IE 534/CS 547 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2019

Lecture 2

A fully-connected network with a single hidden layer is a function

$$\begin{aligned} Z &= Wx + b^1 \\ H_i &= \sigma(Z_i), \quad i = 0, \ldots, d_H - 1, \\ f(x;\theta) &= C^\top H + b^2. \end{aligned} \tag{1}$$

- The neural network model can be used to predict an outcome $Y \in \mathbb{R}^K$ given an input $X \in \mathbb{R}^d$.

- Then, $\rho(z, y) = \|z - y\|^2$ and

$$\mathcal{L}(\theta) = \mathbb{E}_{X,Y}\left[\, \|Y - f(X;\theta)\|^2 \,\right]. \tag{2}$$

A global minimum of the objective function (2) is

$$\theta^* \in \arg\min_{\theta} \mathcal{L}(\theta). \tag{3}$$

Let $Y = f^*(X)$. That is, we would like to learn a model $f(x;\theta)$ for the relationship $y = f^*(x)$ by observing data samples (X, Y). Under mild technical conditions, for any $\epsilon > 0$, there exists a neural network with d_H hidden units such that

$$\mathbb{E}_{X,Y}\left[\, \|Y - f(X;\theta^*)\|^2 \,\right] < \epsilon. \tag{4}$$

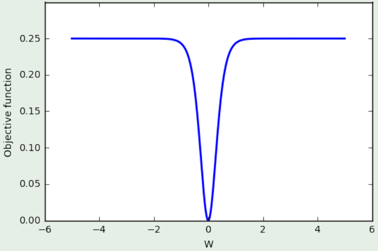
- However, numerically solving for the global minimum is intractable in practice.
- Instead, the objective function $\mathcal{L}(\theta)$ is minimized using stochastic gradient descent, which may converge to a local minimum.
- Remarkably–we can show asymptotically, as the number of hidden units become large, a neural network trained with stochastic gradient descent will converge to the global minimum (under certain conditions).

Neural networks are:

- Non-convex
- Multiple local minima
- Not Globally Lipschitz

Example

A simple example is presented to show that neural networks are non-convex. Consider a neural network with the squared loss (2) and a single hidden unit. The objective function is displayed below and is clearly non-convex.



Example

Neural networks can also have local minima, which are not global minima. As a simple example, consider a one-layer network $f : \mathbb{R} \to \mathbb{R}$ with a single ReLU unit:

$$f(x; \theta) = c \max(Wx + b^1, 0) + b^2. \tag{5}$$

Suppose the dataset consists of two data points $\{(x_0, y_0), (x_1, y_1)\}$ and that the loss function is $\rho(z, y) = (z - y)^2$. Any $c, W \in \mathbb{R}$, $b^1 < \min(-Wx_0, -Wx_1)$, and $b^2 = \frac{y_0+y_1}{2}$ is a local minimum.

- Neural networks can also be used for classification problems where a model is trained to predict a *categorical* outcome given an input.

- In this case, the outcome is one of a set of discrete values $\mathcal{Y} = \{0, 1, \ldots, K - 1\}$.

- The output of the model will be a vector of probabilities for these potential outcomes.

In order to perform classification, a softmax layer is added to the neural network.

$$
\begin{aligned}
Z &= Wx + b^1 \\
H_i &= \sigma(Z_i), \quad i = 0, \ldots, d_H - 1, \\
U &= CH + b^2, \\
f(x;\theta) &= F_{\mathrm{softmax}}(U).
\end{aligned}
\tag{6}
$$

The dimensions of the W, C, b^1, and b^2 remain the same as before.

The objective function is:

$$
\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{(X,Y)}\big[\rho(f(X;\theta), Y)\big], \\
\rho(v, y) &= -\sum_{k=0}^{K-1} \mathbf{1}_{y=k} \log v_k.
\end{aligned}
\tag{7}
$$

Backpropagation algorithm:

- Randomly select a new data sample (X, Y).
- Compute the forward step $Z, H, U, f(X;\theta)$, and $\rho\big(f(X;\theta), Y\big)$.
- Calculate the partial derivative $\frac{\partial \rho}{\partial U} = -\Big(e(Y) - f(X;\theta)\Big)$.
- Calculate the partial derivatives $\frac{\partial \rho}{\partial b^2} = \frac{\partial \rho}{\partial U}$, $\frac{\partial \rho}{\partial C} = \frac{\partial \rho}{\partial U} H^\top$, and $\delta = C^\top \frac{\partial \rho}{\partial U}$.
- Calculate the partial derivatives

$$\begin{aligned} \frac{\partial \rho}{\partial b^1} &= \delta \odot \sigma'(Z), \\ \frac{\partial \rho}{\partial W} &= \Big(\delta \odot \sigma'(Z)\Big) X^\top. \end{aligned} \tag{8}$$

