CS475 - Machine Learning - Outline

Robert B. Heckendorn
University of Idaho

March 19, 2018

1 Neural Networks

Neural networks (NN) are a supervised learning technique. It produces a model that takes n dimensional vectors of real numbers as input and returns an m dimensional real number vectors as output:

$$\mathbb{R}^n \to \mathbb{R}^m$$

For example: **features** like loan worthiness $\to$ confidence in giving a loan or repaying loan.

1. inputs (vector)
2. weights (vector)
3. basic unit of computation: sum + threshold function (aka activation or transfer function)
   (a) output is reals.

Normalizing input and output to match scale between dimensions

$$[0,1]^n \to [0,1]^m \quad \text{or} \quad [-1,1]^n \to [-1,1]^m$$

Goal: What we want is to adjust the weights and threshold functions to get desired results (supervised learning)

1.1 Sigmoids

A **threshold function** is often some kind of **sigmoid function** many types of sigmoid (represents a decision) without it, it is simply a linear function of inputs

- must approach +/- 1 or 0/1 depending on application

- always slope toward
- nice math properties for some algorithms to work

Sigmoid functions with slope of 1 at 0:

- x
- $\text{atan}(2 * x)/2$
- $\text{gd}(2 * x)/2$
- $\dfrac{2}{1 + \exp(-2 * x)} - 1 = \tanh(x)$
- $\text{erf}(x)\sqrt{\pi}/2$

The function gd is the Gudermannian function.
Sigmoid functions with max of 1

- x
- $\dfrac{\text{atan}(2 * x)}{\pi/2}$
- $\text{gd}(2 * x)/(\pi/2)$
- $\dfrac{2}{1 + \exp(-2 * x)} - 1 = \tanh(x)$
- $\text{erf}(x)$

slopes: $1, 4/\pi, 4/\pi, 1, 2/\sqrt{\pi}]$
Can be scaled to $[0, 1]$
Initialization of the weights should try to put the weights scattered randomly in the decision zone of the sigmoid curve (area of maximum slope). This would be 0.5 for the range $(0, 1)$ and 0 for the sigmoid in the range $(-1, +1)$.

1.2 The Perceptron

$y = f(\vec{x} \cdot \vec{w})$ where $\cdot$ is dot product and f is threshold function with proper $\vec{w}$, $\vec{y}$ will approach $\vec{t}$.

Assume $\vec{x}$ is n dimensional data. Adding a -1 entry onto $\vec{x}$ and we have a **bias** value! Now $\vec{x} \in \mathbb{R}^{n+1}$.

This can be done with multiplying a row vector:

DIM: $\vec{x}$ is $(1 \times n+1)$
DIM: $\vec{w}$ is $(n+1 \times 1)$
DIM: $\vec{y}$ is (1×1)

Assume that you now instead have d vectors of vector data in matrix X. This is a $(d \times n+1)$ matrix X. Now:

DIM: X is $(d \times n+1)$
DIM: $\vec{w}$ is $(n+1 \times 1)$

$$f(X\vec{w}) = \vec{y}$$

DIM: $\vec{y}$ is $(d \times 1)$

$$\vec{y} = f(X\vec{w}) \text{ where error} = \vec{y} - \vec{t} \to \vec{0}$$

1.3 Vector of Targets or Batch Mode

Given training vectors X and a vector of targets $\vec{t}$ we want to find an m dimensional vector $\vec{y}$ for each $\vec{x_i}$.

$$Y = f(XW) \text{ where error} = Y - T \to 0$$

DIM: X is $(d \times n+1)$
DIM: W is $(n+1 \times m)$
DIM: Y is $(d \times m)$

Consider data row r

$$X_{r*}W_{*k} \to Y_{rk}$$

where there is an error: $T_{rk} - Y_{rk} \neq 0$

Then we need to adjust some of W_{ik} (input from X_{ri} to node producing Y_{rk})

$$X_{ri}(T_{rk} - Y_{rk}) \to \Delta W_{ik}$$

gets the right direction to change W denoted by Δ.

Do this for all rows r:

$$X_{*i}(T_{*k} - Y_{*k}) \rightarrow \Delta W_{ik}$$

but the dimension are not right!! So really we want the multiply to go this way:

$$X_{i*}{}^{\intercal}(T_{*k} - Y_{*k}) \rightarrow \Delta W_{ik}$$

This gives us the update formula we are looking for

$$W+ = \alpha X^{\intercal}(T - Y)$$

where α is the learning rate.

DIM: W is $(n+1 \times m)$

DIM: X is $(d \times n+1)$
DIM: T is $(d \times m)$
DIM: Y is $(d \times m)$

Remembering the transpose, the dimensions check out!

Problems with parallelization or **batch** mode. doing it all at once poor optimization

Small learning rate keeps stable but slows convergence under repeated training.

1.4 Limits of Single Layer Network

Each single neuron creates a decision boundary

OR vs XOR

OR is easy

XOR is hard because a single line does not divide the space

change of variables can help but then you have to get the right change of variable we'll look at that later

1.5 Multilayer Neural Network

This is the batch mode version. Assume h hidden nodes without the bias node in the hidden layer. Assume d data vectors of dimension n. The bias node is added and then H is computed. Compare with the algorithm on page 78.

Add the bias to the inputs X by adding a -1 column. $X \rightarrow X^{+}$.

DIM: X is $(d \times n)$

$$H = f(X^{+}V)$$

DIM: X^+ is $(d \times n+1)$
DIM: V is $(n+1 \times h)$
DIM: H is $(d \times h)$

Add the bias to the hidden layer by adding a -1 column. $H \rightarrow H^+$.

$$Y = f(H^+ W)$$

DIM: H^+ is $(d \times h+1)$
DIM: W is $(h+1 \times m)$
DIM: Y is $(d \times m)$

Now the backward propagation phase (backprop):

$$W_\Delta = (Y - T) * Y * (1 - Y)$$

DIM: Y is $(d \times m)$
DIM: T is $(d \times m)$

DIM: W_Δ is $(d \times m)$

where $*$ is element by element multiply.

$$H_\Delta = H^+ * (1 - H^+) * (W_\Delta W^\intercal)$$

DIM: H^+ is $(d \times h+1)$
DIM: W_Δ is $(d \times m)$
DIM: W is $(h+1 \times m)$

DIM: H_Δ is $(d \times h+1)$

$$W- = \alpha {H^+}^\intercal W_\Delta$$

where α is the learning rate.

DIM: H^+ is $(d \times h+1)$
DIM: W_Δ is $(d \times m)$

DIM: W is $(h+1 \times m)$

Subtract the bias to the hidden layer adjustment. $H_\Delta \rightarrow H_\Delta^-$. The signal for the adjustment to the hidden layer does not effect the adjustment of the first layer!

$$V- = \alpha {X^+}^\intercal H_\Delta^-$$

DIM: X^+ is $(d \times n+1)$
DIM: H_Δ^- is $(d \times h)$

DIM: V is $(n+1 \times h)$