

Language models

Feedback from after the last lecture

- Am I doing any MSc projects?
- Can you please go over the last question again from last lecture?

30 second questions about perplexity

Question 3:

- Our LM predicts digits between 0 and 9 as our words with even probability:
 - what is the perplexity if our test-set contains 5 words?

$$\text{PPL} = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_n)}}$$

If $n = 5$, $\text{PPL} = (1 / (p(w) * p(w) * p(w) * p(w) * p(w)))^{(1/5)}$

Assume $p(w) = 1 / 10$

$\text{PPL} = 10$

Outline

1. **Sparsity**
2. Feed-forward neural language models
3. Vanilla RNNs for language modelling
4. Bi-directional RNNs
5. LSTMs and GRUs
6. Revision & little tangent on de-biasing



Nihir will be your lecturer
again on Thursday

Outline



Nihir will be your lecturer
again on Thursday

1. **Sparsity**
2. Feed-forward neural language models
3. Vanilla RNNs for language modelling
4. Bi-directional RNNs
5. LSTMs and GRUs
6. Revision & little tangent on de-biasing

My goal today:

For you to leave the room thinking

“Yeah, Joe taught LSTMs quite well”

Language models: sparsity

- WSJ corpus: built over 10 years ago. What would happen if tested on **today's News** articles?
- What happens with unseen n-grams?
 - E.g. “His Majesty” or “Trussonomics”



Sparsity



<UNK>

Language models: sparsity

- Techniques to mitigate sparsity:
 - Add-1 Smoothing
 - Back-off
 - Interpolation

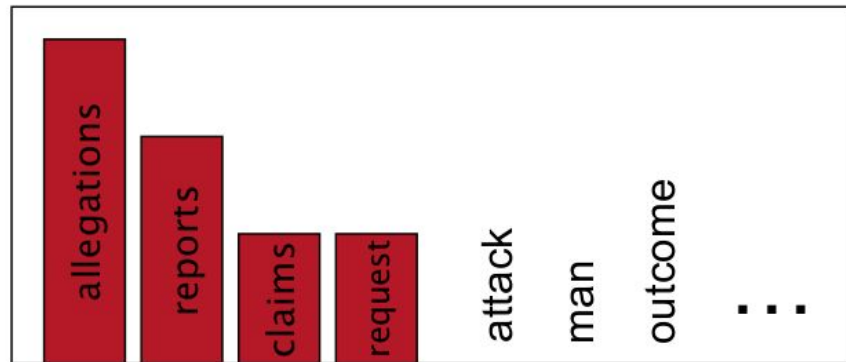
Language models: sparsity

Q4.4 (kind of)
answered



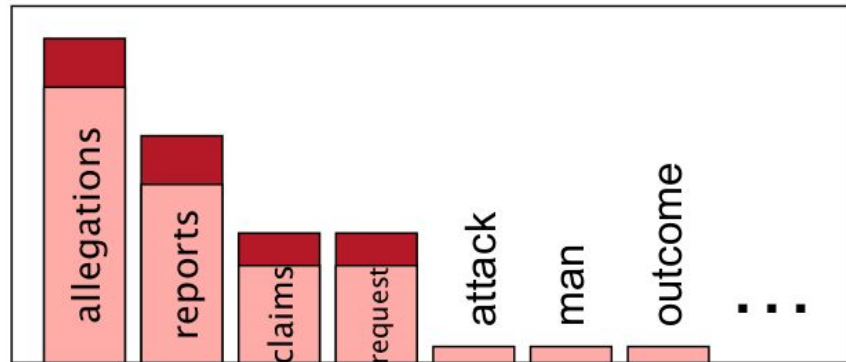
Add-one smoothing

- Given words with sparse statistics, steal probability mass from more frequently words



Bigram example

$$P_{add-1}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$




Language models: sparsity

- 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

Language models: sparsity

- Bigram counts

$$C(\text{"to eat"}) = 686$$


	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

Language models: sparsity

- Bigram **add-1** 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

Language models: sparsity

- Compared to original (not smoothed) version

	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

Language models: sparsity

- Bigram **add-1** smoothed estimates

$$P_{add-1}(w_n|w_{n-1}) = \frac{C(w_{n-1},w_n)+1}{C(w_{n-1})+V}$$

	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

Language models: sparsity

- Compared to original (not smoothed) version

	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

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

Summary of Add-1 smoothing

- Easy to implement
- But takes too much probability mass from more likely occurrences
- Assigns too much probability to unseen events

Summary of Add-1 smoothing

- Easy to implement
- But takes too much probability mass from more likely occurrences
- Assigns too much probability to unseen events
- Could try $+k$ smoothing with a smaller value of k

Back-off smoothing...

Back off smoothing

- If we do not have any occurrences of a 'his royal highness':
 - We could **back-off** and see how many occurrences there are of 'royal highness'

Back off smoothing (“stupid back-off”)

- If we do not have any occurrences of ‘you had covid’:

$$S(w_i | w_{i-2} w_{i-1}) = \begin{cases} \frac{C(w_{i-2} w_{i-1} w_i)}{C(w_{i-2} w_{i-1})} & \text{if } C(w_{i-2} w_{i-1} w_i) > 0 \\ 0.4 \cdot S(w_i | w_{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i | w_{i-1}) = \begin{cases} \frac{C(w_{i-1} w_i)}{C(w_{i-1})} & \text{if } C(w_{i-1} w_i) > 0 \\ 0.4 \cdot S(w_i) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{C(w_i)}{N}$$

N = number of words in the text

Wikipedia tells us:

“This model generally works well in practice, but fails in some circumstances.

For example, suppose that the bigram "a b" and the unigram "c" are very common, but the trigram "a b c" is never seen. Since "a b" and "c" are very common, it may be significant (that is, not due to chance) that "a b c" is never seen.

Perhaps it's not allowed by the rules of the grammar. Instead of assigning a more appropriate value of 0, the method will back off to the bigram and estimate $P(c \mid b)$, which may be too high”

Interpolation

Interpolation

- We combine evidence from different n-grams:

$$\begin{aligned} P_{interp}(w_i | w_{i-2} w_{i-1}) = & \lambda_1 P(w_i | w_{i-2} w_{i-1}) \\ & + \lambda_2 P(w_i | w_{i-1}) \\ & + \lambda_3 P(w_i) \\ & \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{aligned}$$

Questions?

Outline

1. Sparsity
2. **Feed-forward neural language models**
3. Vanilla RNNs for language modelling
4. Bi-directional RNNs
5. LSTMs and GRUs
6. Revision & little tangent on de-biasing



Nihir will be your lecturer
again on Thursday

Neural Language Models

Q5.2 answered

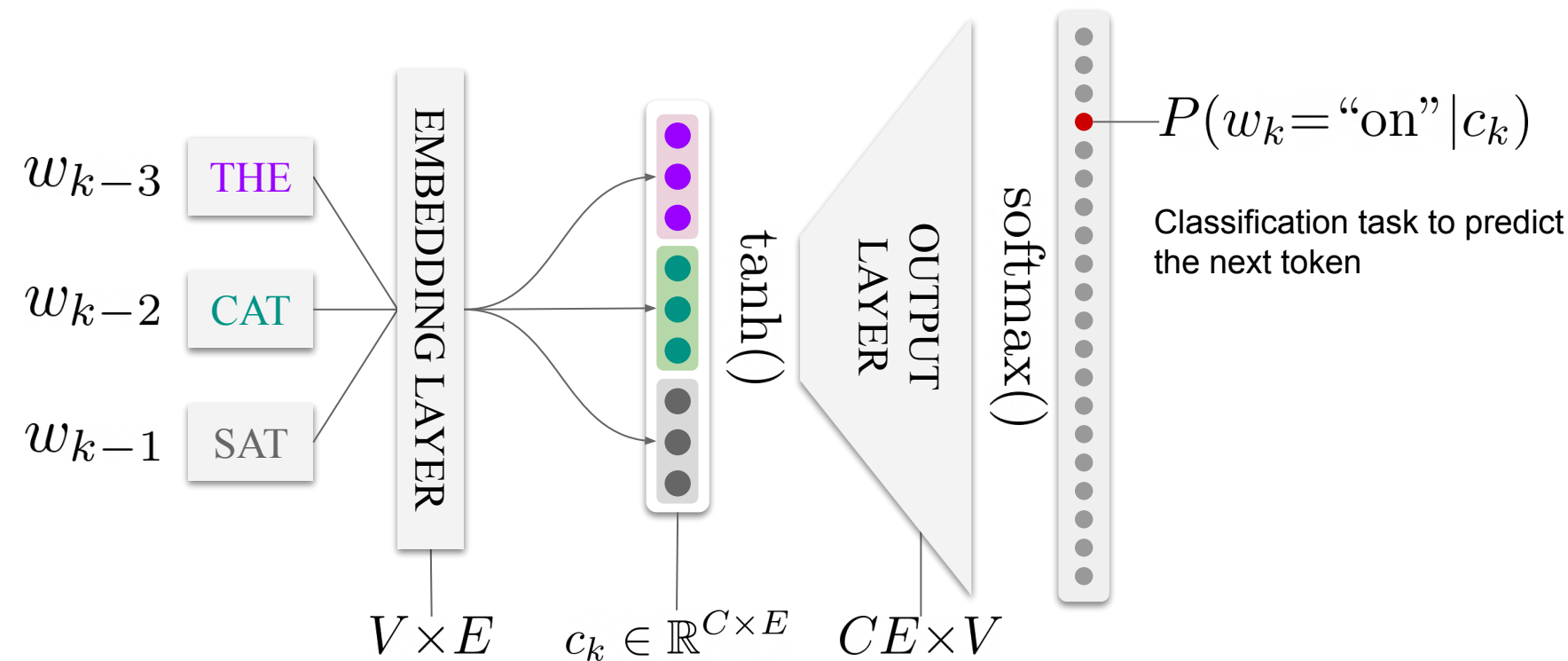


- Neural-based LMs have several improvements:
 - Avoids n-gram sparsity issue
 - Contextual word representations i.e. **embeddings**

