Language Models and RNNs

Lecture Plan

Today, we're diving in several **foundational concepts and tasks**.

1. Introduction to language modeling
2. Recurrent neural networks (RNNs)
3. Machine translation (a very language-y task w/ RNNs)

Intro to language modeling

Are these sentences okay?

Jane went to the store.

store to Jane went the.

Jane went store.

Jane goed to the store.

The store went to Jane.

The food truck went to Jane.

Consider the grammar of the language

Consider morphology and exceptions

Consider semantic categories

and their exceptions

Language modeling

Language modeling is the task of predicting what word comes next

```
The students opened their ____

    books     laptops      exams     minds
```

Given a sequence of words $x_1, x_2, \ldots, x_t$, compute the probability distribution of the next word x_{t+1}:

$$P\left(x_{t+1} \mid x_t, \ldots, x_1\right)$$

where x_{t+1} can be any word in the vocabulary.

Language modeling

For example, if we have some text

`The students opened their laptops`

Remember **the chain rule**!

$$P(A, B) = P(B \mid A)P(A)$$

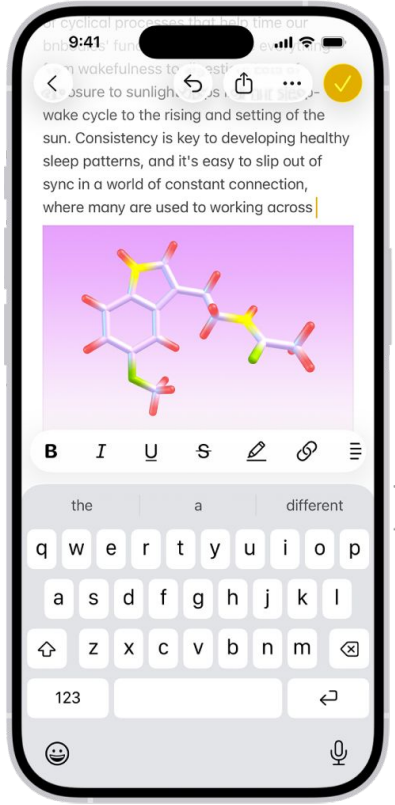
$$P(\text{The, students, opened, their, laptops})$$

$$= P(\text{The}) \times P(\text{students} \mid \text{The}) \times \cdots \times P(\text{laptops} \mid \text{their, opened, students, The})$$

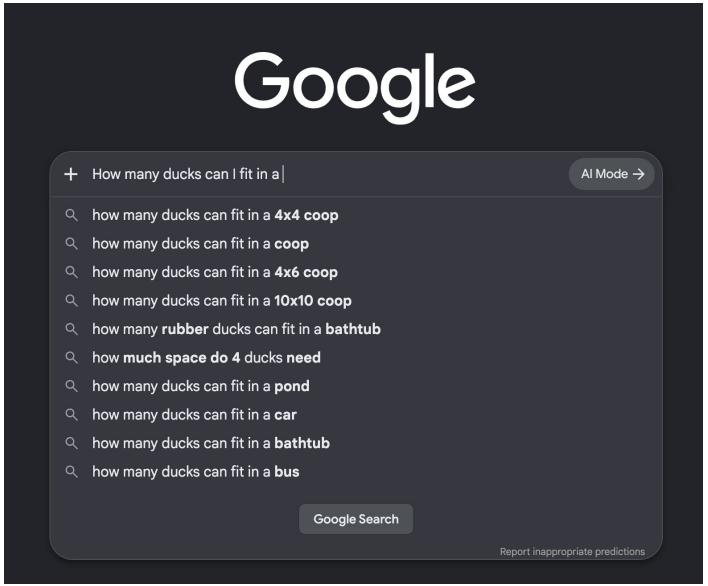
$$= \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots, x_1)$$

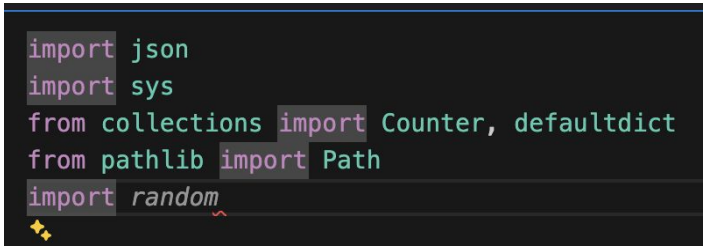
Our LM provides us these probabilities

Everyday examples



Predictive text; tap a suggestion to apply.





What can we do with LMs?

Score the probability of sentences

Jane went to the store. → high
store to Jane went the. → low

(same as calculating loss for training)

Generate sentences

while didn't choose end-of-sentence token (e.g. [EOS]):
calculate probability *P*(next word | context)
sample a new word from the probability distribution

Language modeling

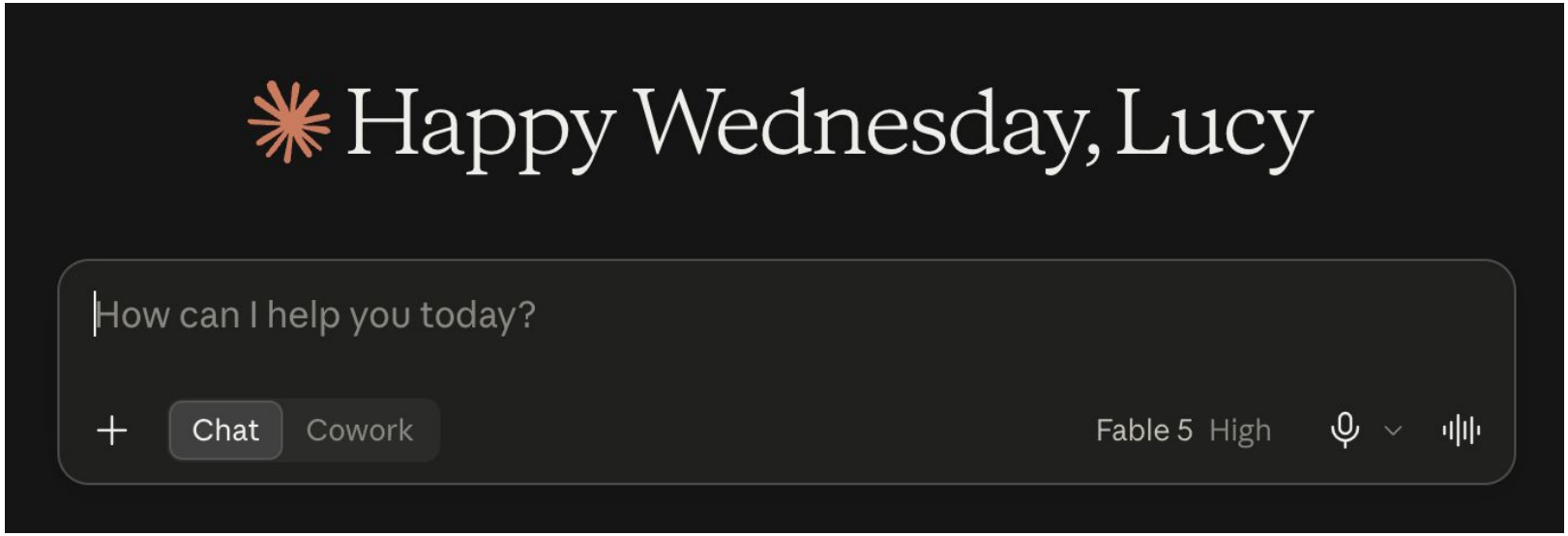
Language modeling is a **benchmark task** that helps us measure our progress on predicting language use

It's a **subcomponent** of many NLP tasks, especially those involving generation or estimating text probabilities:

predictive typing, speech recognition, handwriting recognition, spelling/grammar correction, authorship identification, machine translation, summarization, dialogue, etc …

Everything in NLP is now built as a **language modeling** task

Everyday examples



Why?

A strong language model is strong at lots of intelligent tasks.

With next word prediction, we can do:

Madison has a population of _____ people. [knowledge]
I put ___ fork down on the table. [syntax]
The woman walked across the street, checking for traffic over ___ shoulder. [coreference]
I went to the ocean to see the fish, turtles, seals, and _____. [lexical semantics/topic]
Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___. [sentiment]
The chef dropped the shrimp into the fryer, and watched it ___. [commonsense reasoning]
I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____. [math]

How did we do this before deep learning?

`The students opened their ____`

Learn an *n*-gram language model!

> **Uni**grams : “the”, “students”, “opened”, “their”
> **Bi**grams: “the students”, “students opened”, “opened their”
> **Tri**grams: “the students opened”, “the students opened their”
> **4-**grams: “the students opened their”

Main idea: collect statistics about how frequent different *n*-grams are and use these to predict the next word.

Unigram LM

Independence assumption: $P(x_i \mid x_1, \ldots, x_{i-1}) \approx P(x_i)$

MLE of the dataset, where *c(x)* is the count of word *x*

$$P(x_i) = \frac{c(x_i)}{\sum_{\tilde{x}} c(\tilde{x})}$$

Unigram LM

A unigram LM trained on Shakespeare:

```
To him swallowed confess hear both . Which . Of save on trail for are ay device and rote life have
```

```
Hill he late speaks ; or ! a more to leg less first you enter
```

Higher-order n-gram LMs

Markov assumption:
Predicting the next word only depends on the previous *n*-1 words.

$$P\left(x_i \mid x_{i-1}, \ldots, x_1\right) \approx P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right)$$

For example, for a **bigram LM**:

$$P\left(x_i \mid x_{i-1}, \ldots, x_1\right) \approx P\left(x_i \mid x_{i-1}\right)$$

To calculate these probabilities, count them in some large dataset:

$$P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) \approx \frac{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1}\ x_i)}{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1})}$$

Bigram LM example

Given a training corpus of three sentences:

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

Compute bigram probabilities by counting:

$$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = 0.67 \qquad P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = 0.33 \qquad P(\text{am} \mid \text{I}) = \frac{2}{3} = 0.67$$

$$P(\text{</s>} \mid \text{Sam}) = \frac{1}{2} = 0.50 \qquad P(\text{Sam} \mid \text{am}) = \frac{1}{2} = 0.50 \qquad P(\text{do} \mid \text{I}) = \frac{1}{3} = 0.33$$

Probability of a sentence = product of all bigram probabilities in the sentence.

$$P(\text{<s> I am Sam </s>}) = \left(\frac{2}{3}\right)\left(\frac{2}{3}\right)\left(\frac{1}{2}\right)\left(\frac{1}{2}\right) = \frac{1}{9} = 0.\bar{1}$$

A comparison of *n*-gram LMs

A **unigram** LM trained on Shakespeare:

```
To him swallowed confess hear both . Which . Of save on
trail for are ay device and rote life have
```

```
Hill he late speaks ; or ! a more to leg less first you
enter
```

A **4-gram** LM trained on Shakespeare:

```
King Henry. What! I will go seek the traitor Gloucester.
Exeunt some of the watch. A great banquet serv’d in;
```

```
It cannot be but so.
```

So who even uses *n*-gram LMs today?

Not very many people, but …

Infini-gram (Liu et al. 2024) is a **modernized** *n*-gram LM, or a **∞-gram LM**, that is

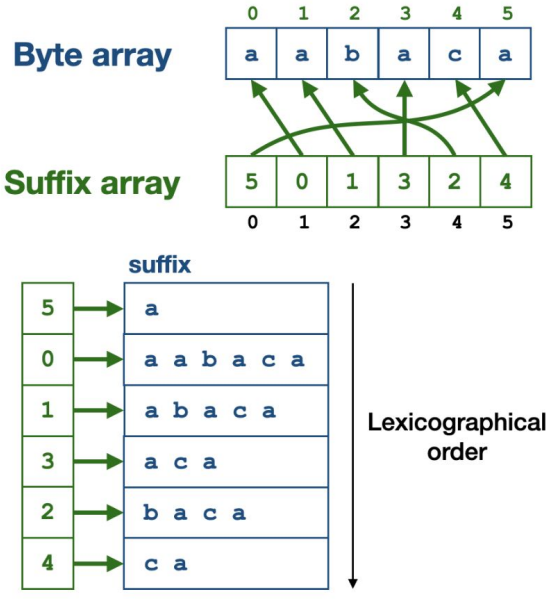
- trained on 5 trillion tokens
- contains *n*-gram counts for ~5 quadrillion (a.k.a. 5 thousand trillion) unique *n*-grams.

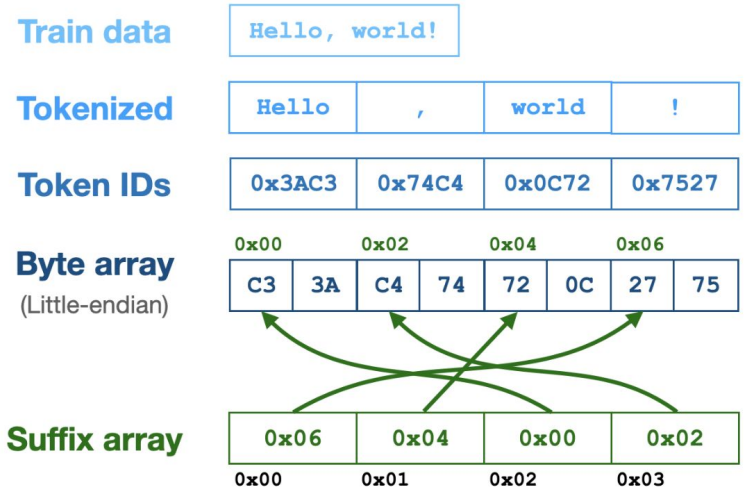
This *n*-gram LM is made possible by ***backoff*** and a special data structure. It can:

- compute *n*-gram LM probabilities and next-token distributions (like classic *n*-gram LMs)
- search for docs that contain specific *n*-grams
  - We can trace model responses back to snippets of training data!

