Lecture 8
Language Modelling

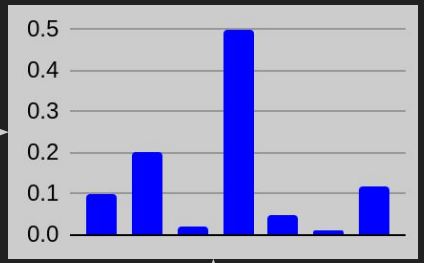
Recurrent Neural Networks

Training data

- Getting tons of data for supervised learning is costly.
- There is a lot of text on the internet, how can we use it?
- Ideal case: use data without any labeling.

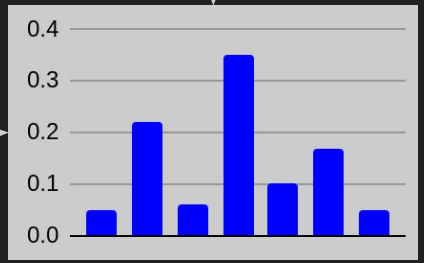
Using unlabeled data: consistency regularization

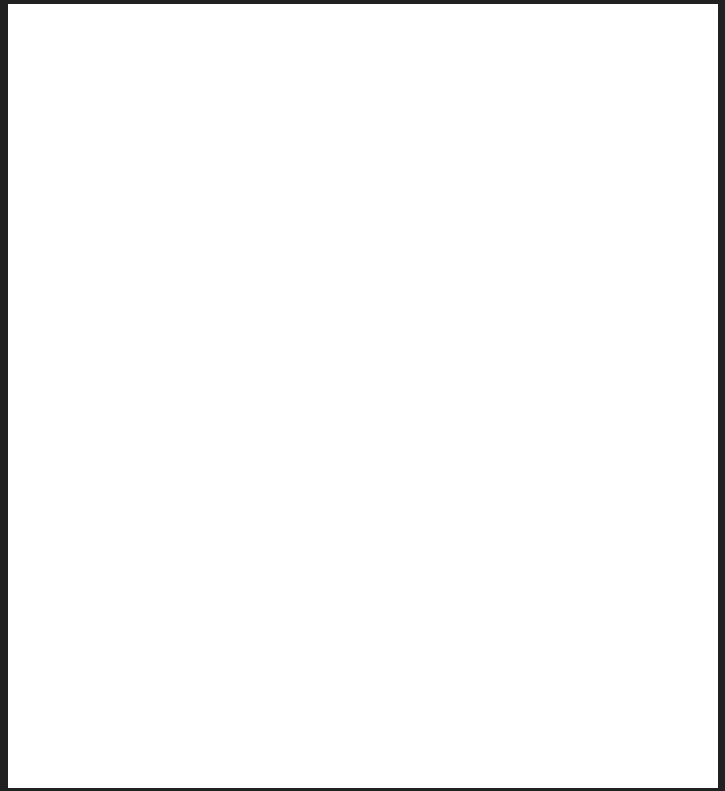
Convnet



Consistency loss

Convnet

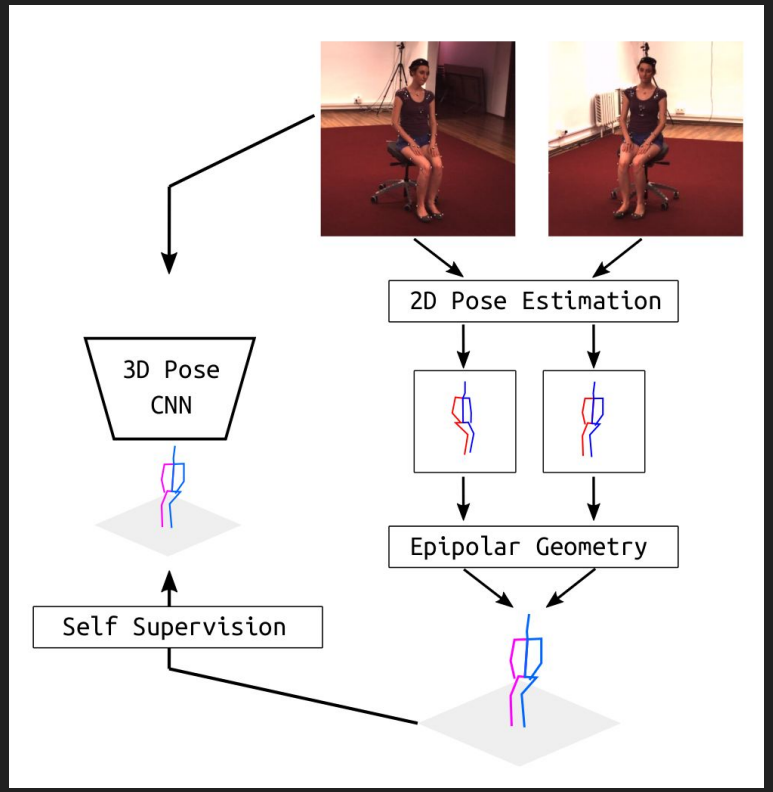




- Two augmentations of the same image should have similar representation, two augmentations of different images should have different representations
- Multiple data augmentation operations are crucial (most importantly crop+color jitter).

(see *SimCLR - A Simple Framework for Contrastive Learning of Visual Representation* - ICML 2020)

Self-supervised 3D pose estimation



The algorithm needs to predict both the 2D pose and a 3D pose from a single image.

Supervision comes from the alignment penalty:

- assuming we know the 3D relationship between cameras, we can reproject points between images.

source

Auxiliary self-supervised tasks

- Coloring images.
- Jigsaw.
- Guess rotation.

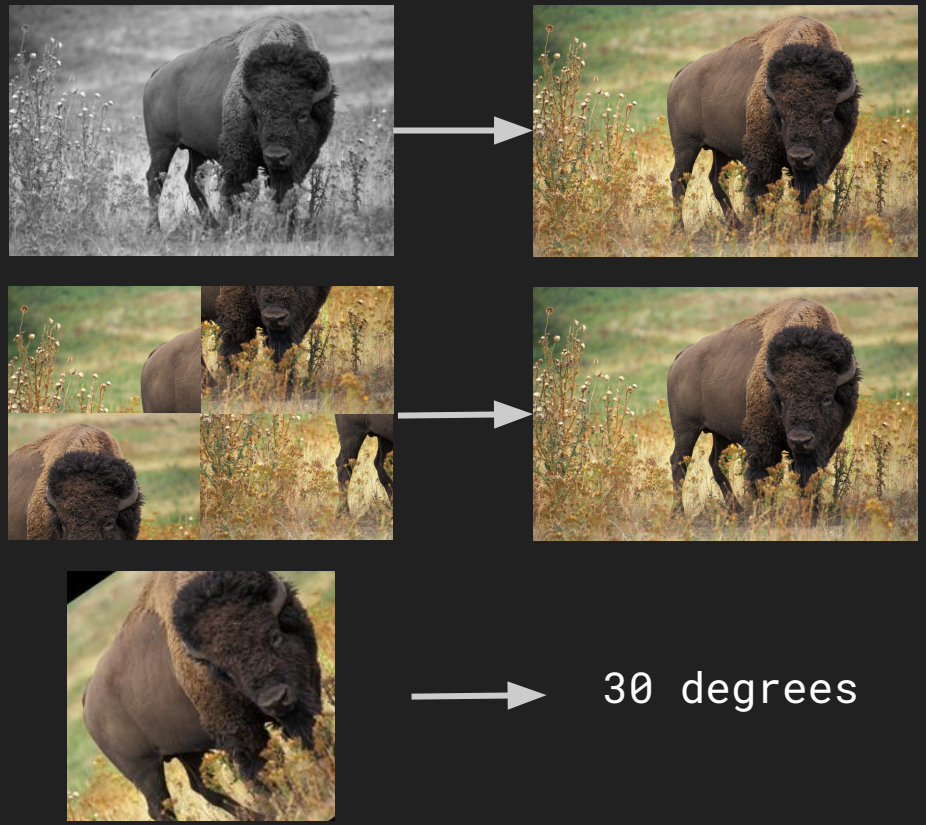


Image by WikiImages from Pixabay

Self-supervision: masking

- Pretend that we don’t know parts of text ends,
  ask the network to predict it.
- No need for labelling, ground truth for free!

Input:
The <BLANK> of Warsaw is the capital and <BLANK> city of <BLANK>

Expected output:
The city of Warsaw is the capital and largest city of Poland.

Self-supervision: autoregression

- Pretend that we don’t know the ending.
- Again, ground truth for free.

Input:
The city of Warsaw is the capital and largest city of

Expected output:
Poland

Autoregressive models - using full sequence

- *The -> city*
- *The city -> of*
- *The city of -> Warsaw*
- *The city of Warsaw -> is*
- *The city of Warsaw is -> the*
- *The city of Warsaw is the -> capital*
- *The city of Warsaw is the capital -> and*
- *The city of Warsaw is the capital and -> largest*
- *The city of Warsaw is the capital and largest -> city*
- *The city of Warsaw is the capital and largest city -> of*
- *The city of Warsaw is the capital and largest city of -> Poland*

Agenda

1. Next token prediction from short context.
2. Sequence to sequence problem.
    - truncated backprop through time.
3. RNN/LSTM.
4. Using a trained NLP model to generate text.

Slides credits

Some slides taken from:

- CS 182 - Berkeley course by Sergey Levine.
- EECS 498-007 / 598-005 - U of Michigan course by Justin Johnson.

Who in turn used (some) slides, or were inspired by:

- CS 224n - Stanford course by Chris Manning on NLP.
- Andrej Karpathy (blogposts, tokenizer, CS 231n).

Next token prediction

Generating the next token

- A model generates conditional probability distribution over the next token, e.g.,
  P(*“Poland”* | *“The city of Warsaw is the capital of”*).

- Loss function: cross-entropy

Note that for tokens $w_1, w_2, \ldots, w_n$

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

Define perplexity as $\frac{\sum_{i=1}^{n} \log(P(w_i | w_1, \ldots, w_{i-1}))}{n}$

Note that for tokens $w_1, w_2, \ldots, w_n$

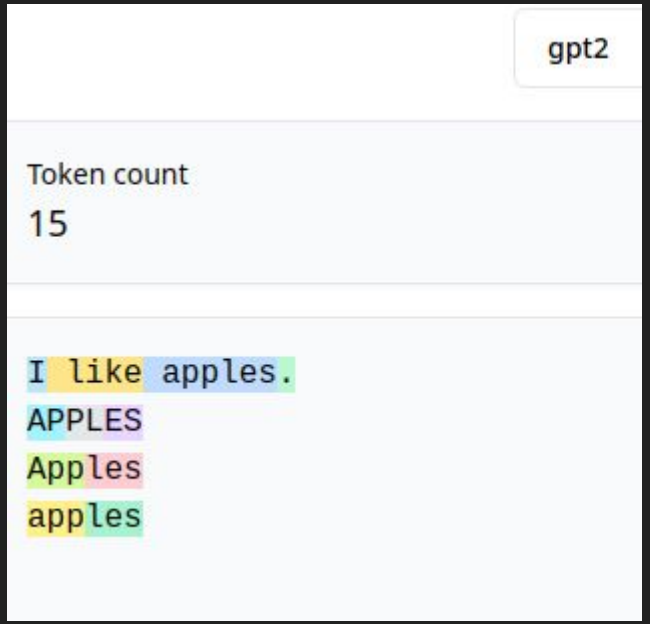
$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

Define perplexity as $\frac{\sum_{i=1}^{n} \log(P(w_i | w_1, \ldots, w_{i-1}))}{n}$

- P(*The* | <*START*>)
- P(*city* | *The*)
- P(*of* | *The city*)
- P(*Warsaw* | *The city of*)
- P(*is* | *The city of Warsaw*)
- P(the | *The city of Warsaw is*)
- P(capital | *The city of Warsaw is the*)

Tokenization

- Extremes:
  - Characters as tokens.
  - Words as tokens.
- Typically something in between
  - tens of thousands of tokens.
- https://tiktokenizer.vercel.app/



Tokenizatization - source of many problems

Due to tokenization LLMs have problems:

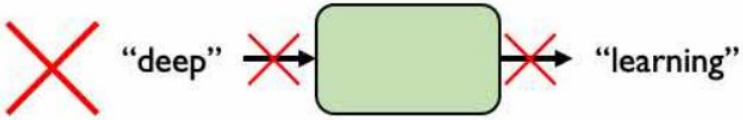
- spelling words,
- doing arithmetic,
- GPT-2 was not good in Python

N-gram models

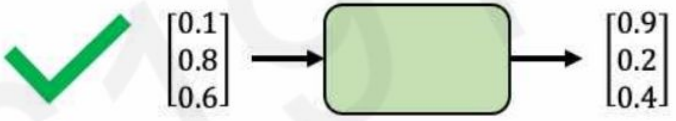
- Based on text corpora compute
  - count[w_1, w_2, …, w_n]
  - P[w_3 | w_1, w_2] = count[w_1, w_2, w_3] / count[w_1, w_2]

- Problems:
  - Small n - not enough context
  - Large n - memory explodes, sparsity issues

