30-Aug-2020: Welcome Video, and What is Machine Learning

- Machine learning: algorithms; supervised, unsupervised, reinforcement, recommender. In this course, also will learn best practices.

31-Aug-2020: Supervised Learning, and Unsupervised Learning

- Supervised learning: right answers are given
- Regression: predicts continuous variable output; Classification: predicts discrete values
- Classification can have $1, \ldots, N, \ldots, \infty$ attributes. E.g. benignness/malignancy based on age, or age and tumor size, etc.
- Unsupervised learning a.k.a. clustering: Right answers aren't given. For example, news that links to different sources for the same topic.
- Cocktail party algorithm: separates two voices in a conversation, with two microphone recordings. Singular value decomposition is key to this algorithm.
- When learning machine learning, use Octave

1-Sep-2020: Model Representation, and Cost Function

- Training set notation: m is number of training examples, x are input examples, and y are the output variables. Together, (x, y) form a training example. Also denoted $(x^{(i)}, y^{(i)})$.
- In a linear regression, $h_\theta(x) = \theta_0 + \theta_1 x \equiv h(x)$.
- Cost function is

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

- Want to minimize J w.r.t. θ_0 and θ_1.

4-Sep-2020: Cost Function, Intuition I&II; Gradient Descent

- Intuition I; Let $\theta_0 = 0$, then $\min_{\theta_1} J(\theta_1)$ is what we want
- Ex: $h_\theta(x) = \theta_1 x$ and let $(x, y) = \{(1,1), (2,2), (3,3)\}$.

$$\begin{aligned} J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &\rightarrow \text{ If } \theta_1 = 0, \ h_\theta(x) \equiv 0 \\ J(0) &= \frac{1}{2 \times 3} (1 + 4 + 9) \\ &= \frac{14}{6} \end{aligned}$$

- $J(\theta_1)$ is parabolic
- We want $\min_\theta J(\theta)$; here, $\theta_1 = 1$ satisfies this criterion
- Intuition II; Let θ_0, θ_1 be free in $J(\theta_0, \theta_1)$ and $h_\theta(x)$.
- $J(\theta_0, \theta_1)$ is a parabloid
- Gradient Descent; Use gradient descent to find (θ_0, θ_1) that minimizes $J(\theta_0, \theta_1)$.
- Differing starting guesses can give different local minima.
- Gradient descent algorithm:

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

- Simultaneously update θ_0, θ_1, α is called the learning rate.
- Ex: $\theta_0 = 1, \theta_2 = 2$ and $\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$.

$$\begin{aligned} \theta_0 &:= \theta_0 + \sqrt{\theta_0 \theta_1} \\ &= 1 + \sqrt{1 \times 2} \\ &= 1 + \sqrt{2} \\ \theta_1 &= \theta_2 + \sqrt{\theta_0 \theta_1} \\ &= 2 + \sqrt{1 \times 2} \quad \text{note here that we used the old value of } \theta_0 \\ &= 2 + \sqrt{2} \end{aligned}$$

5-Sep-2020: Gradient Descent Intuition, Gradient Descent for Linear Regression

- Gradient Descent Intuition: For simplicity, assume $\theta_0 = 0$
- One variable: $\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$; Newton-Raphson
- If α is too small, convergence may be very slow. If too large, it may miss the minimum.
- If θ_1 is already at a local minimum, g.d. leaves θ_1 unchanged since the derivative is zero.
- Gradient Descent for Linear Regression: We need derivatives

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)$$
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \times x^{(i)}$$

- So, gradient descent finds the new θ variables as

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) \times x^{(i)}$$

- This is called "batch gradient descent"; batch implies looking at all the training examples. This is represented by the $\sum_{i=1}^{m}$.
- Quiz Linear Regression with One Variable: 2) $m = \Delta y / \Delta x = (1 - 0.5)/(2 - 1) = 0.5 \implies y = 0.5x + b$; y-intercept is clearly zero since (0,0) is a data point.
- 3) $h_\theta(x)$; $\theta_0 = -1$, $\theta_1 = 2$; $h_\theta(6) = -1 + 2 \times 6 = 11$

9-Sep-2020: Linear Algebra Review

- Matrices and Vectors: Nothing new; in this course, index from 1.
- Addition and Scalar Multiplication: Nothing new
- Matrix Vector Multiplication: Nothing new;
- Ex: Let house sizes be $\{2104, 1416, 1534, 852.\}$. Let the hypothesis be $h_\theta(x) = -40 + 0.25x$.

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 2 & 852 \end{bmatrix} \begin{bmatrix} -40 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -40 \times 1 + 0.25 \times 2104 \\ -40 \times 1 + 0.25 \times 1416 \\ -40 \times 1 + 0.25 \times 1534 \\ -40 \times 1 + 0.25 \times 852 \end{bmatrix} = \begin{bmatrix} h_\theta(2104) \\ h_\theta(1416) \\ h_\theta(1534) \\ h_\theta(852) \end{bmatrix}$$

  This essentially says data matrix × parameters = prediction
- Best to do this with built-in linear algebra function in Octave/Python. You can do it manually in a for-loop, but it'll be really slow.
- Matrix Multiplication: Take the same example. Now we have three hypotheses:

$$h_\theta(x) = -40 + 0.25x$$
$$h_\theta(x) = 200 + 0.1x$$
$$h_\theta(x) = -150 + 0.4x$$

  In matrix form, this becomes

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 2 & 852 \end{bmatrix} \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

- Matrix Multiplication Properties: Not commutative. $AB \neq BA$. But it's associative. $ABC = (AB)C = A(BC)$.
- Identity matrix is I such that $AI = IA = A$. $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ in 2D.

- Inverse of A is A^{-1} such that $AA^{-1} = A^{-1}A = I$.
- Transpose of A is A^T. If $B = A^T$, then $B_{ij} = A_{ji}$.
- Quiz: 4) $u = \begin{bmatrix} 3 \\ -5 \\ 4 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$, then $u^T v = \begin{bmatrix} 3 & -5 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} = -3 + (-10) + 20 = 13$.

