COS 484/584

Spring 2021

L2: Language Models
Reminder: Assignment 0 is out — due Monday, Feb 8, 1:30pm

[COS 584] Readings and questions for this Friday’s precept are out on Perusall

- Make sure you have notifications turned on for Canvas announcements!

- New: FAQ section on the class website - will be continually updated

- If you have a question:

  • Ask in chat! TA will help answer or bring to instructor’s attention during pauses.

  • Or use the raise hand feature
Last class

\[ p(w_1, w_2, w_3, \ldots, w_N) = p(w_1) p(w_2 | w_1) p(w_3 | w_1, w_2) \times \cdots \times p(w_N | w_1, w_2, \ldots w_{N-1}) \]

Sentence: “the cat sat on the mat”

\[ P(\text{the cat sat on the mat}) = P(\text{the}) \times P(\text{cat} | \text{the}) \times P(\text{sat} | \text{the cat}) \times P(\text{on} | \text{the cat sat}) \times P(\text{the} | \text{the cat sat on}) \times P(\text{mat} | \text{the cat sat on the}) \]
Estimating probabilities

With a vocabulary of size \( v \),

\[
\text{# sequences of length } n = v^n
\]

Typical vocabulary \( \sim 40k \) words

- even sentences of length \( \leq 11 \) results in more than \( 4 \times 10^{50} \) sequences!
  
  \( (# \text{ of atoms in the earth } \sim 10^{50}) \)

Implicit order

\[
P(\text{sat}|\text{the cat}) = \frac{\text{count(\text{the cat sat})}}{\text{count(\text{the cat})}}
\]

\[
P(\text{on}|\text{the cat sat}) = \frac{\text{count(\text{the cat sat on})}}{\text{count(\text{the cat sat})}}
\]

\[\vdots\]

Maximum likelihood estimate (MLE)
Markov assumption

- Use only the recent past to predict the next word

- Reduces the number of estimated parameters in exchange for modeling capacity

- 1st order

\[ P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the}) \]

- 2nd order

\[ P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the}) \]
**$k^{\text{th}}$ order Markov**

- Consider only the last $k$ words for context

\[
P(w_i | w_1w_2 \ldots w_{i-1}) \approx P(w_i | w_{i-k} \ldots w_{i-1})
\]

which implies the probability of a sequence is:

\[
P(w_1w_2 \ldots w_n) \approx \prod_i P(w_i | w_{i-k} \ldots w_{i-1})
\]

(assume $w_j = \phi \quad \forall j < 0$)

(k+1) gram
n-gram models

Unigram

\[ P(w_1, w_2, ... w_n) = \prod_{i=1}^{n} P(w_i) \]

Bigram

\[ P(w_1, w_2, ... w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1}) \]

and Trigram, 4-gram, and so on.

Larger the \( n \), more accurate and better the language model (but also higher costs)

Caveat: Assuming infinite data!
Generations

**Unigram**

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

**Bigram**

Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton .

**Trigram**

We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdaZt3WV

\[
\arg\max_{(w_1, w_2, \ldots, w_n)} P(w_1, w_2, \ldots, w_n) = \arg\max_{(w_1, w_2, \ldots, w_n)} \prod_{i=1}^{n} P(w_i | w_{i-k}, \ldots, w_{i-1})
\]
Generations

release millions See ABC accurate President of Donald Will
cheat them a CNN megynkelly experience @ these word
out- the

Thank you believe that @ ABC news, Mississippi tonight
and the false editorial I think the great people Bill Clinton
.

We are going to MAKE AMERICA GREAT AGAIN!
#MakeAmericaGreatAgain https://t.co/DjkdAzT3WV

Typical LMs are not sufficient to handle long-range dependencies

“Alice/Bob could not go to work that day because
she/he had a doctor’s appointment”
Evaluating language models

• A good language model should assign **higher probability** to typical, grammatically correct sentences

• Research process:
  
  • **Train** parameters on a suitable training corpus
  
  • Assumption: observed sentences ~ good sentences
  
  • **Test** on different, unseen corpus
  
  • Training on any part of test set not acceptable!