4-gram Feed-forward LM (FFLM)

Q5.1 answered

Q5.1 Question in Language Modeling and Classification (2003)	
1. Note: Some of the words in the text are not in the vocabulary. This is a common situation in natural language processing. You should ignore these words when computing the probabilities.	



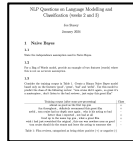
Feed-forward LM (FFLM)

- **First applications of neural networks to LM**
 - Approximates history with last C words
 - C affects model size!
- **Ex:** 4-gram FFLM has a context size of 3
 - Estimates
 - The context is formed by concatenating word embeddings

$$c_k = [\text{EMB}(\text{"the"}); \text{EMB}(\text{"cat"}); \text{EMB}(\text{"sat"})]$$

Feed-forward LM (FFLM)

Q5.3 answered



- **First successful attempt to neural LMs**
 - 10 to 20% perplexity improvement over smoothed 3-gram LM (Bengio et al. 2003)
- **Quickly superseded by more RNN LMs**
 - We are not limited by a fixed size context

Questions?

RNNs for language modelling

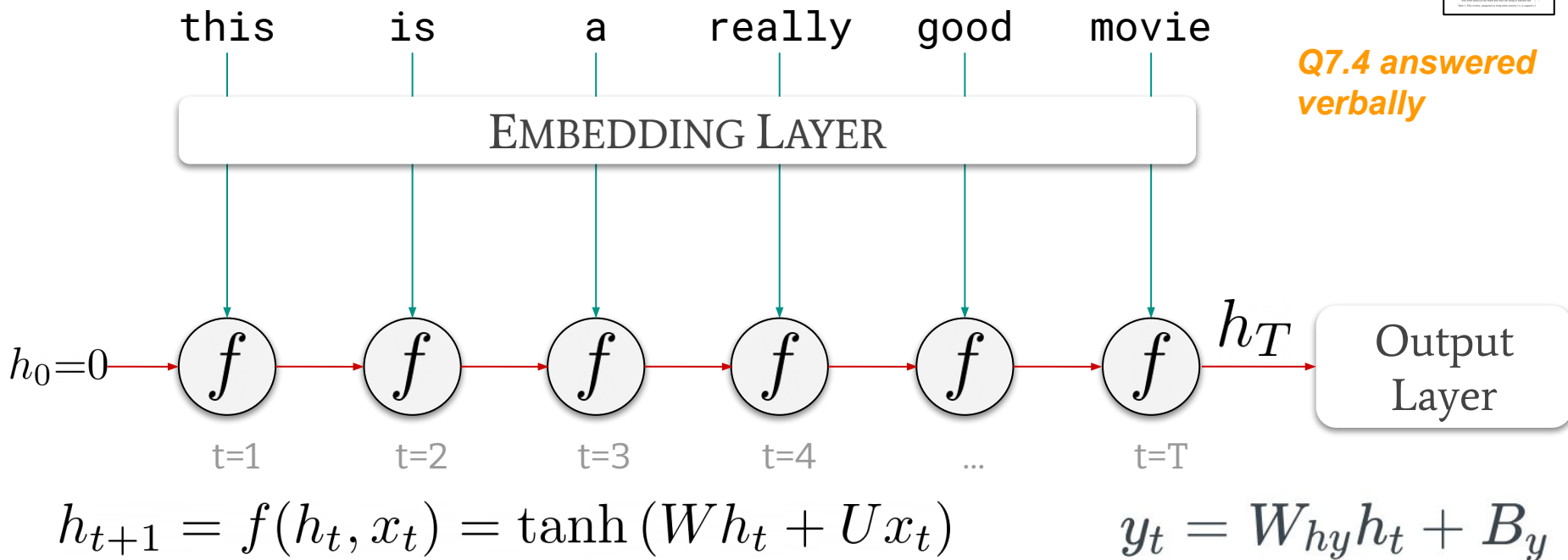
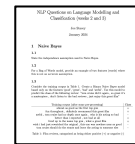
Outline

1. Sparsity
2. Feed-forward neural language models
3. **Vanilla RNNs for language modelling**
4. Bi-directional RNNs
5. LSTMs and GRUs
6. Revision & little tangent on de-biasing

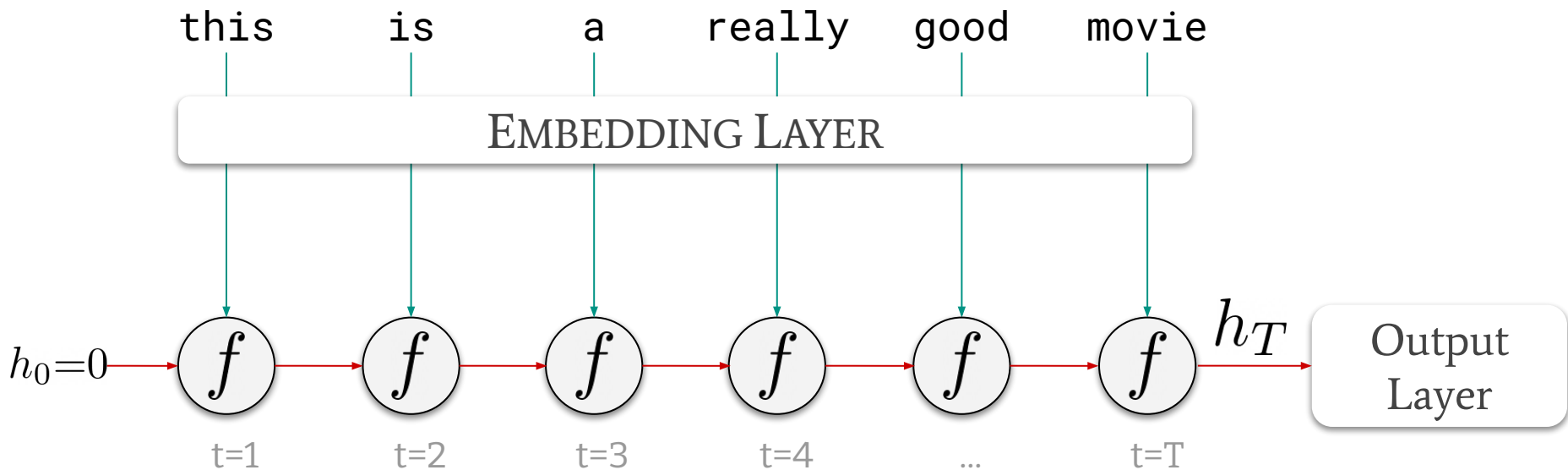


Nihir will be your lecturer
again on Thursday

Recap on vanilla RNNs (classification)