Encoding Language for a Neural Network

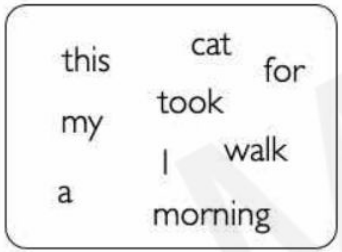


Neural networks cannot interpret words

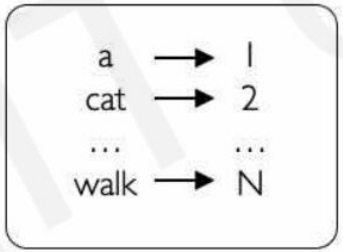


Neural networks require numerical inputs

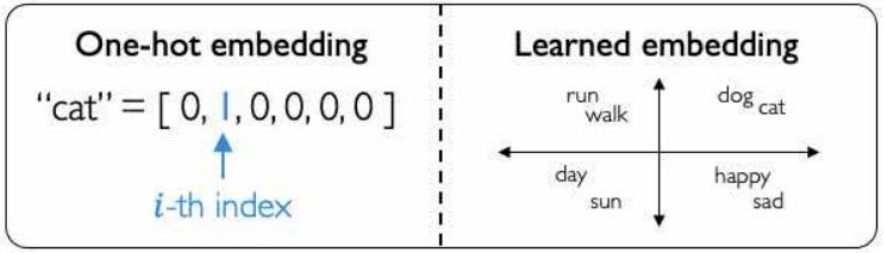
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



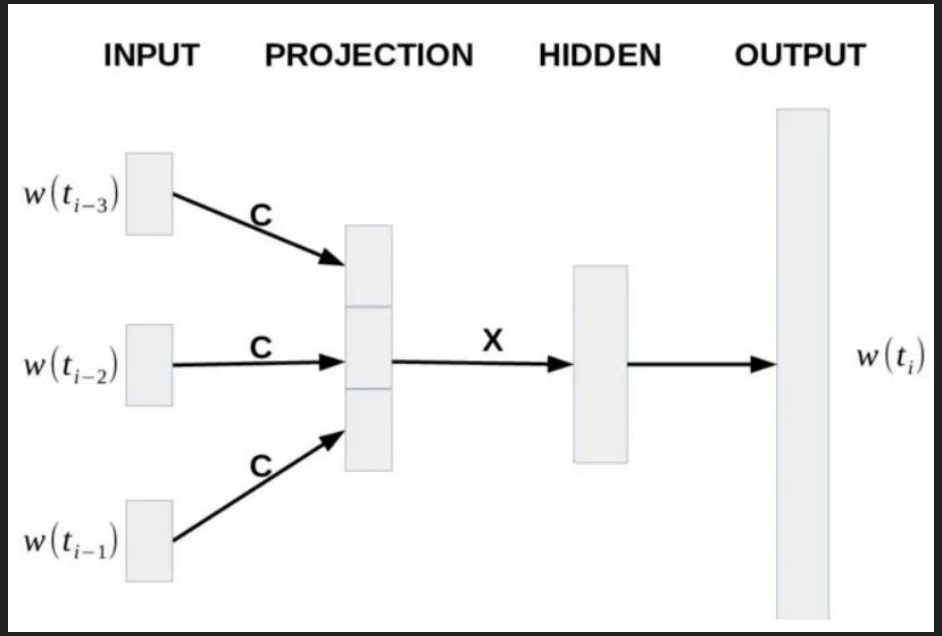
2. Indexing:
Word to index



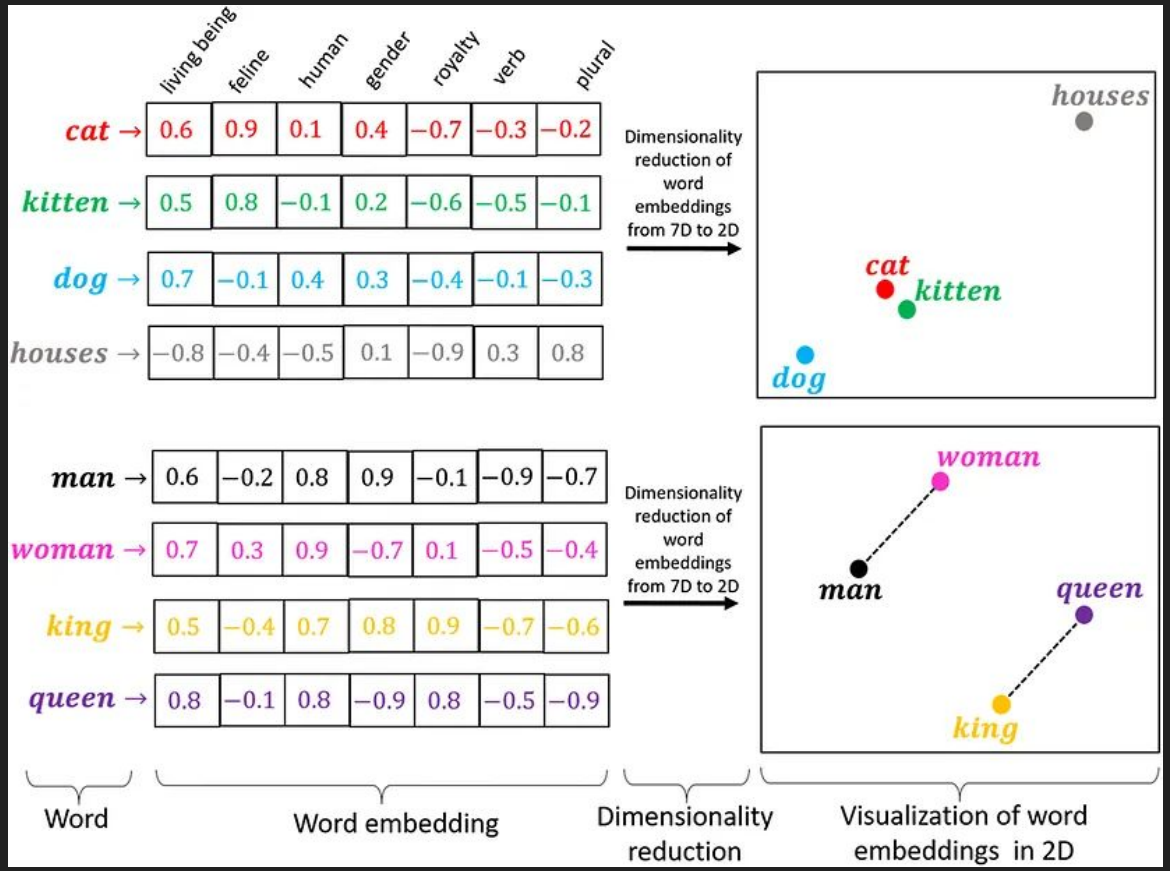
3. Embedding:
Index to fixed-sized vector

2003 - MLP for next token prediction

- Word Embeddings
- Feedforward:
  - Input: concatenate n previous word vectors
- Softmax Output over all possible tokens.



Learned word embeddings

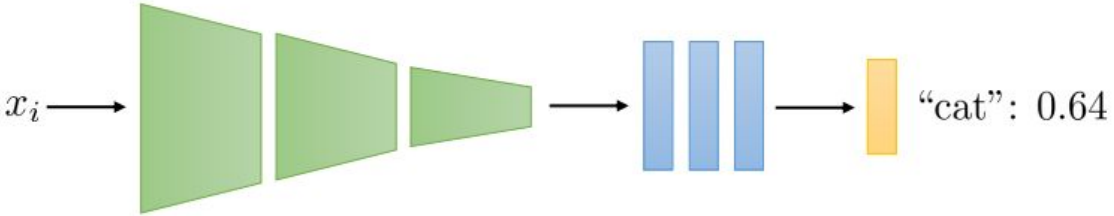


source

Dealing with sequences

What if we have variable-size inputs?

Before:



Now:

$x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4})$

$x_2 = (x_{2,1}, x_{2,2}, x_{2,3})$

$x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5})$

Examples:

classifying sentiment for a phrase (sequence of words)

recognizing phoneme from sound (sequence of sounds)

classifying the activity in a video (sequence of images)

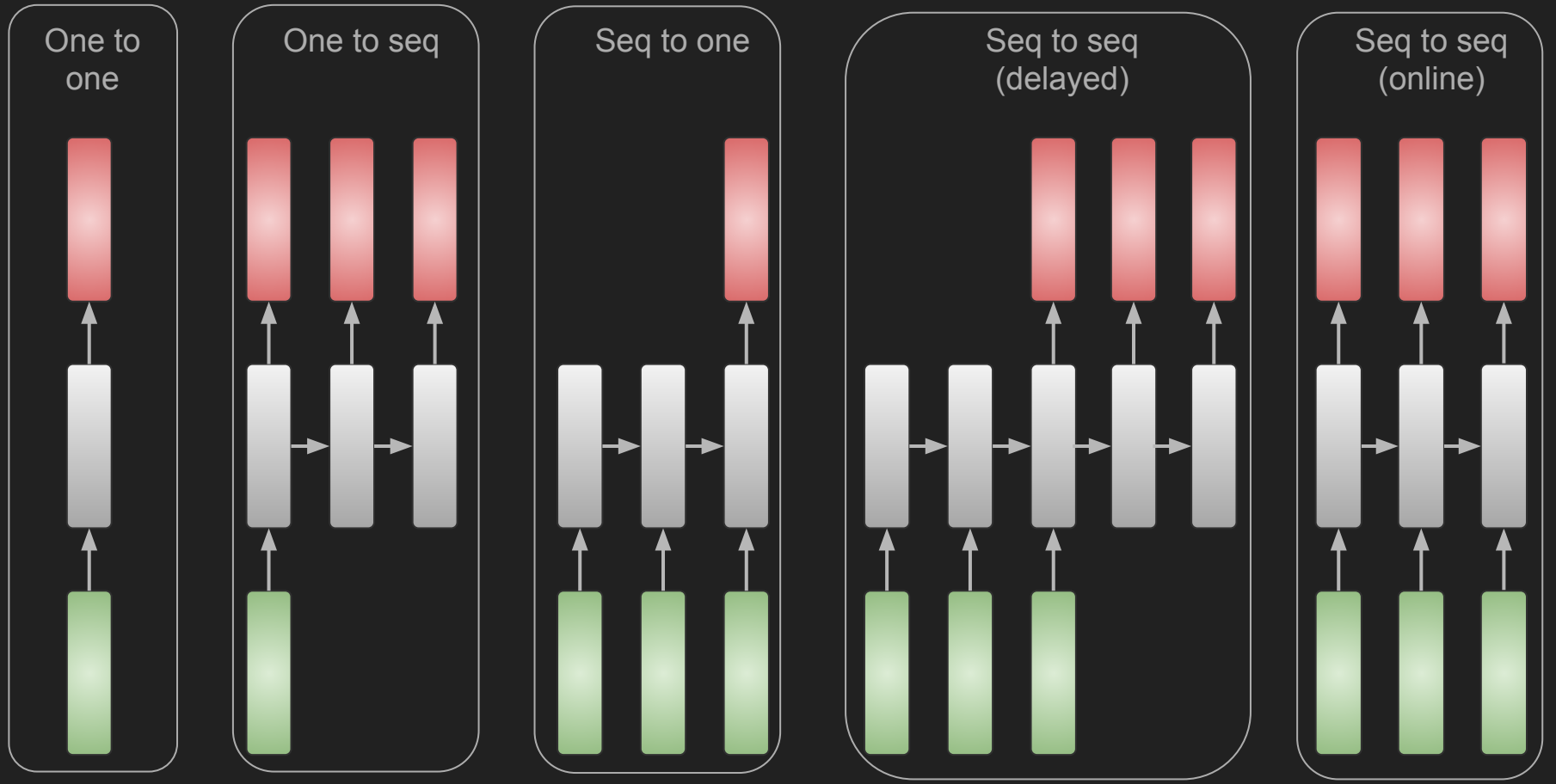
What if we have variable-size outputs?

