Technische Universität Berlin | Fakultät IV | Neuronale Informationsverarbeitung

Notes on Machine Learning

Dr. Rong Guo

Berlin, 2021

Contents

1 Introduction

Machine learning: to learn from experience E with respect to some task T and some performances P, if its performance on T, as measured by P improves with experience E.

Supervised learning: "right answers", there's a relationship between the input and the output

Regression	Classification
predict continuous valued output	predict discrete valued output

To make predictions with infinite number of features

Unsupervised learning: "no labels", to find some structure in the data

Supervised	Unsupervised
Classification	Clustering
Regression	Dimensionality reduction
Semantic segmentation	Feature learning
Image captioning, etc	Density estimation, etc

2 Linear regression

Linear regression with one variable, univariate linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{1}$$

2.1 Cost function

- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
- Parameters: θ_0, θ_1
- Cost function: $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^(i))^2$
  "Squared error function / mean squared error"
- Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

The hypothesis is a function of the training data, whereas the cost function is a function of the parameters. For each set of the parameters, the value of the cost function can be calculated through the hypothesis function (the prediction vs. the true data).

2.2 Gradient descent

Gradient descent is an iterative algorithm of performing the minimisation.
repeat until convergence {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1) \quad (\text{for}\quad j = 0 \text{ and } j = 1) \tag{2}$$

}
The gradient descent will automatically take smaller steps as we approach a local minimum:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = 0 \tag{3}$$

Batch gradient descent: each step of gradient descent uses all the training examples.
Simultaneously update all the parameters at each iteration.

2.2.1 Learning rate

If α is too small, gradient descent can be slow;
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.
α should be adjusted to ensure that the gradient descent algorithm converges in a reasonable time: failure to converge or too much time to obtain the minimum imply that the step size is wrong.

2.2.2 Gradient descent for linear regression

Substitue the actual cost function and the actual hypothesis function into the gradient descent: repeat until convergence {

$$\theta_0 : = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$
$$\theta_1 : = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

3 Linear Algebra review

3.1 Matrix and vectors

Matrices are 2-dimensional arrays.
Dimension of matrix: number of rows $\times$ number of columns, e.g., $\mathbb{R}^{4\times 2}$.
Matrix elements (entries of matrix), e.g., $A_{i,j}$ refers to the elment in the i_{th} row and j_{th} column of the matrix A.

A vector is a matrix with only one column: A vector with n rows is referred to as an n - dimensional vector.

Vector: An $n \times 1$ matrix, e.g., $\mathbb{R}^4$, $y_i = i^{th}$ element.

Scalar means that an object is a single value, not a vector or matrix, e.g., $\mathbb{R}$ refers to the set of scalar real numbers, and $\mathbb{R}^n$ refers to the set of n - dimentional vectors of real numbers.

Addition and subtraction are element-wise. To add or subtract two matrices, their dimensions must be the same. In scalar multiplication/division, we simply multiply/divide every element by the scalar value.

Matrix vector multiplication: Prediction = Datamatrix $\times$ Parameters.

One line of code rather than a for loop; more efficient computation.
A $m \times n$ matrix multiplied by an $n \times 1$ vector results in an $m \times 1$ vector.
Matrix matrix multiplication: Two matrices are multiplied by breaking it into several vector multiplications and concatenating the result.

3.1.1 Matrix properties

Matrices are not commutative: $A \times B \neq B \times A$.
Matrices are associative: $(A \times B) \times C = A \times (B \times C)$.
Identity matrix: $I_{n\times n}$, for any matrix A, $A \cdot I = I \cdot A$. e.g., $I_{3\times 3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Matrix inverse and transpose: If A is an $m \times m$ (square matrix) matrix and if it has an inverse, $AA^{-1} = A^{-1}A = I$. Let A be an $m \times n$ matrix, and let $B = A^T$, then B is an $n \times m$ matrix and $B_{ij} = A_{ji}$.

4 Multivariate linear regression

Multiple features: $x_j^{(i)}$ = value of feature j in i^{th} training example.
m = the number of training examples;
n = the number of features.

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{4}$$

For convenience of notation, define $x_0^i = 1, \text{for } (i \in 1, \cdots, m)$.

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n+1}, \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \vdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}.$$

- Hypothesis:

$$\begin{aligned} h_\theta &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \theta^T x \end{aligned} \tag{5}$$

- Parameters: $\theta \in \mathbb{R}^{n+1}$
- Cost function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$
- Gradient descent:
  repeat until convergence {

$$\begin{aligned} \theta_j :&= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \end{aligned} \tag{6}$$

  (simultaneously update for every $j = 0, \cdots, n$)
  }

4.1 Gradient descent in practice

Feature scaling: to make sure features take on roughly similar ranges of values, i.g., divided by the maximum. To get every feature into approximately a $-1 \leq x_i \leq 1$ or $-0.5 \leq x_i \leq 0.5$ range, so that the gradient descent converges much faster.
Mean normalisation: to make features have approximately zero mean, i.g., $\frac{x_i - \mu_i}{s_i}$, where $\mu_i =$ mean; $s_i =$ (max - min) or the standard deviation.
debugging:

- plot $J(\theta)$ vs."number of iterations". $J(\theta)$ should decrease after every iteration
- declare convergence if $J(\theta)$ decreases by less than ϵ in one iteration, e.g., $\epsilon = 10^{-3}$
- Learning rate α: try $\cdots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \cdots$
  If $J(\theta)$ is increasing or oscillating, use smaller α. If α is too small, $J(\theta)$ will converge very slow.

4.2 Vectorised implementation

$$X = \begin{pmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & & & \vdots \\ 1 & x_1^m & x_2^m & \cdots & x_n^m \end{pmatrix} \in \mathbb{R}^{m\times(n+1)}, y = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{pmatrix} \in \mathbb{R}^m, \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \vdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta = X \cdot \theta \tag{7}$$

$$J(\theta) = \frac{1}{2m}(X \cdot \theta - y)^T (X \cdot \theta - y) \tag{8}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T (h_\theta(x) - y) \tag{9}$$

$$\theta := \theta - \frac{\alpha}{m}(X^T(X \cdot \theta - y)) \tag{10}$$

Note that the summation of the product of two terms can be expressed as the product of two vectors[1].

4.3 Features and polynomial regression

To change the hypothesis function by creating more features from each data point, e.g., quadratic, cubic or square root function:
$h_\theta(x) = \theta_0 + \theta_1 x \rightarrow h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x_1^2 + \theta_3 x^3$
Note that in this way the feature scaling becomes very important

1 $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \vec{x}_j (X\theta - y), \nabla J(\theta) = \frac{1}{m} X^T (X\theta - y)$

4.4 Normal equation

To solve for θ analytically, by explicitly taking $J(\theta)$ derivatives with respect to the θ_j and setting them to 0:

$$\theta = (X^T X)^{(-1)} X^T y \tag{11}$$

There is no need to do feature scaling with the normal equation solution.

Noninvertibility (singular/degenerate, $X^T X$ may be noninvertible):

- Redundant features (linearly dependent)
- Too many features (e.g., $m \leq n \rightarrow$ delete some features, or use regularization)

The Gradient Descent model was trained with normalized data. So you must normalize the test point in the same way before you do the prediction. The theta values in the two solutions (Gradient Descent vs. Normal Equation) will be different, because one model is trained with normalized data and the other is not. The predicted values in the two solutions should be same.

5 Classification

5.1 Logistic regression

Similar to the regression problem, but the values to be predicted take on only a small number of discrete values. Linear regression is not a good idea in the classification problem, because the classification is not actually a linear function.

Binary classification problem:
$y \in \{0, 1\}$
0: "negative Class"
1: "Positive Clas"

5.1.1 Hypothesis representation

Logistic regression model to satisfy $0 \leq h_\theta(x) \leq 1$

$$h_\theta(x) = g(\theta^T x), \quad z = \theta^T x, \quad g(z) = \frac{1}{1 + e^{-z}} \tag{12}$$

$g(z)$: Sigmoid/Logistic function. It crosses 0.5 and the origin, asymptotes at 1 and asymptotes at 0.

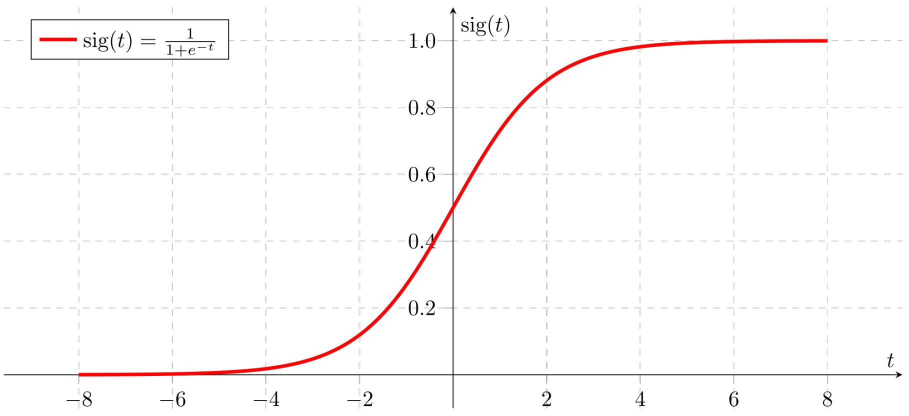

Figure 1: Sigmoid function

Interpretation of hypothesis output: $h_\theta(x) = P(y = 1|x; \theta)$, "probability that $y = 1$, given x, parameterized by θ", the estimated probability that $y = 1$ on input x.