Recap on vanilla RNNs (classification)



$$h_{t+1} = f(h_t, x_t) = \tanh(W h_t + U x_t)$$

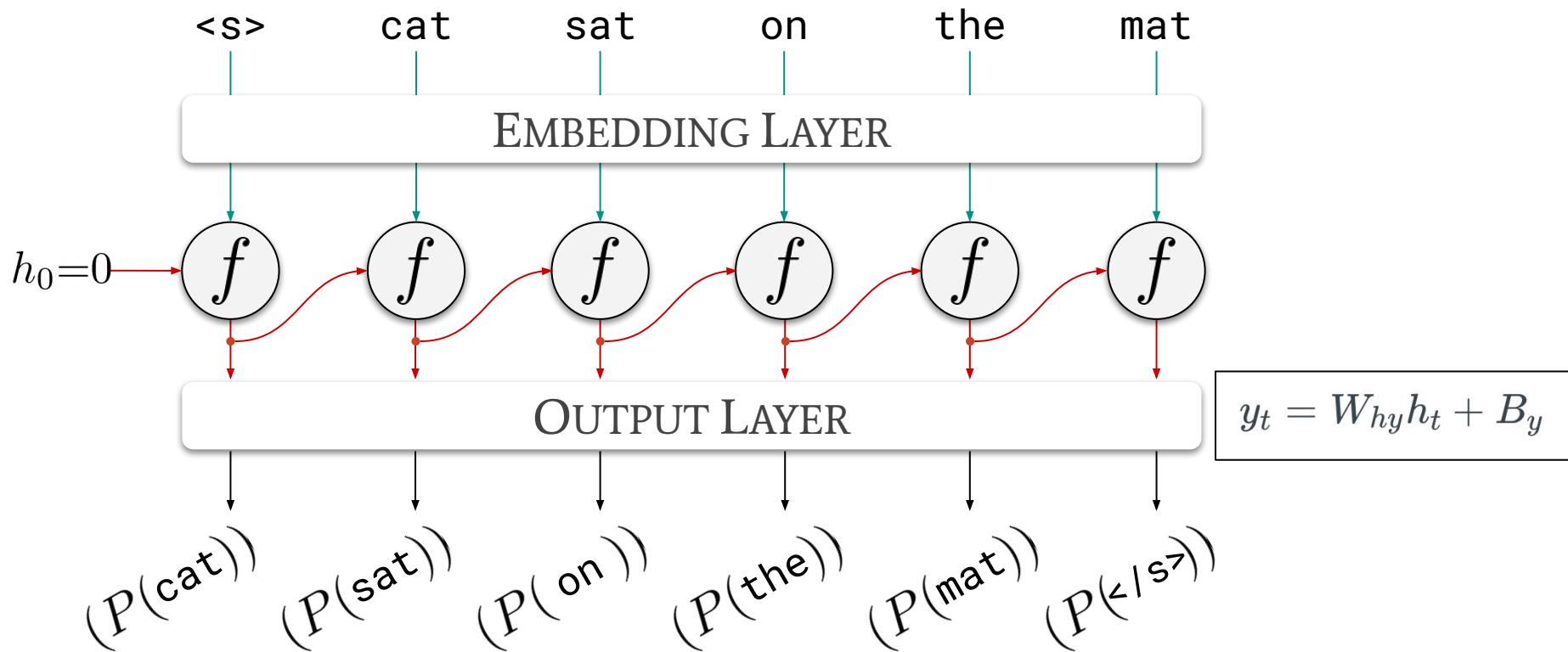
$$y_t = W_{hy} h_t + B_y$$

We “back-propagate through time” (BPTT)

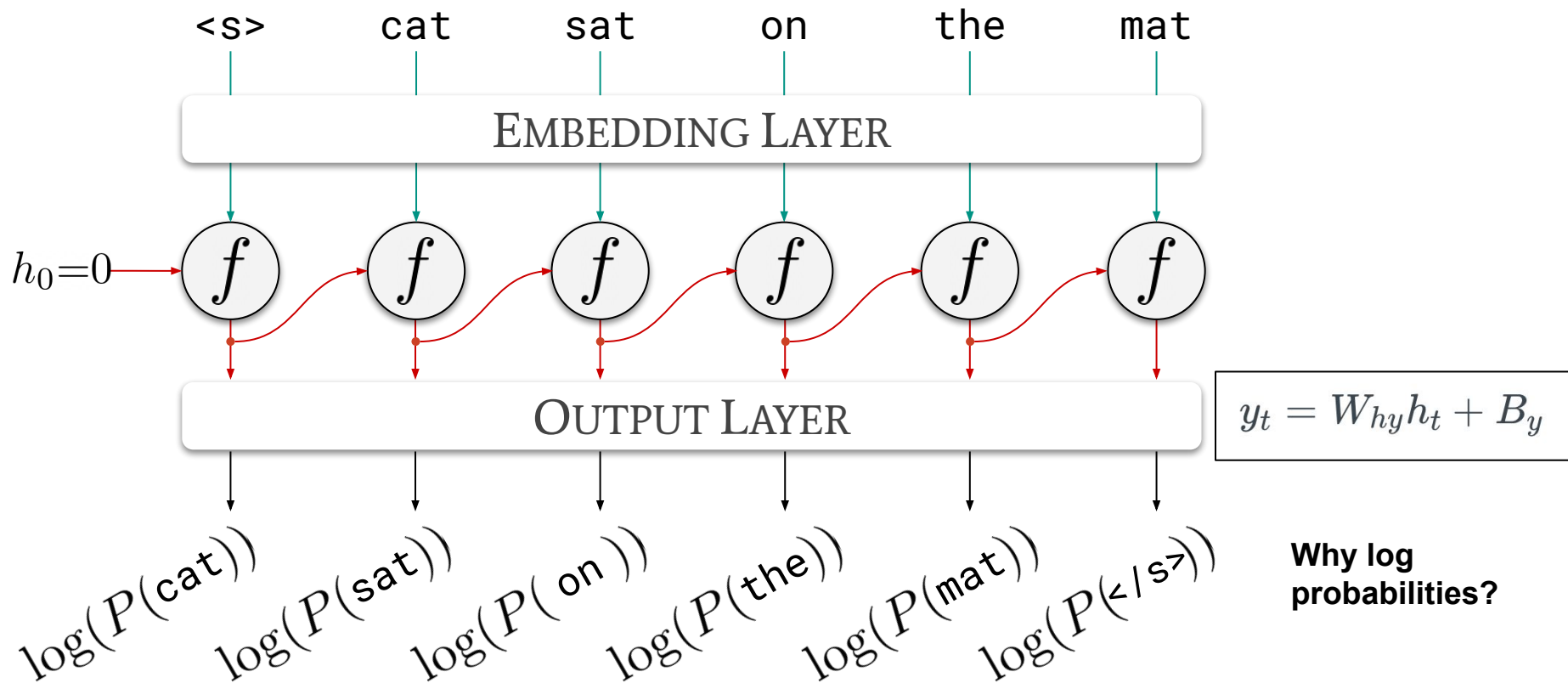
Vanilla RNNs: Many-to-many

- **Every input has a label:**
 - Language modelling -> predicting the next word
- **The LM loss is predicted from the cross-entropy losses from each timestep**

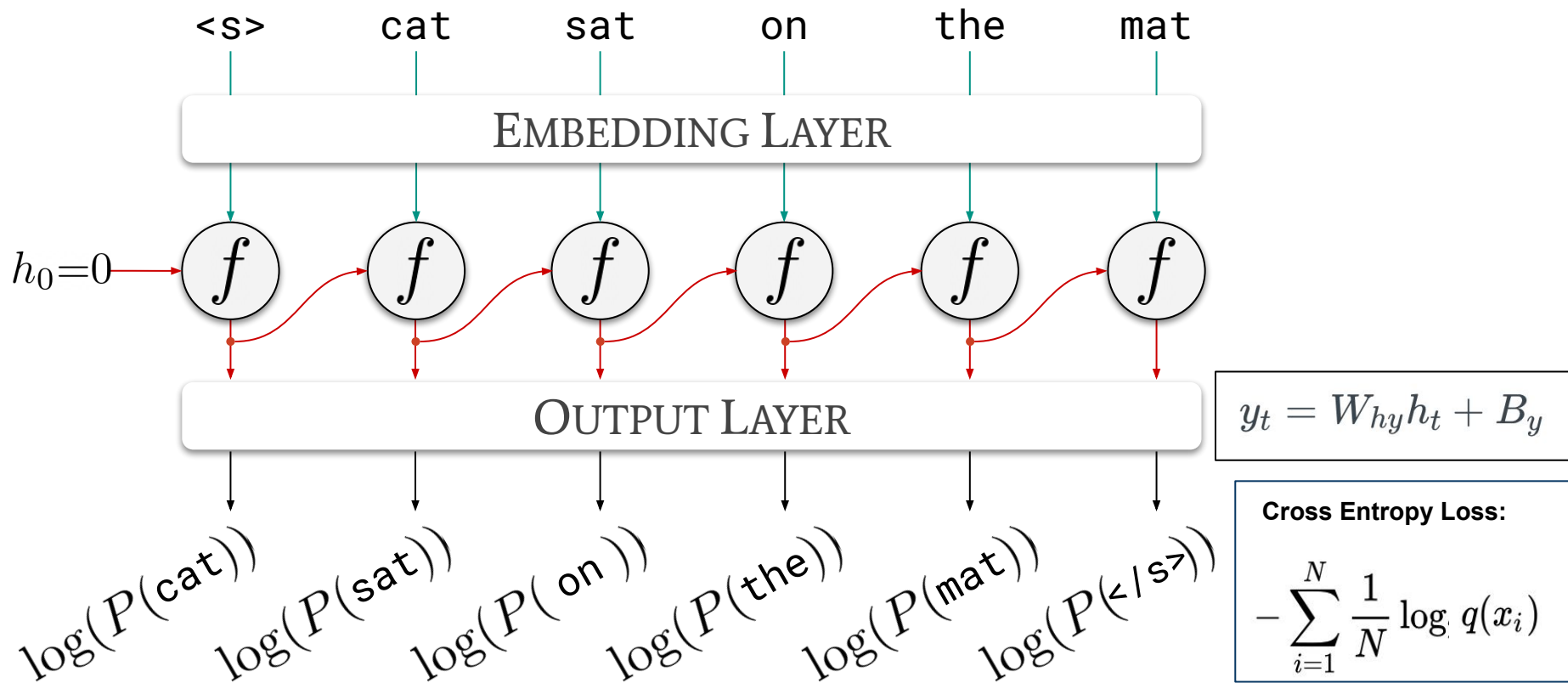
Vanilla RNNs: Many-to-many, during training



