CSC321 Neural Networks and Machine Learning

Lecture 11

March 25, 2020

Agenda

- Deep Residual Networks (CNN)
- Attention (RNN)
- Project 4 Miscellaneous
  - Data augmentation
  - Text generation (see tutorial)

Exam Vote

Exam Date:

- April 6, 9am-11am: 20 votes (17%)
- **April 16, 5pm-7pm: 98 votes (83%)**

Exam Format

- **Open book: 106 votes (90%)**
- Closed book: 12 votes (10%)

We're still thinking about the exam format. If you have a possible accessibility concern please let Lisa know!

Deadline Vote

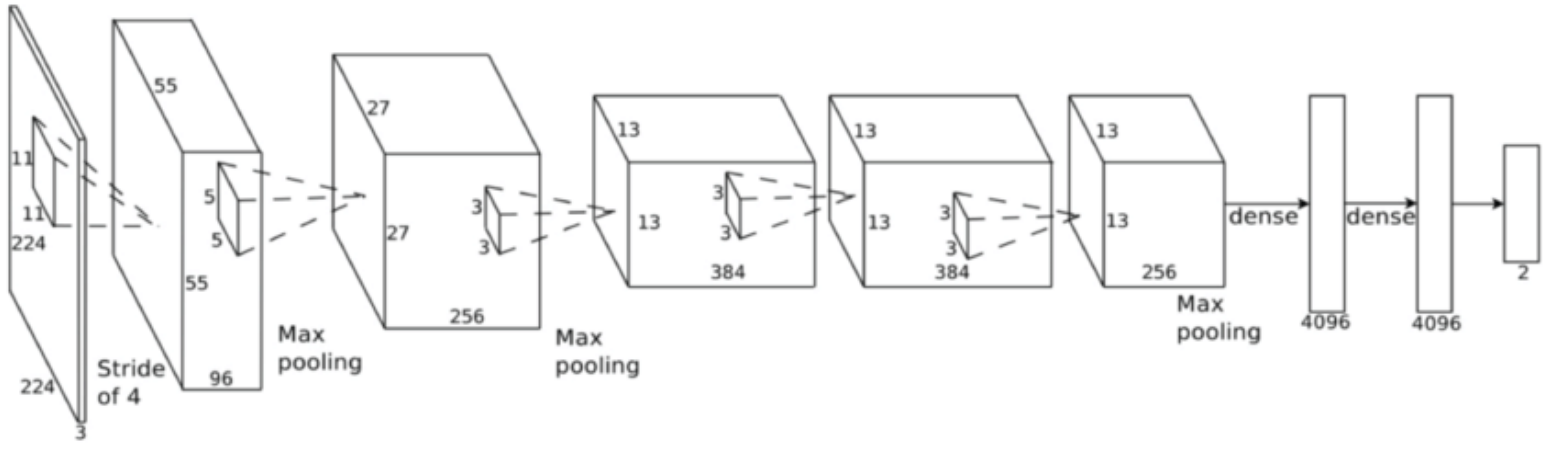
- April 2: 10 votes (8%)
- April 3: 12 votes (10%)
- April 6: 17 votes (14%)
- April 7: 9 votes (8%)
- **April 8: 70 votes (59%)**

This means that homework 5 will be due **Monday, March 30th.**

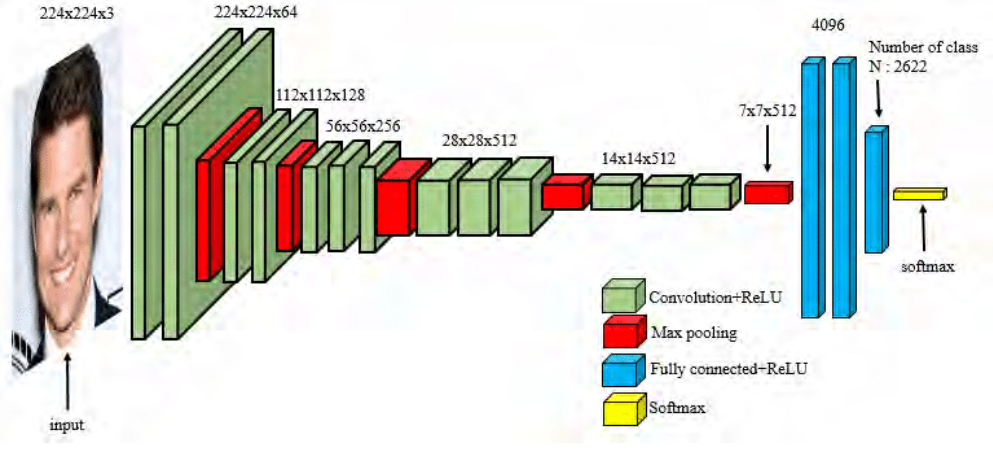
CNN Architecture

Review: two CNN Architectures

AlexNet:



VGG:



Modern CNN Architectures

More recent CNN architectures tend to be **deeper**.

Q: Why do deeper networks generally perform better?

Modern CNN Architectures

More recent CNN architectures tend to be **deeper**.

Q: Why do deeper networks generally perform better?

- Deep networks capture the "hierarchy" in features (e.g. low, mid, high level features in CNN), and therefore learn **better features**
- Deep networks are more **parsimonous**—i.e. requires fewer number of paremeters
  - In theory, two-layers is enough: two-layer neural networks are *universal approximators* (lecture 4). However, a two-layer neural network requires exponentially more units, and therefore more parameters

But learning deep neural networks is hard!

Gradient Explosion and Vanishing in RNNs

Recall: the matrix of partial derivatives (or the **Jacobian**) $\frac{\partial h^{(T)}}{\partial h^{(1)}}$ is the product:

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

If each term in the product is "small" (less than 1 in the case that h is scalar) then the gradients **vanish**.

If each term in the product is "large" (larger than 1 in the case that h is scalar) then the gradients **explode**.

Gradient Explosion and Vanishing in RNNs

Recall: the matrix of partial derivatives (or the **Jacobian**) $\frac{\partial h^{(T)}}{\partial h^{(1)}}$ is the product:

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

If each term in the product is "small" (less than 1 in the case that h is scalar) then the gradients **vanish**.

If each term in the product is "large" (larger than 1 in the case that h is scalar) then the gradients **explode**.

If $h^{(2)} = \phi(Wh^{(1)} + b)$, where W is not reused:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W} \\ &= \frac{\partial \mathcal{L}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(2)}} \phi'(Wh^{(1)} + b) h^{(1)}\end{aligned}$$

Gradient Explosion and Vanishing in CNNs

In a deep convolutional neural network, we have similar multiplication of partial derivatives!

