Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2

Winter Semester 2018-2019

Lesson 6

22 February 2019

Perceptrons, Regression and Gradient Descent

Outline

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
$\vec{y}$	A dependent variable to be estimated.
$\hat{y} = g(\vec{X}, \vec{w})$	A model that predicts $\vec{y}$ from $\vec{X}$.
$\vec{w}$	The parameters of the model.
$\{\vec{X}_m\}$ $\{y_m\}$	Training samples for learning.
M	The number of training samples.

$$L(\vec{w}) = \frac{1}{2M} \sum_{m=1}^{M} (y_m - g(\vec{X}_m, \vec{w}))^2 \quad \text{The average Loss for the function } \hat{y} = g(\vec{X}, \vec{w})$$

MSE Loss estimates the cost of errors as the Mean Square Error.

Perceptrons

History

The Perceptron is an incremental learning algorithm for linear classifiers invented by Frank Rosenblatt in 1956. The first Perceptron was a room-sized analog computer that implemented Rosenblatz's learning function for recognition. However, it was soon recognized that both the learning algorithm and the resulting recognition algorithm are easily implemented as computer programs.

In 1969, Marvin Minsky and Seymour Papert of MIT published a book entitled "Perceptrons", that claimed to document the fundamental limitations of the perceptron approach. Notably, they claimed that a linear classifier could not be constructed to perform an "exclusive OR". While this is true for a one-layer perceptron, it is not true for multi-layer perceptrons.

The Perceptron

When the two classes of data can be separated by a linear decision boundary, the data is said to be "separable". The perceptron is an on-line learning algorithm that learns a linear decision boundary (hyper-plane) for separable training data. If the training data is non-separable, the method may not converge, and must be stopped after a certain number of iterations.

Learning can be incremental and "on-line". At any time, new training samples can be used to update the perceptron. The perceptron algorithm uses errors in classifying the training data to update the decision boundary plane until there are no more errors.

Assume a training set of M observations $\{\vec{X}_m\}$ of D features, with indicators variables, $\{y_m\}$ where

$$\vec{X}_m = \begin{pmatrix} x_{1m} \\ x_{2m} \\ \vdots \\ x_{Dm} \end{pmatrix} \text{ and } y_m = \{-1, +1\}$$

The indicator variable, $\{y_m\}$, tells the class label for each sample.
For binary pattern detection,

   $y_m = +1$ for examples of the target class (class 1)
   $y_m = -1$ for all others (class 2)

The Perceptron will learn the coefficients for a linear boundary

$$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \text{ and } b$$

Such that for all training data, $\vec{X}_m$,

$$\vec{w}^T \vec{X}_m + b \geq 0 \text{ for Class 1 and } \vec{w}^T \vec{X}_m + w_0 < 0 \text{ for Class 2.}$$

Note that $\vec{w}^T \vec{X}_m + b \geq 0$ is the same as $\vec{w}^T \vec{X}_m \geq -b$.
Thus b can be considered as a threshold on the product : $\vec{w}^T \vec{X}_m$

The decision function is the sgn() function: $\quad \text{sgn}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$

The algorithm requires a learning rate, α.

A training sample is correctly classified if:

$$y_m \cdot \left(\vec{w}^T \vec{X}_m + b \right) \geq 0$$

The Perceptron Learning Algorithm

The algorithm will continue to loop through the training data until it makes an entire pass without a single miss-classified training sample. If the training data are not separable then it will continue to loop forever.

Algorithm:

$\vec{w}^{(0)} \leftarrow 0$; $b^{(i)} \leftarrow 0$, $\quad \leftarrow 0$;

$R \leftarrow \max \{ \|\bar{X}_m\| \}$

WHILE update DO

    update $\leftarrow$ FALSE;

    FOR $m = 1$ TO M DO

        IF $\quad y_m \cdot \left(\vec{w}^{(i)T} \bar{X}_m + b^{(i)} \right) < 0$ THEN

            update $\leftarrow$ TRUE

            $\vec{w}^{(i+1)} \leftarrow \vec{w}^{(i)} + \alpha \cdot y_m \cdot \vec{X}_m$

            $b^{(i+1)} \leftarrow b^{(i)} + \alpha \cdot y_m \cdot R^2$

            $i \leftarrow i + 1$

        END IF

    END FOR

END WHILE.

The final classifier is: if $\quad \vec{w}^{(i)T} \bar{X}_m + b^{(i)} \geq 0$ then P else N.

