Chapter 8

Artificial Neural Network

Contents

8 Artificial Neural Network **1**
8.1 ANN Properties 2
8.2 Perceptron 2
8.3 Multi-layer Perceptron 3
8.4 Activation Functions 3
8.5 Feed-forward MLP 4
8.6 Forward Pass 5
8.7 Gradient Descent 5
8.8 Chain Rule 5
8.9 Backward Pass 5
8.10 Backpropagation using Sigmoid 6
8.11 Backpropagation Example 6
8.12 Backpropagation with Batch Gradient Descent 7
8.13 Training 7
8.14 Dropout 7
8.15 Exploding Gradient Problem 8
8.16 Dying ReLU Problem 8

8.1 ANN Properties

- ANN is much like a black-box.
- Many neuron-like threshold switching units.
- Many weighted interconnections amount units.
- Highly parallel and distributed process.

8.2 Perceptron

Linear function: $f : X \to Y$

$$\begin{aligned} f(\mathbf{x}) &= w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d \\ &= w_0 + \sum_{j=1}^{d} w_jx_j \end{aligned}$$

$w_0, w_1, \ldots, w_d$ are weights

8.3 Multi-layer Perceptron

- Multi-layer perceptrons (MLPs) were designed to overcome the computational limitation of a single threshold element (perceptron).
- The idea is to stack several layers of threshold elements, each layer using the output of the previous layer as input.
- A feed-forward MLP network defines a mapping:

$$y = f(x; \theta)$$

- Functions are composed in a chain:

$$f(x) = f_3(f_2(f_1(x)))$$

- Input layer: The process starts with the input layer, which receives the input data. Each neuron in this layer represents a feature of input data.
- Weights and Biases: Connections between neurons have associated weights, which are learned during the training process and is crucial to capture patterns in the data.
- Hidden Layers: The neurons in these layers perform computations on the inputs. The output of each neuron is calculated by applying a weighted sum of its inputs (from the previous layer).
- Activation Functions: The activation function is crucial as it introduces non-linearity into the model, allowing it to learn more complex functions. ...

8.4 Activation Functions

- Non-linearity: This is the most fundamental property; activation functions introduce nonlinearity into the network. This is important because real-world relationships and patterns are rarely linear.
- ...
- Monotonicity: A monotonic activation function either strictly increases or strictly decreases as input values change. This property ensures that as inputs change, the neuron's output moves in a consistent direction.
- Continuity: A continuous activation function produces smooth and continuous changes in output as inputs change slightly. This property helps in smooth gradient computations for updating weights during the learning process.

- Differentiability: Differentiability is essential for gradient-based optimization algorithms like backpropagation. Activation functions that are differentiable across their domain allow gradients to be computed for weight updates during training.
- Sparsity: Some activation functions promote sparsity by having their outputs be zero for a large portion of input space. This can be beneficial in reducing the complexity of neural networks.

Rectified Linear Units (ReLU):

$$\begin{aligned} g(x) &= \max\{0, x\} \\ g'(x) &= \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \end{aligned} \tag{8.1}$$

Sigmoid:

$$\begin{aligned} g(x) &= \frac{1}{1+e^{-x}} \\ g'(x) &= g(x)\left(1 - g(x)\right) \end{aligned} \tag{8.2}$$

Hyperbolic tangent tanh

$$\begin{aligned} g(x) &= \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \\ g'(x) &= 1 - g(x)^2 \end{aligned} \tag{8.3}$$

Table 8.1: Activation Functions

Activation Functions	**Sigmoid**	**tanh**	**ReLU**
Range	$(0, 1)$	$(-1, 1)$	$[0, \infty)$
Vanishing Gradient Problem	Yes	Yes	No
Nature	Non-Linear	Non-Linear	Linear
Zero Centered Activation Function	No	Yes	No
Symmetric Function	No	Yes	No

8.5 Feed-forward MLP

- The weights of the neural network connections are repeatedly adjusted to minimize the difference between the actual output and desired output.
- Aims to minimize the loss function by adjusting the network's weights and biases. The loss function gradients determine the level of adjustment with respect to parameters like activation function, weights, bias, etc.
- The forward step, given the input, computers the output layer-by-layer, starting with the input layer.
- The backward step calculates the error in the output and propagates it backwards; then update the weights layer-by-layer, starting from the output layer.

