RNNs and LSTMs

Simple Recurrent Networks (RNNs or Elman Nets)

Modeling Time in Neural Networks

Yet the simple NLP classifiers we've seen (for example for sentiment analysis) mostly ignore time

Language is inherently temporal

• (Feedforward neural LMs (and the transformers we'll see later) use a "moving window" approach to time.)

Here we introduce a deep learning architecture with a different way of representing time

• RNNs and their variants like LSTMs

Recurrent Neural Networks (RNNs)

Any network that contains a cycle within its network connections.

The value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input.

Simple Recurrent Nets (Elman nets)

The hidden layer has a recurrence as part of its input The activation value h_t depends on x_t but also $h_{t-1}!$

Forward inference in simple RNNs

Very similar to the feedforward networks we've seen!

$$
\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)
$$

$$
\mathbf{y}_t = softmax(\mathbf{V}\mathbf{h}_t)
$$

Inference has to be incremental time *t* 1 mandates an incremental inference algorithm that proceeds from the start

Computing h at time t requires that we first computed h at the previous time step!

```
h_0 \leftarrow 0for i \leftarrow 1 to LENGTH(x) do
        \mathbf{h}_i \leftarrow g(\mathbf{U} \mathbf{h}_{i-1} + \mathbf{W} \mathbf{x}_i)y_i \leftarrow f(Vh_i)return y
```
function FORWARDRNN(x, *network*) returns output sequence y

Training in simple RNNs

- training set,
- a loss function,
- backpropagation

Just like feedforward training:

Weights that need to be updated:

- **W**, the weights from the input layer to the hidden layer,
- **U**, the weights from the previous hidden layer to the current hidden layer,
- V**,** the weights from the hidden layer to the output layer.

Training in simple RNNs: unrolling in time

Unlike feedforward networks:

1. To compute loss function for the output at time *t* we need the hidden layer from $time t - 1.$

2. hidden layer at time t influences the output at time *t* and hidden layer at time *t+1* (and hence the output and loss at *t*+1).

So: to measure error accruing to h*t***,**

• need to know its influence on both the current output *as well as the ones that follow*.

We unroll a recurrent network into a feedforward computational graph eliminating recurrence

- 1. Given an input sequence,
- 2. Generate an unrolled feedforward network specific to input
- 3. Use graph to train weights directly via ordinary backprop (or can do forward inference)

RNNs and LSTMs

Simple Recurrent Networks (RNNs or Elman Nets)

RNNs and LSTMs

RNNs as Language Models

to know how likely the next word is *"fish"* we would compute:

with the chain rule:

with the chain rule:

$P(fish | Thanks for all the)$

Context. For Example 12 and wanter is exampled. For example, if the preceding \blacksquare *P*(*fish|Thanks for all the*)

that language models predict the next word in a sequence given some preceding Language models give us the ability to assign such a conditional probability to every Language models give us the ability to assign such a conditional probability to every possible next word, giving us a distribution over the entire vocabulary. We can also write vocabulary The n-gram language models of C compute the probability of C compute the probability of a word given by counts of its occurrence with the *n*1 prior words. The context is thus of size *n*1. For the feedforward language models of Chapter 7, the context is the window size.

RNN language models (Mikolov et al., 2010) process the input sequence one

The size of the conditioning context for different LMs

The n-gram LM:

Context size is the $n-1$ prior words we condition on.

The feedforward LM:

Context is the window size.

The RNN LM:

No fixed context size; h_{t-1} represents entire history

FFN LMs vs RNN LMs

FFN RNN

…

a) b)

Given input X of of N tokens represented as one-hot vectors This hidden is the state of a state is the language of the state and output layer which is passed to generate the contact of the state of $\frac{1}{2}$ $\frac{1}{\sqrt{2}}$ Given input X of of N tokens represented as one-hot vectors

Forward inference in the RNN LM word, multiples it by the weight matrix W, and then adds it to the weight matrix W, and then adds it to the hi
W, and the hidden layer from additional then added it to the hidden layer from a state of the hidden layer fro Forward inference in the RNN LM

Combine …

$$
\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_t; \dots; \mathbf{x}_N]
$$

Use embedding matrix to get the embedding for current token x_t

$$
e_t = Ex_t
$$

Combine ...