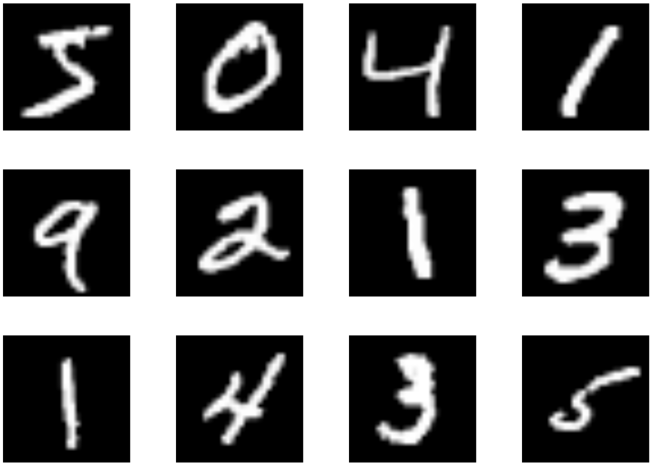
- Update the parameters $\theta = \{C, b^2, W, b^1\}$ with a stochastic gradient descent step.

PyTorch

- PyTorch and TensorFlow are software libraries which can perform automatic differentiation of deep learning models.
- This can significantly accelerate the development and testing of deep learning models!
- PyTorch has a define-by-run framework while TensorFlow is a define-and-run framework.
- PyTorch is more **seamlessly integrated with Python** than TensorFlow.
- PyTorch has better distributed training capabilities.

PyTorch implementation

- Define model
- Convert data to PyTorch Tensors/Variables
- Apply model to data
- Compute objective function
- Backward()
- Update model



Examples of images from the MNIST dataset. Each image is described by a 28×28 array of pixels, which can be re-arranged into a vector $x \in \mathbb{R}^{784}$. The vector x containing the pixel values is the input to the neural network, which attempts to correctly predict the handwritten number in the image.

We now consider a multi-layer neural network.

$$
\begin{aligned}
Z^1 &= W^1 x + b^1, \\
H^1 &= \sigma(Z^1), \\
Z^\ell &= W^\ell H^{\ell-1} + b^\ell, \quad \ell = 2, \ldots, L, \\
H^\ell &= \sigma(Z^\ell), \quad \ell = 2, \ldots, L, \\
U &= W^{L+1} H^L + b^{L+1}, \\
f(x;\theta) &= F_{\text{softmax}}(U).
\end{aligned} \tag{9}
$$

The neural network has L hidden layers followed by a softmax function. Each layer of the neural network has d_H hidden units. The ℓ-th hidden layer is $H^\ell \in \mathbb{R}^{d_H}$. H^ℓ is produced by applying an element-wise nonlinearity to the input $Z^\ell \in \mathbb{R}^{d_H}$. Using a slight abuse of notation,

$$
\sigma(Z^\ell) = \Big(\sigma(Z_0^\ell), \sigma(Z_1^\ell), \ldots, \sigma(Z_{d_H-1}^\ell) \Big). \tag{10}
$$

The SGD algorithm for updating θ is:

- Randomly select a new data sample (X, Y).
- Compute the forward step $Z^1, H^1, \ldots, Z^L, H^L, U, f(X;\theta)$, and $\rho := \rho\big(f(X;\theta), Y\big)$.
- Calculate the partial derivative

$$\frac{\partial \rho}{\partial U} = -\Big(e(Y) - f(X;\theta)\Big). \tag{11}$$

- Calculate the partial derivatives $\frac{\partial \rho}{\partial b^{L+1}}$, $\frac{\partial \rho}{\partial W^{L+1}}$, and δ^L.
- For $\ell = L-1, \ldots, 1$:
  - Calculate δ^ℓ via the formula

$$\delta^\ell = \big(W^{\ell+1}\big)^\top \big(\delta^{\ell+1} \odot \sigma'(Z^{\ell+1})\big). \tag{12}$$

  - Calculate the partial derivatives with respect to W^ℓ and b^ℓ.
- Update the parameters θ with a stochastic gradient descent step.

Total number of arithmetic operations:

$$N\Big[(L-1)(3d_H^2+d_H)+d_H(2d_H+d+3K+1)+2K\Big]. \qquad (13)$$

Memory required to store model parameters:

$$(L-1)(d_H^2+d_H)+d_H(d+K)+K. \qquad (14)$$

Memory cost for backpropagation algorithm:

$$(L-1)(d_H^2+d_H)+d_H(d+K)+K+2Nd_H. \qquad (15)$$