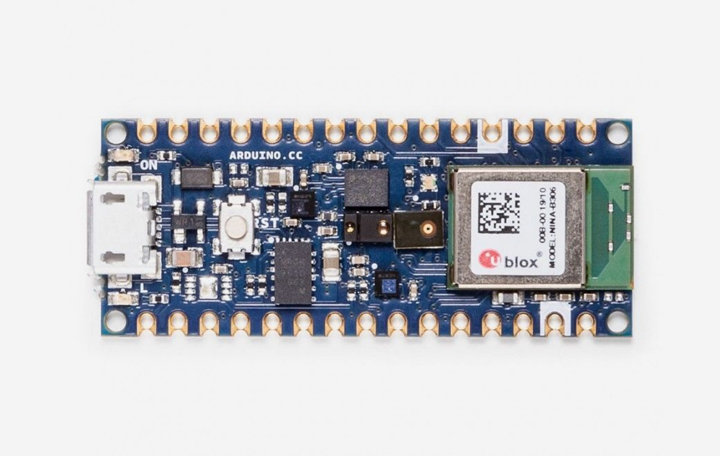
“Latest experiments reveal AI is still terrible at naming paint colors” - bit.ly/ai-paint

Applied Deep Learning

Lecture 8 • Oct 24, 2019

Agenda & announcements

- Arduino demos
- Midterm review
- Intro to sequences
- HW4 (VQA) will be published prob Fri evening

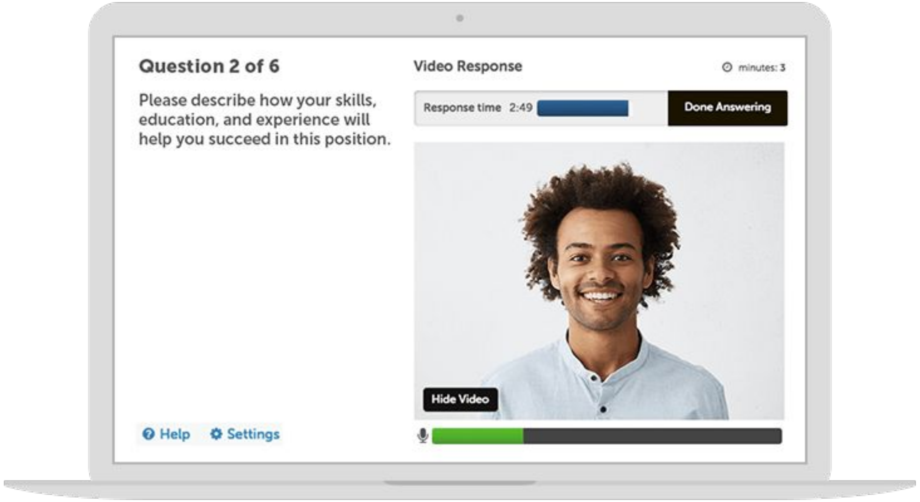


ARDUINO NANO 33 BLE SENSE

News

HireVue

- https://www.hirevue.com/products/video-interviewing
- https://www.washingtonpost.com/technology/2019/10/22/ai-hiring-face-scanning-algorithm-increasingly-decides-whether-you-deserve-job/



HireVue

PRODUCTS + WHY HIREVUE + CUSTOMERS + RESOURCES + COMPANY + LOGIN SEE A DEMO

THE HIREVUE ASSESSMENT MODEL DEVELOPMENT PROCESS

HireVue does not offer a one-size-fits-all algorithm that evaluates all candidates for all job types in the same way. Each assessment model is purpose-built for a specific job role after following these critical steps:

- Ensure that there is a clear performance indicator for the job role that differentiates the strongest from the least promising performers.
- Ask the right questions to elicit responses that can be measured and that are pertinent to predicting job performance based on IO psychology research.
- Train the model to notice everything that is relevant in the interview (what someone says and how they say it), and build a model that uses only the data points that help predict success in the job.
- Rigorously audit the algorithms to ensure that they aren't adversely impacting protected groups.
- Remove features that may cause biased results.
- Re-train the model.
- Re-test the model.
- Repeat these procedures as needed so the algorithm evolves with the customer's data and changing requirements of the job.

THE HIREVUE ASSESSMENT MODEL DEVELOPMENT PROCESS

HireVue does not offer a one-size-fits-all algorithm that evaluates all candidates for all job types in the same way. Each assessment model is purpose-built for a specific job role after following these critical steps:

- Ensure that there is a clear performance indicator for the job role that differentiates the strongest from the least promising performers.
- Ask the right questions to elicit responses that can be measured and that are pertinent to predicting job performance based on IO psychology research.
- Train the model to notice everything that is relevant in the interview (what someone says and how they say it), and build a model that uses only the data points that help predict success in the job.
- Rigorously audit the algorithms to ensure that they aren't adversely impacting protected groups.
- Remove features that may cause biased results.
- Re-train the model.
- Re-test the model.
- Repeat these procedures as needed so the algorithm evolves with the customer's data and changing requirements of the job.