$$
h_t = o(1)h_{t-1} + Me_t
$$

$$
\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)
$$

 $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$ $\mathbf{y}_t - \text{Solution} (\mathbf{v} \mathbf{u}_t)$

This hidden layer is then used to generate an output layer which is passed through a

softmax layer to generate a probability distribution over the entire vocabulary. That is not the entire vocabulary. That is not the entire vocabulary over the entire vocabulary. That is not the entire vocabulary. That is n

Computing the probability that the next word is word *k* \blacksquare The probability of an entire sequence is just the probability of the probabilities of the probabilities of each \blacksquare **item in the sequence, we consider the sequence, we consider the violus word** κ **by Computing the property**

wi at time step *i*.

 $P(w_{1:n}) = \prod$ *n* $i=1$ $P(w_{1:n}) = \prod P(w_i|w_{1:i-1})$ $= \prod \hat{\mathbf{y}}_i[w_i]$ *n* $i=1$ $P(w_{t+1} = k | w_1, \ldots, w_t) = \hat{y}_t[k]$ $i=1$ *^P*(*w*1:*n*) = ^Y

-
-
-
-
-
- - - -
		- -
		- -
		-
	-
- -
- *P*(*wi|w*1:*i*1) (8.8)
-
-
-
-
- -
	-
	-
-
-
-
- -
	-
	-
	-
	-
	-
	-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
- probability that a particular word *k* in the vocabulary is the next word is represented

wi at time step *i*.

Training RNN LM

• **Self-supervision**

- take a corpus of text as training material
- at each time step *t*
- ask the model to predict the next word.
- **Why called self-supervised:** we don't need human labels; the text is its own supervision signal
- We train the model to
	- minimize the error
	- in predicting the true next word in the training sequence,
	- using cross-entropy as the loss function.

The difference between:

- a predicted probability distribution L_{CE} $=$ $I_{CFT} = -\sum_{i} \mathbf{v}_t[w] \log \hat{\mathbf{v}}_t[w]$
- the correct distribution. where the entries is the actual next word is 1, and all the other entries are 0. Thus, and all the other entries are 0. Thus, and all the other entries are 0. Thus, and 0
- CE loss for LMs is simpler!!!

The cross-entropy is determined by the probability the probability the probability the probability the probability the probability of the probability of the probability of the probability of
- the correct distribution y_t *is* a one-hot vector over the vocabulary • where the entry for the actual next word is 1, and all the other entries are 0. correct distribution y_t is a one-hot vector over the vocabulary
- So the CE loss for LMs is only determined by the probability of next word. s for LMs is only determined by the probability of next word. prote the entry for the actual fiext word is 1, and an the other entries are 0.
In a CE lease familiar is analyzed at a major allow the model of the af manternand
- So at time t, CE loss is:

Cross-entropy loss Figure 8.6 Training RNNs as language models.

CE loss is:
$$
L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]
$$

Teacher forcing

We always give the model the correct history to predict the next word (rather than feeding the model the possible buggy guess from the prior time step).

This is called **teacher forcing** (in training we **force** the context to be correct based on the gold words)

What teacher forcing looks like:

- At word position t
- the model takes as input the correct word *wt* together with *ht*−1, computes a probability distribution over possible next words
- That gives loss for the next token *wt*+1
- Then we move on to next word, ignore what the model predicted for the next word and instead use the correct word *wt*+1 along with the prior history encoded to estimate the probability of token $wt+2$.

The input embedding matrix E and the final layer matrix V, are similar Instead of having two sets of embedding matrices, language models use a single mo embedding matrix, Land the marriager matrix, v, are similar at both the input and softmax layers. The input and softmax layers and software s

- The columns of E represent the word embeddings for each word in vocab. E is [d x |V|] we dispense with V and users with V and users with V and users with an area with the start of the start of the
Expected the start of the start
- The final layer matrix V helps give a score (logit) for each word in we also the manager matrix virtigs give a secre (logic) for each word in versions of the weights of the weight of an RNN language weight the weight of t $\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$ at the final laver matrix V helps give a score (logit) for each word in

Instead of having separate E and V, we just tie them together, using E^{T} instead of V:

Weight tying M aight tuing M a *second set* of learned word embeddings.