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

$$\frac{\partial \mathcal{L}}{\partial W^{(1)}} = \frac{\partial \mathcal{L}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W^{(1)}}$$

Here, t is indexed over **layers** rather than over **time**.

If the derivatives $\frac{\partial h^{(t)}}{\partial h^{(t-1)}}$ tend to be "small", we'll get gradient vanishing, and we'll have poor gradient signals.

Why didn't it come up?

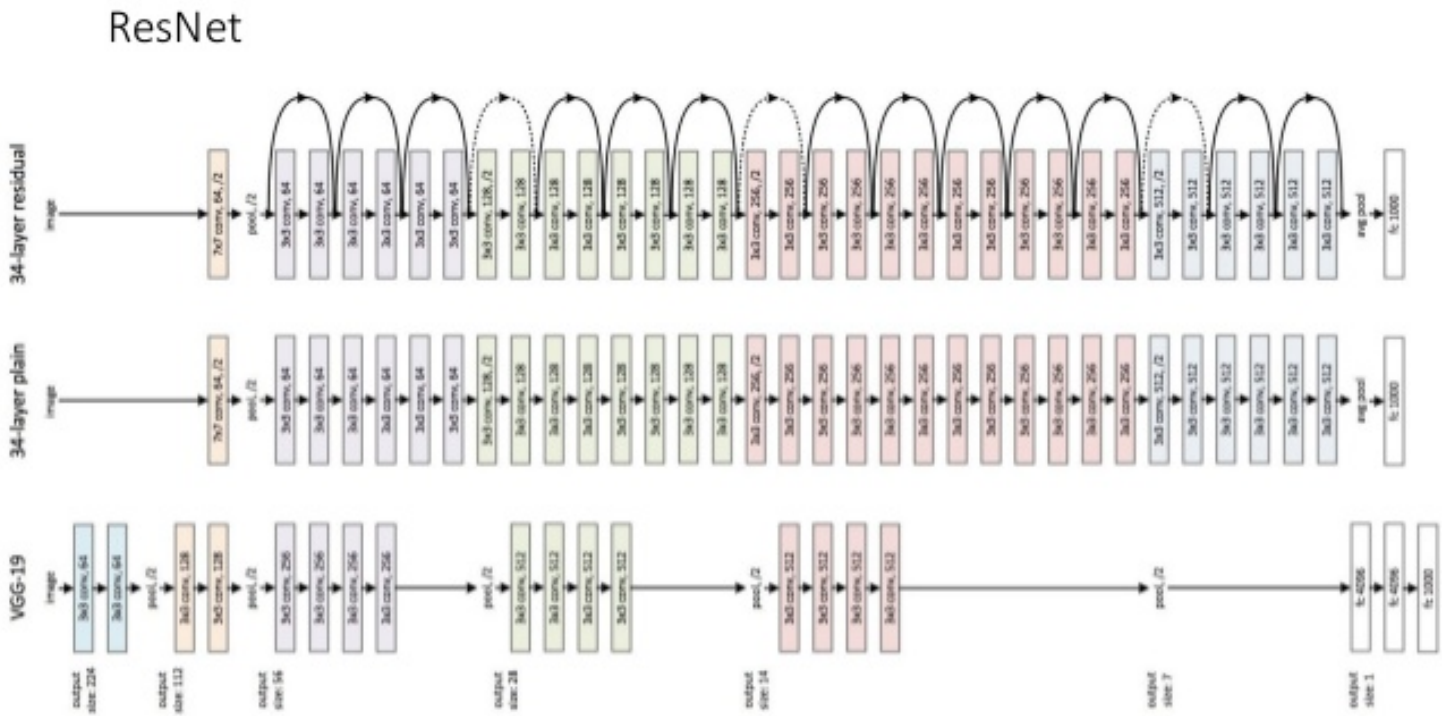
Why didn't we have to worry about exploding and vanishing gradients until we talked about RNNs?

- For a long time, MLPs and conv nets were not very deep (~10 layers)
- In contrast, RNNs could be run over hundreds of time steps

If we want to train a very deep convolutional network (e.g. hundreds of layers) we **do** need to worry exploding/vanishing gradients!

Residual Network

One of the best convolutional network for ImageNet (image classification challenge) is the **residual network**:



This network uses up to more than 100 layers!

The Residual Block

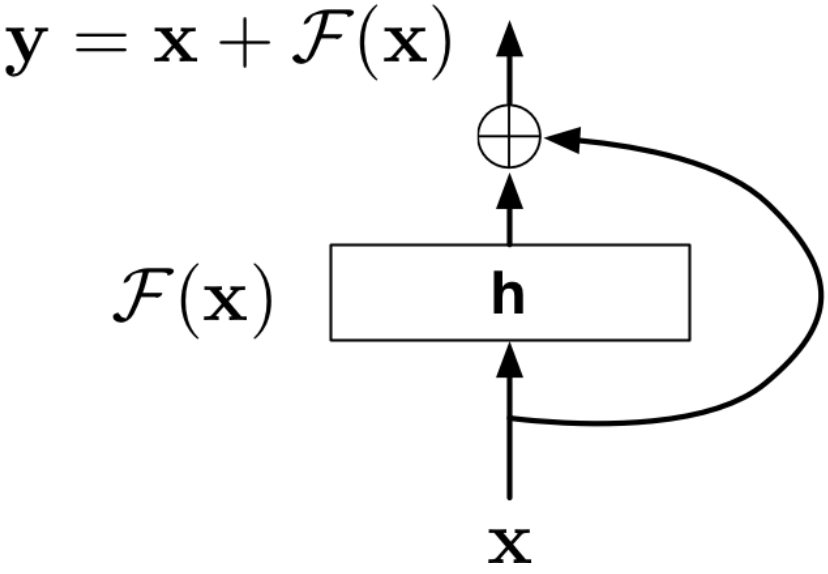
Just like in an RNN, we need to improve the **architecture** of our CNN model to ensure good gradient flow. We can no longer use our usual `conv -> pool -> relu` blocks.

This is called a **Residual Block**.