$y \in \{0, 1\}, P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$

5.1.2 Decision boundary

In order to get the discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:
suppose
predict "$y = 1$" if $h_\theta(x) \geq 0.5 \Rightarrow \theta^T x \geq 0$
predict "$y = 0$" if $h_\theta(x) < 0.5 \Rightarrow \theta^T x < 0$

The decision boundary is the line that separates the area where $y = 0$ and where $y = 1$. In this case, it's the line or curve of $\theta^T X = 0$ for the binary classification. Given θ fitted via a certain procedure, the decision boundary can be plotted in the training data space. The decision boundary is a property, not of the training set, but of the hypothesis under the parameter. The parameters θ defines the decision boundary. The training set is what we use to fit the θ.

Non-linear decision boundaries: $\theta^T x$ does not have to be linear, similar to the polynomial regression, we could add extra higher order polynomial terms to the features to get complex decision boundaries (e.g., $x_1^2 + x_2^2 = 1$ or a more complex shape).

In a 2-D display, the decision boundary can be plotted as a line (linear, only two features, decided arbitrarily by two x_1 points and two x_2 points calculated from $X\theta = 0$) or a curve (polynomial, more features created from the $x_{1,2}$ points, contour plot of $X\theta = 0$ on the mesh grid of X).

5.1.3 Cost function

- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$, m examples

  $$x \in \begin{pmatrix} x_0 \\ x_1 \\ x_2 \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n+1}, \quad x_0 = 1, \quad y \in \{0, 1\}$$

- Cost function [2]:

  $$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if} \quad y = 1 \\ -\log(1 - h_\theta(x)) & \text{if} \quad y = 0 \end{cases} \tag{13}$$

  If $h_\theta(x) = y$, then cost $(h_\theta(x), y) = 0$ for both $y = 0$ and $y = 1$, see also **Figure** 5
  If $y = 0$, then cost $(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 1$
  If $y = 1$, then cost $(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 0$
  This captures the intuition that if the learning algorithm predicts $P(y = 1|x; \theta) = 0$, but $y = 1$, it will be penalized by a very large cost.

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

  Note that $y \in \{0, 1\}$, so we have $\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$, and

  $$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] \tag{14}$$

2 $h_\theta(x)^y (1 - h_\theta(x))^{(1-y)}$

A vectorised implementation is:

$$h = g(X\theta) \tag{15}$$

$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1-y)^T \log(1-h)\right) \tag{16}$$

A particular choice of cost function will give a convex optimisation problem (local optima free). If we take the logistic hypothesis function and plug it in the squared error cost function, it leads to a non-convex function. The gradient descent will not be guaranteed to converge to the global minimum. $J(\theta)$ is in the form of an ***maximum likelihood estimation****.*

- Goal:
  - to fit parameters θ: $\min_\theta J(\theta)$
  - to make a prediction given new x: Output $P(y=1|x;\theta) = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
- Gradient descent:
  repeat until convergence {

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{17}$$

  (simultaneously update all θ_j)
  }
  A vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - y) \tag{18}$$

  Both linear regression and logistic regression (classification) arrive at the same update rule (identical algorithm), thanks to the design of the cost function (a little calculus exercise, gradient of cross entropy function with sigmoidal probabilities, applying the chain rule and the derivative of logistic function[3]).

5.2 Optimization algorithm

Objective: $\min_\theta J(\theta)$.

For a given input value θ, write a function (code) that compute

$$J(\theta), \quad \text{and} \quad \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{for} j = 0, 1, \cdots, n)$$

and then use an optimization solver.

- Gradient descent
- Conjugate gradient
- BFGS
- L-BfGS

3 $\sigma(z) = \frac{1}{1+e^{-z}}, \frac{\mathrm{d}\sigma(z)}{\mathrm{d}z} = \sigma(z)(1-\sigma(z)), \nabla J(\theta) = \frac{1}{m} X^T (g(X\theta) - y)$

5.3 Multiclass classification: One-vs-all

One-vs-all (one-vs-rest): take the raining set and turn it into several separate binary classification problems. Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$, for $y \in \{0, 1, \cdots, n\}$. On a new input x, to make a prediction, pick the class i that maximizes $h_\theta^{(i)}$, i.g., prediction $= \max_i(h_\theta^{(i)}(x))$. Example: 3 classes. We take one class and then lump all the others into a single second class, repeatedly applying binary logistic regression to each class, e.g., class one gets assigned to the positive class, and classes two and three get assigned to the negative class (fake training set).

5.4 Regularization

Regularization allows us to ameliorate (reduce) the overfitting problem.

- "Underfitting": high bias (preconception, hypothesis too simple, too few features)
- "Overfitting": high variance (unnecessarily complicated hypothesis, fail to **generalize** to new examples)
- "Just right"

Addressing overfitting:

- reduce number of features.
  - manualluy select which features to keep
  - Model selection algorithm
- Regularization
  - Keep all the features, but reduce magnitude/values of parameter θ_j, when there are a lot of slightly useful features.

5.4.1 Cost function

Intuition:
Small values for parameters $\theta_0, \theta_1, \cdots, \theta_n$ leads to "simpler" hypothesis, smoother function, less prone to overfitting.
Example:
Features: $x_0, x_1, \cdots, x_n$
Parameters: $\theta_0, \theta_1, \cdots, \theta_n$

To modify the linear regression cost function to shrink all the parameters by **adding an extra regularization term**

$$\min_\theta \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right] \tag{19}$$

λ: regularization parameter. It controls a tradeoff between the goal of fitting the training set well and the goal of keeping the parameters small (keeping the hypothesis simple). In regularized linear regression, if λ is set too large, e.g., $\lambda = 10^{10}$, then $\theta_{1,2,\cdots,n} \approx 0, h_{\theta_0} = \theta_0$, which may smooth out the function too much and cause "underfit". If λ is set too small, the classifier gets almost every training example correct, but draws a very complicated boundary, thus "overfitting" the data.

5.4.2 Regularized linear regression

Gradient descent
repeat until convergence {

$$\begin{aligned}
\theta_0 : &= \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\
\theta_j : &= \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}\theta_j\right] \\
&= \theta_j(1 - \alpha\frac{\lambda}{m}) - \frac{\alpha}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) \qquad (j = 1, 2, 3, \cdots, n)
\end{aligned}$$

}
Regularization is not added to θ_0.
$\theta_{1,\cdots,n}$ is penalized (reducing its value by some amount on every update). usually $1 - \alpha\frac{\lambda}{m} < 1$, slightly smaller than 1.
Normal equation

$$X = \begin{pmatrix} (x^{(i)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix}, \quad y = \begin{pmatrix} y^{(i)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

to $\min\limits_{\theta} J(\theta)$,

$$\theta = (X^TX + \lambda\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix})^{-1}X^Ty \tag{20}$$

If $m < n$, then X^TX is non-invertible. However when $\lambda > 0$, $X^TX + \lambda \cdot L$ becomes invertible, where $L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$

5.4.3 Regularized logistic regression

Cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{21}$$

$(j = 1, 2, 3, \cdots, n)$

Note that θ_0 is explicitly excluded in the regularization term.

Gradient descent

repeat until convergence {

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \theta_j &:= \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}\theta_j\right] \\ &\quad (j \in 1, 2, 3, \cdots, n)\end{aligned}$$

}

A vectorised implementation is:

$$X = \begin{pmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & & & \vdots \\ 1 & x_1^m & x_2^m & \cdots & x_n^m \end{pmatrix} \in \mathbb{R}^{m\times(n+1)}, L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}, \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \vdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m}\cdot\left(-y^T\log(h) - (1-y)^T\log(1-h)\right) + \frac{\lambda}{2m}(L\cdot\theta)^T(L\cdot\theta) \tag{22}$$

$$\frac{\partial J(\theta)}{\partial\theta} = \frac{1}{m}X^T(h_\theta(x) - y) + \frac{\lambda}{m}(L\cdot\theta) \tag{23}$$

6 Neural Networks

6.1 Non-linear hypothesis and neurons in the brain

The problem of learning complex non-linear hypotheses: The number of features (e.g., all polynomial terms) very rapidly becoms unwieldy and impractical. Neural networks offers an alternate way to perform machine learning when we have complex hypotheses with many features.

Example: a grid of pixel intensity values. 50×50 pixel images $\rightarrow$ 2500 pixels in grey scale (750 if RGB), $n = 2500$. In this case, a nonlinear hypothesis by including all the the quadratic features $x_i \times x_j$ would end up a total of 3 million features [4].

Neural networks were developed as simulating neurons or networks of neurons in the brain. The "one learning algorithm" hypothesis: Neuro-rewiring experiments at the 20s.

4 The number of new features for all polynomial terms: $\frac{(n+r-1)!}{r!(n-1)!}$. $n =$ the number of original features; $r =$ the polynomial order. e.g., quadratic: $\mathcal{O}(n^2/2)$; cubic: $\mathcal{O}(n^3)$

6.2 Model representation

6.2.1 Neuron model: Logistic unit

Input: x (features $x_1, x_2, \cdots, x_n$).
Output: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ (result of the hypothesis function).
Terminology: $x_0 = 1$: bias unit; Sigmoid (logistic) activation function $g(z) = \frac{1}{1+e^{-z}}$; Parameters of the model θ: Weights; The first layer: Input layer; Intermediate layers: Hidden layer; The last layer: Output layer; Architecture: How the different neurons are connected to each other.