• Evaluation metric
Extrinsic evaluation

- Train LM -> apply to task -> observe accuracy
- Directly optimized for downstream tasks
  - higher task accuracy -> better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)
Perplexity (ppl)

- Measure of how well a probability distribution (or LM) predicts a sample

- For a corpus $S$ with sentences $S^1, S^2, \ldots, S^n$

  $$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W} \sum_{i=1}^{n} \log_2 P(S^i)$$

  where $W$ is the total number of words in test corpus

- Unigram model:  
  $$x = -\frac{1}{W} \sum_{i=1}^{n} \sum_{j=1}^{m} \log_2 P(w^i_j) \quad \text{(since } P(S) = \prod_j P(w_j) \text{)}$$

- Minimizing perplexity $\sim$ maximizing probability of corpus $P(S^1 S^2 \ldots S^n)$
Intuition on perplexity

If our n-gram model (with vocabulary V) has following probability:

\[ P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i \]

what is the perplexity of the test corpus?

\[ \text{ppl}(S) = 2^x \quad \text{where} \]
\[ x = -\frac{1}{W} \sum_{i=1}^{n} \log_2 P(S^i) \]

\[ \text{ppl} = 2^{-\frac{1}{W}W*\log(1/|V|)} = |V| \]

(model is ‘fine’ with observing any word at every step)

Measure of model’s uncertainty about next word
## Perplexity as a metric

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Perplexity as a metric

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to compute</td>
<td>Requires domain match between train and test</td>
</tr>
<tr>
<td>standardized</td>
<td>might not correspond to end task optimization</td>
</tr>
<tr>
<td>directly useful, easy to use to correct sentences</td>
<td>log 0 undefined</td>
</tr>
<tr>
<td>nice theoretical interpretation - matching distributions</td>
<td>can be ‘cheated’ by predicting common tokens</td>
</tr>
<tr>
<td>size of test set matters</td>
<td></td>
</tr>
<tr>
<td>can be sensitive to low prob tokens/sentences</td>
<td></td>
</tr>
</tbody>
</table>
Generalization of n-grams

- Not all n-grams will be observed in training data
- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*
  - Test set: *Shakespeare*
- \[ P(\text{affray} \mid \text{voice doth us}) = 0 \quad \Rightarrow \quad P(\text{test corpus}) = 0 \]
- Undefined perplexity
Sparsity in language

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

\[ \text{freq} \propto \frac{1}{\text{rank}} \]

Zipf’s Law
Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - **Additive**: Add a small amount to all probabilities
  - **Discounting**: Redistribute probability mass from observed n-grams to unobserved ones
  - **Back-off**: Use lower order n-grams if higher ones are too sparse
  - **Interpolation**: Use a combination of different granularities of n-grams
Smoothing intuition

When we have sparse statistics:

\[ P(w \mid \text{denied the}) \]
- 3 allegations
- 2 reports
- 1 claims
- 1 request
- 7 total

Steal probability mass to generalize better

\[ P(w \mid \text{denied the}) \]
- 2.5 allegations
- 1.5 reports
- 0.5 claims
- 0.5 request
- 2 other
- 7 total

(Credits: Dan Klein)
Laplace smoothing

- Also known as add-alpha

- Simplest form of smoothing: Just add alpha to all counts and renormalize!

- Max likelihood estimate for bigrams:

\[
P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}
\]

- After smoothing:

\[
P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}
\]
Raw bigram counts
(Berkeley restaurant corpus)

- Out of 9222 sentences

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(Credits: Dan Jurafsky)
Smoothed bigram counts

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>828</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>609</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</td>
<td>7</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Add 1 to all the entries in the matrix

(Credits: Dan Jurafsky)
Smoothed bigram probabilities

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

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

(Credits: Dan Jurafsky)
Problem with Laplace smoothing

Raw counts

\[
C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})
\]
Problem with Laplace smoothing

Raw counts

\[ C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1}) \]

<table>
<thead>
<tr>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Reconstituted counts

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

<table>
<thead>
<tr>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>527</td>
<td>0.64</td>
<td>6.4</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>1.2</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>2.7</td>
<td>2.3</td>
<td>0.78</td>
</tr>
<tr>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>430</td>
<td>1.9</td>
<td>0.63</td>
<td>4.4</td>
<td>133</td>
</tr>
<tr>
<td>0.34</td>
<td>0.34</td>
<td>1</td>
<td>0.34</td>
<td>5.8</td>
<td>1</td>
<td>15</td>
<td>0.34</td>
</tr>
<tr>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Linear Interpolation

\[
\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)
\]

\[
\sum_i \lambda_i = 1
\]

• Use a combination of models to estimate probability

• Strong empirical performance
Choosing lambdas

- First, estimate n-gram prob. on training set
- Then, estimate lambdas (hyperparameters) to maximize probability on the held-out development/validation set
- Use best model from above to evaluate on test set
Can we do better than naive interpolation?

