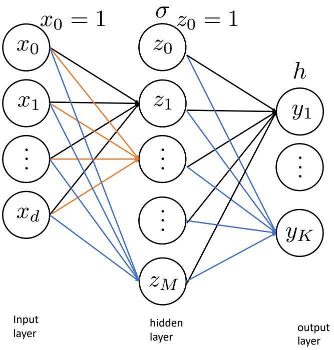
DDA5001 Machine Learning

Neural Networks (Part II): Training Formulation and BP

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen

Recap: One Hidden Layer (Two-Layer) Neural Network



Use $\boldsymbol{V}$ and $\boldsymbol{W}$ to denote the weight matrices of the first layer and second layer. NN model can be written as

$$\boxed{\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{W}\sigma(\boldsymbol{V}\boldsymbol{x})).}$$

- The input of the next layer is the output of the previous layer ($\boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x})$).

Recap: Ingredients and Interpretation of Neural Network

Activation function and last layer:

- σ is the activation function.
- Typical choices for σ are sigmoid, ReLU, SiLU, etc.
- The choice of h in the last layer depends on applications, which is to impose either linear regression or linear classification in the last layer using the learned feature $\boldsymbol{z}$.

Interpretation: One can think neural network model as extracting features by nonlinear network and finally put the extracted feature $\boldsymbol{z}$ into the last layer for linear regression or linear classification.

Universal approximation power: One hidden layer (two layer) NN can approximate almost arbitrary target g.

Training Two Layer Neural Networks

Backpropagation

Training Neural Networks

- Training data: $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{y}_i \in \mathbb{R}^K$
- Neural network model

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{W}\sigma(\boldsymbol{V}\boldsymbol{x})).$$

- We aim to learn the weight parameters $\boldsymbol{\theta} = (\boldsymbol{V}, \boldsymbol{W})$ such that

$$\boldsymbol{y}_i \leftarrow f_{\boldsymbol{\theta}}(\boldsymbol{x}_i).$$

  This is a supervised learning problem.

- We can quantify this approximation by choosing a loss function which we will seek to minimize by picking $\boldsymbol{\theta}$ appropriately

The Learning Problem for Training Neural Networks

Regression: $h(t) = t$ and use squared ℓ_2 loss, resulting in

- $K = 1$

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2 \right\}.$$

- $K > 1$

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2 \right\}.$$

The Learning Problem for Training Neural Networks

Classification (Binary): $K = 1, y = \{+1, -1\}$ and h is logistic function. MLE principle leads to

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \right\}.$$

Classification (Multi-class): $K > 1$, $\boldsymbol{y} = (0, \ldots, 1, \ldots, 0)$, h is soft-max. MLE principle leads to

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i^{\top} \log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \right\}.$$

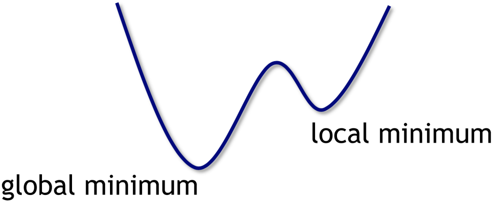
Summary: NN learning formulation is the same as least squares or logistic regression, with the difference being using $\boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x})$ as input.

Training Neural Networks is Nonconvex Optimization

For example, regression training problem can be written as:

$$\min_{\boldsymbol{\theta}=(\boldsymbol{V},\boldsymbol{W})} \left\{ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{y}_i - h(\boldsymbol{W}\sigma(\boldsymbol{V}\boldsymbol{x}_i))\|^2 \right\}.$$

This is a highly nonconvex optimization problem.



Recall that linear supervised learning always gives rise to convex optimization. The nonconvexity of NN learning comes from learning the feature $\boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x})$, where $\boldsymbol{V}$ is part of the learnable parameters.

Training Neural Networks

We put an abstract form of the former learning problems:

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}) \right\}$$

For different applications (regression or classification), ℓ_i has its own form.