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

The Residual Block

General form of the residual block:

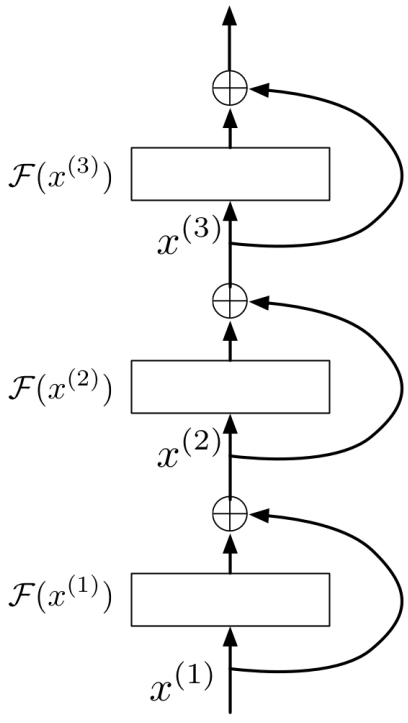


Rather than having each layer produce an entirely new value, each layer *adds something*, i.e. a *residual*, to the previous value.

In general, the network for $\mathcal{F}$ can have multiple layers, be convolutional, etc.

Deep Residual Networks

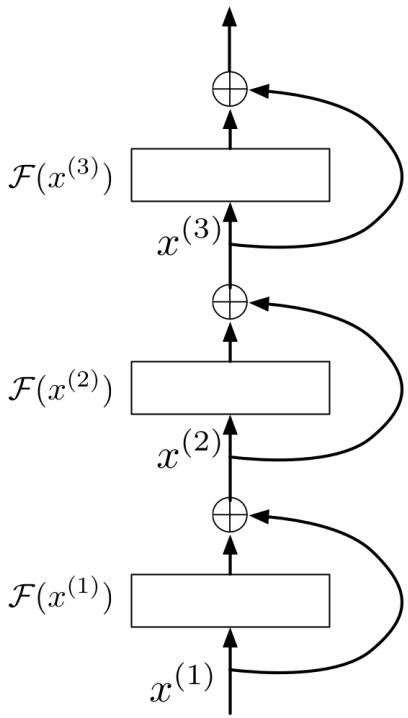
We can string together a bunch of these residual blocks to form a network.



Q: What happens if we set the parameters such that $\mathcal{F}(\mathbf{x}^{(\ell)}) = 0$ in every layer?

Deep Residual Networks

We can string together a bunch of these residual blocks to form a network.



Q: What happens if we set the parameters such that $\mathcal{F}(\mathbf{x}^{(\ell)}) = 0$ in every layer?

A: Then it passes $\mathbf{x}^{(\ell)}$ straight through unmodified! This means it's easy for the network to represent the identity function.

Deep Residual Networks Math

Forward pass:

$$\mathbf{x}^{(\ell+1)} = \mathbf{x}^{(\ell)} + \mathcal{F}(\mathbf{x}^{(\ell)})$$

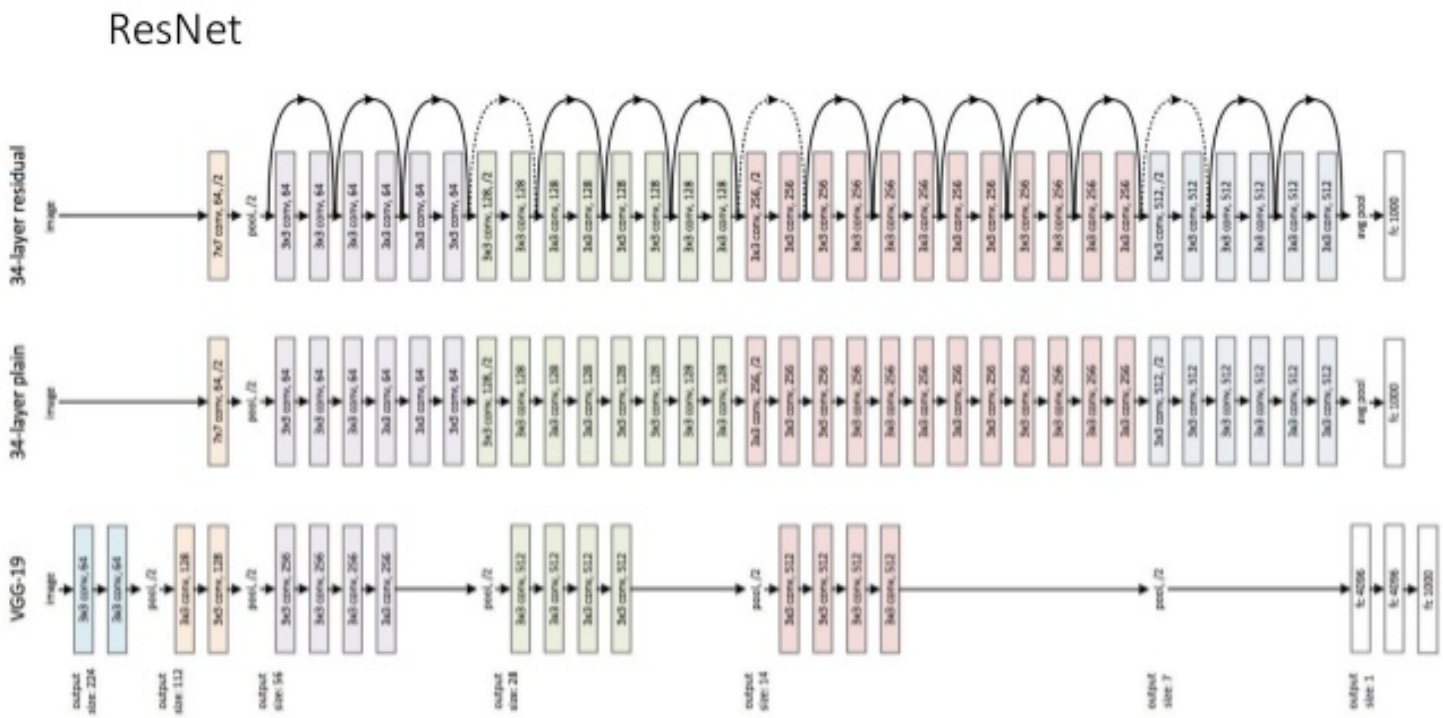
Backward pass:

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

As long as $\frac{\partial \mathcal{F}}{\partial \mathbf{x}}$ is small, the derivatives are stable.