Examples:

generating a text caption for an image

predicting a sequence of future video frames

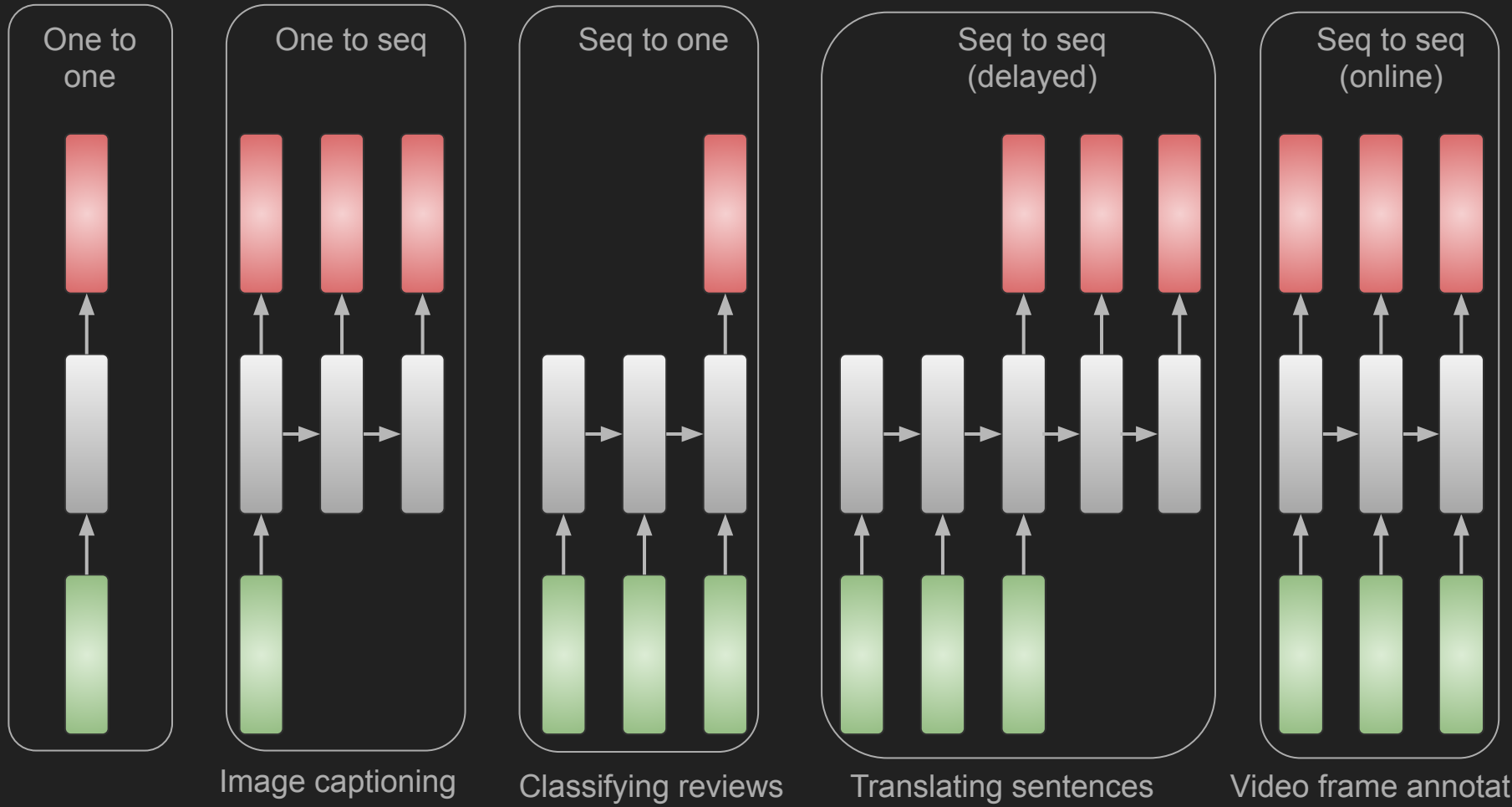
generating an audio sequence


One to one
One to seq
Seq to one
Seq to seq (delayed)
Seq to seq (online)

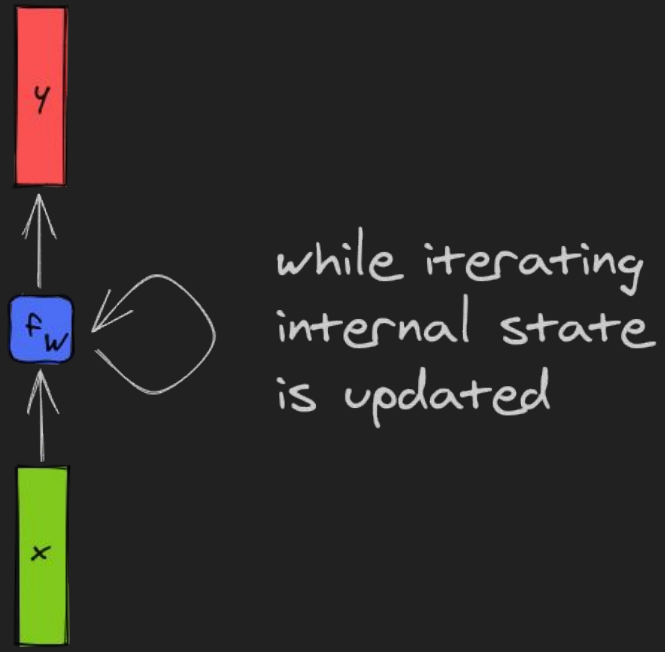
slido

Order the problems so that they match the type depicted on the diagram.A. Image Captioning.B. Video frame annotation.C. Translating a sentence into another language.D. Predict if a written review is positive or not.

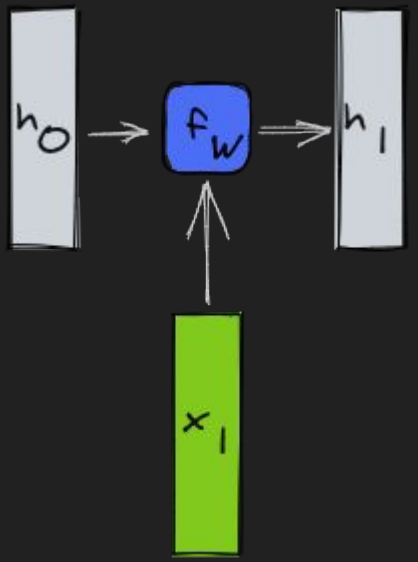
ⓘ Start presenting to display the poll results on this slide.


One to one
One to seq
Seq to one
Seq to seq (delayed)
Seq to seq (online)
Image captioning
Classifying reviews
Translating sentences
Video frame annotation