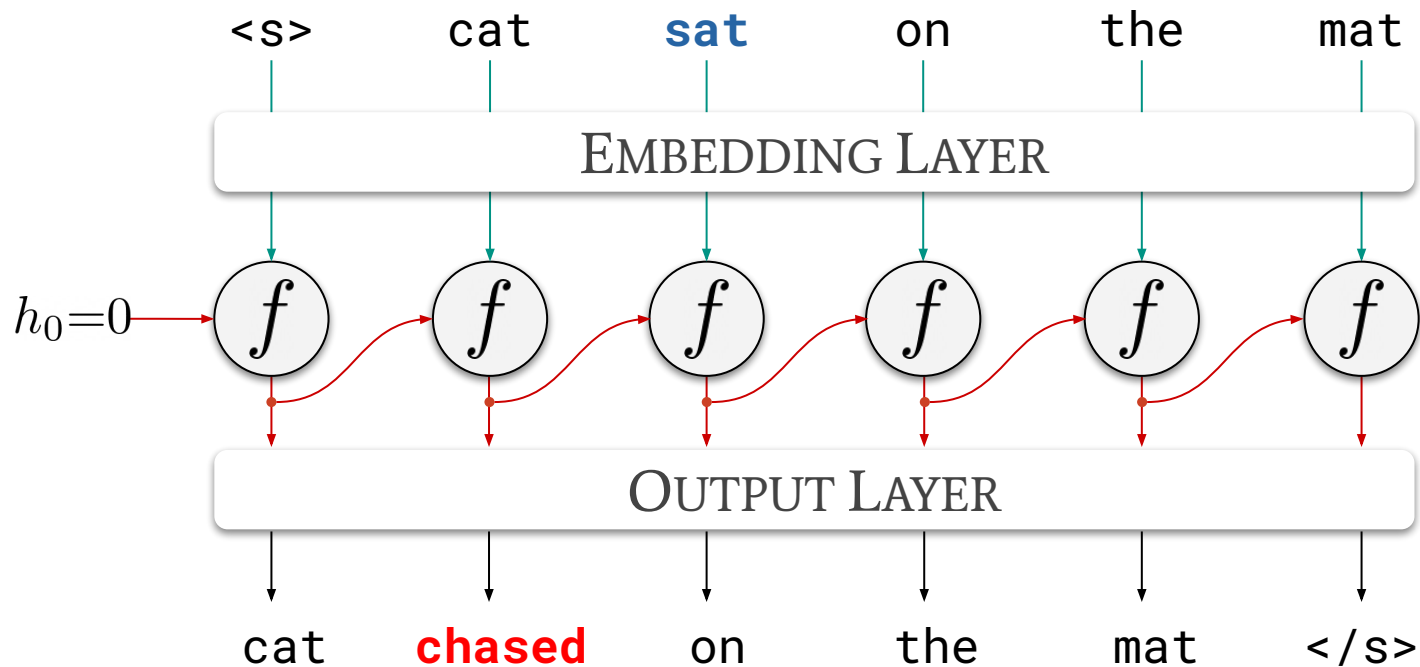
Vanilla RNNs: Many-to-many, during training



Vanilla RNNs: Many-to-many, during training



Vanilla RNNs: Teacher forcing



Note: this is different from what happens at inference

Weight tying, reducing the no. parameters

We can use the same embedding weights in our output layer:

$$e_t = Ex_t$$

$$h_{t+1} = \tanh(Wh_t + Ue_t)$$

$$y_t = \text{softmax}(E^T h_t)$$

The embedding layer E maps our one-hot-label of the input into a word embedding:

- **E has dimensions:** $H \times |V|$
- **E^T therefore has dimensions:** $|V| \times H$

Weight tying, reducing the no. parameters

We can use the same embedding weights in our output layer:

$$e_t = Ex_t$$

$$h_{t+1} = \tanh(Wh_t + Ue_t)$$

$$y_t = \text{softmax}(E^T h_t)$$

The embedding layer E maps our one-hot-label of the input into a word embedding:

- **E has dimensions:** $H \times |V|$
- **E^T therefore has dimensions:** $|V| \times H$

Bonus question:

In the case above, what are the implications for the dimensions of U?

Bi-directional RNNs for classification

Outline

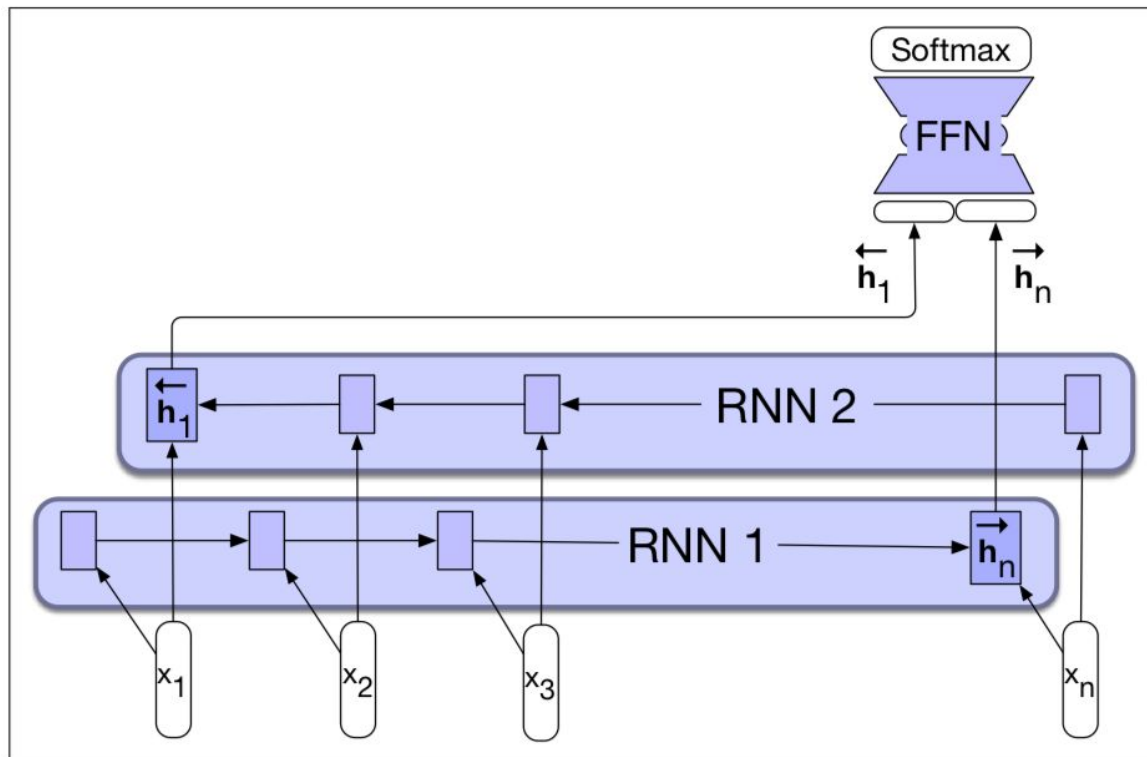
1. Sparsity
2. Feed-forward neural language models
3. Vanilla RNNs for language modelling
4. **Bi-directional RNNs**
5. LSTMs and GRUs
6. Revision & little tangent on de-biasing



Nihir will be your lecturer
again on Thursday

One potential solution for classification:

- We can concatenate the representations at the end of the RNNs for both directions