10-Sep-2020: Multiple Features

- Introduce other features: e.g. house price not just a function of square footage; Now, house price vs. sq. footage, age, number of bedrooms, etc.
- n is the number of features, $x_j^{(i)}$ represents the value of the j^{th} feature for the i^{th} training example; $x^{(i)}$ is a vector of all the features.
- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$. Let $x_0^{(i)} = 1$. Then, we can write this in matrix form as

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \implies h_\theta(x) = \theta^T x$$

11-Sep-2020: G.D. for Multiple Variables, G.D. in Practice I - Feature Scaling, G.D. in Practice II - Learning Rate, Features and Polynomial Regression, Normal Equation

- Gradient Descent for Multiple Variables: For $n \geq 1$, gradient descent is

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \quad \text{for } j = 0, \ldots, n.$$

- G.D. in Practice I - Feature Scaling: ensure features have similar scales. E.g.: Houses in the data set have 1-5 bedrooms, and are between 0-2000 sq. ft. Scale these features to the order of 1. So, divide bedrooms by 5 so it's 0-1, and divide square footage by 2000, so it's 0-2.
- Feature should be $-1 \leq x_i \leq 1$.
- Mean renormalization; Subtract off the mean, and then scale. E.g. $x_1 = (\text{sq. footage} - 1000)/2000$ and $x_2 = (\text{bedrooms} - 2)/5$. More formally,

$$x_i \rightarrow \frac{x_i - \mu_i}{s_i} \quad \text{(mean renormalization)},$$

  where x_i is the feature, μ_i is the mean value of the i^{th} feature, and s_i is the range, or standard deviation, of the i^{th} feature.
- G.D. in Practice II - Learning Rate: We can plot $J(\theta)$ as a function of iterations, N; it should be a decreasing function.
- If $J(\theta)$ vs N diverges, you need a smaller learning rate, α.
- If $J(\theta)$ vs N falls, rises, falls, rises, etc., then use a smaller α.
- Features and Polynomial Regression: In the housing example, hypothesis could be $h_\theta(x) = \theta_0 + \theta_1 \times \text{length} + \theta_2 \times \text{depth}$. Maybe you think the relevant figure is area = length×depth$\equiv x$. The hypothesis is $h_\theta(x) = \theta_0 + \theta_1 \times x$.
- Polynomial regression; e.g.

$$\theta_0 + \theta_1 x + \theta_2 \ x^2 + \theta_3 x^3$$
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

  where $x_1 = x = \text{area}$, $x_2 = x^2 = \text{area}^2$, $x_3 = x^3 = \text{area}^3$. In polynomial regression, feature scaling becomes very important.
- Don't just have to have integer powers: e.g. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^{1/2}$
- Normal Equation: Instead of using gradient descent to find $\min_\theta J$, use normal equation to do it analytically.
- Intuition; in 1-D, if $J(\theta) = a\theta^2 + b\theta + c$, you can find $dJ/d\theta = 0$ to get the extremum. In N-D, set $\partial_{\theta_j} J = 0$ for $j = 1, \ldots, N$.

- Say you have m training examples, each with n features. Let

$$\begin{aligned} X_{ij} &= x_j^{(i)} \\ Y_i &= y^{(i)} \\ \theta &= \left(X^T X\right)^{-1} X^T Y \end{aligned}$$

- If the training examples are $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$, then

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}, X = \begin{bmatrix} \vec{x}^{(1)\,T} \\ \vdots \\ \vec{x}^{(m)\,T} \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \theta = \left(X^T X\right)^{-1} X^T Y,$$

  where $x_0^{(i)} = 1$.
- With normal equation method, features don't have to be scaled.
- Normal equation method is slow if n is very large; Computing $(X^T X)^{-1}$ is costly. Inverting an $N \times N$ matrix costs $O(N^3)$.

12-Sep-2020: Normal Equation and Non-Invertibility

- What if $X^T X$ is signular? Octave's `pinv` (pseudo-inverse) takes care of that
- Causes; redundancy: e.g. area in ft^2 and in m^2; too few equations: $m < n$, fewer training examples compared to features, i.e. too few equations, too many unknowns.
- Quiz: 1) Midterm exam average $\mu_1 = (89 + 72 + 94 - 69)/4 = 81$; range is $s_1 = 94 - 69 = 25$, thus $x_1^{(3)} \to (x^{(3)} - \mu_1)/s_1 = (94 - 81)/25 = 0.52$
- 3) $X = \begin{bmatrix} x_0^{(1)} & \ldots & x_3^{(1)} \\ & \vdots & \\ x_0^{(14)} & \ldots & x_3^{(14)} \end{bmatrix}$ is 14×4.

13-Sep-2020: Octave Quiz

- Quiz: 1) A is 3×2, B is 2×3. Thus, AB and A+B^T are valid
- 4) u, v are 7×1. Calculate $u \cdot v$. This can be done via $u^T v$. In Octave, this is `sum(v.*w)` and `v'*w`

17-Sep-2020: Classification, Hypothesis Representation, Decision Boundary, Cost Function, Simplified Cost Function and Gradient Descent

- Classification: $y \in \{0, 1\}$ (binary), $y \in \{0, 1, 2, \ldots, N\}$ (multiclass)
- Could fit a linear $h_\theta(x) = \theta^T x$ and classify using a threshold of 0.5. Not good, though. Too sensitive to outliers. Also, $h_\theta(x)$ can be negative.
- Hypothesis Representation:

$$\begin{aligned} h_\theta(x) &= g(\theta^T x); \\ g(z) &= \frac{1}{1 + e^{-z}} \quad \text{sigmoid or logistic function; Fermi-Dirac distribution;} \\ \implies h_\theta(x) &= \frac{1}{1 + \exp\left(-\theta^T x\right)}. \end{aligned}$$

- $h_\theta(x)$ represents the probability that $y = 1$ on an input x. E.g. $\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$. If $g(x) = 70\%$, then there is a 70% chance that the tumor is malignant.
- $h_\theta(x) = P\left(y = 1 \middle| x; \theta\right)$ means "the probability that $y = 1$ given x, parametrized by θ"
- Probabilities sum to 1. $P\left(y = 1 \middle| x; \theta\right) + P\left(y = 0 \middle| x; \theta\right) = 1$.
- Decision Boundary: Can say if $h_\theta(x) \geq 0.5 \implies y = 1$, $h_\theta(x) < 0.5 \implies y = 0$.
- $g(z) \geq 0.5 \implies z \geq 0$. So, $h_\theta(x) = g(\theta^T x) \geq 0.5 \implies \theta^T x \geq 0$. Converse is true for < 0.5.
- Decision Boundary; Say $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$. WLOG, let $\theta_1 = \theta_2 = 1$. On an x_2-x_1 diagram, this parametrizes a straight line. $x_2 = -x_1 + \theta_0$. The decision boundary is the set of points (x_2, x_1) s.t. $h_\theta(x) = 0.5$.

- Example; $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0 \implies h_\theta(x) = g(5 - x_1)$. Decision boundary is implied by $x_1 = 5$. Where is $h \geq 0.5$? $5 - x \geq 0 \implies x_1 \leq 5$. This region corresponds to $y = 1$.
- Non-Linear Decision Boundaries; $g(\theta_0 + \theta_1 x_2 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$.
- Cost Function: Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, $x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$. m training examples, each with n features. $x_0^{(i)} \equiv 1$.
- Recall, for linear regression, $J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$. Define

  $$\text{cost}(h_\theta(x), y) = \frac{1}{2} \left(h_\theta(x) - y\right)^2, \quad \text{for linear regression only}$$

  so $J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h_\theta(x^{(i)}, y^{(i)})$.
- For log. regression, $J(\theta)$ is not convex, i.e. it has many local minima. Need a new cost function.
- For log. regression, $\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$
- If $y = 1$, and $h_\theta(x) = 1$, the cost=0. As $h_\theta(x) \to 0$, cost $\to \infty$. Converse is also true for $y = 0$.
- Simplified Cost Function For Gradient Descent: Write out cost function in one equation

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

  Plug into J.

