Language Models (LMs)

- based on the https://medium.com/@philiposbornedata/learning-nlp-language-models-with-real-data-cdff04c51c25
- based on Speech and Language Processing (3rd ed. draft) by Dan Jurafsky and James H. Martin

Enable equation numbering in jupyter notebook for improved readability:

In [ ]:
```
%%javascript
MathJax.Hub.Config({
    TeX: { equationNumbers: { autoNumber: "AMS" } }
});
```

Introduction

Part 1

I. Probabilistic Language Models

Example uses

- **Machine Translation:** P(**high** winds tonight) > P(**large** winds tonight)
- **Spell Correction:** P(...about fifteen **minutes** from...) > P(...about fifteen **mineuts** from...)
- **Speech Recognition:** P(I saw a van) >> P(eyes awe of an)

Aim of Language Models

The goal of paobabilistic language modelling is to calculate the probability of a sentence of sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n) \tag{1}$$

and can be used to find the probability of the next word in the sequence:

$$P(w_5|w_1, w_2, w_3, w_4) \tag{2}$$

a model that computes either of these is called a **language model (LM)**.

II. Initial Method for Calcualting Probabilities

Defn: Conditional Probability

Let A and B be two events with $P(B) \neq 0$, the conditional probability of A given B is:

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{3}$$

Defn: Chain Rule

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1, \ldots, x_{n-1}) \tag{4}$$

The Chain Rule applied to compute the join probability of words in a sentence:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i|w_1 w_2 \ldots w_{i-1}) \tag{5}$$

e.g.

P("its water is so transparent") =

```
P(its)xP(water|its)
x P(is|its water)
x P(so|its water is)
x P(transparent|its water is so)
```

Can we estimate this by simply counting and dividing the results by the following?

$$P(transparent|its\ water\ is\ so) = \frac{count(its\ water\ is\ so\ transparent)}{count(its\ water\ is\ so)} \tag{6}$$

NO! Far to many possible sentences that would need to be calculated, we would never have enough data to achieve this.

III. Methods using the Markov Assumption

Defn: Markov Property

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process. [1]

In other words, the probability of the next word can be estimated given only the previous k number of words.

e.g.

$$P(transparent|its\ water\ is\ so) \approx P(transparent|so) \tag{7}$$

or

$$P(transparent|its\ water\ is\ so) \approx P(transparent|is\ so) \tag{8}$$

General Equation for the Markov Assumption

$$P(w_i|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-k} \dots w_{i-1}) \tag{9}$$

where k is the number of words in the 'state' to be defined by the user.

Unigram Model (k=1)

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i) \tag{10}$$

Bigram Model (k=2)

$$P(w_i|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-1}) \tag{11}$$

IV. N-gram Models (k=n)

The previous two equations can be extended to compute trigrams, 4-grams, 5-grams, etc. In general, this is an insufficient model of language because **sentences often have long distance dependencies**. For example, the subject of the sentence may be at the start whilst our next word to be predicited occurs more than 10 words later.

Estimating Bigram Probabilities using the Maximum Likelihood Estimate

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \tag{12}$$

Small Example

Three sentences:

- < s I am Sam /s >
- < s Sam I am /s >
- < s I do not like green eggs and ham /s >

$$P(I| < s) = \frac{count(< s, I)}{count(< s)} = \frac{2}{3} \tag{13}$$

$$P(am|I) = \frac{count(I, am)}{count(I)} = \frac{2}{3} \tag{14}$$

[1] https://en.wikipedia.org/wiki/Markov_property

Part 2

V. Training and Testing the Language Models (LMs)

The corpus used to train our LMs will impact the output predictions. Therefore we need to introduce a methodology for evaluating how well our trained LMs perform. The best trained LM is the one that can correctly **predict the next word of setences in an unseen test set.**

This can be time consuming, to build multiple LMs for comparison could take hours to compute. Therefore, we introduce the intrinsic evaluation method of **perplexity**. In short perplexity is a measure of how well a probability distribution or probability model predicts a sample. [3]

Defn: Perplexity

Perplexity is the inverse probability of the test set normalised by the number of words, more specifically can be defined by the following equation:

$$PP(W) = P(w_1 w_2 \dots w_N)^{\frac{-1}{N}} \tag{15}$$

e.g. Suppose a sentence consists of random digits [0-9], what is the perplexit of this sentence by a model that asigns an equal probability (i.e. $P = 1/10$) to each digit?

$$PP(W) = (\frac{1}{10} * \frac{1}{10} * \dots * \frac{1}{10})^{\frac{-1}{10}} = (\frac{1}{10}^{10})^{\frac{-1}{10}} = \frac{1}{10}^{-1} = 10 \tag{16}$$

VI. Entropy in Information Theory

Prerequisite: 정보량 (I)

- 어떤 한 사건(event)에서 기대되는 정보량 (I)을 확률과 관련하여 살펴 봄

중요성(significance): 어떤 사건이 일어날 가능성이 작으면 작을수록, 그 사건은 더 많은 정보를 지닌다.

중요성 조건은 어떤 사건의 확률이 높을수록 이 사건으로 알려지는 정보량은 적어짐을 나타낸다. 따라서 확률값을 역으로 취하여 중요성에 따른 정보량을 나타낼 수 있다.