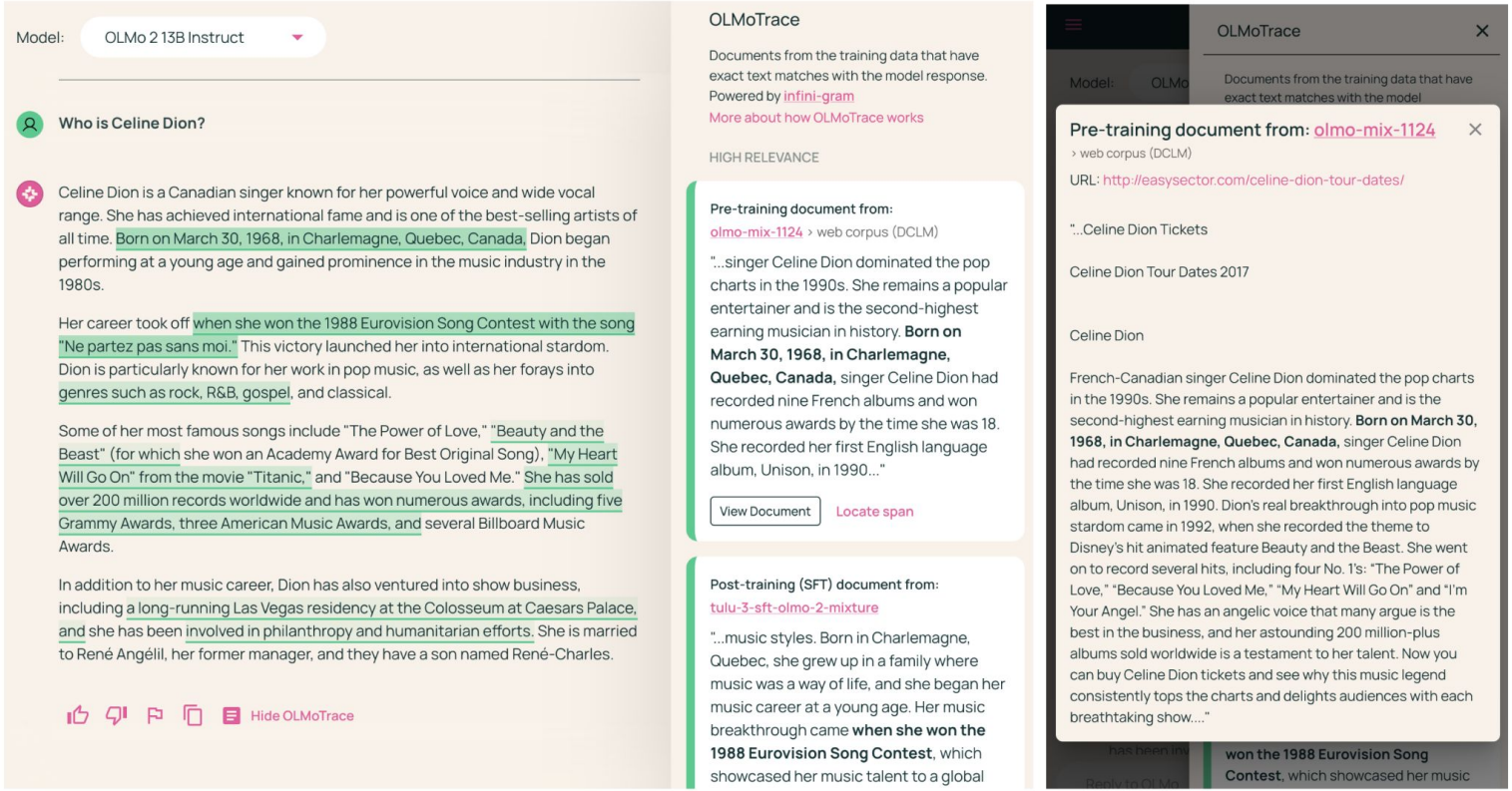
So who even uses *n*-gram LMs today?

Infini-gram (Liu et al. 2024) is implemented using a *suffix array*





Downstream uses of infini-gram



Problems with classic *n*-gram language models

Sparsity issues

What if the phrase in the numerator never occurs in the data?

The probability of valid sequences at test time would become 0!

$$P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) \approx \frac{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1}\ x_i)}{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1})}$$

Sparsity gets worse with higher *n*

Problems with classic *n*-gram language models

Sparsity issues

What if the phrase in the numerator never occurs in the data?

Smoothing: Add a small amount to the count for every word.

This is a type of *regularization*.

$$P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) \approx \frac{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1}\ x_i)+\delta}{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1})+|V|\delta}$$

Problems with classic *n*-gram language models

Sparsity issues

What if the denominator never occurs in the data?

Then we divide by zero!

$$P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) \approx \frac{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1}\ x_i)}{\text{count}(x_{i-n+1} \ldots x_{i-2}\ x_{i-1})}$$

Problems with classic *n*-gram language models

Sparsity issues

What if the denominator never occurs in the data?

Backoff: condition on $x_{i-1}, \ldots, x_{i-n+2}$ instead

$$P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) \approx P\left(x_i \mid x_{i-1}, \ldots, x_{i-n+2}\right)$$

Need to discount the higher-order *n*-gram to prevent sum(prob) > 1

Problems with classic *n*-gram language models

Sparsity issues

Interpolation: mix unigram, bigram, trigram, etc

$$\hat{P}(x_i \mid x_{i-1}, x_{i-2}) = \lambda_1 P(x_i \mid x_{i-1}, x_{i-2}) + \lambda_2 P(x_i \mid x_{i-1}) + \lambda_3 P(x_i)$$

$$\sum_i \lambda_i = 1$$

Choose λs to maximize probability of held-out data:
1) Calculate *n*-gram probs on training data
2) Search for λs that give largest prob for heldout set

Problems with classic *n*-gram language models

No notion of semantic similarity

She bought a car

She purchased a bicycle

Cannot handle long-distance dependencies

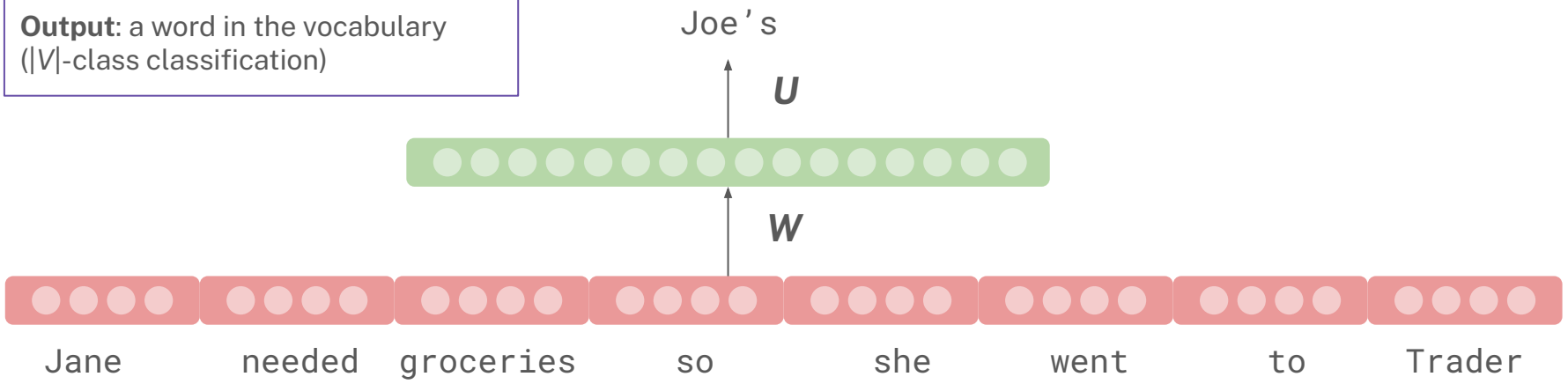
For tennis class he wanted to buy his own racquet

For programming class he wanted to buy his own computer

Fixed window neural LMs

A window-based neural model turns word prediction into **discriminative text classification**

Output: a word in the vocabulary (|*V*|-class classification)



Input: *n*-1 previous words as context

This course is not sponsored by Trader Joe's (yet)

Fixed window neural LMs

We replace the count table with a neural network to approximate the next-word prediction probability

> θ are the weights (e.g. W, U) of our model

$$P_\theta\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right) = \text{Softmax}\left(\text{NeuralNet}\left(x_{i-n+1}, \ldots, x_{i-1}\right)\right)$$

We can use SGD to optimize the network by minimizing the **average negative log-likelihood** of next word prediction

> We sample pairs of (prefix context, next word) from a large dataset

$$\mathcal{L}(\theta) = -\mathbb{E}_{x_i \mid x_{<i}} \log P_\theta\left(x_i \mid x_{i-1}, \ldots, x_{i-n+1}\right)$$

Fixed window neural LMs

Produce an **output distribution**

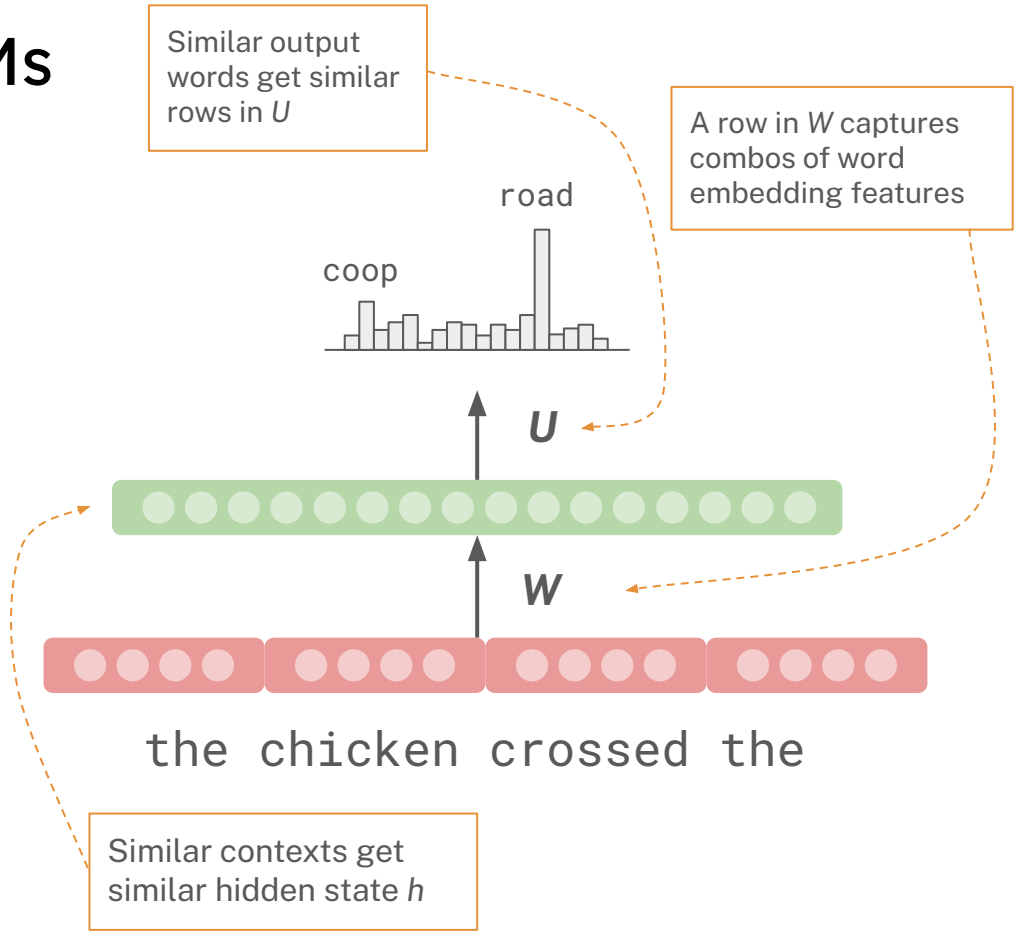
$$\hat{y} = \text{softmax}\left(Uh + b_2\right) \in \mathbb{R}^{|V|}$$

Hidden layer

$$h = f\left(We + b_1\right)$$

Concatenated word embeddings

$$e = [e_1; e_2; e_3; e_4]$$

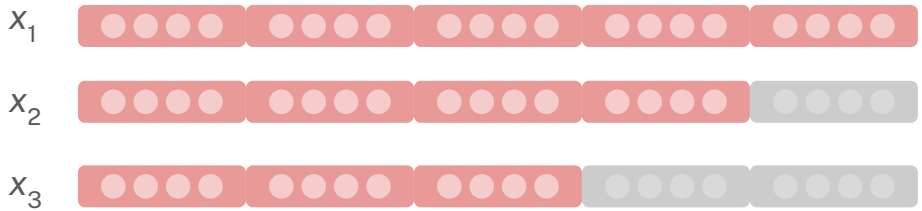


What if we have variable-sized inputs?

Let's say we have a window of size 5, but some of our input sequences are shorter than that.

x_1 = `Alice went the store.`	5 tokens
x_2 = `Bob ate cheese.`	4 tokens
x_3 = `Joe laughed.`	3 tokens

Solution: zero-pad up to the length of the longest sequence



Problems solved

- A neural LM learns to **count** *and* it learns to **compress**
  - Don’t need to store all observed *n*-grams
    - Only need to store model parameters (weights)
  - No sparsity issues!
- Neural LMs can memorize texts and also **generalize**
  - Can share strength among similar words, and enforce implicit similarity within the model
    - Similar text inputs produce similar hidden representations
    - Similar text outputs come from similar hidden representations

Problems not solved

We still can't handle **long range dependencies**

> `For tennis class he wanted to get there early ... Later he stopped at the front desk to buy a new racquet`

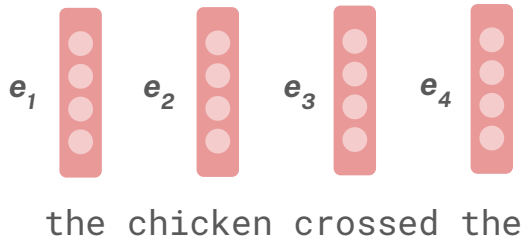
➔ A fixed window is **too small**
➔ Enlarging window enlarges *W*
➔ Each input embedding is multiplied by completely different weights in *W*; there's no sharing or symmetry in how input words are processed

We need a neural architecture that can process *any length input*!

RNNs

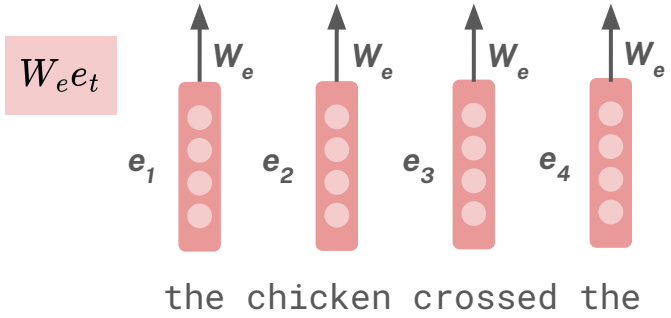
Recurrent Neural Networks

Word embeddings
For an input sequence of any length



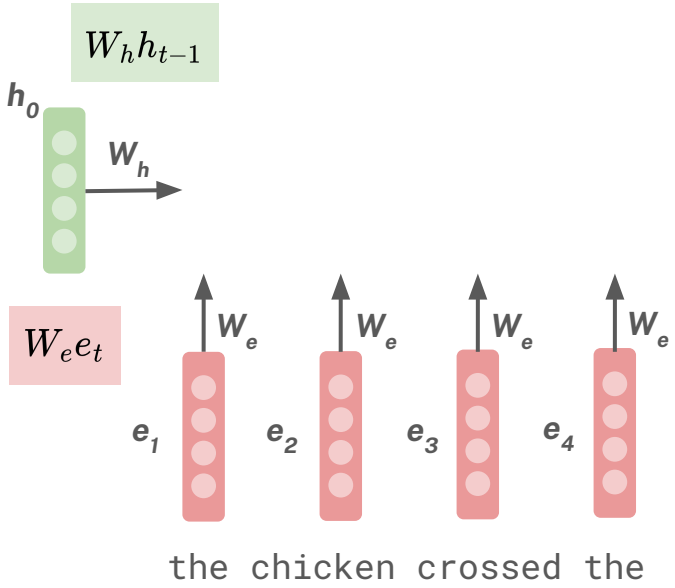
Recurrent Neural Networks

Word embeddings
For an input sequence of any length



Recurrent Neural Networks

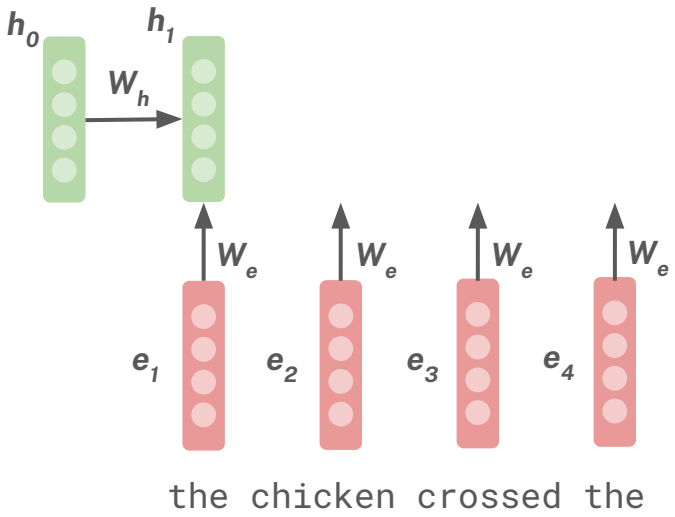
Word embeddings
For an input sequence of any length