- One can apply a gradient-based training algorithm.
- One needs to compute the gradient

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(\boldsymbol{\theta}).$$

- Apply gradient-based training algorithm:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \mathcal{L}(\boldsymbol{\theta}_k).$$

However... The gradient is not easy to compute and GD training algorithm might be too optimistic.

Next: Training Algorithm and its Ingredients

We next dive into the depth of neural network training:

- How to compute the gradient?
  $\rightsquigarrow$ The well-known backpropagation (BP).

- In contemporary applications, n is so large. Applying GD is not feasible.
  $\rightsquigarrow$ Stochastic gradient descent, Adagrad, and Adam family.

Training Two Layer Neural Networks

Backpropagation

Computing The Gradient: Squared ℓ_2-Loss as An Example

- Let us take the regression case where we use squared ℓ_2 loss as an example. The conclusion applies to other cases.
- We consider the general case where $K > 1$. We have

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|^2 = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{\theta})$$

  where

$$\ell_i(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2$$

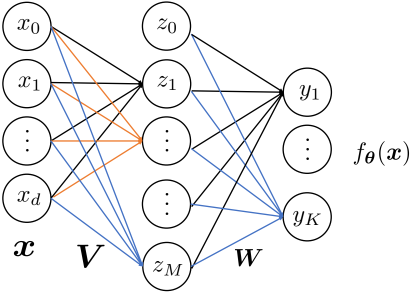
- Knowing how to take gradient for each ℓ_i suffices.

For ease of notation, we will omit the subscription i in ℓ_i and denote

$$\ell(\boldsymbol{\theta}) = \|\boldsymbol{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})\|^2.$$

Task: Computing $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}) = (\frac{\partial \ell}{\partial \boldsymbol{V}}, \frac{\partial \ell}{\partial \boldsymbol{W}})$.

Back to The Neural Network Architecture



The idea: Computing gradient using chain rule:

- **Forward pass**: Given an $\boldsymbol{x}$, compute (using the current parameters $\boldsymbol{V}$ and $\boldsymbol{W}$) $\boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x}), f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{W}\boldsymbol{z}), \ell(\boldsymbol{\theta}) = \|\boldsymbol{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2$
- **Backward pass**:
  - Second layer: Compute $\frac{\partial \ell}{\partial \boldsymbol{W}}$, and compute $\frac{\partial \ell}{\partial \boldsymbol{z}}$.
  - First layer: Compute $\frac{\partial \ell}{\partial \boldsymbol{V}} = \frac{\partial \ell}{\partial \boldsymbol{z}} \times \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{V}}$.

Deriving The Gradient: Second Layer

- We omit the offset w.l.o.g. i.e., no x_0 and z_0.
- By definition of gradient, we have

$$\frac{\partial \ell}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial \ell}{\partial W(1,1)} & \cdots & \frac{\partial \ell}{\partial W(1,M)} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell}{\partial W(K,1)} & \cdots & \frac{\partial \ell}{\partial W(K,M)} \end{bmatrix} \in \mathbb{R}^{K \times M}.$$

- We focus on one element $\frac{\partial \ell}{\partial W(k,m)}$ of $\frac{\partial \ell}{\partial \boldsymbol{W}}$.
- With respect to $\boldsymbol{W}$, we have

$$\ell(\boldsymbol{\theta}) = \|\boldsymbol{y} - h(\boldsymbol{W}\boldsymbol{z})\|_2^2,$$

where $\boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x}) \in \mathbb{R}^M$.