DL is not yet up to this task

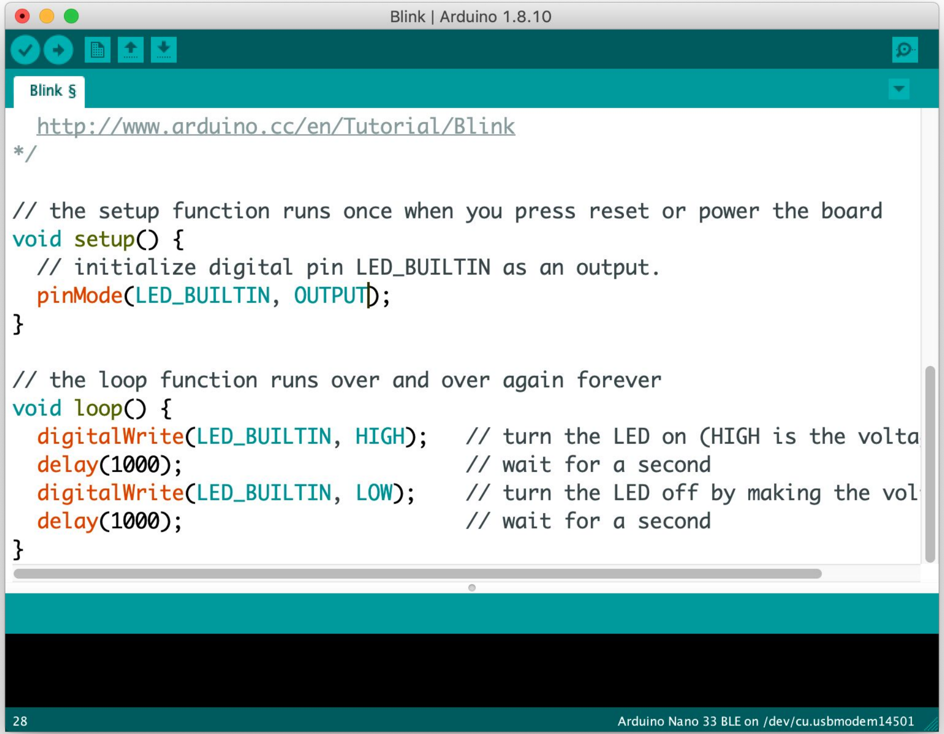
- With imaging, let alone speech.
- You can drastically change the predictions of a commercial computer vision system just by rotating an imge.
- Quick demo (just tossed together a moment ago)

https://cloud.google.com/vision/

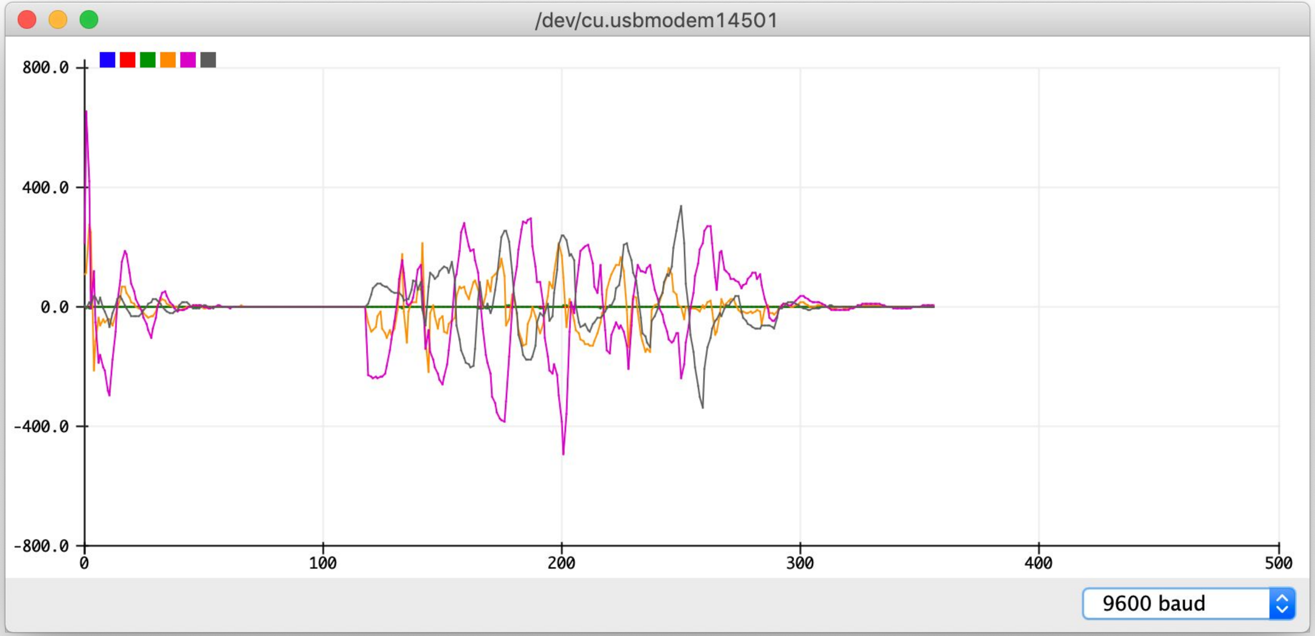
Arduino demos

What’s a microcontroller?