Recurrent Neural Networks

$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

Word embeddings
For an input sequence of any length



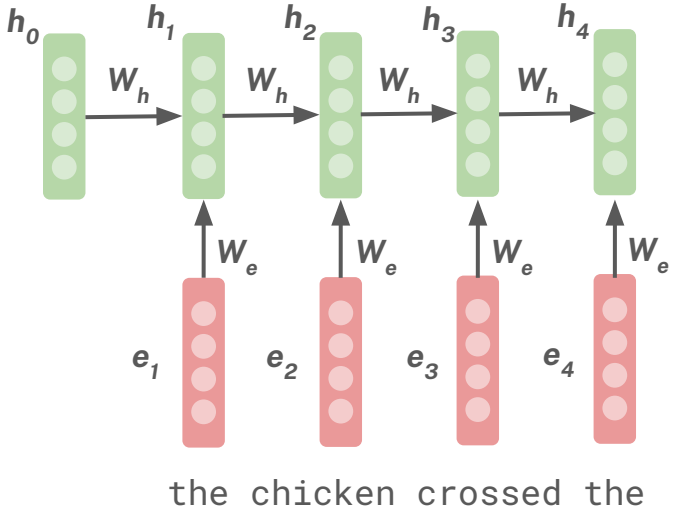
Recurrent Neural Networks

Hidden states

$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

Key idea: Apply the same weights W repeatedly

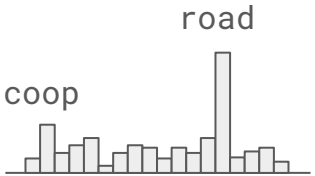
Word embeddings
For an input sequence of any length



Recurrent Neural Networks

Output probability distribution

$$\hat{y}_t = \text{softmax}\left(Uh_t + b_2\right) \in \mathbb{R}^{|V|}; \quad \hat{y}_4 = \text{P}(x_5 \,|\, \text{the chicken crossed the})$$

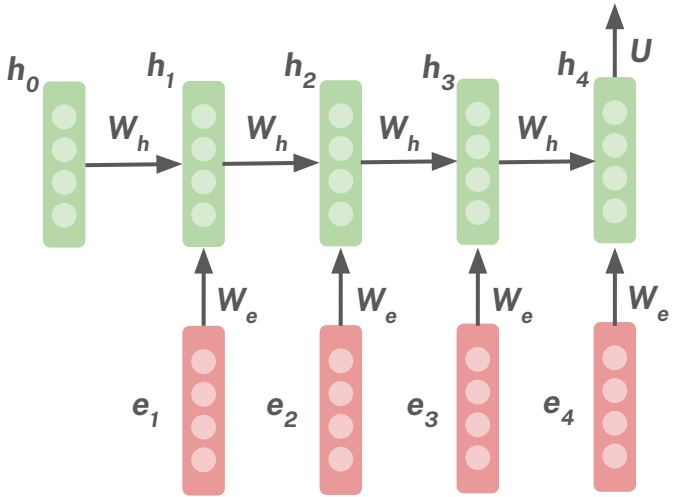


Hidden states

$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

Key idea: Apply the same weights *W* repeatedly

Word embeddings
For an input sequence of any length

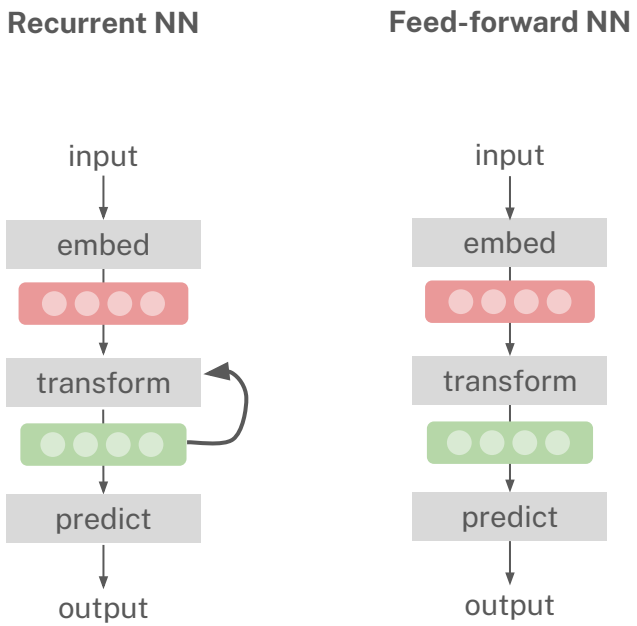


Another view of RNNs

"Remembers" information along the "time dimension"

RNNs are neural nets that share weights across layers, takes in an input at each layer, w/ a variable number of layers

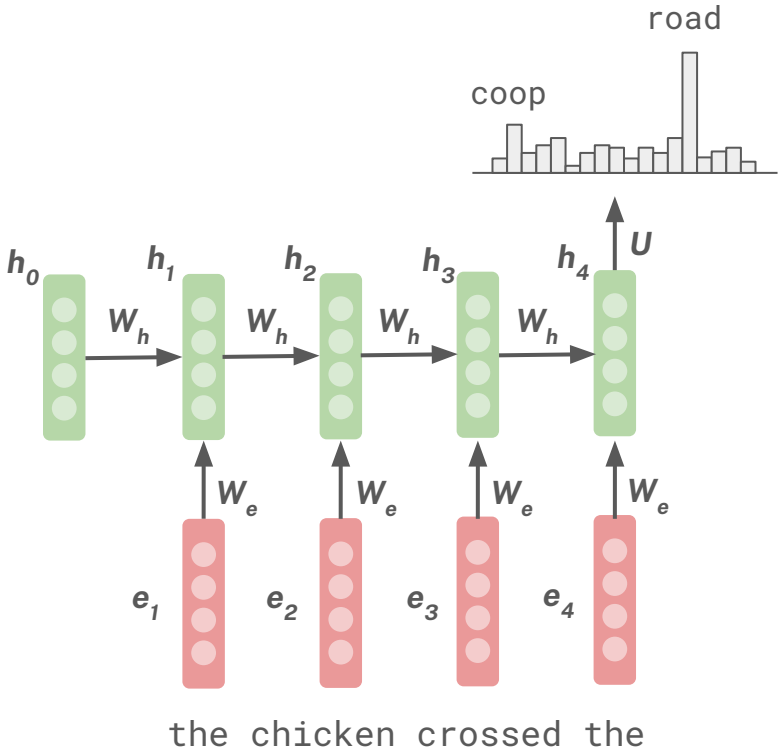
The *transform* step gets its own "previous" value as inputs



Recurrent Neural Networks

Advantages ✔

- Flexible for use with any input length
- Computation for step *t* can use information from multiple steps back
- Model size doesn't increase for longer inputs
- Same weights applied on every timestep, so there is symmetry in how inputs are processed

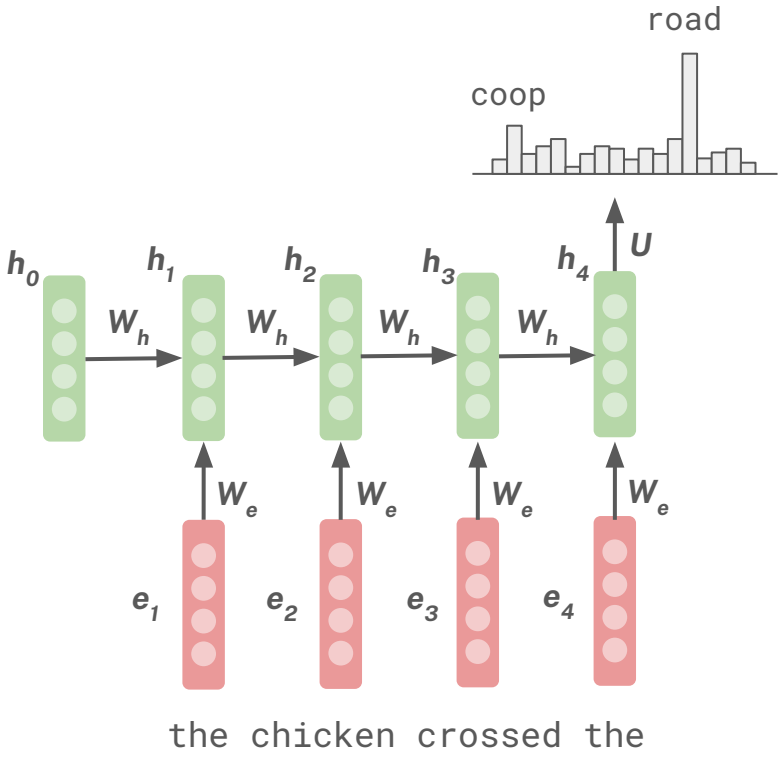


Recurrent Neural Networks

Disadvantages ✗

- Recurrent computation is slow
- In practice, difficult to access info from many steps back

More on these points later!



Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

> Let y_t be a one-hot vector representing the "true" distribution at that step and $\hat{y}_t$ is the predicted probability distribution:

$$J_t(\theta) = CE(y_t, \hat{y}_t)$$

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

> Let y_t be a one-hot vector representing the "true" distribution at that step and $\hat{y}_t$ is the predicted probability distribution:

$$J_t(\theta) = CE(y_t, \hat{y}_t) = -\sum_{w \in V} y_t(w) \log \hat{y}_t(w)$$

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

> Let y_t be a one-hot vector representing the "true" distribution at that step and $\hat{y}_t$ is the predicted probability distribution:

$$J_t(\theta) = CE(y_t, \hat{y}_t) = -\sum_{w \in V} y_t(w) \log \hat{y}_t(w)$$

$$y_t(w) = \begin{cases} 1 & w = x_{t+1} \\ 0 & w \neq x_{t+1} \end{cases}$$

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

> Let y_t be a one-hot vector representing the "true" distribution at that step and $\hat{y}_t$ is the predicted probability distribution:

$$J_t(\theta) = CE(y_t, \hat{y}_t) = -\sum_{w \in V} y_t(w) \log \hat{y}_t(w) = -\log \hat{y}_t(x_{t+1}) \qquad y_t(w) = \begin{cases} 1 & w = x_{t+1} \\ 0 & w \neq x_{t+1} \end{cases}$$

Training an RNN LM

Get a big dataset of text with a sequence of words $x_1, \ldots, x_t, x_{t+1}, \ldots, x_T$

Feed into an RNN LM, and compute output distributions at every step *t* for every word

At each step, calculate loss as the **cross-entropy** between the predicted probability distribution and the true next word x_{t+1}.

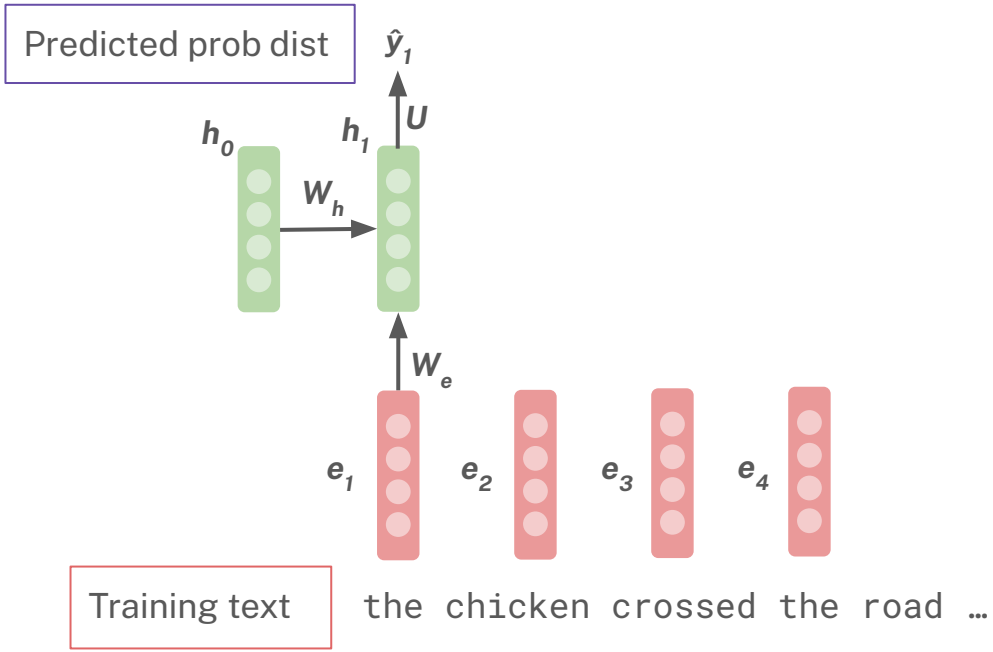
> Let y_t be a one-hot vector representing the "true" distribution at that step and $\hat{y}_t$ is the predicted probability distribution:

$$J_t(\theta) = CE(y_t, \hat{y}_t) = -\sum_{w \in V} y_t(w) \log \hat{y}_t(w) = -\log \hat{y}_t(x_{t+1}) \qquad y_t(w) = \begin{cases} 1 & w = x_{t+1} \\ 0 & w \neq x_{t+1} \end{cases}$$

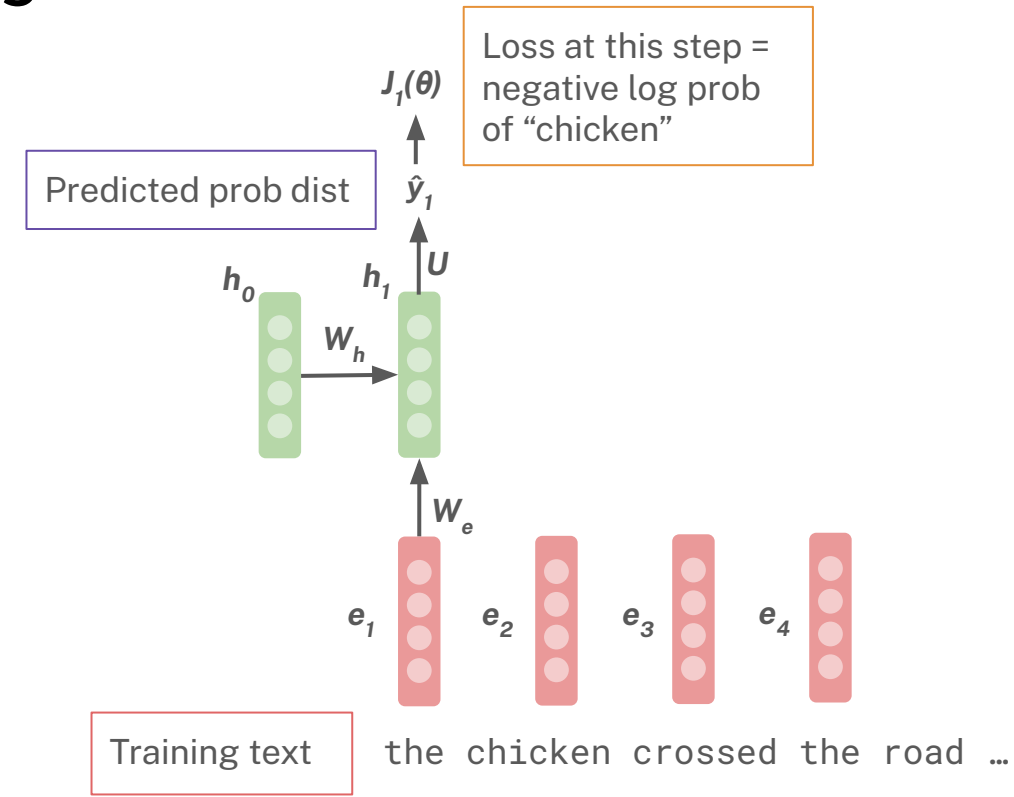
Average these losses to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{y}_t(x_{t+1})$$