ResNet

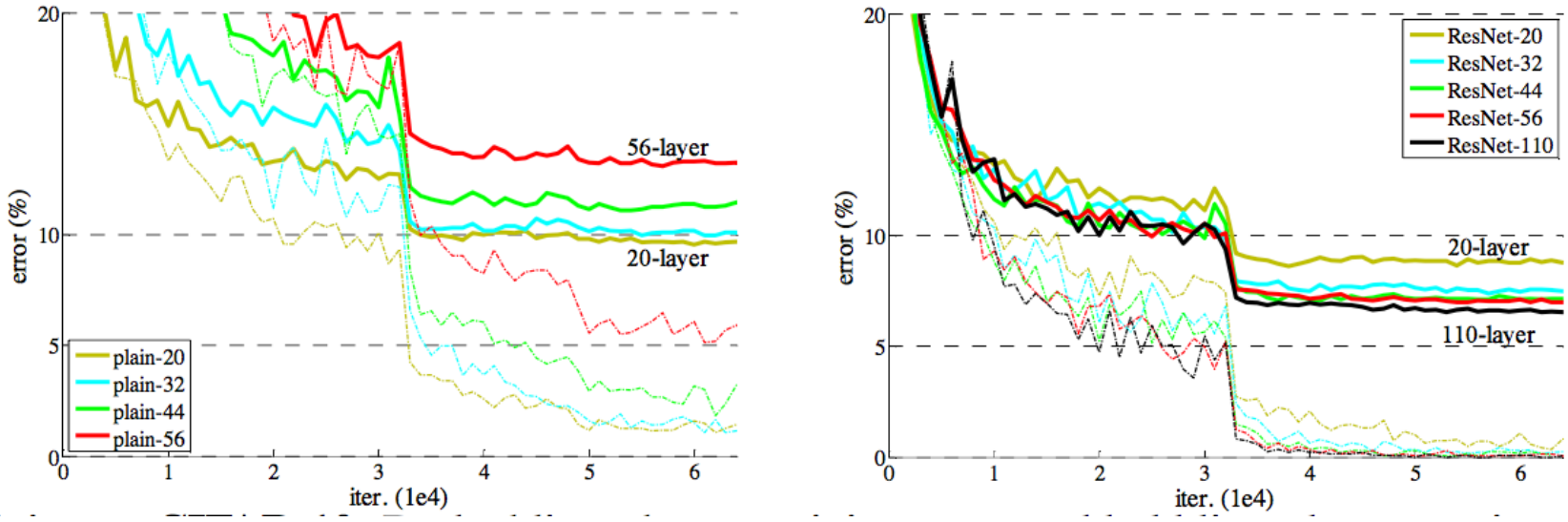
Deep Residual Networks (ResNets) consist of many layers of residual blocks.



For vision tasks, the $\mathcal{F}$ functions are usually 2- or 3-layer conv nets.

Regular ConvNet vs ResNet

Performance on CIFAR-10, a small object recognition dataset:



For a regular convnet (left), performance declines with depth, but for a ResNet (right), it keeps improving!

Deep Residual Networks for ImageNet

ImageNet Performance:

- A 152-layer ResNet achieved 4.49% top-5 error on Image Net. (An ensemble of them achieved 3.57%.)
- Previous state-of-the-art: 6.6% (GoogLeNet)
- Humans: 5.1%

Researchers were able to train ResNets with more than 1000 layers, but classification performance leveled off by 150.

What are all these layers doing? We don't have a clear answer, but the idea that they're computing increasingly abstract features is starting to sound fishy...

Recurrent Neural Networks with Attention

Recurrent Neural Networks

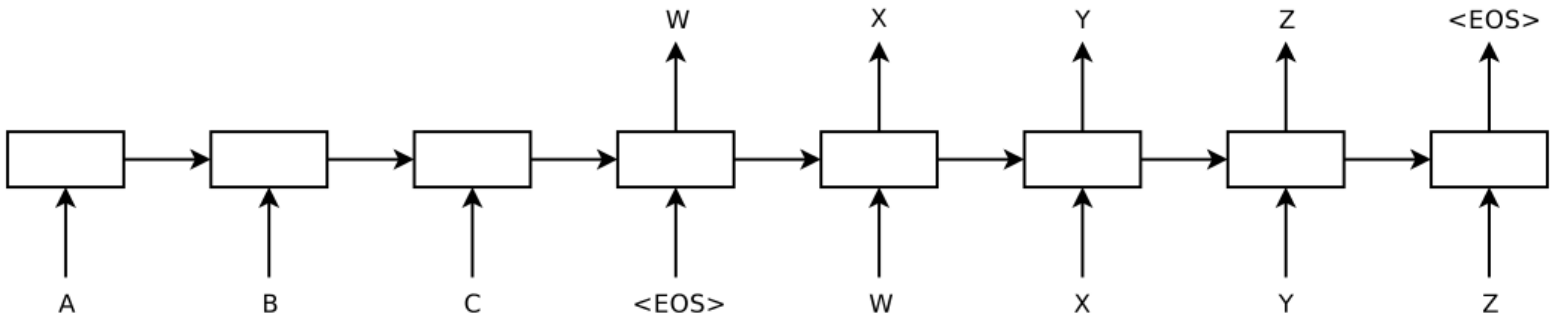
Last week, we showed a **discriminative RNN** that makes a *prediction* based on a sequence (sequence as an input).

In tutorial 11, we build a **generator RNN** to generate sequences (sequence as an output)

In project 4, we use an **encoder/decoder architecture** to perform a sequence-to-task (both inputs and outputs are sequences)

Sequence-to-sequence tasks

Another common example of a sequence-to-sequence task (seq2seq) is **machine translation**.



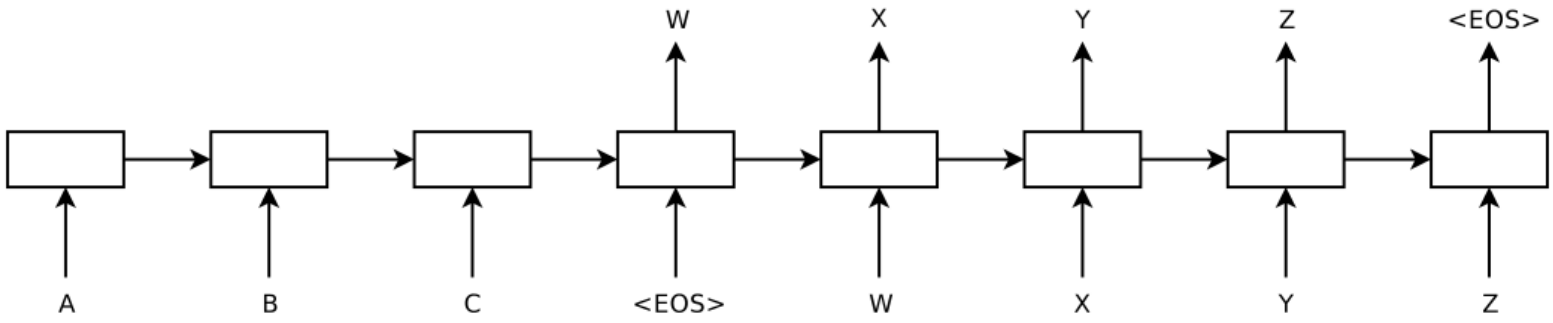
Input: English sentence (as a sequence of words)

Output: French sentence (as a sequence of words)

How Seq2Seq works

The **encoder network** reads an input sentence and stores all the information in its hidden units.