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

  Comes from "max likelihood estimation." Then find θ via $\min_\theta J(\theta)$. USe θ to make predictions $h_\theta(x) = 1/(1 + \exp(\theta^T x))$.
- Use g.d.: $\theta_j := \theta_j - \alpha \partial_{\theta_j} J(\theta)$. Computing the derivatives, we have

  $$\theta_j := \theta_j - \alpha \sum_{i=1}^m \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)},$$

  which is the same as for linear regression, though h has a different meaning.

2-Oct-2020 Advanced Optimization, Multiclass Classification One-Vs.-All, Problem of Overfitting, Cost Function (Regularization), Reg. Lin. Regr., Reg. Log. Regr.

- Advanced Optimization; nothing new
- Multiclass Classification: One-vs.-all; e.g. weather: sunny, rainy, snowy, etc. $h_\theta^{(i)}(x) = P\left(y = i | x; \theta\right)$ represents the boundary separating class i from the rest.
- Quiz; $h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x)$. Let $\theta_0 = 6$, $\theta_1 = -1$, $\theta_2 = 0$. The argument, z, of g is positive. $z = 0 \implies x = 6$, and $z \geq 0 \implies x \leq 6$. So, for $y = 1$, $x \leq 6$.
- Problem of Overfitting; Underfit means high bias, overfit means high variance. Overfitting fails to generalize to new examples. To fix overfitting you can (i) reduce the number of examples, (ii) regularize to reduce the magnitude of the θ_i.
- Cost Function (Regularization); Add terms to the cost function such as $1000\theta_3^2$. This will force θ_3 down.
- But what if we don't know what features we want to be small? Do

  $$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \underbrace{\lambda \sum_{i=1}^m \theta_j^2}_{\text{Regularization term}} \right],$$

  where we do not penalize θ_0. If λ is too large, it underfits.

- Regularized Linear Regression; G.D. for lin. regr.:

$$\begin{aligned}\theta_j &:= \theta_j - \alpha \left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)} + \frac{\lambda}{m}\theta_j\right] \\ &:= \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)},\end{aligned}$$

  where $1 - \alpha\lambda/m < 1$ which reduces θ_j.
- Normal equation becomes

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

  and as long as $\lambda > 0$, the matrix will not be singular.
- Regularized Log. Regression; Cost function is

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

- G.D. becomes (same cosmetically as for lin. regr.)

$$\theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}.$$

- Use `fminunc` (unc means unconstrained). For this, you need to give derivatives.

$$\begin{aligned}\frac{\partial}{\partial\theta_0}J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)} \\ \frac{\partial}{\partial\theta_j}J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j\end{aligned}$$

6-Oct-2020 Neural Networks: Non-Linear Hypothesis, Neurons and the Brain

- e.g. identifying a car. Come up with a classification problem. Take images of a car and select two pixels. Take their intensity, and form an ordered pair $(I(p_1), I(p_2))$. Do this for cars and non-cars. Cars and non-cars will lie in different regions.
- If images were 50x50 pixels, $n = 2500$. (2500 pixels per image). This would make 3×10^6 quadratic features. So, log regression with quadratic features doesn't work.

7-Oct-2020 Model Representation I and II, Examples and Intutitions I and II

- Model Representation I; A neuron is a logistic unit. Input is $\{x_1, \ldots, x_n\} \to h_\theta(x) = g(\theta^T x)$ where $g(z) = (1 + \exp(-z))^{-1}$. Sigmoid activation function.
- We call x_0 the bias unit, and in neural networks, θ are called the weights. We call the inputs the "input layer", the last layer of neurons the "output layer," and the rest "hidden layers."
- $a_i^{(j)}$ is the activation of unit i in the layer j. $\Theta^{(j)}$ is the matrix of weights controlling the function mapping from layer j to layer $j+1$.
- cf. p. 20 of lecture 08.pdf for a detailed picture.
- If a network has s_j units in layer j and s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ is of dimension $s_{j+1} \times (s_j + 1)$.
- Model Representation II; Define $z_1^{(2)} = \Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3$ such that $a_1^{(s)} = g(z_1^{(2)})$

  Then let $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$, $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$. Then, $z^{(2)} = \Theta^{(1)}x$ and $a^{(2)} = g(z^{(2)})$. (These last two are vectors, and g is applied element-wise. Here $\Theta^{(1)}$ is 3×4)

- cf. p. 23 of lecture 08.pdf. We may also call the input layer $a^{(1)}$. To add a bias for $a^{(3)}$, define $a_0^{(2)} = 1$. Thus $a^{(2)} \in \mathcal{R}^4$.
- This is forward propagation. In the example, on p. 23, imagine hiding the input layer. Then we have $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$ feeding to $a^{(3)}$.

$$\begin{aligned} a^{(3)} &= g(\Theta_{10}^{(2)} x_0 + \Theta_{11}^{(2)} x_1 + \Theta_{12}^{(2)} x_2 + \Theta_{13}^{(2)} x_3 \\ &= g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3) \quad \text{drop some super/subscripts} \\ &= g(\theta^T x) \rightarrow \text{ just like log. regression.} \end{aligned}$$

  The difference comes in that it's not the straight inputs passed to the log regression. It's the "learned" features of the first layer.
- An architecture refers to how the networks are connected.
- Examples and Intutions I; x_1 NOR x_2 = NOT(x_1 XOR x_2). XOR is True if x_1 OR x_2 is True.
- cf. p. 34 of lecture 08.pdf for a n.n. implementation of AND, OR, and (NOT x_1) AND (NOT x_2)
- For a NOT, use a large negative weight on the neurons other than the bias.
- cf. p. 40-42 re: multiclass classification.

8-Oct-2020 Cost Function and Back Propagation, Backpropagation Algorithm

- Cost Function and Back Propagation; L = # of layers, s_l is # of units in layer l, excluding bias.
- Binary classification: $s_L = 1$, $h_\theta(x) \in \mathcal{R}$, $K = 1$ (i.e. a real number). $y = \{0, 1\}$.

  Multiclass classification: $s_L = K$, $h_\theta(x) \in \mathcal{R}^K$, $K \geq 3$, $y \in \mathcal{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, etc.
- For a n.n., $h_\theta(x) \in \mathcal{R}^K$, $[h_\theta(x)]_i = i^{th}$ output. The cost function is

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

- Backpropagation Algorithm; Wish to get $\min_\Theta J(\Theta)$. Need $J(\Theta)$ and $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ for all i, j, l.
- Suppose we have just one training example (x, y). See p.6 of lecture 09.pdf for forward propagation.
- δ_j^l represents the error of the j^{th} unit in the l^{th} layer.