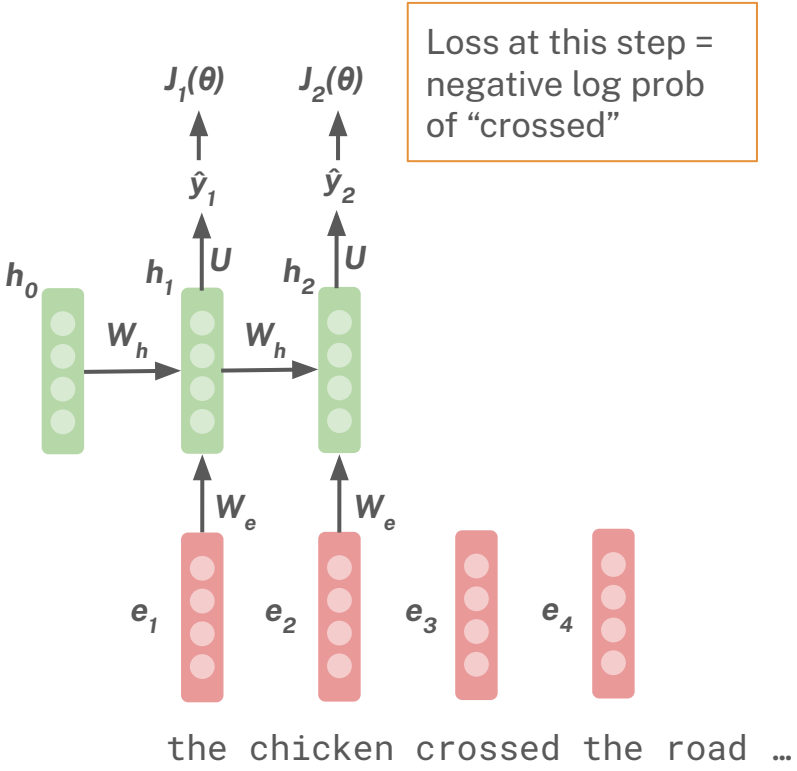
Training an RNN LM



Training an RNN LM

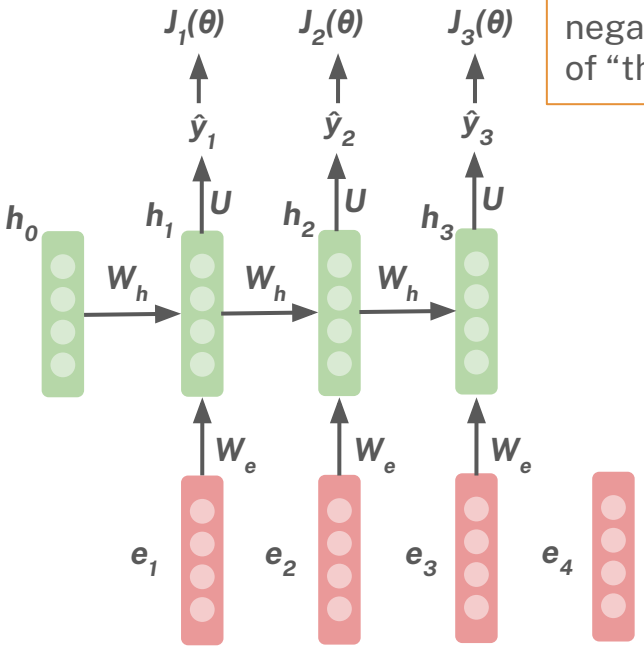


Training an RNN LM



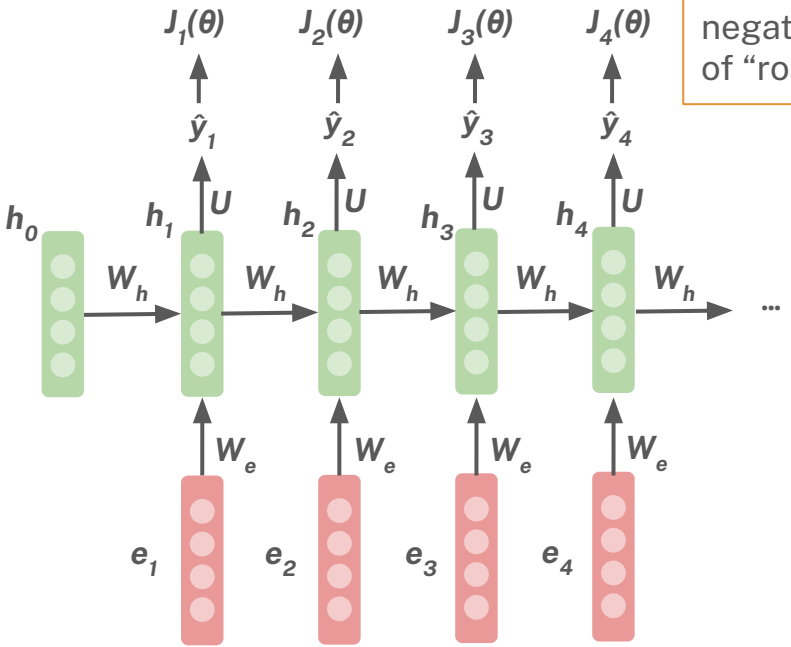
Loss at this step = negative log prob of “crossed”

Training an RNN LM



Loss at this step = negative log prob of “the”

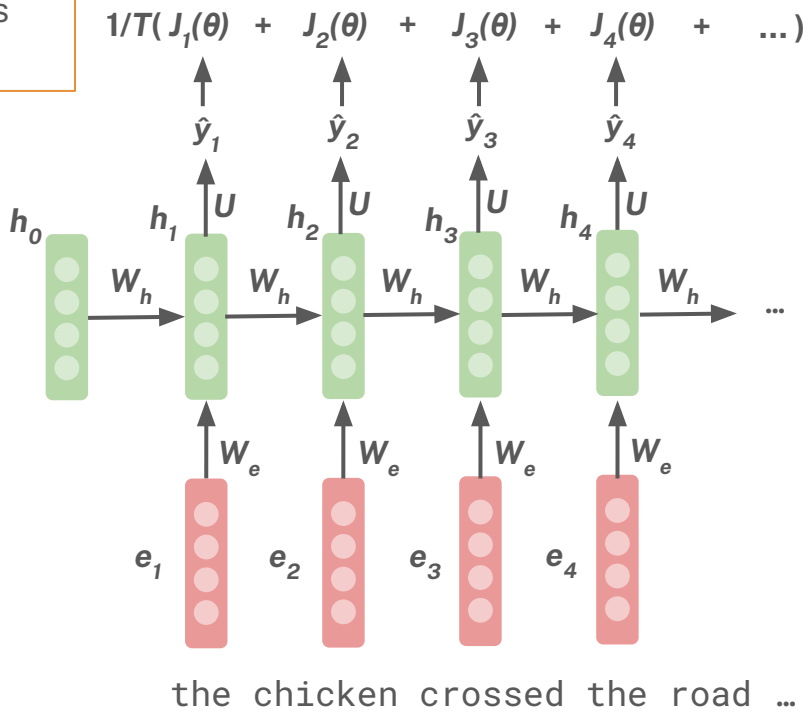
Training an RNN LM



Loss at this step = negative log prob of “road”

Training an RNN LM

Average the loss across all steps



Teacher forcing is a training technique where ground-truth from the training dataset is fed as the input for each time step, instead of the model's own prediction.

Training an RNN LM

Computing loss and gradients across an entire dataset at once is expensive, memory-wise.

Instead: compute loss for a **batch of sentences**, and then compute gradients, update our weights, and repeat.

Selecting the right batch size (when training any type of LM) is a trade-off b/t hardware memory, training speed, and model accuracy.

- **Too small**: noisy updates could help escape local minima, but trains slower, might underutilize compute
- **Too large**: fast & smooth updates, but memory hungry and may cause you to converge to suboptimal minima

Usually batch sizes are powers of two. Adjustments to batch size usually requires adjusting learning rate, too.

Backpropagation

Model needs to learn from its errors

➔ We need to update our weights based on our loss, using SGD

So, how do we compute the gradient of our loss function?

Multivariable chain rule

To calculate the derivative of a function composed of two intermediate functions:

$$\frac{d}{dt} f\left(x\left(t\right), y\left(t\right)\right) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

When a variable influences the output through several paths, the total derivative is the sum of the derivatives along each path.

In an RNN, to update $\boldsymbol{W_h}$, we need to compute $\frac{\partial J}{\partial W_h}$.

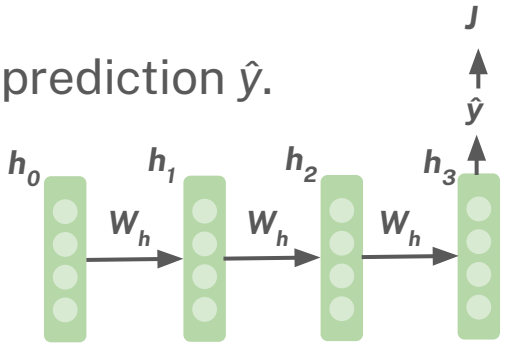
$\boldsymbol{W_h}$ is shared across “time”, influencing the output via multiple paths, at every time step. It appears in the intermediate computation of each hidden state:

$$h_t = \sigma\left(W_h h_{t-1} + W_e e_t + b_1\right)$$

Applying the multivariable chain rule

The rule tells us: computing $\frac{\partial J}{\partial W_h}$ means we should sum up the derivatives from each computation $\boldsymbol{W_h}$ is involved in.

Example: let's say $t = 3$, where h_3 is used to make our prediction $\hat{y}$.



$\boldsymbol{W_h}$ influences J via three paths:

- Used to compute $h_3 \rightarrow$ affects J
- Used to compute $h_2 \rightarrow$ affects $h_3 \rightarrow$ affects J
- Used to compute $h_1 \rightarrow$ affects $h_2 \rightarrow$ affects $h_3 \rightarrow$ affects J

Applying the multivariable chain rule

Example: let's say $t = 3$, where h_3 is used to make our prediction $\hat{y}$.

$\boldsymbol{W_h}$ influences J via three paths, which get summed up based on the rule:

$$\frac{\partial J_3}{\partial W_h} = \frac{\partial J_3}{\partial h_3}\frac{\partial h_3}{\partial W_h} + \frac{\partial J_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_h} + \frac{\partial J_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_h}$$

Applying the **chain rule** along this path: How h_3 changed the loss, times how h_2 changed h_3, times how h_1 changed h_2, times how $\boldsymbol{W_h}$ changed h_1.

General formula:

$$\sum_{k=1}^{t} \frac{\partial J}{\partial h_t}\left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_k}{\partial W_h}$$

In practice, backprop is "truncated" after ~20 timesteps for training efficiency

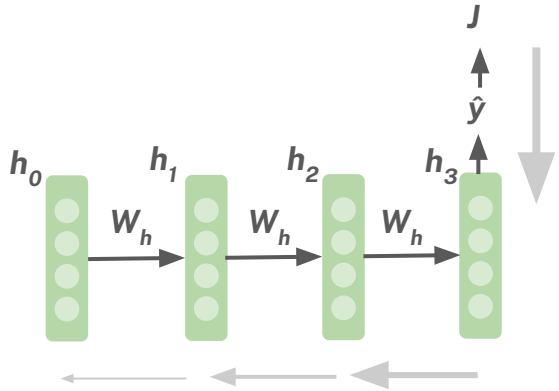
Vanishing gradients

What happens if the values in this product are small?

$$\sum_{k=1}^{t} \frac{\partial J}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W_h}$$

With non-linearities from RNNs' activation functions (e.g. tanh), it's easy to get derivatives less than 1.

The more small values we multiply, and the further we backprop, the gradient signal gets small and smaller!



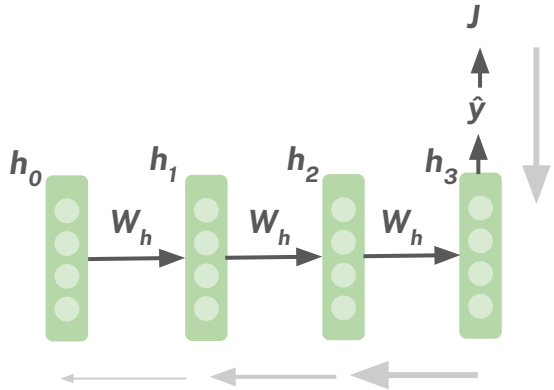
Vanishing gradients

What happens if the values in this product are small?

$$\sum_{k=1}^{t} \frac{\partial J}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W_h}$$

With non-linearities from RNNs' activation functions (e.g. tanh), it's easy to get derivatives less than 1.

This means model params are updated only with respect to near effects, rather than long-term effects!



Effect of vanishing gradient on RNN LM

When she tried to print her **tickets**, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____

To learn from this training example, the LM needs to model the dependency between *tickets* at the 7th step and *tickets* at the end.

But if the gradient is small, the LM can't learn this dependency

➔ So it's unable to predict dependencies like this at test time.

How to fix vanishing gradients

It's difficult for the RNN to learn to preserve information over many timesteps.

This is because the hidden state is constantly being rewritten.

Example fix: an RNN architecture with a "memory" component

➔ Long short-term memory (LSTM) [link]

Another fix: create more direct and linear pass-through connections

➔ Residual connections, attention, etc

Exploding gradients

If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta_{new} = \theta_{old} - \alpha \nabla_\theta J(\theta)$$

learning rate

gradient

This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration, w/ a large loss.

You think you've found a hill to climb, but suddenly you're in Iowa!

In the worse case, this will result in *inf* or *nan* in your network and then you have to restart training from an earlier checkpoint.

How to fix exploding gradients

Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

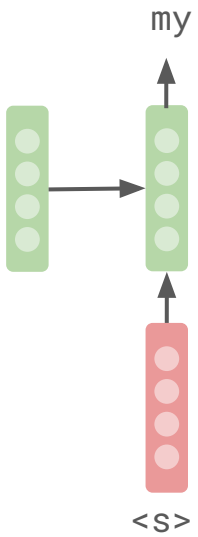
Pascanu et al. 2012

Take a step in the same direction, but a smaller step

Remembering to clip gradients is important, but exploding gradients have a simple solution!