If the data is not separable, then the Perceptron will not converge, and continue to loop. Thus it is necessary to have a limit the number of iterations.

The perceptron learning algorithm can be seen as a form of regression.

Regression

Regression is the estimation of the parameters for a function that maps a set of independent variables into a dependent variable.

$$\hat{y} = g(\bar{X}, \vec{w})$$

Where

$\bar{X}$ is a vector of D independent random (feature) variables.
$\hat{y}$ is an estimate for a variable y that depends on $\bar{X}$.
$\hat{y}$ can be a numerical value or a symbolic class label.

and

$g()$ (the model) is a function that maps $\bar{X}$ onto $\hat{y}$
$\vec{w}$ is a vector of parameters for the model.

$\bar{X}$ is upper case because it is a random (unknown) vector.
For $\hat{y}$, the "hat" indicates an estimated value for the target value y
$\delta = y - \hat{y}$ is the error of the estimation.

Regression analysis refers to a family of techniques for estimating the model.

Linear Models

The equation $\vec{w}^T \bar{X} = 0$ is a hyper-plane in a D-dimensional space, that passes through the origin. $\vec{W}$ is the normal to the plane. The plane may be translated to any parallel hyperplane by adding a bias, b. In this case, the linear model has the form

$$\hat{y} = g(\bar{X}, \vec{w}) = \vec{w}^T \bar{X} + b = w_1 x_1 + w_2 x_2 + ... + w_D x_D + b$$

The vector $\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ and b are the "parameters" of the model that relates $\bar{X}$ to $\hat{y}$.

$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ is the normal to the hyperplane and b is a constant term.

It is sometimes convenient to include b in the parameter vector. This can be done with homogeneous coordinates.

Homogeneous Coordinate Notation

With a linear model, it is possible to include the bias in the model vector by adding an additional term to the model $\vec{w}$ and the feature vector $\vec{X}$. This is referred to as Homogeneous Coordinates. Homogeneous coordinates provide a unified notation for geometric operations and is widely used in Computer Vision, Computer Graphics and Robotics.

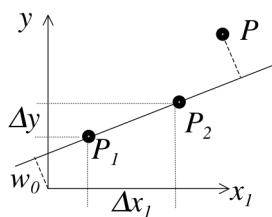The linear model is expressed in homogeneous coordinates as:

$$\hat{y} = g(\vec{X},\vec{w}) = \vec{w}^T \vec{X} \qquad \text{where} \quad \vec{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_D \\ 1 \end{pmatrix} \text{ and } \vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_D \\ b \end{pmatrix}$$

This gives a linear model for a D dimensional space with *D+1* parameters.
For example we can predict Weight from Height as a linear function with a 1d model

$$\hat{y} = g(\vec{X},\vec{w}) = w_1 x_1 + b$$

where $\quad \hat{y} = Weight$, $\quad \vec{X} = \begin{pmatrix} Height \\ 1 \end{pmatrix}$ and $\vec{w}$ are the model parameters

Suppose we have two samples of training data: $X_1 = \begin{pmatrix} x_1^{(1)} \\ 1 \end{pmatrix} \qquad X_2 = \begin{pmatrix} x_1^{(2)} \\ 1 \end{pmatrix}$

We can initialize the model with $\quad w_1 = \dfrac{y^{(2)} - y^{(1)}}{x_1^{(2)} - x_1^{(1)}}$ and $b = -w_1 x_1^{(1)}$

We can then predict weight for any value of height with $\quad \hat{y} = g(\vec{X},\vec{w}) = w_1 x_1 + b$

We can generalize the model to predict Weight from Height and Age. (D=2)

$$\hat{y} = g(\vec{X},\vec{w}) \quad \text{where} \quad \hat{y} = Weight, \quad \vec{X} = \begin{pmatrix} Age \\ Height \\ 1 \end{pmatrix} \text{ and } \vec{W} = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix}$$

are the model parameters, and the surface is a plane that predicts weight, as a function of (age, height).

Gradient Descent

Gradient descent is a popular algorithm for estimating parameters for a large variety of models. Here we will illustrate the approach with estimation of parameters for a linear model $g(\bar{X}, \vec{w})$ for a dependent variable y. This can be extended to estimation of a model for a classifier.

We seek to estimate that parameters $\vec{w}$ for a model expressed in homogenous coordinates, from a training set of M samples $\{\bar{X}_m\}\,\{y_m\}$

$$\hat{y} = g(\bar{X}, \vec{w}) = \vec{w}^T \vec{X} = \begin{pmatrix} w_1 & \cdots & w_D & b \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_2 \\ 1 \end{pmatrix}$$

To determine the model, we will iteratively refine the model to reduce the errors.

