CS490/590 Lecture 12: Recurrent Neural Nets & Attention

Eren Gultepe
SIUE

Adapted from: Jimmy Ba and Bo Wang

Overview

- We've seen how to build neural nets to make predictions from a fixed-size input to a fixed-size output
- Sometimes we're interested in predicting sequences
  - Speech-to-text and text-to-speech
  - Caption generation
  - Machine translation
- If the input is also a sequence, this setting is known as sequence-to-sequence prediction.
- We already saw one way of doing this: neural language models
  - But autoregressive models are memoryless, so they can't learn long-distance dependencies.
  - Recurrent neural networks (RNNs) are a kind of architecture which can remember things over time.

Overview

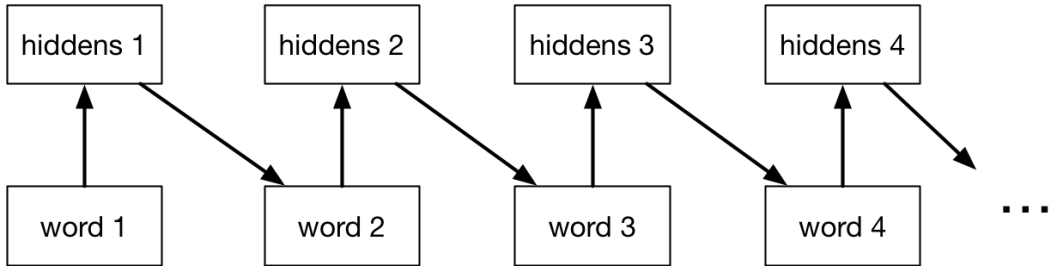
Recall that we made a Markov assumption:

$$p(w_i \mid w_1, \ldots, w_{i-1}) = p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}).$$

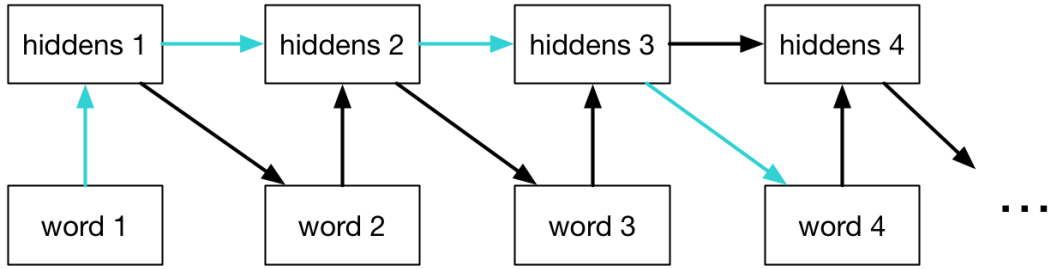
This means the model is memoryless, i.e. it has no memory of anything before the last few words. But sometimes long-distance context can be important.

Overview

- Autoregressive models such as the neural language model are memoryless, so they can only use information from their immediate context (in this figure, context length $= 1$):

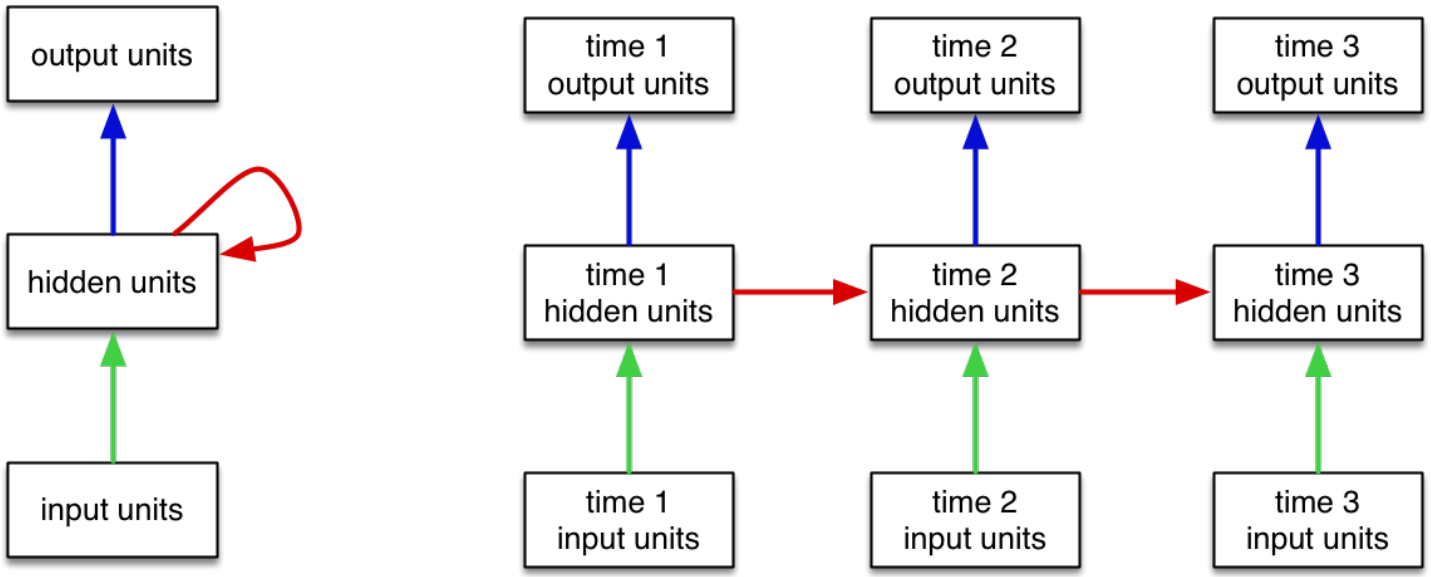


- If we add connections between the hidden units, it becomes a recurrent neural network (RNN). Having a memory lets an RNN use longer-term dependencies:



Recurrent neural nets

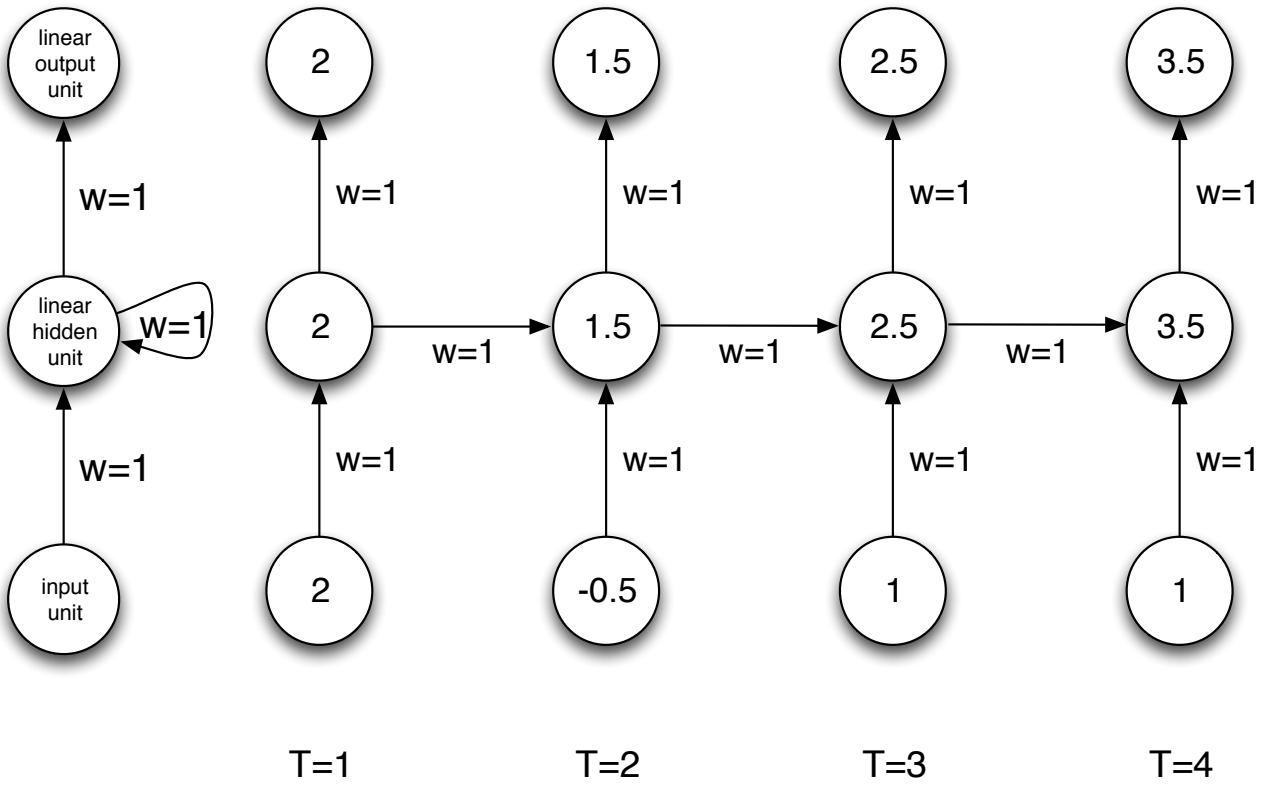
- We can think of an RNN as a dynamical system with one set of hidden units which feed into themselves. The network's graph would then have self-loops.
- We can unroll the RNN's graph by explicitly representing the units at all time steps. The weights and biases are shared between all time steps
  - Except there is typically a separate set of biases for the first time step.



RNN examples

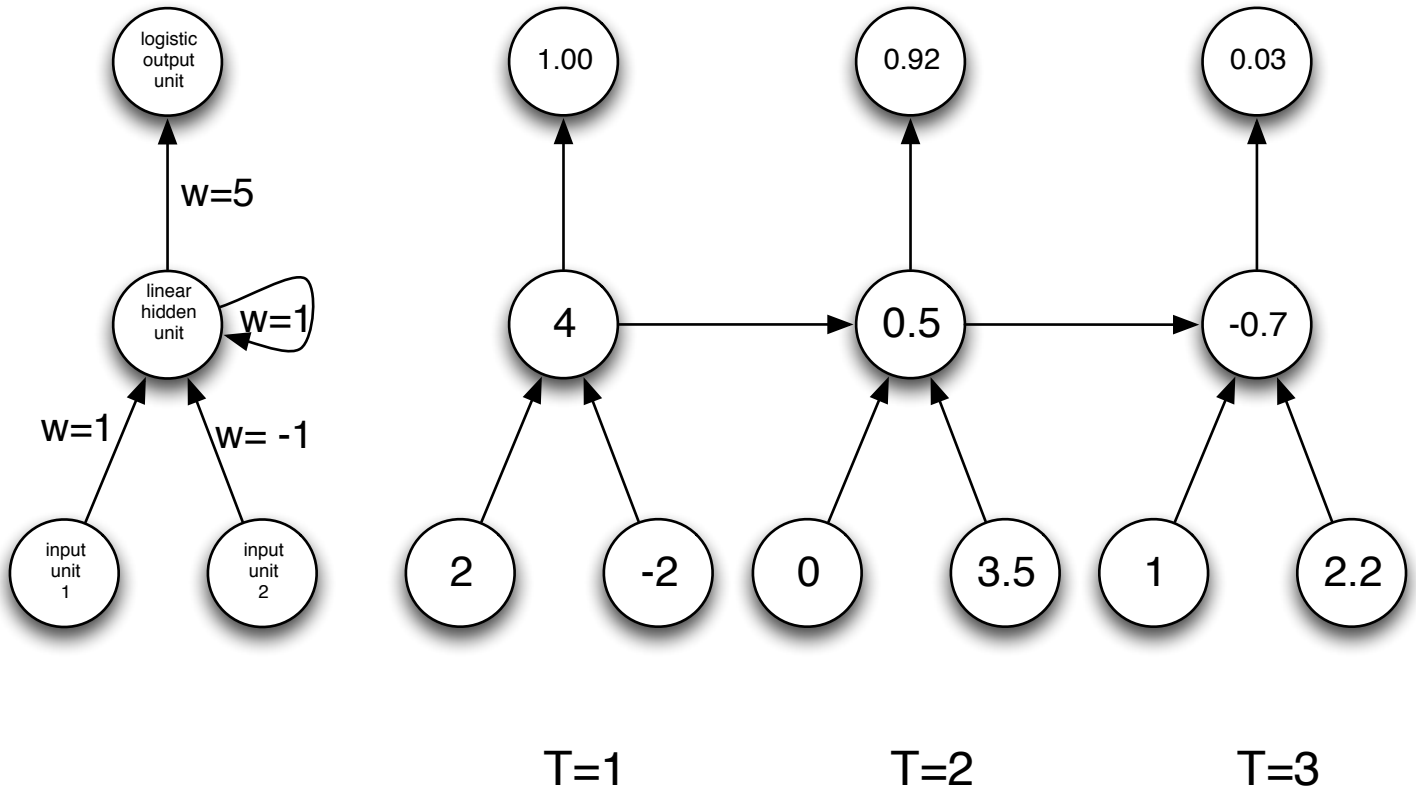
Now let's look at some simple examples of RNNs.

This one sums its inputs:



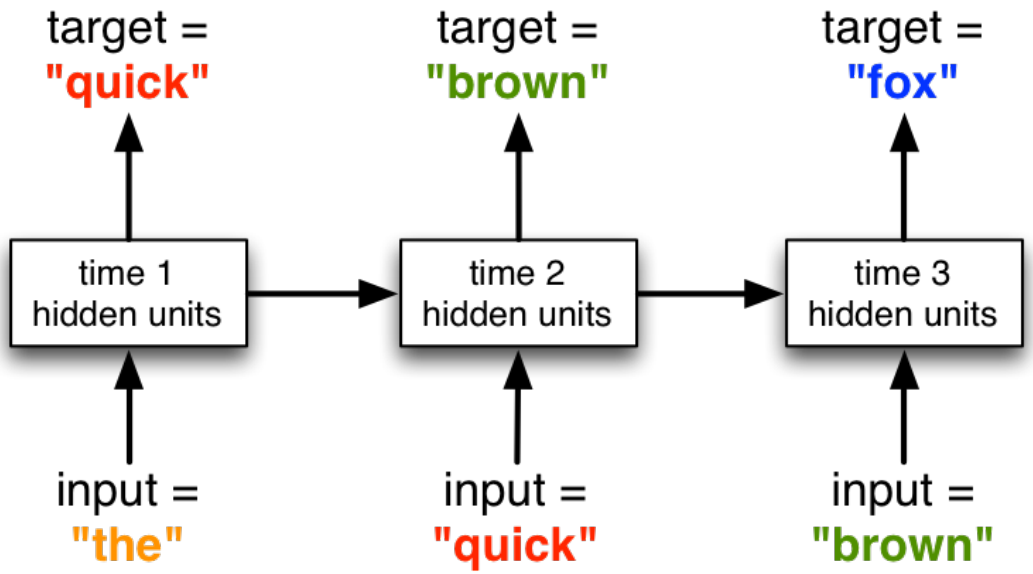
RNN examples

This one determines if the total values of the first or second input are larger:



Language Modeling

Back to our motivating example, here is one way to use RNNs as a language model:

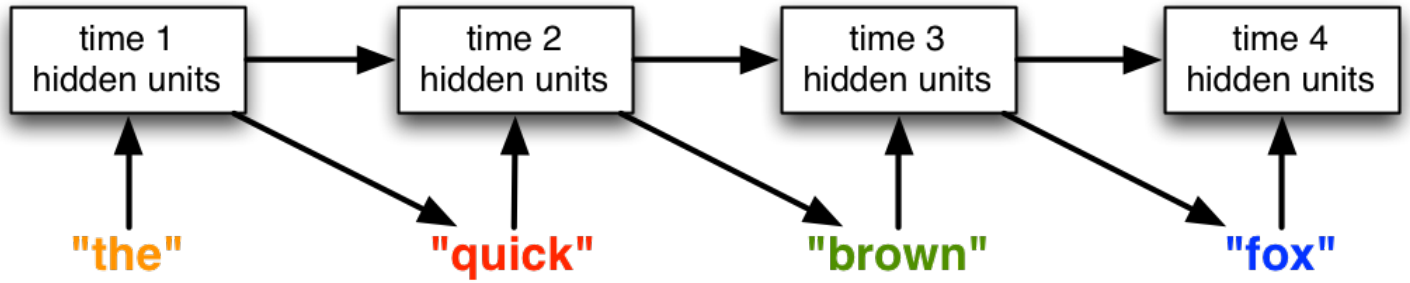


As with our language model, each word is represented as an indicator vector, the model predicts a distribution, and we can train it with cross-entropy loss.

This model can learn long-distance dependencies.

Language Modeling

When we generate from the model (i.e. compute samples from its distribution over sentences), the outputs feed back in to the network as inputs.



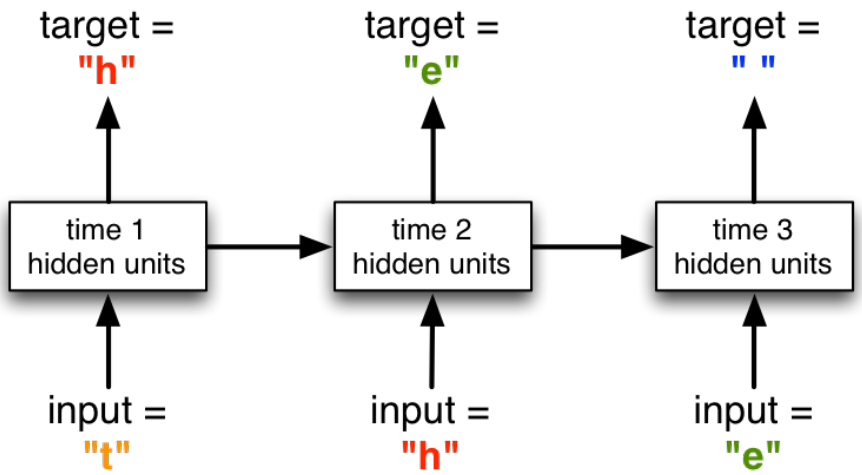
At training time, the inputs are the tokens from the training set (rather than the network's outputs). This is called teacher forcing.

Some remaining challenges:

- Vocabularies can be very large once you include people, places, etc. It's computationally difficult to predict distributions over millions of words.
- How do we deal with words we haven't seen before?
- In some languages (e.g. German), it's hard to define what should be considered a word.

Language Modeling

Another approach is to model text *one character at a time*!



This solves the problem of what to do about previously unseen words. Note that long-term memory is *essential* at the character level!

Note: modeling language well at the character level requires *multiplicative* interactions, which we're not going to talk about.

Language Modeling

From Geoff Hinton's Coursera course, an example of a paragraph generated by an RNN language model one character at a time:

> He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemerable street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade.”

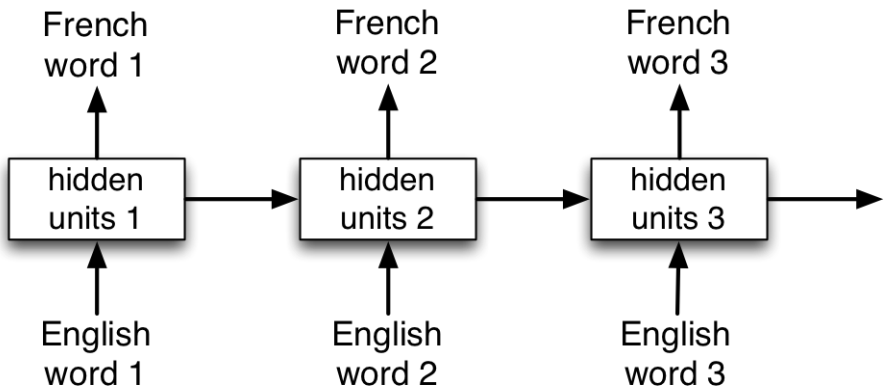
J. Martens and I. Sutskever, 2011. Learning recurrent neural networks with Hessian-free optimization.

`http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Martens_532.pdf`

Neural Machine Translation

We'd like to translate, e.g., English to French sentences, and we have pairs of translated sentences to train on.

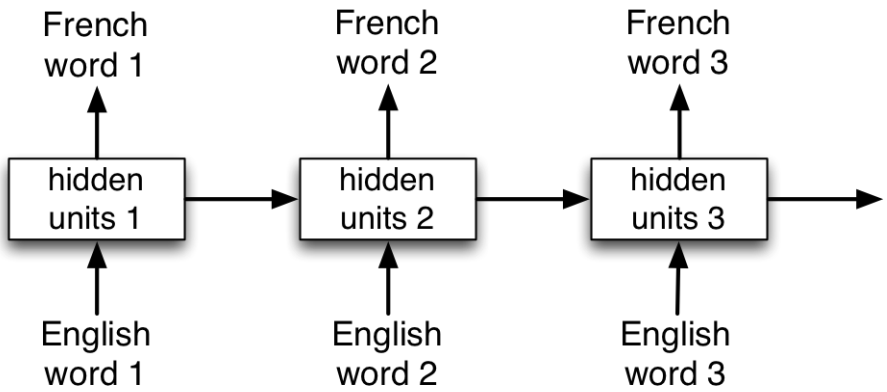
What's wrong with the following setup?



Neural Machine Translation

We'd like to translate, e.g., English to French sentences, and we have pairs of translated sentences to train on.

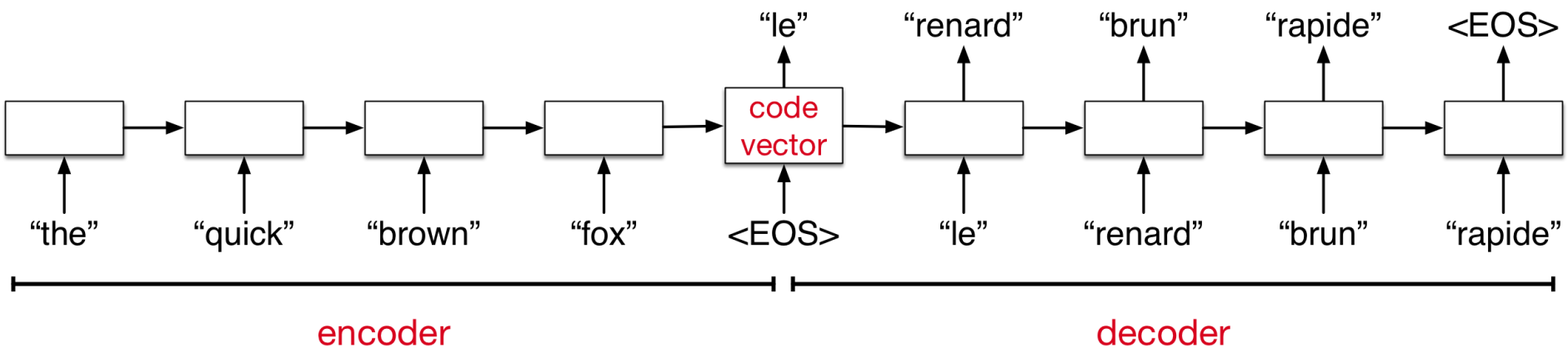
What's wrong with the following setup?



- The sentences might not be the same length, and the words might not align perfectly.
- You might need to resolve ambiguities using information from later in the sentence.

Neural Machine Translation

Sequence-to-sequence architecture: the network first reads and memorizes the sentence. When it sees the end token, it starts outputting the translation.



The encoder and decoder are two different networks with different weights.

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. EMNLP 2014.

Sequence to Sequence Learning with Neural Networks, Ilya Sutskever, Oriol Vinyals and Quoc Le, NIPS 2014.

What can RNNs compute?

In 2014, Google researchers built an encoder-decoder RNN that learns to execute simple Python programs, *one character at a time*!

Input:
```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```
Target: 25011.

Input:
```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else
    5308)
```
Target: 1218.

Example training inputs

Input:
```
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
```
Target: hkhpg

A training input with characters scrambled

W. Zaremba and I. Sutskever, "Learning to Execute." `http://arxiv.org/abs/1410.4615`

What can RNNs compute?

Some example results:

Input:

```
print(6652).
```

Target:	6652.
"Baseline" prediction:	6652.
"Naive" prediction:	6652.
"Mix" prediction:	6652.
"Combined" prediction:	6652.

Input:

```
print((5997-738)).
```

Target:	5259.
"Baseline" prediction:	5101.
"Naive" prediction:	5101.
"Mix" prediction:	5249.
"Combined" prediction:	5229.

Input:

```
d=5446
for x in range(8):d+=(2678 if 4803<2829 else 9848)
print((d if 5935<4845 else 3043)).
```

Target:	3043.
"Baseline" prediction:	3043.
"Naive" prediction:	3043.
"Mix" prediction:	3043.
"Combined" prediction:	3043.

Input:

```
print(((1090-3305)+9466)).
```

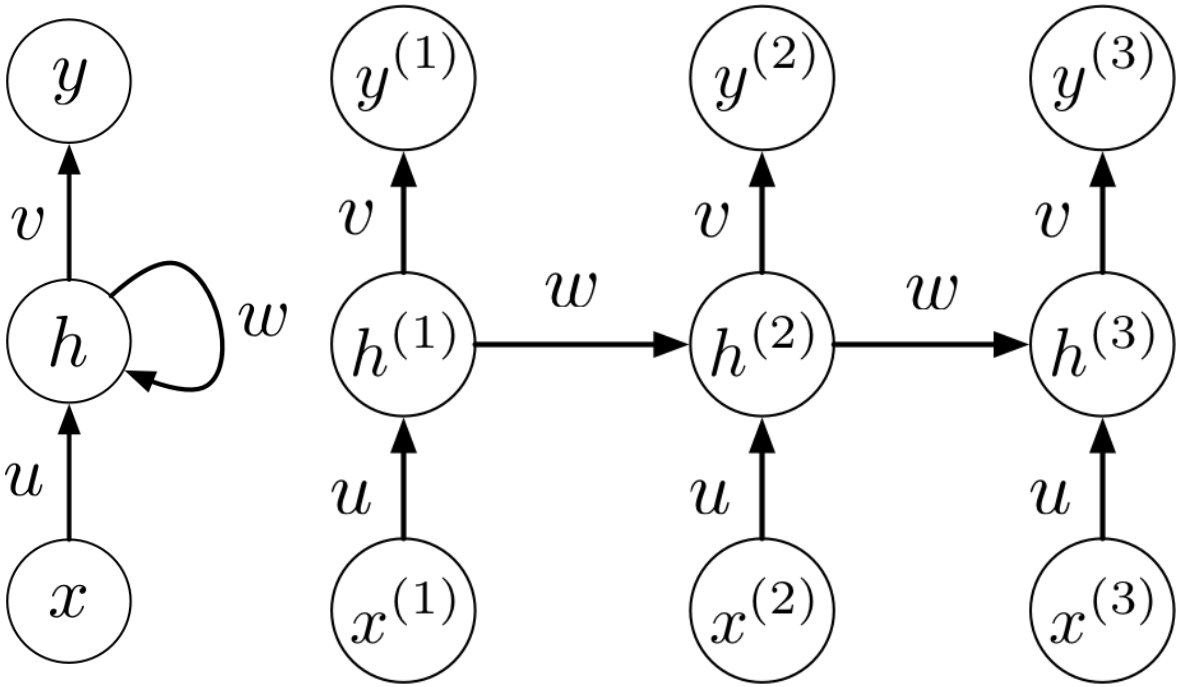
Target:	7251.
"Baseline" prediction:	7111.
"Naive" prediction:	7099.
"Mix" prediction:	7595.
"Combined" prediction:	7699.

Take a look through the results (`http://arxiv.org/pdf/1410.4615v2.pdf#page=10`). It's fun to try to guess from the mistakes what algorithms it's discovered.

Backprop Through Time

As you can guess, we learn the RNN weights using backprop.

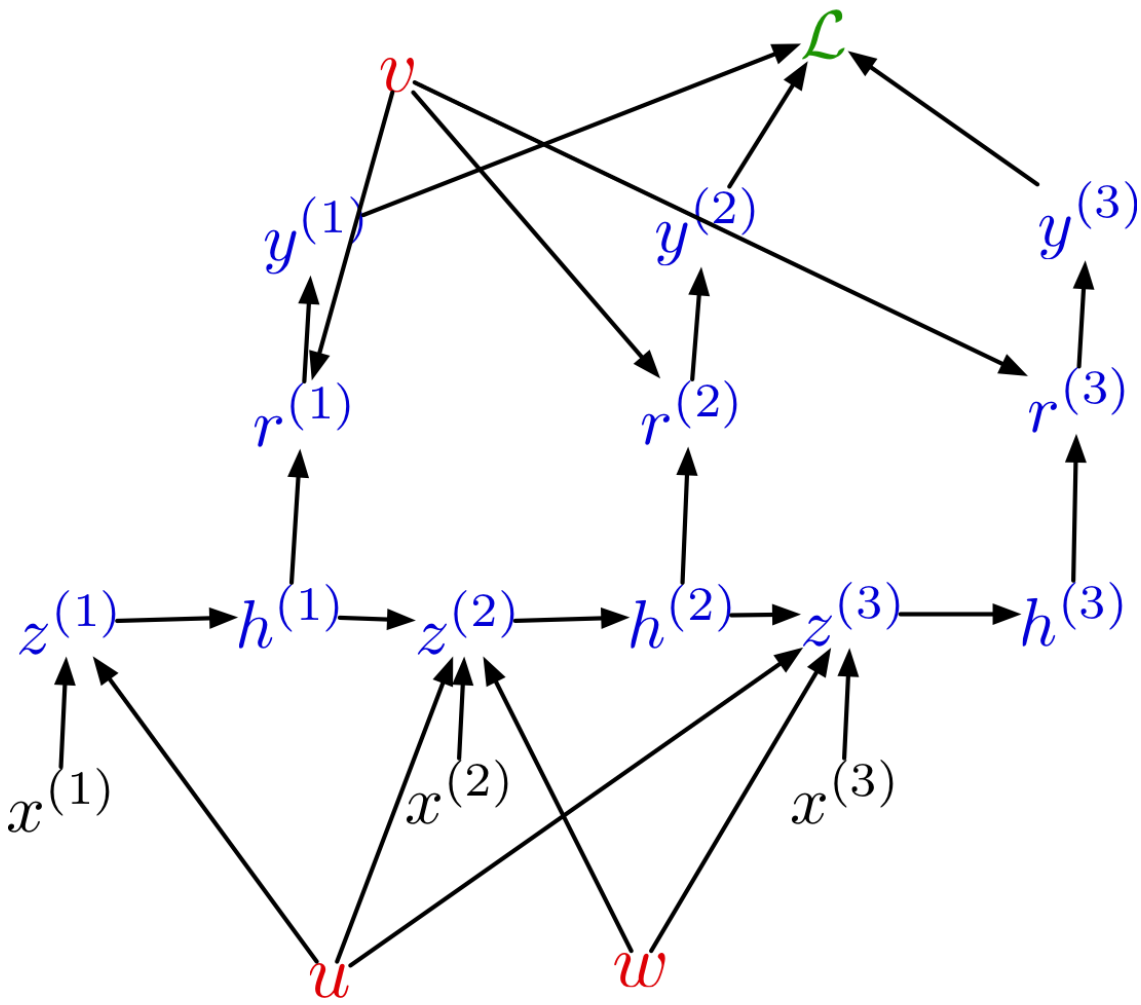
- In particular, we do backprop on the unrolled network. This is known as backprop through time.



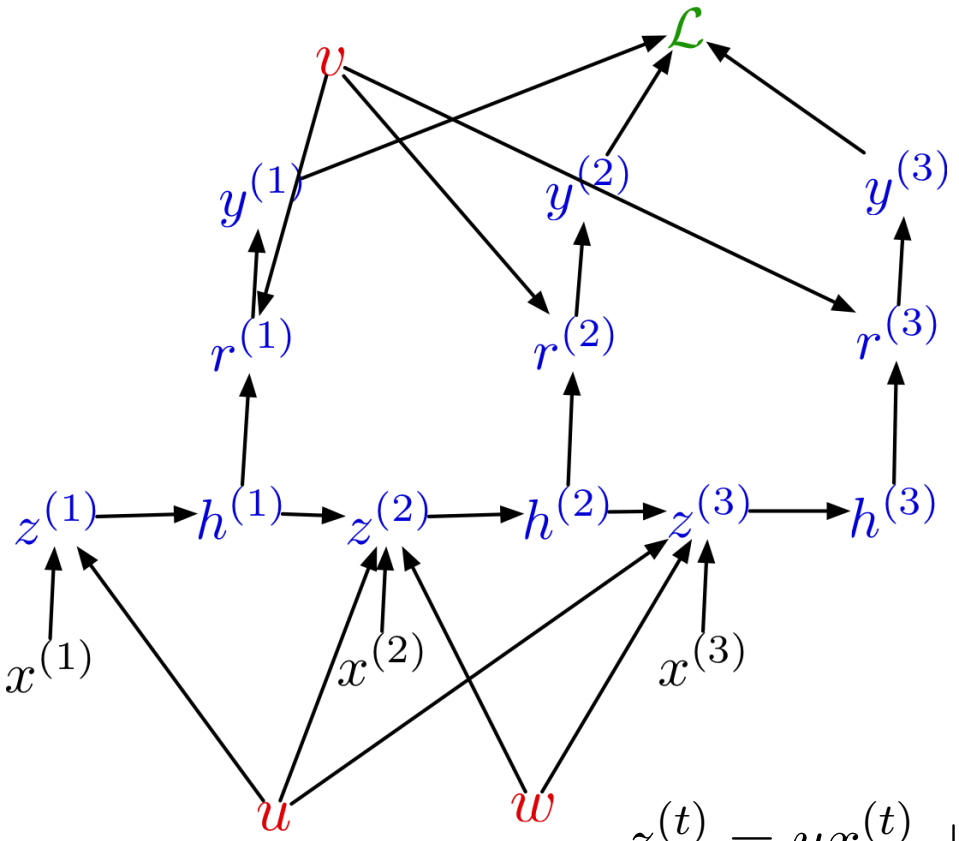
$$z^{(t)} = ux^{(t)} + wh^{(t-1)}$$
$$h^{(t)} = \phi(z^{(t)})$$
$$r^{(t)} = vh^{(t)}$$
$$y^{(t)} = \phi(r^{(t)}).$$

Backprop Through Time

Here's the unrolled computation graph. Notice the weight sharing.



Backprop Through Time



Forward Pass:

$$z^{(t)} = ux^{(t)} + wh^{(t-1)}$$
$$h^{(t)} = \phi(z^{(t)})$$
$$r^{(t)} = vh^{(t)}$$
$$y^{(t)} = \phi(r^{(t)}).$$

Activations:

$$\overline{\mathcal{L}} = 1$$
$$\overline{y^{(t)}} = \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial y^{(t)}}$$
$$\overline{r^{(t)}} = \overline{y^{(t)}}\, \phi'(r^{(t)})$$
$$\overline{h^{(t)}} = \overline{r^{(t)}}\, v + \overline{z^{(t+1)}}\, w$$
$$\overline{z^{(t)}} = \overline{h^{(t)}}\, \phi'(z^{(t)})$$

Parameters:

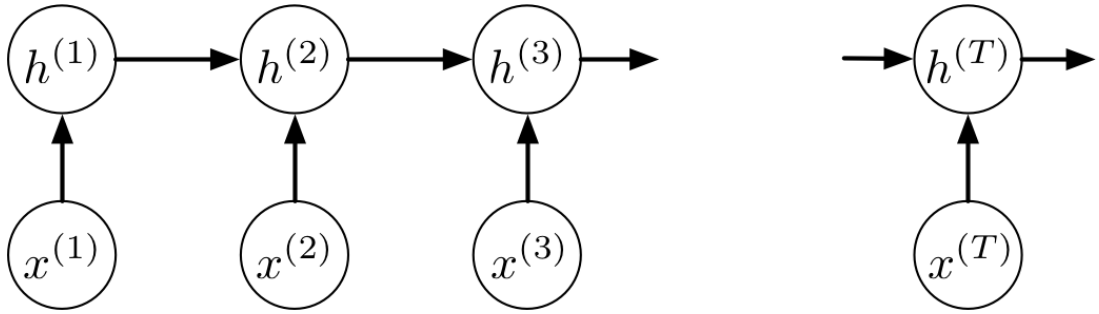
$$\overline{u} = \sum_t \overline{z^{(t)}}\, x^{(t)}$$
$$\overline{v} = \sum_t \overline{r^{(t)}}\, h^{(t)}$$
$$\overline{w} = \sum_t \overline{z^{(t+1)}}\, h^{(t)}$$

Backprop Through Time

- Now you know how to compute the derivatives using backprop through time.
- The hard part is using the derivatives in optimization. They can explode or vanish. Addressing this issue will take all of the next lecture.

Why Gradients Explode or Vanish

Consider a univariate version of the encoder network:



Backprop updates:

$$\overline{h^{(t)}} = \overline{z^{(t+1)}}\, w$$
$$\overline{z^{(t)}} = \overline{h^{(t)}}\, \phi'(z^{(t)})$$

Applying this recursively:

$$\overline{h^{(1)}} = \underbrace{w^{T-1}\phi'(z^{(2)}) \cdots \phi'(z^{(T)})}_{\text{the Jacobian } \partial h^{(T)}/\partial h^{(1)}}\, \overline{h^{(T)}}$$

With linear activations:

$$\partial h^{(T)}/\partial h^{(1)} = w^{T-1}$$

Exploding:

$$w = 1.1, T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 117.4$$

Vanishing:

$$w = 0.9, T = 50 \quad \Rightarrow \quad \frac{\partial h^{(T)}}{\partial h^{(1)}} = 0.00515$$

Why Gradients Explode or Vanish

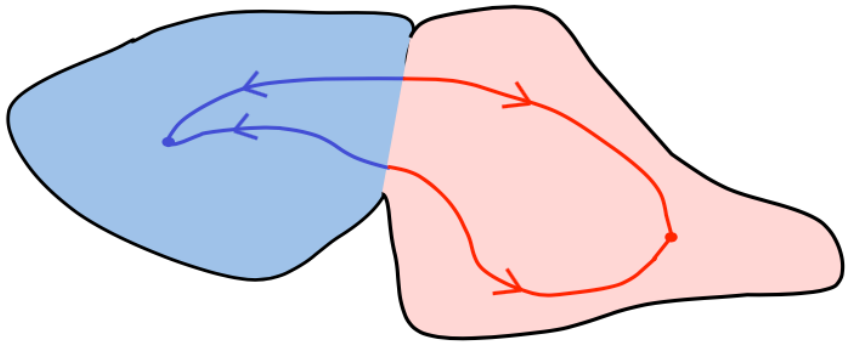
- More generally, in the multivariate case, the Jacobians multiply:

$$\frac{\partial \mathsf{h}^{(T)}}{\partial \mathsf{h}^{(1)}} = \frac{\partial \mathsf{h}^{(T)}}{\partial \mathsf{h}^{(T-1)}} \cdots \frac{\partial \mathsf{h}^{(2)}}{\partial \mathsf{h}^{(1)}}$$

- Matrices can explode or vanish just like scalar values, though it's slightly harder to make precise.
- Contrast this with the forward pass:
  - The forward pass has nonlinear activation functions which squash the activations, preventing them from blowing up.
  - The backward pass is linear, so it's hard to keep things stable. There's a thin line between exploding and vanishing.

Why Gradients Explode or Vanish

- We just looked at exploding/vanishing gradients in terms of the mechanics of backprop. Now let's think about it conceptually.
- The Jacobian $\partial \mathrm{h}^{(T)} / \partial \mathrm{h}^{(1)}$ means, how much does $h^{(T)}$ change when you change $\mathrm{h}^{(1)}$?
- Let's imagine an RNN's behavior as a dynamical system, which has various attractors:



– Geoffrey Hinton, Coursera

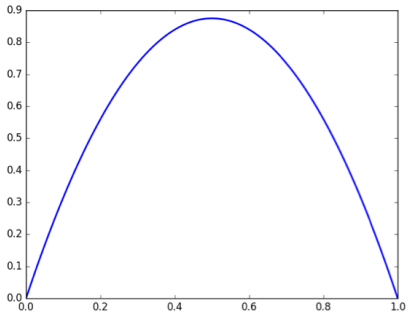
- Within one of the colored regions, the gradients vanish because even if you move a little, you still wind up at the same attractor.
- If you're on the boundary, the gradient blows up because moving slightly moves you from one attractor to the other.

Iterated Functions

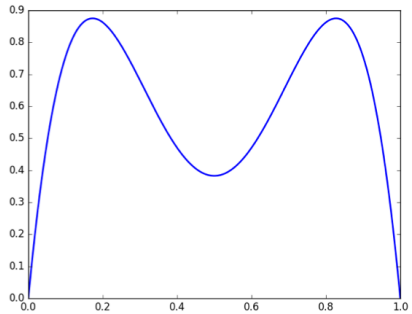
- Each hidden layer computes some function of the previous hiddens and the current input. This function gets iterated:

$$h^{(4)} = f(f(f(h^{(1)}, x^{(2)}), x^{(3)}), x^{(4)}).$$

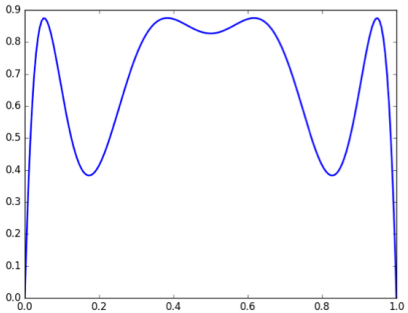
- Consider a toy iterated function: $f(x) = 3.5\,x\,(1-x)$



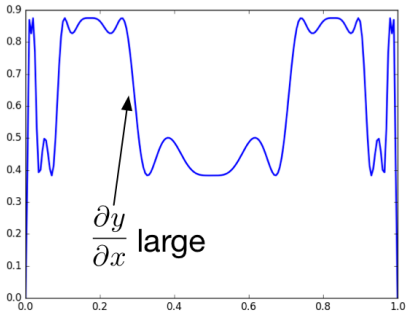
$y = f(x)$



$y = f(f(x))$



$y = f(f(f(x)))$



$y = \underbrace{f \circ \cdots \circ f}_{6 \text{ times}}(x)$

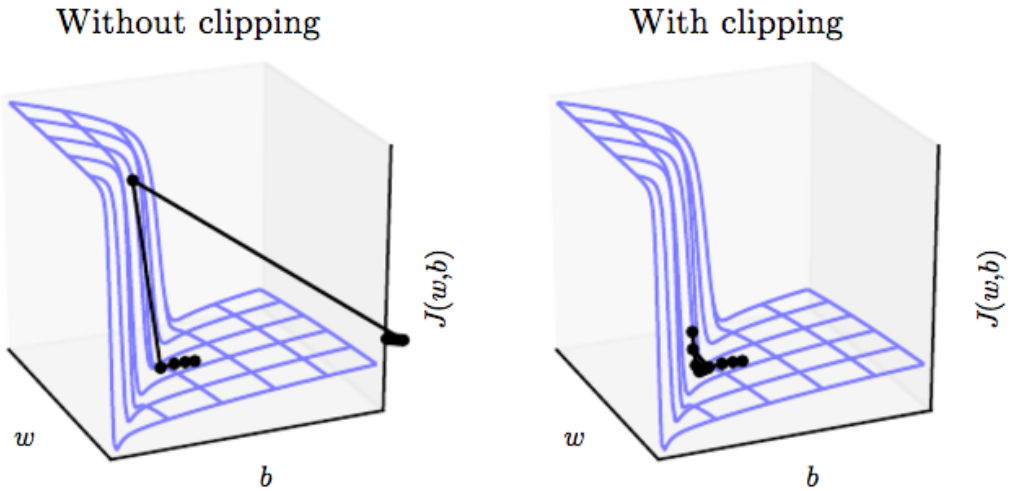
Keeping Things Stable

- One simple solution: gradient clipping
- Clip the gradient g so that it has a norm of at most η:

  if $\|\mathrm{g}\| > \eta$:

$$\mathrm{g} \leftarrow \frac{\eta \mathrm{g}}{\|\mathrm{g}\|}$$

- The gradients are biased, but at least they don't blow up.



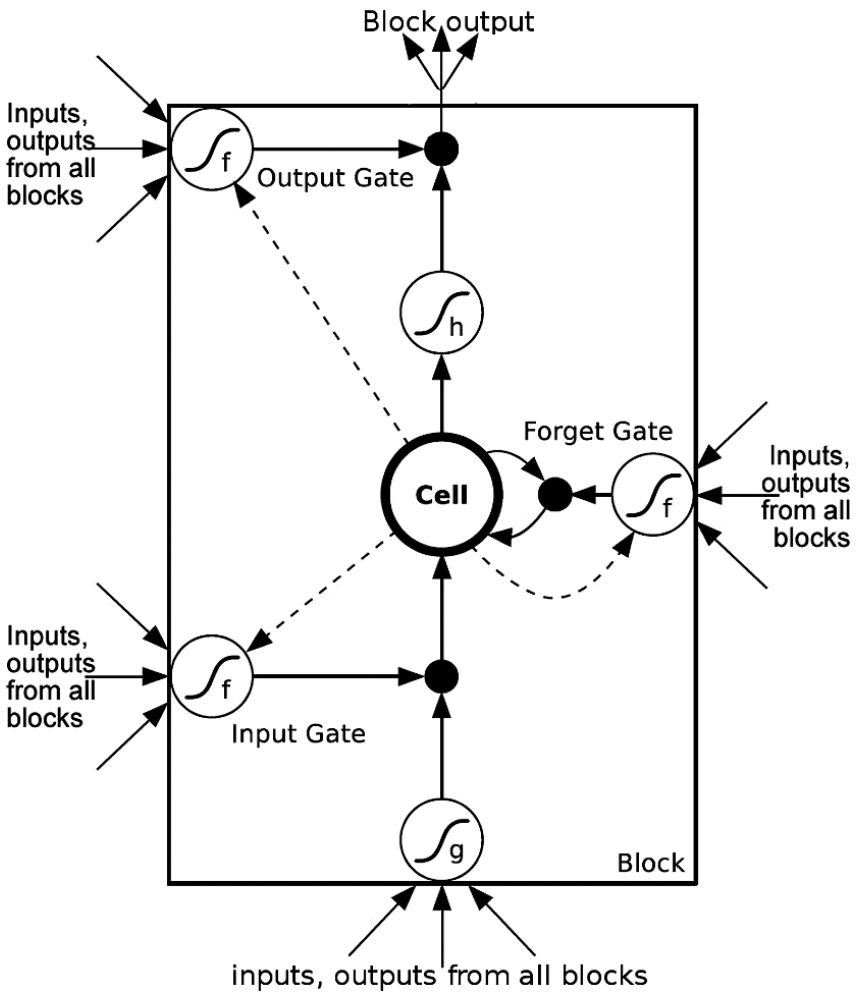
— Goodfellow et al., *Deep Learning*

Long-Term Short Term Memory

- Really, we're better off redesigning the architecture, since the exploding/vanishing problem highlights a conceptual problem with vanilla RNNs.
- Long-Term Short Term Memory (LSTM) is a popular architecture that makes it easy to remember information over long time periods.
  - What's with the name? The idea is that a network's activations are its short-term memory and its weights are its long-term memory.
  - The LSTM architecture wants the short-term memory to last for a long time period.
- It's composed of memory cells which have controllers saying when to store or forget information.

Long-Term Short Term Memory

- Replace each single unit in an RNN by a memory block -



$$c_{t+1} = c_t \cdot \text{forget gate} + \text{new input} \cdot \text{input gate}$$

- $i = 0, f = 1 \Rightarrow$ remember the previous value
- $i = 1, f = 1 \Rightarrow$ add to the previous value
- $i = 0, f = 0 \Rightarrow$ erase the value
- $i = 1, f = 0 \Rightarrow$ overwrite the value

Setting $i = 0, f = 1$ gives the reasonable "default" behavior of just remembering things.

Long-Term Short Term Memory

- In each step, we have a vector of memory cells c, a vector of hidden units h, and vectors of input, output, and forget gates i, o, and f.
- There's a full set of connections from all the inputs and hiddens to the input and all of the gates:

$$\begin{pmatrix} \mathsf{i}_t \\ \mathsf{f}_t \\ \mathsf{o}_t \\ \mathsf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \mathsf{W} \begin{pmatrix} \mathsf{y}_t \\ \mathsf{h}_{t-1} \end{pmatrix}$$

$$\mathsf{c}_t = \mathsf{f}_t \circ \mathsf{c}_{t-1} + \mathsf{i}_t \circ \mathsf{g}_t$$

$$\mathsf{h}_t = \mathsf{o}_t \circ \tanh(\mathsf{c}_t)$$

- Exercise: show that if $\mathsf{f}_{t+1} = 1$, $\mathsf{i}_{t+1} = 0$, and $\mathsf{o}_t = 0$, the gradients for the memory cell get passed through unmodified, i.e.

$$\overline{\mathsf{c}_t} = \overline{\mathsf{c}_{t+1}}.$$

Long-Term Short Term Memory

- Sound complicated? ML researchers thought so, so LSTMs were hardly used for about a decade after they were proposed.
- In 2013 and 2014, researchers used them to get impressive results on challenging and important problems like speech recognition and machine translation.
- Since then, they've been one of the most widely used RNN architectures.
- There have been many attempts to simplify the architecture, but nothing was conclusively shown to be simpler and better.
- You never have to think about the complexity, since frameworks like TensorFlow provide nice black box implementations.

Long-Term Short Term Memory

Visualizations:

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Detour: Deep Residual Networks and Skip Connections

- It turns out the intuition of using linear units to by-pass vanishing gradient problem was a crucial idea behind the best ImageNet models from 2015, deep residual nets.

Year	**Model**	**Top-5 error**
2010	Hand-designed descriptors + SVM	28.2%
2011	Compressed Fisher Vectors + SVM	25.8%
2012	AlexNet	16.4%
2013	a variant of AlexNet	11.7%
2014	GoogLeNet	6.6%
2015	deep residual nets	4.5%

- The idea is using linear skip connections to easily pass information directly through a network.

Detour: Deep Residual Networks and Skip Connections

- Recall: the Jacobian $\partial \mathsf{h}^{(T)} / \partial \mathsf{h}^{(1)}$ is the product of the individual Jacobians:

$$\frac{\partial \mathsf{h}^{(T)}}{\partial \mathsf{h}^{(1)}} = \frac{\partial \mathsf{h}^{(T)}}{\partial \mathsf{h}^{(T-1)}} \cdots \frac{\partial \mathsf{h}^{(2)}}{\partial \mathsf{h}^{(1)}}$$

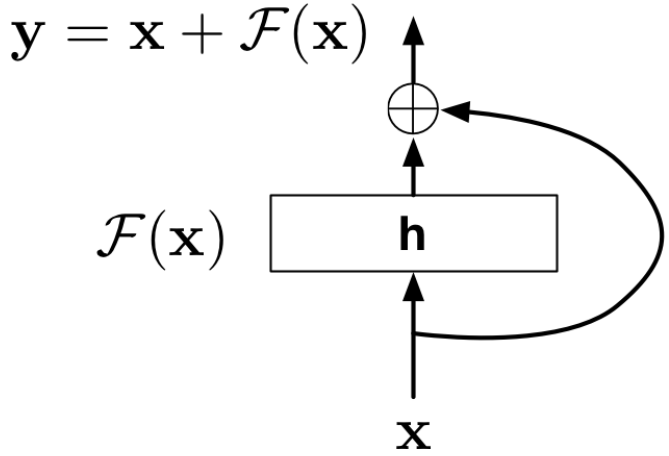
- But this applies to multilayer perceptrons and conv nets as well! (Let t index the layers rather than time.)
- Then how come we didn't have to worry about exploding/vanishing gradients until we talked about RNNs?
  - MLPs and conv nets were at most 10s of layers deep.
  - RNNs would be run over hundreds of time steps.
  - This means if we want to train a really deep conv net, we need to worry about exploding/vanishing gradients!

Detour: Deep Residual Networks and Skip Connections

- The core idea of ResNet is to apply an identity skip connection similar U-Net in Programming Assignment 2:

$$z = W^{(1)}x + b^{(1)}$$
$$h = \phi(z)$$
$$y = x + W^{(2)}h$$

- This is called a residual block, and it's actually pretty useful.
- Each layer adds something (i.e. a residual) to the previous value, rather than producing an entirely new value.
- Note: the network for $\mathcal{F}$ can have multiple layers, be convolutional, etc.

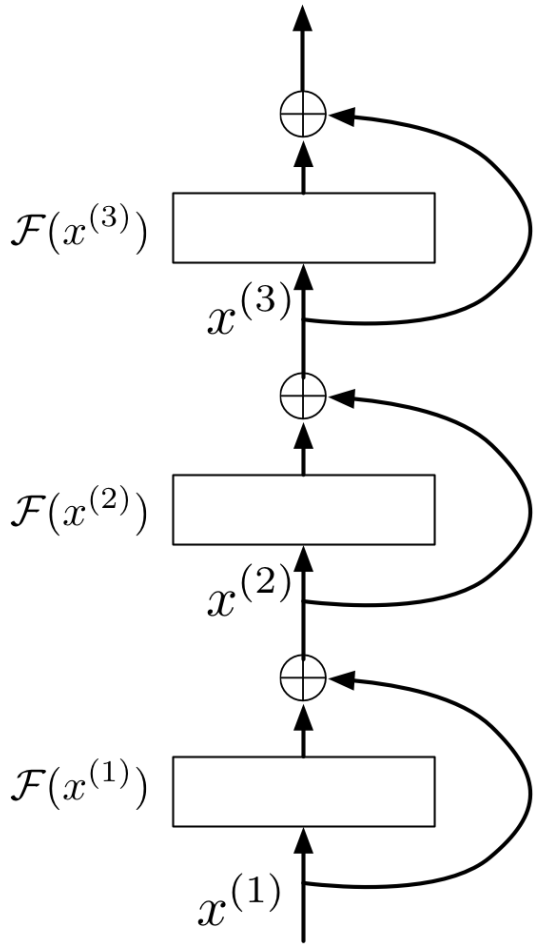


Detour: Deep Residual Networks and Skip Connections

- We can string together a bunch of residual blocks.
- What happens if we set the parameters such that $\mathcal{F}(\mathsf{x}^{(\ell)}) = 0$ in every layer?
  - Then it passes $\mathsf{x}^{(1)}$ straight through unmodified!
  - This means it's easy for the network to represent the identity function.
- Backprop:

$$\begin{aligned}\overline{\mathsf{x}^{(\ell)}} &= \overline{\mathsf{x}^{(\ell+1)}} + \overline{\mathsf{x}^{(\ell+1)}} \frac{\partial \mathcal{F}}{\partial \mathsf{x}} \\ &= \overline{\mathsf{x}^{(\ell+1)}} \left(\mathsf{I} + \frac{\partial \mathcal{F}}{\partial \mathsf{x}} \right)\end{aligned}$$

- As long as the Jacobian $\partial \mathcal{F} / \partial \mathsf{x}$ is small, the derivatives are stable.



Attention

- We have seen a few RNN-based sequence prediction models.
- It is still challenging to generate long sequences, when the decoders only has access to the final hidden states from the encoder.
  - Machine translation: it's hard to summarize long sentences in a single vector, so let's allow the decoder peek at the input.
  - Vision: have a network glance at one part of an image at a time, so that we can understand what information it's using

Attention

- The introduction of attention drastically improves the performance on the long sequences.
- Attention-based models scale very well with the amount of training data. After 40GB text from reddit, the model generates:

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

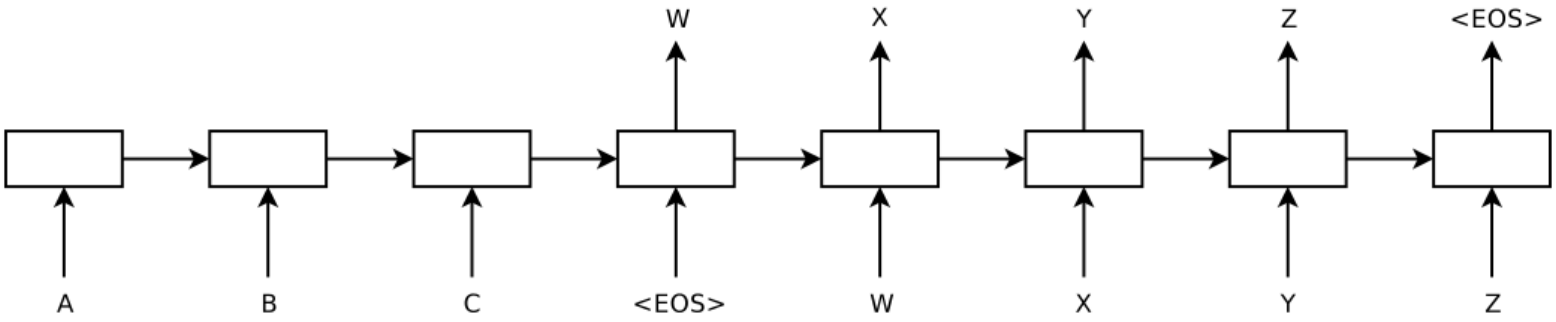
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

Attention-Based Machine Translation

- Remember the encoder/decoder architecture for machine translation:



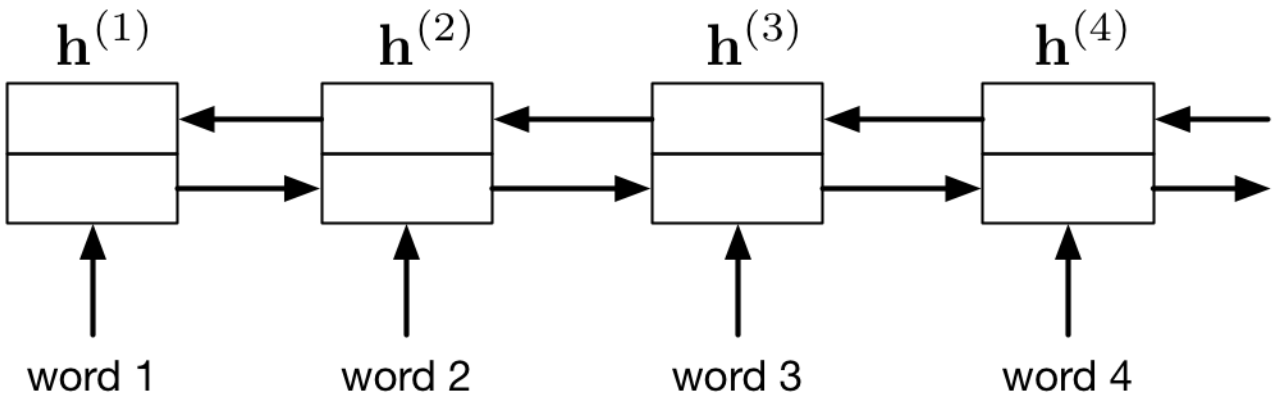
- The network reads a sentence and stores all the information in its hidden units.
- Some sentences can be really long. Can we really store all the information in a vector of hidden units?
  - Let's make things easier by letting the decoder refer to the input sentence.

Attention-Based Machine Translation

- We'll look at the translation model from the classic paper:
  Bahdanau et al., Neural machine translation by jointly learning to align and translate. ICLR, 2015.
- Basic idea: each output word comes from one word, or a handful of words, from the input. Maybe we can learn to attend to only the relevant ones as we produce the output.

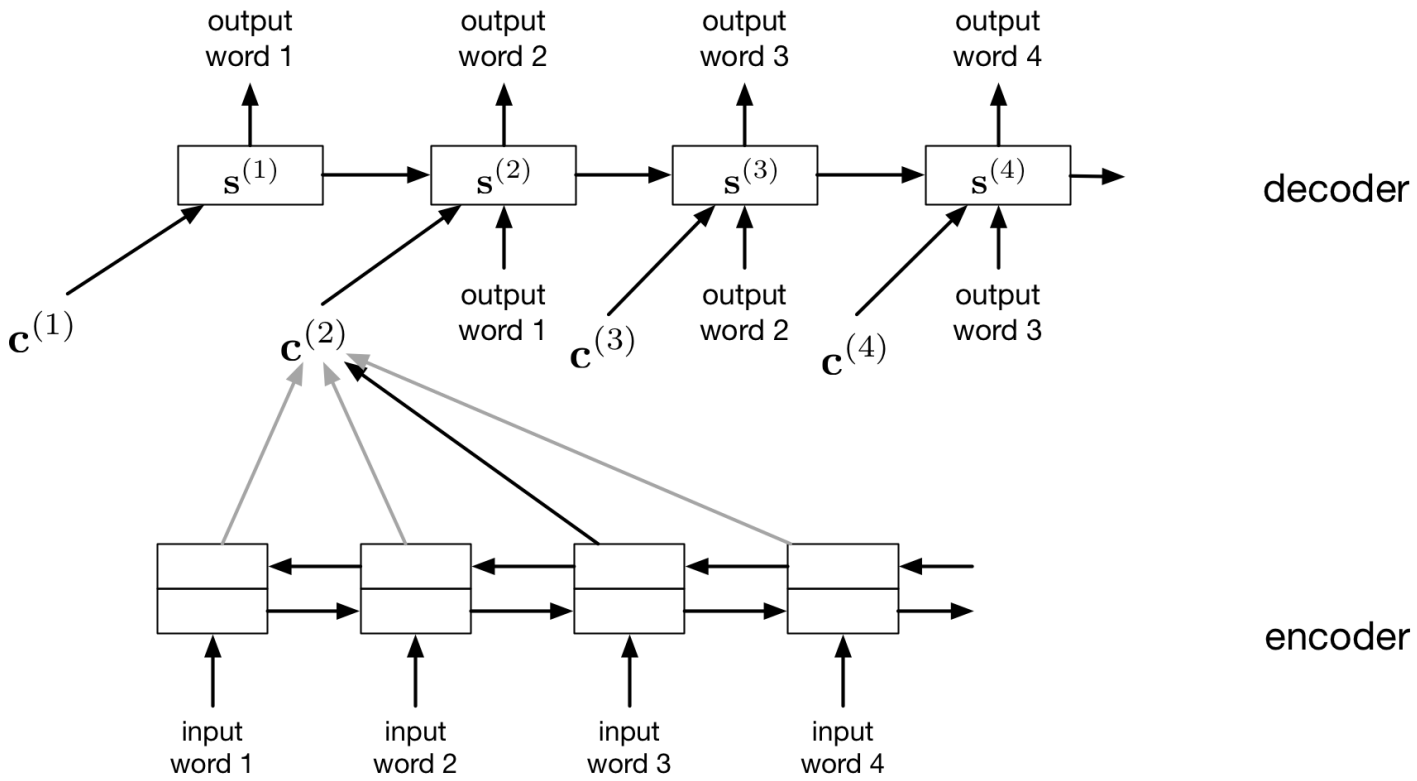
Attention-Based Machine Translation

- The model has both an encoder and a decoder. The encoder computes an annotation of each word in the input.
- It takes the form of a bidirectional RNN. This just means we have an RNN that runs forwards and an RNN that runs backwards, and we concantenate their hidden vectors.
  - The idea: information earlier or later in the sentence can help disambiguate a word, so we need both directions.
  - The RNN uses an LSTM-like architecture called gated recurrent units.



Attention-Based Machine Translation

- The decoder network is also an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and its predictions are fed back in as inputs.
- The difference is that it also receives a context vector $c^{(t)}$ at each time step, which is computed by attending to the inputs.



Attention-Based Machine Translation

- The context vector is computed as a weighted average of the encoder's annotations.

$$\mathsf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

- The attention weights are computed as a softmax, where the inputs depend on the annotation and the decoder's state:

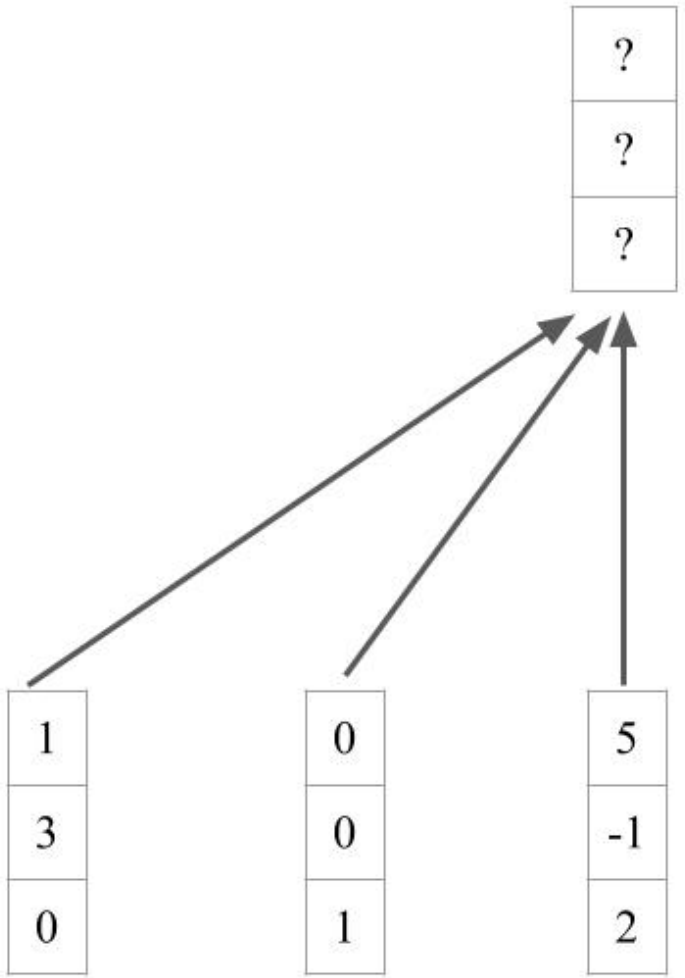
$$\alpha_{ij} = \frac{\exp(\tilde{\alpha}_{ij})}{\sum_{j'} \exp(\tilde{\alpha}_{ij'})}$$

$$\tilde{\alpha}_{ij} = f(\mathsf{s}^{(i-1)}, \mathsf{h}^{(j)})$$

- Note that the attention function, f depends on the annotation vector, rather than the position in the sentence. This means it's a form of content-based addressing.
  - My language model tells me the next word should be an adjective. Find me an adjective in the input.

Example: Pooling

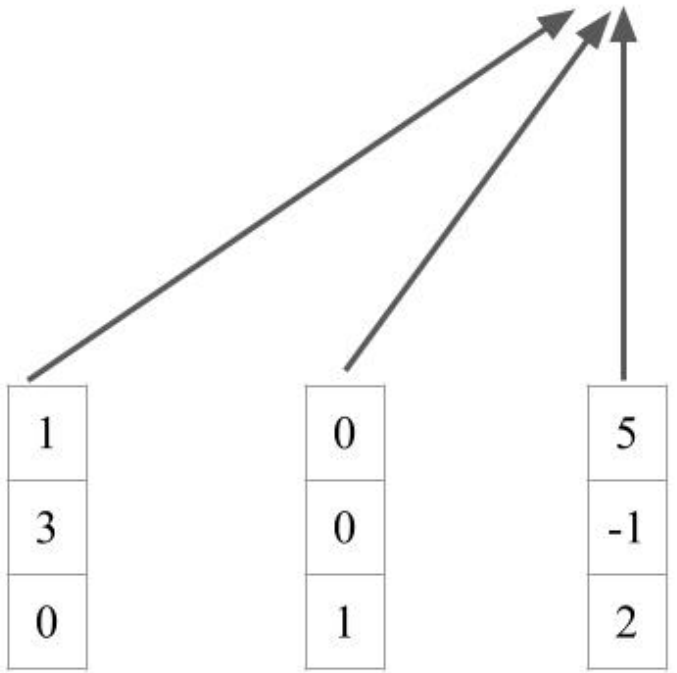
Consider obtain a context vector from a set of annotations.



Example: Pooling

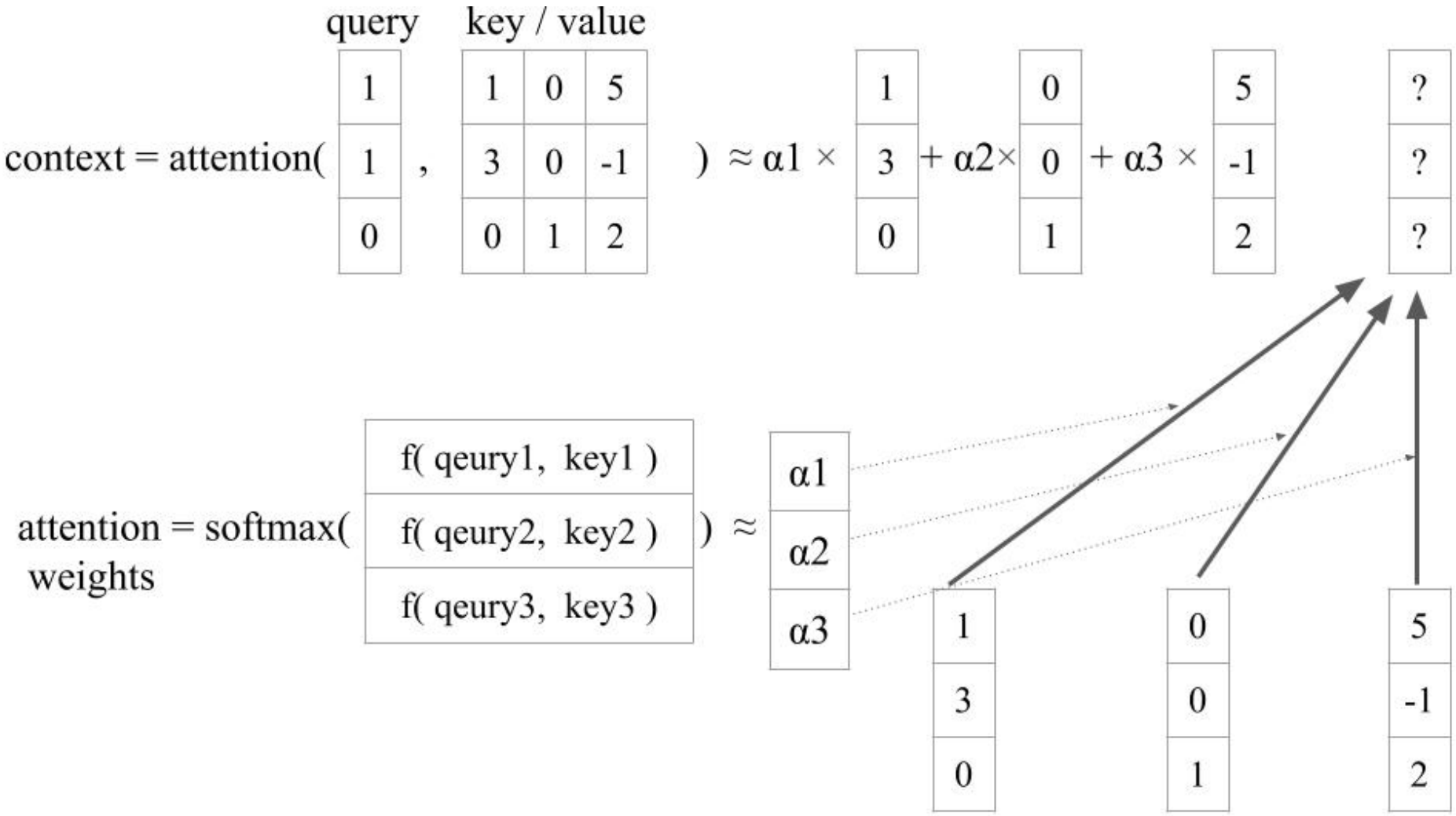
We can use average pooling but it is content independent.

$$\text{context} = \text{avg-pooling}\left(\begin{array}{|c|c|c|}\hline 1 & 0 & 5 \\ \hline 3 & 0 & -1 \\ \hline 0 & 1 & 2 \\ \hline\end{array}\right) = 0.33\times\begin{array}{|c|}\hline 1 \\ \hline 3 \\ \hline 0 \\ \hline\end{array} + 0.33\times\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline\end{array} + 0.33\times\begin{array}{|c|}\hline 5 \\ \hline -1 \\ \hline 2 \\ \hline\end{array} = \begin{array}{|c|}\hline 2 \\ \hline 0.6 \\ \hline 1 \\ \hline\end{array}$$



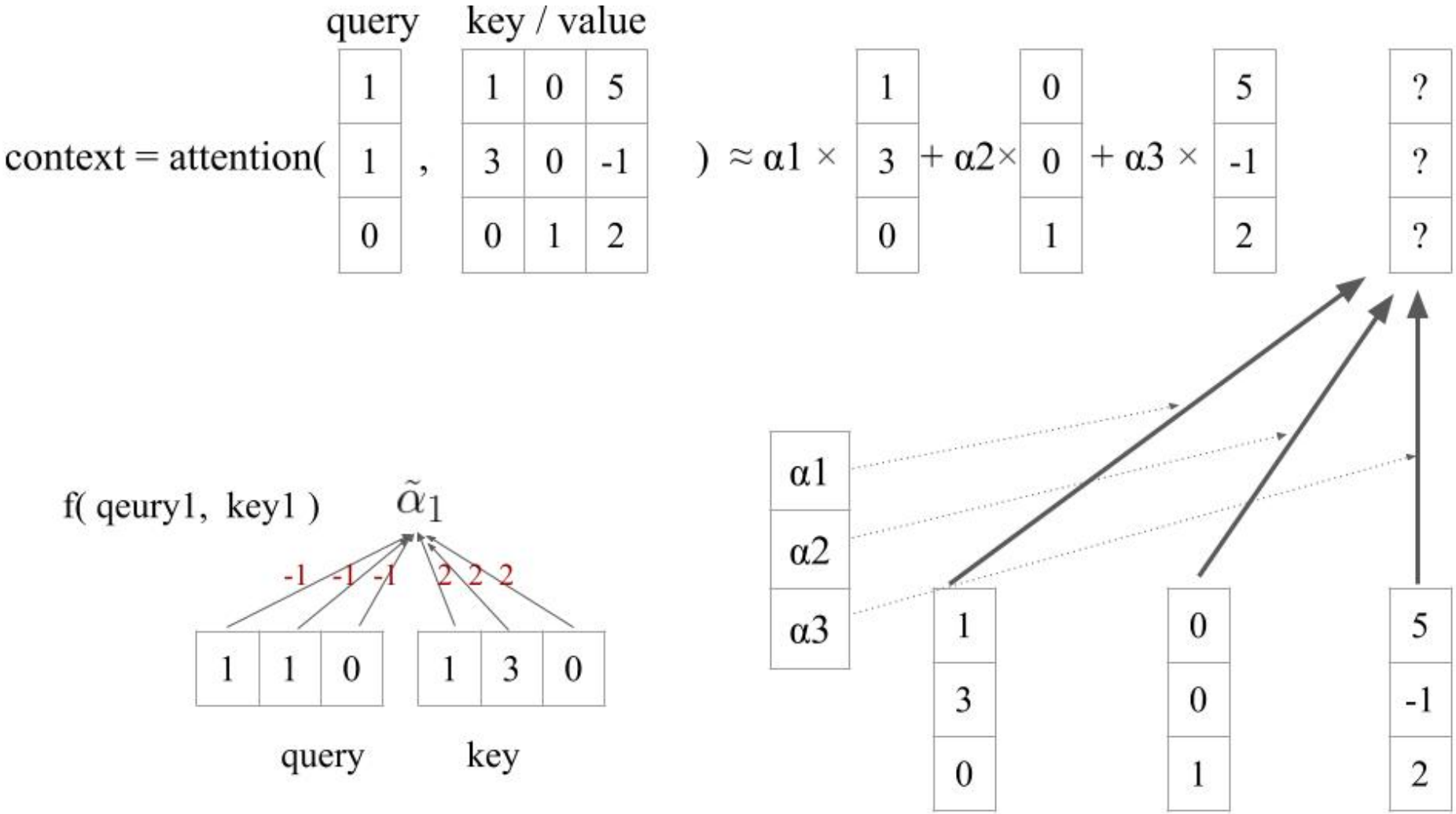
Example1: Bahdanau's Attention

Content-based addressing/lookup using attention.



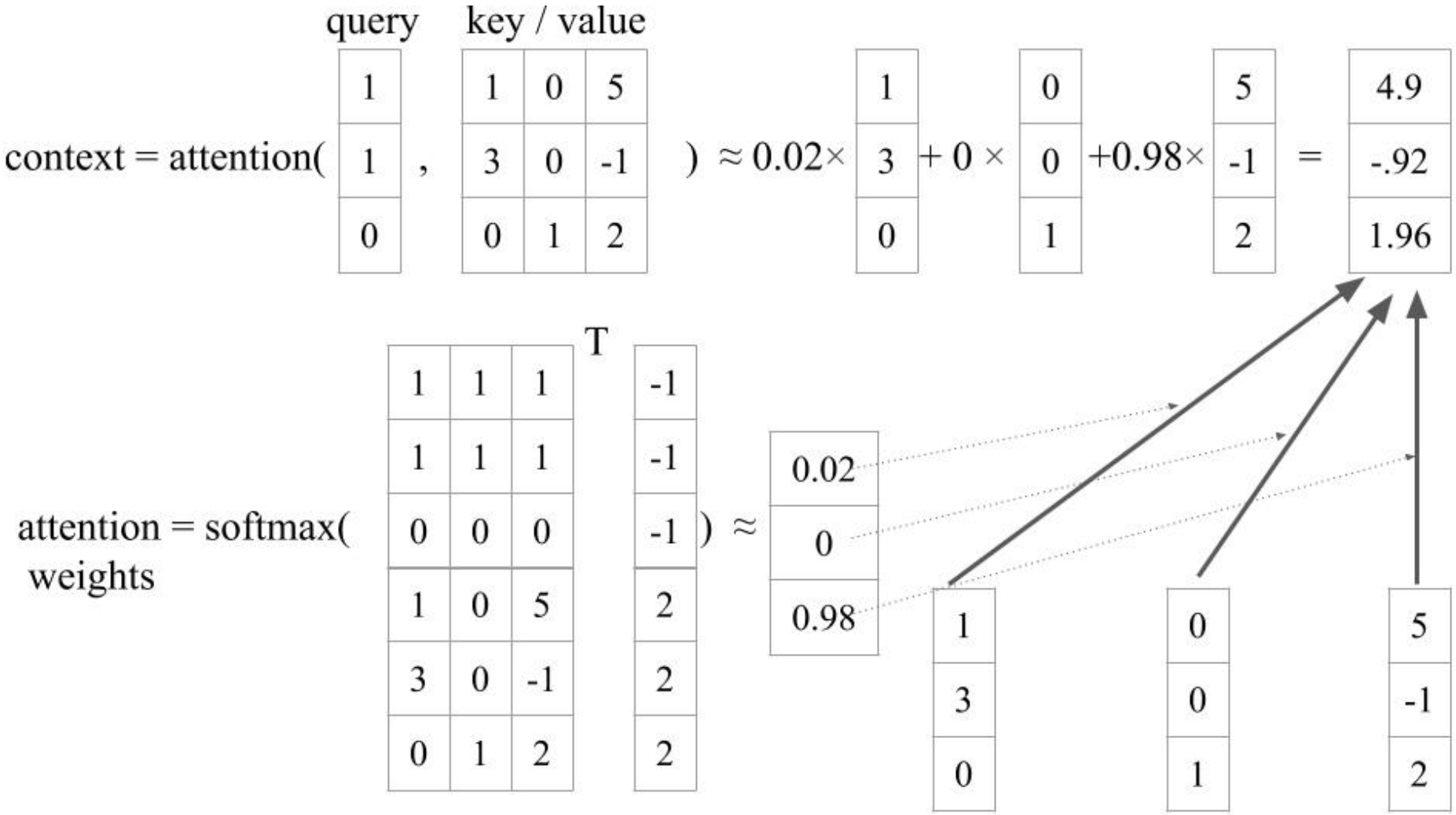
Example1: Bahdanau's Attention

Consider a linear attention function, f.



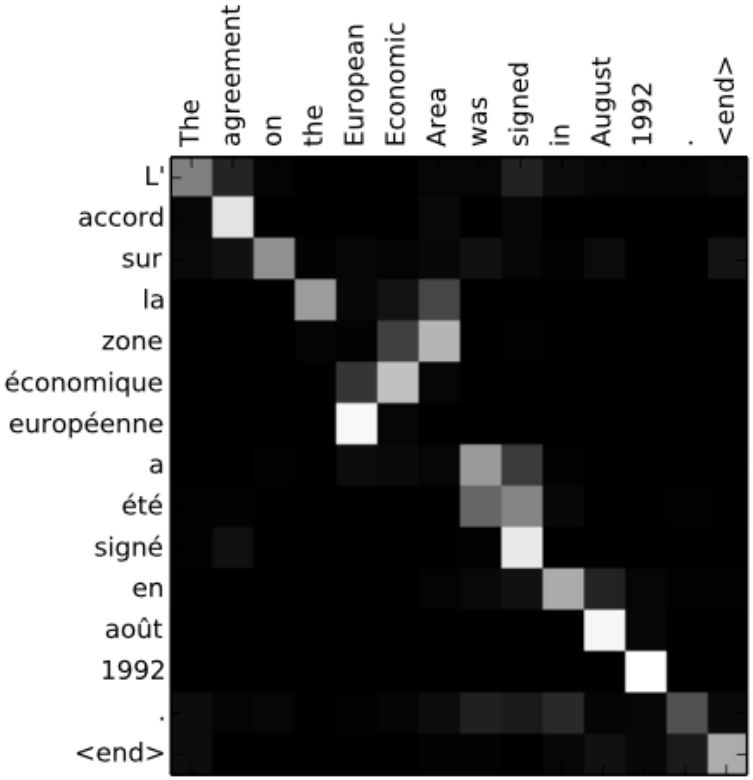
Example1: Bahdanau's Attention

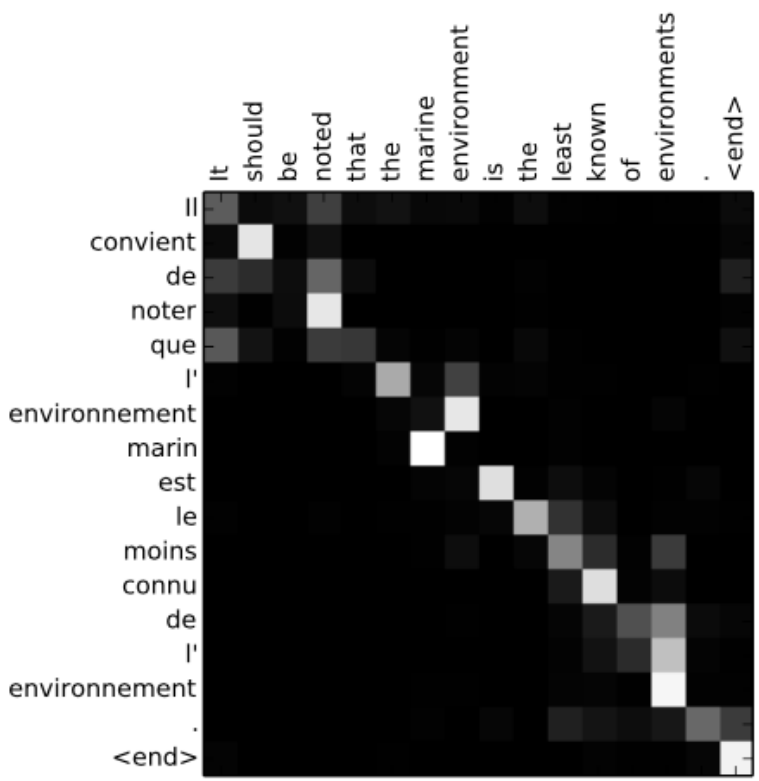
Vectorized linear attention function.



Attention-Based Machine Translation

- Here's a visualization of the attention maps at each time step.

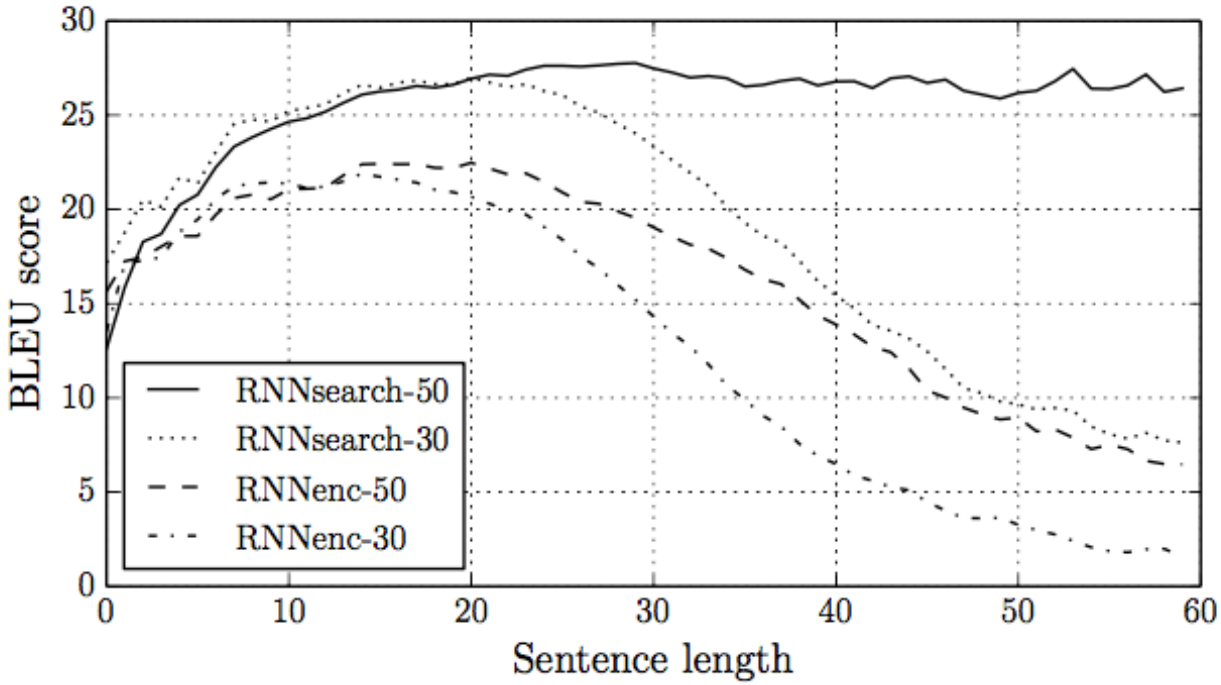




- Nothing forces the model to go linearly through the input sentence, but somehow it learns to do it.
  - It's not perfectly linear — e.g., French adjectives can come after the nouns.

Attention-Based Machine Translation

- The attention-based translation model does much better than the encoder/decoder model on long sentences.



Attention-Based Caption Generation

- Attention can also be used to understand images.
- We humans can't process a whole visual scene at once.
  - The fovea of the eye gives us high-acuity vision in only a tiny region of our field of view.
  - Instead, we must integrate information from a series of glimpses.
- The next few slides are based on this paper from the UofT machine learning group:

  Xu et al. Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

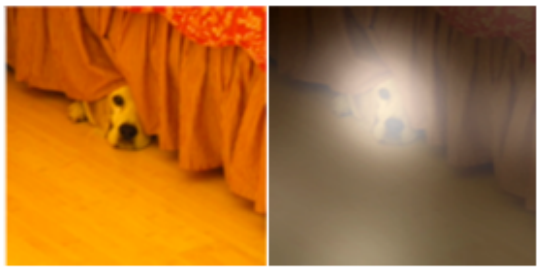
Attention-Based Caption Generation

- The caption generation task: take an image as input, and produce a sentence describing the image.
- **Encoder:** a classification conv net (VGGNet, similar to AlexNet). This computes a bunch of feature maps over the image.
- **Decoder:** an attention-based RNN, analogous to the decoder in the translation model
  - In each time step, the decoder computes an attention map over the entire image, effectively deciding which regions to focus on.
  - It receives a context vector, which is the weighted average of the conv net features.

Attention-Based Caption Generation

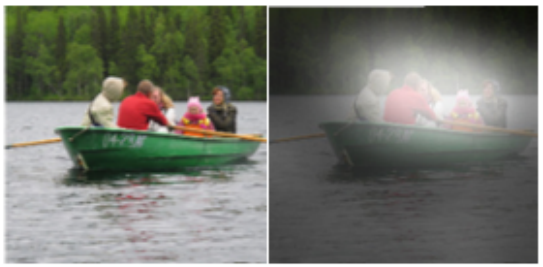
- This lets us understand where the network is looking as it generates a sentence.


A woman is throwing a frisbee in a park.


A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.


A little girl sitting on a bed with a teddy bear.


A group of people sitting on a boat in the water.


A giraffe standing in a forest with trees in the background.

Attention-Based Caption Generation

- This can also help us understand the network's mistakes.

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and
a hat on a skateboard.

A person is standing on a beach
with a surfboard.

A woman is sitting at a table
with a large pizza.

A man is talking on his cell phone
while another man watches.