DS 801: Advanced Optimization for Data Science **KAIST, Fall 2024**
Lecture #15: Training Neural Networks and Lagrangian Duality April 24, 2024
Lecturer: Dabeen Lee

1 Outline

In this lecture, we study training neural networks from the perspective of Lagrangian duality.

2 Neural Networks

Let us consider a neural network with a single hidden layer illustrated as in Figure 15.1. Given the

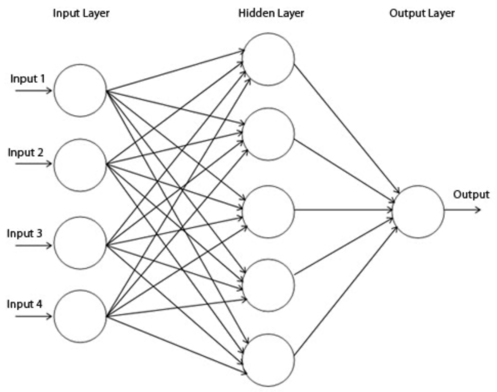

Figure 15.1: Single hidden layer neural network

predictor variable vector $x \in \mathbb{R}^d$, the neural network can model the response variable $y \in \mathbb{R}$ with

$$\mathbb{E}\left[y \mid x\right] = w_2^\top \sigma(W_1 x) \tag{15.1}$$

where

- $W_1 x$ is the output of the input layer,
- σ is an activation function,
- w_2 is the weight vector that the hidden layer applies.

Common choices for an activation function include

- sigmoid function: $\sigma(z) = 1/(1+e^{-z})$,
- Tanh function: $\sigma(z) = (e^z - z^{-z})/(e^z + e^{-z})$,
- ReLU: $\sigma(z) = \max\{0, z\}$.

Note that the output of the input layer $W_1 x$ is a vector with multiple components, and $\sigma(W_1 x)$ applies the activation function on individual components of the vector. Basically, for $z \in \mathbb{R}^d$, $\sigma(z) = (\sigma(z_1), \ldots, \sigma(z_d))$.

2.1 Multiple Hidden Layers

We may generalize the single hidden layer neural network to a neural network with multiple hidden layers. Moreover, we consider the scenario where the response variable $y \in \mathbb{R}^{d_y}$ is vector-valued. The predictor variable is given by $x \in \mathbb{R}^{d_x}$. Then we use a neural network $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ given as

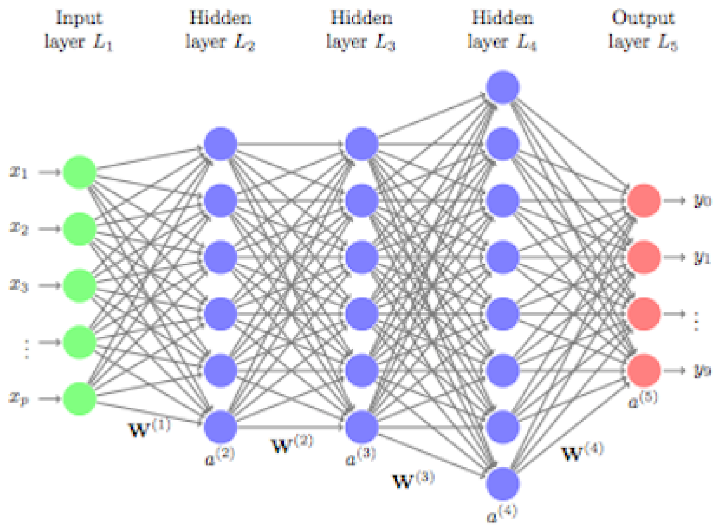

Figure 15.2: Fully connected feedforward neural network

follows.

$$f_\theta(x) = W^L \sigma(W^{L-1}(\cdots \sigma(W^2 \sigma(W^1 x)) \cdots)) \tag{15.2}$$

where

- $W_1 x$ is the output of the input layer,
- there are $L - 1$ hidden layers,
- the ith hidden layer receives $W^i(\cdots \sigma(W^2 \sigma(W^1 x)) \cdots))$ and outputs $W^{i+1}\sigma(W^i(\cdots))$ for $i \in [L-1]$,
- θ represents the collection of all parameters $\theta = (W^1, \ldots, W^L)$.

Note that the neural network f_θ given in (15.2) has a complex composite structure. Another way to represent f_θ is to use the following recursion:

$$\begin{aligned} z^0 &= x, & h^1 &= W^1 z^0, \\ z^1 &= \sigma(h^1), & h^2 &= W^2 z^1, \\ &\vdots & &\vdots \\ z^{L-1} &= \sigma(h^{L-1}), & h^L &= W^L z^{L-1}. \end{aligned}$$

Here, z^0 is the input. For $i = 1, \ldots, L-1$, the ith hidden layer receives h^i and outputs z^i. We often call h^i **pre-activation** and z^i **post-activation**. Note that z^i is multiplied by W^{i+1} as it moves from the ith layer to the $i+1$th layer to become $h^{i+1} = W^{i+1} z^i$.