The intermediate"hidden" layer:
$a_j^{(l)}$ ="activation" of unit j in layer l
$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

The values for each of the "activation" nodes in a model with only one hidden layer:

$$\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}$$

Each layer gets its own matrix of weights, $\Theta^{(l)}$
If network has s_l units in layer l and s_{l+1} units in layer $l+1$, then $\Theta^{(l)}$ will be of dimension $s_{l+1} \times (s_l + 1)$. The $+1$ comes from the addition in $\Theta^{(l)}$ of the "bias nodes" $\Theta_0^{(l)}$ for x_0.

6.2.2 Forward propagation: Vectorized implementation

Example:

$$a^{(1)} = x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \in \mathbb{R}^4, \quad z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \in \mathbb{R}^3$$

$$\begin{aligned}
z^{(2)} &= \Theta^{(1)}a^{(1)} \\
a^{(2)} &= g(z^{(2)}), \quad \text{add} \quad a_0^{(2)} = 1 \\
z^{(3)} &= \Theta^{(2)}a^{(2)} \\
h_\Theta(x) = a^{(3)} &= g(z^{(3)})
\end{aligned}$$

The vector representation:

$$a^{(1)} = x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{(n+1)}, \quad z^{(l)} = \begin{pmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_n^{(l)} \end{pmatrix} \in \mathbb{R}^n$$

$$\begin{aligned}
z^{(l)} &= \Theta^{(l-1)}a^{(l-1)} \\
a^{(l)} &= g(z^{(l)}),
\end{aligned}$$

Rather than using the original features x, the model is using new features $a^{(2,\cdots,l+1)}$, which are learned as functions of the values of the activation nodes. Neural network learns its own features $a^{(2,\cdots,l+1)}$ of the hidden layers.

6.2.3 Examples: Simulating logical gates

Example: AND [5]

$$\begin{aligned}x_1, x_2 &\in \{0,1\}, x_0 = 1\\ y &= x_1 \text{ AND } x_2\\ h_\Theta(x) &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2)\\ &= g(-30 + 20x_1 + 20x_2)\end{aligned}$$

Example: OR $\quad h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$

Example: Negation $\quad h_\Theta(x) = g(10 - 20x_1)$

Example: (NOT $x1$) AND (NOT x_2)) $\quad h_\Theta(x) = g(30 - 20x_1 - 20x_2)$

Example: XNOR

$$\begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix} \to \begin{bmatrix}a_1^{(2)}\\a_2^{(2)}\end{bmatrix} \to \begin{bmatrix}a^{(3)}\end{bmatrix} \to h_\Theta(x)$$

$$\begin{aligned}a^{(2)} &= g(\begin{bmatrix}-30 & 20 & 20\\ 10 & -20 & -20\end{bmatrix} \cdot \begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix})\\ h_\Theta(x) &= g(\begin{bmatrix}-10 & 20 & 20\end{bmatrix}) \cdot a^{(2)}\end{aligned}$$

6.2.4 Multiclass classification

Multiple output units: One-vs-all.
Example: 4 categories.

$$h_\Theta(x) \approx y^i \in \left(\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right)$$

5 $\quad g(x) = \frac{1}{1+e^{-z}}, z = 4.6, g = 0.99; z = -4.6, g = 0.01$

6.3 Cost function

Neural network classification:

Binary classification	Multi-class classification (K classes)
$y \in 0, 1$	$y \in \mathbb{R}^K$ e.g., $\left(\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \right)$
1 output unit	K output units

- variables:
  - $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
  - L = total number of layers in network
  - s_l = number of units (not counting the bias unit) in layer l
  - K = number of output units/classes
- Cost function
  Logistic regression:

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

  Neural network:

$$\begin{aligned} h_\Theta(x) \in \mathbb{R}^K, &\quad (h_\Theta(x))_k = k^{th}\text{output} \\ J(\Theta) = -\frac{1}{m}&\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-h_\Theta(x^{(i)}))_k\right] + \\ &\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2 \end{aligned} \tag{24}$$

The regulation term adds up the squares of all the individual Θs in the entire network.

6.4 Backpropagation algorithm

"Backpropagation" is neural-network terminology for minimizing the cost function.
Gradient computation:
$\delta_j^{(l)}$ = "error" of node j in layer l.

Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j.
For training example $i = 1$ to m

1. Set $a^{(1)} := x^{(1)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \cdots, L$
   using $z^{(j)} = \Theta^{(j-1)}a^{(j-1)}, a^{(j)} = g(z^{(j)})$

3. Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$ (the "error values" for the last layer)
   $L =$ total number of layers
   $a^{(L)} =$ the vector of outputs of the activation

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$
   using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)})) \odot a^{(l)} \odot (1 - a^{(l)})$ [6]

5. $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
   (Vectorized implementation: $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$)

$$D_{ij}^{(l)} := \frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}\right) \qquad \text{if} j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{if} j = 0 \tag{25}$$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} \tag{26}$$

6.4.1 Implementation notes

- Unrolling parameters:
  Example:

$$\begin{aligned} &s_1 = 10, s_2 = 10, s_3 = 1 \\ &\Theta^{(1)} \in \mathbb{R}^{10\times 11}, \Theta^{(2)} \in \mathbb{R}^{10\times 11}, \Theta^{(3)} \in \mathbb{R}^{1\times 11} \\ &D^{(1)} \in \mathbb{R}^{10\times 11}, D^{(2)} \in \mathbb{R}^{10\times 11}, D^{(3)} \in \mathbb{R}^{1\times 11} \end{aligned}$$

  "unroll" all the elements in the matrices and put them into one long vector.

- Numerical gradient checking:

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

$$\frac{\mathrm{d}J(\theta)}{\mathrm{d}\theta_j} \approx \frac{J(\theta_1, \cdots, \theta_j+\varepsilon, \cdots, \theta_n) - J(\theta_1, \cdots, \theta_j-\varepsilon, \cdots, \theta_n)}{2\varepsilon}$$

  $\theta \in \mathbb{R}^n$, e.g., the "unrolled" version of Θ. e.g., $\varepsilon = 10^{-4}$

- Initial value of Θ:

  $\Theta_{ij}^{(l)} = 0$ for al i, j, l. After each update, parameters corresponding to inputs going into each of the hidden units are identical.

  Random initialization (Symmetry breaking):
  Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$, i.e., $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$
  e.g., $\epsilon = \frac{\sqrt{6}}{\sqrt{s_l + s_{l+1}}}$ for $\Theta^{(l)}, \Theta^{(l)} = 2\epsilon\ \mathrm{rand}(s_{(l+1)}, (s_l+1)) - \epsilon$

6 $\odot$: elment-wise multiplication; $g'(z^{(l)}) = a^{(l)} \odot (1 - a^{(l)})$

6.5 Training a neural network

- Pick a network architecture
  - Number of input units: Dimension of features $x^{(i)}$
  - Number of output units: Number of classes
  $$y \in \{1, 2, \cdots, n\} \rightarrow y \in \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right)$$
  - Number of hidden units per layer = usually more the better, but must balance with the cost of computation.

  Reasonable default: For 1 hidden layer, or if more than 1 hidden layer, have same number of hidden units in every layer.

- Training
  1. Randomly initialize the weights
  2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
  3. Implement code to compute cost function $J(\Theta)$
  4. Implement backprop to to compute partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$
  5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$. Then disable gradient checking.
  6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

     $J(\Theta)$ is non-conves, it is susceptible to local minima

- A vectorised implementation

$$
\begin{aligned}
X &= \begin{pmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & & & \vdots \\ 1 & x_1^m & x_2^m & \cdots & x_n^m \end{pmatrix} \in \mathbb{R}^{m\times(n+1)} \\
Y &= \left[\begin{bmatrix}1\\0\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\\vdots\\0\end{bmatrix}, \cdots, \begin{bmatrix}0\\0\\0\\0\\\vdots\\1\end{bmatrix} \right] \in \mathbf{Boolean}^{(m)\times(K)} \\
&\Theta^{(j)} \in \mathbb{R}^{s_{(l+1)}\times(s_l+1)} \\
&a^{(l)} = g(a^{(l-1)}(\Theta^{(l-1)})^T) \\
&a^{(l)} = [\text{ones}(m,1), a^{(l)}] \\
&h_\Theta(X) = a^{(L)} \\
&\theta = \text{“unroll”} \quad \Theta^{(1,\cdots,L)}(:,2:(s_l+1))
\end{aligned}
$$

$$J(\theta) = \frac{1}{m}\cdot\sum((-Y\odot\log(h) - (1-Y)\odot\log(1-h))) + \frac{\lambda}{2m}\theta^T\theta \tag{27}$$

$$\delta^{(L)} = a^{(L)} - Y \tag{28}$$

$$\delta^{(l)} = (\delta^{(l+1)}\Theta^{(l)}))\odot a^{(l)}\odot(1-a^{(l)}) \tag{29}$$

$$\delta^{(l)} = \delta^{(l)}(:,2:end) \tag{30}$$

$$\Theta^{(l)}(:,1) = 0 \tag{31}$$

$$D^{(l)} = \frac{1}{m}(\delta^{(l+1)})^T a^{(l)} + \frac{\lambda}{m}\Theta^{(l)} \tag{32}$$

7 Machine learning diagnostic

7.1 debugging a learning algorithm (trouble shooting)

- get more training examples → fixes high variance
- try smaller sets of features → fixes high variance
- try getting additional features → fixes high bias
- try adding polynomial features → fixes high bias
- try increasing or decreasing λ → fixes high bias/variance

7.2 Evaluating a hypothesis

Training/testing procedure:
Learning parameter Θ from training data (minimizing training error $J_{\text{train}}(\Theta)$)
Compute test set error $J_{\text{test}}(\Theta)$) (e.g., split up the data: Training set (random 70%) vs. Test set (random 30%)):

- for linear regression
  $J_{\text{test}}(\Theta) = \frac{1}{2m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} (h_\Theta(x_{\text{test}}^{(i)}) - y_{\text{test}}^{(i)})^2$
- for logistic regression
  $J_{\text{test}}(\Theta) = -\frac{1}{2m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} (y_{\text{test}}^{(i)} \log h_\Theta(x_{\text{test}}^{(i)}) + (1 - y_{\text{test}}^{(i)}) \log(1 - h_\Theta(x_{\text{test}}^{(i)}))$
- Misclassification error (0/1 misclassification error):

$$err(h_\Theta(x), y) = \begin{cases} 1 & \text{if} \quad h_\Theta(x) \geq 0.5, y = 0; \\ & \text{or if} \quad h_\Theta(x) \leq 0.5, y = 1. \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

$$\text{Test error} = \frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} err(h_\Theta(x_{\text{test}}^{(i)}, y_{\text{test}}^{(i)}) \tag{34}$$

The proportion of the test data that was misclassified.