- System on a chip
- Digital & analog input and output
- Read from (and write to) pins

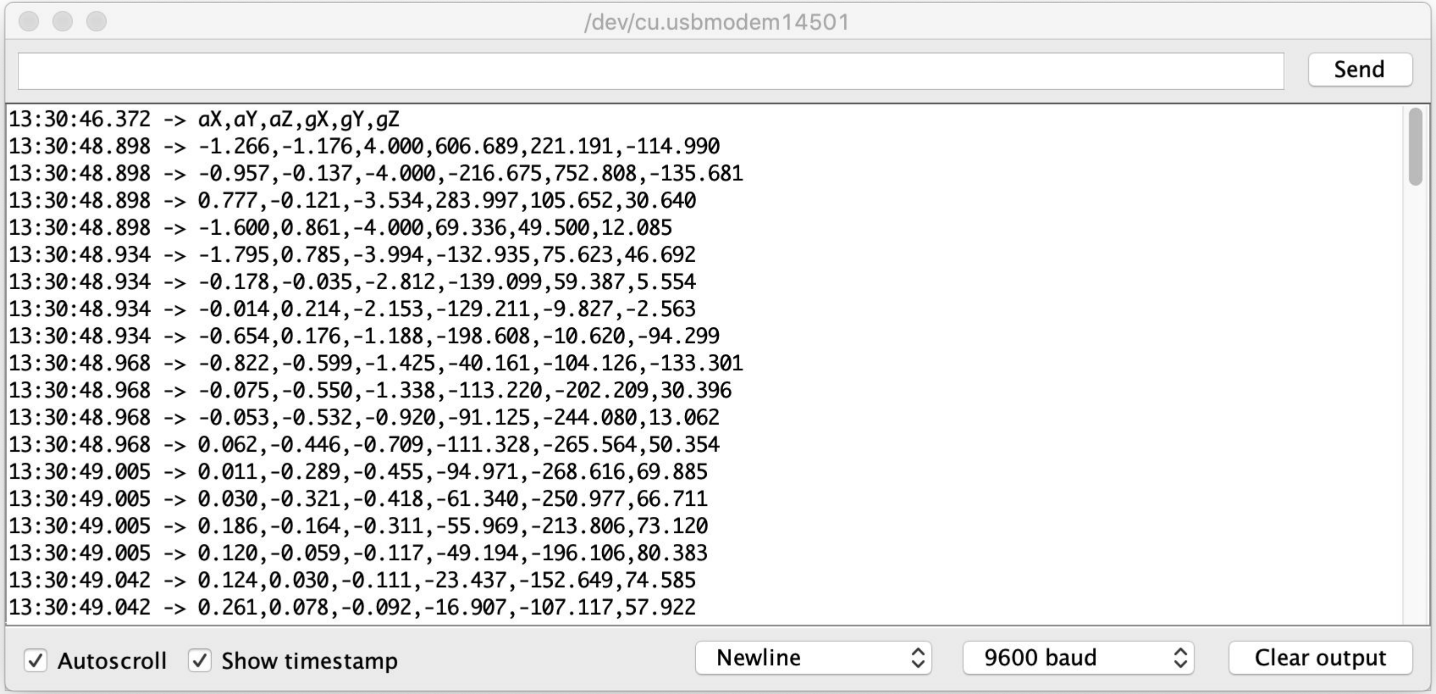


IMU Capture


/dev/cu.usbmodem14501
800.0
400.0
0.0
-400.0
-800.0
0
100
200
300
400
500
9600 baud

Raw data as a CSV

Samples at >100Hz



Training models

- Good news, this will feel familiar to you (in fact, you can probably spot a couple of things to improve in the existing sample)

- [Walkthrough]

Deploying models

- Demo

References

[How-to Get Started with Machine Learning on Arduino]

- I spent time going through each example, they work as expected with a (relatively) low amount of friction for something this new.

- Also have a starter kit you're welcome to borrow wires etc from.

Midterm review

About the exam

On 11/7, you have the full class period (should take about 90 mins). Closed book / closed notes / closed laptops. Please bring a pencil or pen.

- Questions are short answer, multiple choice, and “circle the bug” programming questions (with code similar to the HW).

- Questions drawn from slides and the reading (always on the last slide of each class).