$P(x1) > P(x2) \Rightarrow I(x1) < I(x2)$

$I(x) = 1/P(x)$

가법성(additivity): 만일 x1, x2 가 독립적인 사건이라면 다음을 만족해야 한다.

$I(x1x2) = I(x1) + I(x2)$

예)

$P(x1) = 1/2$이면$I(x1) = 2$

$P(x2) = 1/4$이면$I(x2) = 4$

따라서 중요성의 조건이 만족된다. 만일 두 사건이 서로 독립적이라면,

$P(x1x2) = P(x1) * P(x2) = 1/2 * 1/4 = 1/8$

$I(x1x2) = 1/P(x1x2) = 8$

그러나 가법성에 따라

$I(x1x2) = I(x1) + I(x2) = 2 + 4 = 6$

이다. 따라서 가법성의 조건이 충족되지 못한다.

두 독립 사건의 확률값은 곱으로 이루어지지만, 두 사건의 결합된 정보 내용은 더해져야만 한다. 정보의 가법성을 위해서 곱이 아닌 더하기가 필요하다. 따라서 이와 유사한 기능을 하는 log를 도입하게 된다

$log(xy) = logx + logy$

즉 확률을 역으로 하여 중요성 기준을 만족시킬 수 있고, log를 사용 하여 가법성의 조건을 만족시킬 수 있다. 이제 이 둘을 결합하면 어떤 확률 변수 x 가 지니는 정보량은 다음과 같이 계산될 수 있다.

$I(x) = log1/P(x) = -logP(x)$

예)

$P(x1) = 1/2$이면$I(x1) = -log(1/2) = log2 = 1$

$P(x2) = 1/4$이면$I(x2) = -log(1/4) = log4 = 2$

$P(x1) > P(x2) \Rightarrow I(x1) < I(x2)$ 인 중요성의 조건을 만족한다.

$P(x1x2) = P(x1)P(x2) = 1/8 \Rightarrow I(x1x2) = -log(1/8) = 3$

$I(x1x2) = I(x1) + I(x2) = 1 + 2 = 3$

으로 가법성의 조건도 만족하게 된다.

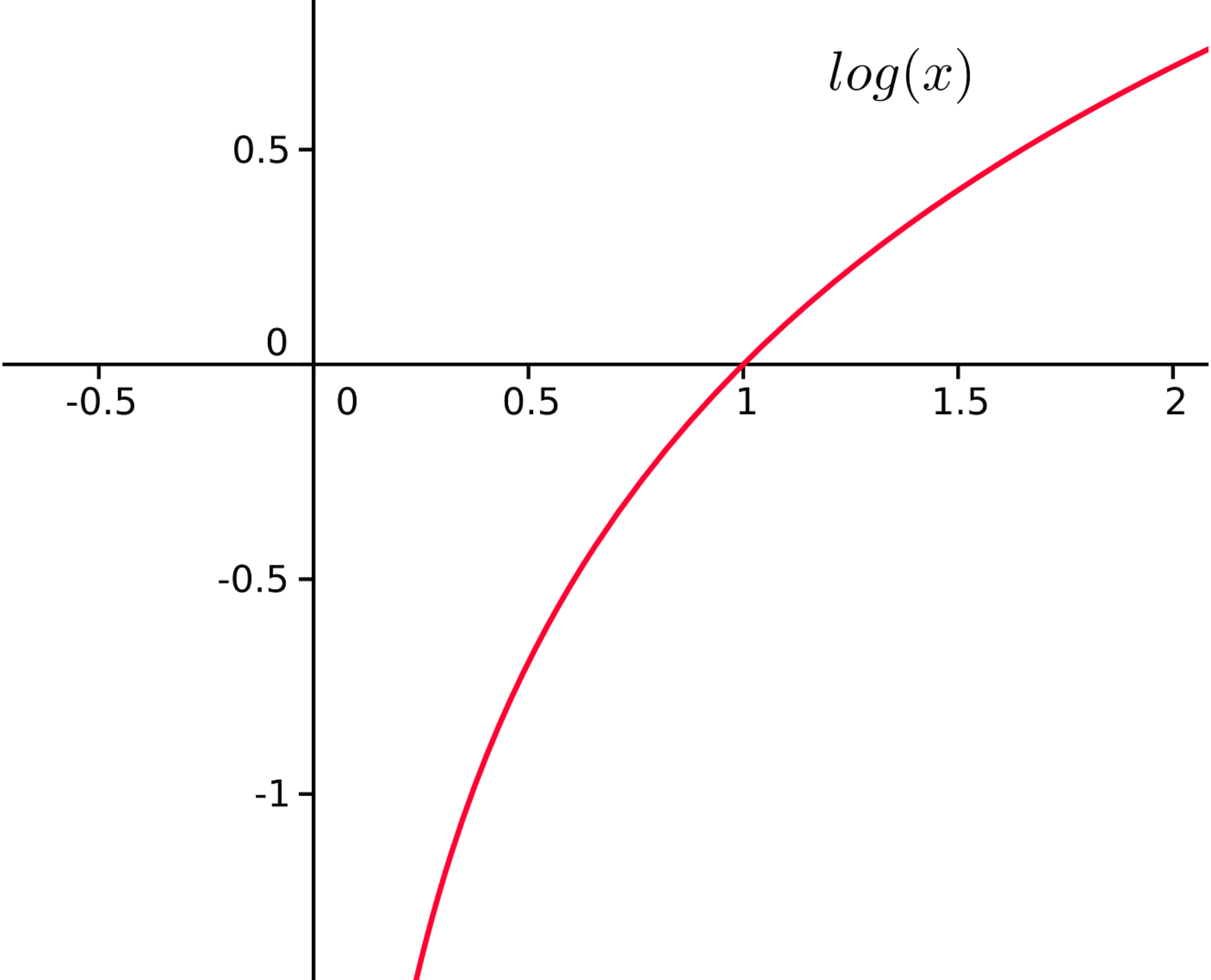
In Information Theory, entropy (denoted $H(X)$) of a random variable X is the expected log probabiltiy:

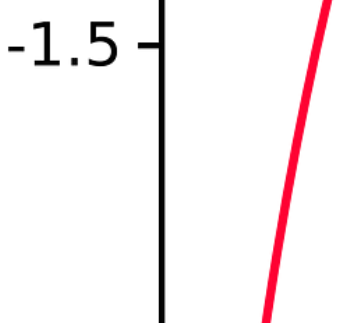
$$H(X) = -\sum P(x) log_2 P(x) \tag{17}$$

and is a measure of uncertainty. [4]

Reason for negative sign:

- log(p(x))<0 for all p(x) in (0,1) . p(x) is a probability distribution and therefore the values must range between 0 and 1.


log(x)
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2



A plot of log(x). For x values between 0 and 1, log(x) <0 (is negative)

In other words, entropy is the number of possible states that a system can be.

Entropy of a bias coin toss

Say we have the probabilities of heads and tails in a coin toss defined by:

- $P(heads) = p$
- $P(tails) = 1 - p$

Then the entropy of this is:

$$H(X) = -\sum P(x) log_2 P(x) = -[p log_2 p + (1-p) log_2 (1-p)] \tag{18}$$

If the coin is fair, i.e. p = 0.5, then we have:

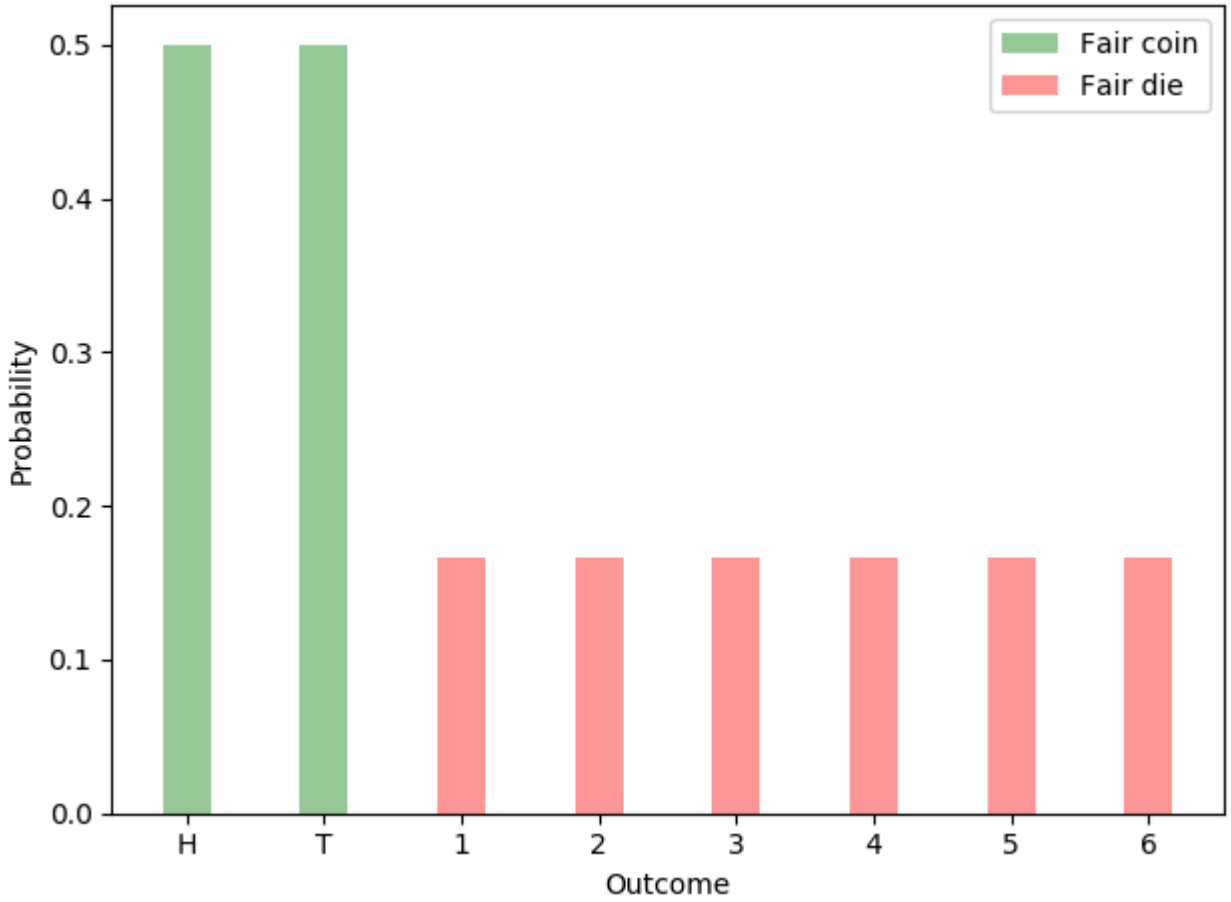
$$H(X) = -[0.5 log_2 0.5 + (1-0.5) log_2 (1-0.5)] = -[-0.5 - 0.5] = 1 \tag{19}$$

The full entropy distibution over varying bias probabilities is shown below.