Deriving The Gradient: Second Layer

$$
\begin{aligned}
\frac{\partial \ell}{\partial W(k,m)} &= \frac{\partial}{\partial W(k,m)} \|h(\boldsymbol{W}\boldsymbol{z}) - \boldsymbol{y}\|_2^2 \\
&= \frac{\partial}{\partial W(k,m)} \sum_{j=1}^{K} \left(h(\boldsymbol{w}_j^\top \boldsymbol{z}) - y[j]\right)^2 \\
&= \frac{\partial}{\partial W(k,m)} \left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right)^2 \qquad \text{using chain rule} \rightarrow \\
&= \frac{\partial \left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right)^2}{\partial \left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right)} \times \frac{\partial \left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right)}{\partial \boldsymbol{w}_k^\top \boldsymbol{z}} \times \frac{\partial \boldsymbol{w}_k^\top \boldsymbol{z}}{\partial W(k,m)} \\
&= 2\left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right) \times h'(\boldsymbol{w}_k^\top \boldsymbol{z}) \times z[m] \\
&:= \delta[k] \times z[m],
\end{aligned}
$$

where $\boldsymbol{w}_k^\top \in \mathbb{R}^M$ is the k-th row of $\boldsymbol{W}$ and we have defined

$$
\delta[k] = 2\left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right) h'(\boldsymbol{w}_k^\top \boldsymbol{z}).
$$

Deriving The Gradient: Second Layer

Since

$$\frac{\partial \ell}{\partial W(k,m)} = \delta[k]z[m]$$

Denote

$$\boldsymbol{\delta} = \begin{bmatrix} \delta[1] \\ \delta[2] \\ \vdots \\ \delta[K] \end{bmatrix} = 2(h(\boldsymbol{W}\boldsymbol{z}) - \boldsymbol{y}) \odot h'(\boldsymbol{W}\boldsymbol{z}) \in \mathbb{R}^K$$

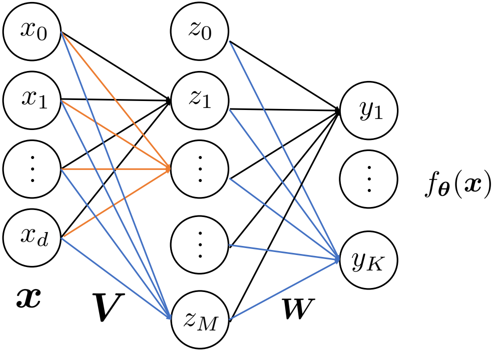
where $\odot$ is the Hadamard (element-wise) product.

We have

$$\frac{\partial \ell}{\partial \boldsymbol{W}} = \underbrace{\boldsymbol{\delta}}_{K\times 1} \underbrace{\boldsymbol{z}^\top}_{1\times M} \in \mathbb{R}^{K\times M},$$

which can be calculated using the current values of $\boldsymbol{W}$ and $\boldsymbol{z}$.

Deriving The Gradient: First Layer



Recall that the forward pass gives

$$\ell(\boldsymbol{\theta}) = \|\boldsymbol{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2, \quad f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{W}\boldsymbol{z}), \quad \boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x}).$$

We first back-propagate the gradient back to $\boldsymbol{z}$, and then calculate the gradient of $\boldsymbol{z}$ w.r.t. $\boldsymbol{V}$ (this is chain rule):

$$\frac{\partial \ell}{\partial \boldsymbol{V}} = \frac{\partial \ell}{\partial \boldsymbol{z}} \times \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{V}}$$

Deriving The Gradient: First Layer

To begin with, we back-propagate the gradient back to $\boldsymbol{z}$.
For an individual node of $\boldsymbol{z}$:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \boldsymbol{z}[m]} &= \frac{\partial}{\partial \boldsymbol{z}[m]} \|h(\boldsymbol{W}\boldsymbol{z}) - \boldsymbol{y}\|_2^2 \\
&= \frac{\partial}{\partial \boldsymbol{z}[m]} \sum_{k=1}^{K} \left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right)^2 \quad \text{using chain rule} \rightarrow \\
&= \sum_{k=1}^{K} 2\left(h(\boldsymbol{w}_k^\top \boldsymbol{z}) - y[k]\right) \times h'(\boldsymbol{w}_k^\top \boldsymbol{z}) \times w_k[m] \\
&= \sum_{k=1}^{K} \delta[k] w_k[m]
\end{aligned}
$$

Combining the individual nodes of $\boldsymbol{z}$, we denote

$$
\frac{\partial \ell}{\partial \boldsymbol{z}} = \boldsymbol{W}^\top \boldsymbol{\delta}
$$