hidden layer of the network through the calculation of Vh. V is of shape [*|V|*⇥*d*].

$$
\begin{array}{rcl}\n\mathbf{e}_t &= \mathbf{E} \mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U} \mathbf{h}_{t-1} + \mathbf{W} \mathbf{e}_t) \\
\hat{\mathbf{y}}_t &= softmax(\mathbf{E}^\mathsf{T} \mathbf{h}_t)\n\end{array}
$$

RNNs and LSTMs

RNNs as Language Models

RNNs and LSTMs

RNNs for Sequences

RNNs for sequence labeling

Assign a label to each element of a sequence Part-of-speech tagging

RNNs for sequence contact

Instead of taking the last state, could use some pooling function of all the output states, like **mean pooling** pools all the *n* hidden states by taking their element-wise mean: *n*

$$
\mathbf{h}_{mean} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{h}_i
$$

Autoregressive generation

Stacked RNNs

those cases we would like to use words from the context to the right of *t*. One way

to do this is to run two separate RNNs, one left-to-right, and one left-to-right, and one right-to-left, and o

t represents everything the network knows about the sequence up to that point. The

senting eventy the network has gleaned from the sequence so far. The sequence so far α far α

the left of the current time.

basis for a local labeling decision of a local labeling decision. In the case of a local labeling decision. In

This new notation h*^f*

Bidirectional RNNs In the left-to-right RNNNs we've discussed so far a given time at a given time so far \mathbb{R}^n h*f ^t* = RNNforward(x1*,...,*x*t*) (8.16) **discript Bidirectional RNNs** sequence from *t* to the end of the sequence.

8.4 • STACKED AND BIDIRECTIONAL RNN ARCHITECTURES 13.4 • STACKED AND BIDIRECTIONAL RNN ARCHITECTURES 13.4 • ST

$$
\mathbf{h}_t^f = \text{RNN}_{\text{forward}}(\mathbf{x}_1, ..., \mathbf{x}_t)
$$
\n
$$
\mathbf{h}_t^b = \text{RNN}_{\text{backward}}(\mathbf{x}_t, ..., \mathbf{x}_n)
$$
\n
$$
\mathbf{h}_t = [\mathbf{h}_t^f; \mathbf{h}_t^b]
$$
\n
$$
= \mathbf{h}_t^f \oplus \mathbf{h}_t^b
$$

the forward and backward and backward pass. Other simple ways to combine the forward and backward and forward a
The forward and backward and the

backward contexts include element-wise addition or multiplication or multiplication. The output at α

each step in time the time that time the left and time the left and time the right of the right of the current

input . In sequence labeling applications, the sequence labeling applications, the server as the server as the
In server as the server as

This new notation h*^f*

Here, the hidden state h*^b*

the left of the left of the current time. The current time of the current time of the current time. The current

A bidirectional RNN (Schuster and Paliwal, 1997) combines two independent bidirectional

Bidirectional RNNs for classification

RNNs and LSTMs

RNNs for Sequences

RNNs and LSTMS

The LSTM

Motivating the LSTM: dealing with distance

- It's hard to assign probabilities accurately when context is very far away:
	- The flights the airline was canceling were full.
- Hidden layers are being forced to do two things:
	- Provide information useful for the current decision,
	- Update and carry forward information required for future decisions.
- Another problem: During backprop, we have to repeatedly multiply gradients through time and many h's
	- The "vanishing gradient" problem

The LSTM: Long short-term memory network

LSTMs divide the context management problem into two subproblems:

- removing information no longer needed from the context,
- adding information likely to be needed for later decision making
- LSTMs add:
	- explicit context layer
	- Neural circuits with **gates** to control information flow

Forget gate

by the context vector to remove the information from context that is no longer re-end of the information from
In the information from context that is no longer re-end of the information from context that is no longer reelement-wise multipliers. Element-wise multiplication of two vectors (represented by the operator \mathcal{L} and sometimes called the Hadamard product) is the vector of the same dimension The next task is to compute task is to compute the actual information we need to extract from the previ-from the pre ous hidden state and current inputs—the same basic computation we've been using

input vectors:

b Deletes information from the context that is no longer needed.

$$
\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)
$$
\n
$$
\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t
$$

put and passes that the through a sigmoid. This mask is the multiplied element-wise then multiplied element-wis

for all our recurrent networks.