[3] https://en.wikipedia.org/wiki/Perplexity [4] https://en.wikipedia.org/wiki/Entropy_(information_theory)

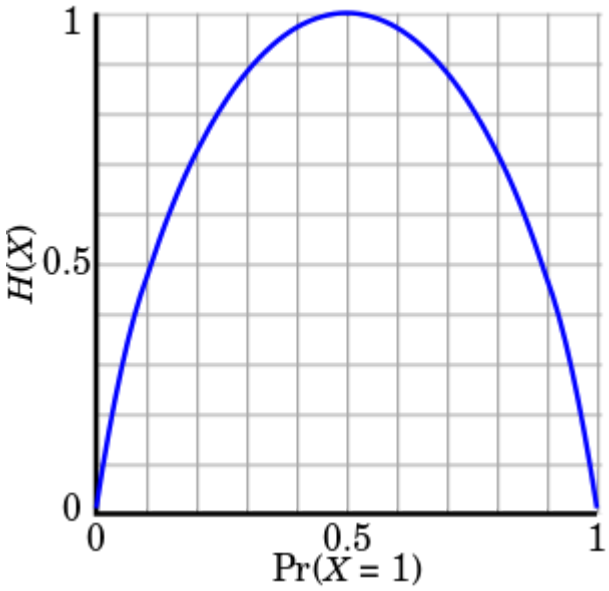
Basic property 1: Uniform distributions have maximum uncertainty

If your goal is to minimize uncertainty, stay away from **uniform probability distributions**.



uniform distributions have maximum entropy for a given numger of outcomes

Here is the plot of the Entropy function as applied to Bernoulli trials (events with two possible outcomes and probabilities p and 1-p):



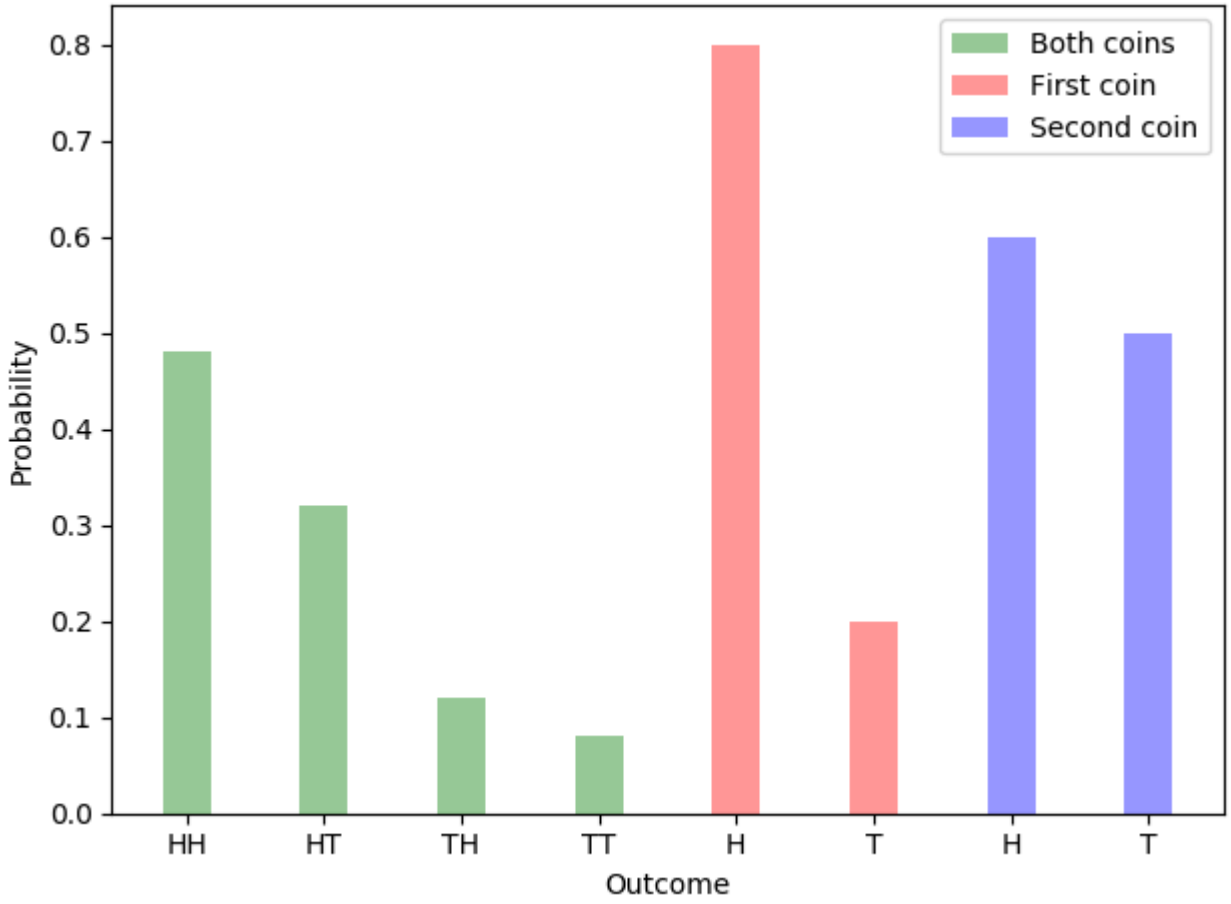
Basic property 2: Uncertainty is additive for independent events