Loss Function

For any training data sample $\bar{X}_m$ for which the true value of the model, y_m, is known, the error is the difference between the true value and the estimated value provided by the model.

$$\delta_m = y_m - g(\bar{X}_m, \vec{w}) = y_m - \vec{w}^T \vec{X}_m$$

The average error for the training set is

$$\delta = \frac{1}{M} \sum_{m=1}^{M} \delta_m = \frac{1}{M} \sum_{m=1}^{M} (y_m - \vec{w}^T \vec{X}_m)$$

To estimate $\vec{w}$, we estimate a "Loss" function as the "Mean Square Error" (MSE).

The loss for an individual sample is

$$L(\bar{X}_m, \vec{w}) = \frac{1}{2} \left(y_m - g(\bar{X}_m, \vec{w}) \right)^2$$

where we have included the term $\dfrac{1}{2}$ to simplify the algebra.

The average loss for the all M Training samples is:

$$L(\vec{w}) = \frac{1}{2M} \sum_{m=1}^{M} (y_m - g(\bar{X}_m, \vec{w}))^2$$

The model parameters $\vec{w}$ that minimizes the loss are said to be "optimum". For all other parameters, the increases. That is the derivative of the loss is zero at the optimum and positive for other values.

The gradient of the loss is the derivative of the loss with respect to each model parameter.

$$\vec{\nabla}L(\bar{X}_m,\vec{w}) = \frac{\partial L(\bar{X}_m,\vec{w})}{\partial\vec{w}}$$

the gradient tells us how much to correct the model for each training sample.
For the optimum model, the gradient is zero.

$$\vec{\nabla}L(\bar{X}_m,\vec{w}) = 0$$

For a linear model, the loss for a training sample is

$$L(\bar{X}_m,\vec{w}) = \left(y_m - \vec{w}^T\vec{X}_m\right)^2 = \left(\vec{y}_m - \vec{w}^T\vec{X}_m\right)\left(y_m - \vec{w}^T\vec{X}_m\right) = \left(y_m^2 - 2y_m\vec{w}^T\vec{X}_m + (\vec{w}^T\vec{X}_m)^2\right)$$

The gradient of the loss is:

$$\vec{\nabla}L(\bar{X}_m,\vec{w}) = \frac{\partial L(\bar{X}_m,\vec{w})}{\partial\vec{w}} = -2y_m\vec{X}_m + 2(\vec{w}^T\vec{X}_m)\vec{X}_m) = \left(-y_m + \vec{w}^T\vec{X}_m\right)\vec{X}_m = -\left(y_m - \vec{w}^T\vec{X}_m\right)\vec{X}_m$$

which we can also write as $\vec{\nabla}L(\bar{X}_m,\vec{w}) = -\delta_m\vec{X}_m$ because $\delta_m = (y_m - \vec{w}^T\vec{X}_m)$

We can use the gradient to "correct" the model parameters for each training sample.

$$\Delta\vec{w}_m = -\delta_m\vec{X}_m$$

The correction is weighted by a very small learning rate "η" to stabilize learning.

$$\vec{w}^{(i)} = \vec{w}^{(i-1)} - \eta\Delta\vec{w}_m$$

However, the values for any individual training sample are random. Individual training samples will send the model in arbitrary directions. While, updating with each sample will eventually converge, but tends to be inefficient. Thus, it is generally more efficient to correct the model with the average of a large set of training samples. This is called in "batch mode".

$$\Delta \vec{w} = \frac{1}{M} \sum_{m=1}^{M} \Delta \vec{w}_m = \frac{1}{M} \sum_{m=1}^{M} \delta_m \vec{X}_m$$

we then update the model with the average error.

$$\vec{w}^{(i)} = \vec{w}^{(i-1)} - \eta \Delta \vec{w}$$

Each pass through the training data is referred to as an "epoch". Gradient descent may require many epochs to reach an optimal (minimum loss) model. Even better is to divide the training data into "folds" and update with the average of each fold.

