Machine Learning

Author: Pannenets.F

Date: October 6, 2020

Je reviendrai et je serai des millions. — «Spartacus»

Introduction

Nothing.

Pannenets F October 6, 2020

Contents

Part I

part

Chapter 1 welcome

Definition 1.1 (Machine Learning) *Get computers to learn without being explicitly programmed.*

Neural network just mimics human's brain's working.

Applications:

- Database mining
- Application that human cannot do (too immediate, big)
- Self-customizing programs
- Understand human activities.

Chapter 2 What is Machine Learning?

2.1 Introduction

Definition 2.1 (Machine Learning) *Field of study that gives computers thr ability to learn without being explicitly programmed. (older, informal)*

Improve performance from task experience. (more modern)

- *A **task***
- *Some **experience***
- *Some way to **measure***

Two board type of ML algorithms:

- Supervised Learning
- Unsupervised Learning

Others:

- Reinforcement Learning
- Recommender Systems

2.1.1 Supervised Learning

Supervised learning has some known relationship between input and output.

Supervised learning problems are categorized into **regression** and **classification** problems.

Fitting a straight line or quadratic or 2nd-order curve to a function is a regression. The main idea is to give a continuous solution based on data we have.

Example 2.1 Estimate weekly income of a company.

Estimating some possbility to be certain type or classification is also a supervised learning.

Example 2.2 Classify tumor type based on its size.

When the scale of classes turns to infinity, we need **Support Vector Machine**.

2.1.2 Unsupervised Learning

Unsupervised learning has no specific relationship between input and output. Data has no lables, but the data could be devided into severval clusters.

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

Example 2.3 Two recordings of audio are mixed, and computer should get them departed.

2.2 Linear Regression with One Variable

2.2.1 Model and Cost Function

For a training set, Notation:

- m: number of training example
- x: input variable / features
- y: output variable / target variable

Learning algorithm applis to training set to find a **hypothesis** that could gives back a estimated answer.

The goal of supervised learing is to learn a function (hypothesis) from the training set.

2.2.2 Cost Function

In a univariate problems, we have:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

And the θ_i is so called **Parameters**.

Regression is to minimize the difference between $h(x)$ and y. The cost function defined as:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Of course, for each parameters's vector we can compute its J value. Then we can get the global minimum to get the best fit curve. Later, we will learn an efficient algorithm to find the minimum cost.

2.3 Parameter Learning

Have some function $J(\theta_0, \theta_1)$ and want to minimize it. Here is the outline:

- start with some θ_0, θ_1
- change the parameters until the loss has been minimized

We need a way to minimize loss function.

2.3.1 Gradient Descent

At a parameters' place, take a little baby step to change. And choose the step direction that has the quickest reduction speed.

And the process is to repeat the equantion until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \text{where} j = 0, 1$$

And α is called **learning rate** which controlls the step of learning. The partical or derivative part controlls the direction.

If α is too small, gradient descent can be really slow. But if it's too large, gradient descent may not converge or even diverge.

With this algorithm and appropriate learning rate, the parameters are always towards an (local) optimal valley.

2.3.2 Gradient Descent for Linaer Regression

With gradient descent, we can handle with the cost function provided in previous sections.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=0}^{m} \frac{\partial}{\partial \theta_i} h_\theta(x^{(i)}) \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

Informally, **convex function** has no local optimum but only one global optimum.

Batch gradient descent: each step takes all the training examples.

2.4 Linear Algebra Review

2.4.1 Matrices and Vectors

Definition 2.2 (Matrix) *Rectangular array of numbers, dimension of matrix is # of rows times # of columns. For a matrix A, its (i, j) entry is writen as A_{ij}.*

Definition 2.3 (Vector) *Vector can be treated as a $n \times 1$ matrix in $\mathbb{R}^n$.*

2.4.2 Addition and Scalar Multiplication

Matricx addition is to add elements at the same place of their matrix, of course, the matrices should have the same size. (element wise)

Scalar multiplication is to multiply the scalar number to every element in the matrix.

2.4.3 Matrix Vector Multiplication

For a matrix and a vector to be multiplied, they should respectively has size of $m \times n$, $n \times 1$. This relationship of size accords to that the answer is the linear combination of columns of matrix with coefficient of respected vector elements.

2.4.4 Matrix Matrix Multiplication

For 2 matrices to be multiplied, they should respectively has size of $m \times n$, $n \times p$.