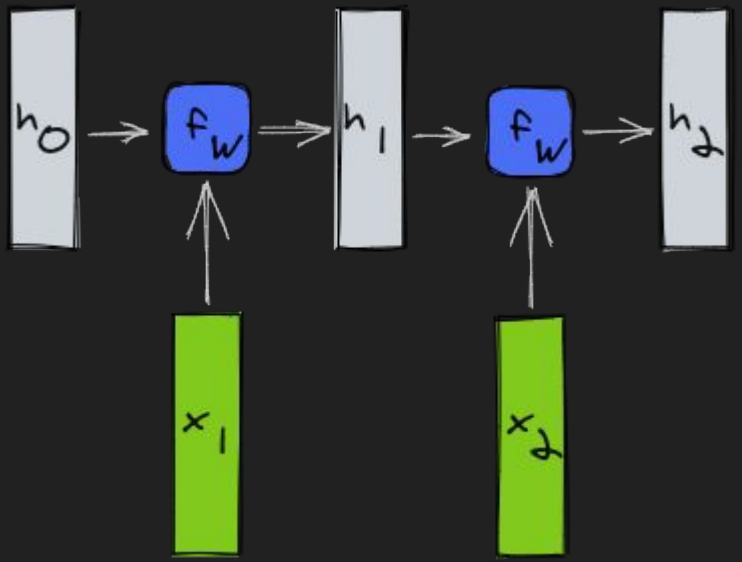
RNN



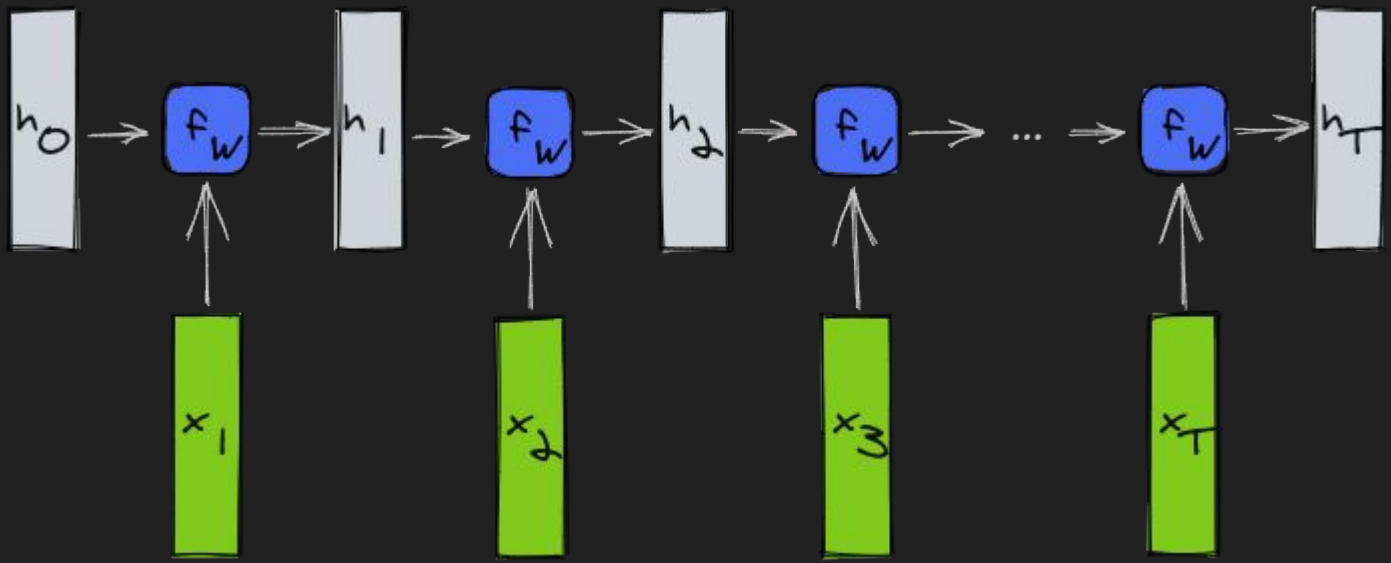
RNN graph



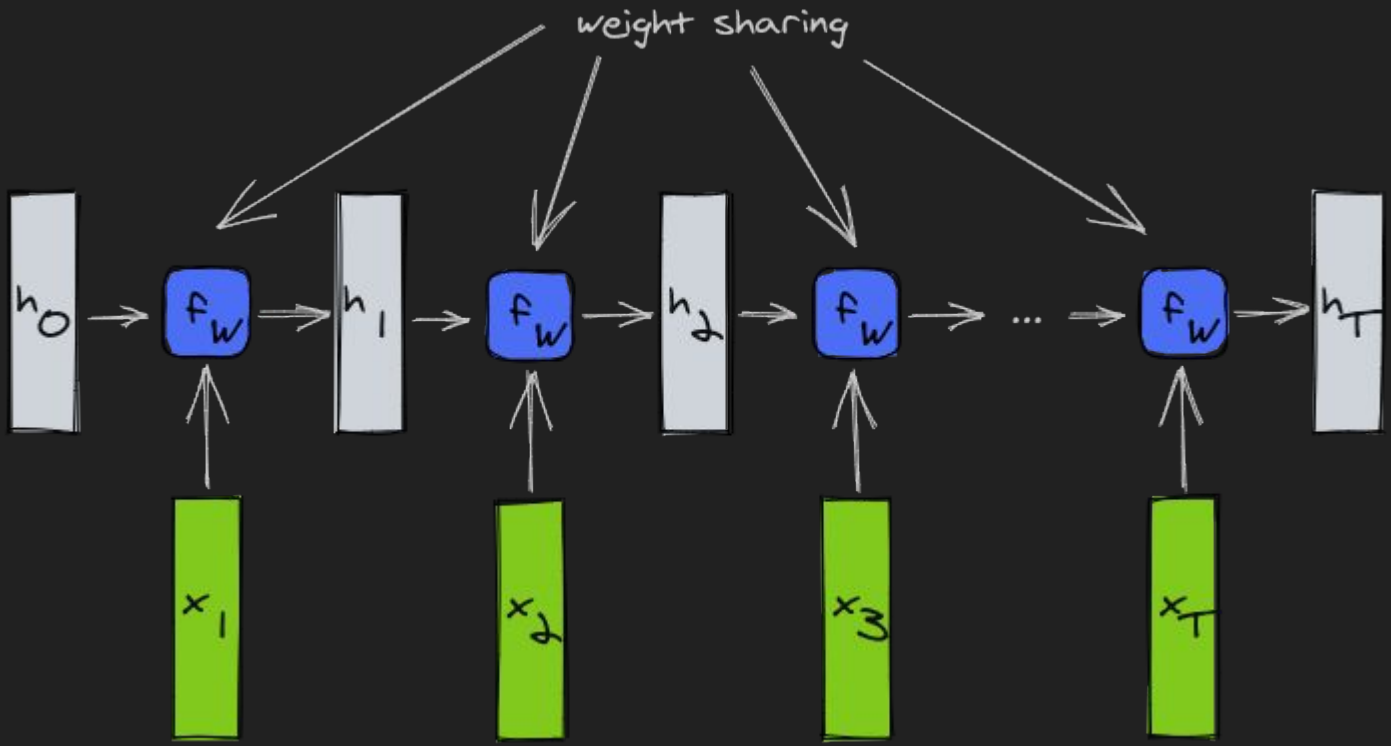
RNN graph



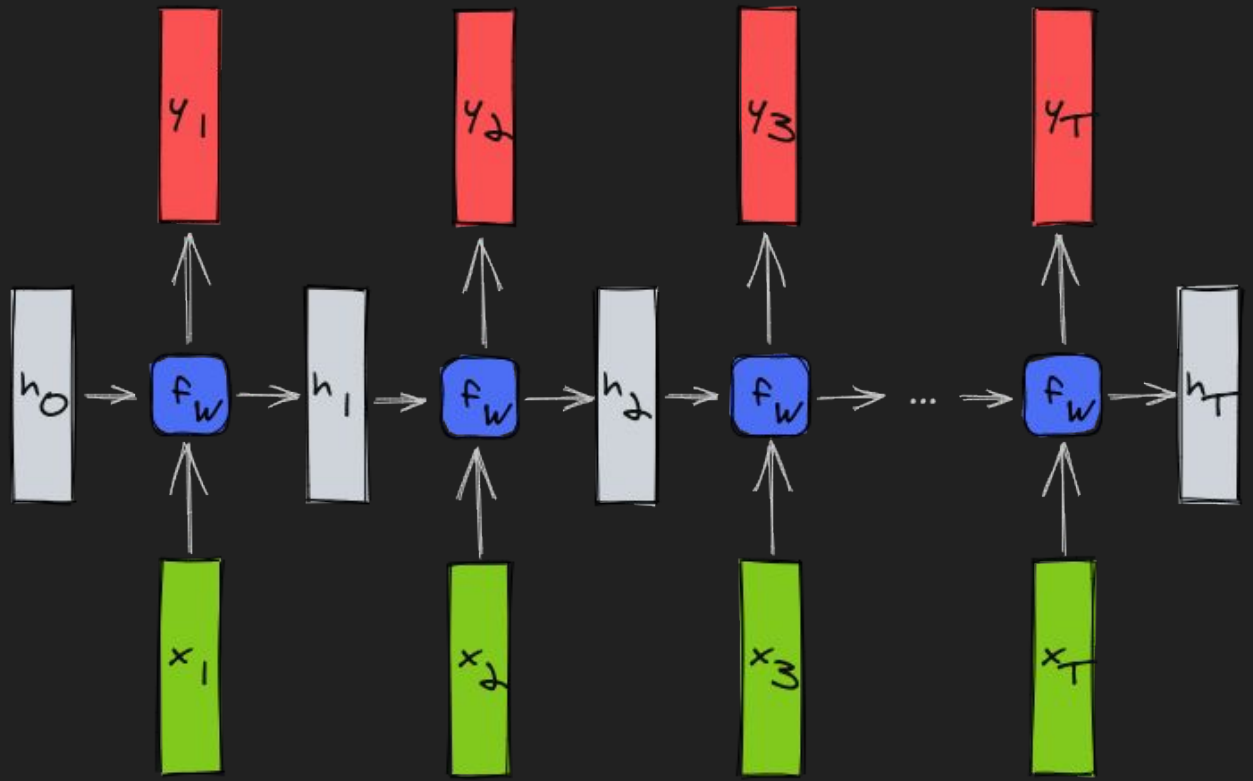
RNN graph

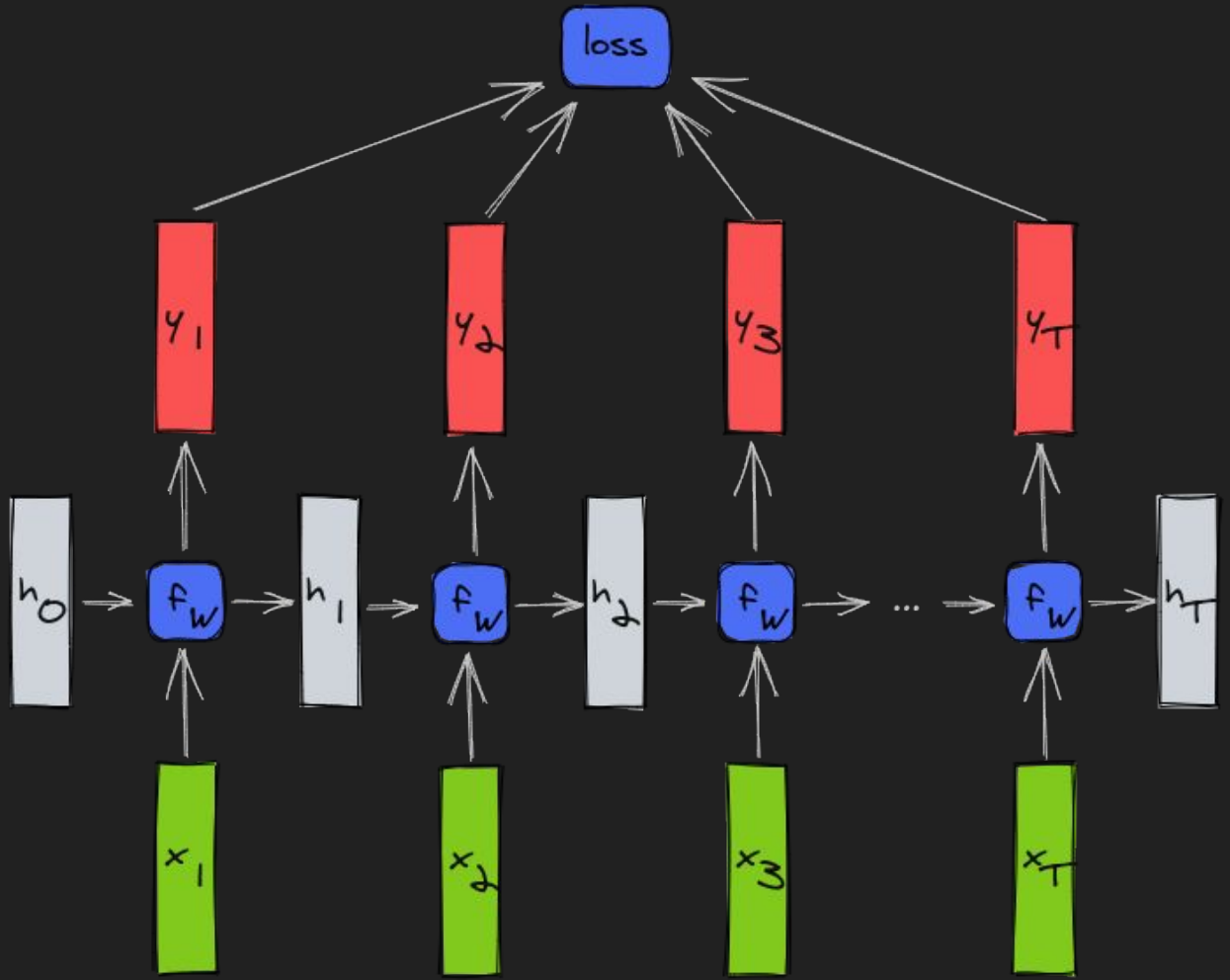


RNN graph

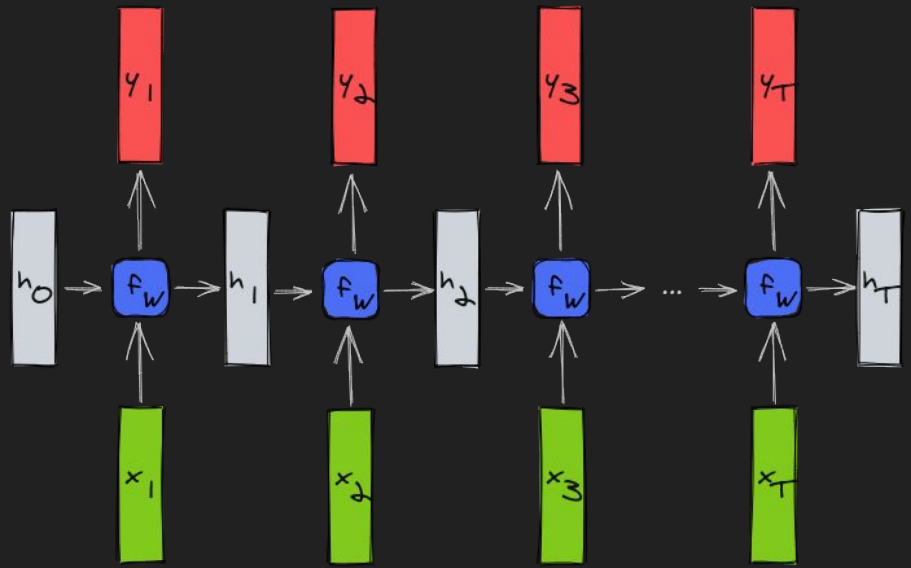


RNN graph - many outputs




loss
y_1
y_2
y_3
y_T
h_0
f_W
h_1
f_W
h_2
f_W
...
f_W
h_T
x_1
x_2
x_3
x_T

RNN graph



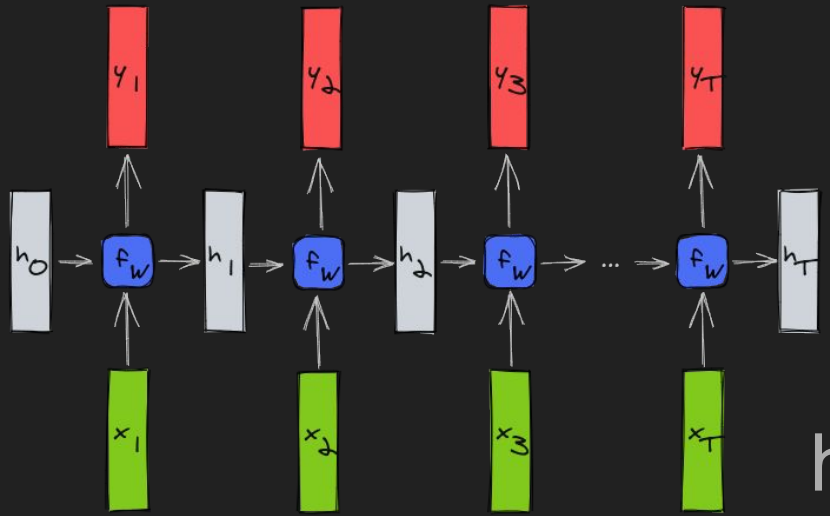
new state old state

$$h_i = f_W(h_{i-1}, x_i)$$

function with shared parameters W

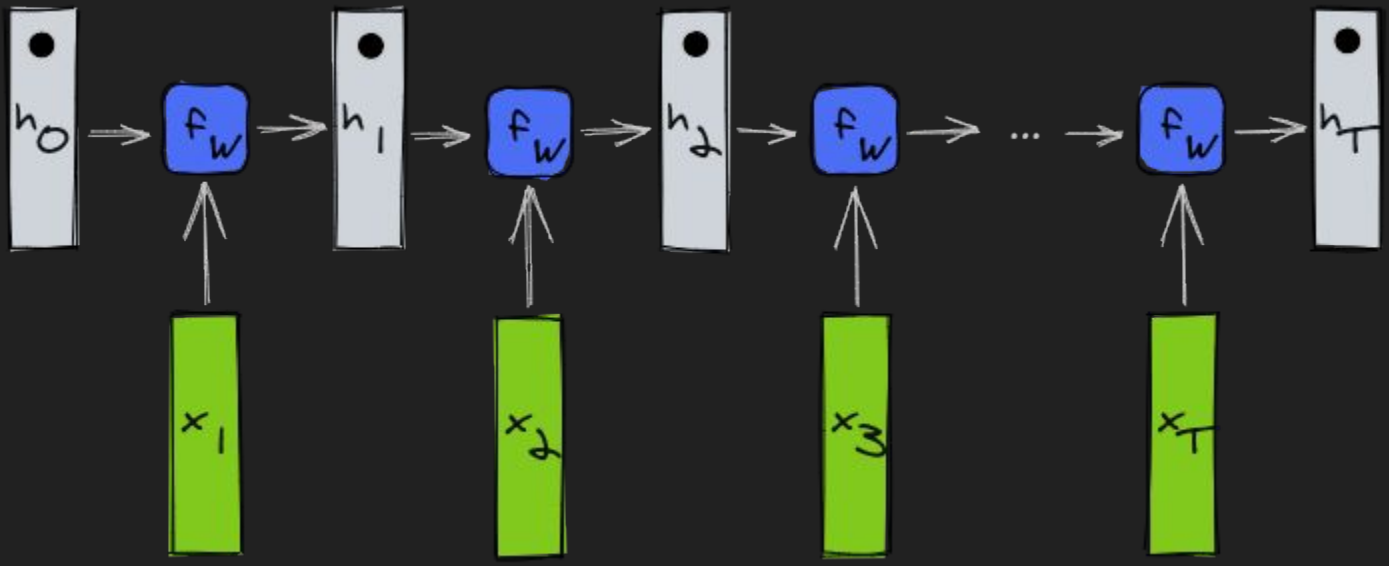
current element

“Vanilla RNN” or “Elman RNN”