  For the output layer, (cf. p. 7), $\delta_j^{(4)} = a_j^{(4)} - y_j = [h_\theta(x)]_j - y_j$. Vectorize this to have $\delta^{(4)} = a^{(4)} - y$.
- Look at $\delta^{(3)}$.

$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} \underbrace{.*}_{\text{element-wise}} g'\left(z^{(3)}\right),$$

  where g' is the derivative of the activation function. It can be shown that $g'(z^{(3)}) = a^{(3)} .* \left(1 - a^{(3)}\right)$. There is no $\delta^{(1)}$ since we assume there is no error to the input.
- We call this backpropagation since we start at the output layer and get $\delta^{(4)}$, and use it to get $\delta^{(3)}$, etc.
- Without regularization, we have

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

- Now say the training set is $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(,)})\}$. Let $\Delta_{ij}^{(l)} = 0$ for all i, j, l. Pseudocode:
- For i=1 to m:
  - $a^{(1)} = x^{(i)}$
  - Do forwards prop to get $a^{(l)}$ for $l = 2, \ldots, L$
  - Use $y^{(i)}$ to get $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - Use back propagation to get $\delta^{(L-1)}, \ldots, \delta^{(2)}$
  - $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \rightarrow$ vectorize: $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

  Then, calculate

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

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

9-Oct-2020: Backpropagation Intutition, Implementation Note, Gradient Checking

- Backpropagation Intuition; Quiz: Suppose $\Theta_{11}^{(2)} = \Theta_{21}^{(2)} = 0$. After back propagation, what can you say about $\delta_1^{(3)}$? We know that $\delta_1^{(3)} = \Theta_{11}^{(3)}\delta_1^{(4)}$. So, there is insufficient information to tell.
- Implementation Note; Unrolling Parameters. In Octave, say `v` is a matrix. Then `v(:)` makes it a vector.
- Gradient Checking; Use symmetric difference. In 1-D

$$\frac{d}{d\theta}J(\theta) = \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}.$$

In N-D,

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{J(\theta_1,\ldots,\theta_j+\epsilon,\ldots,\theta_n) - J(\theta_1,\ldots,\theta_j-\epsilon,\ldots,\theta_n)}{2\epsilon}$$

- Use this to compare to derivs we get from back-propagation. Don't use it for training! It will be very slow.

13-Oct-2020 Random Initialization, Putting it Together, Autonomous Driving

- Random Initialization; Have to give a guess for $\Theta^{(l)}$; can't just guess all 0s. If the initial Θ are all zero, it means all units are looking at the same features. Initialize $\Theta_{ij}^{(l)}$ to a random $[-\epsilon, \epsilon]$.
- Putting it Together; Pick some network architecture. Input layer will have a size of $x^{(1)}$. Output layer is the number of classes. As a default, use one hidden layer. If > 1 hidden layer, then use the same number of units in each hidden layer.
- To train a neural network:
  1. Randomly initialize the weights
  2. Do forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$.
  3. Implement code to compute $J(\Theta)$.
  4. Implement back propagation algorithm to get the partial derivatives.
     The backpropagation algorithm will loop over the training examples. Get $a^{(l)}$ and $\delta^{(l)}$ for $l = 1, \ldots, L$.
  5. Use gradient checking to compare partial derivatives from back propagation. Then discard gradient checking.
  6. Use optimization algorithm to minimize $J(\Theta)$.
- Note that $J(\Theta)$ is non-convex. So, the algorithm can get stuck in a local minimum, but this is rare in practice.
- Autonomous Driving; nothing new.

15-Oct-2020 Deciding What to Try Next, Evaluating Hypotheses, Model Selection and Train/Validation/Test Sets, Diagnosing Bias vs. Variance, Regularization and Bias/Variance

- Deciding What To Try Next; Suppose, for e.g., you've implemented regularized linear regression, and it makes predictions with large errors. Try (i) get more training e.g.s, (ii) reduce the number of features, (iii) try more features, (iv) add polynomial features, (v) increase/decrease λ.
- Simple way to rule out solutions: diagnostics. It can take a while to implement, but can ultimately pay off.
- Evaluating a Hypothesis; Split the data into a training/test set. Split the data 70/30 randomly.
- For lin. regr., analagous to the training set, the test error is $J_{test}(\theta) = 1/(2m_{test})\sum_{i=1}^{m}\left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2$.
- Analagous J_{test} for log. regr. For log. regr., also have misclassification error.

$$\text{err}(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5 \text{ and } y = 0 \text{ OR } \text{ if } h_\theta(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases}$$

Could then define the test error to be

$$J_{test}(\theta) = \frac{1}{m_{test}}\sum_{i=1}^{m_{test}} \text{err}\left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)}\right) \quad \text{misclassification error}$$

- Model Selection and Train/Validation/Test Sets; How to choose the polynomial degree d?

$$\begin{aligned}
&d=1 && h_\theta(x)=\theta_0+\theta_1 x\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(1)}\rightarrow J_{test}(\theta^{(1)})\\
&d=2 && h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(2)}\rightarrow J_{test}(\theta^{(2)})\\
&\vdots\\
&d=n && h_\theta(x)=\textstyle\sum_{k=0}^{n}\theta_k x^k\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(n)}\rightarrow J_{test}(\theta^{(n)})
\end{aligned}$$

  A problem can be that we're essentially fitting d to the training set. E.g. we find that $J_{test}(\theta^{(5)})$ is the lowest, then we've fit $d=5$ to the training set.
- Split the data into a training, cross-validation (CV), and test set (60-20-20). Get the training, CV, and test errors. Repeat the process.

$$\begin{aligned}
&d=1 && h_\theta(x)=\theta_0+\theta_1 x\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(1)}\rightarrow J_{cv}(\theta^{(1)})\\
&d=2 && h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(2)}\rightarrow J_{cv}(\theta^{(2)})\\
&\vdots\\
&d=n && h_\theta(x)=\textstyle\sum_{k=0}^{n}\theta_k x^k\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(n)}\rightarrow J_{cv}(\theta^{(n)})
\end{aligned}$$

  Pick a d such that J_{cv} is the smallest, e.g. $\theta^{(4)}$, and then get $J_{test}(\theta^{(4)})$, and assess it. So, use cross-validation to select the model (here, polynomial degree).
- Diagnosing Bias (Underfit) vs. Variance (overfit); cf. lecture 10, slide 17 for a plot of J_{cv}, J_{test} vs d.
- Bias (underfit): $J_{train}(\theta)$ is high; $J_{train}\approx J_{cv}$.
  Variance (overfit): $J_{train}(\theta)$ is low; $J_{train}\ll J_{cv}$.