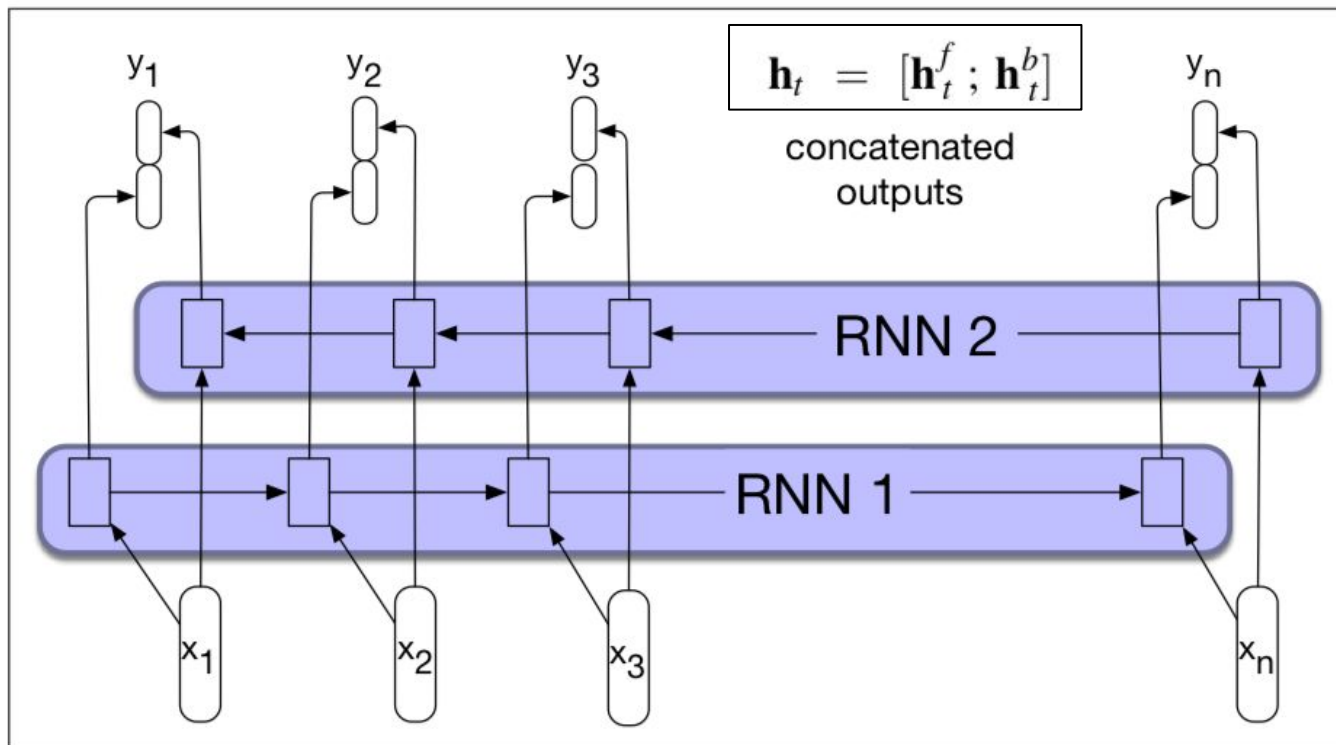


Q8.5 follows from this knowledge



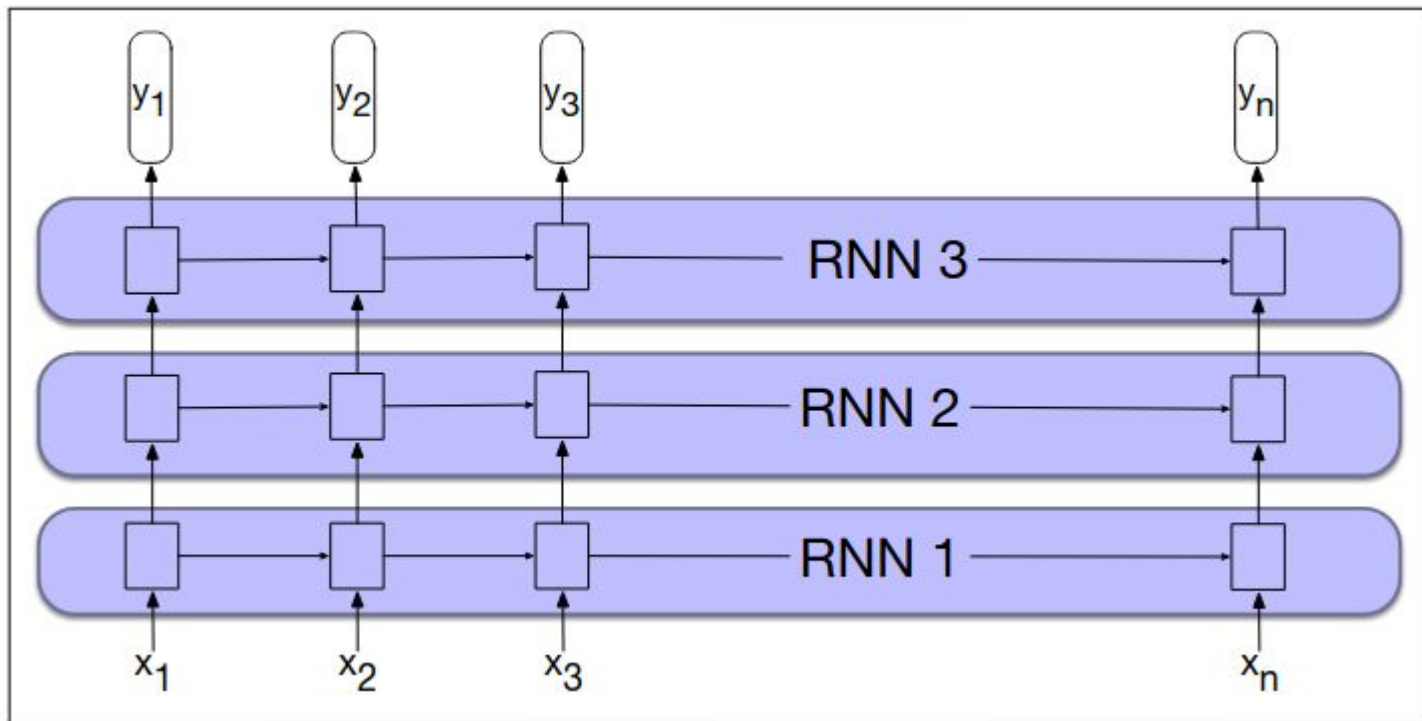
Bi-directional RNNs:

- We could do grammatical error detection in this way
- Could we do sentence classification tasks?



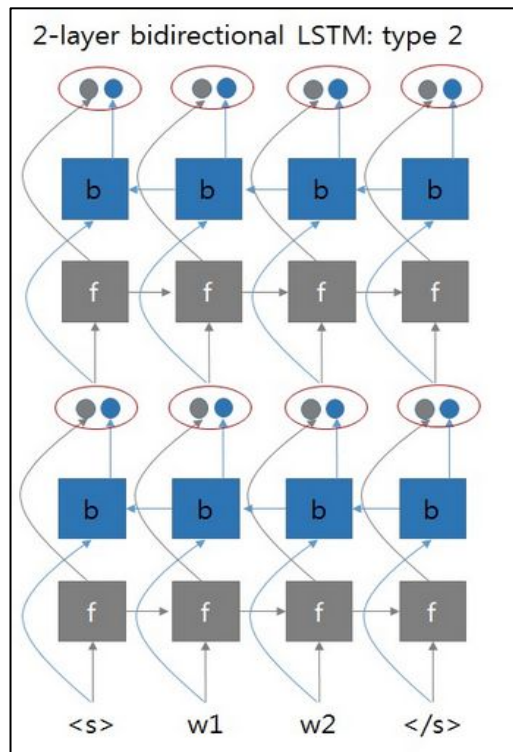
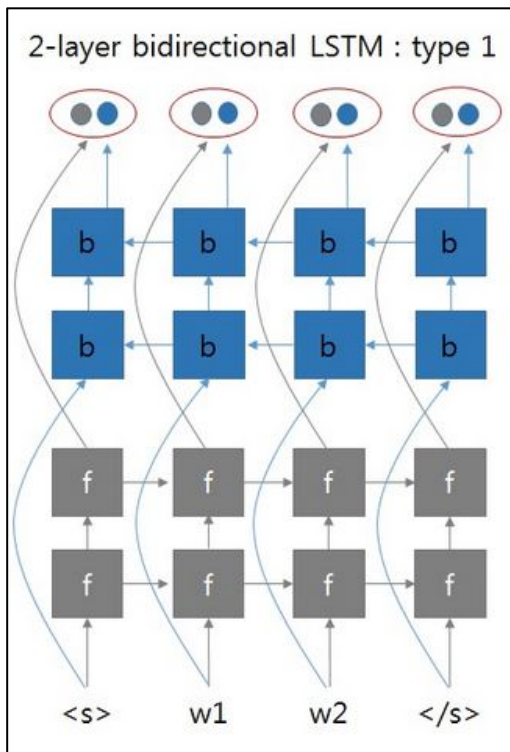
Multi-layered RNNs

- We can feed in our hidden state from an earlier layer into the next layer.



Bidirectional multi-layered RNNs

- There are a few different ways this can be done:



LSTM (and GRUs)

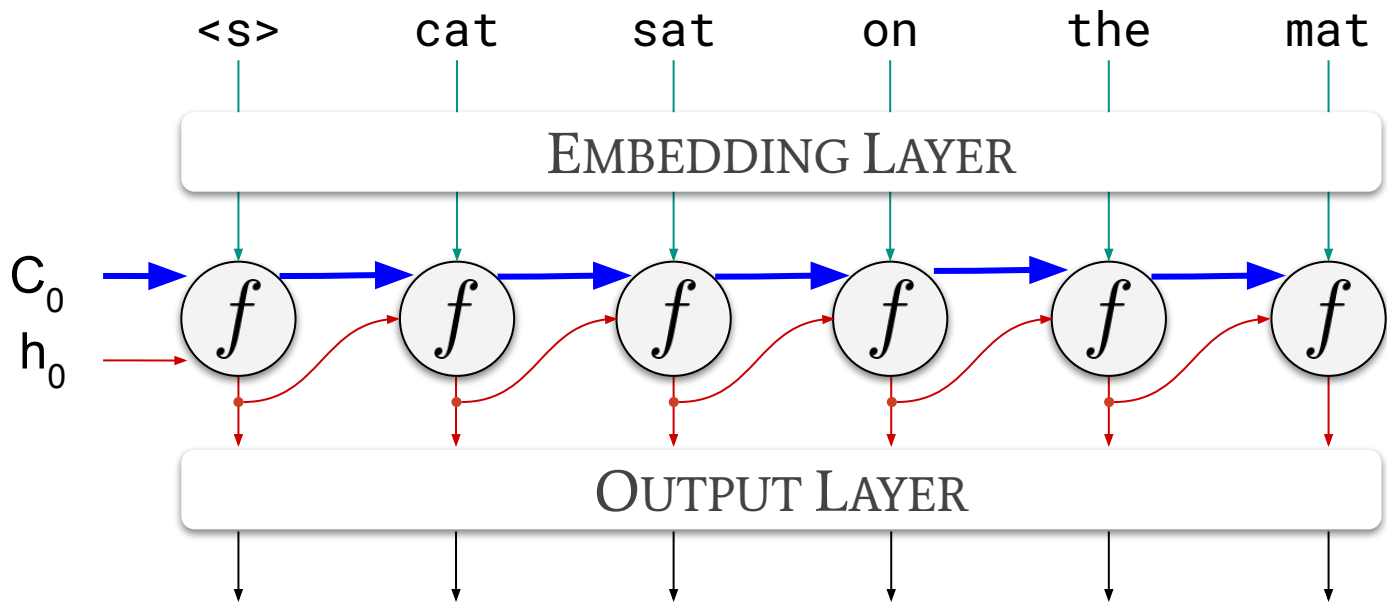
Outline

1. Sparsity
2. Feed-forward neural language models
3. Vanilla RNNs for language modelling
4. Bi-directional RNNs
5. **LSTMs and GRUs**
6. Revision



Nihir will be your lecturer
again on Thursday

Introducing the LSTM



Introducing the cell state

- **Cell states** (C_t) represent 'long term memory'
- **Hidden states** (h_t) is current working memory (e.g. the same as for vanilla RNN)

How LSTMs work:

Q7.1 answered



Full equations:

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f)$$

$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o)$$

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

$$h_t = o_t * \tanh(c_t)$$

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our **cell state**:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

This is our long term memory, as the model may choose to keep \mathbf{c}_t very similar to \mathbf{c}_{t-1}

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our **cell state**:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

Our cell state from the previous time step

*The output from step 1
(what we have from this new timestep)*

This is our long term memory, as the model may choose to keep \mathbf{c}_t very similar to \mathbf{c}_{t-1}

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$



*How much do we use the
long-term memory:
(vector with values between 0,1)*



*How much do we use the output from
step 1:
(vector with values between 0 and 1)*

“Forget gate”

“Input gate”

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

How much do we use the
long-term memory:
(vector with values between 0,1)

How much do we use the output from
step 1:
(vector with values between 0 and 1)

“Forget gate”

“Input gate”

“Forget gate”

$$f_t = \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f)$$

Just a way we can learn what to forget based on content so far (last hidden state), and our new input

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

How much do we use the
long-term memory:
(vector with values between 0,1)

How much do we use the output from
step 1:
(vector with values between 0 and 1)

“Forget gate”

“Input gate”

“Input gate”

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i)$$

We do the same again, allow our model to create a vector of numbers between 0 and 1

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

Step 3: prepare next hidden state:

$$h_t = o_t * \tanh(c_t)$$

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

Step 3: prepare next hidden state:

$$h_t = o_t * \tanh(c_t)$$



An opportunity to scale C_t , multiplying by a vector of values between 0,1

“Output gate”

$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o)$$

We do the same again, allow our model to create a vector of numbers between 0 and 1

“Output gate”

How LSTMs work:

Breaking down an LSTM:

Step 1: we start from what we know already:

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

Step 2: we define our cell state:

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

Step 3: prepare next hidden state:

$$h_t = o_t * \tanh(c_t)$$



The output gate and tanh is what differentiates h_t and c_t

“Output gate”

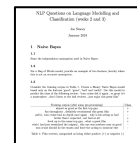
$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o)$$

We do the same again, allow our model to create a vector of numbers between 0 and 1

“Output gate”

How LSTMs work:

Q7.2 answered



Full equations:

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f)$$

$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o)$$

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g)$$

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$

$$h_t = o_t * \tanh(c_t)$$

How LSTMs work:

Q7.2 answered



Full equations:

$$\begin{aligned}i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\c_t &= f_t * c_{(t-1)} + i_t * g_t \\h_t &= o_t * \tanh(c_t)\end{aligned}$$

A fun question:

If we have:

- 100 dimensional hidden states
- 500 dimensional embeddings
- A classification task with 5 classes (no bias term in output layer)

Excl. any embedding layer, how many parameters do we have?

How LSTMs work:

Q7.2 answered



Full equations:

$$\begin{aligned}i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\c_t &= f_t * c_{(t-1)} + i_t * g_t \\h_t &= o_t * \tanh(c_t)\end{aligned}$$

A fun question:

If we have:

- 100 dimensional hidden states
- 500 dimensional embeddings
- A classification task with 5 classes (no bias term in output layer)

Excl. any embedding layer, how many parameters do we have?

$$\begin{aligned}&= (100*500 + 100*100 + 100)*4 \\&+ 100*5 = 24,900\end{aligned}$$

How LSTMs work:

Q7.2 answered



Full equations:

$$\begin{aligned}i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\c_t &= f_t * c_{(t-1)} + i_t * g_t \\h_t &= o_t * \tanh(c_t)\end{aligned}$$

A fun question:

If we have:

- 100 dimensional hidden states
- 500 dimensional embeddings
- A classification task with 5 classes (no bias term in output layer)

Try again, but for a bi-directional LSTM!

How LSTMs work:

Q7.2 answered



Full equations:

$$\begin{aligned}i_t &= \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \\f_t &= \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \\o_t &= \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \\g_t &= \tanh(W_{ig}x_t + W_{hg}h_{t-1} + b_g) \\c_t &= f_t * c_{(t-1)} + i_t * g_t \\h_t &= o_t * \tanh(c_t)\end{aligned}$$

A fun question:

If we have:

- 100 dimensional hidden states
- 500 dimensional embeddings
- A classification task with 5 classes (no bias term in output layer)

Try again, but for a bi-directional LSTM!

$$\begin{aligned}&= (100*500 + 100*100 + 100)*4*2 \\&+ 200*5\end{aligned}$$

Why do LSTMs help with vanishing gradients?

Consider our cell state (long range memory storage):

Q7.3 answered

$$c_t = f_t * c_{(t-1)} + i_t * g_t$$



What helps?

- The gradients through the **cell states** are hard to vanish

Two reasons why:

1. Additive formula means we don't have repeated multiplication of the same matrix (we have a derivative that's more 'well behaved')
2. The forget gate means that our model can learn when to let the gradients vanish, and when to preserve them. This gate can take different values at different time steps.

A simplified architecture (GRU)

How GRUs work:

Our gates: Reset gate (z_t) and update gate (r_t)

$$r_t = \sigma(W_{ir}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{iz}x_t + W_{hz}h_{t-1} + b_z)$$

Our equations:

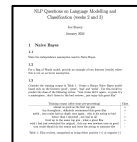
$$g_t = \tanh(W_{ig}x_t + r_t * (W_{hg}h_{t-1} + b_g))$$

$$h_t = (1 - z_t) * g_t + z_t * h_{t-1}$$

**Incorporates x_t with
the last hidden state**

**Based on the previous
hidden state**

Q6.1 answered



How GRUs work:

Our gates: Reset gate (z_t) and update gate (r_t)

$$r_t = \sigma(W_{ir}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{iz}x_t + W_{hz}h_{t-1} + b_z)$$

Our equations:

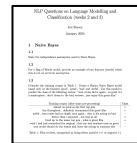
$$g_t = \tanh(W_{ig}x_t + r_t * (W_{hg}h_{t-1} + b_g))$$

$$h_t = (1 - z_t) * g_t + z_t * h_{t-1}$$

**Incorporates x_t with
the last hidden state**

**Based on the previous
hidden state**

Q6.1 answered



Differences

- No output gate...
- $(1 - z_t)$ and z_t ...
- Reset gate...

Outline

1. Sparsity
2. Feed-forward neural language models
3. Vanilla RNNs for language modelling
4. Bi-directional RNNs
5. LSTMs and GRUs
6. **Revision & little tangent on de-biasing**



Nihir will be your lecturer
again on Thursday

Recap - we have covered a lot of things:

- What is NLP classification
- naive bayes
- binary naive bayes
- +1 smoothing
- logistic regression
- deterministic vs generative algorithms
- neural networks for classification
- RNNs
- vanishing gradients
- CNNs
- micro/macro F1 vs accuracy
- What is language modelling
- n-gram modelling
- Perplexity
- cross entropy vs perplexity
- intrinsic vs extrinsic evaluation
- +K smoothing for n-gram modelling
- back-off smoothing
- interpolation smoothing
- feed-forward language models
- RNNs for language modelling
- teacher forcing
- weight tying
- bidirectional RNNs
- multi-layer RNNs
- bidirectional multi-layer RNNs
- LSTMs
- GRUs

Revision: 1) question sheet, 2) know how many parameters for each type of model, 3) quick questions in lectures (e.g. 10-30 second questions), 4) derivations (when examinable)

Please note, the debiasing material was out of scope. I included one related True/False question in the example sheet, but this wouldn't be in scope for the exam.

Break

Revision time!

Naive Bayes

Question sheet

1 Naive Bayes

1.1

State the independence assumption used in Naive Bayes.

1.2

For a Bag of Words model, provide an example of two features (words) where this is not an accurate assumption.

1.3

Consider the training corpus in Table 1. Create a Binary Naive Bayes model based only on the features 'good', 'great', 'bad' and 'awful'. Use this model to

.....

Question sheet

1 Naive Bayes

1.1

State the independence assumption used in Naive Bayes.

1.2 *Answer:* $P(x_1, \dots, x_n|y) = P(x_1|y) \times P(x_2|y) \times \dots \times P(x_n|y)$

For a Bag of Words model, provide an example of two features (words) where this is not an accurate assumption.

1.3

Consider the training corpus in Table 1. Create a Binary Naive Bayes model based only on the features 'good', 'great', 'bad' and 'awful'. Use this model to

.....

Question sheet

1 Naive Bayes

1.1

State the independence assumption used in Naive Bayes.

1.2 Answer: $P(x_1, \dots, x_n|y) = P(x_1|y) \times P(x_2|y) \times \dots \times P(x_n|y)$

For a Bag of Words model, provide an example of two features (words) where this is not an accurate assumption. Answer: Any two highly correlated features (given the class). So any x_1, x_2 where $P(x_1, x_2|y)$ is not equal to $P(x_1|y) \times P(x_2|y)$

1.3

Consider the training corpus in Table 1. Create a Binary Naive Bayes model based only on the features 'good', 'great', 'bad' and 'awful'. Use this model to

Question sheet

1 Naive Bayes

1.1

State the independence assumption used in Naive Bayes.

1.2 Answer: $P(x_1, \dots, x_n|y) = P(x_1|y) \times P(x_2|y) \times \dots \times P(x_n|y)$

For a Bag of Words model, provide an example of two features (words) where this is not an accurate assumption. Answer: Any two highly correlated features (given the class). So any x_1, x_2 where $P(x_1, x_2|y)$ is not equal to $P(x_1|y) \times P(x_2|y)$

1.3 For example, in a sentiment analysis task with movie reviews, the words 'mind blowing' are likely to be correlated for the positive class.

Consider the training corpus in Table 1. Create a Binary Naive Bayes model based only on the features 'good', 'great', 'bad' and 'awful'. Use this model to

Logistic regression

Question sheet

2 Logistic Regression

2.1

What is one limitation of a Naive Bayes model that logistic regression helps to overcome.

2.4

How many parameters do we have for a 3-class logistic regression model with 10 different Bag of Words input features?

Question sheet

2 Logistic Regression

2.1

What is one limitation of a Naive Bayes model that logistic regression helps to overcome.

Answer: Logistic regression is better at handling highly correlated features, as in logistic regression the model can learn to give a lower weighting to one of the highly correlated features.

2.4

How many parameters do we have for a 3-class logistic regression model with 10 different Bag of Words input features?

Logistic Regression with multiple classes

Sentiment analysis with 3 classes: +, - and neutral:

Input features:

x_1	count of bad	1
x_2	count of good	1
x_3	count of and	2

Logistic Regression

Weights are learnt per class

$$w_+ = \begin{bmatrix} 0.0 \\ 1.9 \\ 0.0 \end{bmatrix}$$

$$w_- = \begin{bmatrix} 1.5 \\ 0.4 \\ 0.0 \end{bmatrix}$$

$$w_n = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Logistic Regression

$$z_1 = ([1.0, 1.0, 2.0] \cdot [0.0, 1.9, 0.0]) + 0.1$$

$$z_2 = ([1.0, 1.0, 2.0] \cdot [1.5, 0.4, 0.0]) - 0.9$$

$$z_3 = ([1.0, 1.0, 2.0] \cdot [0.0, 0.0, 0.0]) + 0.1$$

**Q2.4 answered
verbally**



Question sheet

2 Logistic Regression

2.1

What is one limitation of a Naive Bayes model that logistic regression helps to overcome.

Answer: Logistic regression is better at handling highly correlated features, as in logistic regression the model can learn to give a lower weighting to one of the highly correlated features.

2.4

How many parameters do we have for a 3-class logistic regression model with 10 different Bag of Words input features?

Answer: For our weights W we have: $3 \times 10 = 30$ parameters. We also have 3 bias term parameters.

So in total, we have 33.

Feed-forward neural language model

Feed-forward LM (FFLM)

5.1

Consider a feed-forward language model using 4 words of context, 200-dimensional word embeddings and a vocabulary of 20,000 words. Assume no bias terms in the model.

How many parameters are there in the model? Include parameters in the embedding layer.

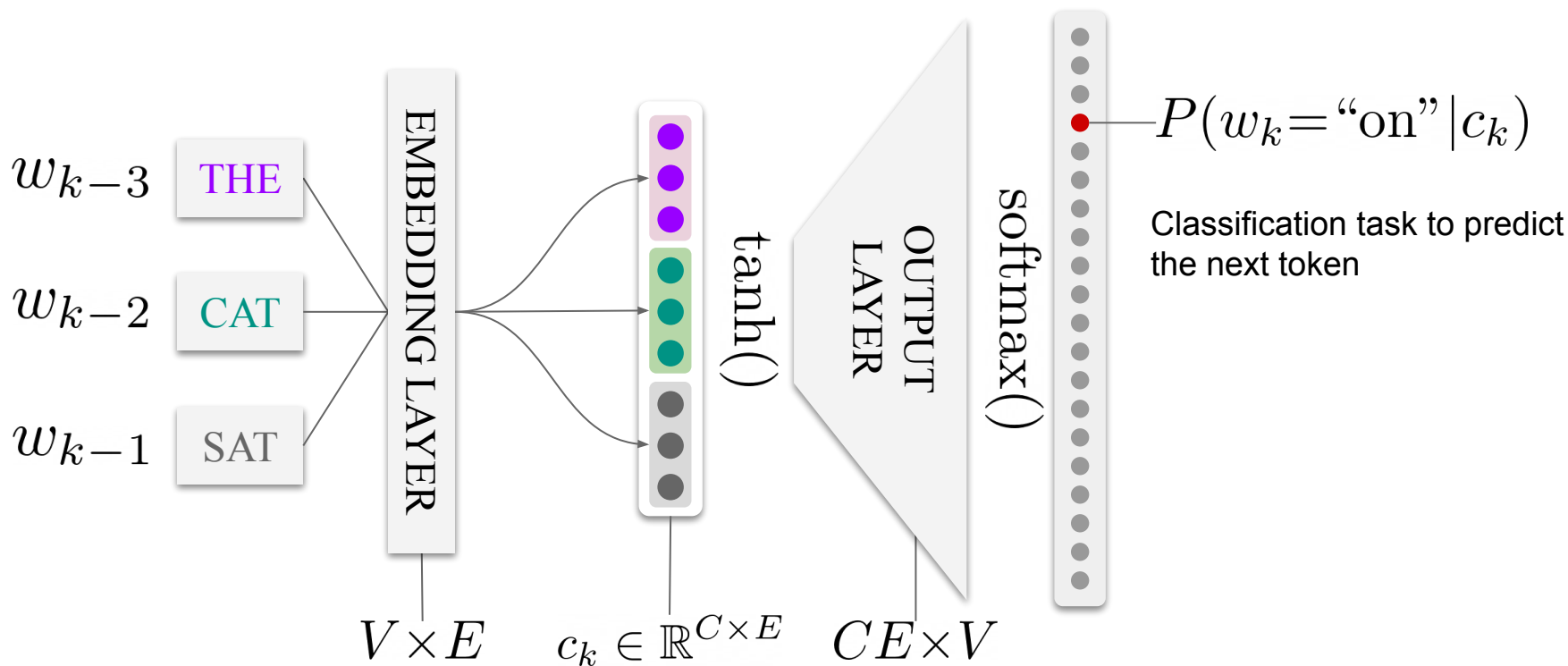
5.2

Name two advantages that a feed-forward language model has over an N-gram model.

5.3

With a large training corpus, would you expect a feed-forward language model to outperform an n-gram model?

4-gram Feed-forward LM (FFLM)



Feed-forward LM (FFLM)

5.1

Consider a feed-forward language model using 4 words of context, 200-dimensional word embeddings and a vocabulary of 20,000 words. Assume no bias terms in the model.

How many parameters are there in the model? Include parameters in the embedding layer.

Answer: Embedding layer ($20,000 \times 200$),

Output layer ($4 \times 200 \times 20,000$)

Total: $4,000,000 + 16,000,000 = 20,000,000$

5.2

Name two advantages that a feed-forward language model has over an N-gram model.

5.3

With a large training corpus, would you expect a feed-forward language model to outperform an n-gram model?

Feed-forward LM (FFLM)

5.1

Consider a feed-forward language model using 4 words of context, 200-dimensional word embeddings and a vocabulary of 20,000 words. Assume no bias terms in the model.

How many parameters are there in the model? Include parameters in the embedding layer.

Answer: Embedding layer ($20,000 \times 200$),

Output layer ($4 \times 200 \times 20,000$)

Total: $4,000,000 + 16,000,000 = 20,000,000$

5.2

Name two advantages that a feed-forward language model has over an N-gram model.

Answer: 1) The model can make use of pre-trained word embeddings, and 2) the feed-forward language model avoids the sparsity issues that we have with N-gram models

5.3

With a large training corpus, would you expect a feed-forward language model to outperform an n-gram model?

Feed-forward LM (FFLM)

- **First successful attempt to neural LMs**
 - Simple and elegant NN perspective to n-gram LMs
 - 10 to 20% perplexity improvement over smoothed 3-gram LM (Bengio et al. 2003)
- **Quickly superseded by more expressive RNN LMs**

We are not limited by a fixed size context

Feed-forward LM (FFLM)

5.1

Consider a feed-forward language model using 4 words of context, 200-dimensional word embeddings and a vocabulary of 20,000 words. Assume no bias terms in the model.

How many parameters are there in the model? Include parameters in the embedding layer.

Answer: Embedding layer ($20,000 \times 200$),

Output layer ($4 \times 200 \times 20,000$)

Total: $4,000,000 + 16,000,000 = 20,000,000$

5.2

Name two advantages that a feed-forward language model has over an N-gram model.

Answer: 1) The model can make use of pre-trained word embeddings, and 2) the feed-forward language model avoids the sparsity issues that we have with N-gram models

5.3

With a large training corpus, would you expect a feed-forward language model to outperform an n-gram model?

Answer: Yes (you may see 10-20% improvement over a smoothed 3-gram language model).

N-grams models

N-gram models

	hamish	is	the	sweetest	cat
hamish	0.01	0.4	0.2	0	0
is	0.02	0	0.4	0.08	0.01
the	0.01	0	0	0.45	0.3
sweetest	0.01	0	0.01	0	0.5
cat	0	0.15	0.03	0	0

Table 2: The column on the left hand side are the words in the history, while the top row are the words being predicted. The values in the table are the model probabilities for the words in the top row, given the words in the left hand side column.

4.3

Given the probability table in Table 2, calculate the probability of the sentence “hamish is the sweetest cat”, using a bi-gram model. You can use the information that $P(\text{hamish} | < s >) = 0.1$, and $P(< /s > | \text{cat}) = 0.1$.

N-gram models

	hamish	is	the	sweetest	cat
hamish	0.01	0.4	0.2	0	0
is	0.02	0	0.4	0.08	0.01
the	0.01	0	0	0.45	0.3
sweetest	0.01	0	0.01	0	0.5
cat	0	0.15	0.03	0	0

Table 2: The column on the left hand side are the words in the history, while the top row are the words being predicted. The values in the table are the model probabilities for the words in the top row, given the words in the left hand side column.

4.3

Given the probability table in Table 2, calculate the probability of the sentence “hamish is the sweetest cat”, using a bi-gram model. You can use the information that $P(\text{hamish} | < s >) = 0.1$, and $P(< /s > | \text{cat}) = 0.1$.

$$\begin{aligned}\text{Answer: Probability} &= P(\text{hamish} | < s >) \times P(\text{is} | \text{hamish}) \times P(\text{the} | \text{is}) \times P(\text{sweetest} | \text{the}) \times \\ &P(\text{cat} | \text{sweetest}) \times P(< /s > | \text{cat}) \\ &= 0.1 \times 0.4 \times 0.4 \times 0.45 \times 0.5 \times 0.1 \\ &= \frac{1}{10} \times \frac{4}{10} \times \frac{4}{10} \times \frac{4.5}{10} \times \frac{5}{10} \times \frac{1}{10} \\ &= 360 * 10^{-6} \\ &= 3.6 * 10^{-4}\end{aligned}$$

Final tangent on debiasing
(section not examinable)

Debiasing

- You will find results something like

	In-distribution test set	out-of-distribution test set
Normal training	Model performs great	Not so great
Training with PoE	Little bit worse than normal training	Better than normal training

Debiasing

Some notation for the next slide:

- I will refer to the probabilities from our bias model as (b_i) ...
 - I may informally call these the “bias probabilities”
- The probabilities from our main model (p_i)

Debiasing

b_i here refers to the predicted probability for the correct class from the biased model...

We can weight the loss of examples based on the performance of our biased model:

- When training the robust model, multiply the loss by:
 $1 - b_i$ (b_i refers to the bias model probabilities for the correct class)

Clark et al (2019):

“Don’t Take the Easy Way Out Ensemble Based Methods for Avoiding Known Dataset Biases”

In this implementation, the loss is normalized across each minibatch so the total minibatch loss remains the same

**Let's try now more approach:
Product of Experts**

It looks simple, but it's really interesting

Debiasing

Now, b_i refers to the predicted probability distribution from the biased model...

We combine together:

- The probabilities of each class given our bias (b_i)...
 - our “bias probabilities”
- ... and the probabilities from our model (p_i)

When training your robust model:

$$\hat{p}_i = \text{softmax}(\log(p_i) + \log(b_i))$$

Debiasing

Now, b_i refers to the predicted probability distribution from the biased model for obs i...

E.g.

- If the label is (0, 0, 1), and the bias model gives (0, 0.1, 0.9)
 - Our main model predictions have a **smaller loss**
(less learning from biased example)
- If the label is (0, 0, 1), and the bias model gives (0, 0.9, 0.1)
 - Our main model predictions have a **greater loss**
(more learning from example not following the bias)

When training your robust model:

$$\hat{p}_i = \text{softmax}(\log(p_i) + \log(b_i))$$

Debiasing

We combine together:

- The probabilities of each class given our bias (b_i)...
- ... and the probabilities from our model (p_i)

$$\hat{p}_i = \text{softmax}(\log(p_i) + \log(b_i))$$

```
class BiasProduct(ClfDebiasLossFunction):  
    def forward(self, hidden, logits, bias, labels):  
        logits = logits.float() # In case we were in fp16 mode  
        logits = F.log_softmax(logits, 1)  
        return F.cross_entropy(logits+bias.float(), labels)
```

Code from Clark et al (2019):

“Don’t Take the Easy Way Out Ensemble Based Methods for Avoiding Known Dataset Biases”

Debiasing

Creating 'biased' probabilities:

1. Hand calculations for the probability of each class given a bias feature
2. Use a model not powerful enough for the task (e.g. BoW model, or TinyBERT)
3. Use incomplete information (e.g. only the hypothesis in NLI)
4. Train a classifier on a very small number of observations

Wrap-up