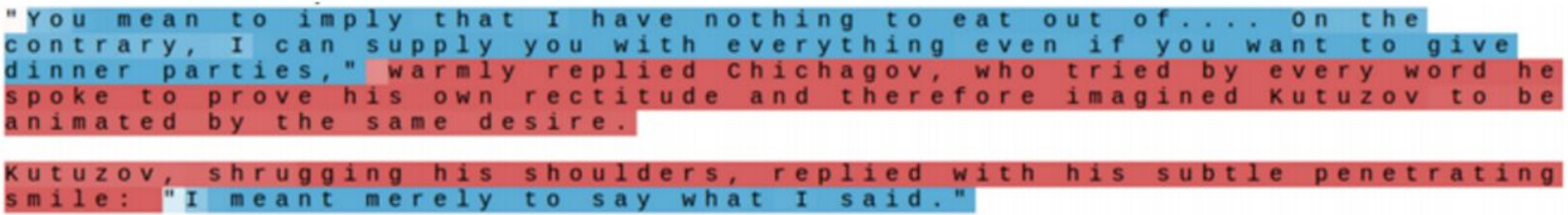


$$h_i = \tanh(W_{hh} h_{i-1} + W_{hx} x_i + b_h)$$
$$y_i = W_{yh} h_i + b_y$$

Interpreting neurons in the internal state



Searching for Interpretable Hidden Units



quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

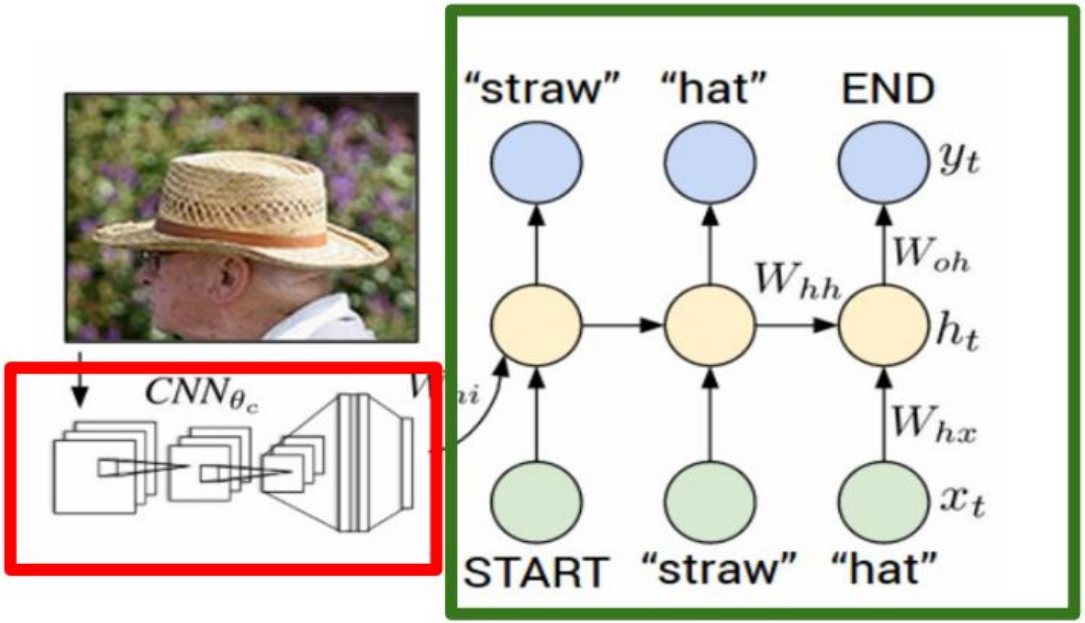
Searching for Interpretable Hidden Units

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Example: Image Captioning

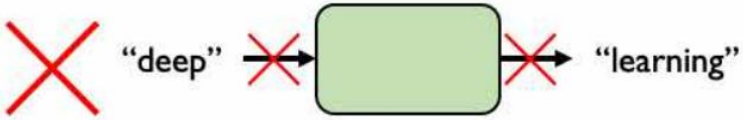


Recurrent Neural Network

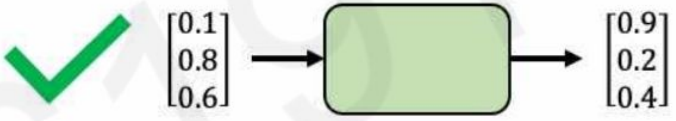
Convolutional Neural Network

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Encoding Language for a Neural Network

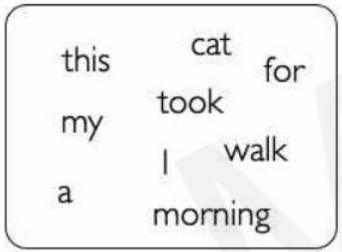


Neural networks cannot interpret words

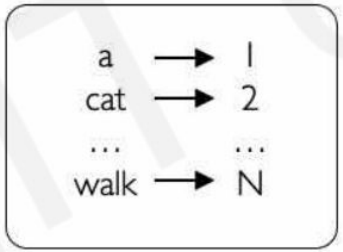


Neural networks require numerical inputs

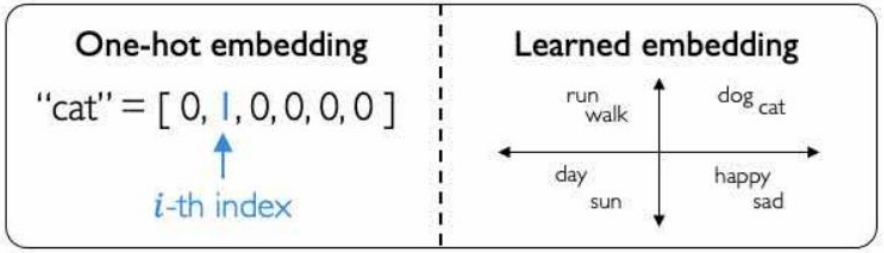
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



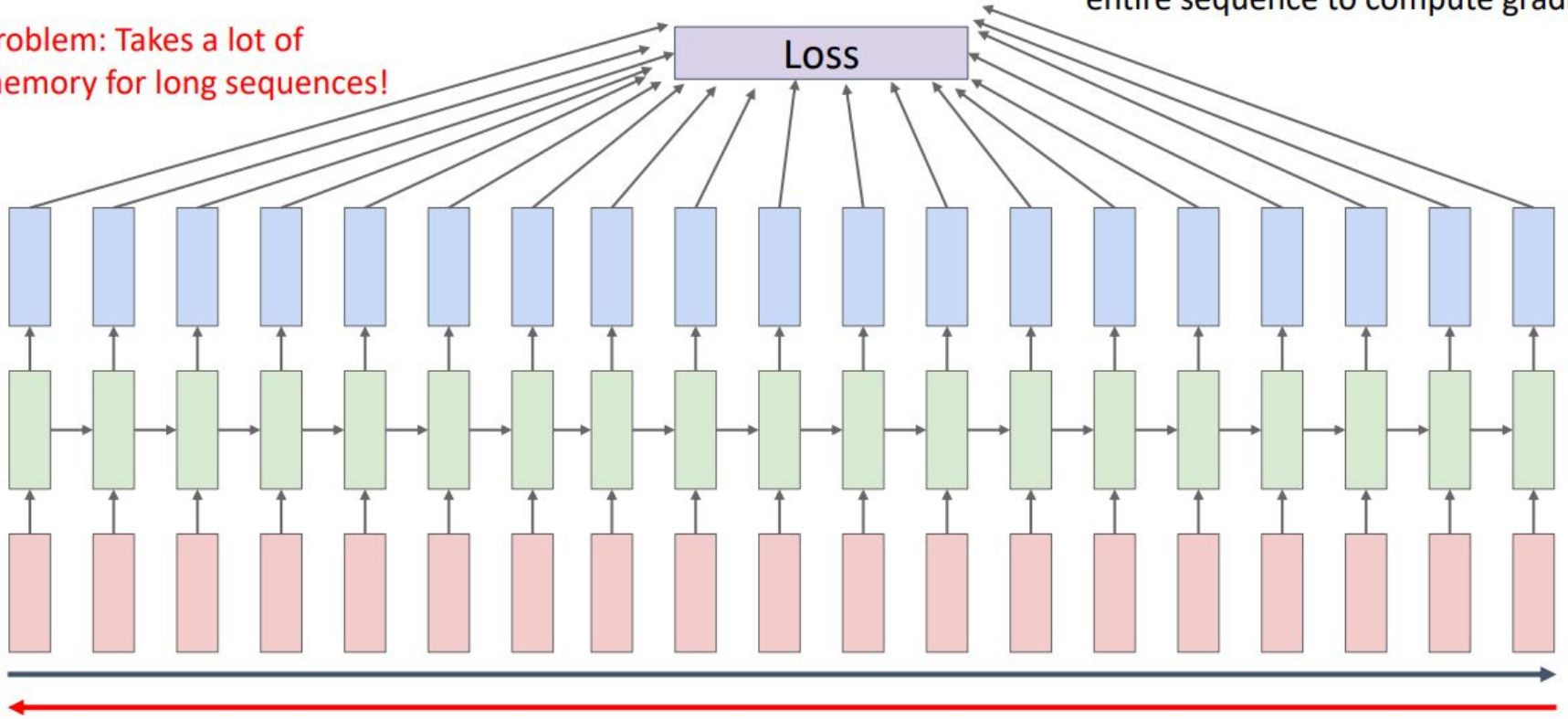
3. Embedding:
Index to fixed-sized vector

Training RNNs

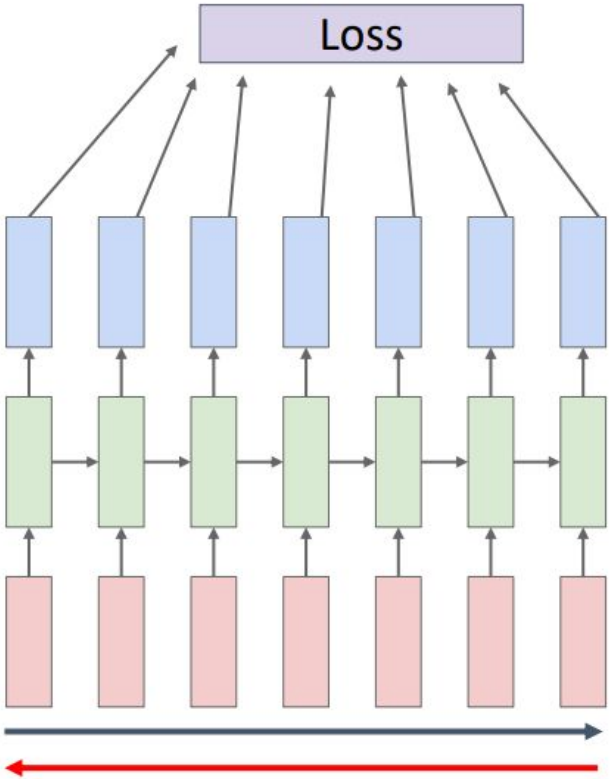
Backpropagation Through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

Problem: Takes a lot of memory for long sequences!