2.4.5 Matrix Multiplication Properties

For matrix multiplication, it's a way to pack a series of hypothesis which has the same structure and different parameters.

2.4.6 Inverse and Transpose

Not all numbers have an inverse. So do matrices.

Only square matrices **may** have inverse. Else matrices could have pseudo inverse.

Words

quadratic

ellipse

contour

by convention

Chapter 3 Multivariate Linear Regression and Octave Tutorial

3.1 Multivariate Linear Regression

With **Multivariate Linear Regression**, we can do predictions with more infomation. It appears that the expression will get more inputs or features.

Note $x_j^{(i)}$ *refers to the* i^{th} *training example's* j_{th} *feature value.*

For example, for a n-varible hypothesis, its math form should be like:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

Its vector form:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}, \text{where } x_0 = 1$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T$$

3.2 Gradient Descent for Multiple Varibles

For n-varible hypothesis, the cost function expresses like:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^(i)) - y^{(i)})^2$$

So the gradient descent performs like:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}, \text{where } \theta = 0, 1, 2, \cdots, n$$

3.3 Gradient Descent in Practice

After the basic knowledge introduced, we are going to learn something in practice.

3.3.1 Feature Scaling

Make sure features are on a similar scale and the cost function can converge more quickly. If $x_1 \in (0, 2000)$ and $x_2 \in (1, 5)$, the step of gradient descent may fit with x_2 but be too small to quickly converge.

A typical and useful opration is to scale feature into $(0, 1)$. More generally, get every feature into approximately a $[-1, 1]$ range.

Or take mean normalization. Replace x_i with $x_i - \mu_i$ or $(x_i - \mu_i)/(\max(x) - \min(x))$ to make features have approximately zero mean.

3.3.2 Learning Rate

Make sure that gradient descent is working correctly, that is $J(\theta)$ should decrease after each iteration.

We can do automatic convergence test that if the decrease of $J(\theta)$ is less that a threshold, we declare the convergence.

If the α is too large, the loss may not converge or even diverge. If too small, it will take too long time to end the task.

3.4 Features and Polynomial Regression

Sometimes a straight line model would not fit a curve well, so that we choose polynomial regression to fit it with a polynomial.

For example, a 3-order polynomial like $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$, we can do mapping like

$$x \to x_1, x^2 \to x_2, x^3 \to x_3$$

Of course, in this example it's important to do scaling.

3.5 Computing Parameters Analytically

3.5.1 Normal Equation

Normal equantion is a method to solve for θ analytically.

Using calculus we can solve for the global optimal for equation (single or multiple varibles).

Or, we can display our data in matrix form, then use the least square method.

For features, use X; for outputs, use Y:

$$X = \begin{bmatrix} (x^{(0)})^T \\ (x^{(1)})^T \\ \vdots \\ (x^{(n)})^T \end{bmatrix} \in \mathbb{R}^{n+1}, Y = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

With least square method, get Θ like:

$$\Theta = (X^T X)^{-1} X^T Y$$

In Octave, it could compute by `pinv(x'*t)*x'*y`.

With normal equation, you do not need to choose a α and to iterate. BUT, when the dimesion is large, the computation $(X^T X)^{-1}$ could be very slow (it's a $O(n^3)$ algorithm). With gradient descent, it could work well even dimesion is large.

3.5.2 Normal Equation Noninvertibility

What will happen if $X^T X$ is not invertible? The pseudo inverse is used when

- redundant features / linearly dependent
- too many features

3.6 Octave Tutorial

Octave is a good language for algorithm prototyping.

3.6.1 Basic Operations

In fact, it's just so similar to MATLAB. For example, the `ones, zeros, eye, rand`, and the vector form.

```
w = rand(1, 10000) + (-6) % uniform distribution
hist(w, 100)
```

```
w = randn(1, 10000) + (-6) % normal distribution
hist(w, 100)
```

3.6.2 Moving Data Around

It's about how to move data into Octave for certain tasks.

For data file, use the `load` and `save`. For matrix, use the column and row index with ":", like `vec(col1:col2)` and matrix concat could be rather easy, like `c = [a b]`.

3.6.3 Computing on Data

For coresponding or elemenet-wise operations, there is a dot sign befor the opration, like `c = a .* b`, where a and b get the same size. Without the dot sign, it will be a matrix multiplication. And transpose is important to matrix which is bind to quatation mark, like `AT = A'`.