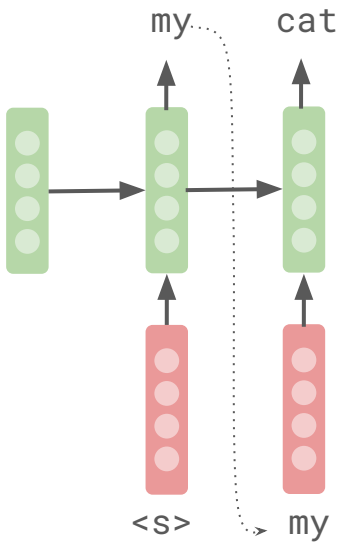
Generating roll outs

Just like with an *n*-gram language model, you can generate text by repeated sampling, in an **autoregressive** way. Sampled output = next step's input.



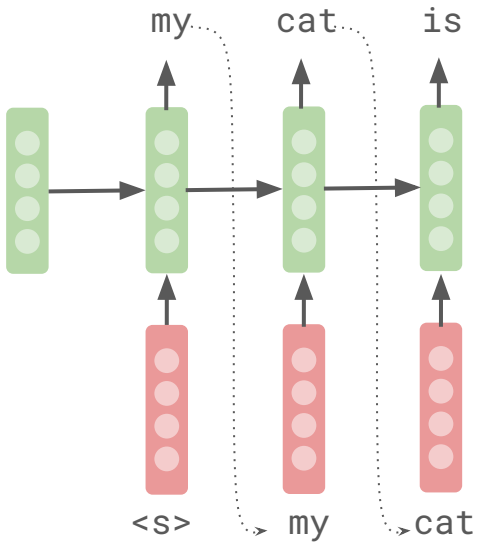
Generating roll outs

Just like with an *n*-gram language model, you can generate text by repeated sampling, in an **autoregressive** way. Sampled output = next step's input.



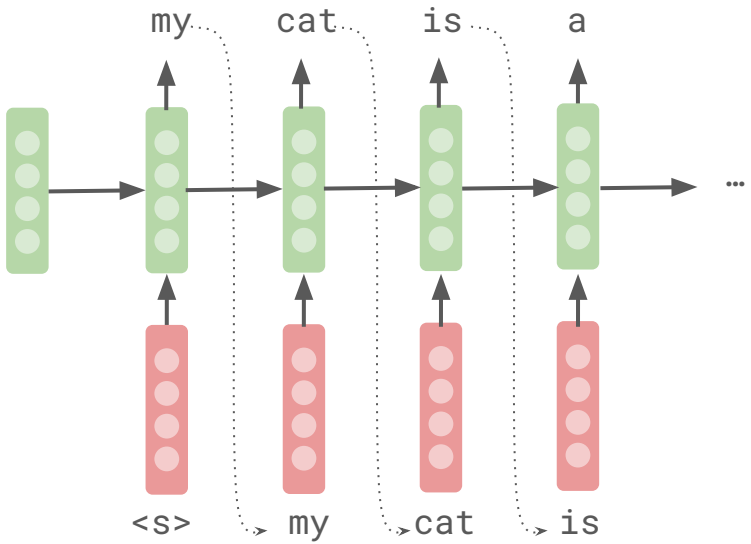
Generating roll outs

Just like with an *n*-gram language model, you can generate text by repeated sampling, in an **autoregressive** way. Sampled output = next step's input.



Generating roll outs

Just like with an *n*-gram language model, you can generate text by repeated sampling, in an **autoregressive** way. Sampled output = next step's input.



Generating roll outs

Train an RNN on recipes, and then generate (source):

```
    Title: CHOCOLATE RANCH BARBECUE
    Categories: Game, Casseroles, Cookies, Cookies
    Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten

  Place each pasta over layers of lumps. Shape mixture into the moderate oven
and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

  Combine the cheese and salt together the dough in a large skillet; add the
ingredients and stir in the chocolate and pepper.
```

Evaluation

The standard evaluation metric for language models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}\left(x_{t+1} \mid x_t, \ldots, x_1\right)} \right)^{1/T}$$

Geometric mean, so it's normalized by the # of words

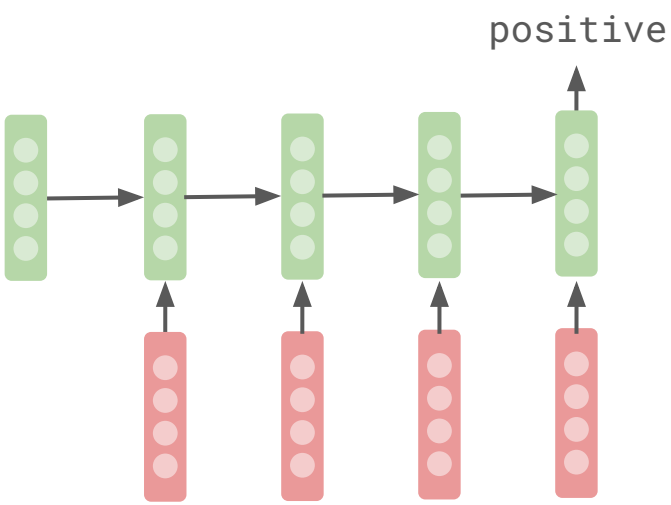
Inverse probability of dataset, according to the language model

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{y}_t\left(x_{t+1}\right)} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log y_t\left(\hat{x}_{t+1}\right) \right) = \exp\left(J\left(\theta\right)\right)$$

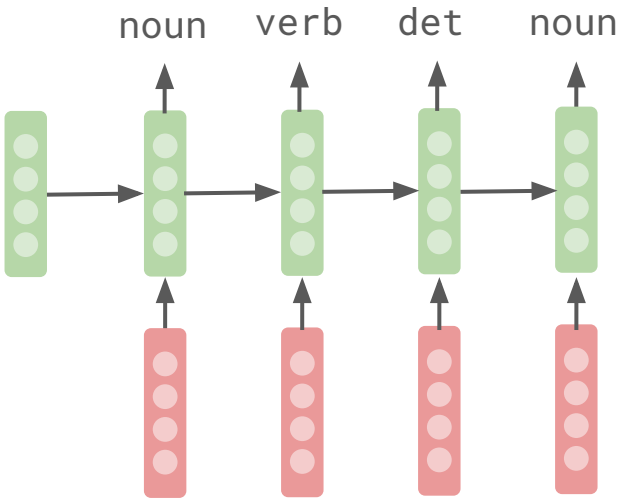
Exponential of the cross-entropy loss!

Lower perplexity is better.

Can use RNNs for other tasks too!



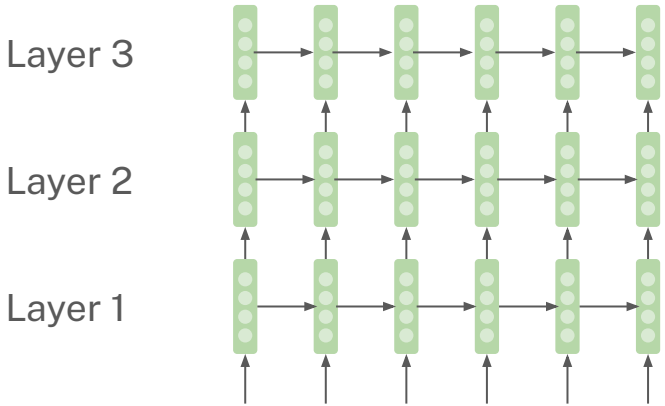
Encoding sentences for sentence classification. Can also use hidden states to get sentence embeddings!



Tagging sentences with labels for each word, e.g. part-of-speech, named entity recognition.

Back in my day, we RNN'd in all sorts of ways

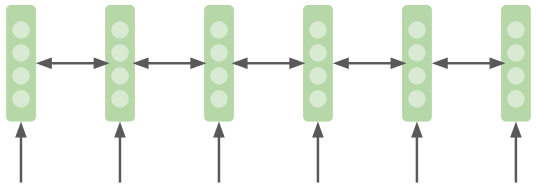
Multilayer RNNs: hidden states from layer *i* are inputs to layer *i* + 1.



Transformers (later lecture) can be up to 24 layers

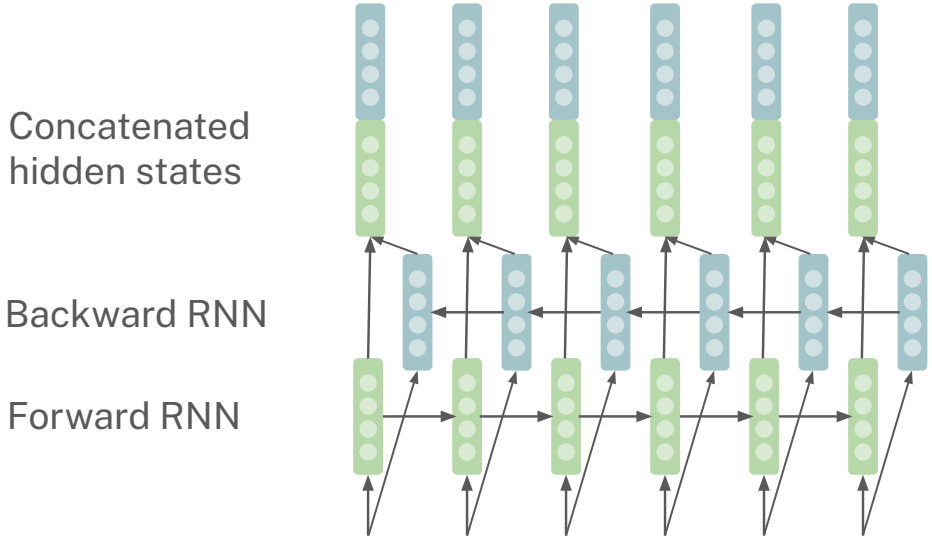
Back in my day, we RNN’d in all sorts of ways

Bidirectional RNNs



Back in my day, we RNN'd in all sorts of ways

Bidirectional RNNs



Question

Can we use bidirectional RNNs in the following tasks?

Text classification, sequence tagging, text generation

(a) Yes, Yes, Yes
(b) Yes, No, Yes
(c) Yes, Yes, No
(d) No, Yes, No

RNNs summary

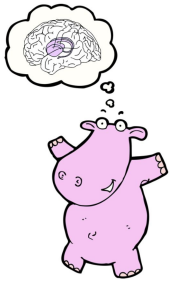
Advantages

- Can process any length input
- Computation for each step can (in theory) use info from many steps back
- Model size doesn’t increase for longer input context

Disadvantages

- Recurrent computation is slow and can’t parallelize
- In practice, difficult to access information from many steps back

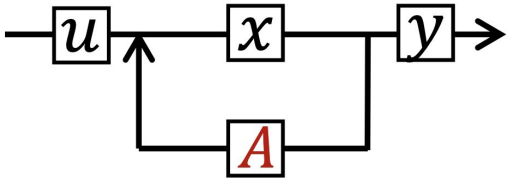
Many RNN variants!



We won't dive into a history lesson on LSTMs, GRUs, etc, but...

State-space models are modern RNNs

- Linear recurrent update
- Long-range with large hidden units
- Efficiency: parallelizable on GPUs
- "A refreshing departure from Transformers"
- Mamba is an example SSM



$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

From the S4 paper

Seq2seq Task Example: Machine Translation

Multilinguality

There are over 7,000 documented languages in the world!

You can browse languages of the world on Ethnologue, from ’Auhelawa to Zuni.

To facilitate communication among people, *translation* is needed.

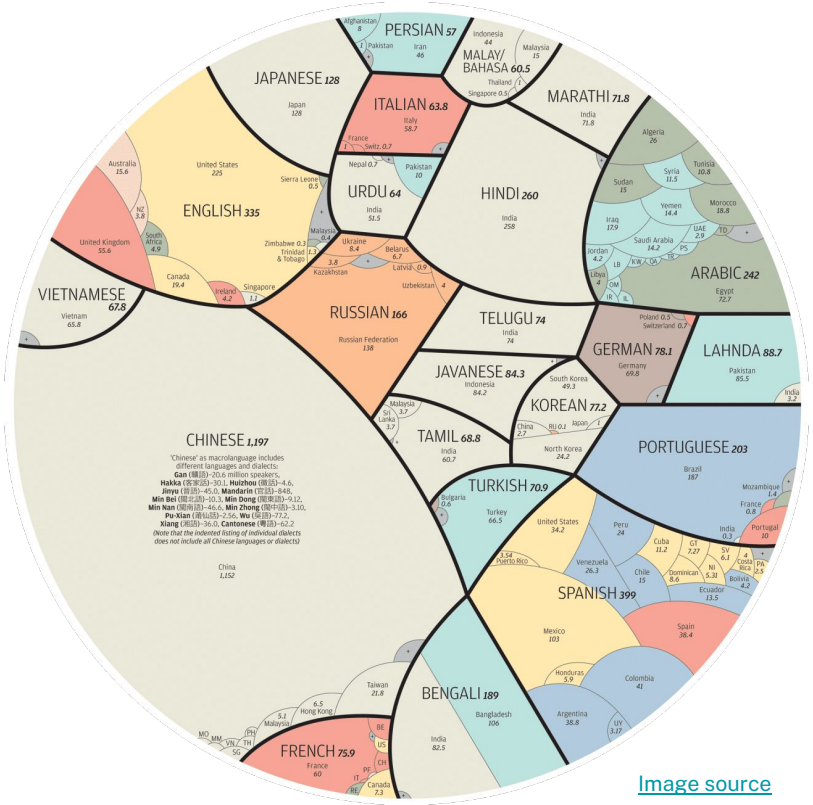


Image source

Machine translation

Goal: translate a sentence in a source language (input) to a sentence in a target language (output)

I like apples ↔ ich mag Äpfel (German)

For awhile, MT was one of the “holy grail” problems in AI

Still very challenging for low-resource languages

Machine translation

MT is challenging because

- Single words may be replaced with multi-word phrases:

  I like **apples** ↔ J'aime **les pommes** (French)

- Phrases can be reordered

  I like **red** apples ↔ J'aime les pommes **rouges** (French)

- Some translations are context-dependent

  les ↔ **the** but **les** pommes ↔ apples

Extremely large output space! Decoding is NP-hard.

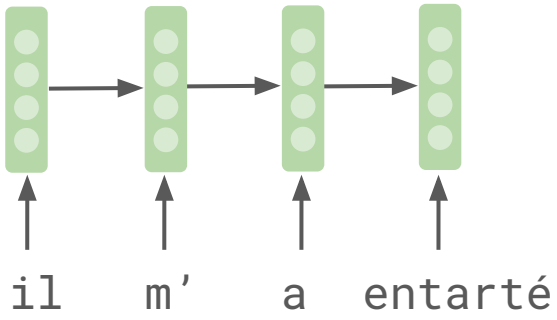
Neural machine translation

NMT was a **fringe research attempt** in 2014 and became a **leading standard method** in 2016.