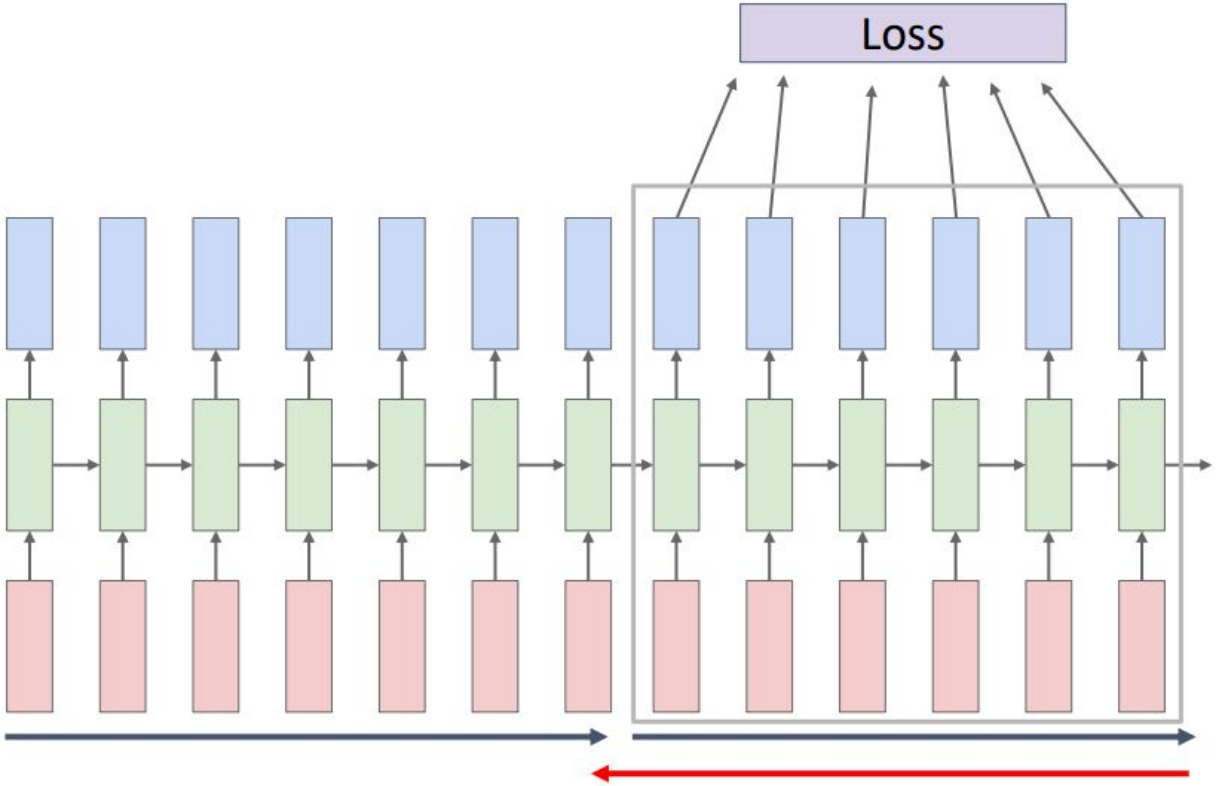


Truncated Backpropagation Through Time



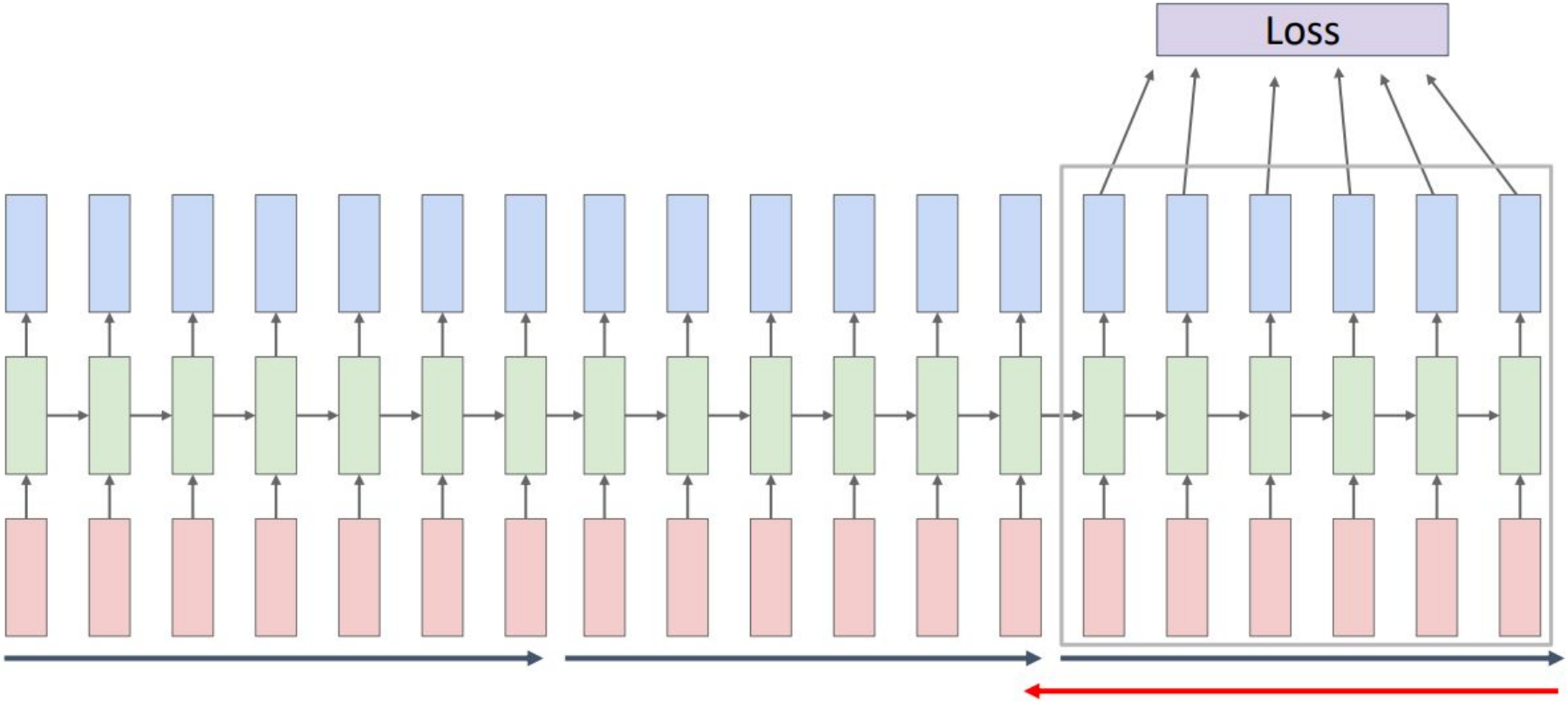
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation Through Time



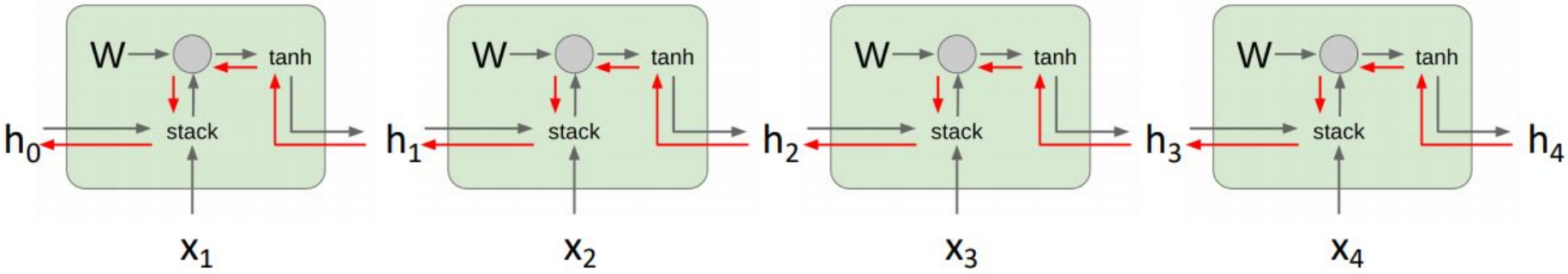
Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation Through Time



Gradient problems

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

Two vectors at each timestep:

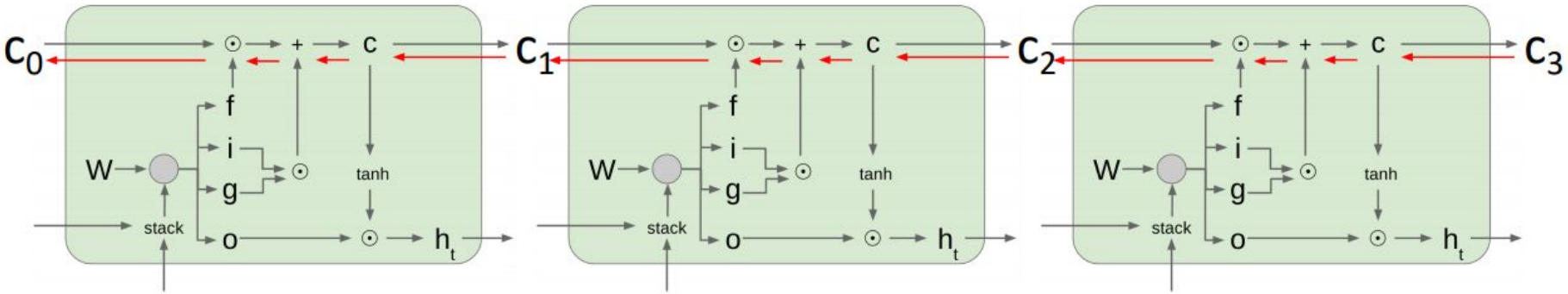
Cell state: $c_t \in \mathbb{R}^H$

Hidden state: $h_t \in \mathbb{R}^H$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Language models

Context-depending representation

"She couldn't **bear** the cold any longer."

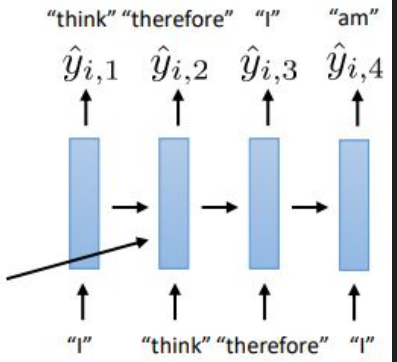
"A **bear** wandered into their campsite at dawn."

Can we learn representations that **depend on context?**

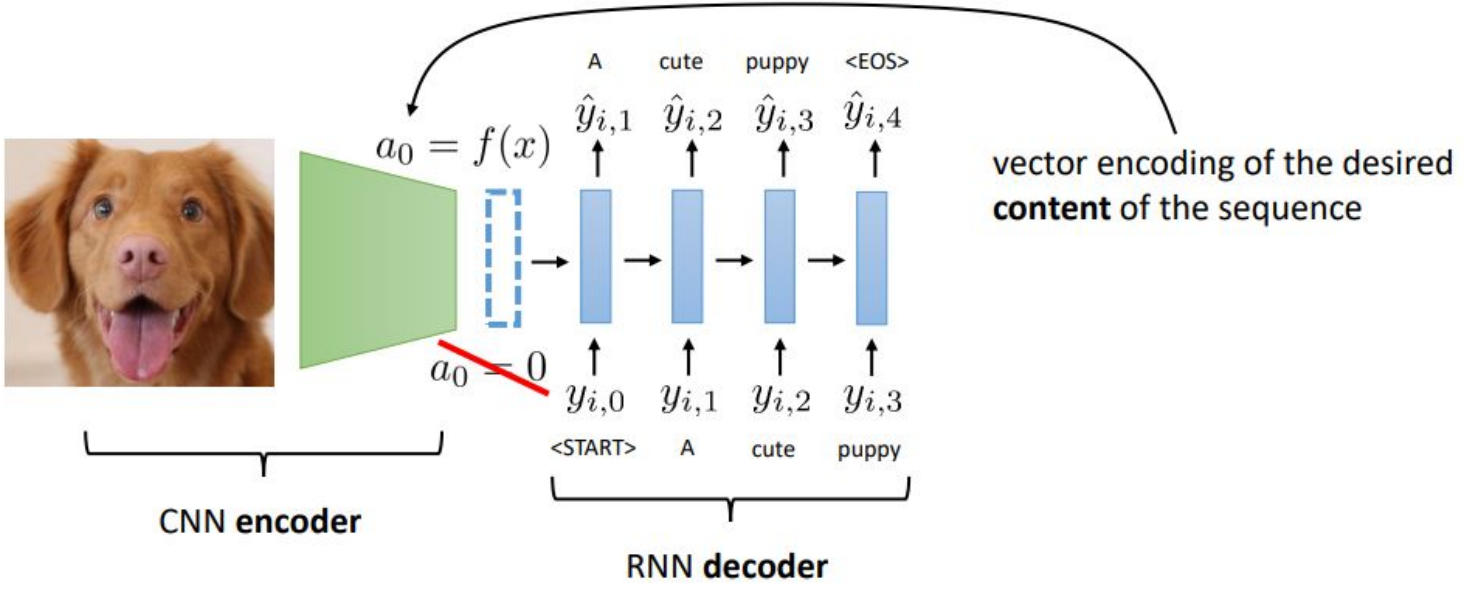
High level idea:

1. Train a **language model**
2. Run it on a sentence
3. Use its **hidden state**

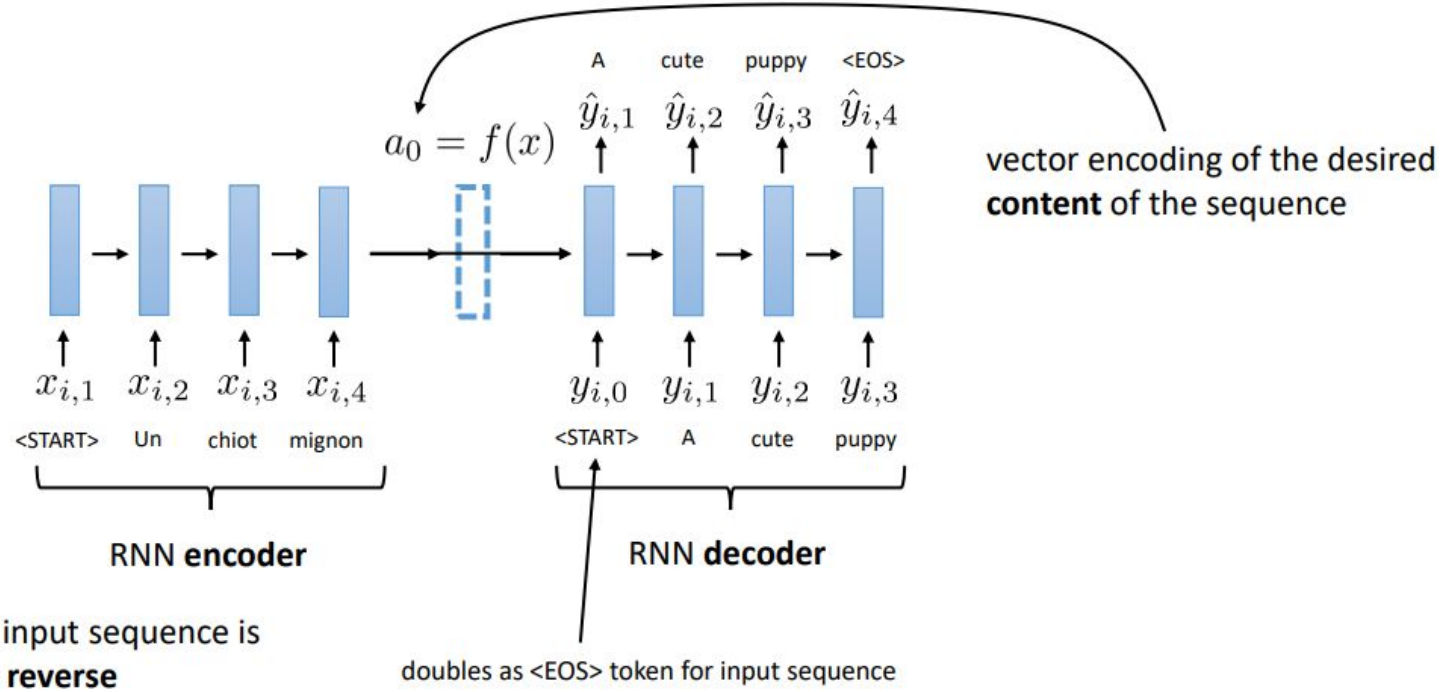
use the hidden state as the representation for downstream tasks