Let A and B be independent events. In other words, knowing the outcome of event A does not tell us anything about the outcome of event B. The uncertainty associated with both events — this is another item on our wish list — should be the sum of the individual uncertainties:

$$H(X,Y) = H(X) + H(Y)$$ uncertainty is additive for independent events

Let's use the example of flipping two coins to make this more concrete. We can either flip both coins simultaneously or first flip one coin and then flip the other one. Another way to think about this is that we can either report the outcome of the two coin flips at once or separately. The uncertainty is the same in either case.

To make this even more concrete, consider two particular coins. The first coin lands heads (H) up with an 80% probability and tails (T) up with a probability of 20%. The probabilities for the other coin are 60% and 40%. If we flip both coins simultaneously, there are four possible outcomes: HH, HT, TH and TT. The corresponding probabilities are given by [0.48, 0.32, 0.12, 0.08].



The joint entropy (green) for the two independent events is equal to the sum of the individual events (red and blue).

Plugging the numbers into the entropy formula, we see that:

$$H(0.48, 0.32, 0.12, 0.08) = H(0.8, 0.2) + H(0.6, 0.4)$$

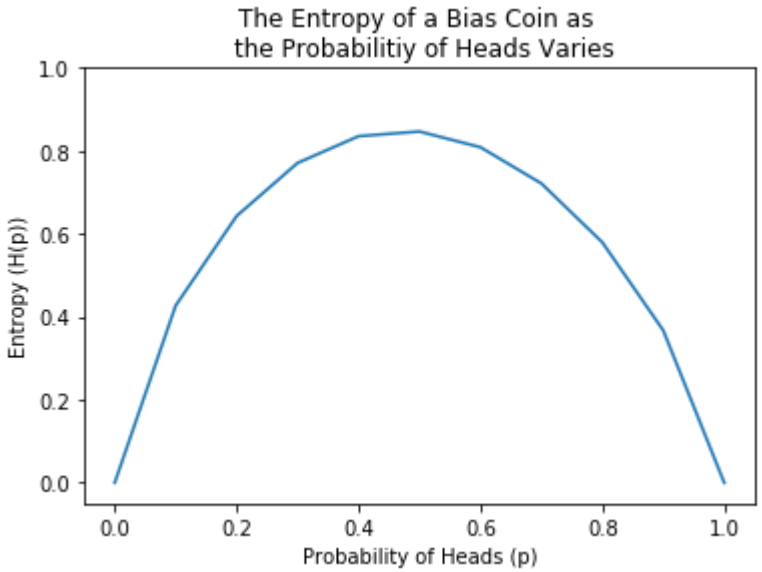
In [5]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
p = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
H = [-(p*np.log2(p) + (1-p)*np.log(1-p)) for p in p]
# Replace nan output with 0
H = [0 if math.isnan(x) else x for x in H]

plt.plot(p,H)
```

```
plt.xlim([-0.05,1.05])
plt.ylim([-0.05,1])
plt.xlabel('Probability of Heads (p)')
plt.ylabel('Entropy (H(p))')
plt.title('The Entropy of a Bias Coin as \n the Probabilitiy of Heads Varies')
```