The most general setting is to consider bias terms, in which case we get

$$f_\theta(x) = W^L\sigma(W^{L-1}(\cdots\sigma(W^2\sigma(W^1x+b^1)+b^2)\cdots)+b^{L-1})+b^L$$

and

$$\begin{aligned} z^0 &= x, & h^1 &= W^1z^0+b^1,\\ z^1 &= \sigma(h^1), & h^2 &= W^2z^1+b^2,\\ &\vdots & &\vdots\\ z^{L-1} &= \sigma(h^{L-1}), & h^L &= W^Lz^{L-1}+b^L. \end{aligned}$$

For a simpler presentation, we omit the bias terms and consider (15.2).

Suppose that we have n data points $(x_1,y_1),\ldots,(x_n,y_n)$. Then we want to choose the parameter θ so that the predicted outcome $f_\theta(x_i)$ is close to y_i. For a loss function $\ell:\mathbb{R}^{d_y}\times\mathbb{R}^{d_y}\to\mathbb{R}$, we consider

$$\min_\theta \quad F(\theta) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, f_\theta(x_i)). \tag{15.3}$$

For regression, we typically take the squared loss function $\ell(y,\hat{y}) = \|y-\hat{y}\|_2^2$. For binary classification, we take $\ell(y,\hat{y}) = \log(1+\exp(-y\hat{y}))$.

To find a parameter θ minimizing the loss function $F(\theta)$ given in (15.3), we may apply gradient descent:

$$\theta_{t+1} = \theta_t - \eta_t\nabla_\theta F(\theta_t).$$

Applying stochastic gradient descent, we sample a data point (x,y) and take

$$\theta_{t+1} = \theta_t - \eta_t\nabla_\theta \ell(y, f_\theta(x)).$$

Here, let alone the non-convexity of $F(\theta)$ and $\ell(y,f_\theta(x))$, we need to consider the complex composite structure of the function $\ell(y,f_\theta(x))$ from multiple layers in the neural network. **Backpropagation** allows efficient computation of the gradient for training neural networks.

2.2 Connection to Linear Regression

Recall that linear regression is to learn the linear model with coefficient vector $w\in\mathbb{R}^d$ such that

$$y = w^\top x$$

where $x\in\mathbb{R}^d$ is the predictor variable and $y\in\mathbb{R}$ is the response variable. Given n data points $(x_1,y_1),\ldots,(x_n,y_n)$, we consider

$$\min_w \quad \frac{1}{n}\sum_{i=1}^n (y_i - w^\top x_i)^2.$$

Note that the linear model is equivalent to a neural network with no hidden layer.

2.3 Connection to Matrix Factorization

We discussed the matrix factorization problem where the goal is to approximate an $n \times p$ matrix D by the product of two low rank matrices $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{p\times k}$:

$$D \simeq UV^\top.$$

Recall that the matrix factorization problem can be solved by

$$\min_{U\in\mathbb{R}^{n\times k},\ V\in\mathbb{R}^{p\times k}} \quad \left\| D - UV^\top \right\|_F^2. \tag{15.4}$$

For $i \in [n]$, let $y_i \in \mathbb{R}^p$ denote the ith row of D, and let $x_i \in \mathbb{R}^n$ denote the ith unit vector in $\mathbb{R}^n$. Then it follows that

$$\left\| D - UV^\top \right\|_F^2 = \sum_{i=1}^n \left\| y_i - VU^\top x_i \right\|_2^2.$$

Then (15.4) is equivalent to

$$\min_{W_1\in\mathbb{R}^{k\times n},\ W_2\in\mathbb{R}^{p\times k}} \quad \sum_{i=1}^n \|y_i - W_2W_1x_i\|_2^2.$$

Hence, the matrix factorization problem reduces to a neural network with a single hidden layer and without an activation function.