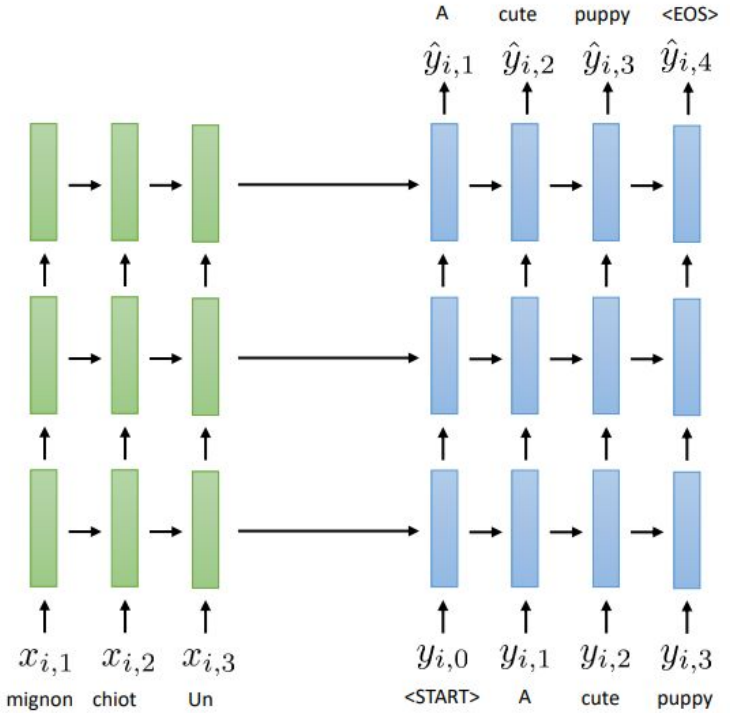
A conditional language model



What if we condition on *another* sequence?



A more realistic example



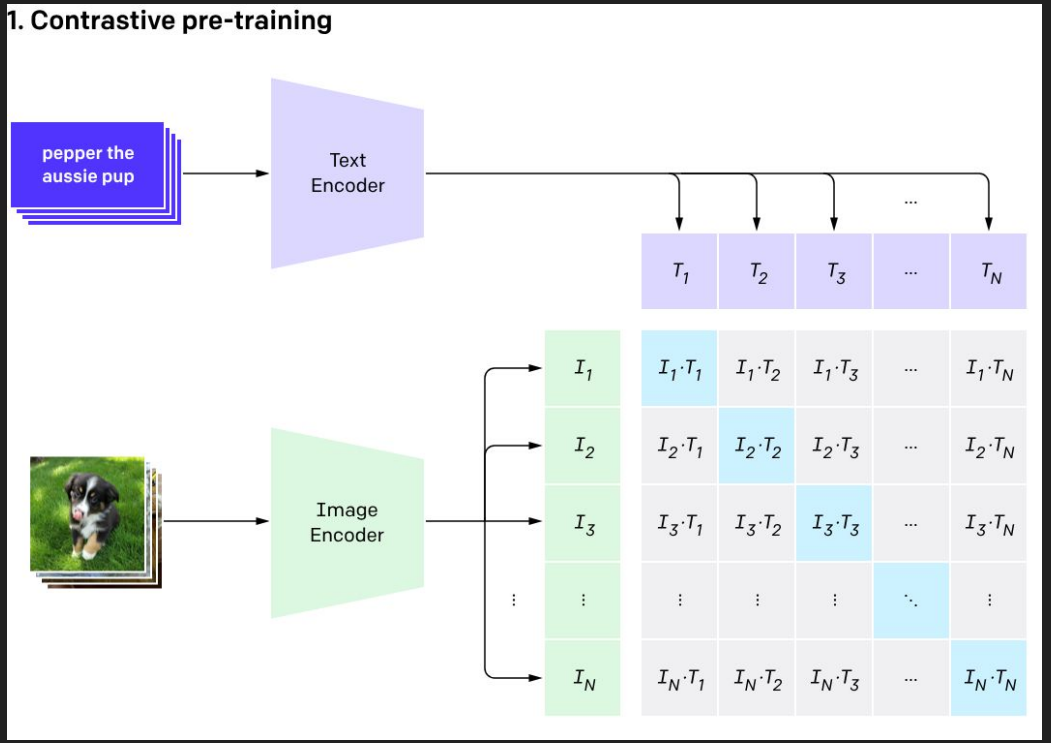
- Multiple RNN layers
- Each RNN layer uses LSTM cells (or GRU)
- Trained end-to-end on pairs of sequences
- Sequences can be different lengths

- Translate **one language** into **another language**
- Summarize a **long sentence** into a **short sentence**
- Respond to a **question** with an **answer**
- Code generation? **text** to **Python code**

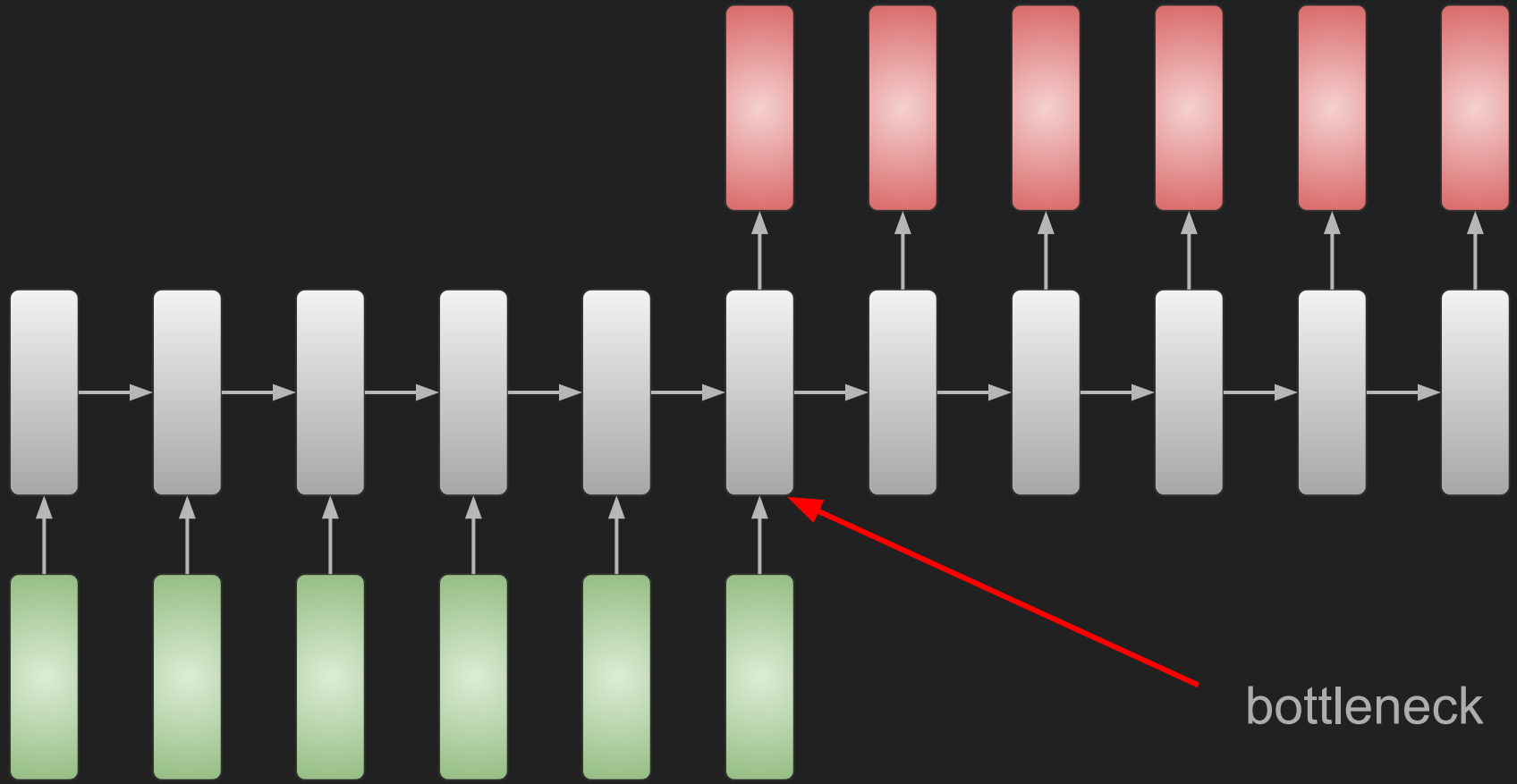
For more, see: Ilya Sutskever, Oriol Vinyals, Quoc V. Le. **Sequence to Sequence Learning with Neural Networks**. 2014.

CLIP

- Dataset: pairs of corresponding (text, image)
- Contrastive pretraining:
  - matching pairs should have close vectors,
  - other pairs should have distant vectors.



Bottleneck and attention


bottleneck

Bottleneck in RNNs

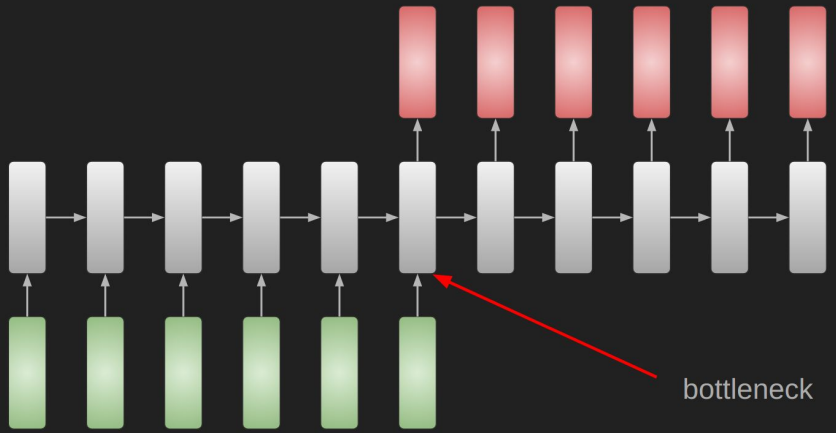
Consider the following setting:

- The network reads a paragraph of an article, then reads a question and has to produce an answer.

- For example, there is a description:
  - Warsaw is the capital of Poland,
  - Berlin is the capital of Germany,
  - Paris is the capital of France, etc.
  - Question: what is the capital of <country>?

Attention

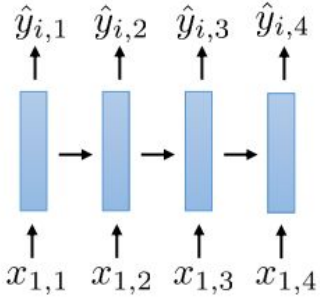
- Attention - circumvents the problem.
- The model is allowed to look back far (thousands of tokens) anytime.
- More on this in the next two lectures.



How to generate good output sequences?