8.6 Forward Pass

- In this example, we will be using a three-layer neural network with the layers being the input, hidden, and output layers.
- The activation …

8.7 Gradient Descent

- Goal of NN: Given n training samples (x_i, y_i), find w to minimize:

$$E[w] = \frac{1}{2n} \sum_{i=1}^{n} (y_i - o_i)^2 \tag{8.4}$$

- …

8.8 Chain Rule

- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$
- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$

8.9 Backward Pass

- Function 1 (error):

$$\begin{aligned} \frac{\partial E_h}{\partial o_h} &= -(y_h - o_h) \\ &= o_h - y_h \end{aligned}$$

- Function 2 (differentiable activation function):

$$\begin{aligned} \frac{\partial o_h}{\partial net_h} &= \frac{e^{-net_h}}{(1 + e^{-net_h})^2} \\ &= \frac{1}{1 + e^{-net_h}} \times \left(1 - \frac{1}{1 + e^{-net_h}}\right) \\ &= o_h(1 - o_h) \end{aligned}$$

- Function 3 (linear gate):

$$\frac{\partial net_h}{\partial w_{jh}} = x_j$$

Output neuron backpropagation error:

$$\begin{aligned}\delta_1^y &= \frac{\partial E_1}{\partial o_1^y}\frac{\partial o_1^y}{\partial net_1} \\ &= (o_1^y - y_1)o_1^y(1 - o_1^y)\end{aligned} \tag{8.5}$$

Hidden neuron backpropagation error:

$$\delta_1^{h_2} = \delta_1^{h_2}(1 - o_1^{h_2})\sum_{m=1}^{3}\delta_m^y w_{h_2^1 y_m} \tag{8.6}$$

Weight updates:

$$\begin{aligned}w_{h_2^1 y_1} &= w_{h_2^1 y_1} - \eta\delta_1^y o_1^{h_2} \\ w_{h_1^1 h_2^1} &= w_{h_{1}^{1}{}_{2}^{1}} - \eta\delta_1^{h_2} o_1^{h_1}\end{aligned}$$

8.10 Backpropagation using Sigmoid

Repeat for each training example until end of training epoch:

1. In the forward pass, compute the outputs o of each neuron
   (a) The outputs for the input layer neurons stay unchanged.
   (b) The outputs for the hidden and output layer neurons are computed using the sigmoid activation function.
2. In the backward pass, propagate the errors δ from output layer
   (a) For each output neuron $k : \delta_k = (o_k - y_k)o_k(1 - o_K)$
   (b) For each hidden neuron: $h : \delta_h = o_h(1-o_h)\sum_{m=1}^{|h'|}\delta_m w_{hm}$, where h' is the subsequent layer.
3. Update every weights $w_{ij} \leftarrow w_{ij} - \eta\delta_j o_i$ where w_{ij} denotes the weight between nodes i and j.

8.11 Backpropagation Example

- x are the inputs, h are the neurons with sigmoid activation, y are the outputs.
- The initial weights of all the layers are shown in the figure.
- Our objective is to learn ... $\eta = 1$.

8.12 Backpropagation with Batch Gradient Descent

- The backpropagation algorithm we covered so far, as well as the example we practiced, both used stochastic gradient descent for weight updates i.e., the weights are updated for every training instances.
- In backpropagation using bgd, the weights are updated ...

8.13 Training

Backpropagation is a gradient estimation method used to train NNs –

- No guarantee of convergence since NNs form non-convex functions with multiple local minima.
- Many epochs (tens of thousands) may be needed for adequate training.
- Large data sets may require many hours of CPU.
- Termination criteria: Number of epochs, threshold on training set error, early stopping, increased error on validation set.
-
- Underfitting –
  - Using too few hidden neurons in the NN.
  - Inadequate or less data to train the NN on.
- Overfitting –
  - Training the NN over too manu epochs.
  - Using too many hidden layers in the NN.
  - Using too many neurons in a hidden layer in the NN.

8.14 Dropout

- Large weights in NN are a sign or a more complex network that has overfit the training data.
- Probabilistically dropping out nodes in the network is a simple and effective regularization method.

8.15 Exploding Gradient Problem

8.16 Dying ReLU Problem

- A dying RELU always outputs the same value i.e., 0 on any input value.
- This condition is known as the dead state of the ReLU neurons.
- In this state, it is difficult to recover because the gradient of 0 is 0.
- This becomes a problem when most of the input ranges are negative, or the derivative of the ReLU function is 0.
- Can be caused by a high learning rate or a large negative bias.
- Using Leaky ReLU instead can resolve the dying ReLU problem.

$$\begin{aligned} g(x) &= \max\{0.01x, x\} \\ g'(x) &= \begin{cases} 1, & x \geq 0 \\ 0.01, & x < 0 \end{cases} \end{aligned} \tag{8.7}$$