3 Backpropagation and Lagrangian Duality

Let us consider the following composite optimization problem.

$$\min_\theta \quad f(g(\theta)).$$

Here, the problem can be rewritten as

$$\begin{aligned} \min \quad & f(z) \\ \text{s.t.} \quad & z = g(\theta). \end{aligned}$$

Taking the Lagrangian, we get

$$\mathcal{L}(\theta, z, \mu) = f(z) - \mu(z - g(\theta)).$$

Applying KKT conditions,

$$\begin{aligned} 0 &= \nabla_\theta \mathcal{L}(\theta, z, \mu) = \mu g'(\theta) \\ 0 &= \nabla_z \mathcal{L}(\theta, z, \mu) = f'(z) - \mu \\ 0 &= \nabla_\mu \mathcal{L}(\theta, z, \mu) = -z + g(\theta). \end{aligned}$$

This implies that

$$0 = \mu g'(\theta) = f'(z)g'(\theta) = f'(g(\theta))g'(\theta),$$

which basically is the chain rule. Hence, the dual formulation does implement backpropagation!

In general, let us consider a composite function given by the following procedure. We consider what we call a **computation graph** G where

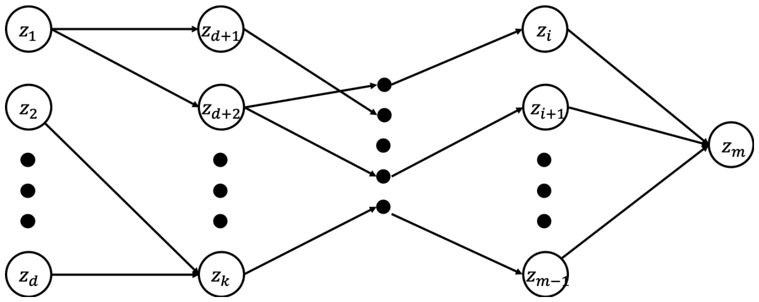

Figure 15.3: Computation graph

- $G = (N, A)$ is a directed acyclic graph,
- $|N| = m$,
- $\alpha(i) = \{k \in N : (k, i) \in A\}$ denotes the set of ancestors of i for $i \in N$,
- $\beta(i) = \{j \in N : (i, j) \in A\}$ denotes the set of successors of i for $i \in N$.

For each node $i \in N$, we declare a variable z_i. Here,

- $z_1, \dots, z_d$ are the input variables, i.e., $\theta = (z_1, \dots, z_d)$,
- $z_{d+1}, \dots, z_{m-1}$ are the intermediate variables,
- z_m is the output,
- $z_{\alpha(i)}$ collects the variables z_k for $k \in \alpha(i)$,
- the output of node $i \in N$ is given by $f_i(z_{\alpha(i)})$.

Here, the problem can be reformulated as

$$\begin{array}{ll} \min & z_m \\ \text{s.t.} & z_i = f_i(z_{\alpha(i)}) \quad \text{for } i = d+1, \dots, m. \end{array}$$

The Lagrangian of this formulation is given by

$$\mathcal{L}(z, \mu) = z_m - \sum_{i=d+1}^{m} \mu_i (z_i - f_i(z_{\alpha(i)})).$$

Here, we apply KKT conditions.

1. Setting $\nabla_{\mu_i}(\mathcal{L}) = 0$, we obtain $z_i = f_i(z_{\alpha(i)})$. This is basically the **forward pass**.
2. Setting $\nabla_{z_m}\mathcal{L} = 0$, we deduce
$$\nabla_{z_m}\mathcal{L} = 1 - \mu_m = 0,$$
which implies $\mu_m = 1$.

3. Setting $\nabla_{z_j}\mathcal{L} = 0$ for $j < m$, we deduce

$$\begin{aligned} 0 &= \nabla_{z_j}\mathcal{L} \\ &= -\mu_j + \sum_{i \in \beta(j)} \mu_i \frac{\partial f_i(z_{\alpha(i)})}{\partial z_j}, \end{aligned}$$

which implies

$$\mu_j = \sum_{i \in \beta(j)} \mu_i \frac{\partial f_i(z_{\alpha(i)})}{\partial z_j}.$$

Here, this is the **backward pass**.

Theorem 15.1. *For $d + 1 \leq j \leq m$, we have*

$$\mu_j = \frac{\partial f(\theta)}{\partial z_j}.$$

Proof. We apply induction on j. For $j = m$, note that $f(\theta) = f_m(z_{\alpha(m)}) = z_m$. Since $1 = \mu_m$, we satisfy

$$\frac{\partial f(\theta)}{\partial z_m} = \frac{\partial z_m}{\partial z_m} = 1 = \mu_m.$$

For $j < m$, we have

$$\begin{aligned} \lambda_j &= \sum_{i \in \beta(j)} \mu_i \frac{\partial f_i(z_{\alpha(i)})}{\partial z_j} \\ &= \sum_{i \in \beta(j)} \frac{\partial f(\theta)}{\partial z_i} \frac{\partial f_i(z_{\alpha(i)})}{\partial z_j} \\ &= \frac{\partial f(\theta)}{\partial z_j} \end{aligned}$$

where the second equality holds due to the induction hypothesis. □