- The gradient descent algorithm uses only the first derivative $\nabla_\theta \mathcal{L}(\theta)$ to update the parameter θ.

- If it takes too large of a step, the first derivative no longer accurately describes the change in the objective function.

Gradient descent requires computing the gradient $\nabla_\theta \mathcal{L}(\theta^{(\ell)})$, which can be computationally costly since it involves an integral over (x, y):

$$\begin{aligned} \nabla_\theta \mathcal{L}(\theta^{(\ell)}) &= \nabla_\theta \mathbb{E}_{(X,Y)}\big[\rho(f(X;\theta^{(\ell)}), Y)\big] \\ &= \mathbb{E}_{(X,Y)}\big[\nabla_\theta \rho(f(X;\theta^{(\ell)}), Y)\big]. \end{aligned} \tag{10}$$

Stochastic gradient descent (SGD) is a computationally efficient scheme for minimizing $\mathcal{L}(\theta)$.

It follows a *noisy* (but unbiased) descent direction:

$$\theta^{(\ell+1)} = \theta^{(\ell)} - \alpha^{(\ell)} \nabla_\theta \rho(f(x^{(\ell)}; \theta^{(\ell)}), y^{(\ell)}), \tag{11}$$

where $(x^{(\ell)}, y^{(\ell)})$ are i.i.d. samples from the distribution $\mathbb{P}_{(X,Y)}$.

The *average* descent direction in (11) equals the GD algorithm's descent direction since

$$\begin{aligned} & \mathbb{E}\left[\nabla_\theta \rho(f(x^{(\ell)}; \theta^{(\ell)}), y^{(\ell)}) \middle| \theta^{(\ell)}\right] \\ = \; & \mathbb{E}\left[\nabla_\theta \rho(f(X; \theta^{(\ell)}), Y) \middle| \theta^{(\ell)}\right] \\ = \; & \nabla_\theta \mathcal{L}(\theta^{(\ell)}). \end{aligned} \tag{12}$$

The distribution $\mathbb{P}_{(X,Y)}$ is usually unknown.

Instead, data samples $(x_n, y_n)_{n=1}^{N}$ are available from the distribution $\mathbb{P}_{(X,Y)}$.

Then, objective function can be approximated as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \rho(f(x^n;\theta), y^n). \tag{13}$$

The gradient descent algorithm for (13) is

$$\theta^{(\ell+1)} \quad = \quad \theta^{(\ell)} - \alpha^{(\ell)} \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \rho(f(x^n;\theta^{(\ell)}), y^n). \tag{14}$$

The stochastic gradient descent algorithm is:

- Randomly initialize the parameter $\theta^{(0)}$.
- For $\ell = 0, 1, \ldots, L$:
  - Select a data sample $(x^{(\ell)}, y^{(\ell)})$ at random from the dataset $(x_n, y_n)_{n=1}^{N}$.
  - Calculate the gradient for the loss from the data sample $(x^{(\ell)}, y^{(\ell)})$:

$$G^{(\ell)} = \nabla_\theta \rho(f(x^{(\ell)}; \theta^{(\ell)}), y^{(\ell)}) \tag{15}$$

  - Update the parameters:

$$\theta^{(\ell+1)} = \theta^{(\ell)} - \alpha^{(\ell)} G^{(\ell)}, \tag{16}$$

where $\alpha^{(\ell)}$ is the learning rate.

- **In practice, stochastic gradient descent typically converges much more rapidly than gradient descent!**
- GD converges slowly since in order to take a single step, it must calculate the gradients for every data sample in the dataset.
- In contrast, SGD can rapidly take many steps since each step only requires calculating the gradient for a single data sample.
- **SGD is especially advantageous when the size of the dataset N is large.**

The learning rate must satisfy the following conditions in order for SGD to converge:

$$\begin{aligned} \sum_{\ell=0}^{\infty} \alpha^{(\ell)} &= \infty, \\ \sum_{\ell=0}^{\infty} \left(\alpha^{(\ell)}\right)^2 &< \infty. \end{aligned} \tag{17}$$

A learning rate which satisfies these conditions is

$$\alpha^{(\ell)} = \frac{C_0}{C_1 + \ell}. \tag{18}$$

It is often sufficient in practice to simply use a piecewise learning rate schedule for $\ell = 0, 1, \ldots, K_4$ such as

$$\alpha^{(\ell)} = \begin{cases} C & \ell \leq K_1 \\ C \times 10^{-1} & K_1 < \ell \leq K_2 \\ C \times 10^{-2} & K_2 < \ell \leq K_3 \\ C \times 10^{-3} & K_3 < \ell \leq K_4 \end{cases}$$

If the learning rate is too small, convergence may be very slow.

If the learning rate is too large, the algorithm may oscillate and make no progress.

Theorem

Suppose that $\nabla_\theta \mathcal{L}(\theta)$ is globally Lipschitz and bounded. Furthermore, assume that the condition (17) holds and $\mathcal{L}(\theta)$ is bounded. Then,

$$\mathbb{P}\left[\lim_{\ell \to \infty} \nabla_\theta \mathcal{L}(\theta^{(\ell)}) = 0\right] = 1.$$

- Neural networks are not globally Lipschitz.
- Neural networks are not bounded.
- Neural networks are non-convex: SGD may converge to a local minimum and not a global minimum!

The **mini-batch stochastic gradient descent algorithm** is:

- Randomly initialize the parameter $\theta^{(0)}$.
- For $\ell = 0, 1, \ldots, L$:
  - Select M data samples $(x^{(\ell,m)}, y^{(\ell,m)})_{m=1}^{M}$ at random from the dataset $(x_n, y_n)_{n=1}^{N}$, where $M \ll N$.
  - Calculate the gradient for the loss from the data samples:

$$G^{(\ell)} = \frac{1}{M} \sum_{m=1}^{M} \nabla_\theta \rho(f(x^{(\ell,m)}; \theta^{(\ell)}), y^{(\ell,m)}) \tag{19}$$

  - Update the parameters:

$$\theta^{(\ell+1)} \quad = \quad \theta^{(\ell)} - \alpha^{(\ell)} G^{(\ell)}, \tag{20}$$

where $\alpha^{(\ell)}$ is the learning rate.

- The mini-batch update $G^{(\ell)}$ is clearly still an unbiased estimate for the gradient $\nabla_\theta \mathcal{L}(\theta^{(\ell)}$.

- It is less noisy than the stochastic gradient descent update with a single sample, i.e.

$$\begin{aligned}
\mathrm{Var}\left[G^{(\ell)} \middle| \theta^{(\ell)} \right] &= \mathrm{Var}\left[\frac{1}{M} \sum_{m=1}^{M} \nabla_\theta \rho(f(x^{(\ell,m)}; \theta^{(\ell)}), y^{(\ell,m)}) \middle| \theta^{(\ell)} \right] \\
&= \frac{1}{M} \mathrm{Var}\left[\nabla_\theta \rho(f(x^{(\ell)}; \theta^{(\ell)}), y^{(\ell)}) \middle| \theta^{(\ell)} \right]. \qquad (21)
\end{aligned}$$

- The conditional variance of a mini-batch update is smaller by a factor of $\frac{1}{M}$ than stochastic gradient descent with a single sample, where M is the mini-batch size.