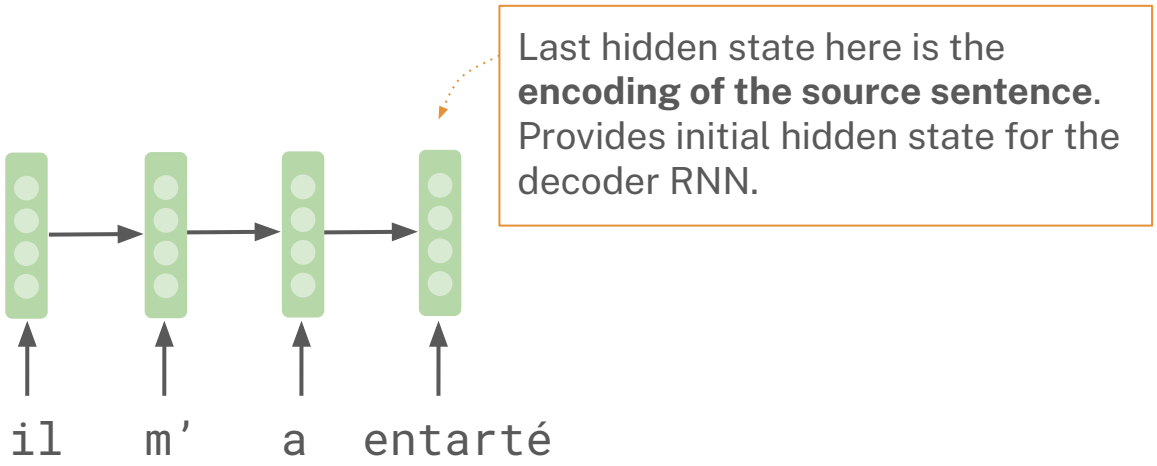
- 2014: first seq2seq paper published (Sutskever et al. 2014)
- 2016: Google Translate switches from statistical machine translation (SMT) to NMT, and by 2018 everyone else (Microsoft, Facebook, Baidu, Tencent, etc) has as well

SMT systems, build by **hundreds of engineers over many years**, were outperformed by NMT systems trained by **small groups of engineers in a few months**.

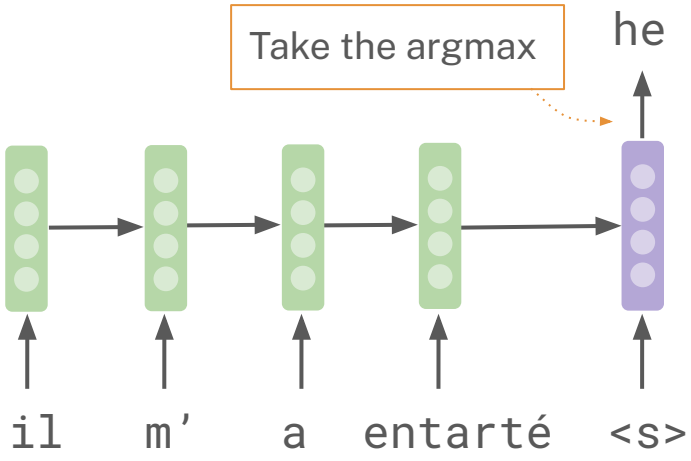
Neural machine translation



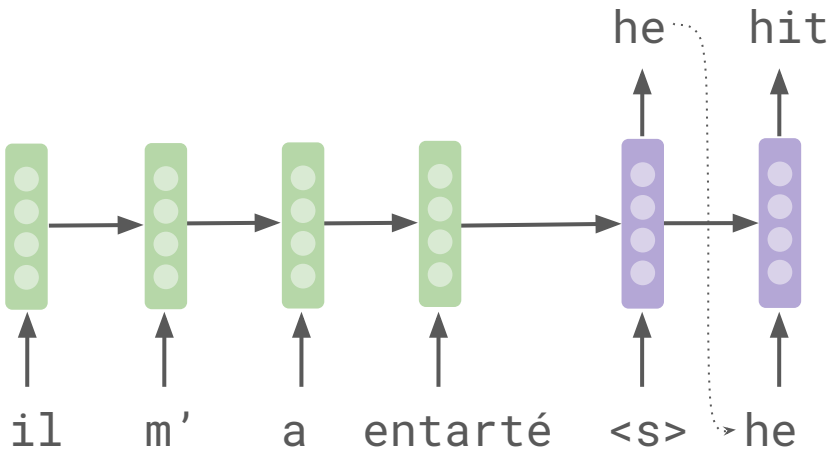
Neural machine translation



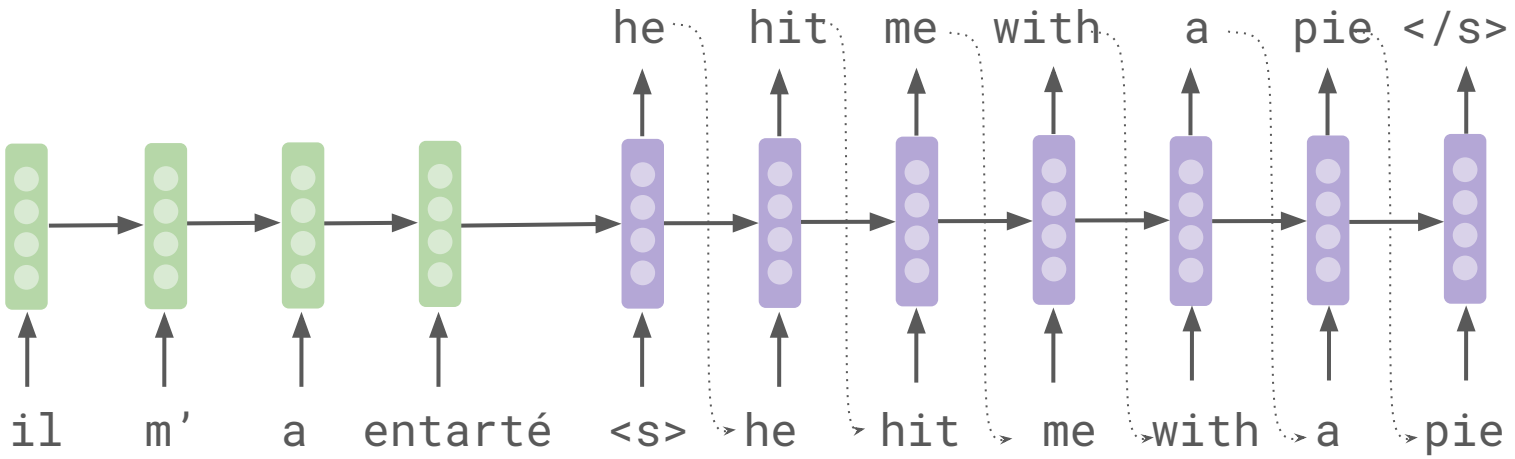
Neural machine translation



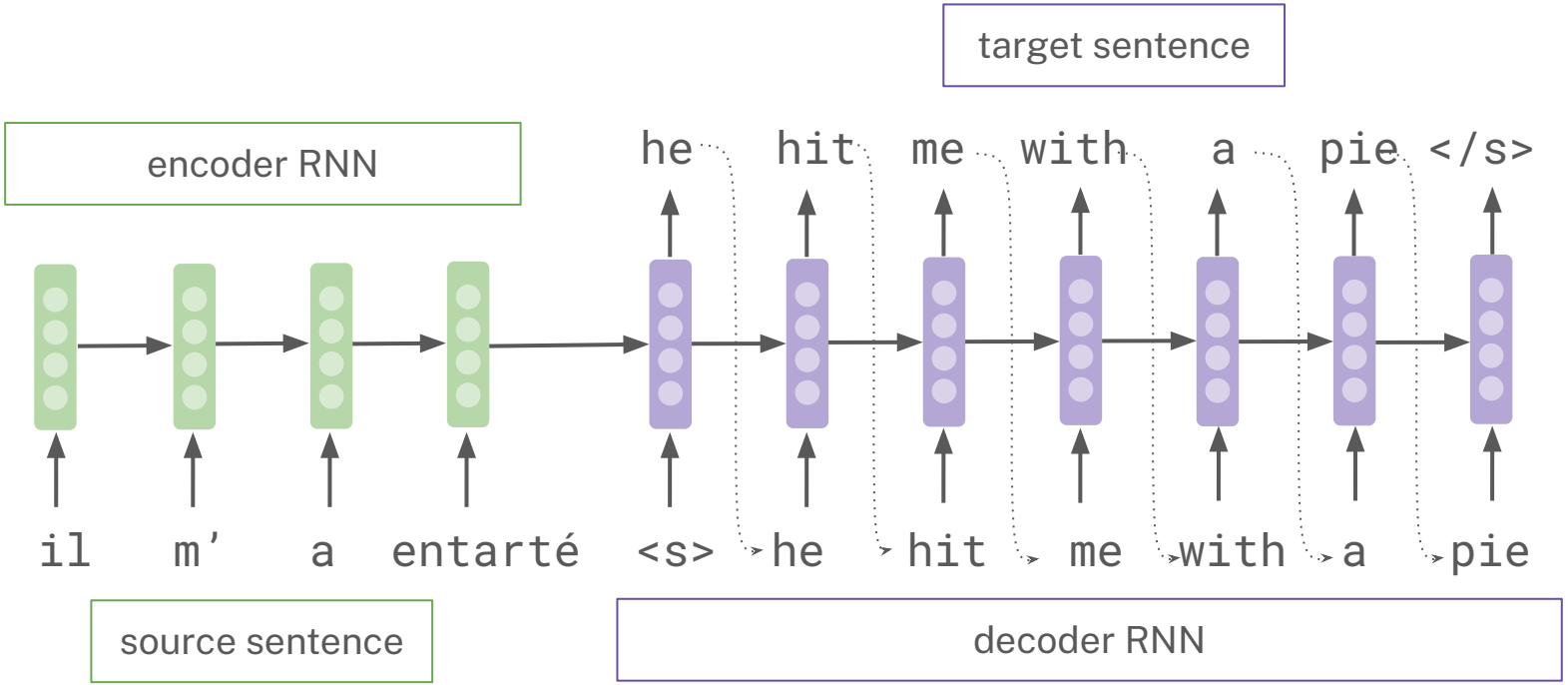
Neural machine translation



Neural machine translation

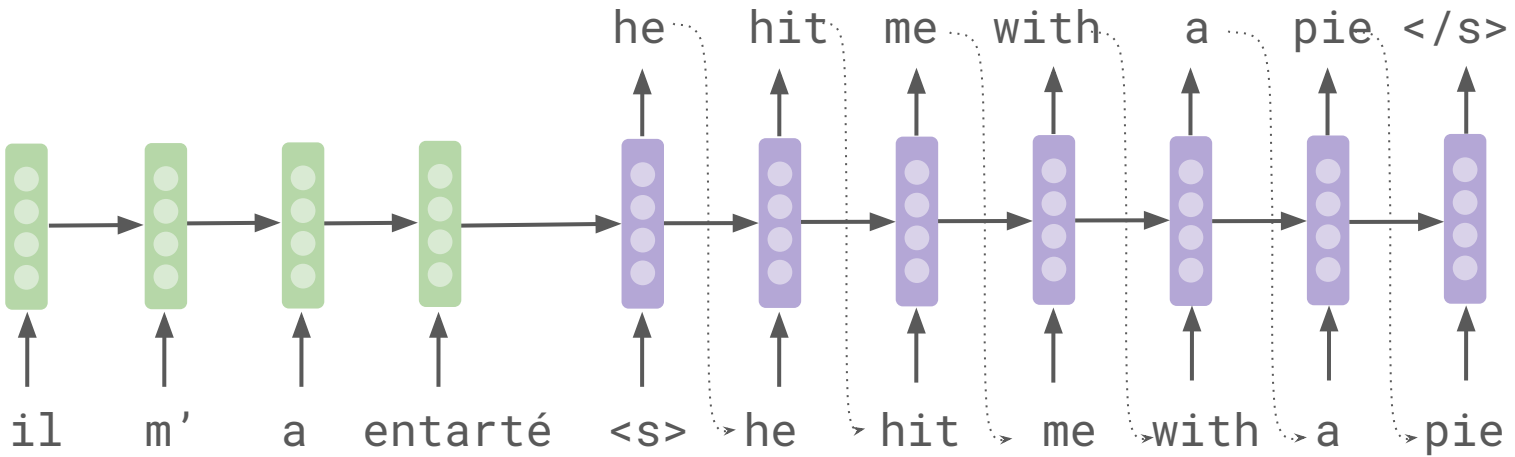


Neural machine translation



A sequence-to-sequence model!

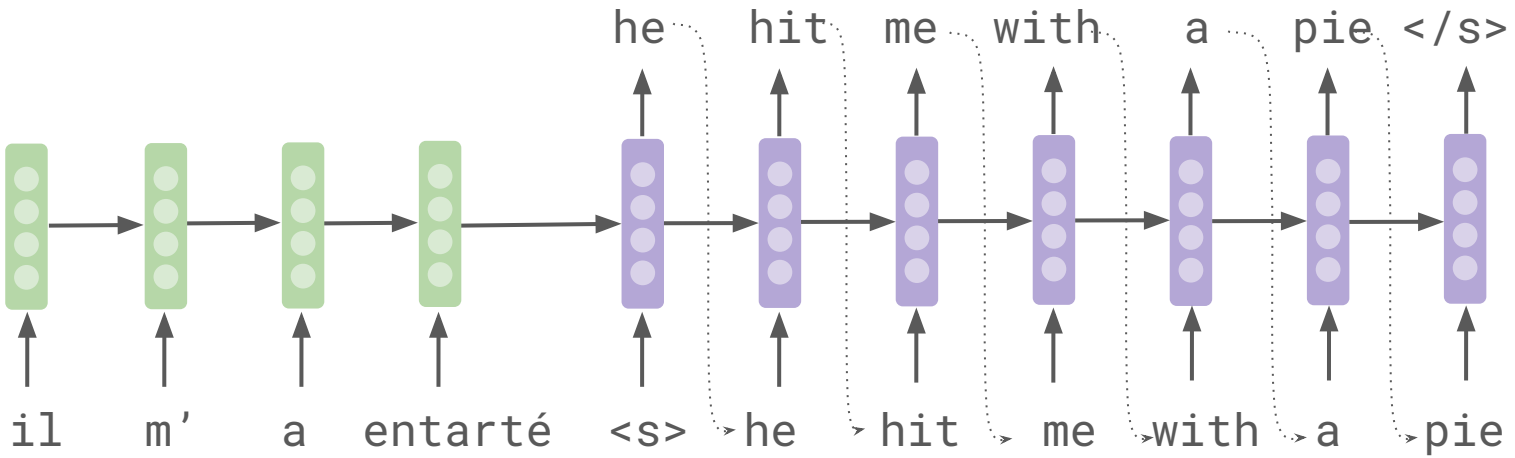
Neural machine translation



Encoder RNN inputs **word embeddings** for the source language

Decoder RNN inputs **word embeddings** for the target language

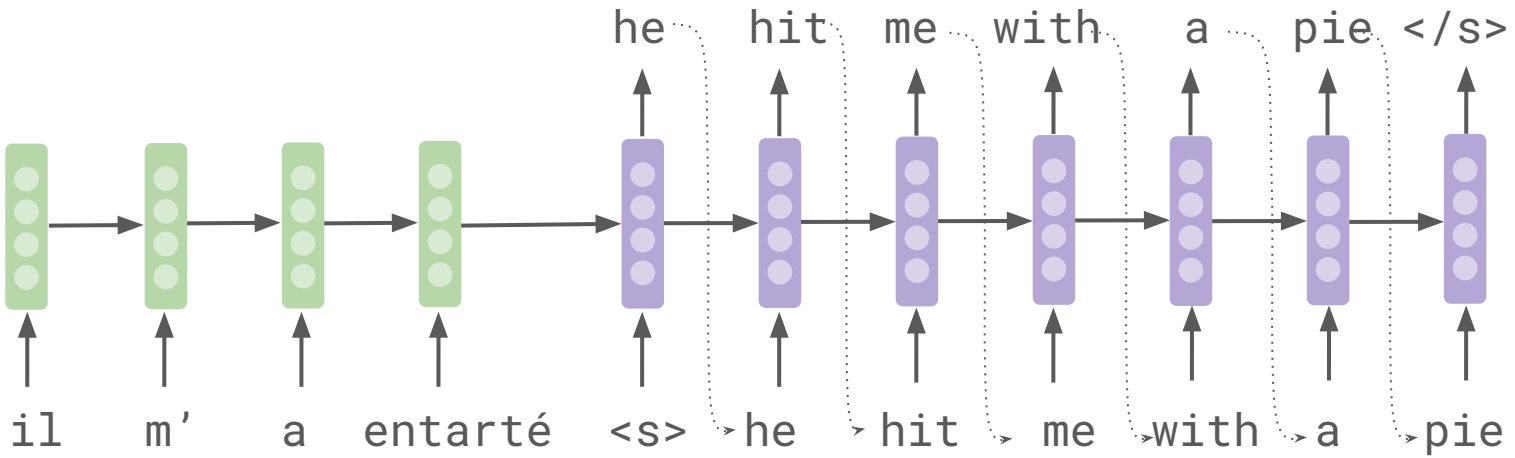
Neural machine translation



Encoder RNN has a set of parameters e.g. ***W***, ***U***, ***b***.

