Comprehensive Review: Neural Networks Handout

DSC 255 - Machine Learning Fundamentals

Contents

1 Overview of Feedforward Neural Networks

A feedforward neural network is a layered model where inputs are passed forward through hidden layers to produce an output. Each hidden unit applies a nonlinear function to a weighted sum of its inputs.

2 Network Architecture

- Input layer: Receives feature vector x
- Hidden layers: Apply transformations with weights and biases
- Output layer: Produces prediction y

Each hidden unit computes:

$$h = \sigma(w_1 z_1 + w_2 z_2 + \cdots + w_m z_m + b)$$

3 Common Activation Functions

(a) **Threshold (Heaviside)**:

$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

(b) **Sigmoid**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(c) **Tanh**:

$$\sigma(z) = \tanh(z)$$

(d) **ReLU (Rectified Linear Unit)**:

$$\sigma(z) = \max(0, z)$$

4 Why Use Nonlinear Activations?

Without nonlinearities, the entire network reduces to a single linear transformation:

$$h_2 = W_2 W_1 x$$

Nonlinear functions introduce expressive power and allow modeling of complex relationships.

5 Output Layer and Softmax

For classification with k labels:

- The final layer produces real-valued scores $y_1, \ldots, y_k$
- The softmax function converts them to probabilities:

$$\Pr(\text{label } j) = \frac{e^{y_j}}{\sum_{i=1}^{k} e^{y_i}}$$

6 Universal Approximation Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function. Then there exists a neural network with one hidden layer that approximates f arbitrarily well.

- One hidden layer may require many nodes
- Using multiple layers can reduce the number of required nodes per layer

7 Loss Function for Training

For classification with k labels, define:

$$L(W) = -\sum_{i=1}^{n} \log \Pr_W(y^{(i)} \mid x^{(i)})$$

- This is the **cross-entropy loss**
- W are all weights and biases in the network

8 Optimization Techniques

Gradient Descent Variants

- **Batch Gradient Descent**: Full dataset per update
- **Stochastic Gradient Descent (SGD)**: Single data point per update
- **Mini-batch SGD**: A fixed-size subset per update (widely used)

9 Chain Rule for Gradients

- If $h(x) = g(f(x))$, then:
$$h'(x) = g'(f(x)) \cdot f'(x)$$
- For nested functions:
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

10 Backpropagation in a Simple Chain

In a network with one node per hidden layer:

$$h_i = \sigma(w_i h_{i-1} + b_i)$$

- To update w_i, compute:
$$\frac{dL}{dw_i} = \frac{dL}{dh_i} \cdot \sigma'(w_i h_{i-1} + b_i) \cdot h_{i-1}$$
- For previous layer:
$$\frac{dL}{dh_i} = \frac{dL}{dh_{i+1}} \cdot \sigma'(w_{i+1} h_i + b_{i+1}) \cdot w_{i+1}$$

11 Backpropagation Algorithm

1. **Forward Pass**: Compute h_i for all layers
2. **Backward Pass**: Use the chain rule to compute $\frac{dL}{dh_i}$ from output layer to input
3. Update each parameter using its gradient

12 PyTorch Code Example

Model Initialization

```
d, H = 2, 8
model = torch.nn.Sequential(
    torch.nn.Linear(d, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, 1),
    torch.nn.Sigmoid()
)
lossfn = torch.nn.BCELoss()
```

Training Step

```
ypred = model(x)
loss = lossfn(ypred, y)
model.zero_grad()
loss.backward()
with torch.no_grad():
    for param in model.parameters():
        param -= eta * param.grad
```

13 Summary

- Feedforward networks compute layer-wise transformations using nonlinearities.
- They can approximate any continuous function with enough capacity.
- Softmax is used for output probabilities in classification.
- Training involves minimizing cross-entropy loss using variants of gradient descent.
- Gradients are computed efficiently using backpropagation.