Regular passing of information Realism is the next task is to compute the actual information we need to extract $\frac{1}{2}$ ous hidden state and current inputs—the same basic computation we've been using

for all our recurrent networks.

$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$

add gate Next, we generate the mask for the mask for the mask for the add gate to select the information to th
Additional to the information to the information to add to the information to the information to the informati

Add gate g*^t* = tanh(U*g*h*t*¹ +W*g*x*t*) (8.22)

current context.

Selecting information to add to current context *information to add to current context*

add gate Next, we generate the mask for the mask for the mask for the add gate to select the information to th
The information to the information

$$
\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)
$$

$$
\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t
$$

Add this to the modified context vector to get our new context vector.

$$
\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t
$$

Output gate Output gate we'll use it the output gate

Decide what information is required for the current hidden state (as opposed to what information needs to to be preserved for future decisions). Γ is republic tion is required for the current hidden state (as opposed to what information needs to

$o_t = \sigma(U_0 h_{t-1} + W_0 x_t)$ $h_t = o_t \cdot \text{tanh}(c_t)$

Fig. 8.13 illustrates the complete computation for a single LSTM unit. Given the

appropriate weights for the various gates, and α and α and α input the context as input the context as input the context as input the context as in α

layer, and hidden layer from the previous time step, along with the current input

The LSTM

 \rightarrow c_t $-h_t$

Units

RNNs and LSTMS

The LSTM

RNNs and LSTMs

The LSTM Encoder-Decoder Architecture

Four architectures for NLP tasks with RNNs

3 components of an encoder-decoder

1. An encoder that accepts an input sequence, *x*1:*n*, and generates a corresponding sequence of contextualized representations, $h1:n$.

2. A context vector, *c*, which is a function of $h1:n$, and conveys the essence of the input to the decoder.

3. A decoder, which accepts *c* as input and generates an arbitrary length sequence of hidden states *h*1:*m*, from which a corresponding sequence of output states *y*1:*m*, can be obtained

Encoder-decoder

More formally, if *g* is an activation function like *tanh* or ReLU, a function of

set of possible vocabulary items, then at time *t* the output y*^t* and hidden state h*^t* are

We only have to make one slight change to turn this language model with au-

toregressive generation into an encoder-decoder model that is a translation model

that can translate from a source text in one language text in one language to a target text in a second:

$h_t = g(h_{t-1}, x_t)$ $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{h}_t)$ Recall that in any language model, we can break down the probability as follows: $p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1,y_2)...p(y_m|y_1,...,y_{m-1})$ $\sigma(\cdot-\tau)$ to $\sigma(\cdot-\tau)$ $\mathcal{J}l$ we then $\langle \cdot \cdot \cdot \rangle$ Regular language modeling

In this section we'll describe an encoder-decoder network based on a pair of

Encoder-decoder for translation **the incoder-decoder for translation** RNNs, but we'll see in Chapter 13 how to apply them to transformers as well. We'll \blacksquare but the encoder-decoder for translation

RNN language model *p*(*y*), the probability of a sequence *y*.

generate the next token.

computed as: the computed as: the computed as:

More formally, if *g* is an activation function like *tanh* or ReLU, a function of

the input at time *t* and the hidden state at time *t* 1, and the softmax is over the

set of possible vocabulary items, then at time *t* the output y*^t* and hidden state h*^t* are

- Let $x =$ The green witch arrive $\langle s \rangle$
- Let $v \theta$ token *y* and *y y* α is the target text α (i.e. α). Then are in Spanish measurements in Spanish , α Let y = *llego ́la bruja verde*

Encoder-decoder for translation Concateur docc LITE SUSE ACCOULT TO LIGHBIGHON, AND LITERATOR TO A CONTROLLER

an English source text ("the green witch arrived"), to a Spanish sentence ("*llego* **lace bruja bruja bruja verdere word-by-word-by-witch can be glossed word-by-witch word-by-witch green's can be c** \blacksquare we can also induced models with a question-answer pair \blacksquare Let x be the source text plus a separate token <s> and y the target

add a sentence separation marker at the end of the source text, and the source text, and then simply sentence

encoder-decoder model computes the probability *p*(*y|x*) as follows:

section).