The **decoder network** then generates the output sentence one word at a time.



But some sentences can be really long. Can we really store all the information in a vector of hidden units?

Human translators **refer back to the input**.

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.*

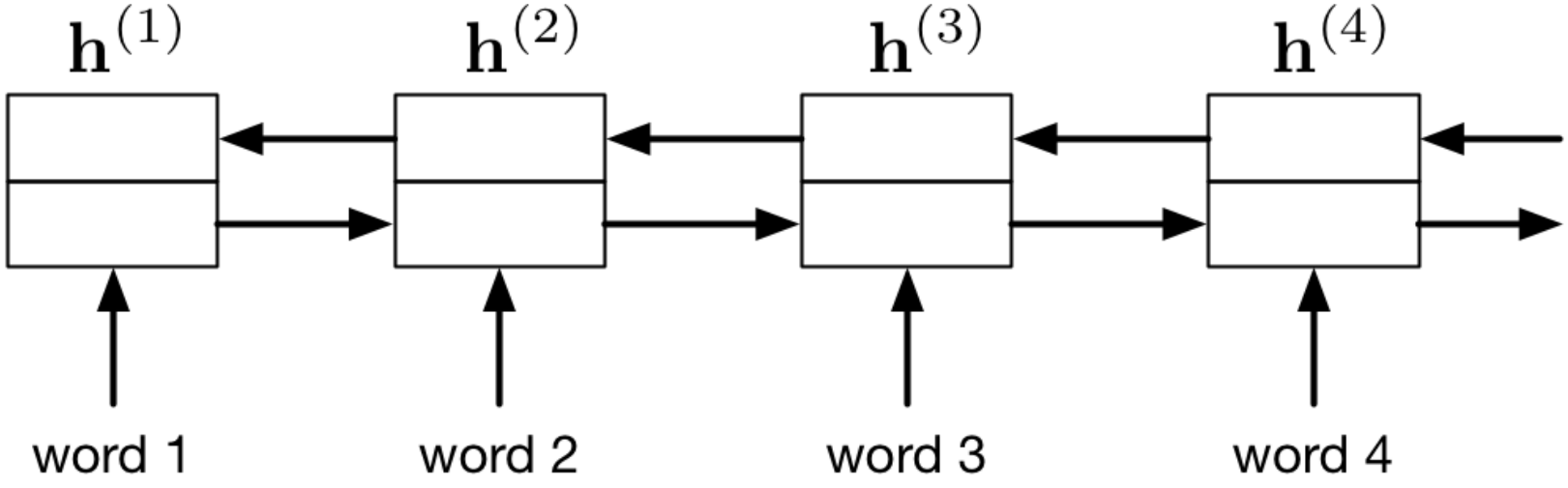
Basic idea: each **output word** comes from **one input word**, or a handful of input words. Maybe we can learn to attend to only the relevant ones as we produce the output.

Attention-Based Machine Translation Encoder

The encoder computes an **annotation** (hidden state) of each word in the input.

The encoder is a **bidirectional RNN**. We have two RNNs: one that runs forward and one that runs backwards. These RNNs can be LSTMs or GRUs.

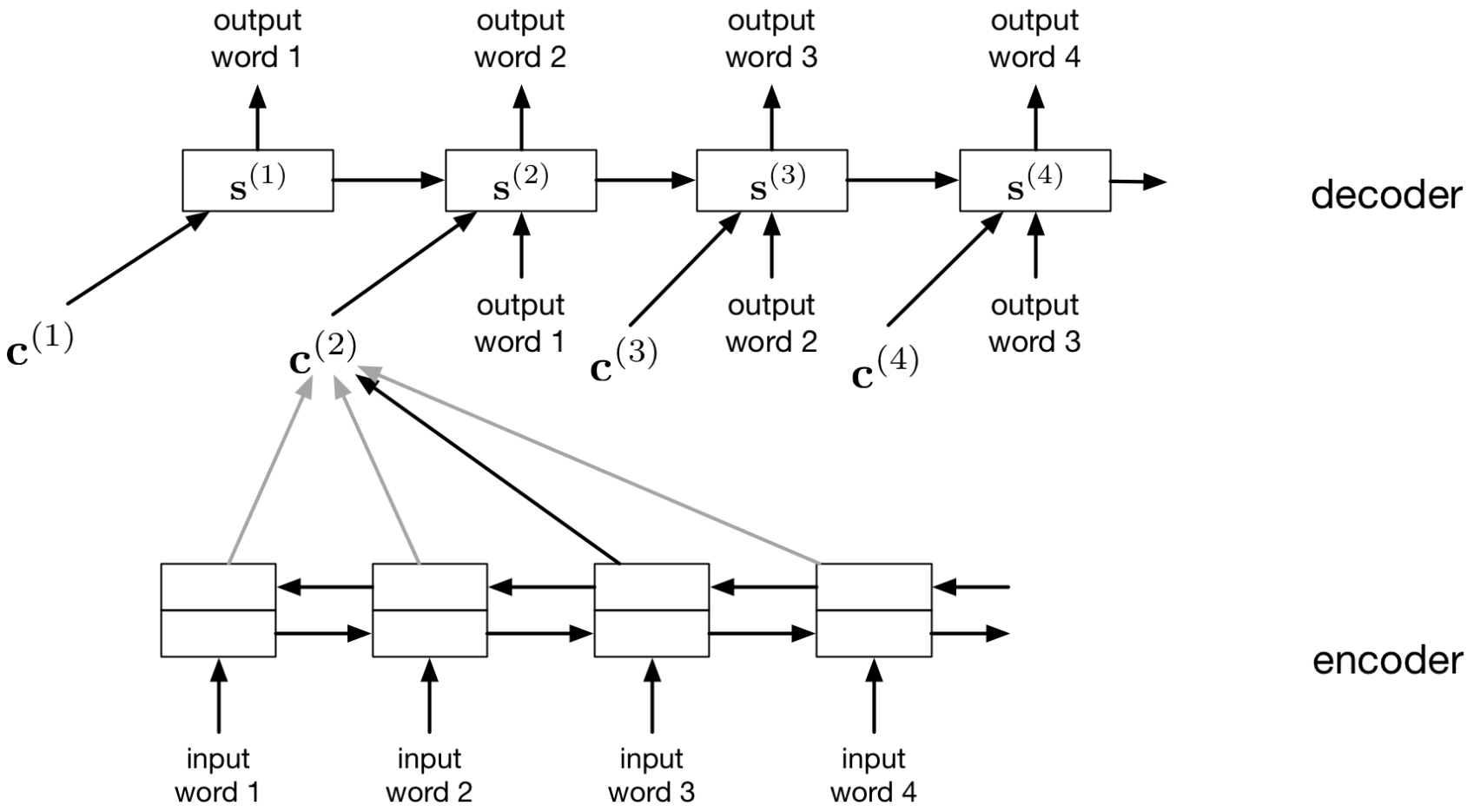
The annotation of a word is the concatenation of the forward and backward hidden vectors.