7.3 Model selection and train/validation/test sets

Just because a learning algorithm fits a training set well, that doesn't mean it's a good hypothesis. The training error is likely to be lower than the actual generalization error.

Example Given many models with different polynomial degrees: split the data into: e.g., Training set (random 60%); Cross validation set (random 20%); Test set (random 20%).

- Training error: $J(\Theta)$, optimize the parameters Θ using the training set for each polynomial degree.
- Cross validation error: $J_{\text{cv}}(\Theta)$, find the polynomial degree d with the least error using the cross validation set. Pick the hypothesis with the lowest cross validation error.
- Test error: $J_{\text{test}}(\Theta^{(d)})$ The test set is then used to estimate the generalization error of the model that was selected.

7.4 Diagnosing bias vs. variance

Training error: $J_{\text{train}}(\Theta) = \frac{1}{2m_{\text{train}}} \sum_{i=1}^{m_{\text{train}}} (h_\Theta(x_{\text{train}}^{(i)}) - y_{\text{train}}^{(i)})^2$
Cross validation error: $J_{\text{cv}}(\Theta) = \frac{1}{2m_{\text{cv}}} \sum_{i=1}^{m_{\text{cv}}} (h_\Theta(x_{\text{cv}}^{(i)}) - y_{\text{cv}}^{(i)})^2$

High bias: underfitting; $J_{\text{train}}\Theta \approx J_{\text{cv}}\Theta$
High variance: overfitting; $J_{cv}\Theta \gg J_{\text{train}}\Theta$

Bias-variance tradeoff: Models with high bias are not complex enough for the data and tend to underfit, while models with high variance might overfit the training data.

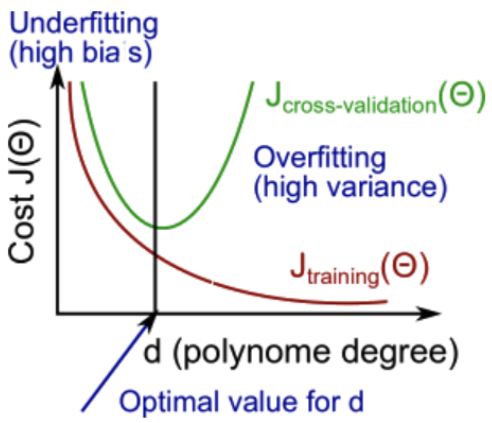

Figure 2: Training error vs. Cross validation error

7.4.1 Regularization and Bias/Variance

Linear regression with regularization

$$\text{Model} : h_\Theta(x) = \Theta_0 + \Theta_1 x + \Theta_2 x^2 + \Theta_3 x^3 + \Theta_4 x^4$$
$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2$$
$$J_{\text{train}}(\Theta) = \frac{1}{2m_{\text{train}}}\sum_{i=1}^{m_{\text{train}}}(h_\Theta(x_{\text{train}}^{(i)}) - y_{\text{train}}^{(i)})^2$$
$$J_{\text{cv}}(\Theta) = \frac{1}{2m_{\text{cv}}}\sum_{i=1}^{m_{\text{cv}}}(h_\Theta(x_{\text{cv}}^{(i)}) - y_{\text{cv}}^{(i)})^2$$
$$J_{\text{test}}(\Theta) = \frac{1}{2m_{\text{test}}}\sum_{i=1}^{m_{\text{test}}}(h_\Theta(x_{\text{test}}^{(i)}) - y_{\text{test}}^{(i)})^2$$

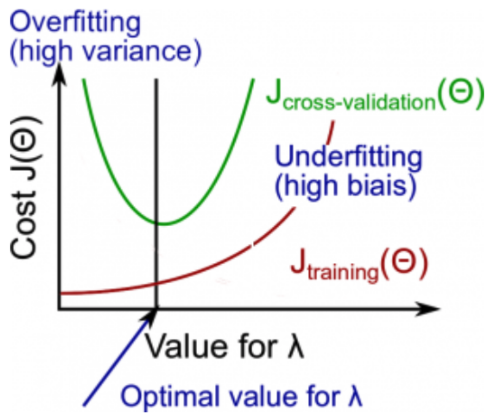

Figure 3: Regularization and the hypothesis

How to automatically choose a good regularization parameter λ?

Try $\lambda = \{0, 0.01, 0.02, 0.04, \cdots, 10\} \rightarrow \min J(\Theta) \rightarrow \Theta \rightarrow J_{\text{cv}}(\Theta) \rightarrow J_{\text{test}}(\Theta)$
$J_{\text{cv}}(\Theta)$ without regularization ($\lambda = 0$).

The training set is used to learn the model parameters θ at a given λ. The cross validation set is used to evaluate how good each λ value is and the best λ is selected accordingly.

- large λ: high bias (underfit)
- intermediate λ: "just right"
- small λ: high variance (overfit)

7.4.2 Learning curves

If a learning algorithm is suffering from high bias, getting more training data will not help much.
If a learning algorithm is suffering from high variance, getting more training data is likely to help.
Plotting learning curves can often help figure out the problem suffering by the algorithm.

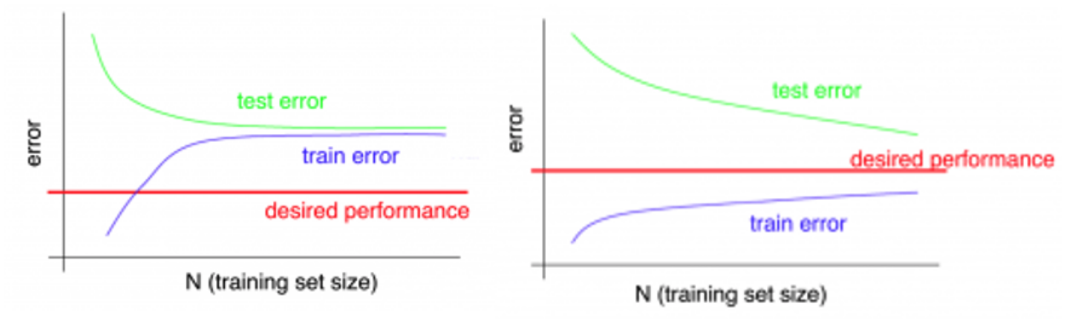

Figure 4: Typical learning curve for high bias (left) and high variance (right) at fixed model complexity

8 Machine learning system design

8.1 Prioritizing what to work on

Example: Building a spam classifier
Supervised learning.
x = features of email. y = spam (1) or not spam (0).
Features x: Choose 100 words indicative of spam/not spam $\rightarrow$ encode it into a feature vector, $x \in \mathbb{R}^{100}$

$$x_j = \begin{cases} 1 & \text{if word } \quad j \quad \text{appears in the email} \\ 0 & \text{otherwise} \end{cases}$$

In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather that manually pick 100 words. How to spend your time to make it have low error?

- Collect lots of data, e.g., "honeypot" project, but doesn't always work
- Develop sophisticated features based on email routing information (from email header)
- Develop sophisticated features for message body, e.g., "discount" vs. "discounts", "deal" vs. "Dealers"
- Develop sophisticated algorithm to detect misspellings

8.2 Error analysis

- Start with a simple algorithm, implement it quickly, and test it early on the cross validation data, e.g., quick and dirty running to see if the different ideas are improving the performance of the algorithm, to decide what to fold in and to incorporate into the learning algorithm.

- plot learning curves of training and test errors to figure out what problems the learning algorithms suffer, e.g., more data, more features are likely to help. "premature optimization".

- Error analysis: manually examine the examples (in the cross validation set) that the algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.
  Example spam classifier: what type of email it is; what cues (features) you think would have helped the algorithm classify them correctly.
  Numerical evaluation, e.g., a single real number evaluation metric, cross validation error, accuracy, error.
  "stemming" software. e.g., "Porter stemmer".

8.3 Error metrics for skewed classes

Example: Cancer classification example. Train logistic regression model $h_\theta(x)$. ($y = 1$ if cancer, $y = 0$ otherwise)
got 1% error on test set.
only 0.5% patients have cancer
skewed classes: a lot of more examples from one class than the other class. It becomes much harder to use classification accuracy.

8.3.1 Precision/recall

as an evaluation metric for classification problems with skewed classes.

$y = 1$ in presence of rare class to be detected.