Decoder RNN has a separate set of parameters e.g. ***W’***, ***U’***, ***b’***.

Neural machine translation



Encoder RNN can be bidirectional

Decoder RNN has to be unidirectional

Neural machine translation

The seq2seq model is an example of a **conditional language model**

- Language model because the decoder is predicting the next word of the target sentence *y*
- Conditional because its predictions are also conditioned on the source sentence *x*

$$P(y \mid x) = P(y_1 \mid x)P(y_2 \mid y_1 x) \dots P(y_T \mid y_1, \dots, y_{T-1}, x)$$

Probability of the next target word, given target words so far and source sentence *x*

Training NMT

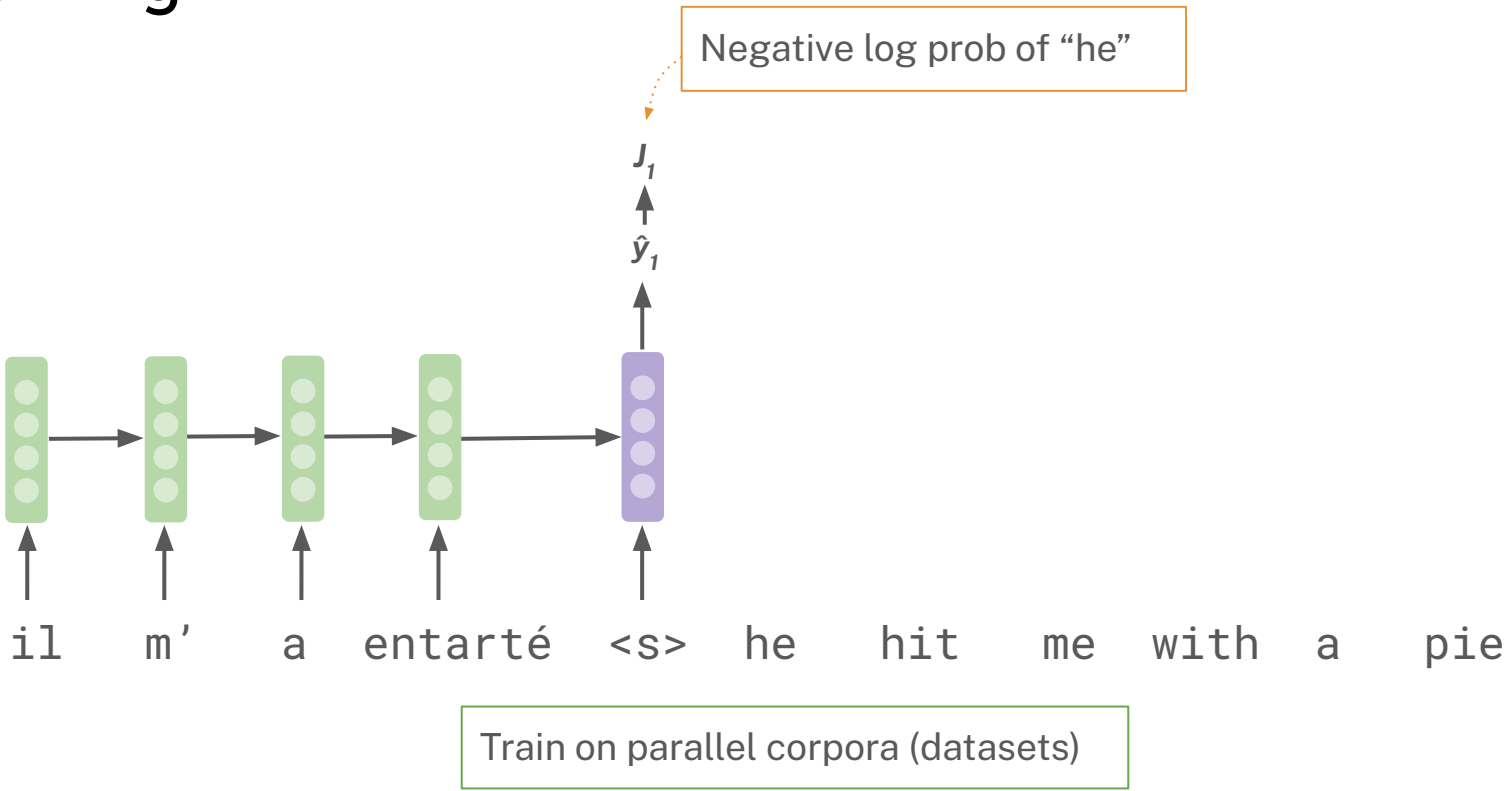
il m’ a entarté <s> he hit me with a pie

Train on parallel corpora (datasets)

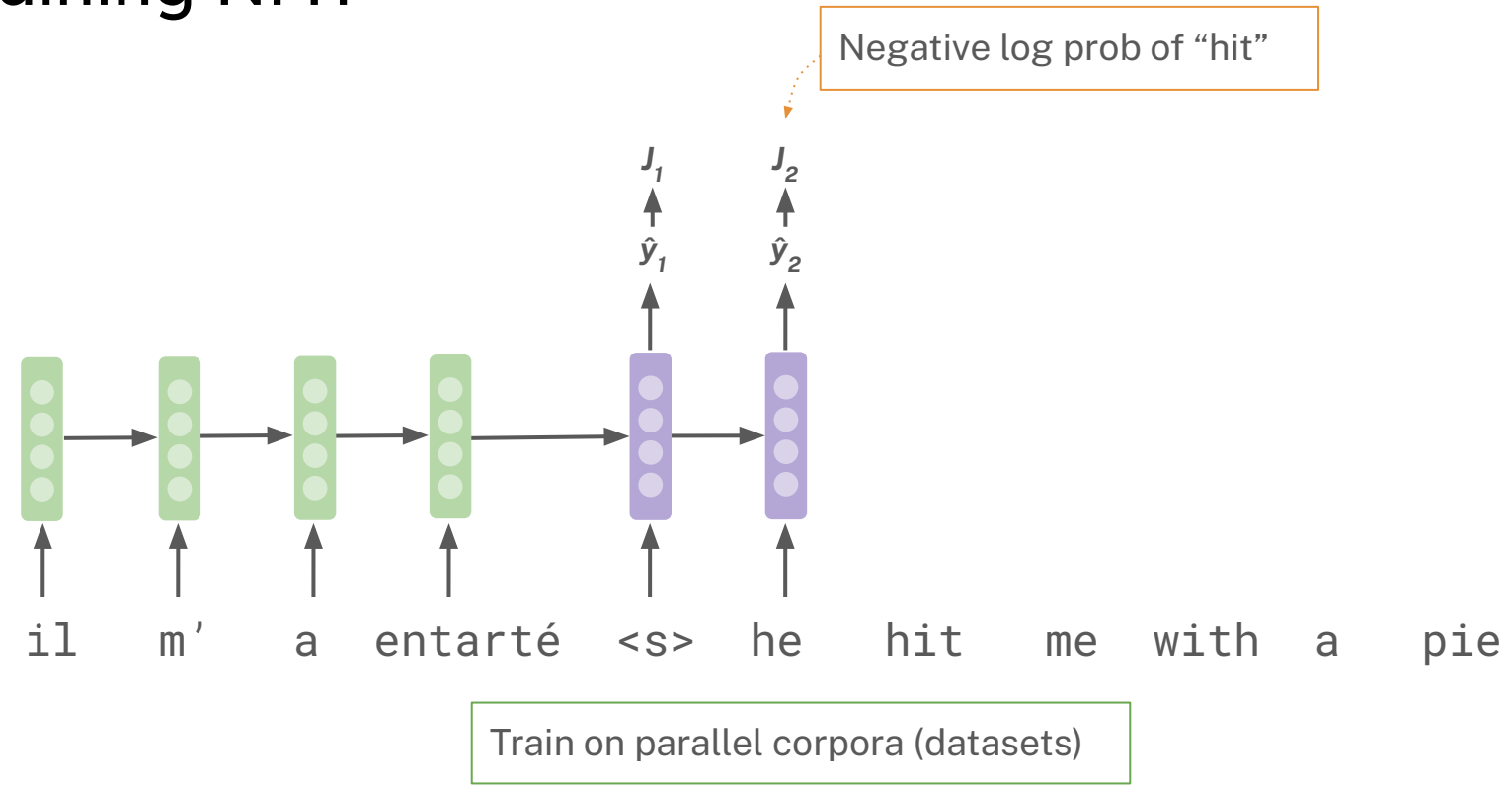
Training NMT

Train on parallel corpora (datasets)

Training NMT

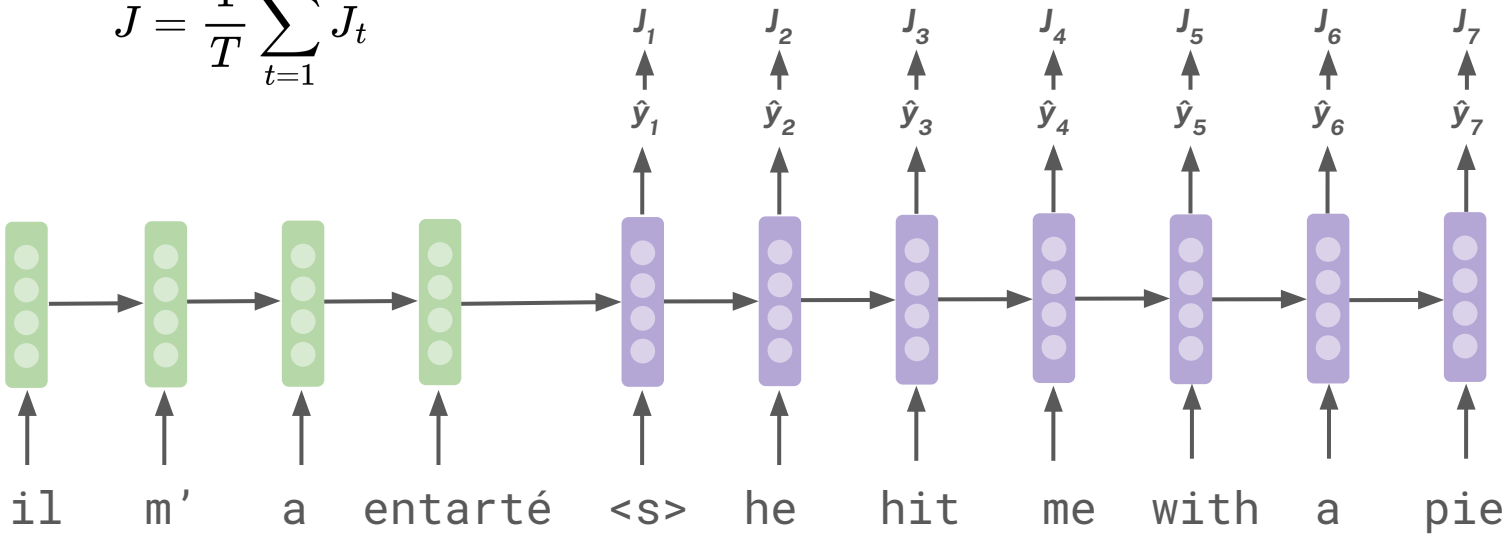


Training NMT



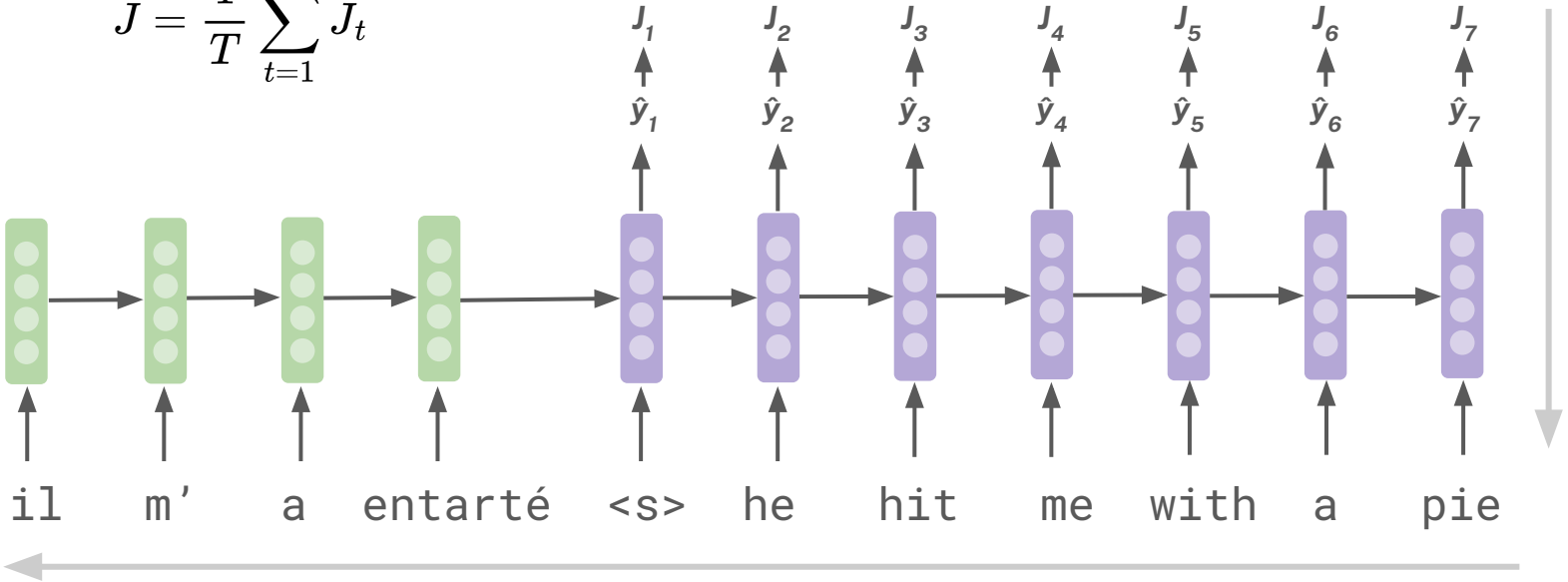
Training NMT

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t$$



Training NMT

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t$$



Seq2seq is optimized as **a single system**. Backprop operates end-to-end.