Case 1: C (on the mat) = 10, C(on the cat) = 10, C(on the rat) = 10, C(on the bat) = 10, ...

Case 2: C (on the mat) = 40, C(on the cat) = 5, C (on the rat) = 0, C(on the bat) = 0, ...

Which provides a better trigram estimate for \( P(\text{mat} \mid \text{on the}) \)?

Larger weights (\( \lambda \)) on non-sparse estimates
Average-count (Chen and Goodman, 1996)

\[ P_{\text{interp}}(w_i | w_{i-n+1}^{i-1}) = \]
\[ \lambda_{w_{i-n+1}^{i-1}} P_{\text{ML}}(w_i | w_{i-n+1}^{i-1}) + \]
\[ (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{\text{interp}}(w_i | w_{i-n+2}^{i-1}) \]

• Like simple interpolation, but with more specific lambdas, \( \lambda_{w_{i-n+1}^{i-1}} \)

• Partition \( \lambda_{w_{i-n+1}^{i-1}} \) according to average number of counts per non-zero element:

\[ \frac{c(w_{i-n+1}^{i-1})}{|w_i : c(w_{i-n+1}^{i-1}) > 0|} \]

• Larger \( \lambda_{w_{i-n+1}^{i-1}} \) for denser estimates of n-gram probabilities
Discounting

- Determine some “mass” to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value to discount (usually <1)

\[
P_{\text{abs\_discount}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \quad \text{if } c(w_{i-1}, w_i) > 0
\]

Unigram probabilities

\[
\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \quad \text{for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0
\]
Absolute Discounting

• Define $\text{Count}^*(x) = \text{Count}(x) - 0.5$

• Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(\text{the}) = 10 \times 0.5/48 = 5/48$$

• Divide this mass between words $w$ for which $\text{Count}(\text{the, } w) = 0$

<table>
<thead>
<tr>
<th>$x$</th>
<th>Count($x$)</th>
<th>Count$^*(x)$</th>
<th>$\frac{\text{Count}^*(x)}{\text{Count}(x)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the, dog</td>
<td>15</td>
<td>14.5</td>
<td>14.5/48</td>
</tr>
<tr>
<td>the, woman</td>
<td>11</td>
<td>10.5</td>
<td>10.5/48</td>
</tr>
<tr>
<td>the, man</td>
<td>10</td>
<td>9.5</td>
<td>9.5/48</td>
</tr>
<tr>
<td>the, park</td>
<td>5</td>
<td>4.5</td>
<td>4.5/48</td>
</tr>
<tr>
<td>the, job</td>
<td>2</td>
<td>1.5</td>
<td>1.5/48</td>
</tr>
<tr>
<td>the, telescope</td>
<td>1</td>
<td>0.5</td>
<td>0.5/48</td>
</tr>
<tr>
<td>the, manual</td>
<td>1</td>
<td>0.5</td>
<td>0.5/48</td>
</tr>
<tr>
<td>the, afternoon</td>
<td>1</td>
<td>0.5</td>
<td>0.5/48</td>
</tr>
<tr>
<td>the, country</td>
<td>1</td>
<td>0.5</td>
<td>0.5/48</td>
</tr>
<tr>
<td>the, street</td>
<td>1</td>
<td>0.5</td>
<td>0.5/48</td>
</tr>
</tbody>
</table>
Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

$$
P_{bo}(w_i | w_{i-n+1} \cdots w_{i-1}) = \begin{cases} 
  d_{w_{i-n+1} \cdots w_i} \frac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\
  \alpha_{w_{i-n+1} \cdots w_{i-1}} P_{bo}(w_i | w_{i-n+2} \cdots w_{i-1}) & \text{otherwise}
\end{cases}
$$

(Katz back-off)

- $d =$ amount of discounting

- $\alpha =$ back-off weight
Other language models

• Discriminative models:
  ‣ train n-gram probabilities to directly maximize performance on end task (e.g. as feature weights)

• Parsing-based models
  ‣ handle syntactic/grammatical dependencies

• Topic models

• Neural networks
  We’ll see these later on