	Actual 1	Actual 0
Predicted 1	True positive	False positive
Predicted 0	False negative	True negative

Precision
$$\frac{\text{True positives}}{\sharp\text{predicted as positive}} = \frac{\text{True positives}}{\text{True positives + False positives}}$$

Recall
$$\frac{\text{True positives}}{\sharp\text{actual positives}} = \frac{\text{True positives}}{\text{True positives + False negatives}}$$

If a classifier is getting high precision and high recall, we are confident that the algorithm has to be doing well, even if we have very skewed classes.

8.3.2 Trading off precision and recall

Logistic regression: $0 \leq h_\theta(x) \geq 1$
Predict 1 if $h_\theta(x) \geq 0.5 \to 0.7, 0.9, 0.3$
Predict 0 if $h_\theta(x) < 0.5 \to 0.7, 0.9, 0.3$

To control the trade-off by choosing the threshold parameter.
Suppose we want to predict $y = 1$ (cancer) only if very confident.
$\to$ Higher precision, lower recall
Suppose we want to avoid missing too many cases of cancer (avoid false negatives).
$\to$ Higher recall, lower precision

F_1 **Score** for comparing precision/recall numbers. "Average" of P and R.

$\frac{2}{\frac{1}{P}+\frac{1}{R}}$, "Harmonic mean".

$2\frac{PR}{P+R}$, F score is one of the possible formulas for combing precision and recall.

8.4 Data for machine learning

"It's not who has the best algorithm that wins. It's who has the most data.".

Large data rationale
Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately.
Useful test: Given the input x, can a human expert confidently predict y?

Use a learning algorithm with many parameters, e.g., neural network with many hidden units (low bias algorithms).
$\to J_{\text{train}}(\theta)$ will be small.

Use a very large training set (unlikely to overfit, low variance)
$\to J_{\text{test}}(\theta)$ will be small.
$\to J_{\text{test}}(\theta) \approx J_{\text{train}}(\theta)$.

9 Support vector machines

9.1 Optimization objective

Alternative view of logistic regression

$$h_\theta(x) = g(z), \quad z = \theta^T x, \quad g(z) = \frac{1}{1+e^{-z}}$$

If $y = 1$, we want $h_\theta(x) \approx 1, \theta^T x \gg 0$
If $y = 0$, we want $h_\theta(x) \approx 0, \theta^T x \ll 0$

Cost of example:

$$\begin{aligned} & -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x)) \\ = & -y\log\left(\frac{1}{1+e^{-\theta^T x}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{-\theta^T x}}\right) \end{aligned}$$

A single training example contributes to the overall objective function.
If $y=1$, we want $\theta^T x \gg 0 \rightarrow -\log(\frac{1}{1+e^{-\theta^T x}}) \approx 0$
If $y=0$, we want $\theta^T x \ll 0 \rightarrow -\log(1-\frac{1}{1+e^{-\theta^T x}}) \approx 0$

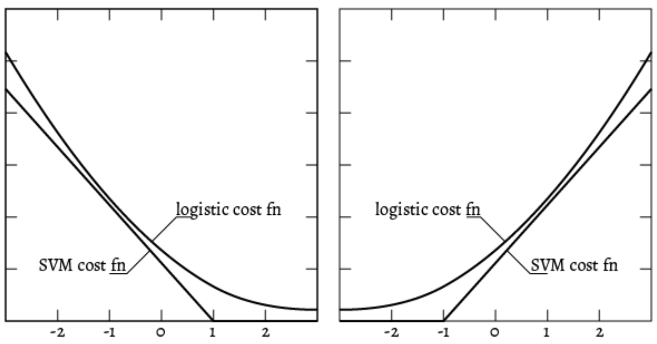

Figure 5: Cost function of logistic regression vs. SVM

Logistic regression:

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

Support vector machine (with hinge loss function):

$$\min \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})(\text{cost}_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{35}$$

$$\min C\sum_{i=1}^{m}\left[y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})(\text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 \tag{36}$$

$$\text{cost}_0 = \max(0, k(1+z))$$
$$\text{cost}_1 = \max(0, k(1-z))$$

Hypothesis:

$$h_\theta(x) = \begin{cases} 1 & \text{if} \quad \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Informally, the parameter C is a positive value that controls the penalty for misclassified training examples. A large C tells the SVM to try to classify all the examples correctly. C plays a role similar to $\frac{1}{\lambda}$.

9.2 Large margin intuition

If $y = 1$, we want $\theta^T x \geq 1$ (not just $\theta^T x \geq 0$)
If $y = 0$, we want $\theta^T x < -1$ (not just $\theta^T x < 0$)
This builds in an extra safety factor or safety margin factor into the support vector machine.

9.2.1 SVM decision boundary

If C is very large (e.g., 100,000), then minimizing the optimization objective leads to setting

$\sum_{i=1}^{m}\left[y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right]$ to 0:

Whenever $y^{(i)} = 1 : \theta^T x^{(i)} \geq 1$
Whenever $y^{(i)} = 0 : \theta^T x^{(i)} \leq -1$

$$\begin{aligned} & \min_{\theta}\frac{1}{2}\sum_{i=1}^{n}\theta_j^2 \\ s.t. \quad & \theta^T x^{(i)} \geq 1, \quad \text{if} \quad y^{(i)} = 1 \\ & \theta^T x^{(i)} \leq -1, \quad \text{if} \quad y^{(i)} = 0 \end{aligned} \tag{37}$$

- Linearly separable case (Large margin classifier)
  a robust separator, a larger minimum distance from any of the training examples. This distance is called the **margin** of the support vector machine and this gives the SVM a certain robustness, because it tries to separate the data (positive vs. negative examples) with as large a margin as possible.

- Large margin classifier in presence of outliers
  When C is not very large, SVM can do a better job ignoring the few outliers.

9.2.2 Maths behind large marge classification

Vector inner product:

$$\begin{aligned} u &= \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \\ \|u\| &= \text{the norm or the euclidean length of the vector } u. \\ &= \sqrt{u_1^2 + u_2^2} \in \mathbb{R} \\ P &= \text{the signed length of the projection of } v \text{ on to } u, P \in \mathbb{R} \\ u^T v &= P \cdot \|u\| = u_1 v_1 + u_2 v_2 \\ &= v^T u \end{aligned}$$

SVM decision boundary

$$\begin{aligned}
&\min_{\theta} \frac{1}{2} \sum_{i=1}^{n} \theta_j^2 \\
&= \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2 \qquad (38) \\
s.t.\quad &\theta^T x^{(i)} \geq 1, \text{ if } y^{(i)} = 1 \\
&\theta^T x^{(i)} \leq -1, \text{ if } y^{(i)} = 0 \\
&\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\| = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} \qquad (39) \\
&\text{where } p^{(i)} \text{ is the projection of } x^{(i)} \text{ onto the vector } \theta \\
&\text{Simplification: } \theta_0 = 0 \\
&\text{i.e., the decision boundary must pass through the origin, (0,0)} \\
\rightarrow & \\
s.t.\quad &p^{(i)} \cdot \|\theta\| \geq 1, \text{ if } y^{(i)} = 1 \\
&p^{(i)} \cdot \|\theta\| \leq -1, \text{ if } y^{(i)} = 0
\end{aligned}$$

The vector for θ is perpendicular to the decision boundary. In order for the optimization objective (above) to hold true, we need the absolute value of our projections $p^{(i)}$ to be as large as possible.

The magnitude of the margin (the gap that separates positive and negative examples) is exactly the values of $p^{(i)}$. By making the projections ($p^{(i)}$) large, the SVM can end up with a smaller value for $\|\theta\|$ in the objective.

9.3 Kernels

9.3.1 Non-linear decision boundary

One way to do so is to come up with a set of complex polynomial features:

Predict $y = 1$,
if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + +\theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$

Another notation:
$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \cdots$, where $f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, \cdots$

Is there a different / better choice of the features $f_1, f_2, f_3, \cdots$?
Given x, compute new features depending on the proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

$$\begin{aligned}
\text{Given} x: \quad f_1 &= \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\left\|x - l^{(1)}\right\|^2}{2\sigma^2}\right) \\
f_2 &= \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\left\|x - l^{(2)}\right\|^2}{2\sigma^2}\right) \\
f_3 &= \text{similarity}(x, l^{(3)}) = \exp\left(-\frac{\left\|x - l^{(3)}\right\|^2}{2\sigma^2}\right)
\end{aligned}$$

9.3.2 Kernels and similarity

similarity = kernel[7], i.e., $k(x, l^{(i)})$, here is Gaussian kernels. σ determines how fast the similarity metric decreases (to 0) as the examples are further apart.

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\left\|x - l^{(1)}\right\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}\left\|x_j - l_j^{(1)}\right\|^2}{2\sigma^2}\right) \tag{40}$$

$$\text{If } x \approx l^{(1)}: \quad f_1 \approx \exp\left(-\frac{0}{2\sigma^2}\right) \approx 1$$

$$\text{If } x \text{ is far from } l^{(1)}: \quad f_1 \approx \exp\left(-\frac{\infty}{2\sigma^2}\right) \approx 0$$

These features measure how similar (close) x is from each of the landmarks.

The hypothesis given these features are:
Predict 1 when
$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + +\theta_3 f_3 \geq 0$
The extra features are defined using landmarks and similarity functions to learn more complex nonlinear classifier.

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \cdots$?

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \cdots, l^{(m)} = x^{(m)}$.

