CSE6243: Advanced Machine Learning **Fall 2024**

Lecture 9: Neural Network Revisit

Lecturer: Bo Dai *Scribes: Shuojiang Liu, Donaven Lobo*

Note: *LaTeX template courtesy of UC Berkeley EECS Department.*

9.1 Recap

The present lesson marks the conclusion of Module 1 of the course. The module 1 is comprised of three principal topics:

- Convex Optimization
- Density Parametrization
- Sampling Strategies

9.2 New Content

In today's lecture, we will demonstrate that a neural network is, in essence, just a function approximator or function parametrization. The manner in which it is learned is contingent upon the specific learning problem at hand.

For instance, polynomial is also a simple neural network.

9.2.1 Linear Model

The basic function of linear model is:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} \tag{9.1}$$

where we have data pairs $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}$.

If $\mathcal{X} = \mathbb{R}^D, \mathcal{Y} = \mathbb{R}$, where D is the dimension of the number of input features, the model is **regression**.

For regression, we use quadratic loss for the loss function. The empirical risk associated with the use of quadratic loss is equivalent to the mean squared error (MSE):

$$loss(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|^2 \tag{9.2}$$

From a different perspective, we can model the uncertainty in the prediction as Gaussian:

$$\mathcal{N}\left(y|\mu,\sigma^2\right)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}(y-\mu)^2},\mu=f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right) \tag{9.3}$$

We can get the conditional probability distribution:

$$p\left(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}\right)=\mathcal{N}\left(y_n|f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n\right),\sigma^2\right) \tag{9.4}$$

Similarly, if $\mathcal{X}=\mathbb{R}^D,\mathcal{Y}=\{1,2,\cdots,C\}$, where D is the dimension of the number of input features and C is the number of classes, the model is **classification**.

From the frequentist's point of view, the negative log likelihood of the training set is defined as:

$$loss\left(\boldsymbol{\theta}\right)=-\frac{1}{N}\sum_{i=1}^{N}\log p\left(y_i|f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right)\right) \tag{9.5}$$

By minimizing this quantity, we can compute the maximum likelihood estimate (MLE):

$$\hat{\boldsymbol{\theta}}_{MLE}=\arg\min_{\boldsymbol{\theta}}loss\left(\boldsymbol{\theta}\right) \tag{9.6}$$

9.2.2 Basic Neural Network Modeling

A neural network $f\left(\boldsymbol{x};\boldsymbol{\theta}\right)$, can be modeled as:

$$f\left(\boldsymbol{x};\boldsymbol{\theta}\right)=f_M\left(f_{M-1}\left(\cdots\left(f_1\left(\boldsymbol{x}\right)\cdots\right)\right)\right) \tag{9.7}$$

where:

$$f_l\left(\boldsymbol{x}\right)=f\left(\boldsymbol{x};\boldsymbol{\theta}_l\right)=\boldsymbol{\varphi}_l\left(\mathbf{W}_l\boldsymbol{x}+\boldsymbol{b}_l\right) \tag{9.8}$$

The function $\boldsymbol{\varphi}_l$ is the activation function such as Sigmoid: $\sigma\left(\boldsymbol{z}\right)=\frac{1}{1+e^{-\boldsymbol{z}}}$, and ReLU: $\text{ReLU}\left(\boldsymbol{z}\right)=\max\left(\boldsymbol{z},\mathbf{0}\right)$.

A multilayer perceptron is a type of feedforward neural network comprising fully connected neurons with a nonlinear activation function, organized in at least three layers.

9.2.3 Kernel Methods

Kernel method is a kind of non-parametric method that involves using linear classifiers to solve nonlinear problems.

Kernel methods rely on mapping input data into a higher-dimensional space. Let $\boldsymbol{x}\in\mathbb{R}^n$ be an input feature vector. A feature mapping function ϕ transforms $\boldsymbol{x}$ into a higher-dimensional space:

$$\phi:\boldsymbol{x}\longmapsto\phi\left(\boldsymbol{x}\right)\in\mathbb{R}^m,m\gg n \tag{9.9}$$

In this space, data points that are not linearly separable in the original space can become linearly separable. Instead of computing the mapping $\phi(\boldsymbol{x})$ explicitly, kernel methods compute the inner product between two transformed points directly using a kernel function $\mathcal{K}$:

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle \tag{9.10}$$

In SVMs, the decision function for a classification problem is expressed as:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{9.11}$$

The objective is to find α_i that maximize the margin between classes in the transformed space.

Deep neural networks can be conceptualized as a form of learned feature mapping, wherein the feature space is learned from data during the training phase. In contrast, kernel methods employ predefined functions (e.g., Gaussian, polynomial) to transform the data into a higher-dimensional space.

9.2.4 Special Structred Neural Networks (Mentioned in the Slides)

9.2.4.1 Convolutional Neural Network (CNN)

CNNs are designed for grid-like data such as images. The key operations include convolution and pooling.

Given an input $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ and filter $\mathbf{W} \in \mathbb{R}^{k_H \times k_W \times C}$, the convolution output $\mathbf{Y}$ is:

$$\mathbf{Y}(i,j) = \sum_{m=1}^{k_H} \sum_{n=1}^{k_W} \sum_{c=1}^{C} \mathbf{X}(i+m, j+n, c) \cdot \mathbf{W}(m,n,c) \tag{9.12}$$

where:

- $\mathbf{X}$: Input tensor (image)
- $\mathbf{W}$: Filter weights
- $\mathbf{Y}$: Feature map

Typically, ReLU is used as activation function:

$$f(x) = \max(0, x) \tag{9.13}$$

Pooling (max pooling) reduces the spatial dimensions by selecting the maximum value in a window.

$$\mathbf{P}(i,j) = \max_{m,n} \mathbf{Y}(i+m, j+n) \tag{9.14}$$

9.2.4.2 Residual Neural Network (ResNet)

ResNet introduces shortcut connections to solve the problem of vanishing gradients in deep networks. The key idea is to add a shortcut (identity) connection:

$$\mathbf{y}_l = f(\mathbf{x}_l, \mathbf{W}_l) + \mathbf{x}_l \tag{9.15}$$

where:

- $\mathbf{x}_l$: Input to the layer
- $\mathbf{y}_l$: Output from the layer
- $f(\mathbf{x}_l, \mathbf{W}_l)$: Transformation (convolution, activation, etc.) with learnable weights $\mathbf{W}_l$

9.2.4.3 Recurrent Neural Network (RNN)

RNNs handle sequential data by maintaining a hidden state that evolves over time steps. For an input sequence $\mathbf{x}_t$ at time step t, the hidden state $\mathbf{h}_t$ is updated as:

$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}_h) \tag{9.16}$$

where:

- $\mathbf{x}_t$: Input at time t
- $\mathbf{h}_t$: Hidden state at time t
- $\mathbf{W}_h, \mathbf{W}_x$: Weight matrices
- $\mathbf{b}_h$: Bias term
- $f(\cdot)$: Non-linear activation (e.g., tanh)

The output $\mathbf{y}_t$ at time t is typically given by:

$$\mathbf{y}_t = \varphi(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{9.17}$$

where $\varphi(\cdot)$ is an activation function (e.g., softmax for classification).

9.2.4.4 Self-Attention (Building Blocks for Transformer)

Given queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, keys $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, and values $\mathbf{V} \in \mathbb{R}^{n \times d_v}$, the attention output $\mathbf{Z}$ is:

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{9.18}$$

where:

- $\mathbf{Q}, \mathbf{K}, \mathbf{V}$: Query, key, and value matrices
- d_k: Dimensionality of queries/keys (scaling factor)

9.2.4.5 Transformer

In transformer, multiple attention heads are used to capture different aspects of the input:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{Z}_1, \ldots, \mathbf{Z}_h)\mathbf{W}_O \tag{9.19}$$

where $\mathbf{Z}_i$ is the output of the i-th attention head, and $\mathbf{W}_O$ is a learned projection matrix.

After self-attention, a feed-forward network is applied to each position independently:

$$\text{FFN}(\mathbf{X}) = \max(0, \mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{9.20}$$

where:

- $\mathbf{W}_1, \mathbf{W}_2$: Weight matrices
- $\mathbf{b}_1, \mathbf{b}_2$: Bias terms

The Transformer combines multi-head self-attention and FFN with residual connections and layer normalization:

$$\mathbf{Y} = \text{LayerNorm}(\mathbf{X} + \text{MultiHead}(\mathbf{X}, \mathbf{X}, \mathbf{X})) \tag{9.21}$$

$$\mathbf{Z} = \text{LayerNorm}(\mathbf{Y} + \text{FFN}(\mathbf{Y})) \tag{9.22}$$

where:

- $\mathbf{X}$: Input to the layer
- $\mathbf{Y}, \mathbf{Z}$: Intermediate outputs

Layer normalization is applied after both the self-attention and feed-forward sub-layers:

$$\text{LayerNorm}(\mathbf{X}) = \frac{\mathbf{X} - \mu}{\sigma + \epsilon} \cdot \gamma + \beta \tag{9.23}$$

where:

- μ: Mean of the elements in $\mathbf{X}$
- σ: Standard deviation of elements in $\mathbf{X}$
- γ and β: Learnable scaling and shifting parameters

- ϵ: Small constant to avoid division by zero

Positional encoding is used to inject sequence order:

$$\text{PE}(i, 2k) = \sin\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \tag{9.24}$$

$$\text{PE}(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \tag{9.25}$$

where:

- i: Position index
- k: Dimension index
- d_{model}: Dimensionality of the model