Overview

- Fitting nonlinear curves: choosing features
- Gradient Descent in Practice
- Closed-form Solution
- Regularisation and Model Selection
Fitting nonlinear curves: choosing features

Example:

• Hypothesis: \( h_\theta(x) = \theta_0 + \theta_1 x^2 \)
• Define feature vector \( \vec{z} \) with \( z = x^2 \). This new \( z \) can be computed given input \( x \), so it's known.
• Using this new feature vector our hypothesis can be rewritten as \( h_\theta(z) = \theta_0 + \theta_1 z \), so can directly apply all the ideas we've just discussed.
Gradient Descent in Practice: Feature Scaling & Mean Normalisation

For better numerical behaviour:

- Try to make sure that features are on a similar scale, ideally every feature lies approximately in range $-1 \leq x_j \leq 1$.

- Replace $x_j$ with $x_j - \mu_j$ (where mean $\mu_j = \frac{1}{m} \sum_{i=1}^{n} x_j^{(i)}$) to make features have approximately zero mean (do not apply to $x_0 = 1$ though).

e.g.

- In advertising example TV budget values lie in range 0.7 to 296.4 with mean 147, so rescaling as $\frac{TV - 147}{296}$ gives a feature with values in interval $-0.5 \leq x_1 \leq 0.5$.

- In general can use $x_1 := \frac{x_1 - \mu_1}{\max(x_1) - \min(x_1)}$ or $x_1 := \frac{x_1 - \mu_1}{\sigma_1}$ (where standard deviation $\sigma_j = \sqrt{\frac{1}{m} \sum_{i=1}^{n} (x_j^{(i)} - \mu_j)^2}$).
Gradient Descent in Practice: Learning Rate

\[ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta). \]

- “Debugging”: How to make sure gradient descent is working correctly.
- How to choose learning rate \( \alpha \).

- \( J(\theta) \) should decrease after every iteration
- Example stopping criterion: stop when decreases by less than \( 10^{-3} \)
Gradient Descent in Practice: Learning Rate

- Use smaller $\alpha$
- But if $\alpha$ too small then can be slow to converge
- E.g. to choose $\alpha$ try 0.001, 0.005, 0.01, 0.05, 0.1
Gradient Descent in Practice: Learning Rate

There are also many automated approaches for adjusting $\alpha$ at each iteration. E.g. using line search:

- Repeat {
  
  Choose descent direction, e.g.
  for $j=0$ to $n$ \{ $\delta_j := \frac{2}{m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$ \}
  
  Select $\alpha$ that makes $J(\theta + \alpha \delta)$ smallest
  $\theta := \theta - \alpha \delta$

} 

But these involve greater computation cost than using a fixed $\alpha$
Gradient Descent in Practice: Scalability

**Batch Gradient Descent:**
- Repeat {
  - for $j=0$ to $n$ {
    - temp$j$ := $\theta_j - \frac{2\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
  }
  - for $j=0$ to $n$ { $\theta_j$ := temp$j$ }
}

at each iteration (i) uses all $m$ training data, (ii) updates all $n + 1$ parameters. Common alternatives:

**Co-ordinate Gradient Descent:**
- $j = 0$
- Repeat {
  - j:= (j+1)mod (n+1)
  - $\theta_j$ := $\theta_j - \frac{2\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
}

at each iteration only update a single parameter.

**Stochastic Gradient Descent:**
- Repeat {
  - for $i=1$ to $m$ {
    - for $j=0$ to $n$ {
      - temp$j$ := $\theta_j - \frac{2\alpha}{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
    }
    - for $j=0$ to $n$ { $\theta_j$ := temp$j$ }
  }
}

repeatedly runs through training set, each time updating the parameters with respect to a single training example.
Closed-form solution

We can find the $\theta$ that minimises $J(\theta)$ in closed form.

- Example: suppose we can one feature $x_1$ and one parameter $\theta_1$
- Goal is to select $\theta_1$ to minimise $J(\theta_1) = \sum_{i=1}^{m}(\theta_1 x_1^{(i)} - y^{(i)})^2$
- Compute derivative with respect to $\theta_1$:

$$\frac{dJ}{d\theta_1} = \frac{1}{m} \sum_{i=1}^{m}(\theta_1 x_1^{(i)} - y^{(i)}))x_1^{(i)} = \frac{1}{m} \left( \theta_1 \sum_{i=1}^{m}(x_1^{(i)})^2 - \sum_{i=1}^{m}y^{(i)}x_1^{(i)} \right)$$

- Set derivative equal to 0 and solve for $\theta_1$:

$$\theta_1 \sum_{i=1}^{m}(x_1^{(i)})^2 - \sum_{i=1}^{m}y^{(i)}x_1^{(i)} = 0$$

i.e.  $\theta_1 = \sum_{i=1}^{m}y^{(i)}x_1^{(i)}/\sum_{i=1}^{m}(x_1^{(i)})^2$
Closed-form solution

\[ \theta_1 = \frac{\sum_{i=1}^{m} y^{(i)} x_1^{(i)}}{\sum_{i=1}^{m} (x_1^{(i)})^2} \]

In vector-matrix notation:

\[ \theta_1 = (X^T X)^{-1} X^T y \]

where \( y = \begin{bmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \), \( X = \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(m)} \end{bmatrix} \)
Closed-form solution (optional)

With multiple features, the minimising vector $\theta$ is

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

where $y = \begin{bmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$, $X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$

- $(X^T X)^{-1}$ is the inverse of matrix $X^T X$
- It satisfies $(X^T X)^{-1}(X^T X) = I$ where $I$ is the identity matrix.
Comparison of Solution Approaches

$m$ training examples, $n$ features

**Gradient Descent**
- Need to choose $\alpha$
- May need many iterations
- Works well even when $n$ is large  
  e.g. $n = 10^6$
- Stochastic gradient descent variant works well even when $m$ is very large (big data)

**Closed-form Solution**
- No need to choose $\alpha$
- Don’t need to iterate
- Need to compute $(X^T X)^{-1}$, $n$ by $n$ matrix and $O(n^3)$ operation. Slow if $n$ is large-ish  
  e.g. ok up to about $n = 1000$.
- Matrix $X$ is $m$ by $n$ so can be hard to work with if $m$ is very large (big data)
Regularisation & Model Selection

Advertising example again. Thin out data by taking every 10th point. Try a few different hypothesis:

- \( h_\theta(x) = \theta_0 + \theta_1 x \)
- \( h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \)
- \( h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_6 x^6 \)
- \( h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_{10} x^{10} \)

- As we add more parameters, we start to fit the “noise” in the training data, called **overfitting**.
- But of use too few parameters then will get a poor fit, **underfitting**.
- How to strike the right balance between these? This is an example of the **bias-variance trade-off**.
More data can help, e.g. when don’t thin out data:

- But even with more data, still our hypothesis doesn’t generalise well i.e. doesn’t predict well for data outside the training set.
Hold-Out Method\textsuperscript{1}

One tactic is to use \textit{cross-validation} for \textit{feature/model selection}. Simplest approach is the \textbf{hold out} method.

- Divide our data into two parts, a training part and a test part. E.g. use 80\% of data for training part and 20\% for test part.
- Train model on the training part of the data.
- Evaluate its performance on the test part of the data e.g. calculate the cost function $J(\theta)$ for this data. Since parameters $\theta$ have been trained using separate data we hope that this gives us a rough idea of the performance of our model on new data.

Note:

- There is a trade-off in how we split the data. If we use more for the training part then we expect our model to be better trained, but then we have less data to test the model on. And vice versa. An 80/20 or 90/10 split is common.
- Typically split the data randomly. So as avoid inadvertently introducing bias.

\textsuperscript{1}https://en.wikipedia.org/wiki/Cross-validation_(statistics)
Hold-Out Method: Model Selection

Advertising data again. Split data 80/20 and plot value of $J(\theta)$ obtained on test data as the order $q$ of the polynomial used in the hypothesis is varied from 1 to 10, $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_q x^q$

- Error is similar for all polynomials up to about 4, then increases rapidly. Linear fit ($q = 1$) thus seems reasonable.
- Note that since we have randomly split the data 80/20 into training and test parts, these plotted values are themselves “noisy” (will change depending on the random sample used).
**k-Fold Cross-validation**

Hold-out method only gives us a single estimate of the cost function $J(\theta)$ from the test data. So cannot estimate the spread of values. We can make better use of our data by using $k$-fold cross-validation:

- Divide our data into $k$ equal sized parts
- Use part 1 as test data and the rest as training data. i.e. train model using all of the data except part 1, then calc $J(\theta)$ for part 1 data
- Use part 2 as test data and the rest as training data, and so on.
- This gives us $k$ estimates of $J(\theta)$ and so we can use this to estimate the average and the spread of values.
- How to choose $k$? Common choices are $k = 5$ or $k = 10$. 
**k-Cross-validation: Model Selection**

Advertising data again. Use *k*-fold cross-validation with \( k = 5 \). Plot value of **mean** and **standard deviation** of \( J(\theta) \) obtained on test data as the order \( q \) of the polynomial used is varied.

- The centre point is the average over \( k = 5 \) samples so it is “smoother” than in the leave one out case (see earlier). The increase in cost for \( q > 1 \) is now more evident.
- Importantly, we now have error bars that give us an idea of the spread in cost values at each point. In this way it can be seen that the increase in cost with \( q \) is “significant”.

![Graph showing the mean square error as a function of the order of the polynomial, with error bars indicating the standard deviation. The graph demonstrates that the mean square error increases significantly with higher order polynomials.]
Another tool in our armoury is **regularisation** i.e. add constraints/penalties on the parameters $\theta$. That is, change $J(\theta)$ to

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

where $R(\theta)$ is a penalty function. Two common regularisation penalties:

- **Quadratic/L2 penalty:** $R(\theta) = \theta^T \theta = \sum_{j=1}^{n} \theta_j^2$. Also called Tikhonov regularisation. Ridge regression. Encourages elements of $\theta$ to have small value.

- **L1 penalty:** $R(\theta) = \sum_{j=1}^{n} |\theta_j|$. LASSO regression. Encourages sparsity of solution i.e. few non-zero elements in $\theta$. 


Regularisation: Ridge Regression

Select \( \theta \) to minimise

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

This is called **ridge regression**.

- When \( \lambda \to 0 \), then we are saying that we are certain \( \theta = 0 \).
- When \( \lambda \) is large we are saying that we know little about the value of \( \theta \) prior to making the observations. The penalised estimate is then close to the non-penalised estimate.
- We can also use cross-validation to help choose \( \lambda \)
Regularisation: Ridge Regression

Example: training data is $y^{(i)} = x^{(i)} + N^{(i)}$, $i = 1, \ldots, m$ where $N^{(i)}$ is noise.
Regularisation: Ridge Regression

Impact of $\lambda$:

![Graph showing the impact of $\lambda$ on the estimate of $\theta$. The graph compares regularised and non-regularised estimates. As $\lambda$ increases, the regularised estimate approaches the non-regularised estimate.]
Regularisation

Regularisation really kicks in when we only have a small number of observations, yet still need to make a prediction. Our prior beliefs regarding whether $\theta$ is small or not are then especially important.
Regularisation

- But as number $m$ of observations grows, other things being equal the impact of regularisation tends to decline.