Gradient Descent Algorithm (Batch mode)

Initialization: $(i=0)$ and set $\vec{w}^{(0)}$ to some initial (random). values, for example, for a linear model, using $m=1, 2$ $\qquad w_d^{(0)} = \dfrac{y_2 - y_1}{x_{d2} - x_{d1}}$, $b^{(0)} = y_2 - y_1$

Choose some value for the learning rate η (typically 0.001)

WHILE $\left\| L(\vec{w}^{(i+1)}) - L(\vec{w}^{(i)}) \right\| > \varepsilon$ DO

$\qquad i \leftarrow i+1$

$$\Delta \vec{w} = \frac{1}{M} \sum_{m=1}^{M} \vec{\nabla} L(\vec{w}^{(i-1)}, \vec{X}_m)$$

$$\vec{w}^{(i)} = \vec{w}^{(i-1)} - \eta \Delta w$$

$$L(\vec{w}^{(i)}) = \frac{1}{2M} \sum_{m=1}^{M} (y_m - g(\vec{X}_m, \vec{w}^{(i)}))^2$$

END

The algorithm halts when the change in $\Delta L(\vec{w}^{(i)})$ becomes small:

$$\left\| L(\vec{w}^{(i)}) - L(\vec{w}^{(i-1)}) \right\| < \varepsilon$$

For some small constant ε. (or after "N" iterations.)

It is common to make many iterations through the training data. Each pass through all the training data is called an epoch.

Practical Considerations for Gradient Descent
The following are some practical issues concerning gradient descent.

Feature Scaling
For a training set $\{\vec{X}_m\}$ of M training samples with D values, if the individual features do not have a similar range of values, than large values will dominate the gradient. Make. One way to assure sure that features have similar ranges is to normalize the training data.

A simple technique is to normalize the range of sample values.

For example, $$\forall_{m=1}^{M} : x_{dm} := \frac{x_{dm} - \min(x_d)}{\max(x_d) - \min(x_d)}$$
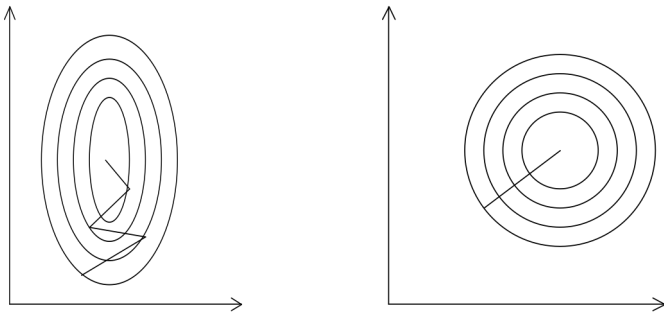
After estimating the model, use μ_d and σ_d to project the data back to the original space.

Stochastic Gradient Descent
Batch gradient descent will generally converge much more efficiently than updating with each sample. However, gradient descent assumes that the loss function is convex. This depends on the training data. In many real examples, the Loss function is not completely convex but contains local minimum.

Batch gradient descent may efficiently converge to a local minimum and become stuck. This can be avoided with stochastic gradient descent.

With Stochastic gradient descent we choose a single training sample randomly and update with that sample.

Logistic Regression

The regression algorithm described above estimated a linear model for a value.
For pattern detection (or classification) we are estimating a class label. This requires a decision.

The classification function $R(\vec{X})$ can typically be decomposed into two parts:

$$\hat{C}_k \leftarrow R(\vec{X}) = d\left(\vec{g}\left(\vec{X}\right)\right)$$

where $\vec{g}(\vec{X})$ is a discriminant function and $d\left(\vec{g}\left(\vec{X}\right)\right)$ is a decision function.

$\vec{g}(\vec{X}): \quad \vec{X} \to \mathbb{R}^K$ A discriminant function that maps features to discriminants

$d\left(\vec{g}\left(\vec{X}\right)\right): \ \mathbb{R}^K \to \hat{C}_k \in \{C_k\}$ A decision function that maps discriminants to a class (or an ordered list of classes).

In order to use gradient descent, the decision function must be differentiable.
The sgn() function is not differentiable.

A popular choice for decision function is the sigmoid: $\quad f(z) = \dfrac{1}{1 + e^{-z}}$
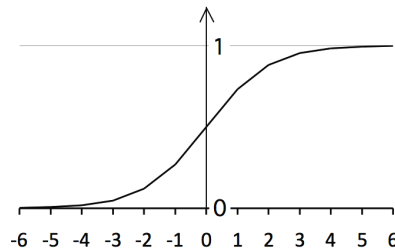
This function is useful because the derivative is: $\quad \dfrac{df(z)}{dz} = f(z)(1 - f(z))$

This gives a decision function: if $f(g(\vec{X})) > 0.5$ POSITIVE else NEGATIVE