Deriving The Gradient: First Layer

$$\frac{\partial \ell}{\partial \boldsymbol{V}} = \begin{bmatrix} \frac{\partial \ell}{\partial V(1,1)} & \cdots & \frac{\partial \ell}{\partial V(1,d)} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell}{\partial V(M,1)} & \cdots & \frac{\partial \ell}{\partial V(M,d)} \end{bmatrix} \in \mathbb{R}^{M \times d}.$$

For an individual element of $\boldsymbol{V}$:

$$\frac{\partial \ell}{\partial V(m,j)} = \underbrace{\frac{\partial \ell}{\partial \boldsymbol{z}[m]}}_{\text{gradient from next layer}} \times \underbrace{\frac{\partial \boldsymbol{z}[m]}{\partial V(m,j)}}_{\text{local gradient}} \qquad \text{(chain rule)}$$

$$= \frac{\partial \ell}{\partial \boldsymbol{z}[m]} \times \frac{\partial \sigma(\boldsymbol{v}_m^\top \boldsymbol{x})}{\partial V(m,j)} \qquad (\text{since } \boldsymbol{z} = \sigma(\boldsymbol{V}\boldsymbol{x}))$$

$$= \sum_{k=1}^{K} \delta[k] w_k[m] \times \sigma'(\boldsymbol{v}_m^\top \boldsymbol{x}) \times x[j]$$

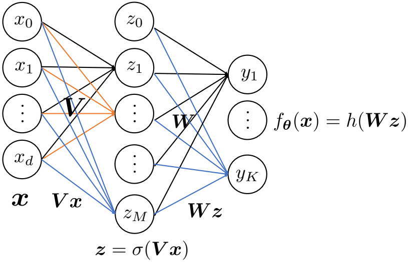
Combining the gradients of the individual elements:

$$\frac{\partial \ell}{\partial \boldsymbol{V}} = \underbrace{\Big((\boldsymbol{W}^\top \boldsymbol{\delta}) \odot \sigma'(\boldsymbol{V}\boldsymbol{x}) \Big)}_{M \times 1} \times \underbrace{\boldsymbol{x}^\top}_{1 \times d} \in \mathbb{R}^{M \times d}$$

Summary of Backpropagation: Forward pass

Forward pass: Feed data sample $\boldsymbol{x}_i$ into the network, use the current parameters $\boldsymbol{V}$ and $\boldsymbol{W}$ to compute and store

- $\boldsymbol{V}\boldsymbol{x}_i$
- $\boldsymbol{z}_i = \sigma(\boldsymbol{V}\boldsymbol{x}_i)$
- $\boldsymbol{W}\boldsymbol{z}_i$
- $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = h(\boldsymbol{W}\boldsymbol{z}_i), \ \ \ell_i(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2$



Summary of Backpropagation: Backward pass

Backward pass:

Compute the gradient of the second layer

- $\boldsymbol{\delta}_i = 2(h(\boldsymbol{W}\boldsymbol{z}_i) - \boldsymbol{y}_i) \odot h'(\boldsymbol{W}\boldsymbol{z}_i)$
- $\frac{\partial \ell_i}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_i \boldsymbol{z}_i^\top$

Backpropagate the gradient to $\boldsymbol{z}_i$:

- $\frac{\partial \ell}{\partial \boldsymbol{z}_i} = \boldsymbol{W}^\top \boldsymbol{\delta}_i$

Compute the gradient of the first layer:

- $\frac{\partial \ell_i}{\partial \boldsymbol{V}} = \left(\frac{\partial \ell}{\partial \boldsymbol{z}_i} \odot \sigma'(\boldsymbol{V}\boldsymbol{x}_i) \right) \boldsymbol{x}_i^\top$

Thus, backpropagation is an efficient way of computing the gradient of NN learning problem.

Given the computed gradient, we can apply gradient-based training algorithm for training NN. $\rightsquigarrow$ Which algorithm should we choose? Gradient descent or accelerated gradient descent? Next lecture.