7 Kernels generalize the notion of "inner product similarity"

Given example x:

$$
\begin{aligned}
f_1 &= \text{similarity}(x, l^{(1)}) \\
f_2 &= \text{similarity}(x, l^{(2)}) \\
f &= \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}, f_0 = 1
\end{aligned}
$$

For training example $(x^{(i)}, y^{(i)})$:

$$
\begin{aligned}
f_1^{(i)} &= \text{similarity}(x^{(i)}, l^{(1)}) \\
f_2^{(i)} &= \text{similarity}(x^{(i)}, l^{(2)}) \\
&\vdots \\
f_i^{(i)} &= \text{similarity}(x^{(i)}, l^{(i)}) = \exp\left(-\frac{0}{2\sigma^2}\right) = 1 \\
&\vdots \\
f_m^{(i)} &= \text{similarity}(x^{(i)}, l^{(m)}) \\
x^{(i)} \in \mathbb{R}^{n+1} \rightarrow f^{(i)} &= \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \in \mathbb{R}^{m+1}
\end{aligned}
$$

9.3.3 SVM with kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$
Pridict "$y = 1$", if $\theta^T f \geq 0$, $\theta \in \mathbb{R}^{m+1}$
Training:

$$
\min C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)})(\text{cost}_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2 \tag{41}
$$

Note that the effective number of features is the dimension of f, so that the number of features n is going to be equal to the number of training samples m. SVM uses $\theta^T M \theta$ instead of $\theta^T \theta$, which allows it to scale to much bigger training sets (i.e., an implementational detail, primarily for reasons of computational efficiency). The computational tricks that apply for SVM don't generalize well to other algorithms like logistic regression.

9.3.4 SVM bias and variance trade-offs

- $C(\approx \frac{1}{\lambda})$
  Large C: Lower bias, high variance.
  Small C: High bias, low variance.

- σ^2
  Large σ^2: Features f_i vary more smoothly. Higher bias, lower variance.
  Small σ^2: Features f_i vary less smoothly. Lower bias, higher variance.

9.4 Using an SVM

Use SVM software package (e.g., liblinear, libsvm, $\cdots$) to solve for parameters θ.
Need to specify:

- Choice of parameter C

- Chocie of kernel (similarity function)

  - No kernel ("linear kernel"):
    Predict "$y = 1$", if $\theta^T x \geq 0$. In case n is large and m is small, $x \in \mathbb{R}^{n+1}$
  - Gaussian kernel:
    $f_i = \exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$. In case n is small and m is large, $x \in \mathbb{R}^{n+1}$
    Need to choose σ^2.
    Note: Do perform feature scaling before using the Gaussian kernel.

  - Other choices of kernels:
    Note: Not all similarity functions similarity(x, l) make valid kernels. Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge.
    Many off-the-shelf kernels available:
    Polynomial kernel: $k(x, l) = (x^T l)^2, (x^T l)^3, (x^T l{+}1)^2, (x^T l{+}5)^4, (x^T l{+}\text{constant})^{\text{degree}}$, etc.
    More esoteric: String kernel, chi-square kernel, hitogram intersection kernel, etc.

- Multi-class classification
  One-vs.-all method: $y \in \{1, 2, \cdots, K\}$. Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \cdots, K$, get $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(K)}$. Pick class i with largest $(\theta^{(i)})^T x$.

9.5 Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples
If n is large (relative to m), e.g., $n \gg m$:
  Use logistic regression, or SVM without a kernel ("linear kernel").
If n is small, m is intermediate:
  Use SVM with Gaussian kernel.
If n is small, m is large:
  Create/add more features, then use logistic regression or SVM with a kernel.

10 Unsupervised learning

- Supervised learning:
  Given a labeled training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$, to find the decision boundary that separates the positive labeled examples and the negative labeled examples.
  $\rightarrow$ Given a set of labels to fit a hypothesis.

- Unsupervised learning:
  Given data that does not have any labels. Training set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$.
  The algorithm is to find some structure (pattern) in the data.

10.1 Clustering

Example applications: Market segmentation; Social network analysis; Organizing computer clusters; Astronomical data analysis, etc.

Input:

- K (number of clusters)
- Training set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
  $x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

10.1.1 K-means algorithm

```
Randomly initialize $K$ cluster centroids $\mu_1, \mu_2, \cdots, \mu_K \in \mathbb{R}^n, k \in \{1, 2, \cdots, K\}$
  Repeat {

  –*Cluster assignment*–
  for $i = 1$ to $m$
    $c^{(i)}$ := index (from 1 to $K$) of cluster centroid closest to $x^{(i)}$.
    $c^{(i)} = \underset{k}{\operatorname{argmin}} \|x^{(i)} - \mu_k\|^2$

  –minimize $J$, w.r.t., $c^1, c^2, \cdots, c^m$, holding $\mu_1, \mu_2, \cdots, \mu_k$ fixed–

  –*Move centroid*–
  for $k = 1$ to $K$
    $\mu_k$ := average (mean) of points assigned to cluster $k$
    $\mu_k = \frac{1}{n}[x^{(k_1)} + x^{(k_2)} + \cdots + x^{(k_n)}] \in \mathbb{R}^n$

  –minimize $J$, w.r.t., $\mu_1, \mu_2, \cdots, \mu_k$–
}
```

K-means is also applied to data sets where there may not be several well separated clusters. K-means can still evenly segment such data into K subsets, so can still be useful in this case.

- Optimization objective (distortion function):
  $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

$$J(c^1, c^2, \cdots, c^m, \mu_1, \mu_2, \cdots, \mu_k) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2 \tag{42}$$
$$\min_{\substack{c^1, c^2, \cdots, c^m \\ \mu_1, \mu_2, \cdots, \mu_k}} J(c^1, c^2, \cdots, c^m, \mu_1, \mu_2, \cdots, \mu_k)$$

  With k-means, it is not possible for the cost function to sometimes increase. J should always decrease as K is increased.

- Random initialization:
  - Should have $K < m$
  - Randomly pick K training examples.
  - Set $\mu_1, \mu_2, \cdots, \mu_k$ equal to these K examples.

  Try multiple random initialization to ameliorate K-means getting stuck in local optima. When the initial centroids are selected, be sure that they are each unique.

```
for $i = 1, 2, \cdots 100${
    Randomly initialize K-means.
    Run K-means. Get $c^1, c^2, \cdots, c^m, \mu_1, \mu_2, \cdots, \mu_k$.
    Compute cost function (distortion):
    $J(c^1, c^2, \cdots, c^m, \mu_1, \mu_2, \cdots, \mu_k)$
}
```

  Pick clustering that gave lowest cost $J(c^1, c^2, \cdots, c^m, \mu_1, \mu_2, \cdots, \mu_k)$

- Choosing the number of clusters (the right value of K): The value of K can be ambiguous, and hard to be decided.

  Manually by looking at visualizations, output of the clustering algorithm, etc.

  Elbow method (worth the shot but no necessary high expectation of it working).

  Evaluate K-means based on a metric for how well it performs for that later purpose.

10.2 Dimensionality reduction

Data compression:
Example: Reduce data from 3D to 2D. $x \in \mathbb{R}^3 \rightarrow z \in \mathbb{R}^2$
Data visualization:
Example: $x \in \mathbb{R}^{50} \rightarrow z \in \mathbb{R}^{2 \text{ or } 3}$

10.2.1 Principal component analysis

Example:

- 2D $\rightarrow$ 1D, find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error (i.e., the average of all the distances of every feature to the projection line).

- n-D $\rightarrow$ k-D, k vector $(u^{(1)}, u^{(2)}, \cdots, u^{(k)})$ onto which to project the data so as to minimize the projection error.

PCA is not linear regression, e.g., the "projection error" (i.e., shortest orthogonal distances) is different from the "mean squared error" (i.e., vertical distances to the label), all the features are treated symmetrically. PCA trys to find a lower dimensional surface onto which to project the data, so as to minimize the squared project error.

10.2.2 PCA algorithm

Training set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$.
Preprocessing (feature scaling/mean normalization):

- $\mu_j = \frac{1}{m}\sum_{i=1}^m x_j^{(i)}$ (mean of each features)
  Replace each x_j^i with $x_j^i - \mu_j$ (so that each feature has zero mean)

- If different features on different scales, scale features to have comparable range of values.
  $x_j^i \leftarrow \frac{x_j^i - \mu_j}{s_j}$

Reduce data from $n-$dimensions to $k-$dimensions: $x^{(i)} \in \mathbb{R}^{n\times 1} \rightarrow z^{(i)} \in \mathbb{R}^{k\times 1}$
Compute "covariance matrix":

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T \in \mathbb{R}^{n\times n} \tag{43}$$

Compute "eigenvectors or singular vectors " of matrix Σ, e.g., singular value decomposition [8] $\Sigma = USV^T$, here $\Sigma \in \mathbb{R}^{n\times n}$, so $U \in \mathbb{R}^{n\times n}$[9], $S \in \mathbb{R}^{n\times n}$ diagonal, $V \in \mathbb{R}^{n\times n}$. We take the first k vectors of the matrix $U \in \mathbb{R}^{n\times n}$ as $U_{\text{reduced}} \in \mathbb{R}^{n\times k}$

$$z^{(i)} = U_{\text{reduced}}^T x^{(i)} \tag{44}$$

$$X \in \mathbb{R}^{m\times n}, \Sigma = \frac{1}{m}X^T X$$

$$Z = XU_{\text{reduced}} \tag{45}$$

10.2.3 Reconstruction from compressed representation

Example: $z^{(i)} \in \mathbb{R} \rightarrow x^{(i)} \in \mathbb{R}^2$, go back to the original number of features and get approximations of the original data.

$$x_{\text{approx}}^{(i)} = U_{\text{reduced}} \cdot z^{(i)} \tag{46}$$

$$Z \in \mathbb{R}^{m\times k}$$

$$X_{\text{approx}} = ZU_{\text{reduced}}^T \tag{47}$$

8 a factorization of a real or complex matrix that generalize the eigendecomposition.
9 U is a Unitary matrix, $U^{-1} = U^H$, with real numbers here so $U^{-1} = U^T$

10.2.4 Choosing the number of PCs

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{\text{approx}}\|^2$
(The difference between the original data and the projection on the lower dimensional surface).