What if we don't have enough data?

Data augmentation using monolingual LMs and bilingual lexicons

A boy is holding a bat.

computer vision augmentation

A boy is holding a bat.

A boy is holding a bat. → *translation augmentation* → A boy is holding a **backpack**.

Ein Junge hält einen Schläger. → *augmentation* → Ein Junge hält einen *Rucksack*.

[Fadaee et al. 2017]

Plausible substitutes are suggested by a monolingual LM:

`My sister drives a [car / motorbike]` ✓

`My uncle sold his [house / motorbike]` ✓

`Alice waters the [plant / motorbike]` ✗

`John bought two [shirts / motorbike]` ✗

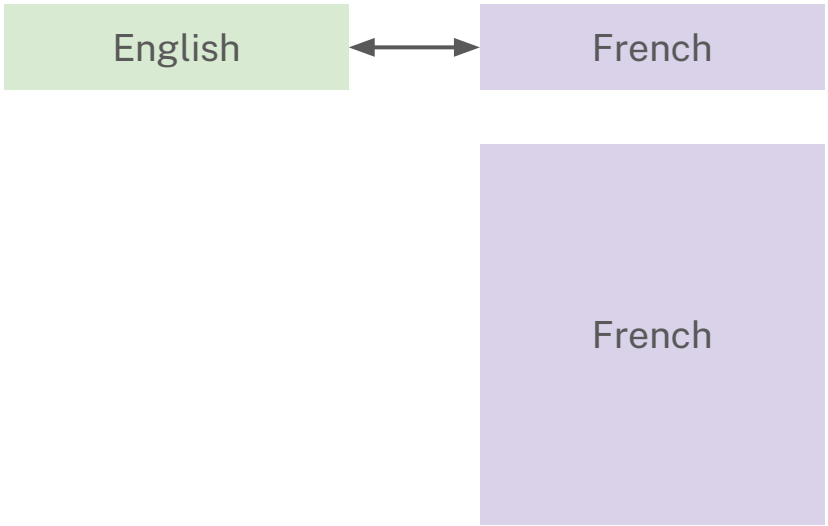
Prioritize including examples involving **rarer words**.

Provided by a bilingual lexicon

What if we don’t have enough data?

Back translation

Given a small parallel dataset b/t source (e.g. English) and target (e.g. French) + a huge monolingual target dataset



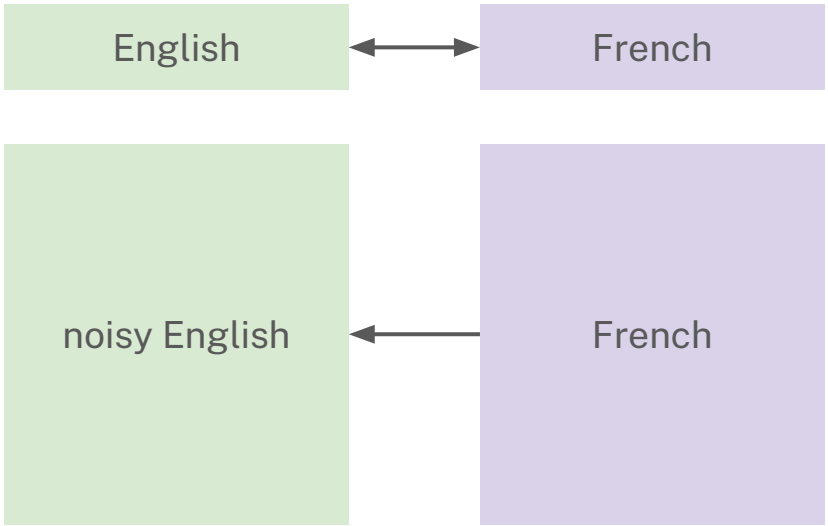
What if we don’t have enough data?

Back translation

Given a small parallel dataset b/t source (e.g. English) and target (e.g. French) + a huge monolingual target dataset

Train a target → source model

Use this model to translate target monolingual corpus → source lang



What if we don't have enough data?

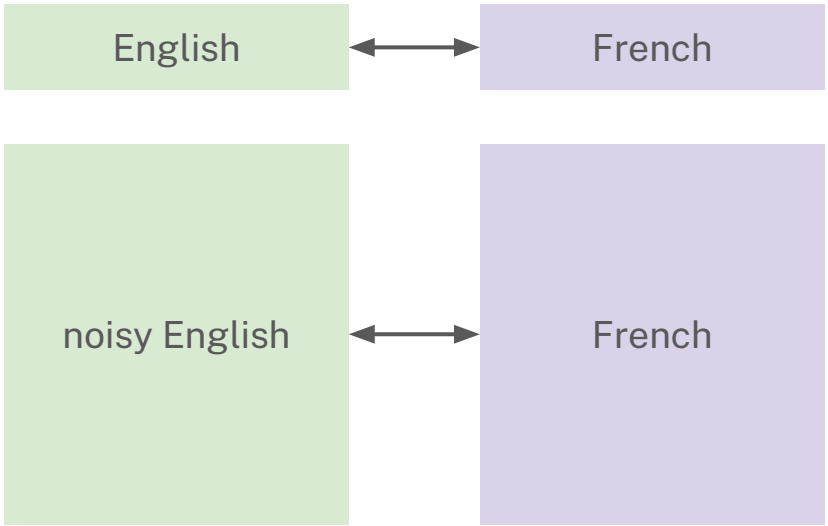
Back translation

Given a small parallel dataset b/t source (e.g. English) and target (e.g. French) + a huge monolingual target dataset

Train a target → source model

Use this model to translate target monolingual corpus → source lang

Use all of this data to train a **source → target** model!



Seq2seq is versatile

Sequence-to-sequence is useful for more than just MT

Many NLP tasks can be phrased as sequence-to-sequence:

- Summarization (long text → short text)
- Dialogue (previous utterances → next utterance)
- Parsing (input text → output parse as sequence)
- Code generation (natural language → Python code)

MT in the LLM era

Team	Approach
Algharb	LLM
AMI	LLM
COILD-BHO	LLM
CommandA-WMT	LLM
CUNI-Doc Transformer	enc-dec
CUNI-EdUKate-v1	LLM
CUNI-SFT	LLM
CUNI-Transformer	enc-dec
DLUT_GTCOM	LLM, enc-dec
HYT	LLM
GemTrans	LLM
In2x	LLM
NTTSU	LLM
SalamandraTA	LLM
SH	LLM
Shy-hunyuan-MT	LLM
Systran	LLM
TranssionMT	enc-dec, LLM
UvA-MT	LLM
Wenyiil	LLM
Yandex	LLM
Yolu	LLM

Kocmi et al. 2025

Though LLMs dominate the MT landscape now, challenges like **data quality and size** still remain.

Another key challenge has also stuck around…

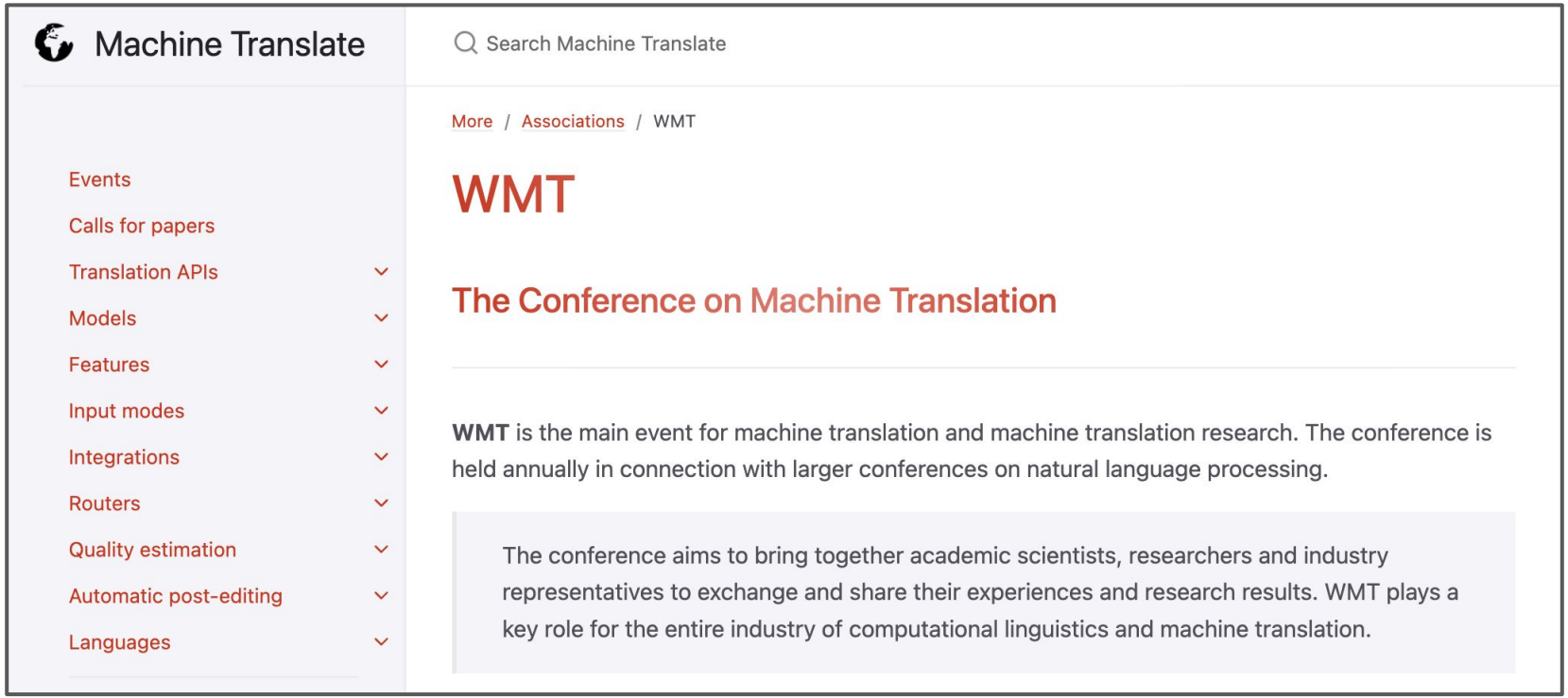
Metrics for **evaluating** MT, e.g. BLEU, COMET, BLEURT:

- may penalize valid lexical choices and phrase reorderings that don't match the ground-truth reference
- may delude us into thinking a translation is "high quality" but still miss significant grammatical errors

Much discussion around how to *best* evaluate translations!

We'll talk about stuff like this more in our evaluation & benchmarking lecture later.

Resources and competitions for MT

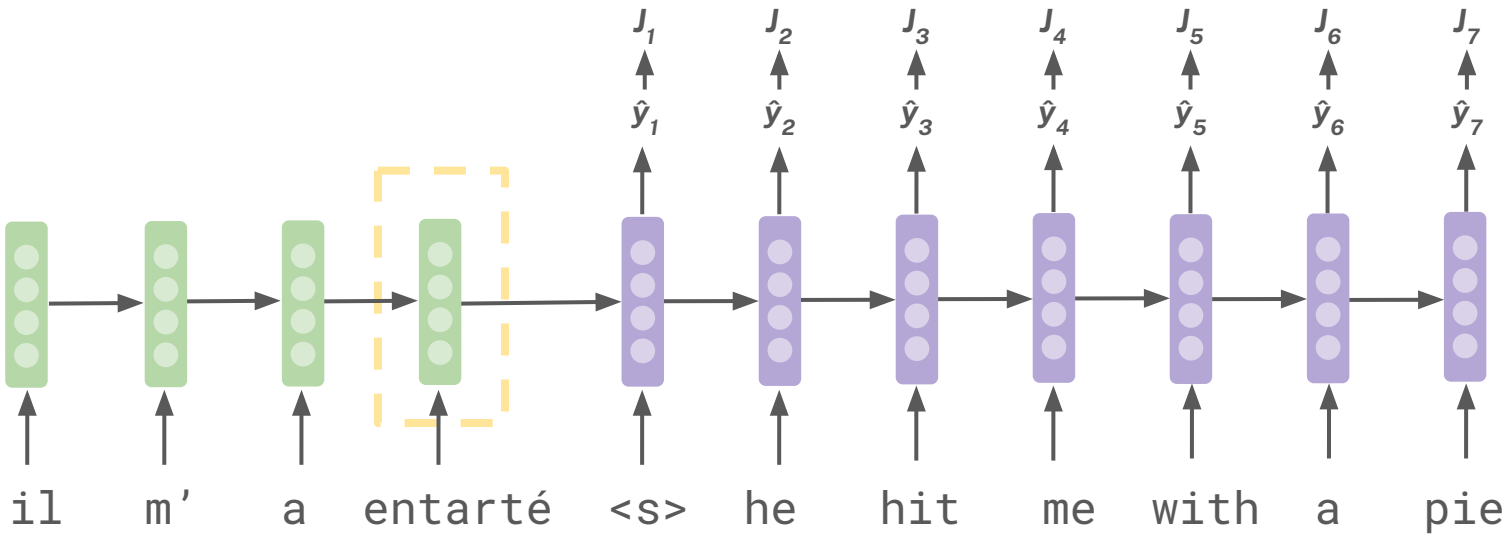


Foreshadowing

The bottleneck

One encoding vector needs to capture all of the info about the source sentence.

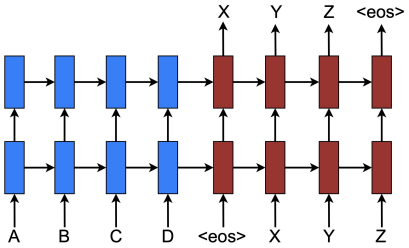
+ Longer sentences can lead to vanishing gradients!



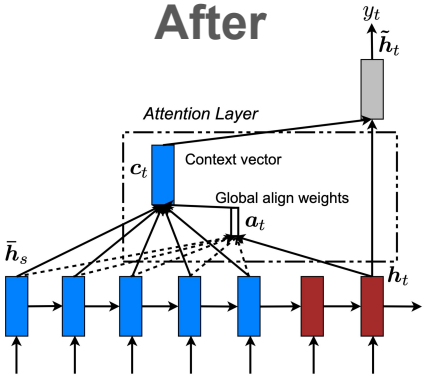
Attention

Attention provides a solution to the bottleneck problem

Before



After



Next class!!

Figures from Luong et al. 2015, ACL Test of Time Award winner

What we covered today

1. Language modeling (e.g. *n*-gram LMs, window-based neural LMs)

2. Recurrent neural networks (RNNs)

3. Machine translation (a very language-y task involving RNNs)

Please form your project groups by EOD!

If you don’t yet have a group, come up to me during the remaining class time and we can figure out group match-ups right now, right here.

Otherwise I will assign you groups.

Upcoming milestone

Project proposal presentation (due Sept 22)

- Time per presentation TBD; will announce in class and on course website “Assignments” page after all groups are formed.
- This **live pitch** should include why the question/s being asked are important and why your project can be completed feasibly within this semester. Grading rubric will be finalized by next class.
- Each project team will be assigned to me or Sonia as the “main” mentor based on the presentation pitch.
- It’s okay to modify your topic between your brainstorming pitch and your actual written proposal, but if you do this, explicitly say you did