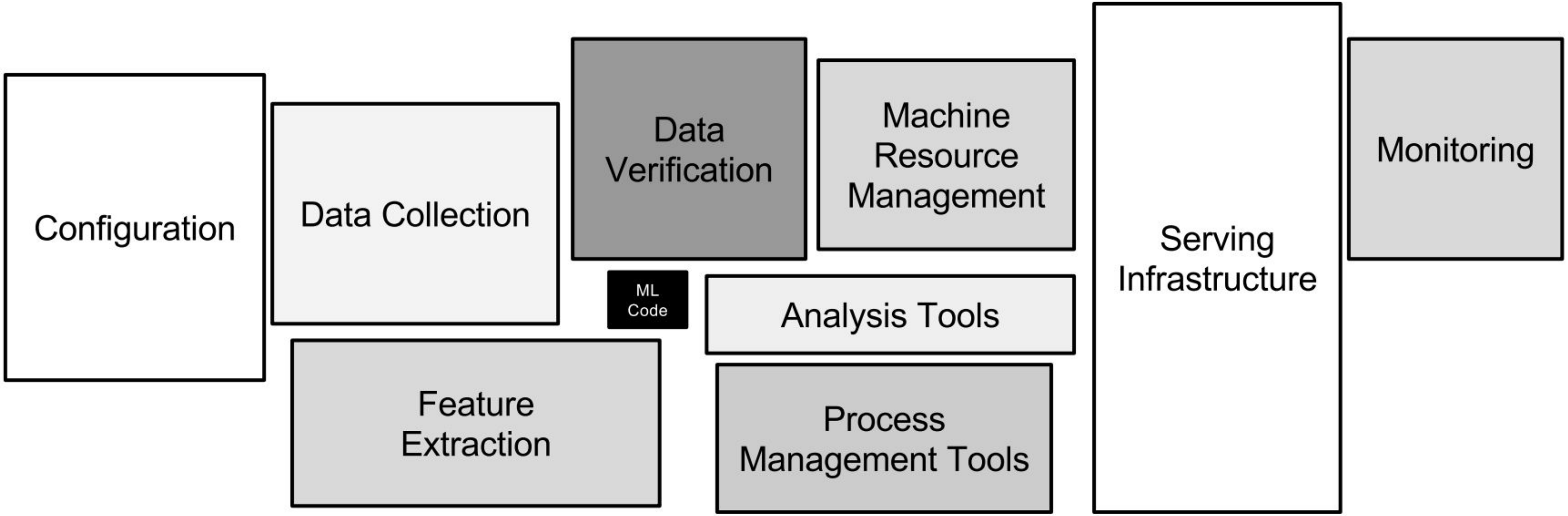
Testing (will be on there)

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.

Defining and tuning an accurate model.

[Hidden Technical Debt in Machine Learning Systems]

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.



[Hidden Technical Debt in Machine Learning Systems]

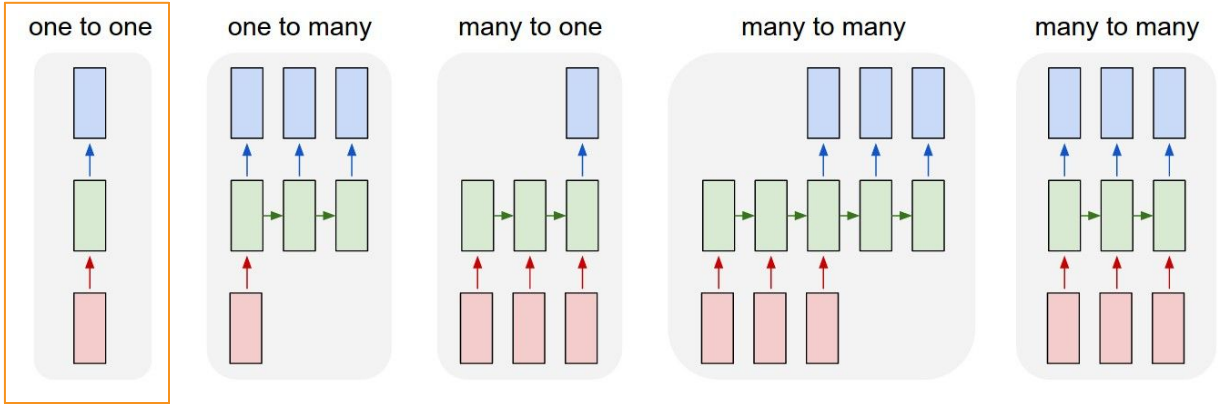
Anti-pattern

In order to serve an embedding trained with an Estimator, you can send out the lower dimensional representation of your categorical variable along with your normal prediction outputs. Embedding weights are saved in the SavedModel, and one option is to share that file itself. Alternatively, you can serve the embedding on demand to clients of your machine learning team—which may be more maintainable,because those clients are now only loosely coupled to your choice of model architecture. They will get an updated embedding every time your model is replaced by a newer, better version.