For most functions, they could be applied to vector or matrix form, like `abs([1,-2,3])`, `a = [1 3 5 -9]; a_b = a > 2`.

Some functions:

1. `magic`: return a n by n magic square matrix
2. `sum`: return sum of matrix
3. `prod`: return prod of matrix
4. `A(:)`: turn A into a vector

3.6.4 Plotting Data

In implying algorithm, plot is a very important tools. Just like in MATLAB, use `vectors, plot, figure, subplot, imagesc`. Some controll command like `colorbar, colormap, hold on, xlable, ylable, legend, title,`

3.6.5 Control Statements and Functions

In most programming languages, there are `for, while if` statements for controll.

Besides, we can define our own functions in seperate files and use them by `addpath` or `cd`

3.6.6 Vectorization

With vectorization, code can be more efficient and quick to imply.

For example, we can use vector transpose to compute the inner product rather than for-loop.

Words

prototyping

clunky

Chapter 4 Classification

4.1 Classification and Representation

4.1.1 Classification

Classification is to predict the varible y into discrete values, in other workds, an assignment to classify features into classes. For example, divide eamils by spam or not.

We can set a list of threshold of some parameters of hypothesis to devide the dataset into different classes.

But if we apply linear regression, some extreme data will have a nonnegligible effect on the hypothesis.

As we all know, the hypothesis usually gives a continuous value, while the classification problem gives discrete values. But apply hypothesis and map values into discrete values sometimes cannot work well.

Here we will focus on the binary classification which has a postive class and a negetive class.

4.1.2 Hypothesis Representation

A logistic regression model want it's $h_\theta(x) \in [0, 1]$. We could apply **Sigmoid Function**:

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

It's like

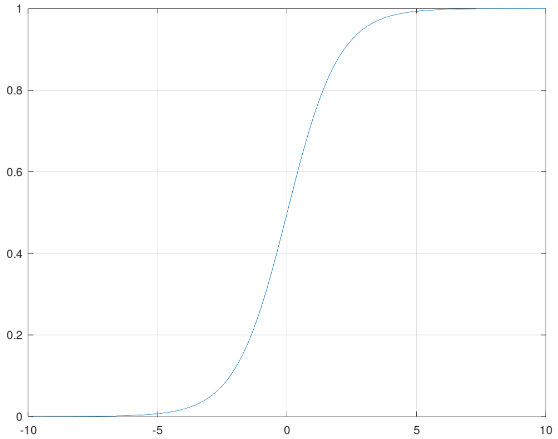

Figure 4.1: Sigmoid Function

Then, we get:

$$h_\theta(x) = g(\theta^T x)$$

Let's interprete the hypothesis' output: $h_\theta(x)$ is the estimated probability that $y = 1$ on input x. That is **probability that $y = 1$, given a x parameterized by θ**:

$$h_\theta(x) = P(y = 1|x; \theta)$$

4.1.3 Decision Boundary

Why Sigmoid Function makes sense? When $h_\theta(x) > 0.5$ we predict $y = 1$ and predict $y = 0$ in else condition. And it means that $\theta^T x > 0$ or not.

$$h_\theta(x) = g(\theta^T x)$$

By Sigmoid Function we can classify the hypothesis by it's values.

When we get 2 varibles, like $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ and we still make binary classification, wo we can still predict $y = 1$ if $h_\theta(x) \geq 0$ and $y = 0$ if $h_\theta(x) \leq 0$.

Further more, how about we need a non-linear decision boundaries, like a circle or a a ellipse? Just use non-linear functions! For a ellipse:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

And we define the boundary like:

$$\begin{cases} y = 1, \text{ if } x_1^2 + x_2^2 \geq 1 \\ y = 0, \text{ else} \end{cases}$$

4.2 Logistic Regression Model

4.2.1 Cost Function

For a classification problem, we need the cost function, too. This is related to how to choose the parameters.

In chapters before, we defined as a sum of squared error:

$$\text{Cost}\left(h_\theta(x^{(i)}, y^{(i)})\right) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}, y^{(i)}))$$

For a simple cost function, it's non-convex so it does not guarantee to converge.

In logistic regression cost function, we define:

$$\text{Cost}\,(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) \text{ if } y = 1 \\ -\log(1 - h_\theta(x)) \text{ if } y = 0 \end{cases}$$

If the hypothesis gives an approximate output as y, the cost is rather small; but when it goes to the other end, the cost will grow in a very fast speed.