```
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: divide by zero encountered in log2
  """
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: invalid value encountered in double_sc
alars
  """
/home/hpshin/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: divide by zero encountered in log
  """
```

Out[5]: Text(0.5,1,'The Entropy of a Bias Coin as \n the Probabilitiy of Heads Varies')



Basic property 3: Entropy is measured in bits, if we use log base 2

Cover and Thomas (1991) suggest the following example. Imagine that we want to place a bet on a horse race but it is too far to go all the way to Yonkers Racetrack, and we'd like to send a short message to the bookie to tell him which horse to bet on. Suppose there are eight horses in this particular race.

One way to encode this message is just to use the binary representation of the horse's number as the code; thus horse 1 would be `001`, horse 2 `010`, horse 3 `011`, and so on, with horse 8 coded as `000`. If we spend the whole day betting, and each horse is coded with 3 bits, on the average we would be sending 3 bits per race.

Can we do better? Suppose that the spread is the actual distribution of the bets placed, and that we represent it as the prior probability of each horse as follows:

Horse 1	$\frac{1}{2}$	Horse 5	$\frac{1}{64}$
Horse 2	$\frac{1}{4}$	Horse 6	$\frac{1}{64}$
Horse 3	$\frac{1}{8}$	Horse 7	$\frac{1}{64}$
Horse 4	$\frac{1}{16}$	Horse 8	$\frac{1}{64}$

The entropy of the random variable X that ranges over horses gives us a lower bound on the number of bits, and is:

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i)\log p(i) \\ &= -\tfrac{1}{2}\log\tfrac{1}{2} - \tfrac{1}{4}\log\tfrac{1}{4} - \tfrac{1}{8}\log\tfrac{1}{8} - \tfrac{1}{16}\log\tfrac{1}{16} - 4(\tfrac{1}{64}\log\tfrac{1}{64}) \\ &= 2 \text{ bits} \end{aligned}$$

5)

A code that averages 2 bits per race can be built by using short encodings for

more probable horses, and longer encodings for less probable horses. For example, we could encode the most likely horse with the code `0`, and the remaining horses as `10`, then `110`, `1110`, `111100`, `111101`, `111110`, and `111111`.

What if the horses are equally likely? We saw above that if we use an equal-length binary code for the horse numbers, each horse took 3 bits to code, and so the average was 3. Is the entropy the same? In this case each horse would have a probability of $\frac{1}{8}$. The entropy of the choice of horses is then:

$$H(X) = -\sum_{i=1}^{i=8} \frac{1}{8} \log \frac{1}{8} = -\log \frac{1}{8} = 3 \text{ bits}$$

VII. Entropy of Language

1. Entropy of a sequence of words:

$$H(w_1 w_2 \dots w_n) = -\sum_{w_1 \dots wn} P(w_1 \dots w_n) log_2 P(w_1 \dots w_n) \tag{20}$$

2. The per-word entropy rate of a sequence of words

$$\frac{1}{n} H(w_1 w_2 \dots w_n) = \frac{-1}{n} \sum_{w1 \dots wn} P(w_1 \dots w_n) log_2 P(w_1 \dots w_n), 2) \tag{21}$$

3. Entropy of a language $L = \{w_1 \dots w_n | 1 < n < \infty\}$:

$$H(L) = -\lim_{n \to \infty} \frac{1}{n} H(w_1 \dots w_n) \tag{22}$$

$$H(L) = \lim_{n \to \infty} -\frac{1}{n} \log p(w_1 w_2 \dots w_n)$$

Defn: Cross Entropy

The cross entropy, H(p,m), of a true distribution **p** and a model distribution **m** is defined as:

$$H(p, m) = -\sum_{x} p(x) log_2 m(x) \tag{23}$$

The lower the cross entropy is the closer it is to the true distribution.

Defn: Cross Entropy of a Sequence of Words

$$H(p, m) = -\lim_{n \to \infty} \frac{1}{n} \sum_{w1 \dots wn} p(w_1 \dots w_n) log_2 m(w_1 \dots w_n) \tag{24}$$

$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N)$$

VIII. Perplexity and Entropy

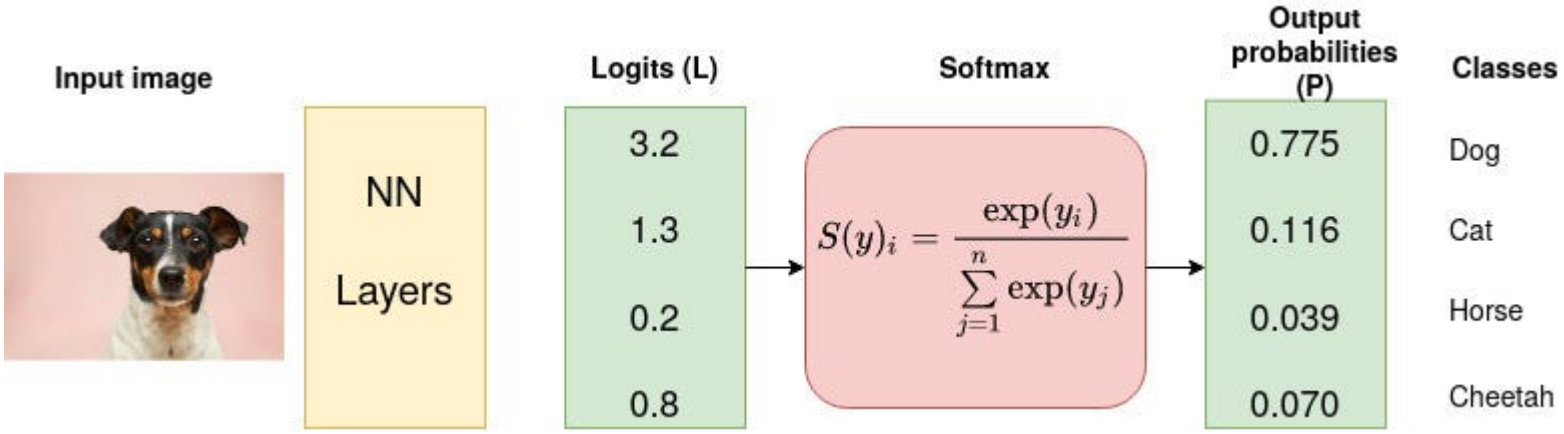