Better: versioning

https://tfhub.dev/google/universal-sentence-encoder-multilingual-qa/1

Sequences

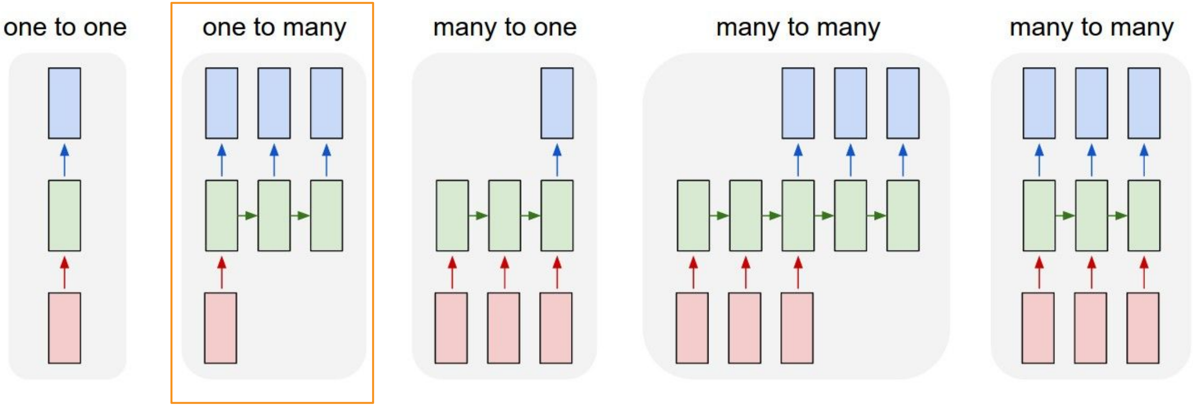


A tensor in, a tensor out.

Image classification.

images in labels out

Diagram shamelessly borrowed from this excellent article.



Text generation

Example here (basic), and here (advanced) - start with the basic example.

```
BIANCA:
Fol, lead; he may drum!
Wear-bloos here, that where she buses
To that shampered as I am here?
```

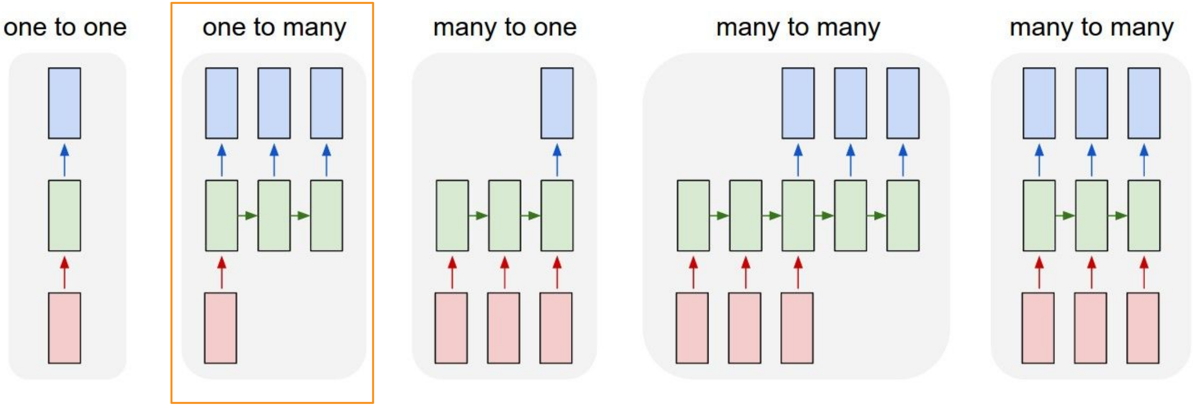


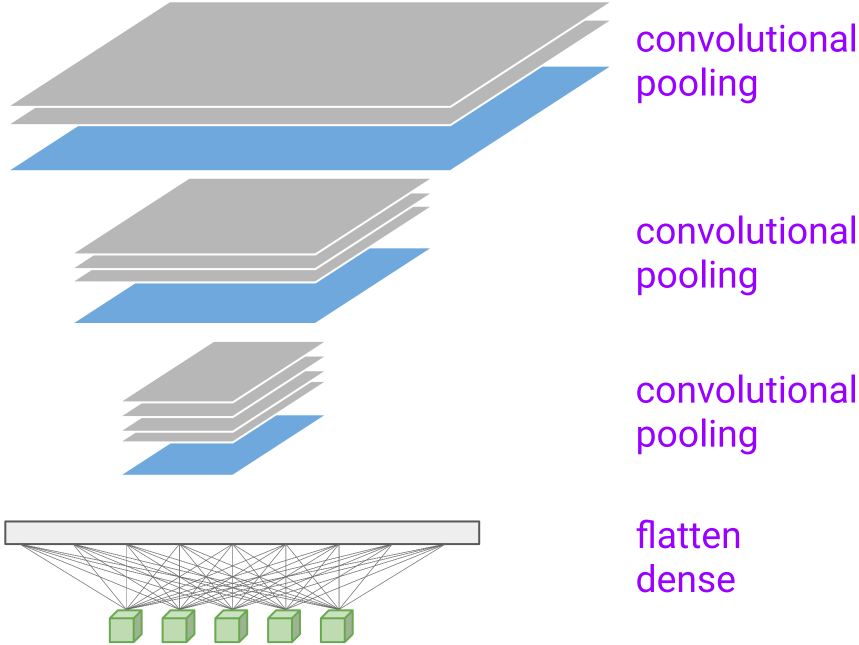
Image captioning

Example [here](#)

Encoder (CNN), decoder (RNN). At training time, input to the RNN is the image embedding produced by the CNN + the desired caption. At test time, input to decoder is embedding + start token.

"a surfer riding on a wave"

[Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.](#)

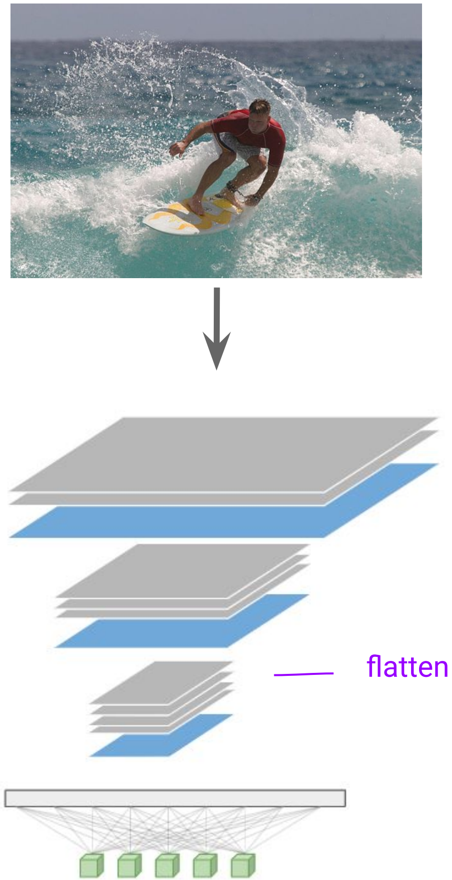


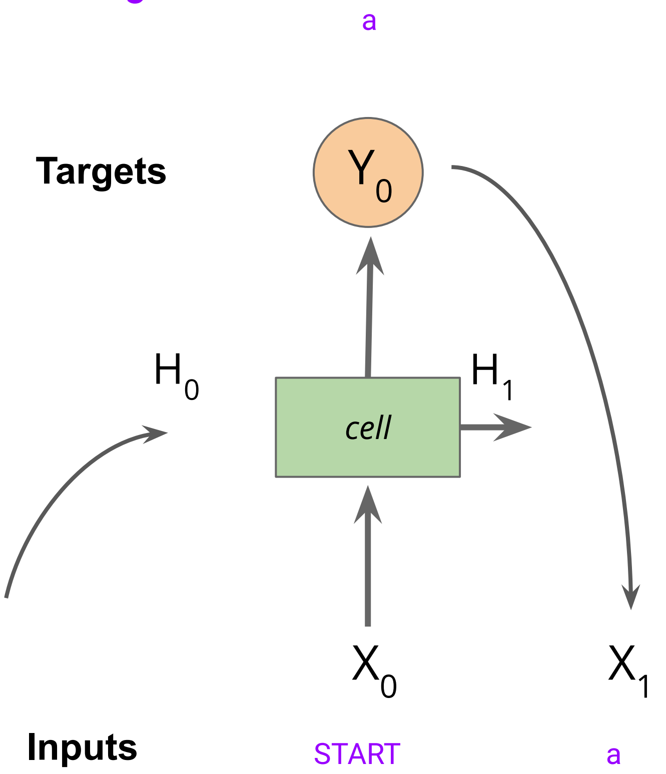
A CNN trained to classify images, per usual.

2. Forward an image. Extract and flatten the activation map from a convolutional layer. This is a image embedding!

(How do we know which layer works best? Experiment).

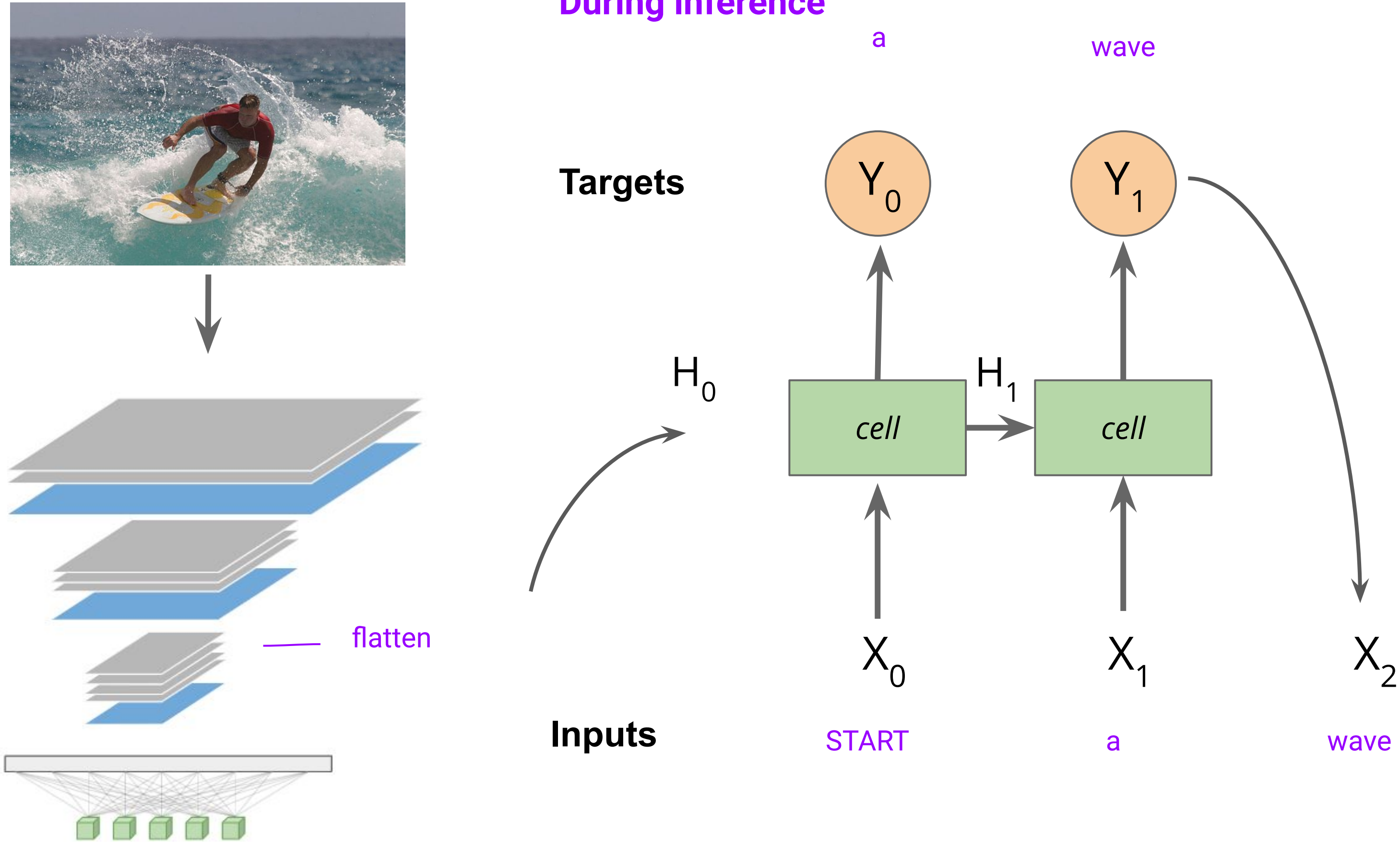
During inference

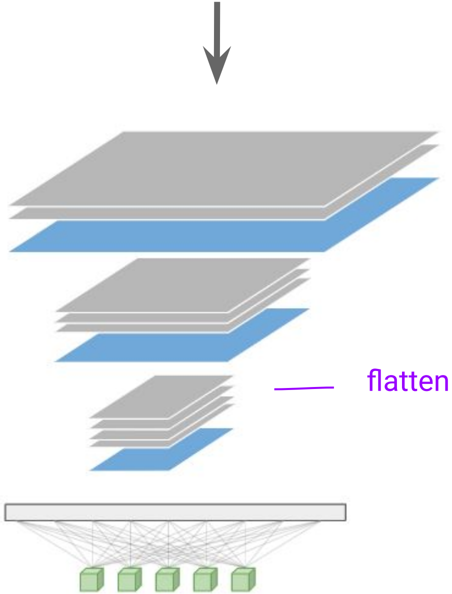




Sample from the output - more on this later. Feed that and in the cell state as inputs to the next step.