And this cost function could be convex in our problem.

4.2.2 Simplified Cost Function and Gradient Descent

Let's re-write the cost function in a more compact way as:

$$\text{Cost}\,(h_\theta(x), y) = -y\log(h_\theta(x), y) - (1-y)\log(1 - h_\theta(x))$$

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}, y^{(i)}) \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1 - h_\theta(x^{(i)}))\right] \end{aligned}$$

To fit parameters θ, we need to minimize the $J(\theta)$. Then we can apply the gradient descent as in previous chapters.

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}, \text{where } \theta = 0, 1, 2, \cdots, n$$

Note: the $h_\theta(x)$ is different! Now we have $h_\theta(x) = 1/(1 + e^{-\theta x})$.

4.2.3 Advanced Optimization

With this section, we can get logistic regression run more quickly. Like, gradient descent, conjugate gradient, BFGS[1], L-BFGS[2]

We can just simply call `fminunc` to get gradient descent.

4.3 Multi-class Classification

For multiclass problem, we can encode each class with some num like 1, 2, 3, 4 and so on. One-verse-all algorithm is to seperate different class to a special postive class in turn. Or: train a logistic regersss classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$. Finally, pick the $\max_i h_\theta^{(i)}(x)$.

[1] Broyden-Fletcher-Goldfarb-Shanno algorithm,

[2] Limited-memory BFGS

4.4 Solve Overfitting: Regularization

4.4.1 The Problem of Overfitting

What's overfitting? If a little bit more complicate dataset cannot be fitted with a line, the linear fit will have a high preconception or bias(underfitting). If we apply a polynomial regression with too high order, hte learned hypothesis will fit the dataset very well but fall to have the ablity to generalize the problem(overfitting).

If we want to address the overfitting, we can

- reduce number of features
  - manually select features to keep
  - model selecttion algorithm
- aplly regularization
  - keep all features, but reduce magnitude or value of some parameters θ_j
  - works well when get lots of features

4.4.2 Cost Function for Regularization

If we want to make some parameters really small, we can add some terms like:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 1000\theta_4^2$$

Small parameters will turn to simpler hypothesis and get more smoth and less prone to overfit. If we want to shrink all parameters, the cost could be implemented as[3]:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}\theta_i^2\right]$$

The latter term is called regularization term. The regularization will put off a more general hypothesis.

4.4.3 Regularized Linear Regression

If we do gradient descent, the gradient will be different for θ_0 because the extra term does nothing with it.

$$\theta_j := \begin{cases} \theta_j - \alpha\frac{1}{m}\left[\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \text{, where } \theta = 1, 2, \cdots, n \\ \theta_j - \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \text{, where } \theta = 0 \end{cases}$$

[3] remeber that: θ starts with 0, but we just ignore the first term

Similarly, we can still apply the normal equation in conditions:

$$\theta = \left(X^T X + \lambda M\right)^{-1} X^T y, \text{ where } M = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

4.4.4 Regularized Logistic Regression

$$J(\theta) = -\frac{1}{m}\left[y^{(i)}\log(h_\theta(x^{(i)}), y^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\theta_i^2$$

$$\theta_j := \begin{cases} \theta_j - \alpha\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}, \text{ where } \theta = 1, 2, \cdots, n \\ \theta_j - \alpha\left[\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right], \text{ where } \theta = 0 \end{cases}$$

Words

ameliorate

preconception

contort

penalize

prone

Chapter 5 Neural Network: Representation

5.1 Neural Network's Motivation

Neural Network has appeared for a long time and become state-of-the-art technique recently.

5.1.1 Non-linear Hypotheses

In non-linear classification, there could be a LOT of parameters or features.

For example, a image classifier get input in a matrix form. We need to allocate some certain pixels to a space, then classify the sample space. If a image has a size of 50×50 then the quadratic features could be $(2500)^2/2 \approx 3 \times 10^6$. So, this kind of classifier could have realy lots of features.

5.1.2 Neurons and the Brain

We will get to know some background of NNs, and have a sense of what they do.

The NN algorithms are aimed to mimic the brain's work. Though it could do many many things like math, writings, only "the one learning algorithm" is needed.

5.1.3 Model Representation

We will learn how to represent hypothesis in NN model.