- Regularization and Bias/Variance; For linear regression with regularization,

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

  Suppose we're using a fourth-order polynomial, so $h_\theta(x)=\sum_{k=0}^{4}\theta_k x^k$.
  For very large λ, $\theta_1,\ldots,\theta_4\approx 0$ (heavily penalized) → underfit → high bias.
  For very small λ, $\theta_1,\ldots,\theta_4\neq 0$ (very little regularization) $rightarrow$ overfit → high variance.
  We need a goldilocks λ.
- With regularization, change definitions a bit.

$$\begin{aligned}
J_{train}(\theta)&=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)^2 \quad \text{(NO regularization terms)}\\
J_{cv}(\theta)&=\frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta(x_{cv}^{(i)})-y_{cv}^{(i)}\right)^2\\
J_{test}(\theta)&=\frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta(x_{test}^{(i)})-y_{test}^{(i)}\right)^2
\end{aligned}$$

- So, how to select λ? Get θ from $\min_\theta J(\theta)$ with the regularization.

$$\begin{aligned}
&1) && \lambda=0\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(1)}\rightarrow J_{cv}(\theta^{(1)})\\
&2) && \lambda=0.02\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(2)}\rightarrow J_{cv}(\theta^{(2)})\\
&\vdots\\
&n) && \lambda=10\rightarrow && \min_\theta J(\theta)\rightarrow \theta^{(n)}\rightarrow J_{cv}(\theta^{(n)})
\end{aligned}$$

  Pick a model that gives the lowest sum of squares J_{cv}, e.g. $\theta^{(5)}$. In other words, we fit λ to the c.v. set. Then, see how well $J_{test}(\theta^{(5)})$ is.
- See lecture 10, slide 23 for a plot of J_{train}, J_{cv} vs. λ.

16-Oct-2020 Learning Curves, Deciding What To Do Next

- Learning Curves; Plot J_{cv}/J_{train} as a function of number of training examples, m.ARtificially reduce the training set to m examples, for a fixed polynomial degree $h_\theta(x)$, e.g. $h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2$.

- High bias: See lec. 10, p.26: $h_\theta(x) = \theta_0 + \theta_1 x$ doesn't fit the data well. More data won't help the straight line fit any better. Adding training examples only improes J_{cv} up to a certain point. We find J_{train} approaches J_{cv}.
  $\implies$ if a learning algorithm has high bias, more training e.g.s won't help!
- High variance: see lec.10, p.27: e.g. $h_\theta(x) = \sum_{k=0}^{100} \theta_k x^k$. For small m, it'll fit the data very well (overfitting). So J_{train} is small for small m. But as we include more training examples (larger than the polynomial degree), $h_\theta(x)$ doesn't generalize well, and J_{train} increases. J_{cv} starts out high (with a few training examples, $h_\theta(x)$ generalizes very poorly. With more training e.g.s, J_{cv} decreases.
  $\implies$ if a learning algorithm has high variance, more training e.g.s will help!
- Deciding What To Do Next Revisited; Recall that we're fitting house prices, but the hypothesis made bad predictions.
  1. More training e.g.s → fixes high variance
  2. Smaller feature set → fixes high variance
  3. More features → fixes high bias
  4. Add polynomial features → fixes high bias
  5. Decrease λ → fixes high bias
  6. Increase λ → fixes high varaince
- Neural Networks: Small networks (few units, few hidden layers) are prone to underfitting, but computationally cheaper
  Large networks (more units, more hidden layers) are prone to overfitting, and computationally more expensive.
  You can use regularization to address overfitting.
- How many hidden layers? A sensible default is one hidden layer. But you can use more layers and check J_{cv} and J_{test}.
- Quiz: n.n. with one hidden layer and $J_{cv}(\theta) \gg J_{train}(\theta)$. Will increasing the number of hidden units help? No. Since there is a gap between cross validation and training sets, it indicates high variance. So it's overfitting the data, and adding more layers/units won't help.

20-Oct-2020 Prioritizing What To Work On, Error Analysis, Error Metrics for Skewed Classes, Trading Off Precision/Recall, Data for Machine Learning, Large Margin Classification: Optimization Objective, Large Margin Intuition

- Prioritizing What to Work On; Building a spam classifier. Let $y = 0 \implies$ not spam, and $y = 1 \implies$ spam. Features? Choose 100 words, e.g. $\underbrace{\text{deal, buy, discount,}}_{\text{spam words}}$ $\underbrace{\text{andrew, now, home}}_{\text{non-spam words}}$. Then, for an e-mail, build a feature vector

$$\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_{100} \end{array} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \begin{array}{l} \to\text{deal} \\ \to\text{buy} \\ \vdots \quad \vdots \\ \to\text{now} \end{array}$$

  where

$$x_j = \begin{cases} 1 & \text{if the word } j \text{ is in the e-mail} \\ 0 & \text{otherwise} \end{cases}$$

- How to get low error? More data, possibly. Could use more sophisticated features based on the header (e.g. routing info). Could also use more sophisticated features based on words (e.g. deal vs. Dealer; are these the same words?). Could also use more features based on punctuation. Could introduce features based on misspelling.
- Error Analysis; Look at examples the algorithm gets wrong. Look at the features. What additional features would help improve the accuracy?
- Numerical error: Have a single real number as a metric for how good the algorithm is. One example is J_{cv}.
- See "porter stemming" for natural language processing (NLP).
- *Always* start with a quick and dirty implementation of an algorithm. Then check learning curves, and then analyze features.
- Error Metrics for Skewed Classes; Consider classifying benign/malignant ($y = 0/y = 1$) tumors where 99.5% of the cases are benign. Say we've trained a logistic regression, and we get 99.0% accuracy. But, suppose our algorithm just always retuend $y = 0$ regardless of any features x. Then, we'd get 0.5% errror! The

reason is that this is a skewed class; there is much more of one class than the other. For skewed classes, just classification error isn't a good metric.

- Precision/Recall:

Actual Predicted	1	0
1	True Positive	False Positive
0	False Negative	True Negative

$$\begin{aligned}
\text{Precision} &= \frac{\text{True Positives}}{\text{Predicted Positives}} \\
&= \frac{\text{True Positives}}{\text{True Positives + False Positives}} \quad \text{denom is sum of first row in table} \\
\text{Recall} &= \frac{\text{True Positives}}{\text{Actual Positives}} \\
&= \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \quad \text{denom is sum of first column in table}
\end{aligned}$$

- So, the classifier which just sets $y = 0$ has a recall of zero, since there are no true positives! Having high precision and recall are good metrics.
- Let $y = 1$ be the rarer class when doing precision/recall.
- Trading Off Preicsion/Recall; Continue with the cancer example. Suppose we want to report $y = 1$ only if we're very positive. We could increase the threshold: $h_\theta(x) \geq 0.8 \implies$ higher precision (reduces the false positives) but gives lower recall.
- Suppose we want to avoid missing malignant cases. Reduce the threshold e.g. $h_\theta(x) \geq 0.4 \implies$ lower precision and higher recall.
- See lecture 11, p. 13 to see the tradeoff graph between Precision/Recall. (Very loosely, $P = 1 - R$.)
- How do we choose the threshold? The problem is that P and R are two numbers. Suppose you have two algorithms. Algorithm 1 gives $P = 0.2, R = 0.7$ and algorithm 2 gives $P = 0.8, R = 0.1$. Which one is better? It's better to have a single real number.
- F_1 Score:
$$F_1 = \frac{2PR}{P + R}$$
- Data for Machine Learning; For some algorithms, more data can really improve accuracy.
- Large data rationale: Assume that features $x \in \mathcal{R}^{n+1}$ can predict y.
  e.g. "I ate ____________ eggs for breakfast." Should the blank be too/two/to? The features {ate, eggs, breakfast} are enough to tell us.
  e.g. Just the square footage of a house; this is not enough to predict a house price. Need more info such as location, bedrooms, etc.
  A litmus test is "can a human predict y given x?"
- Now a learning algorithm with many parameters (low bias) will fit a training set well. With a very, very large training set, unlikely to overfit. The low bias $\implies J_{train} \approx 0$. "Unlikely to overfit" $\implies J_{train} \approx J_{cv} \approx 0$.
- Large Margin Classification - Optimization Objective; Recall with logistic regreesion
$$\begin{aligned}
h_\theta(x) &= \frac{1}{1 + \exp\left(-\theta^T x\right)} \\
z &= \theta^T x \\
g(z) &= \left(1 + e^{-z}\right)^{-1}.
\end{aligned}$$
  We know if $z \gg 0 \implies g(z) \approx 1$ and that $z \ll 0 \implies g(z) \approx 0$. Cost (mod a factor of $1/m$):
$$\text{cost} = -y \log h_\theta(x) - (1 - y) \log(1 - h_\theta(x))$$
- See lecture 12, p. 3 for plots of the log(...). Then, come up with a piecewise linear approximation of these functions. Call them cost_1 and cost_0, corresponding to the $y = 1, 0$ terms.
- For logistic regression, our objective was
$$\min_\theta \frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \left(-\log h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \left(-\log \left(1 - h_\theta(x^{(i)}) \right) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

- For support vector machines (SVM), drop $1/m$ by convention

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

- We parameterize log. regression as $\underbrace{A}_{\text{cost}} + \lambda \underbrace{B}_{\text{regression}}$. For SVM, parameterize as $CA + B$ where C is the regularization parameter. All regularization does is control the relative size of the different terms. Roughly speaking, $C = 1/\lambda$. So in SVM, we're doing

$$\min_\theta C \sum_{i=1}^{m} \left(y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right) + \frac{\lambda}{2} \sum_{j=1}^{m} \theta_j^2$$

- SVM's hypothesis

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

- Large Margin Intuition; See lecture 12, slide 7 for a summary of SVM cost function and hypothesis.
- Suppose we pick a huge C. Then we'll want $\sum_{i=1}^{m} \left(y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right) = 0$.
  So whenever $y^{(i)} = 1$, need $\theta^T x^{(i)} \geq 1$ (changed this threshold from 0 to 1 to make the algorithm more "sure"). Conversely, whenever $y^{(i)} = 0$, need $\theta^T x^{(i)} \leq -1$. Really, go look at the slide.
- SVM makes decision boundaries that give the largest margin between the classes, hence Large Margin Classifier.

21-Oct-2020 Large Margin Intuition cont'd, Math. behind LMC, Kernel I

- Large Margin Intuition (Continued); Outliers: when C is not too large, SVM can ignore some outliers.
- Mathematics Behind Large Margin Classifiers; For a very large C, we are trying to minimize $\sum_{j=1}^{n} \theta_j^2$, such that $\theta^T x^{(i)} \geq 1$ for $y = 1$ and $\theta^T x^{(i)} \leq -1$ for $y = 0$.
- Say $n = 2$. Then, our minimization objective is

$$\frac{1}{2} \left(\theta_1^2 + \theta_2^2 \right) = \frac{1}{2} |\theta|^2 ,$$

  where, WLOG, $\theta_0 = 0$ and θ is a vector.
- See lecture 12, p. 13. We can write $\theta^T x^{(i)} = p^{(i)} |\theta|$, where $p^{(i)}$ is the projection of $x^{(i)}$ onto θ. The optimization objective is now

$$\frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \quad \text{if} \quad p^{(i)} |\theta| \begin{cases} \geq 1 & \text{if } y^{(i)} = 1 \\ \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$$

- θ is orthogonal to the decision boundary. Algorithm chooses decision boundary such that $p^{(i)}$ is larger and so that $|\theta|$ is smaller. (Based on the requirement that $p^{(i)} |\theta| \geq / \leq -1/+1$.
- We assumed $\theta_0 = 0$. If we allow $\theta_0 \neq 0$, we allow boundaries that don't pass through the origin.
- Kernels I; We saw complex decision boundaries based on polynomal features. Let's redefine our features. In x-space, define some landmarks $l^{(1)}, l^{(2)}, l^{(3)}$. Make new features given x.

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

- We call the similarity the "kernel." Gaussian kernel here. Write $k(x, l^{(i)})$. The new features die off exponentially the further they are from the landmarks.

22-Oct-2020: Kernels II, Using an SVM

- Where do we get the landmarks? Put the landmarks at the training examples. Given $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$, let $l^{(k)} = x^{(k)}$.
- Given a training example $x^{(i)}$

$$\begin{aligned} x^{(i)} \to & f_1^{(i)} &= \text{sim}(x^{(i)}, l^{(1)}) \\ & f_2^{(i)} &= \text{sim}(x^{(i)}, l^{(2)}) \\ & \vdots & \\ & f_i^{(i)} &= \text{sim}(x^{(i)}, l^{(i)}) = 1 \\ & \vdots & \\ & f_m^{(i)} &= \text{sim}(x^{(i)}, l^{(m)}) \end{aligned}$$

  (Hold the training e.g. fixed, and get the similarity with every landmark. We've taken an $n(+1)$ dimensional vector $(x^{(i)})$ and turned it into an m-dimensional one $(f^{(i)})$. Note that now with these new $f^{(i)}$, θ will be $m+1$ dimensional.)
- So $x^{(i)} \in \mathcal{R}^{n+1}$ gets mapped to $\begin{bmatrix} f_0^{(i)} \\ \vdots \\ f_n^{(i)} \end{bmatrix}$.
- Hypothesis: Given x, compute $f \in \mathcal{R}^{m+1}$
  Predict $y = 1$ if $\theta^T f \geq 1$.
  Train:

$$\min_\theta \sum_{i=1}^{m} \left(y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) \right) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2,$$

  and this will give θ.
- How to choose C?
  Large $C \implies$ high variance
  Small $C \implies$ low variance
  Large $\sigma^2 \implies$ High Bias (Low Variance) Small $\sigma^2 \implies$ High Variance
- Using an SVM; Use `liblinear` or `libsvm`. We need to chose C and the kernel. (No kernel is the same as linear kernel).
- No kernel means that similarity$(x, l) = 1$. This does $y = 1$ if $\theta^T x \geq 0$. Use this if we have a large number of features and few training examples.
- Gaussian kernel means we need to choose σ^2 too. Use this kernel if number of features and/or number of training examples is large. Perform feature scaling first!
- Not all kernels are valid. They must satisfy Mercer's theorem. Use linear or Gaussian kernels. Another kernel is a polynomial kernel $k(x, l) = (x^T l + \text{const})^d$. Intuition: if x and l are similar, $x^T l$ will be large.
- Multiclass: most pacakges have multiclass built in. Otherwise, use one-vs.-all.
- Logistic Regression vs. SVM:
  - If $n \geq m$, use log. regr. or use SVM without a kernel.
  - If n is small, and m is intermediate, use SVM with a Gaussian kernel.
  - If n is small, and m is huge, then (i) create or add features and/or (ii) use SVM without a kernel or (iii) use logistic regression.
- SVMs have a convex cost function, so they will always find the global minimum.

26-Oct-2020: Clustering, K-Means Algorithm

- Unsupervised Learning: You have a training set $\{x^{(1)}, \dots, x^{(m)}\}$ but no y's. The algorithm will "find paterns in the data." This is clustering.
- K-Means Algorithm; This is the most common clustering algorithm. Cf. lecture 13, pp. 6-14.
  - Randomly initialize two centroids. Call them 1 and 2.
  - Go through all the points and find out if they're closer to centroid 1 or centroid 2.
  - For all the points that were labelled as centroid 1, find the "centre of mass." Do the same thing for centroid 2.
  - Replace the cluster centroids at the centres-of-mass.
  - Repeat the previous four steps util we're done. We're done when the centroids don't move anymore.

- The input to this algorithm is K, the number of clusters, the training set $\{x^{(1)}, \ldots, x^{(m)}\}$, $x^{(i)} \in \mathcal{R}^n$ (drop $x_0 = 1$). Randomly assign the centroids $\mu_1, \ldots, \mu_K$.

```
for i=1:m
  – $c(i) :=$ index of centroid that is closest to $x^{(i)}$, i.e. $c(i) = \min_k \left|x^{(i)} - \mu_k\right|^2$.
for k=1:K
  – $\mu_k :=$ the average of points assigned to cluster $k$.
    (e.g. if $c^{(1)} = 2, c^{(5)} = 2, c^{(6)} = 2$, then $\mu_k = (x^{(1)} + x^{(5)} + x^{(6)})/3$.
    (Note: if there are no points assigned to one centroid, eliminate it [most common], or re-randomize it.)
```

- What if the points are not well separated? E.g. suppose you have (weight,height) pairs, and you're trying to figure out for which (w, h) should correspond S,M,L t-shirt sizes. Apparently, K-means will divide up the data and figure out what (w, h) correspond to S,M,L. (Unclear about this point.)

27-Oct-2020: Optimization Objective, Random Initialization, Choosing the Number of Clusters, Motivation I: Data Compression, Motivation II: Data Visualization

- Optimization Objective; Introduce $\mu_{c^{(i)}}$ = cluster centroid *of cluster* to which $x^{(i)}$ has been assigned. e.g. $x^{(i)}$ belongs to cluster 5. Then $c^{(i)} = 5$ and $\mu_{c^{(i)}} = \mu_5$.
- The cost function

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} \left|x^{(i)} - \mu_{c^{(i)}}\right|^2,$$

  and the optimization objective

$$\min_{\substack{c^{(1)}, \ldots, c^{(m)} \\ \mu_1, \ldots, \mu_k}} J\left(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k\right).$$

- In our algorithm the first step ($c^{(i)}$ = index of cluster centroid closest to $x^{(i)}$, i.e. cluster assignment step) is minimizing J w.r.t. $c^{(1)}, \ldots, c^{(m)}$ while holding k fixed.
  The "move centroid step" ($\mu_k :=$ mean of points assigned to cluster k) minimizes J w.r.t. $\mu_1, \ldots, \mu_k$.
- Random Initialization; Let $K < m$. Randomly assign the centroids to K training examples.
- The optimization can run into local optima if initialization is unlucky (e.g. if two centroids get randomly picked within the same cluster.) To avoid this, use many random initializations and run K-means many times.
- Run K-means many times (e.g. 50-1000). For each iteration, you'll get a $J\left(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k\right)$. Choose the iteration that gives the lowest cost.
- If you have few clusters e.g. K=2-10, then multiple random initiations can be quite helpful. This is less helpful for large $K \sim 10^2$ to 10^3.
- Choosing the Number of Clusters; Elbow: plot J vs. k. If there is a clear "kink" or elbow, choose the number of clusters at the kink. Though, this doesn't always happen.
- Can also consider the purpose: e.g. S,M,L or XS,S,M,L,XL t-shirts. Three or five clusters.
- Motivation I: Data Compression; Highly redundant features, e.g. x_1 = length in inches, x_2 = length in cm. On an x_2-x_1 diagram, they lie on a straight line. $x^{(i)} \in \mathcal{R}^2 \to z^{(i)} \in \mathcal{R}$.
- Motivation II: Data Visualization; E.g. Countries have GDP, population, HDI, etc. Maybe $x^{(i)} \in \mathcal{R}^{100}$. If we can reduce this to two features, then we can plot it as (z_1, z_2).
- Note that when we reduce dimensionality, the algorithm doesn't ascribe any meaning to the new features; that's up to us.

28-Oct-2020: Principal Component Analysis, PCA Algorithm, Reconstruction from Compressed Representation

- Principal Component Analysis; Project the data onto a lower dimensional surface such that the "projection error" (i.e. sum of squares of the distance between the point and the surface) is minimal. E.g. For 2-D data, find a line (the lower dim. surface) s.t. the sum of squares of the distance between the point and its projection onto the line is minimal.) See lecture 14, p.11.
- PCA problem:
  To reduce 2-D to 1-D: find a direction $u^{(1)} \in \mathcal{R}^2$ onto which to project the data so as to minimize the projection errors.
  To reduce n-D to k-D: find k vectors $u^{(1)}, \ldots, u^{(k)} \in \mathcal{R}^n$ onto which to project the data so as to minimize the projection errors.