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$
(The average distance of the training examples from the origin).

Typically, choose k to be smalles value such that

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{\text{approx}}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%)$$

(99% of variance is retained)

Algorithm:

Try PCA with $k = 1, 2, \cdots$

Compute $U_{\text{reduced}}, z^{(1)}, z^{(2)}, \cdots, z^{(m)}, x^{(1)}_{\text{approx}}, x^{(2)}_{\text{approx}}, \cdots, x^{(m)}_{\text{approx}}$

Check if $\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{\text{approx}}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01$

Algorithm:
S: a diagonal matrix, with the non-negative sigular values of Σ on the diagonal, by convention $s_1 \geq s_2 \cdots \geq s_k \geq 0$

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{\text{approx}}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}}$$

Pick smallest value of k for which

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

10.2.5 Supervised learning speedup

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
Extract inputs:
Unlabeled dataset: $x^{(1)}, x^{(2)}, \cdots, x^{(m)} \in \mathbb{R}^{10,000}$
apply PCA $\rightarrow z^{(1)}, z^{(2)}, \cdots, z^{(m)} \in \mathbb{R}^{1000}$
New training set: $\{(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \cdots, (z^{(m)}, y^{(m)})\}$

PCA defines a mapping from x to z and this mapping should be defined by running PCA (including the feature scaling and mean normalization) only on the training set, not to the cross validation or test sets. Having found all the parameters on the training set, the same mapping can then be applied to other examples in the cv or test sets. Bad use of PCA: trying to prevent overfitting, while PCA does not consider the labels y. Try the full machine learning algorithm without PCA first.

11 Anomaly detection

Density estimation:
Given dataset: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
Is the new example x_{test} anomalous?
Define a model $p(x)$ (the probability density that the example is not anomalous)
$p(x_{\text{test}}) < \epsilon \rightarrow$ flag anomaly
$p(x_{\text{test}}) \geq \epsilon \rightarrow$ OK

11.1 Anomaly detection example

- Fraud detection:
  - $x^{(i)}$ = features of user i's activities
  - Model $p(x)$ from data
  - Identify unusual users by checking which have $p(x) < \epsilon$ (anomaly detector flags too many anomalous examples $\rightarrow$ decrease ϵ)
- Manufacturing
- Monitoring computers in a data center.
  - $x^{(i)}$ = features of machine i
  - x_1 = memory use, x_2 = number of disk accesses/sec, x_3 = CPU load, x_4 = network traffic.

11.2 Gaussian distribution

$x \in \mathbb{R}$ is a distributed Gaussian with mean μ, variance σ^2.

$$X \sim \mathcal{N}(\mu,\, \sigma^2) \tag{48}$$

$$p(x; \mu,\, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{49}$$

11.2.1 Parameter estimation

Given the dataset: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
to estimate what are the values of μ, σ^2:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{50}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2 \tag{51}$$

Maximum likelihood estimation.

11.3 Anomaly detection algorithm

Given the unlabelled training set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}, \quad x^{(i)} \in \mathbb{R}^n$

$$\begin{aligned} p(x) &= p(x_1 \mid \mu_1, \sigma_1^2)p(x_2 \mid \mu_2, \sigma_2^2)p(x_3 \mid \mu_3, \sigma_3^2)\cdots p(x_n \mid \mu_n, \sigma_n^2) \\ &= \prod_{j=1}^{n} p(x_j \mid \mu_j, \sigma_j^2) \end{aligned}$$

Independence assumption.

1. Choose features x_j that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \mu_2, \cdots, \mu_n, \sigma_1^2, \sigma_2^2, \cdots, \sigma_n^2$,

$$\begin{aligned} \mu_j &= \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \\ \sigma_j^2 &= \frac{1}{m}\sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2 \end{aligned}$$

3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j \mid \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right) :$$

   Anomaly if $p(x) < \epsilon$

11.3.1 Developing and evaluating an anomaly detection system

The importance of real-number evaluation: When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous (e.g., 20) and non-anomalous (e.g. 10000) examples. ($y = 0$ if normal, $y = 1$ if anomalous).

Training set (e.g., 6000 good): $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$ (assume normal examples / not anomalous)
$\rightarrow p(x) = \prod_{j=1}^{n} p(x_j \mid \mu_j, \sigma_j^2)$

Cross validation set (e.g., 2000 good and 10 anomalous): $\{(x_{\text{cv}}^{(1)}, y_{\text{cv}}^{(1)}), (x_{\text{cv}}^{(2)}, y_{\text{cv}}^{(2)}), \cdots, (x_{\text{cv}}^{(m)}, y_{\text{cv}}^{(m)})\}$

Test set (e.g., 2000 good and 10 anomalous): $\{(x_{\text{test}}^{(1)}, y_{\text{test}}^{(1)}), (x_{\text{test}}^{(2)}, y_{\text{test}}^{(2)}), \cdots, (x_{\text{test}}^{(m)}, y_{\text{test}}^{(m)})\}$

Algorithm evaliuation

Fit model $p(x)$ on training set $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$, assuming the vast majority are normal examples

On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if} \quad p(x) < \epsilon \quad \text{anomaly} \\ 0 & \text{if} \quad p(x) \geq \epsilon \quad \text{normal} \end{cases}$$

Possible evaluation metrics, see Section 8:
(Because the data is very skewed ($y = 0$ is much more common), classification accuracy would not be a good evaluation metrics)

- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1−score

Can also use cross validation set to choose parameter ϵ.

11.3.2 Anomaly detection vs. supervised learning

Anomaly detection	Supervised learning
Very small number of positive examples ($y = 1$), typically $0 - 20$. Large number of negative ($y = 0$) examples	Large number of positive and negative examples.
Many different "types" of anomalies. hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.	Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.

In anomaly detection, often the number of positive examples is so small that it is not possible for a learning algorithm to learn that much from them. Instead, we take a large set of negative examples and have the learning algorithm learn a lot from them. The small number of positive examples are reserved for evaluating the algorithms.

Application

Anomaly detection	Supervised learning
Fraud detection	Email spam classification
Manufacturing (e.g., aircraft engines) Monitoring machines in a data center $\vdots$	Weather prediction (sunny/rainy/ect.) Cancer classification $\vdots$

If there are equal numbers of positive and negative examples, we would tend to shift all the applications to the supervised learning column. Anomaly detection only models the negative examples, whereas a supervised learning algorithm learns to discriminate between positive and negative examples, so the supervised learning algorithm will perform better when there are many positive and negative examples.

11.3.3 Choosing what features to use

Sanity check: Plot the histogram of the data, to make sure that the data looks vaguely Gaussian before feeding it to the anomaly detection algorithm.
If the data looks non-Gaussian, play with different transformations of the data in order to make it look more Gaussian, e.g., $x \leftarrow \log(x)$, $x \leftarrow \log(x + c)$, $x \leftarrow x^{\frac{1}{2}}$, $x \leftarrow x^{\frac{1}{3}}$.

11.3.4 Error analysis for anomaly detection

Want:

$p(x)$ large for normal examples x.

$p(x)$ small for anomalous examples x.

Most common problem:

$p(x)$ is comparable (e.g., both large) for normal and anomalous examples.

Look at the anomaly that the algorithm is failing to flag (finding something unusual), and use that to create some new features, so that with these new features it becomes easier to distinguish the anomalies from the good examples.

Example: Monitoring computers in a data center
Choose features that might take on unusually large or small values int he event of an anomaly.

x_1 = memory use of computer

x_2 = number of disk accesses/sec

x_3 = CPU load

x_4 = network traffic

Create a new feature e.g., $x_5 = \frac{\text{CPU load}}{\text{network traffic}}$, $x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$

11.4 Multivariate Gaussian distribution

$x \in \mathbb{R}^n$. Instead of model $p(x_1), p(x_2), \cdots$, etc. separately, model $p(x)$ all in one go.
Parameters: $\mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n\times n}$ (covariance matrix).

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{52}$$

Note that this $p(x)$ is for a single training example. An efficient vectorized implementation with the whole training data matrix X might involve summation over dot products.

Multivariate Gaussian (Normal) examples
$\mu = \begin{bmatrix}0\\0\end{bmatrix}, \Sigma = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$ vs.

- $\Sigma = \begin{bmatrix}0.6 & 0\\0 & 0.6\end{bmatrix} \rightarrow$
  narrower distribution (the concentric ellipsis shrunk a little bit)

- $\Sigma = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix} \rightarrow$
  wider distribution (the variability became wider)

- $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \rightarrow$
  the distribution falls off more slowly in one direction and falls off very rapidly in the other direction
- $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix} \rightarrow$
  change the off diagonal entries to model correlations between data.
- $\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}, \mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \rightarrow$
  change the peak of the distribution

11.5 Anomaly detection using the multivariate Gaussian distribution

Parameter fitting: Given training set $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$
$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

2. Given a new example x, compute

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

   Flag an anomaly if $p(x) < \epsilon$

11.5.1 Relationship to the original model

The original model $p(x) = \prod_{j=1}^{n} p(x_j \mid \mu_j, \sigma_j^2)$
corresponds to a special case the of multivariate Gaussian distribution, by constructing

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & & & \\ & \sigma_2^2 & & 0 & \\ & & \cdot & & \\ & & & \cdot & \\ & 0 & & & \cdot & \\ & & & & & \sigma_n^2 \end{bmatrix}$$

the covariance matrix sigma must have 0's on the off diagonal elements. The controus of $p(x; \mu, \Sigma)$ are axis-aligned

Original model	Multivariate Gaussian
Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values, e.g., $x_3 = \frac{x_1}{x_2}$	Automatically captures correlations between features
Computationally cheaper (scales better to large n)	Computationally more expensive (Σ^{-1})
Okay even if m (training set size) is small	Must have $m > n$ or else Σ is non-invertible

If the covariance matrix Σ is singular or non-invertible, check for $m \gg n$, or redundant features (e.g., $x_1 = x_2, x_3 = x_4 + 5$).