Features are input, the hypothesis is output, and we need something between them. Input sometimes gets an extra **bias**. The previous θ is called parameters.

The NN is just putting every features strong together. Normally, the first layer is called input layer, the last is called output layer and the else are called the hidden layers.

Let's define some symbols:

- $a_i^{(j)}$: activation of unit i in layer j
- $\Theta^{(j)}$: matrix of weights controlling functions mapping from layer j to layer $j+1$

For example, if layer 1 gets 2 input and the layer 2 gets 4 input, the $\Theta^{(1)}$ will have a size of 4×3. Then let's vectorize the representation. That is[1]:

$$X^{(j+1)} = g\left(\Theta^{(i)} \times \begin{bmatrix} bias^j \\ X^j \end{bmatrix}\right)$$

The architectures are how the neurons are connected.

[1] by default, bias is set to 1

5.2 Applications of NNs

We can adjust the weights of an one-layer network wo get AND or OR functions. But XOR needs two layers.

Deeper networks could compute more complex features.

5.3 Multi-Class Classification

Still, we will apply "one-vs-all" algorithm in NN method. For a n-class problem, we need a n-dimision output at the last layer of NN, which could generate a onehot encode after sigmoid.

Word

resurgence

cortex

dendrite

Chapter 6 Neural Network: Learning

6.1 Cost Function and Backpropagation

6.1.1 Cost Function

Let's define symbols for n-class classification:

- the input feature and its class: $(x^{(1)}), y^{(1)}, (x^{(2)}), y^{(2)}, \cdots, (x^{(n)}), y^{(n)}$
- L is total number of layers
- s_l is the number of units in layer l

For logistic regression, the cost function is

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}\theta_i^2\right]$$

For a neural network, it's:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right]$$
$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

where: $\quad h_\Theta(x) \in \mathbb{R}^K, \quad (h_\Theta(x))_i = i^{th}$ output

The first term is for all K dimension output, and the last is the regular term of all weight in the neural network.

6.1.2 Backpropagation Algorithm

Backpropagation algorithm is a way to minimize the cost.

To use gradient descent, we need $J(\theta)$ and $\dfrac{\partial J(\theta)}{\partial \Theta_{i,j}^{(l)}}$.

So we have to compute the partical terms. We define the error of the L layer's node j:

$$\delta_j^{(l)} = a_j^{(l)} - y_j$$

Then, for earlier layers:

$$\delta^{(l-1)} = (\Theta^{(l-1)})^T\delta^{(l)}. * g'(z^{(l-1)})$$

Where:

$$g'(z^{(l)}) = a^{(l)}. * (1 - a^{(l)})$$

Finally:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}, \text{when } \lambda = 0$$

For a training set $(x^{(1)}), y^{(1)}, (x^{(2)}), y^{(2)}, \cdots, (x^{(m)}), y^{(m)}$:

```
Delta(l)(i,j) = 0
for i = 1 : m
   set a(1) = x(i)
   compute for a(l) for l = 2,3,..,L
   with y(i), compute delta(L)
   then compute delta(L-1), ...,delta(1)
   Delta(l)(i,j) = Delta(l)(i,j) + a(l)(j) * delta(l+1)(i)
endfor
```

Or in vector form:

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

Then:

$$\begin{aligned} D_{i,j}^{(l)} &= \frac{1}{m}\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(i)}, \text{ if } j \neq 0 \\ D_{i,j}^{(l)} &= \frac{1}{m}\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(i)}, \text{ if } j = 0 \end{aligned}$$

And:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)1 = D_{i,j}^{(l)}$$

6.1.3 Backpropagation Intuition

Backpropagation is more likely to be a blankbox than previous algorithm.

What do forward propagation do in the NNs? Doing non-linear matrix multiplication by introducing bias. And similarly, the former error is due to latter layers.

6.2 Backpropagation in Practice

6.2.1 Advanced Optimization

Learn to unroll parameters matrices into vectors.

```
thetaVec = [Theta1(:); Theta2(:)];
```

The unrolled weights could be passed into `fminunc(@cost, initTheta, option)`, where the `cost` takes the unrolled vectors.

6.2.2 Gradient Checking

There could be some subtile bugs, so we need to check gradient.

We need to numerially compute the gradient at a point, and it just needs

$$\frac{\mathrm{d}}{\mathrm{d}\theta}J(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$$

As for every parameter in the big vector, we can apply this by treat others as constant. Then we need to check that $gradApprox \approx DVec$. Besides remember to disable check when you are training.