- PCA is not linear regression; linear regression minimizes a different square distance rror. In PCA, we have no special variable y.
- PCA Algorithm; Pre-process the data with feature scaling!!!
- PCA needs to (i) calculate the $u^{(j)}$ for $j = 1, \dots, k$ vectors and (ii) calculate $z^{(i)}$ from $x^{(i)}$.
- Algorithm: reduce data from n-D to k-D.
  Compute covariance matrix: $\Sigma = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \left(x^{(i)}\right)^T$
  Compute eigenvectors: `[U,S,V] = svd(Σ)`. `U` is $n \times n$ with each of the column representing an eigenvector
  Take the first k columns (i.e. the first k eigenvectors) to get $U_{red} = \begin{bmatrix} | & & | \\ u^{(1)} & \dots & u^{(k)} \\ | & & | \end{bmatrix}$
  Then, to get the k-dimensional feature z, go $z = U_{red}^T x$.
- This takes $x \in \mathcal{R}^n \to z \in \mathcal{R}^k$.
- Reconstruction from Compressed Representation; $x_{approx} = U_{red} z$.

29-Oct-2020: Choosing the Number of Principal Components, Advice for Applying PCA

- k is the number of principal components
  Average squared projection error: $\frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} - x_{approx}^{(i)} \right|^2$
  Total variance of data: $\frac{1}{m} \sum_{i=1}^{m} \left| x^{(i)} \right|^2$.
- Choose k to be the smallest value such that

$$\frac{\sum_{i=1}^{m} \left| x^{(i)} - x_{approx}^{(i)} \right|^2}{\sum_{i=1}^{m} \left| x^{(i)} \right|^2} \leq 0.01,$$

  i.e. 99% of the variance is retained.
- One algorithm (don't use this) would be as follows:
  - Try PCA with $k = 1$
  - Computer $U_{red}, z^{(1)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$.
  - Check if $\frac{\sum_{i=1}^{m} \left| x^{(i)} - x_{approx}^{(i)} \right|^2}{\sum_{i=1}^{m} \left| x^{(i)} \right|^2} \leq 0.01$.
  - If it's not, try $k = 2$, and so on.
- However, this is very inefficient. There is a better way. We did `[U,S,V] = svd(Sigma)`. The S matrix (covariance matrix) is $\begin{bmatrix} s_{11} & & \\ & \ddots & \\ & & s_{nn} \end{bmatrix}$. It turns out that:

$$\frac{\sum_{i=1}^{m} \left| x^{(i)} - x_{approx}^{(i)} \right|^2}{\sum_{i=1}^{m} \left| x^{(i)} \right|^2} = \frac{1 - \sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}.$$

- So, we just need to look at the values of S, and keep summing up the diagonal elements until we retain 99% of the variance.
- Advice for Applying PCA; PCA can be used to speed up supervised learning problems. Train on compressed data.
- Training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$. Say that $x^{(i)} \in \mathcal{R}^{10{,}000}$.
  Now, $x^{(1)}, \dots, x^{(m)} \xrightarrow[\text{PCA}]{} z^{(1)}, \dots, z^{(m)} \in \mathcal{R}^{1{,}000}$.
  This gives a new training set $\{(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})\}$
  Now use this training set to train the learning algorithm.
- PCA gives a mapping from $x \mapsto z$. The mapping should be defined only by running PCA on the training set. *Apply* the mapping to x_{cv} and x_{test}.
- For visualization, choose $k = 2$ or $k = 3$.
- Might think PCA can be used to avoid overfitting. In other words, reduce the number of features from $n = 10,000$ to $k = 1,000$. You might think that fewer features means it's less likely to overfit. This is bad! Just use regularization.
- Don't automatically jump to PCA; try training learning algorithms on raw data. Only if the algorithm runs too slow, or you run out of disk space, then apply PCA.

3-Nov-2020: Problem Motivation, Density Estimation, Gaussian Distribution, Algorithm, Developing and Evaluating an Anomaly Detection System, Anomaly Detection vs. Supervised Learning, Choosing What Features to Use, Multivariate Gaussian Distribution

- Problem Motivation - Density Estimation; Suppose you're building aircraft engines. You test two features x_1 = heat generated, and x_2 = vibration intensity. You have this data for m examples. $\{x^{(1)}, \ldots, x^{(m)}\}$. Now, you have a new engine, with features x_{test}. How do you know if it's anomalous? Based on the training set, construct a model $p(x)$. If $p(x_{test}) < \epsilon$, then you can flag it as anomalous. If $p(x_{test}) \geq \epsilon$ means it's ok.
- Gaussian Distribution; Nothing new. $\mu = \frac{1}{m}\sum_i x^{(i)}$, $\sigma^2 = \frac{1}{m}\sum_i (x^{(i)} - \mu)^2, x^{(i)} \sim \mathcal{N}(x^{(i)}; \mu, \sigma^2)$.
- Algorithm;
  $p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$ where $p(x_j; \mu_j, \sigma_j^2) \sim \mathcal{N}(x_j; \mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(\frac{-(x_j - \mu_j)^2}{2\sigma_j^2}\right)$.
  1. Choose features x_i that may indicate anomalies.
  2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1, \ldots, \sigma_n$
  3. Given a new example, x, compute $p(x)$

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

     Anomaly if $p(x) < \epsilon$.
- Developing and Evaluating an Anomaly Detection System; It's best to have a single real numbet as a metric to assess our anomaly detection algorithm.
- Assume we have some labelled data $\{x^{(1)}, \ldots, x^{(m)}\}$, so we know if $x^{(i)}$ is anomalous or not. Split the data
  - Training Set: 6-% of the data, almost all non-anomalous.
  - Cross Validation: 2-% of the non-anomalous data, 50% of the anomalous data.
  - Test: 2-% of the non-anomalous data, 50% of the anomalous data.
- Training: $\{x^{(1)}, \ldots, x^{(m)}\}$
  Cross-Validation: $\{(x_{cv}^{(1)}, y^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y^{(m_{cv})})\}$
  Test : $\{(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})\}$
- Fit the model $p(x)$ on the training set. On the cross validation/test set, predict $y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomalous)} \\ 0 & \text{if } p(x) \geq \epsilon \text{ (ok)} \end{cases}$
  This is a *skewed class*! Classification rate is not a good metric. Consider true positives, false positives, false negatives, true negatives. Use precision/recall or F_1.
  Use the cross validation set to fit ϵ.
- Anomaly Detection vs. Supervised Learning; cf. lecture 15, p. 18.
- Choosing What Features To Use;
  Transform: Suppose you have a feature, and its histogram is non-Gaussian. Transform it to try to make it more Gaussian. Take the logarithm, raise it to different powers (e.g. 0.5, 0.1, 0.03, etc.) to make it Gaussian.
  Error Analysis: Most common problem is that $p(x)$ is comparable for anomalous and non-anomalous features. Look at the anomalous examples. Can you come up with another feature that will differentiate the anomalous feature from the non-anomalous one? Cf. lecture 15, p.23.
- Multivariate Gaussian; Let $\mu \in \mathcal{R}^n$, $\Sigma \in \mathcal{R}^{n \times n}$.

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

  where $|\Sigma|$ is the determinant of Σ.
- Σ controls the width/orientation. Cf. lecture 15, pp.27-32.