Attention-Based Machine Translation: Decoder

The decoder network is also an RNN, and makes predictions one word at a time.

The difference is that it also derives a **context vector** $\mathbf{c}^{(t)}$ at each time step, computed by **attending to the inputs**



The math behind "attending to the input"

The context vector is computed as a **weighted average** of the encoder's annotations:

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

The attention weights are computed as a softmax, where the input depends on the *annotation* $\mathbf{h}^{(j)}$ and the *decoder states* $\mathbf{s}^{(t)}$:

$$e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} exp(e_{ij'})}$$

The attention function depends on the **annotation vector**, rather than the **position in the sentence**. It is 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"

Intuition behind "attending to the input"

> *"My language model tells me the next word should be an adjective. Find me an adjective in the input"*

We would like to *refer back* to one (or a few) of the input words to help with the translation task (e.g. find the adjective)

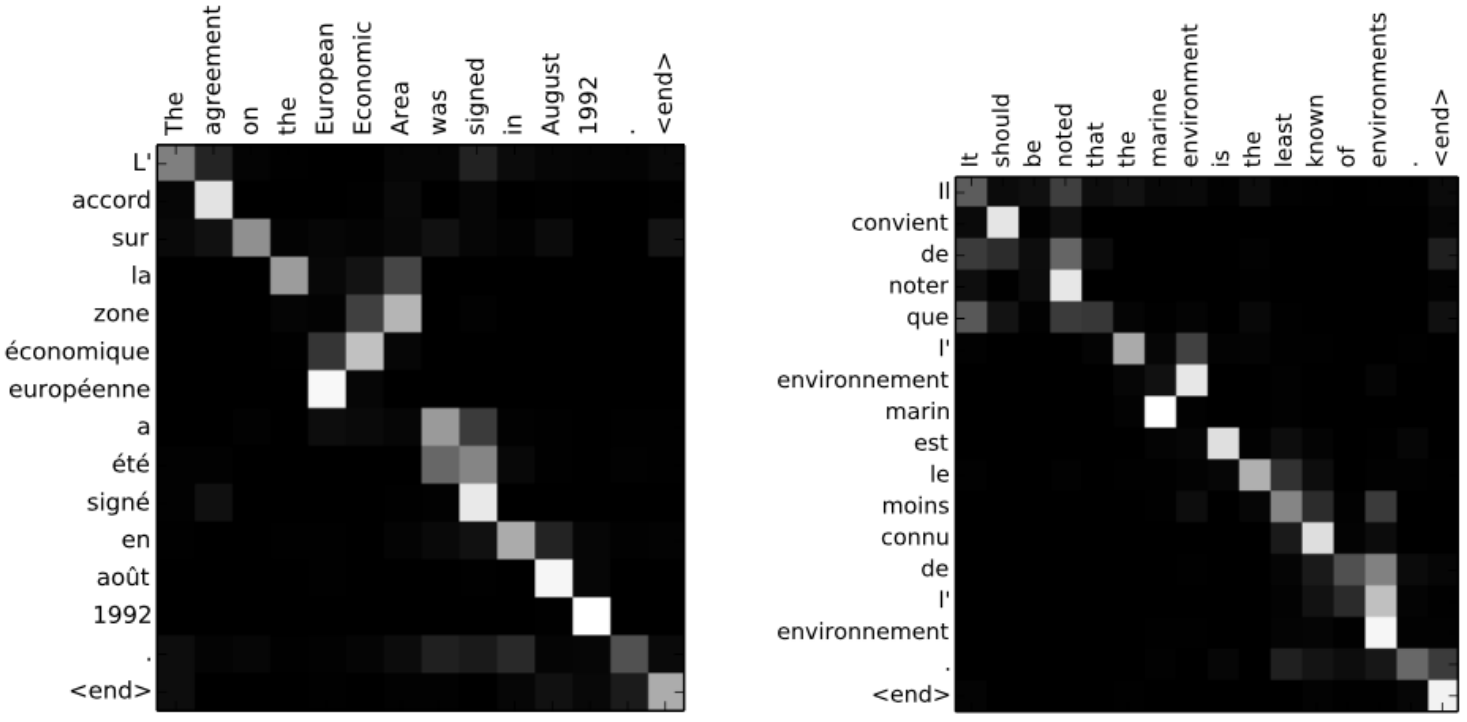
If you were programming a translator, you might

1. find an input word that is most likely an adjective (the attention function)
2. look up the input word that is the adjective (the weighted average)
3. translate the input word, e.g. using the input word, and a dictionary (the projection MLP)

An attentional decoder is like a *continuous* form of these last three steps.

Visualization of Attention

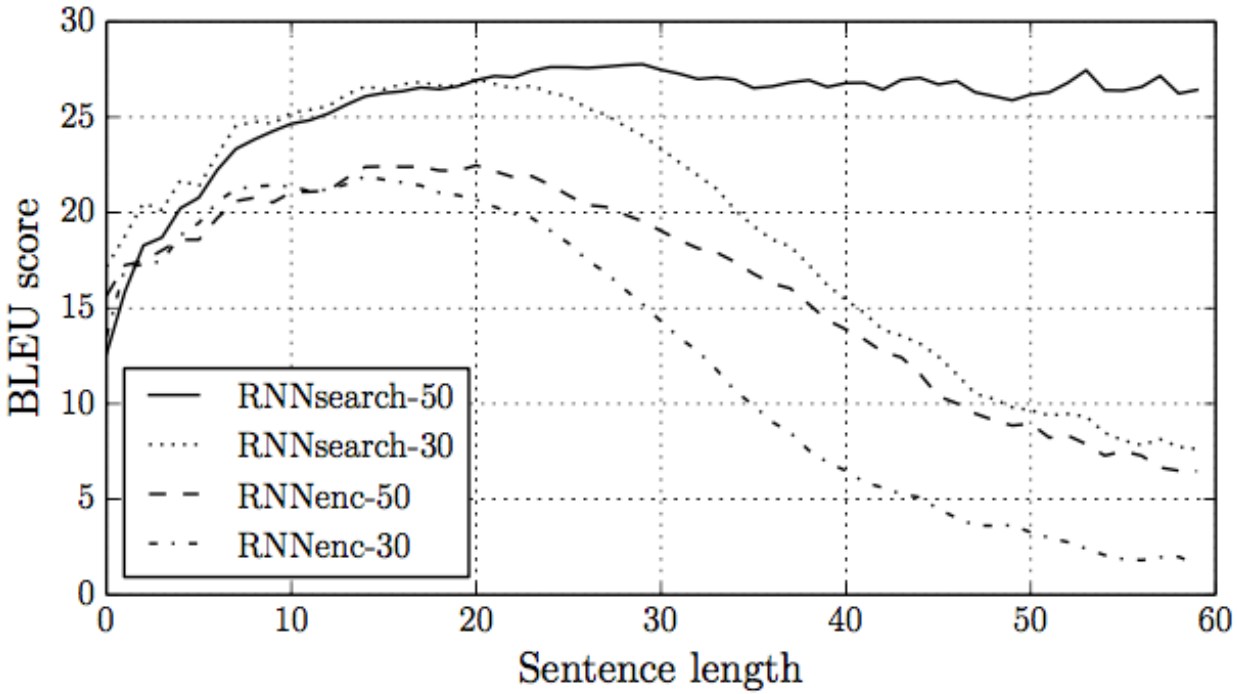
Visualization of the attention map (the α_{ij}s at each time step)



Nothing forces the model to go (roughly) linearly through the input sentences, but somehow it learns to do it!

Attention Performance

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



Attention-Based Caption Generation

Caption Generation Task:

- Input: Image
- Output: Caption (sequence of words or characters)

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 for Caption Generation

The caption generation task: take an image as input, and produce a sentence describing the image.

- **Encoder**: a classification conv net like VGG. 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.

Similar math as before: difference is that j is a pixel location

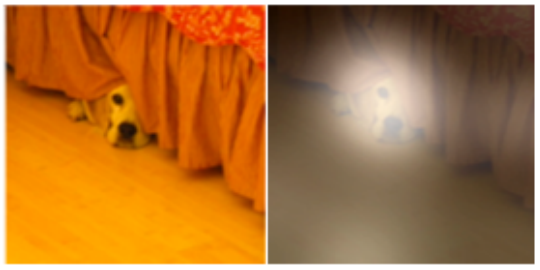
$$e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} exp(e_{ij'})}$$

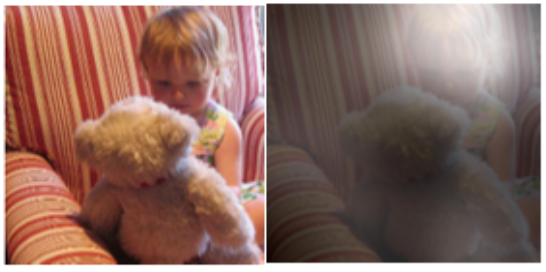
What Attention tells us

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.

What Attention tells us about mistakes

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.

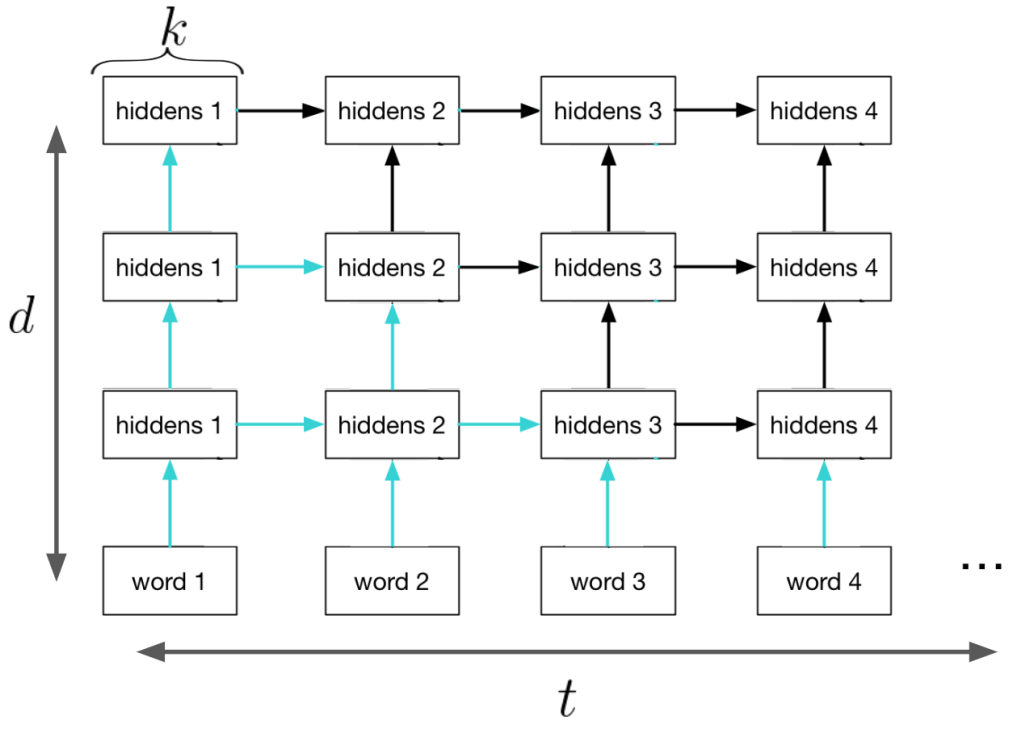
A woman is sitting at a table
with a large pizza.

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

Language Models Today

Multi-Layer RNNs (not on exam)

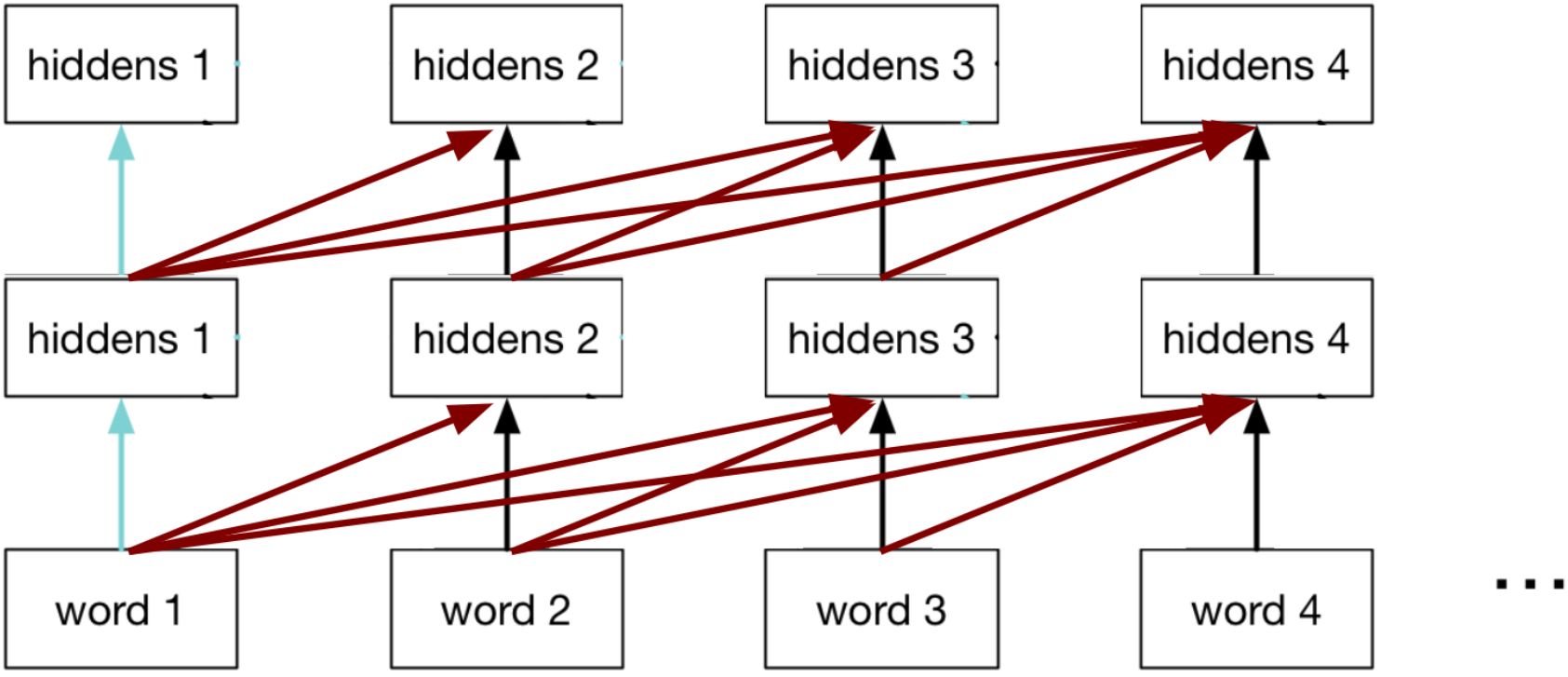
If you need to have multi-layer RNN's, then the hidden state of your first RNN becomes the input to your second layer RNN.



One disadvantage of RNNS (and especially multi-layer RNNs) is that they require a long time to train, and are more difficult to parallelize. (Need the previous hidden state $h^{(t)}$ to be able to compute $h^{(t+1)}$)

Transformer (not on exam)

Idea: Do away with recurrent networks altogether; instead exclusively use attention.



Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

Better Language Models (not on exam)

https://openai.com/blog/better-language-models/

- Input: Human-Written Prompt (small paragraph)
- Output: Article about the topic (many paragraphs)

Project 4

Data Augmentation

Another way to prevent overfitting artificially increase the amount of training data by *adding random noise* to the data.

Example: AlexNet used random crops of input images

Data Augmentation

Another way to prevent overfitting artificially increase the amount of training data by *adding random noise* to the data.

Example: AlexNet used random crops of input images

Q: Given that the goal of data augmentation is to prevent overfitting, would you also augment your validation data?

Data Augmentation

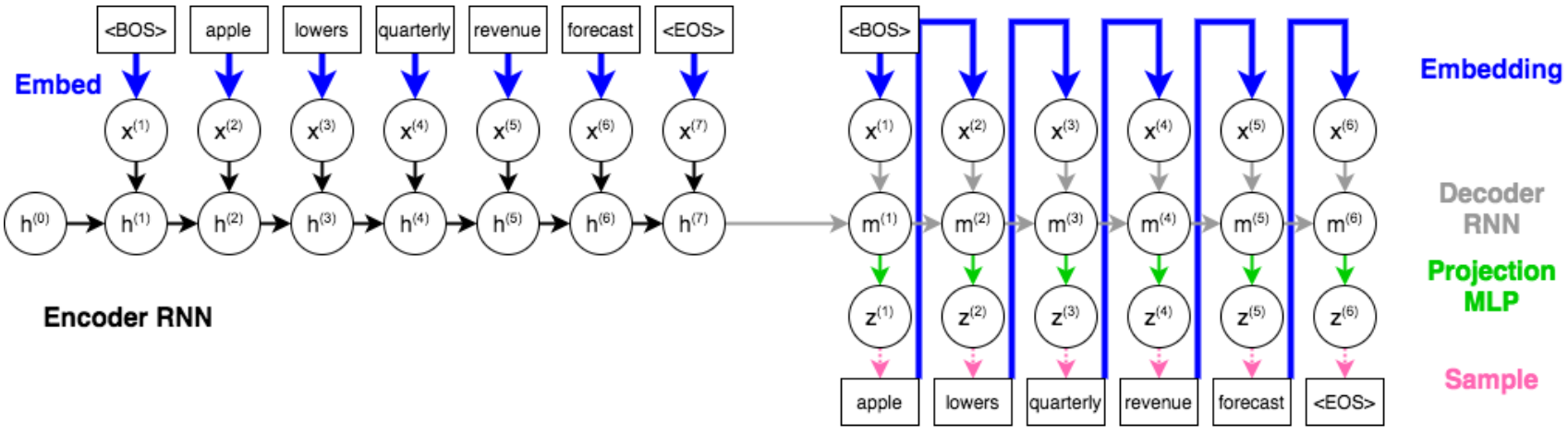
Another way to prevent overfitting artificially increase the amount of training data by *adding random noise* to the data.

Example: AlexNet used random crops of input images

Q: Given that the goal of data augmentation is to prevent overfitting, would you also augment your validation data?

Q: Given that the goal of data augmentation is to prevent overfitting, would you also augment your test data?

Project 4 Model



Project 4 Model: Teacher Forcing