Regression and Classification

This note introduces Linear Neuron and Stochastic Gradient Descent (SGD) and Gradient Descent (GD) for linear neuron

In Artificial Intelligence, Machine Learning

Primarily, neural networks are used to *predict* output **labels** from input **features**.

Prediction tasks can be classified into two categories:

- **Regression**: The labels are continuous (age, income, height, etc.)
- **Classification**: The labels are discrete (sex, digits, type of flower, etc.)

Training finds networks weights and biases that are optimal for **prediction** of labels from features.

Linear Neuron

Synaptic input u to a neuron is given by $u = w^T x + b$ A linear neuron has a *linear activation function*. That is, $y = f(u) = u$

A linear neuron with weights **w** = $(w_1 \dots w_n)^T$ and bias b has an output: $y = w^T x + b$ where input $x = (x_1\ x_2\ \dots x_n)^T \in R^n$ and output $y \in R$

Representing a dependent (output) variable as a linear combination of independent (input) variables is known as **Linear Regression**.

The output of a linear neuron can be written as $y = w^T x + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$ where $w^T x$ can be expanded via matrix multiplication and $x_1, x_2, \dots, x_n$ are the inputs. That is, a linear neuron performs linear regression and the weights and biases (that is, b and $w_1, \dots, w_n$) acts as regression coefficients. The above function forms a **hyperplane** in Euclidean space R^n.

Given a training examples $\{(x_n, d_p)\}_{p=1}^{p}$ where input $x_p \in R^n$ and target $d_p \in R$, training a linear neuron finds a linear mapping ϕ: $R\hat{}n \rightarrow R\$\ given\ by$: $\$y = w\hat{}Tx + b$

Stochastic Gradient Descent (SGD) for linear neuron

The **cost function** J for regression is usually given as the *square error* (s.e.) between neuron outputs and targets.

Given a training pattern $(\$x, d\$)$, $\frac{1}{2}$ square error cost J is defined as $J = \frac{1}{2}(d - y)^2$where y is neuron output for input pattern x and d is the target label.

$$y = w^T x + b$$

The $\frac{1}{2}$ in the cost function is introduced to simplify learning equations and does not affect the optimal values of the parameters (weights and bias).

$$J = \frac{1}{2}(d - y)^2$$

$$y = u = w^T x + b$$

(A) $\frac{\partial J}{\partial u} = \frac{\partial J}{\partial y} = -(d - y)$

- For the case of a linear activation, $y = u$ since the activation does not do anything apart from scaling the value.

(B) $\nabla_w J = \frac{\partial J}{\partial w} = \frac{\partial J}{\partial u}\frac{\partial J}{\partial w}$

$$u = w^T x + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

(C)

$$\frac{\partial u}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial u}{\partial w_1} \\ \frac{\partial u}{\partial w_2} \\ \vdots \frac{\partial u}{\partial w_n} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots x_n \end{pmatrix} = \mathbf{x}$$

$$\frac{\partial u}{\partial w} = x$$

From the expression of $u = w^T x + b$, if we do partial differentiation of each $w_i \in w^T$,

when we are differentiating u w.r.t w_i, $\forall i \neq j$, the individual term $\frac{\partial u}{\partial w_i} w_j x_j = 0$, whereas

for $i == j$, $\frac{\partial u}{\partial w_i} w_i x_i = x_i$ since w_1 is raised to power 1.

Substituting (A) and (C) in (B),

(D)$\nabla_w J = -(d - y)x$

Similarly, since $\frac{\partial u}{\partial b}$ = 1,

$$\nabla_b J = \frac{\partial J}{\partial u}\frac{\partial u}{\partial b} = -(d - y)$$

Gradient learning equations:

$$w \leftarrow w - \alpha \nabla_w J$$

$$b \leftarrow b - \alpha \nabla_b J$$

$\nabla_w J$ means the change in w that will result in the change in J, hence, we want to reduce the change in J, hence the coefficient of it is negative to reduce the change in J, i.e. if there is no change in J, is it likely to be in the minimum point.