$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N)$$

$$\begin{aligned}
\text{Perplexity}(W) &= 2^{H(W)} \\
&= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\
&= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}
\end{aligned}$$

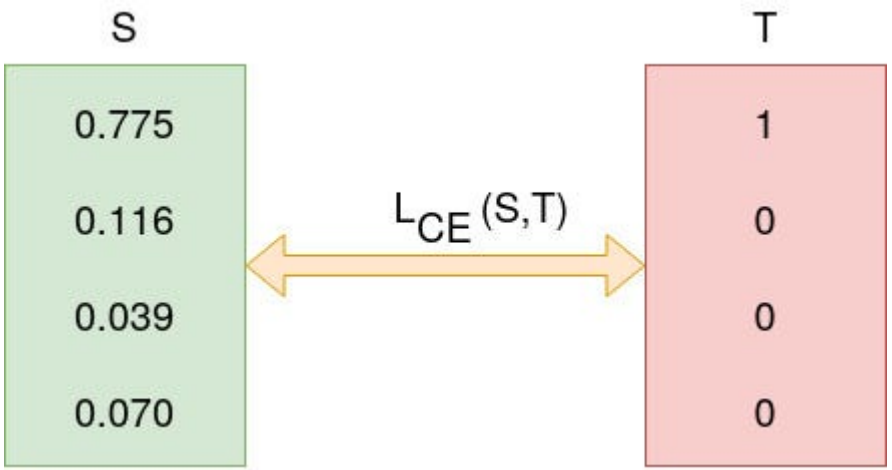
Cross-Entropy Loss Function

- Also called logarithmic loss, log loss or logistic loss. Each predicted class probability is compared to the actual class desired output 0 or 1 and a score/loss is calculated that penalizes the probability based on how far it is from the actual expected value.
- The penalty is logarithmic in nature yielding a large score for large differences close to 1 and small score for small differences tending to 0.

- Consider a 4-class classification task where an image is classified as either a dog, cat, horse or cheetah.



- In the above Figure, Softmax converts logits into probabilities.
- The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values (as shown in Figure below).



- For the example above the desired output is [1,0,0,0] for the class dog but the model outputs [0.775, 0.116, 0.039, 0.070] .
- The objective is to make the model output be as close as possible to the desired output (truth values).
- During model training, the model weights are iteratively adjusted accordingly with the aim of minimizing the Cross-Entropy loss.
- The process of adjusting the weights is what defines model training and as the model keeps training and the loss is getting minimized, we say that the model is learning.

- Cross-entropy loss is used when adjusting model weights during training.
- The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0.
- Cross-entropy is defined as

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Binary Cross-Entropy Loss

- For binary classification, we have binary cross-entropy defined as

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log(p_i) \\ &= -\left[t \log(p) + (1-t) \log(1-p)\right] \end{aligned}$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class.

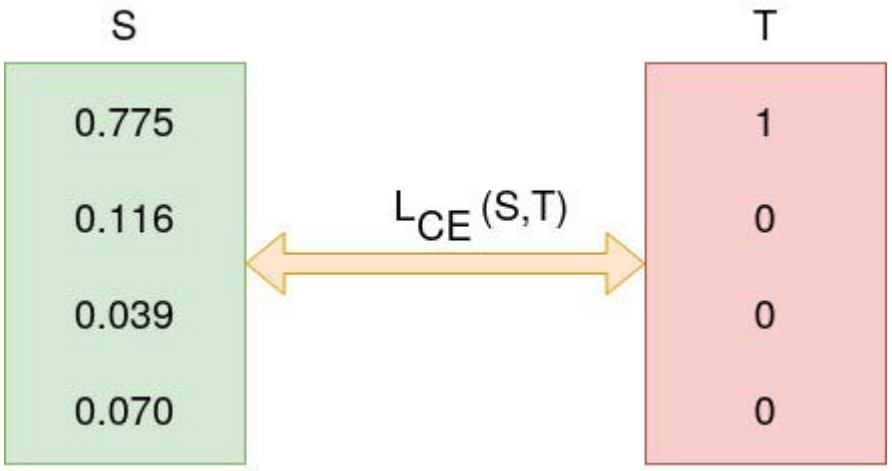
- Binary cross-entropy is often calculated as the average cross-entropy across all data examples

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]$$

for N data points where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} data point.

Example

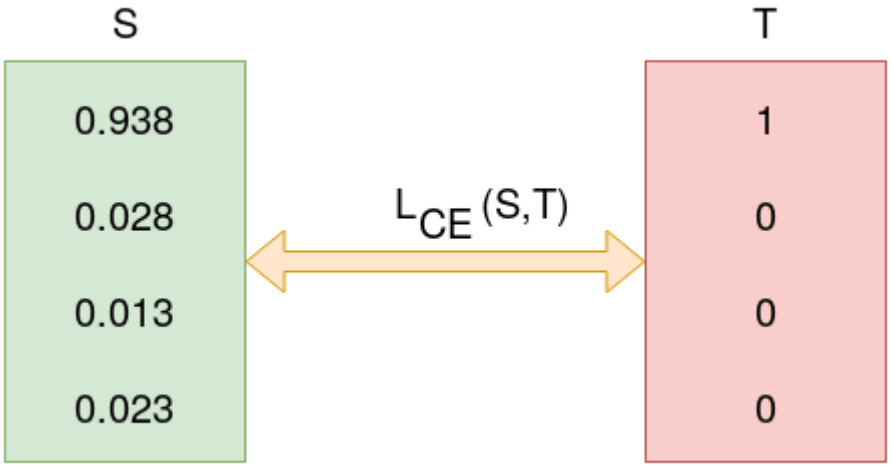
- Consider the classification problem with the following Softmax probabilities (S) and the labels (T).
- The objective is to calculate for cross-entropy loss given these information.



- The categorical cross-entropy is computed as follows

$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1 \log_2(0.775) + 0 \log_2(0.126) + 0 \log_2(0.039) + 0 \log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

- Softmax is continuously differentiable function.
- This makes it possible to calculate the derivative of the loss function with respect to every weight in the neural network.
- This property allows the model to adjust the weights accordingly to minimize the loss function (model output close to the true values).
- Assume that after some iterations of model training the model outputs the following vector of logits



$$
\begin{aligned}
L_{CE} &= -1 \log_2(0.936) + 0 + 0 + 0 \\
&= 0.095
\end{aligned}
$$

- 0.095 is less than previous loss, that is, 0.3677 implying that the model is learning.
- The process of optimization (adjusting weights so that the output is close to true values) continues until training is over.

Part 3

Challenges in Fitting LMs

Due to the output of LMs is dependent on the training corpus, N-grams only work well if the training corpus is similar to the testing dataset and we risk overfitting in training.