6.2.3 Random Initialization

For gradient descent and other advanced methods we need to initialize all weight. If we set all weights to zero, the gradient will be all the same for each value will be zero and all weights in the same layer are in the identical.

To get through this, random initialization is applied. The key idea is to break symmetry.

6.2.4 Putting It Together

Let's do some overall summary of neural network.

First we need a neural network architecture (a connectivity pattern between neurons). The number of input units is the dimension of features. The number of output units is the number of classes.

Then we need randomly initialize weights and implement the forward propagation. compute the cost function and do back propagation for partial derivatives. Of course we need to compute the gradient checking and after that remember to disable it. Finally, use something to compute.

6.3 Application of Neural Network: Autonomous Driving

Word

subtile

Chapter 7 Advice for Applying Machine Learning

This chapter is about how to implement powerful algorithm.

Sometimes, a better algorithm is better than more data. But usually, we should try :

- more training examples
- smaller sets of features
- additional features
- polynomial features
- changing λ

We need **Machine Learning Diagnostic** to test whether we can gain insight that if it works on it. A diagnostic will take time to implementm, but it will be worth the time.

7.1 Evaluating a Hypothesis

Less error is not always meaning better, for it's possible to generalize to new examples.

We can divide dataset into training set and test set. Normally training set is of 70%.

Learn from training set and compute via test set. You can apply the cost or misclassification error.

$$\text{err}(h_\theta(x), y) = \begin{cases} 1, \text{ if } h_\theta(x) \geq 0.5, y = 0 \\ 1, \text{ if } h_\theta(x) \leq 0.5, y = 1 \\ 0, \text{ else} \end{cases}$$

And the total cost is

$$\text{Test Err} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \text{err}(h_\theta(x_{test}^{(i)}), y^{(i)})$$

Via this, we can simply check whether the algorithm is proper or not.

7.1.1 Model Selection and Train/Validation/Test Sets

How to decide the degree of polynomial or regularized parameters?

Once the parameters are overfitting a dataset, the error will be lower than it should be.

We can do model selection over the degree of the polynomial functions. But the cost in training is not a fair estimate of the model generalization. We should divide the dataset into 3 parts rather than 2: training set(60%), cross validation set(20%), test set(20%). And get error for each part, then treat the loss of cross validation set as a quality of generalization.

7.2 Bias vs. Variance

7.2.1 Diagnosing Bias vs. Variance

High bias turns to underfitting and high variance turns to overfitting. It's important to determine which kind of problem we are facing.

We can see them from the training error and cross validation error. As degree of polynomial increasing, the error will decrease. And the cross validation error will be higher than it, like **Figure** 7.1.

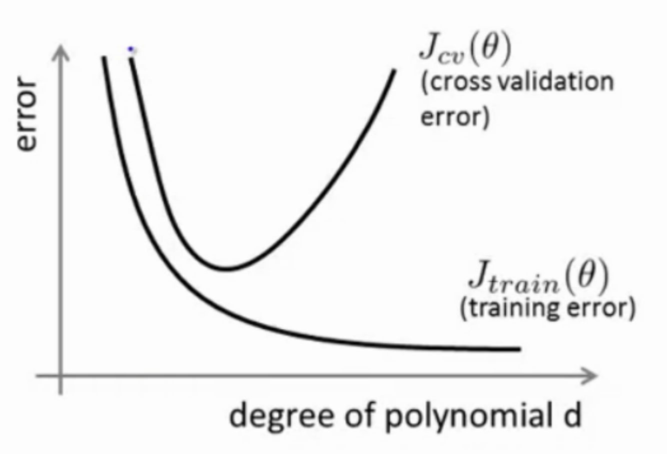

Figure 7.1: The error with degree of polynomial

If the bias is too big, the $J_{train}(\theta) \approx J_{cv}(\theta)$ will be high; If the variance is too big, $J_{train}(\theta) \ll J_{cv}(\theta)$

7.2.2 Regularization and Bias/Variance

If we have applied regularization to our algorithm, we will get an extra λ. If the λ is too large, the parameters will be rather small and then the bias gets too big then underfitting. If too small, similarly, overfitting.

So, we need to try different λ, and observe the relationship between loss and λ.