(assuming having a trained model)

Autoregressive models and structured prediction



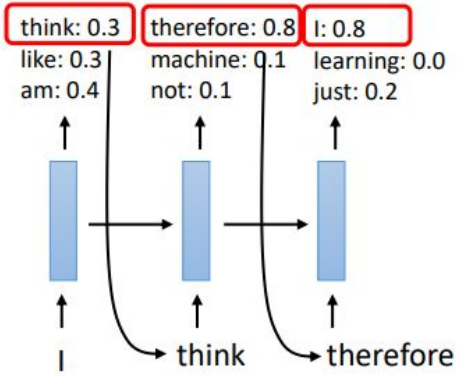
most RNNs used in practice look like this

why?

most problems that require multiple outputs have strong **dependencies** between these outputs

this is sometimes referred to as **structured** prediction

Example: text generation



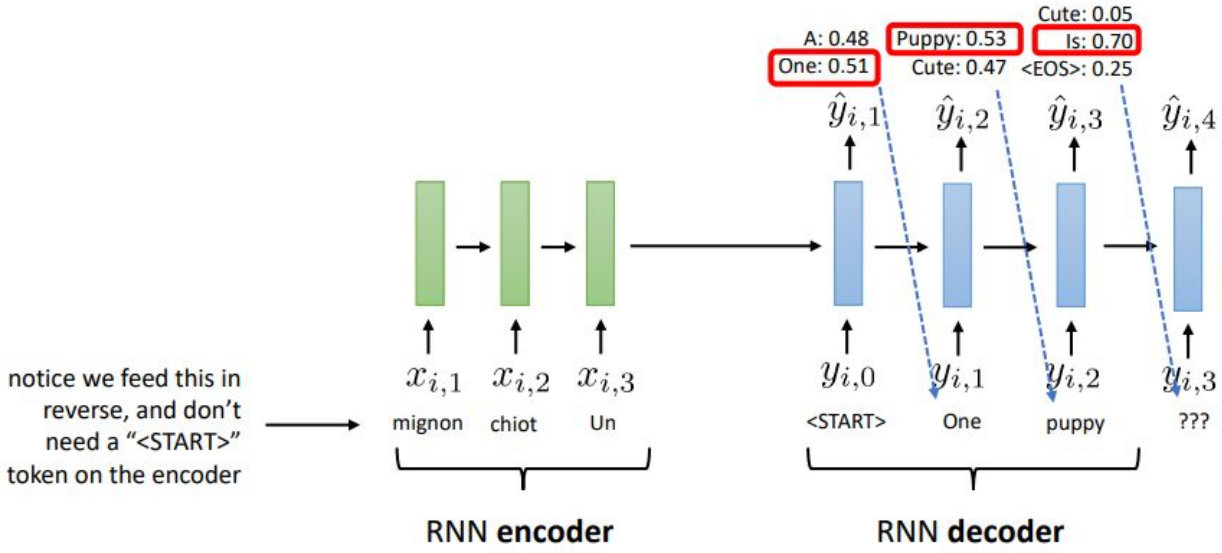
I think therefore I am

I like machine learning

I am not just a neural network

Key idea: past outputs should influence future outputs!

Decoding the most likely sequence



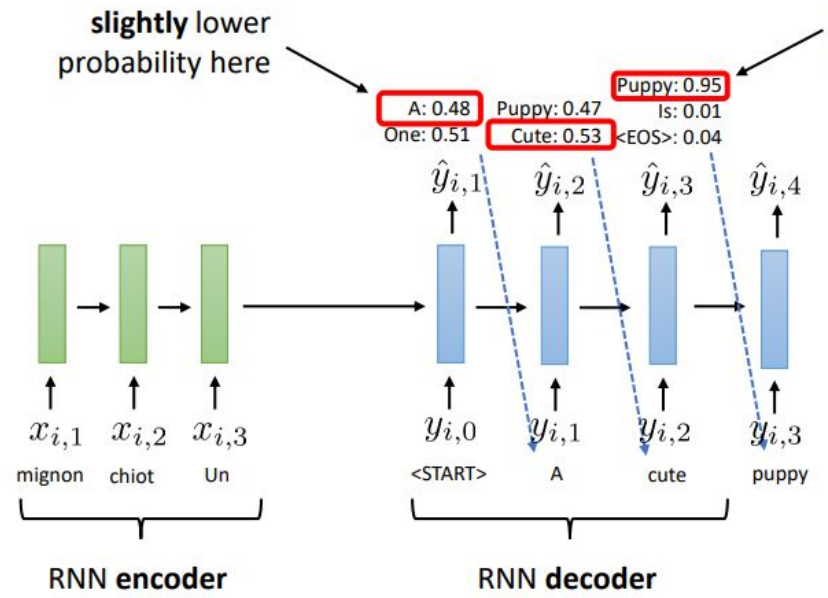
What we *should* have done

what does each output represent?

$$p(y_{i,t}|x_{i,1:T}, y_{i,0:t-1})$$

depends on whole input sequence

and output sequence so far!



If we want to maximize the product of **all** probabilities, we should not just greedily select the highest probability on the first step!

notice we feed this in reverse, and don't need a "<START>" token on the encoder

$$p(y_{i,1:T_y}|x_{i,1:T}) = \prod_{t=1}^{T_y} p(y_{i,t}|x_{i,1:T}, y_{i,0:t-1})$$

probabilities at each time step

How many possible decodings are there?

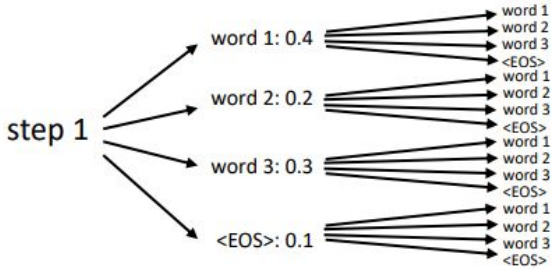
Word 1: 0.4	Word 1: 0.1	Word 1: 0.7	Word 1: 0.1	Word 1: 0.0
Word 2: 0.2	Word 2: 0.5	Word 2: 0.1	Word 2: 0.8	Word 2: 0.1
Word 3: 0.3	Word 3: 0.2	Word 3: 0.1	Word 3: 0.0	Word 3: 0.0
<EOS>: 0.1	<EOS>: 0.2	<EOS>: 0.1	<EOS>: 0.1	<EOS>: 0.9

.....

for M words, in general there are M^T sequences of length T

any one of these might be the optimal one!

Decoding is a **search** problem



We could use *any* tree search algorithm

But exact search in this case is **very** expensive

Fortunately, the **structure** of this problem makes some simple **approximate search** methods work **very well**

Decoding with approximate search

Basic intuition: while choosing the **highest-probability** word on the first step may not be optimal, choosing a **very low-probability** word is very unlikely to lead to a good result

Equivalently: we can't be greedy, but we can be *somewhat* greedy

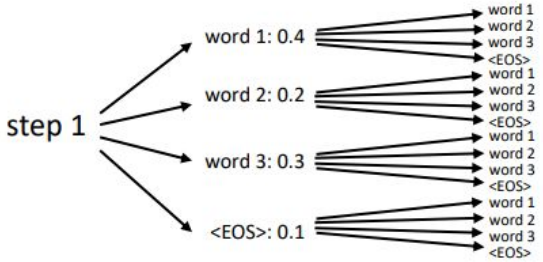
This is not true in general! This is a guess based on what we know about sequence decoding.

Beam search intuition: store the **k** best sequences **so far**, and update each of them.

special case of **k** = 1 is just greedy decoding

often use **k** around 5-10

Decoding is a **search** problem



Beam search example

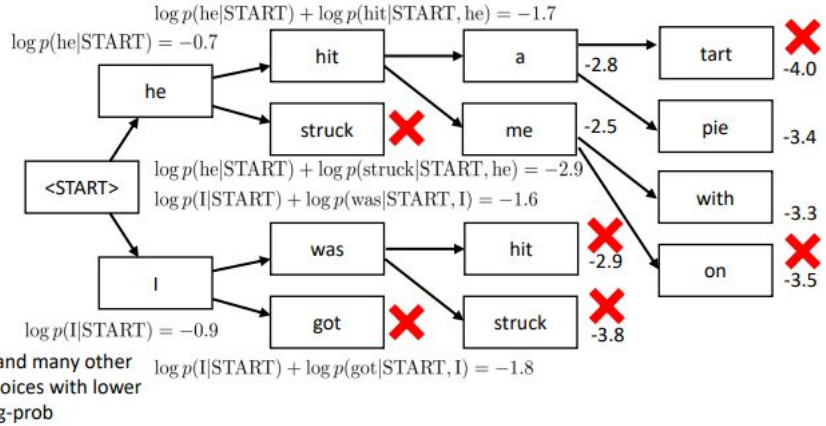
$$p(y_{i,1:T_y}|x_{i,1:T}) = \prod_{t=1}^{T_y} p(y_{i,t}|x_{i,1:T}, y_{i,0:t-1})$$

$$\log p(y_{i,1:T_y}|x_{i,1:T}) = \sum_{t=1}^{T_y} \log p(y_{i,t}|x_{i,1:T}, y_{i,0:t-1})$$

in practice, we **sum up** the log probabilities as we go (to avoid underflow)

Example (CS224n, Christopher Manning): **translate (Fr->En):** il a m'entarté (he hit me with a pie)

k = 2 (track the 2 most likely hypotheses) no perfectly equivalent English word, makes this hard



Beam search summary

$$\log p(y_{i,1:T_y}|x_{i,1:T}) = \sum_{t=1}^{T_y} \log p(y_{i,t}|x_{i,1:T}, y_{i,0:t-1})$$

there are k of these

at each time step t:

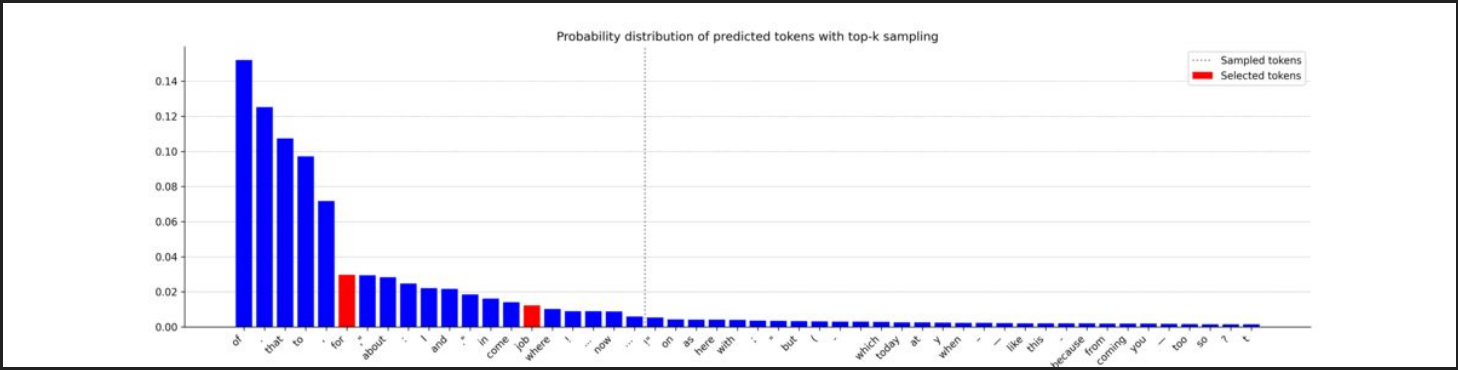
1. for each hypothesis $y_{1:t-1,i}$ that we are tracking:
   find the top k tokens $y_{t,i,1}, ..., y_{t,i,k}$ — very easy, we get this from the softmax log-probs
2. sort the resulting k^2 length t sequences by their *total* log-probability
3. keep the top k
4. advance each hypothesis to time $t+1$

Two missing pieces:

1. If one sequence is a prefix of another, it will have higher score.
   - Divide by seq length.
2. When to stop decoding?
   - Cut a branch after reaching <EOS>.
   - Generate until N hypotheses collected, or length H reached.

Alternatives to beam search

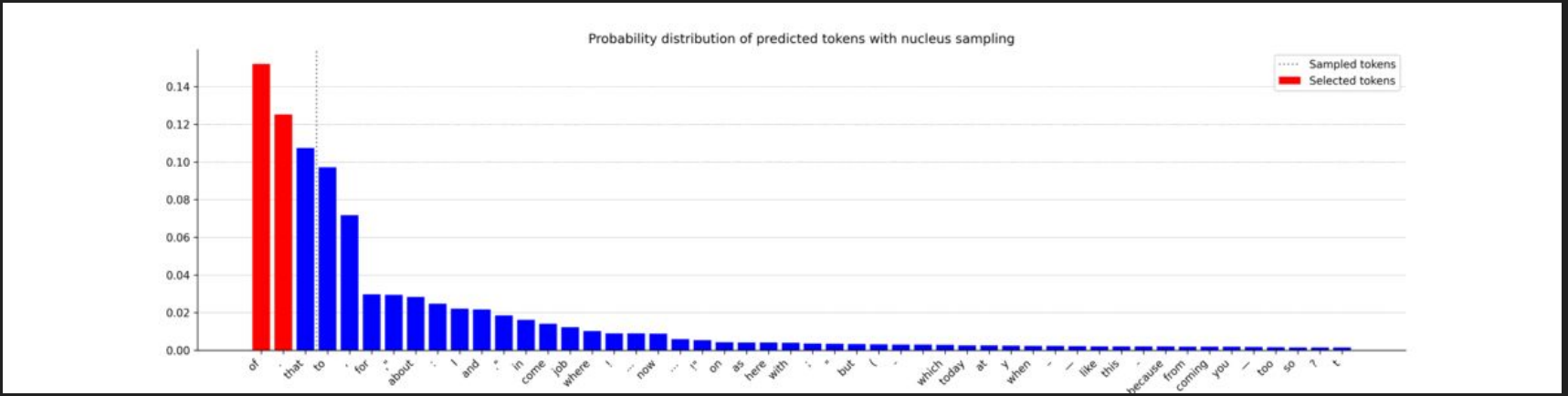
- Top-k sampling
  - Draw from the pool of k most probable tokens
  - Respect relative probabilities



source

Alternatives to beam search

- Top-p sampling (nucleus sampling)
  - Similar to top-k sampling
  - Fix probability mass (p) not token number (k)



source

Feedback is a gift

https://tinyurl.com/dnn-2025-11-26