As with any machine learning method, we would like results that are generalisable to new information.

Even harder is how we deal with words that do not even appear in training but are in the test data.

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

IX. Dealing with Zero Counts in Training: Laplace +1 Smoothing

To deal with words that are unseen in training we can introduce add-one smoothing. To do this, we simply add one to the count of each word.

This shifts the distribution slightly and is often used in text classification and domains where the number of zeros isn't large. However, this is not often used for n-grams, instead we use more complex methods.

First, let us create a dummy training corpus and test set from the original data:

Adjusted Counting

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

- I added Adjusted count for better understanding

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 estimation is a blunt instrument

So add-1 isn't used for N-grams:

- We'll see better methods

But add-1 is used to smooth other NLP models

- For text classification
- In domains where the number of zeros isn't so huge.

X. Futher Smoothing Methods

Laplace +1 smoothing is used in text classification and domains where the number of zeros isn't large. However, it is not often used for n-grams, some better smothing methods for n-grams are:

- Add-k Laplace Smoothing
- Good-Turing
- Kenser-Ney
- Witten-Bell

Part 4

Selecting the Language Model to Use

We have introduced the first three LMs (unigram, bigram and trigram) but which is best to use?

Trigrams are generally provide better outputs than bigrams and bigrams provide better outputs than unigrams but as we increase the complexity the computation time becomes increasingly large. Furthermore, the amount of data available decreases as we increase n (i.e. there will be far fewer next words available in a 10-gram than a bigram model).

XI. Back-off Method: Use trigrams (or higher n model) if there is good evidence to, else use bigrams (or other simpler n-gram model).

XII. Interpolation: Use a mixture of n-gram models

Defn: Simple Interpolation:

$$P(w_3|w_1,w_2) = \lambda_1 P(w3|w_1,w_2) + \lambda_2 P(w_3|w_2) + \lambda_3 P(w_3) \tag{25}$$

where $\sum_i \lambda_i = 1$.

Defn: Contidional Context Interpolation:

$$P(w_3|w_1,w_2) = \lambda_1(w_1^2) P(w3|w_1,w_2) + \lambda_2(w_1^2) P(w_3|w_2) + \lambda_3(w_1^2) P(w_3) \tag{26}$$

Calculating λs:

Using a held-out subset of the corpus (validation set), find λs that maximise the probability of the held out data:

$$P(w_1,w_2,\ldots,w_n|M(\lambda_1,\lambda_2,\ldots,\lambda_k)) = \sum_i log P_{M(\lambda_1,\lambda_2,\ldots,\lambda_k)}(w_i|w_{i-1}) \tag{27}$$

Where unknown words are assigned an unknown word token '< Unk >'.

Small Interpolation Example

Say we are given the following corpus:

- < s I am Sam /s >
- < s Sam I am /s >
- < s I am Sam /s >

- < s I do not like green eggs and Sam /s >

Using linear interpolation smoothing with a bigram and unigram model with $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = \frac{1}{2}$, what is $P(Sam|am)$? (note: include '< s' and '/s >' in calculations)

Using the following equation:

$$P(w_2|w_1) = lambda_1 P(w_2|w1) + lambda_2 P(w2) \tag{28}$$

We have in our case:

$$P(Sam|am) = \frac{1}{2} P(Sam|am) + \frac{1}{2} P(Sam) \tag{29}$$

where

$$P(Sam|am) = \frac{count(am, Sam)}{count(am)} = \frac{2}{3} \tag{30}$$

and

$$P(Sam) = \frac{count(Sam)}{Total\ num\ words} = \frac{4}{25} \tag{31}$$

Therefore,

$$P(Sam|am) = \frac{1}{2} * \frac{2}{3} + \frac{1}{2} * \frac{4}{25} \approx 0.413 \tag{32}$$

In [ ]: