AdvNLP/E Seminar 3

Language Modelling 2

Dr Julie Weeds, Spring 2026

Warm-up

- These words only occur once in the Gutenberg training data
- What information can we use to make more reliable estimates about what words might appear before or after them?

Rare word
inanity
picion
Entirely
boyishness
reawakened
grey-green
stumbles
steamy

Group discussions

5 minutes to discuss the warm-up question

Language Models

PREVIOUSLY

- N-gram language modelling
  - Markov assumptions
  - perplexity and evaluation
  - smoothing

THIS TIME

- Neural language models
  - feed forward
  - recurrent
  - long-short term memory
  - convolutional

This may all be new to you

- Who has studied machine learning and neural networks previously?
- Who is studying machine learning (and neural networks) currently? How much have you covered on:
  - Neural networks
  - Gradient descent / back propagation
  - Activation functions
  - Loss functions

Questions from Lecture?

Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

3.1.1 Training neural networks

- For a **hard classification** task (i.e., one where there is only one correct answer)
  - **cross-entropy loss** or **negative log-likelihood loss** is simply the log probability, computed by the network, of the correct class. Let $\hat{y}$ be the output vector from the network and y be a k-dimensional one-hot vector encoding the correct answer (i)

$$\mathrm{J} = \mathrm{Loss}(\hat{y}, y) = -\sum_{i=1}^{k} y_i \log \hat{y}_i = -\log \hat{y}_i$$

  - we want to find the parameters / weights which **minimize the loss function**

loss functions are also sometimes referred to as *cost functions* and *objective functions*. Loss or cost function should always refer to penalties which we want to minimize. An objective function is anything we want to optimize on (and could include a reward function we want to maximise)

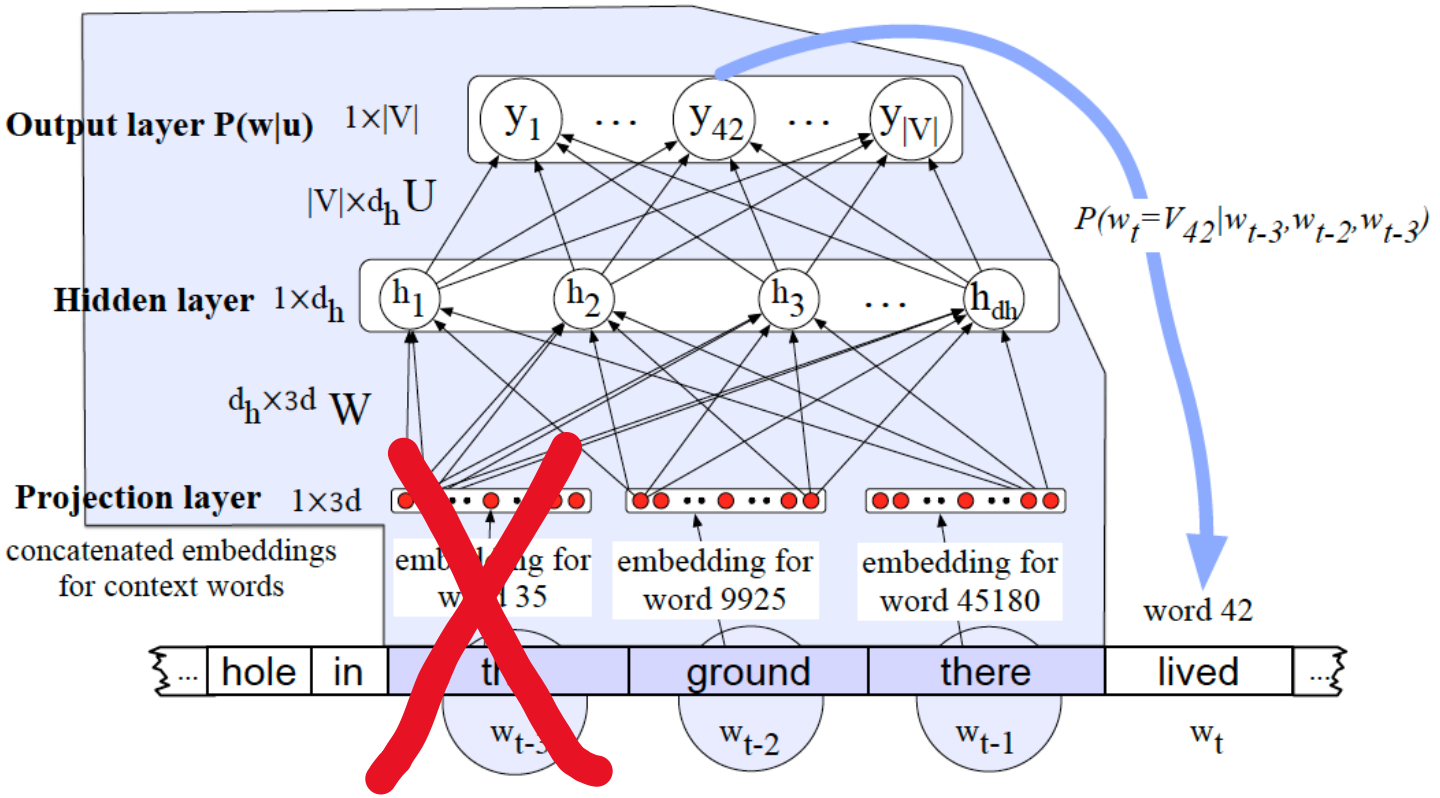
Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

3.1.2 FF-NLMs (Bengio et al. 2003)

Output at time t: a probability distribution over possible next words

Input at time t: a representation of some number of previous words (w_{t-1}, w_{t-2}, etc)

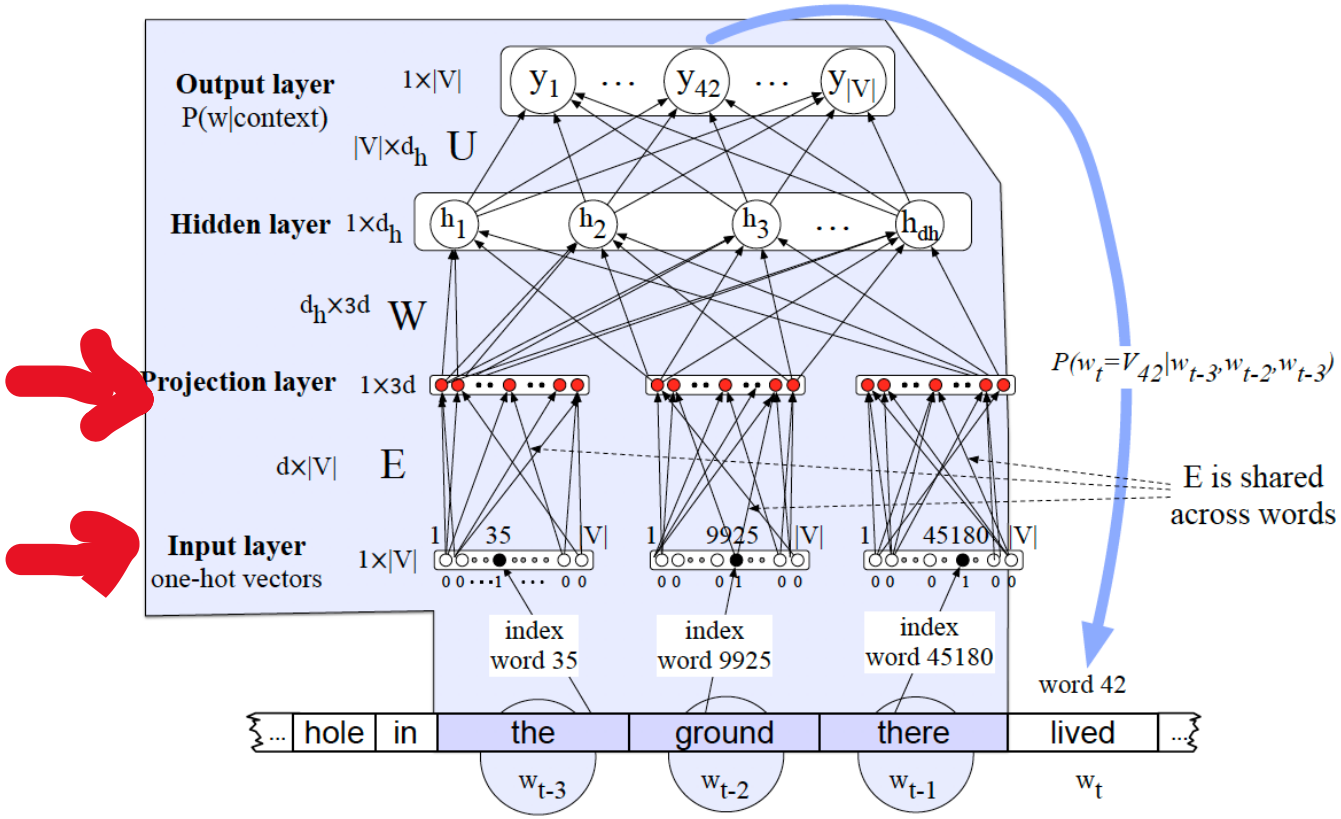


This is a trigram model – how to adapt it to make it a bigram model?

Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

3.2.3 Where do the embeddings come from?

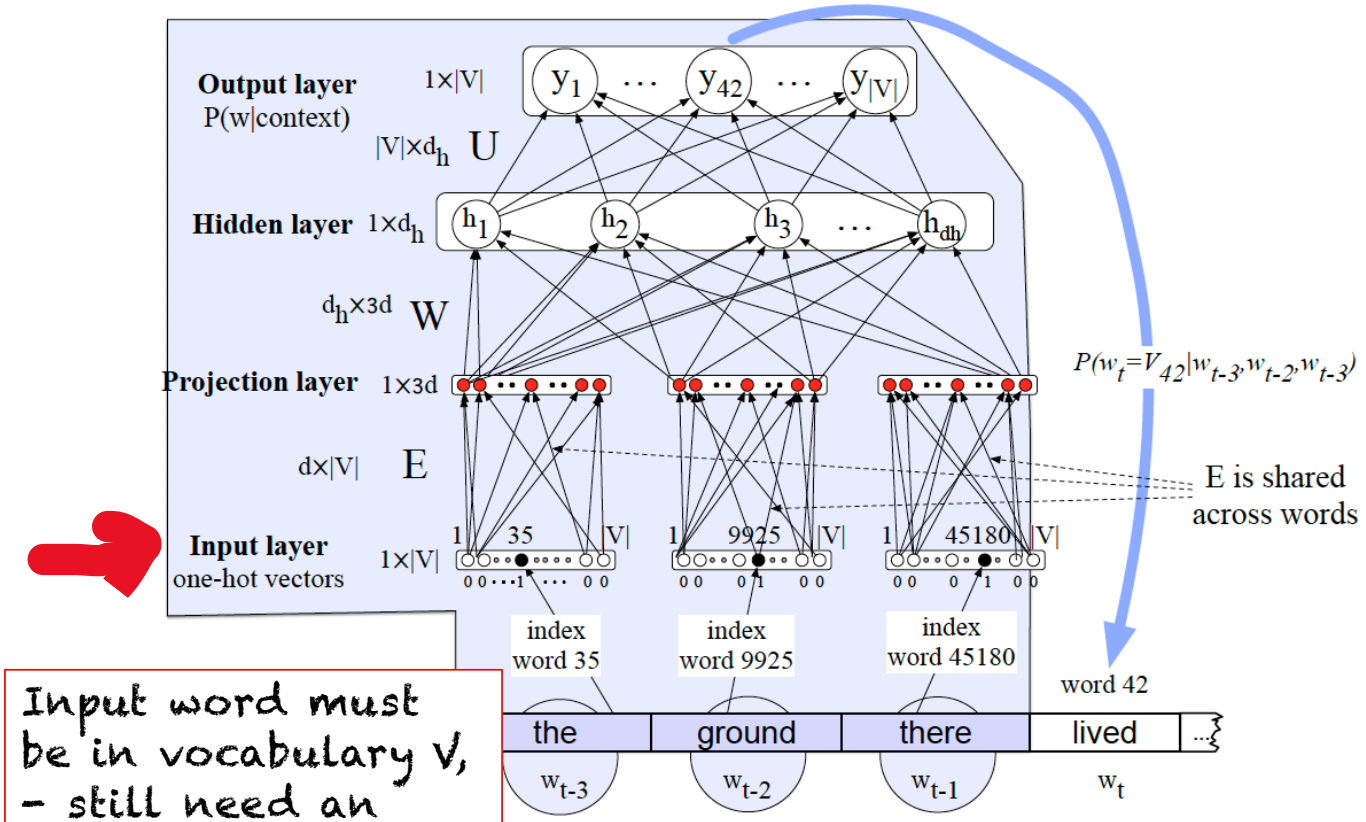


- Embedding weight matrix E learnt at the same time as W and U
  - 3 layer network
  - random initialisation
- Pre-trained embeddings (e.g., LSA)
  - 2 layer network (or 3 layer network with **frozen** embedding matrix)
  - or **tuned** in a 3 layer network

Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

3.2.4 Where do the embeddings come from?



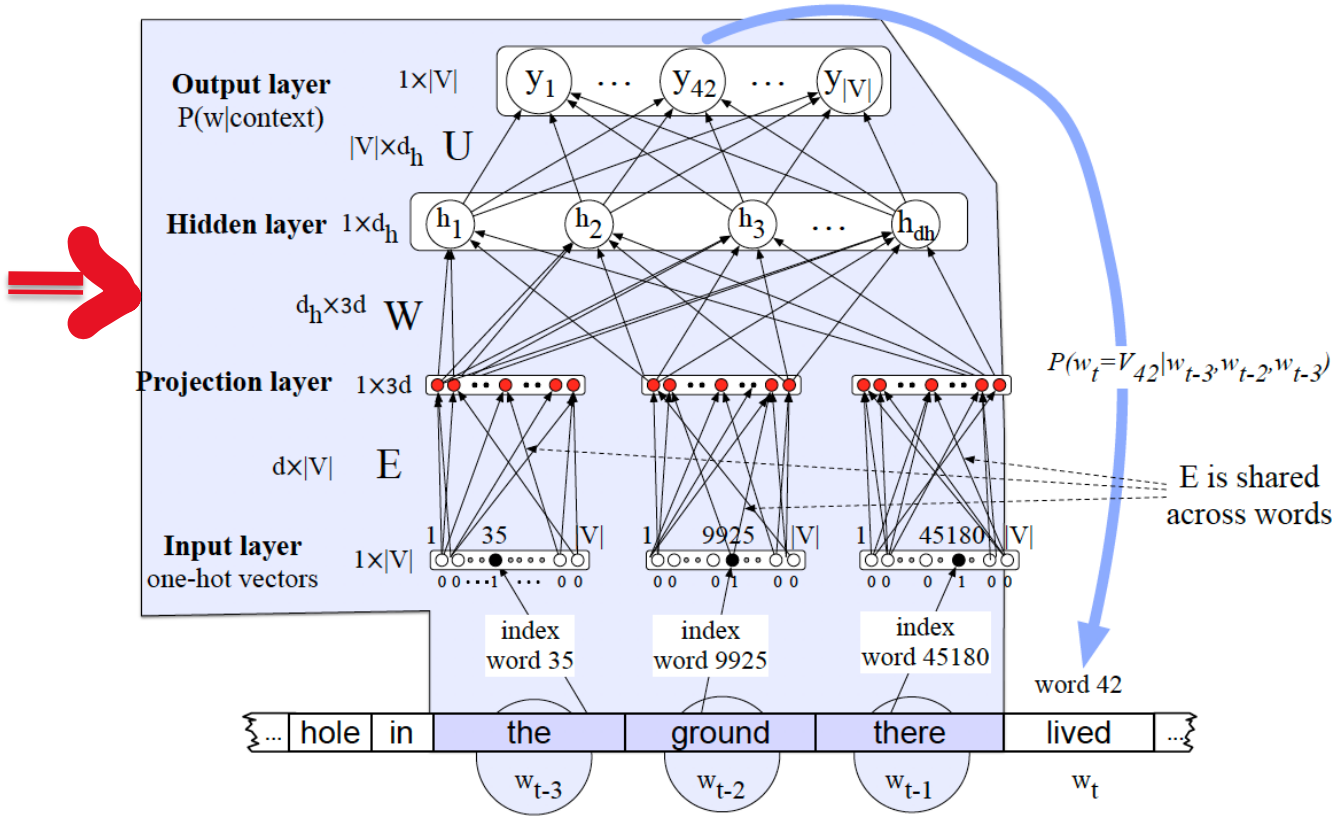
Input word must be in vocabulary V, - still need an <UNK> token

- Embedding weight matrix E learnt at the same time as W and U
  - 3 layer network
  - random initialisation
- Pre-trained embeddings (e.g., LSA)
  - 2 layer network (or 3 layer network with **frozen** embedding matrix)
  - or **tuned** in a 3 layer network

Questions from Lecture 3.1

1. In ML, is a loss function the same as an objective function?
2. Explain how a bigram model of language could be built with a feed-forward neural network?
3. What is the difference between a one-hot encoding of a word and a word embedding?
4. What will a neural language model typically do with OOV words?
5. If a combination of words is seen at test time which has not been seen before, what will happen?

3.2.5 Where do the embeddings come from?



- Embedding weight matrix E learnt at the same time as W and U
  - 3 layer network
  - random initialisation
- Pre-trained embeddings (e.g., LSA)
  - 2 layer network (or 3 layer network with **frozen** embedding matrix)
  - or **tuned** in a 3 layer network

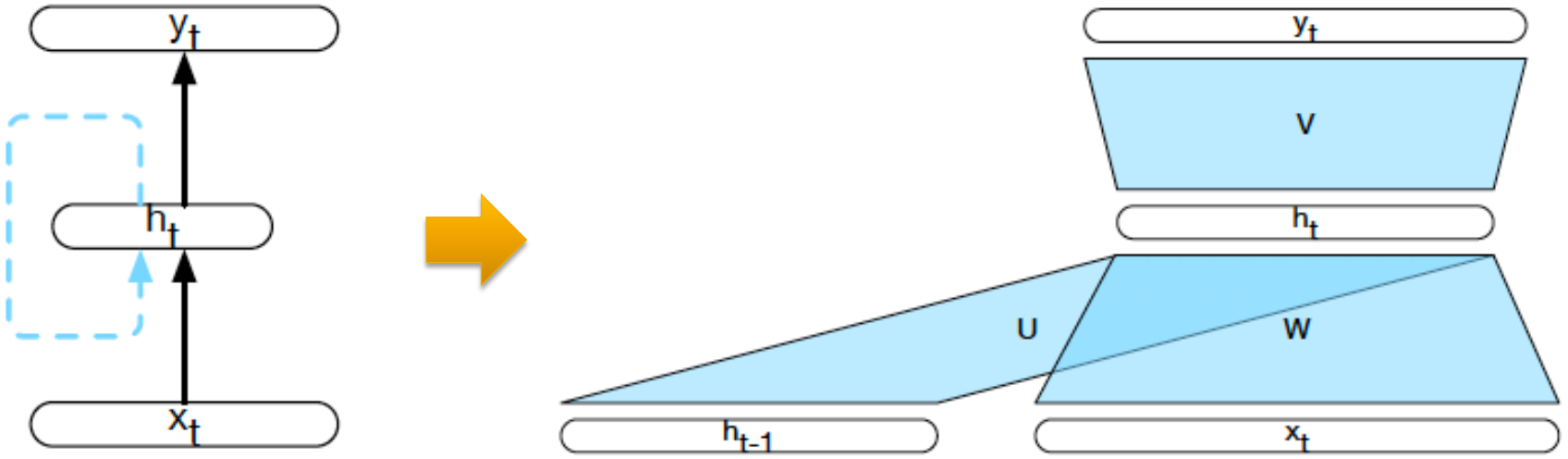
Questions from Lecture 3.2

1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

Questions from Lecture 3.2

1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

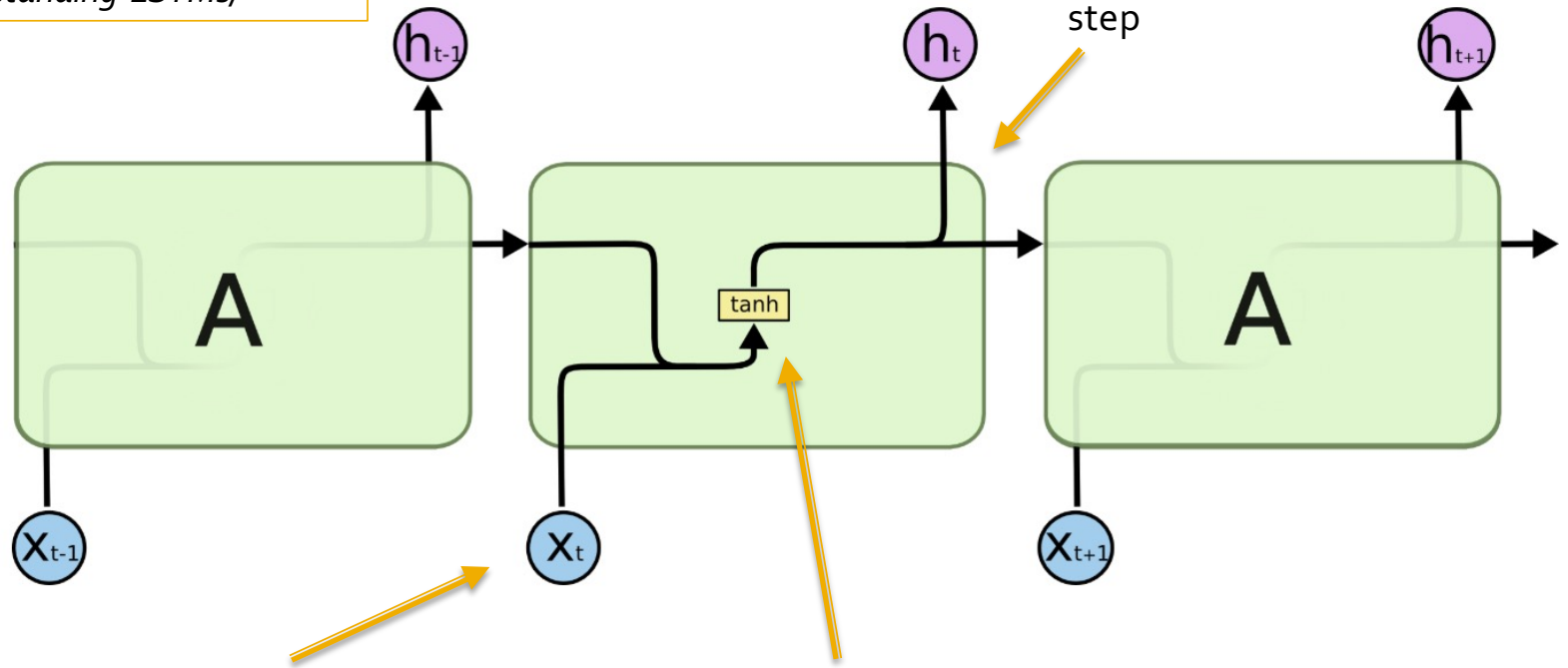
3.2.1 Simple Recurrent Networks



- activation value of the hidden layer depends on
  - the current input; and
  - the activation value of the hidden layer from the previous step

3.2.1 Unrolling a simple RNN

Diagrams from Colah's blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



3.2.1 RNN

ADVANTAGES

- Hidden layer from previous timestep provides memory
- No fixed length limit on amount of prior context
  - The weights U determine how the network should use past context in calculating output for current input

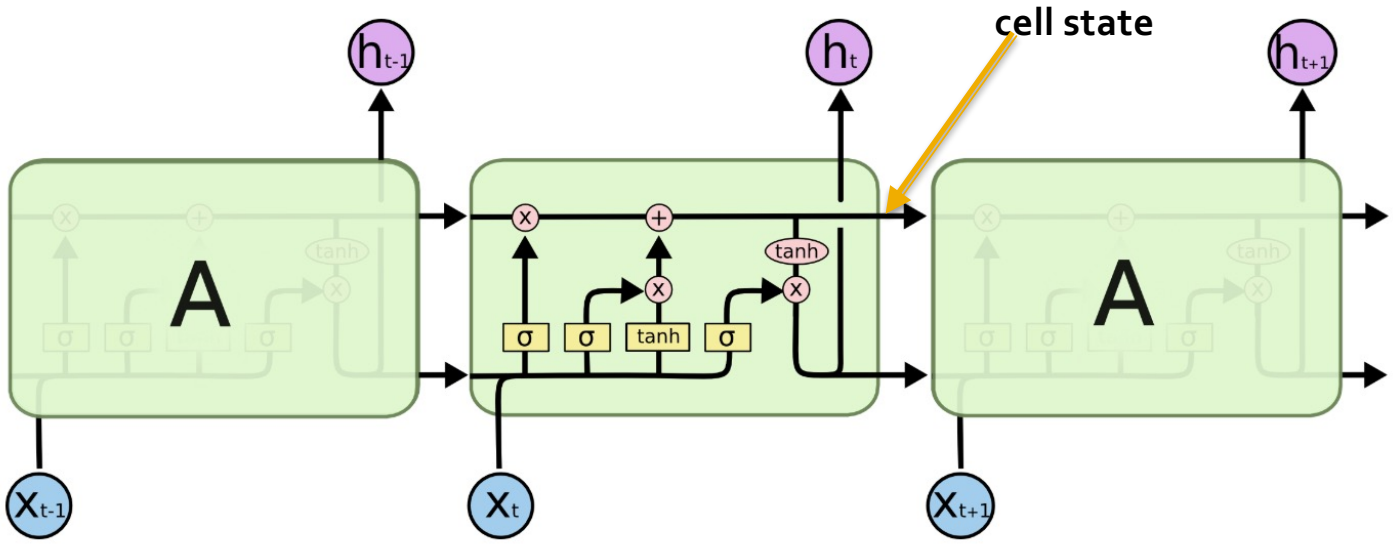
DISADVANTAGES

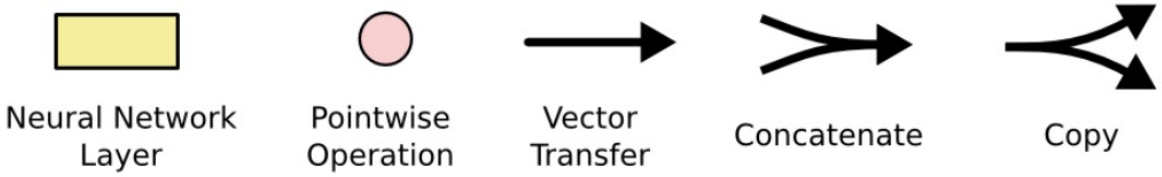
- Whilst RNN theoretically has access to the entire preceding sequence, in practice, information tends to be fairly local
- Difficult to capture long-range semantic dependencies
  - Hidden layer weights need to capture information for short range dependencies (e.g., pluralisation agreement) and long range dependencies

Questions from Lecture 3.2

1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

3.2.2 Unrolling an LSTM

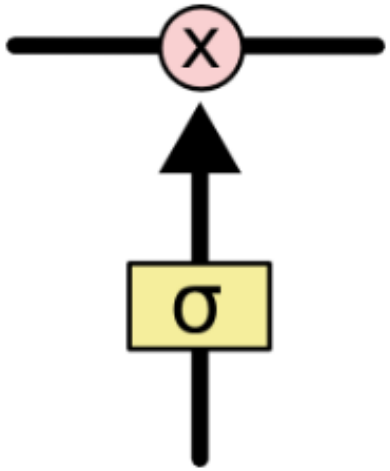




Core idea is that at each time step, hidden layer activation values depend on

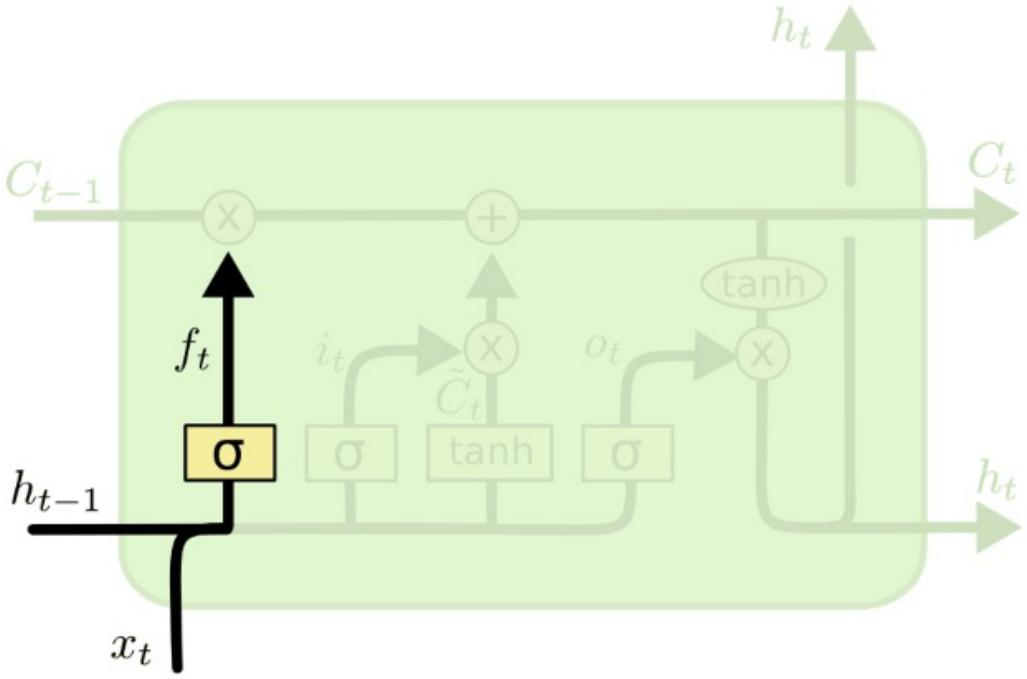
- current input
- activation values from the hidden layer at previous time step (*short term memory*)
- activation values from the **cell state** (*long term memory*)

3.2.2 Gates inside neural units



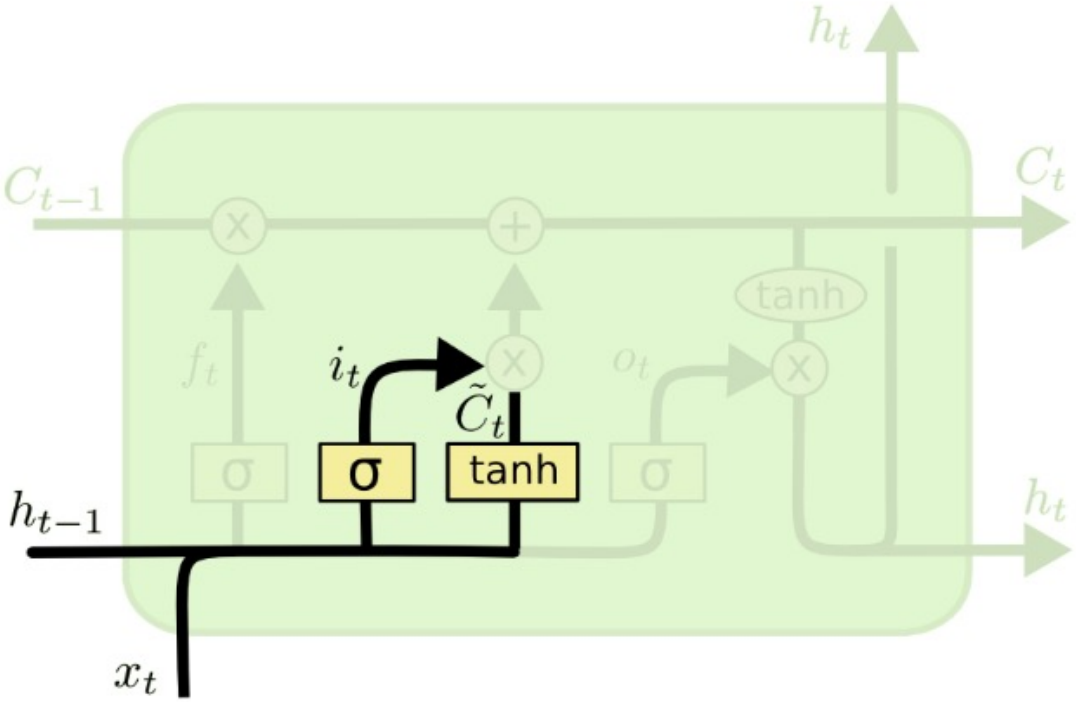
- a way to control flow of information
  - optionally allow information through
- composed of
  - a sigmoid neural net layer
  - a pointwise multiplication
- Sigmoid outputs numbers between 0 and 1, describing how much of each component should be let through
  - it controls the gate

3.2.2 Forget



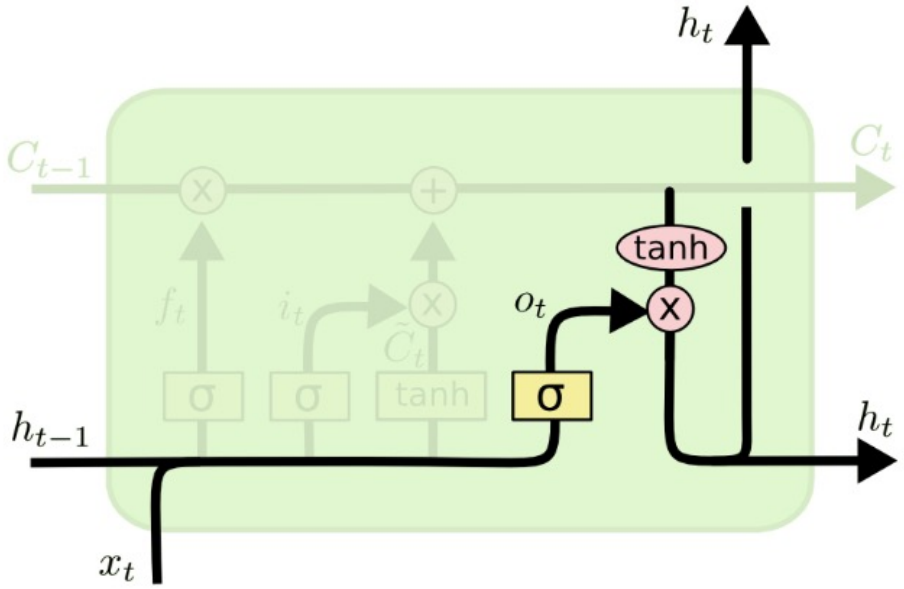
- What in the current cell state should be forgotten?
- Controlled by the current input and the short term memory

3.2.2 Input



- What of the current input and short-term memory should be put into the long-term memory?
  - sigmoid decides which values to update
  - tanh decides what those values should be

3.2.2 Output

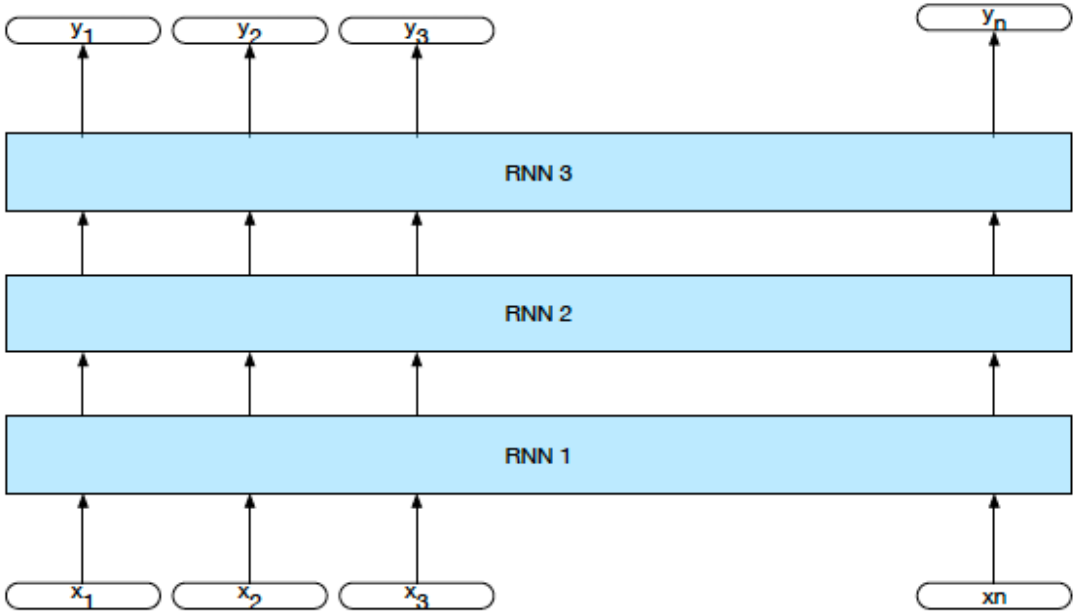


What should be **output** at this timestep?

Questions from Lecture 3.2

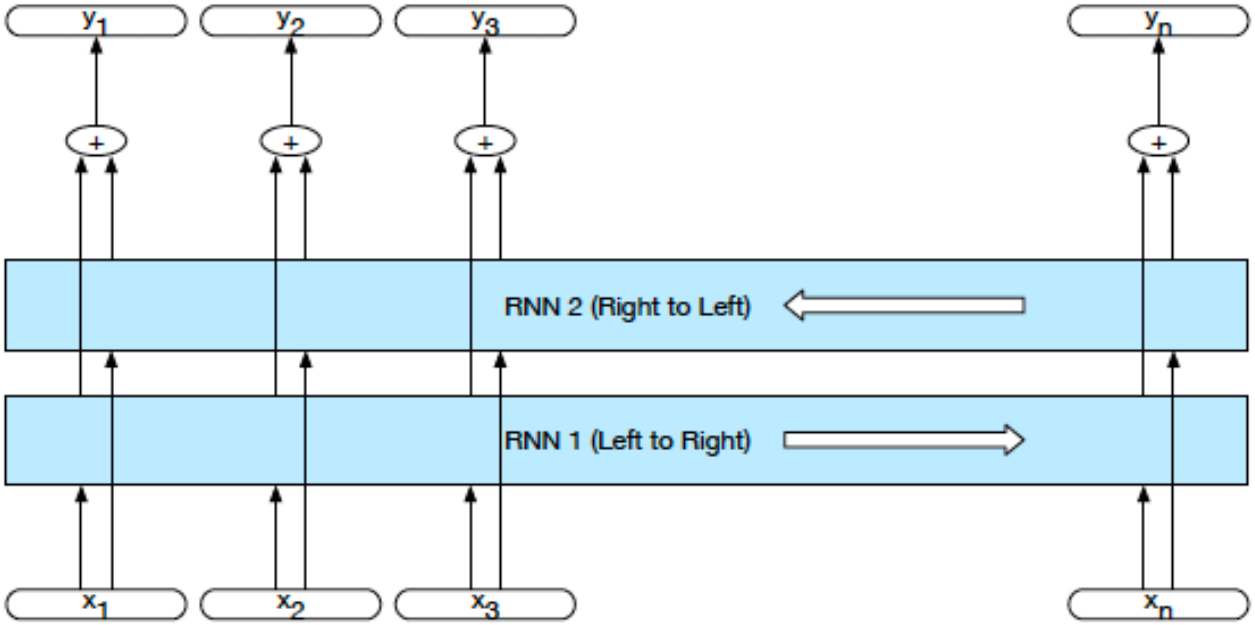
1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

3.2.3 Stacked RNNs



- Output (at each time step) from RNN is fed as input into another RNN
- Stacking leads to **deep** networks and **deep** learning
- Different layers induce representations at different levels of abstraction
  - short range syntactic dependencies in early layers
  - long range semantic dependencies in deeper layers

3.2.3 Bidirectional RNNs

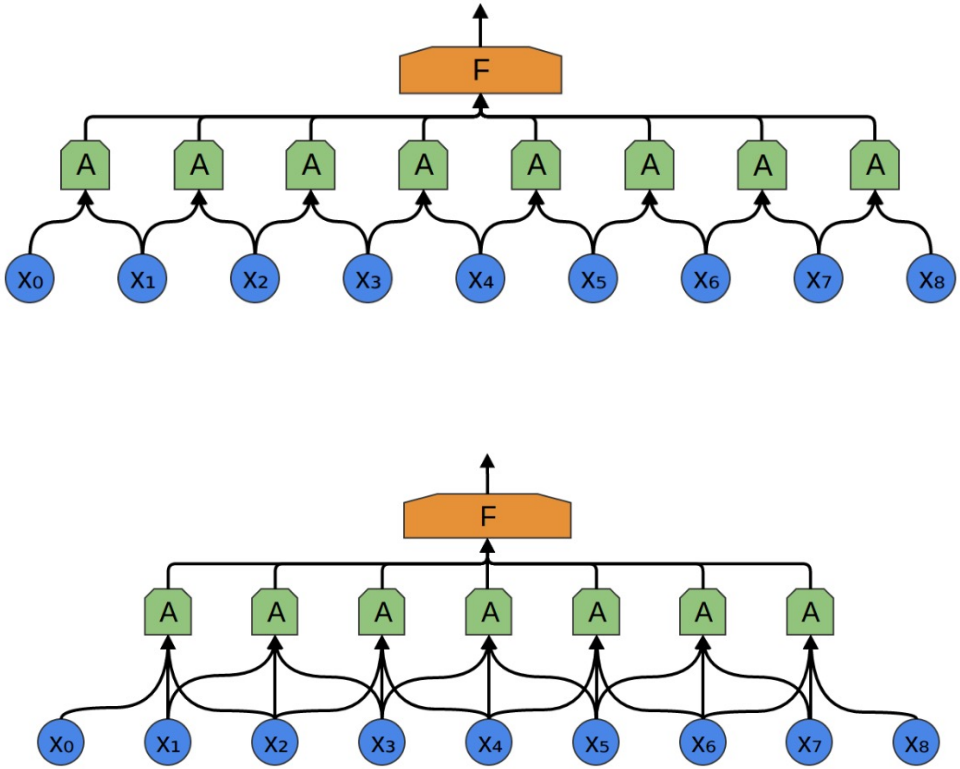


- Take advantage of right context as well as left context
- Pair of RNNs
  - one is trained from left-to-right
  - the other is trained from right-to-left
- Add or concatenate hidden layers from both RNNs to produce output at each timestep

Questions from Lecture 3.2

1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

3.2.4 Convolutions



- Convolutional layers look at groups of inputs or neurons, e.g.,
  - convolution with window size 2
  - convolution with window size 3
- **kernel** function A is learnt which looks for or filters for a particular pattern / feature in the input e.g.,
  - 2 character sequence "un"
  - 3 character sequence "ing"

Questions from Lecture 3.2

1. What is an RNN and what advantage(s) does it have over a FF-NN (particularly when applied to language modelling)?
2. What is an LSTM?
3. How might 2 RNNs be used in the same network?
4. What is a CNN?
5. Why might it be useful to have character based language models rather than word based language models?

3.2.5 Advantages of character-aware NLMs

- allows morphological information to be utilised
- particular useful for low-frequency words

Discussion of paper

Character aware neural language models (Kim et al. 2016)

Kim et al. (2016) provide a comparison of character-aware neural language models with word-level models. Read the paper and consider the following questions.

1. What problem with using n-gram models is addressed by the use of neural language models? Why is it not completely addressed? How might character-aware models help?

2. Why are LSTMs generally preferred over vanilla RNNs in language modelling?

3. In a character-level CNN, what is the purpose of a filter or kernel? How many filters do they state are typically used in NLP applications? How many do the authors use in each of their `small` and `large` architectures?

4. How is the character-level CNN incorporated into the overall architecture of the RNN-LM?

5. How are OOV words handled in these experiments? What potential improvement could the authors have made and why didn't they do it?

6. Which model(s) performs best in the optimization experiments on the Penn Treebank?

7. Why do the authors expect the performance gains to be more in other languages such as Arabic than in English? Are their expectations met in the experimental results?

8. What advantages does the authors' model have over the MLBL model of Botha and Blunsom (2014)?

9. What observations can you make of the nearest neighbours of 'richard' using each of the word representations?

10. What are the main conclusions of the paper? Are you convinced?

Group discussions

15 mins

Question 1

- What problem with using n-gram models is addressed by the use of neural language models?
- Why is it not completely addressed?
- How might character-aware models help?

Neural Language Models (NLM) address the n-gram data sparsity issue through parameterization of words as vectors (word embeddings) and using them as inputs to a neural network (Bengio, Ducharme, and Vincent 2003; Mikolov et al. 2010). The parameters are learned as part of the training process. Word embeddings obtained through NLMs exhibit the property whereby semantically close words are likewise close in the induced vector space (as is the case with non-neural techniques such as Latent Semantic Analysis (Deerwester, Dumais, and Harshman 1990)).

While NLMs have been shown to outperform count-based n-gram language models (Mikolov et al. 2011), they are blind to subword information (e.g. morphemes). For example, they do not know, a priori, that *eventful*, *eventfully*, *uneventful*, and *uneventfully* should have structurally related embeddings in the vector space. Embeddings of rare words can thus be poorly estimated, leading to high perplexities for rare words (and words surrounding them). This is especially problematic in morphologically rich languages with long-tailed frequency distributions or domains with dynamic vocabularies (e.g. social media).

In this work, we propose a language model that leverages subword information through a character-level convolutional neural network (CNN), whose output is used as an input to a recurrent neural network language model (RNN-LM). Unlike previous works that utilize subword in-

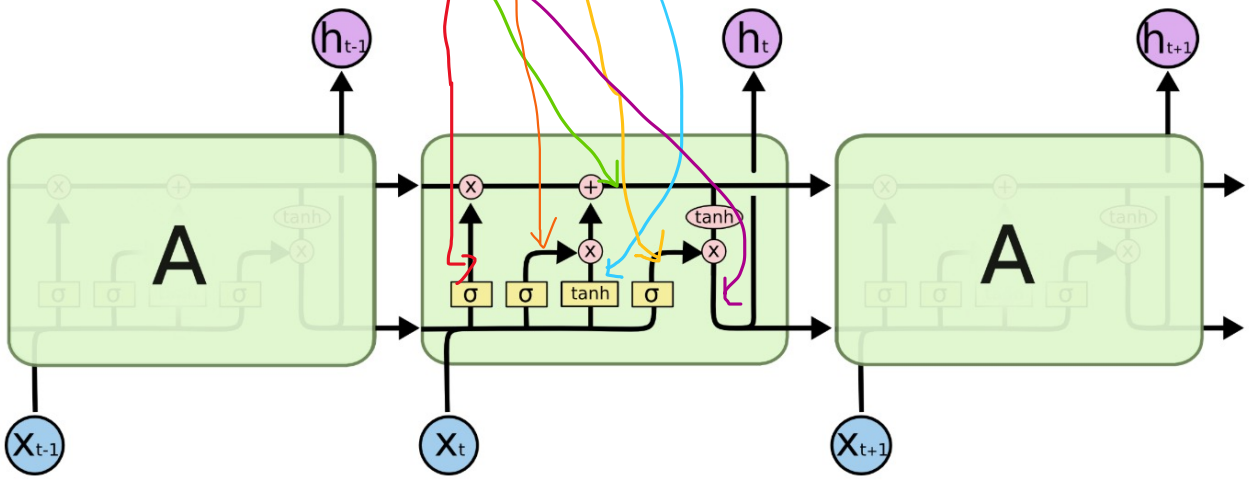
Question 2

- Why are LSTMs generally preferred over vanilla RNNs in language modelling?

Long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) addresses the problem of learning long range dependencies by augmenting the RNN with a memory cell vector $\mathbf{c}_t \in \mathbb{R}^n$ at each time step. Concretely, one step of an LSTM takes as input $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}$ and produces $\mathbf{h}_t$, $\mathbf{c}_t$ via the following intermediate calculations:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i \mathbf{x}_t + \mathbf{U}^i \mathbf{h}_{t-1} + \mathbf{b}^i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f \mathbf{x}_t + \mathbf{U}^f \mathbf{h}_{t-1} + \mathbf{b}^f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}^o \mathbf{x}_t + \mathbf{U}^o \mathbf{h}_{t-1} + \mathbf{b}^o) \\
\mathbf{g}_t &= \tanh(\mathbf{W}^g \mathbf{x}_t + \mathbf{U}^g \mathbf{h}_{t-1} + \mathbf{b}^g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2}
$$

Note: Slide updated after seminar as arrows to forget and input gate round the wrong way!



Question 3

- In a character-level CNN, what is the purpose of a filter or kernel?
- How many filters do they state are typically used in NLP applications?
- How many do the authors use in each of their small and large architectures?

We apply a narrow convolution between $\mathbf{C}^k$ and a *filter* (or *kernel*) $\mathbf{H} \in \mathbb{R}^{d \times w}$ of width w, after which we add a bias and apply a nonlinearity to obtain a *feature map* $\mathbf{f}^k \in \mathbb{R}^{l-w+1}$. Specifically, the i-th element of $\mathbf{f}^k$ is given by:

$$\mathbf{f}^k[i] = \tanh(\langle \mathbf{C}^k[*, i : i + w - 1], \mathbf{H} \rangle + b) \qquad (5)$$

where $\mathbf{C}^k[*, i : i+w-1]$ is the i-to-$(i+w-1)$-th column of $\mathbf{C}^k$ and $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{Tr}(\mathbf{A}\mathbf{B}^T)$ is the Frobenius inner product. Finally, we take the *max-over-time*

$$y^k = \max_i \mathbf{f}^k[i] \qquad (6)$$

as the feature corresponding to the filter $\mathbf{H}$ (when applied to word k). The idea is to capture the most important feature—the one with the highest value—for a given filter. A filter is essentially picking out a character n-gram, where the size of the n-gram corresponds to the filter width.

We have described the process by which *one* feature is obtained from *one* filter matrix. Our CharCNN uses multiple filters of varying widths to obtain the feature vector for k. So if we have a total of h filters $\mathbf{H}_1, \ldots, \mathbf{H}_h$, then $\mathbf{y}^k = [y_1^k, \ldots, y_h^k]$ is the input representation of k. For many NLP applications h is typically chosen to be in $[100, 1000]$.

Question 4

- How is the character-level CNN incorporated into the overall architecture of the RNN-LM?

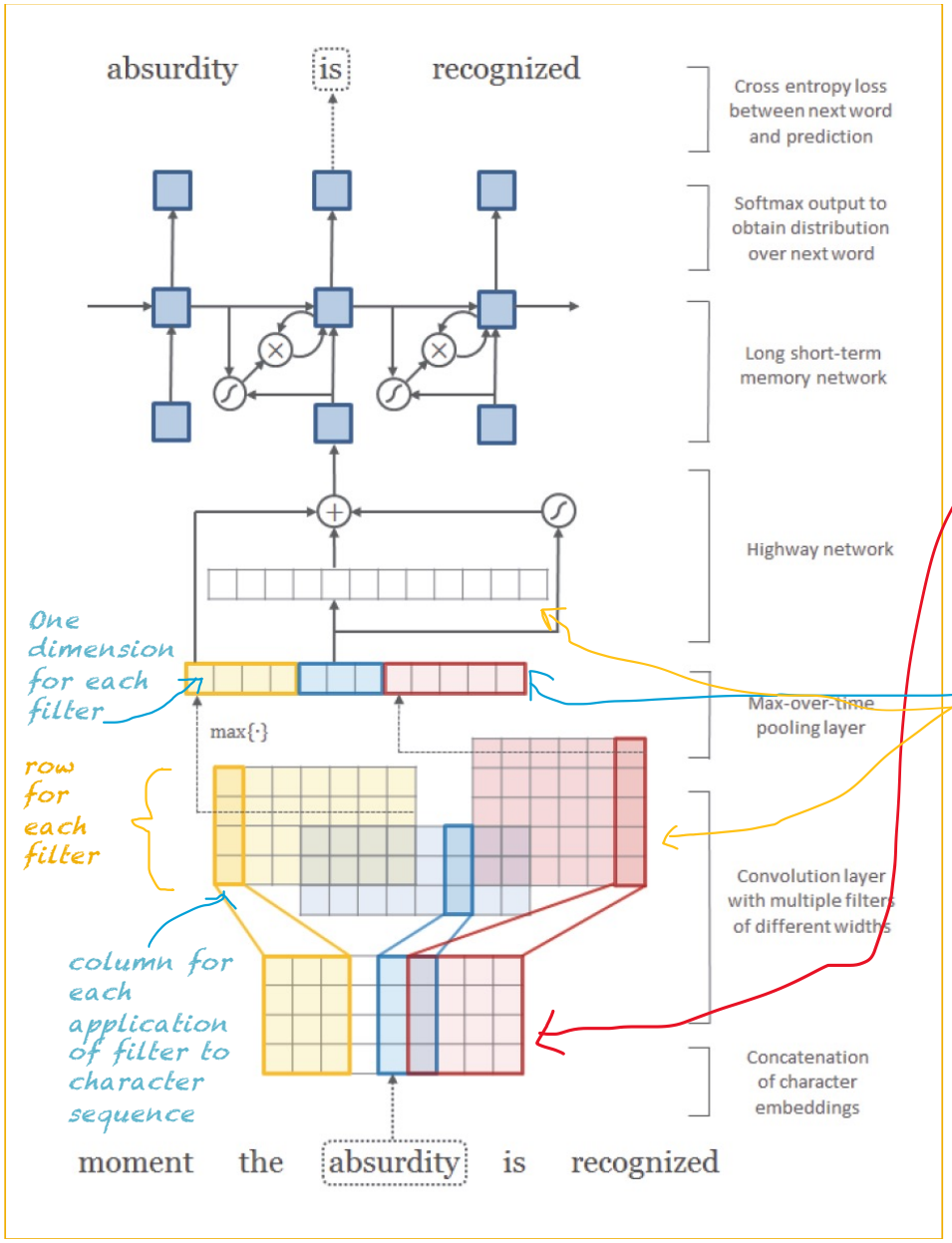


Figure 1: Architecture of our language model applied to an example sentence. Best viewed in color. Here the model takes *absurdity* as the current input and combines it with the history (as represented by the hidden state) to predict the next word, *is*. First layer performs a lookup of character embeddings (of dimension four) and stacks them to form the matrix $\mathbf{C}^k$. Then convolution operations are applied between $\mathbf{C}^k$ and multiple filter matrices. Note that in the above example we have twelve filters—three filters of width two (blue), four filters of width three (yellow), and five filters of width four (red). A max-over-time pooling operation is applied to obtain a fixed-dimensional representation of the word, which is given to the highway network. The highway network's output is used as the input to a multi-layer LSTM. Finally, an affine transformation followed by a softmax is applied over the hidden representation of the LSTM to obtain the distribution over the next word. Cross entropy loss between the (predicted) distribution over next word and the actual next word is minimized. Element-wise addition, multiplication, and sigmoid operators are depicted in circles, and affine transformations (plus nonlinearities where appropriate) are represented by solid arrows.

Question 5

- How are OOV words handled in these experiments?
- What potential improvement could the authors have made and why didn't they do it?

In these datasets only singleton words were replaced with <unk> and hence we effectively use the full vocabulary. It is worth noting that the character model can utilize surface forms of OOV tokens (which were replaced with <unk>), but we do not do this and stick to the preprocessed versions (despite disadvantaging the character models) for exact comparison against prior work.

Question 6

- Which model(s) performs best in the optimization experiments on the Penn Treebank?

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

Table 3: Performance of our model versus other neural language models on the English Penn Treebank test set. PPL refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. †For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

Question 7

- Why do the authors expect the performance gains to be more in other languages such as Arabic than in English?
- Are their expectations met in the experimental results?

With the optimal hyperparameters tuned on PTB, we apply the model to various morphologically rich languages: Czech, German, French, Spanish, Russian, and Arabic. Non-Arabic data comes from the 2013 ACL Workshop on Machine Translation,[7] and we use the same train/validation/test splits as in Botha and Blunsom (2014). While the raw data are publicly available, we obtained the preprocessed versions from the authors,[8] whose morphological NLM serves as a baseline for our work. We train on both the small datasets (DATA-S) with 1m tokens per language, and the large datasets (DATA-L) including the large English data which has a much bigger $|\mathcal{V}|$ than the PTB. Arabic data comes from the News-Commentary corpus,[9] and we perform our own preprocessing and train/validation/test splits.

		DATA-S					
		CS	DE	ES	FR	RU	AR
Botha	KN-4	545	366	241	274	396	323
	MLBL	465	296	200	225	304	–
Small	Word	503	305	212	229	352	216
	Morph	414	278	197	216	290	230
	Char	401	260	182	189	278	196
Large	Word	493	286	200	222	357	172
	Morph	398	263	177	196	271	**148**
	Char	**371**	**239**	**165**	**184**	**261**	**148**

		DATA-L					
		CS	DE	ES	FR	RU	EN
Botha	KN-4	862	463	219	243	390	291
	MLBL	643	404	203	227	**300**	273
Small	Word	701	347	186	202	353	236
	Morph	615	331	189	209	331	233
	Char	**578**	**305**	**169**	**190**	313	**216**

Question 8

- What advantages does the authors' model have over the MLBL model of Botha and Blunsom (2014)?

A specific class of FNLMs leverages morphemic information by viewing a word as a function of its (learned) morpheme embeddings (Luong, Socher, and Manning 2013; Botha and Blunsom 2014; Qui et al. 2014). For example Luong, Socher, and Manning (2013) apply a recursive neural network over morpheme embeddings to obtain the embedding for a single word. While such models have proved useful, they require morphological tagging as a preprocessing step.

Question 9

- What observations can you make of the nearest neighbours of ‘richard’ using each of the word representations?

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Table 6: Nearest neighbor words (based on cosine similarity) of word representations from the large word-level and character-level (before and after highway layers) models trained on the PTB. Last three words are OOV words, and therefore they do not have representations in the word-level model.

Question 10

- What are the main conclusions of the paper?
- Are you convinced?

Conclusion

We have introduced a neural language model that utilizes only character-level inputs. Predictions are still made at the word-level. Despite having fewer parameters, our model outperforms baseline models that utilize word/morpheme embeddings in the input layer. Our work questions the necessity of word embeddings (as inputs) for neural language modeling.

Analysis of word representations obtained from the character composition part of the model further indicates that the model is able to encode, from characters only, rich semantic and orthographic features. Using the CharCNN and highway layers for representation learning (e.g. as input into word2vec (Mikolov et al. 2013)) remains an avenue for future work.

Insofar as sequential processing of words as inputs is ubiquitous in natural language processing, it would be interesting to see if the architecture introduced in this paper is viable for other tasks—for example, as an encoder/decoder in neural machine translation (Cho et al. 2014; Sutskever, Vinyals, and Le 2014).

References

- Bengio, Y. et al. 2003. A neural probabilistic language model. In *Journal of Machine Learning Research*.
- Botha, J., and Blunsom, P. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of ICML*
- Kim, Y. et al. 2016. Character-aware neural language models. In *Proceedings of the AAAI*.
- Mikolov, T. et al. 2010. Recurrent neural network based language model. In *Proceedings of Interspeech*
- Mnih and Hinton, G. 2008. A scalable hierarchical distributed language model. In *Proceedings of NIPS*.