12 Recommender systems

Example: Predicting movie ratings
User rates movies using zero (one) to five stars.
n_u = number of users
n_m = number of movies

Predict how the users would have rated other movies that they have not yet rated.

12.1 Content-based recommender systems

Suppose a set of features for each of the movies, e.g., x_1 romance, x_2 action, (on a scale of 0-1). For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

12.1.1 Problem formulation

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)
$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j) = 1$)

$\theta^{(j)}$ = parameter vector for user j
$x^{(i)}$ = feature vector for movie i
For user j, movie i, predicted rating: $(\theta^{(j)})^T x^{(i)}$

$m^{(j)}$ = number of movies rated by user j

To learn $\theta^{(j)} \in \mathbb{R}^{n+1}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

to simplify, eliminate the constant $m^{(j)} \rightarrow$

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

To learn $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$:

$$\begin{aligned} & J(\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}) \\ & = \min_{\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \end{aligned} \tag{53}$$

12.1.2 Optimization algorithm

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)}, (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right), (\text{for } k \neq 0)$$

12.2 Collaborative filtering

Given a dataset without knowledge of its features $\rightarrow$ feature learning (feature finders), e.g., given the user preference, to learn the movie features.

Given $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$ (and movie ratings), can estimate $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$.

Given $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$, can estimate $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$.

Back and forth process (simultaneously learn the features and the parameters):
θ (initial random guess) $\rightarrow x \rightarrow \theta \rightarrow x \cdots$

12.2.1 Optimization algorithm

Given $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$, to learn $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$:

$$\min_{x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 \tag{54}$$

12.2.2 Collaborative filtering algorithm

Combining Equations 53 and 54, minimizing $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$ and $\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$ simultaneously:

$$\begin{aligned} & J(x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}) \\ & = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 \end{aligned} \tag{55}$$

$$\min_{\substack{x^{(1)}, x^{(2)}, \cdots, x^{(n_m)} \\ \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}}} J(x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}) \tag{56}$$

Here $x \in \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$, no need of the convention for the interception term ($x_0 = 1, \theta_0$).

1. Initialize $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(n_u)})$
   Using gradient descent (or an advanced optimizatin algorithm),
   e.g., for every $j = 1, 2, \cdots, n_u, i = 1, 2, \cdots, n_m$:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$
$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

12.2.3 Vecotrization: low rank matrix factorization

Predicted ratings:

$$\begin{aligned}
&\text{Given } X \in \mathbb{R}^{n_m \times n}, \Theta \in \mathbb{R}^{n_u \times n} \\
&Y = X\Theta^T \\
&\begin{bmatrix} (\theta^{(1)})^T x^{(1)} & (\theta^{(2)})^T x^{(1)} & \cdots & (\theta^{(n_u)})^T x^{(1)} \\ (\theta^{(1)})^T x^{(2)} & (\theta^{(2)})^T x^{(2)} & \cdots & (\theta^{(n_u)})^T x^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T x^{(n_m)} & (\theta^{(2)})^T x^{(n_m)} & \cdots & (\theta^{(n_u)})^T x^{(n_m)} \end{bmatrix}
\end{aligned}$$

Mean normalization as a sort of pre-processing step for collaborative filtering, e.g., take the average ratings of the movies for the prediction of an new user who hasn't yet rated any movie.

12.2.4 Finding related products

for each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^{(n)}$.
How to find movies j related to movie i?
Find the most similar movies by using $\|x^{(i)} - x^{(j)}\|$

13 Learning with large datasets

To come up with computationally reasonable ways in large-scale ML. We mainly benefit from a very large dataset when our algorithm has high variance when m is small.

13.1 Stochastic gradient descent

Batch gradient descent (looking at all the training examples at a time);
Stochastic gradient descent (using one single training example each time).

Batch gradient descent Use all m examples in each iteration	**Stochastic gradient descent** Use 1 example in each iteration
$J_{\text{train}}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$	$\text{cost}(\theta, x^{(i)}, y^{(i)}) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ $J_{\text{train}}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}(\theta, x^{(i)}, y^{(i)})$
repeat until convergence $\{$ $\theta_j := \theta_j - \alpha\frac{1}{m}\underbrace{\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}J(\theta)}$ (for every $j = 0, \cdots, n$) $\}$	1. Randomly 'shuffle' dataset 2. repeat until convergence $\{$ for $i = 1, 2, \cdots, m\{$ $\theta_j := \theta_j - \alpha\underbrace{(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}_{\frac{\partial}{\partial\theta_j}\text{cost}(\theta, x^{(i)}, y^{(i)})}$ (for every $j = 0, \cdots, n$) $\}$ $\}$

When the training set size m is very large, stochastic gradient descent can be much faster. The cost function of batch gradient descent should go down with every iteration (assuming a well-tuned learning rate α) but not necessarily with stochastic gradient descent. Stochastic gradient descent will be unlikely to converge at the global minimum and will instead wander around it randomly, but usually yields a result that is close enough.

13.2 Mini-batch gradient descent

Mini-batch gradient descent can sometimes be even faster than stochastic gradient descent. Instead of using all m examples as in batch gradient descent, and instead of using only 1 example as in stochastic gradient descent, we will use some in-between number of examples b. Use b (mini-batch size, typically 2-100) examples in each iteration

1. Get b examples $\{(x^{(i)}, y^{(i)}), (x^{(i+1)}, y^{(i+1)}), \cdots, (x^{(i+b-1)}, y^{(i+b-1)})$
2. $\theta_j := \theta_j + \alpha\frac{1}{b}\sum_{k=i}^{i+b-1}((h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)})$
   (for every $j = 0, \cdots, n$)
3. $i := i + b$

With a good vectorizated implementation (parallelize the gradient computations over the b examples), mini-batch gradient descent is likely to outperform stochastic gradient descent, though the extra parameter b might take extra time.

13.3 Stochastic gradient descent convergence

13.3.1 Checking for convergence

Batch gradient descent:
Plot $J_{\text{train}}(\theta)$ as a funciton of the number of iterations of gradient descent.

Stochastic gradient descent:
During learning, compute $\text{cost}(\theta, x^{(i)}, y^{(i)}) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ before updating θ using $(x^{(i)}, y^{(i)})$
Every 1000 iterations (say), plot $\text{cost}(\theta, x^{(i)}, y^{(i)})$ averaged over the last 1000 or so examples processed by algorithm (increase the number of examples averaged over to plot the performance $\rightarrow$ smoother line).

Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge, e.g., $\alpha = \frac{\text{const1}}{\text{iterationNumber+const2}}, \alpha \rightarrow 0$

13.4 Online learning

With a continuous stream of users to a website, we can run an endless loop that gets (x, y), where we collect some user actions for the features in x to predict some behavior y.
Example: Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service $(y = 1)$, sometimes not $(y = 0)$.

A learning algorithm to help us to optimize what is the asking price that we want o offer to our users?
Feature x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x; \theta)$ to optimize price, e.g., using logistic regression.

```
Repeat forever {
Get $(x, y)$ corresponding to user.
Update $\theta$ using $(x, y)$
    $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
    $(j = 0, 1, \cdots, n)$ }
```

Look at one example at a time, and then discard that example. The algorithm can adapt to changing user preferences.

Example: Product search (learning to search)
The problem of learning the predicted click-through rate (CTR), i.e., learning the probablility that the user click on the specific link that the algorithm offer them $p(y = 1|x; \theta)$.

Other example: Choosing special offers to show user; customized selection of news articles; product recommendation; etc.

13.5 Map-reduce and data parallelism

We can divide up batch gradient descent and dispatch the cost function for a subset of the data to many different machines so that we can train our algorithm in parallel.
Example: 4x speedup. Take the training set, split it equally into 4 subsets and send the 4 subsets of the training data to 4 different computers, each computer computes the summation over one quarter of the training set, and then finally take each computer's results, send them to the centralized server which combines the results together.

1. Split the training set into z subsets corresponding to the number of machines
2. On each machine, calculate $\sum_{i=p}^{q}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$
   data split staring at p and ending at q.
3. MapReduce takes all the dsipatched (mapped) jobs and reduce them by:
   $\theta_j := \theta_j - \alpha\frac{1}{m}(\text{temp}_j^{(1)} + \text{temp}_j^{(2)} + \cdots + \text{temp}_j^{(z)})$
   for all $j = 1, 2, \cdots, n$.

Many learning algorithms are MapReduceable if they can be expressed as computing sums of functions over the training set. Linear regression and logistic regression are easily parallelizable. For neural networks, you can compute forward propagation and back propagation on subsets of your data on many machines. Those machines can report their derivatives back to a 'master' server that will combine them.

Multi-core machines
Split the training set and send them to different cores. Some numerical linear algebra libraries can automatically parallelize their linear algebra operations across multiple cores within the machine.