By substituting from (D) and (E), SGD equations for a linear neuron are given by

$$w \leftarrow w - \alpha(d - y)x$$

$$b \leftarrow b - \alpha(d - y)$$

SGD algorithm for linear neuron

` **Given a training dateset** ` $\{(x_p, d_p)\}_{p=1}^{p}$

` **Set learning parameter** ` α

` **Initialize** ` w ` **and** ` b

` **Repeat unit convergence** `

` **For every training pattern** ` (x_p, d_p) :

$$y_p = w^T x_p + b$$

$$w \leftarrow w + \alpha(d_p - y_p)x_p$$

$$b \leftarrow b + \alpha(d_p - y_p)$$

To calculate if training has converged, we find the mean square error (m.s.e): $m.s.e = \frac{1}{6}\sum_{p=1}^{6}(d_p, y_p)^2$

m.s.e will converged to zero but it will usually not be zero, this is due to

- the learning rate value (if we decrease it as the number of epoch increase, we might reach zero)
- The noise in the dataset

Gradient Descent (GD) for linear neuron

Given a training dataset $\{(x_p, d_p)\}_{p=1}^{p}$, cost function J is given by the sum of square error (s.s.e.):

$$J = \frac{1}{2}\sum_{p=1}^{P}(d_p, y_p)^2$$

where y_p is the neuron output for input pattern x_p.

(F) $J = \sum_{p=1}^{p} J_{\square} where\, J_p = \frac{1}{2}(d_p - y_p)^{\square}$ is the square error for the p^{th} pattern.

From (F):

$$\nabla_w J = \sum_{p=1}^{P} \nabla_w J_p$$

$$= -\sum_{p-1}^{P}(d_p - y_p)x_p \leftarrow \text{from (D)}$$

$$= -\Big((d_1 - y_1)x_1 + (d_2 - y_2)x_2 + \cdots + (d_p - y_p)x_p\Big)$$

$$= -(x_1\ x_2\ \ldots x_p)\begin{pmatrix}(d_1 - y_1)\\(d_2 - y_2)\\\vdots (d_p - y_p)\end{pmatrix}$$

$$= -\mathbf{X^T}(\mathbf{d} - \mathbf{y}) \leftarrow (\mathrm{G})$$

where $\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots\ x_p^T \end{pmatrix}$ *is the data matrix,* $\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots\ d_p \end{pmatrix}$ *is the target vector, and* $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots\ y_p \end{pmatrix}$ is the output vector.

Similarly, $\nabla_b J$ can be obtained by considering inputs of +1 and substituting a vector of +1 in (G).

$\nabla_b J = -1_p^T(d - y)$ where $\mathbf{1_p} = \begin{pmatrix} 1 \\ 1 \\ \vdots\ 1 \end{pmatrix}$ has P element of 1.

The output vector y for the batch of P patterns is given by

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots\ y_p \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots\ u_p \end{pmatrix} = \begin{pmatrix} x_1^T w + b \\ x_2^T w + b \\ \vdots\ x_p^T w + b \end{pmatrix} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots\ x_p^T \end{pmatrix}\mathbf{w} + b\begin{pmatrix} 1 \\ 1 \\ \vdots\ 1 \end{pmatrix} = \mathbf{X}\mathbf{w} + b\mathbf{1_p}$$

Substituting (G) and (H) in gradient descent equations:

$$w \leftarrow w - \alpha\nabla_w J$$

$$b \leftarrow b - \alpha\nabla_b J$$

We get GD learning equations for the linear neuron as

$$w \leftarrow w - \alpha\mathbf{X^T}(\mathbf{d} - \mathbf{y})$$

$$b \leftarrow b - \alpha\mathbf{1_p^T}(\mathbf{d} - \mathbf{y})$$

and α is the learning factor

Where: $y = \mathbf{X}\mathbf{w} + b\mathbf{1_p}$

`Given a training dataset` $(\mathbf{X}, d)$

`Set learning parameter` α

`Initialize` $\mathbf{w}$ **`and`** b

`Repeat until convergence:`

$$y = \mathbf{X}\mathbf{w} + b\mathbf{1_p}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\mathbf{X^T}(\mathbf{d} - \mathbf{y})$$

$$b \leftarrow b + \alpha\mathbf{1_p^T}(\mathbf{d} - \mathbf{y})$$