 $p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x) \dots p(y_m|y_1,...,y_{m-1},x)$

(we'll see the full model, which requires the full model, which requires the new concept of attention, in the n
In the next of attention, in the

Encoder-decoder simplified

Encoder-decoder showing context

this by adding *c* as a parameter to the computation of the current hidden state. using

 \mathbf{h}^d_t $\frac{d}{t} = g(\hat{y})$ $(t-1, h_{t-1}^d, c)$

Encoder-decoder equations \blacksquare Foroder-decoder equations s and the source software superior that the solution \mathbb{F}_q

$$
c = h_n^e
$$

\n
$$
h_0^d = c
$$

\n
$$
h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)
$$

\n
$$
\hat{y}_t = softmax(h_t^d)
$$

encoder model, with context available at each decoder model, with context available at each decoding timestep.

Thus *g* is a stand-in for some flavor of RNN

Section ??.

y[^]_{*t*−1} is the embedding for the output sampled from the softmax at the previous step ↑ The ^oy_t is a vector of probabilities over the vocabulary, representing the probability of each most probability of each word occurring at time to To generate to the sample from this distribution \hat{C} word occurring at time *t*. To generate text, we sample from this distribution \hat{v}_t . $\frac{1}{2}$ we'll becaming at third the generate text, we sample nomitations astribution $\frac{1}{2}$.

Training the encoder-decoder with teacher forcing

embedding layer

softmax \hat{y} hidden

layer(s)

answers

loss

RNNs and LSTMs

The LSTM Encoder-Decoder Architecture

RNNs and LSTMS

LSTM Attention

Problem with passing context c only from end

Requiring the context *c* to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Solution: attention

instead of being taken from the last hidden state, the context it's a weighted average of all the hidden states of the decoder. veighted average of all the hidden states of the decoder.
It's a weighted average of all the hidden states of the decoder.

this weighted average is also informed by part of the decoder state as this weighted average is also liftornied by part of the decoder state as
well, the state of the decoder right before the current token *i*.

$$
\mathbf{c} = f(\mathbf{h}_1^e \dots \mathbf{h}_n^e, \mathbf{h}_{i-1}^d)
$$

Attention

 \mathbf{h}^d_i $\frac{d}{i} = g(\hat{y})$

context available during decoding by conditioning the computation of the current

decoder hidden states on it (along with the prior hidden state and the prior hidden state and the prior α generated by the decoder), as we see in this equation (and Fig. 8.21): $i-1, h_{i-1}^d, c_i)$

We'll create a score that tells us how much to focus on each encoder state, how *relevant* each encoder state is to the decoder state: $\frac{1}{\sqrt{2}}$ similar to an encoder decoder hidden state is to an encoder the decoder hidden state is to an encoder hidden state is to an encoder the state is to an encoder the state is to an encoder the state is to an enc state, bow relevant as a product the dot in the dot product the dot product between the dot product between the
State bow relevant each encoder state is \blacksquare We'll create a score that tells us how much to focus on each encoder

for each encoder state *j*.

How to compute c? The score that results from this dot product is a scalar that reflects the degree of

decoder.

And then use this to help create a weighted average: iel ate a weighted avera *j* \overline{a} *^k*)) (8.36)

$$
score(\mathbf{h}_{i-1}^d, \mathbf{h}_{j}^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_{j}^e
$$

We'll normalize them with a softmax to create weights α_{ij} , that tell us the relevance of encoder hidden state *j* to hidden decoder state, h^d_{i-1} **The score score is set in the score that results from the score is a** scalar tellier weights α_i , that tell us similarity between the relevance of encoder hidden state *j* to hidden decoder state, h^d_{i-1} α_{ij} = softmax(score($\mathbf{h}_{i-1}^d, \mathbf{h}_j^e$)) \blacksquare We'll normalize them with a softmax to $\binom{e}{j}$ α_{ij}

$$
\mathbf{c}_i = \sum_j \alpha_{ij} \, \mathbf{h}_j^e
$$

the current decoder state by taking a weighted average over all the encoder hidden average over all the encoder

decoder.

states.

Encoder-decoder with attention, focusing on the computation of c

 ~ 100

RNNs and LSTMS

LSTM Attention