7.2.3 Learning Curves

Learning curves are easy to plot in learning and could show the learning condition. It's an (error-m(training set size)) curve. As the set size increasing from 0, the error would increase and the speed to increase in decreasing. At the same time, the cv-error would decrease.

If trapped in high bias, finally we will get $J_{cv} = J_{train}$. So, if algorithm is suffering from high bias, more data will not help so much/

If trapped in high variance, more data will help.

7.2.4 Decide What to Do next

All we mentioned told us what's useful for our ML work. Let's have a summary:

- Get more training examples to fix high variance
- Try less sets of features to fix high vairan
- Try more features to fix high bias
- Try more polynomial features to fix high bias
- Try decreasing λ to fix high bias
- Try increasing λ to fix high variance

Small NNs has fewer parameters and more prone to underfitting. Large ones have more parameters and likely t overfitting and **MORE EXPENSIVE**. Try regularization to address overfitting.

Low order polynomials have high bias and load variance, while high order polynomials have high variance and low bias.

7.3 Build a Classifier

It's a problem about 1 and 0. We need x as the features of mails and y is the class of the mails.

First choose some features that indicative of spam or not.

How to make it have low error ?

- collect lots of data
- develop sophisticated features based on email routing info from email header.
- develop sophisticated features for the message
- develop algorithm for misspellings

7.3.1 Error Analysis

Recommended approach:

- Start with a simple algorithm that can implement very quickly.
- Plot learning curves to decide if more data, more features that could help.
- Error analysis, manually examine the examples.

Usually over the evaluation set.

7.3.2 Error Metrics for Skewed Classes

If the data is skewed to a side, the model's precision is not so impressive.

Use precision and recall. Like true positive, false positive, false positive and true negative. You get the precision and recall to see the prediction and the actual fact.

Then we need to trade off between precision and recall. If we want to predict 1, we have to be very confident. On the other hand we need to avoid false negatives.

We can use the **F score**. Simply, take the average. But the importance is different for the two part. The F score takes the $2PR/(P+R)$.

7.4 Data for Machine Learning

First need to assume the amount of features has sufficient information to predict the answer accurately.

A very large dataset is more unlikely to overfit.

Word

Avenues

winnow

Chapter 8 Optimization Objective

8.1 Optimization Objective

We will introduce supported vector machine, which is supervised.

For a sigmoid function, if the y = 1, we want the $h_\theta(x) \approx 1$ and $\theta^T x \gg 0$.

Support vector machine simplifies the cost function of logistic regression into a straight line.

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{i=0}^{n}\theta_i^2$$

As the SVM is two order, the co-efficient before the expression could be dismissed.

The hypothesis will be this:

$$h_\theta(x) = 1 \text{ if } \theta^T x \leq 0; 0, \text{ else}$$

8.2 Large Margin Intuition

SVM is called large margin intuition sometimes. The threshold 1 and -1 make the SVM "safer".

SVM would give a decision boundary that seperate the region with a margin. A large co-efficient will turn the SVM to a sensitive one.

The SVM decision boundary is decided by inner product.

8.3 Kernels

If we need a non-linear decision boundary, we can apply some landmarks and compute new features depending on proximity to landmarks.

The similarity could be expressed as:

$$\text{similarity}(x, l^{(i)}) = \exp(-\frac{||x - l^{(i)}||^2}{2\sigma^2 s})$$

It's called Guass Kernel, for it's based on Guass Distribution. The problem is how to get the landmarks.

But how to choose the landmarks? We can set the training examples as landmarks. With all the landmarks we can get a feature vector contains the similarity of each landmarks.

So for a SVM with kernels we can predict $y = 1$ if $\theta^T f \geq 0$. The training is to :(ingore the

θ_0 for the squared items)

$$\min\left[C\sum_{i=1}^{m}y^{(i)}cost_1(\theta^Tf^{(i)})+(1-y^{(i)})cost_0(\theta^Tf^{(i)})+\frac{1}{2}\sum_{j=1}^{m}\theta_j^2\right]$$

We need to apply the scaling to avoid a very BIG distance in the Guassian Kernel.

8.4 SVM and Logistic regression

If number of features is large, use logistic regression or SVM with linear kernel.

If n is small, m is intermediate, use SVM with Guassian Kernel.

If n is small, m is large, add some features then use logistic regression or SVM with linear kernel.

Chapter 9 chap

9.1 sec