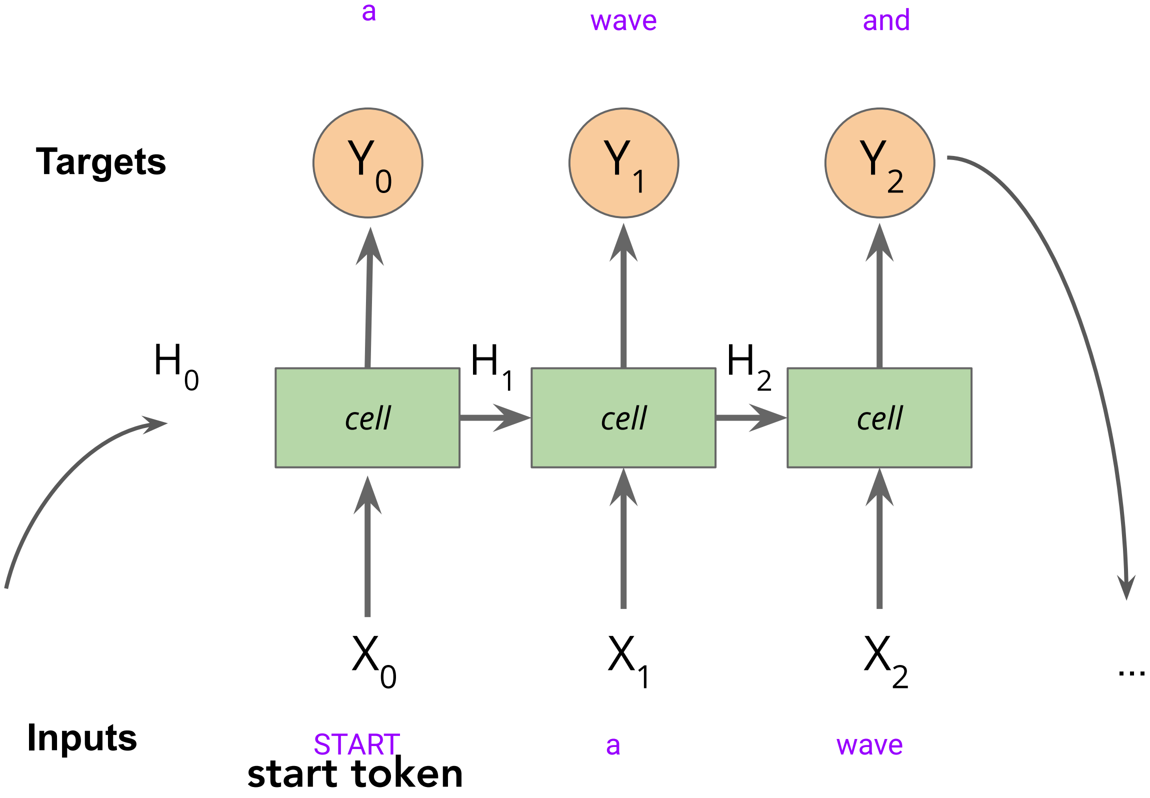
During inference



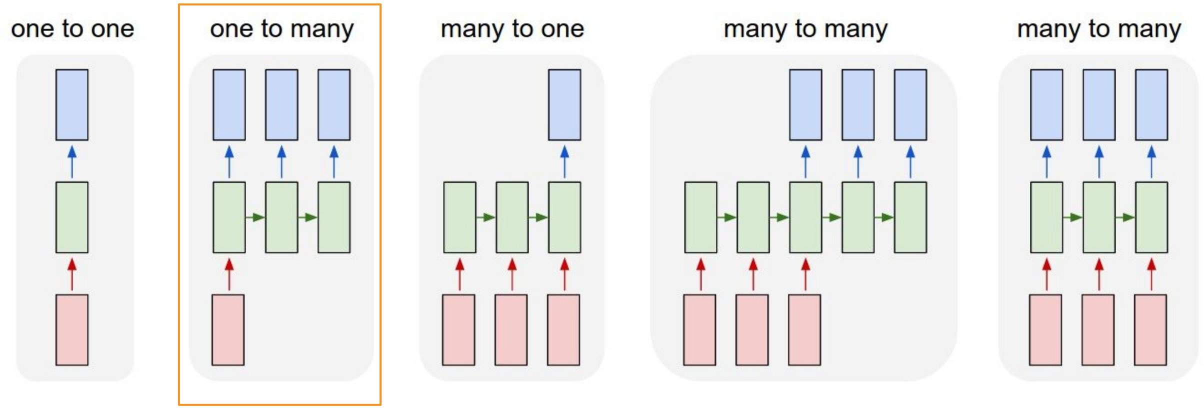


flatten

During inference

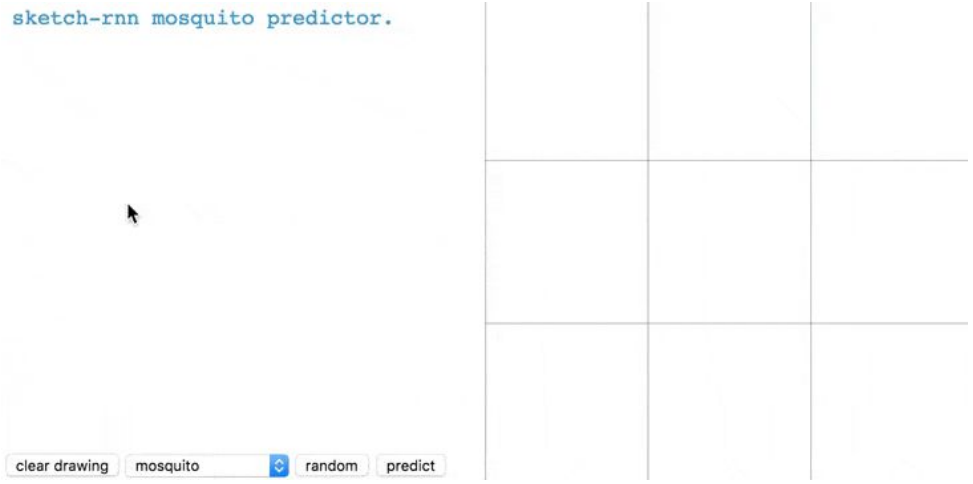


Sample until max_len or special END character reached.

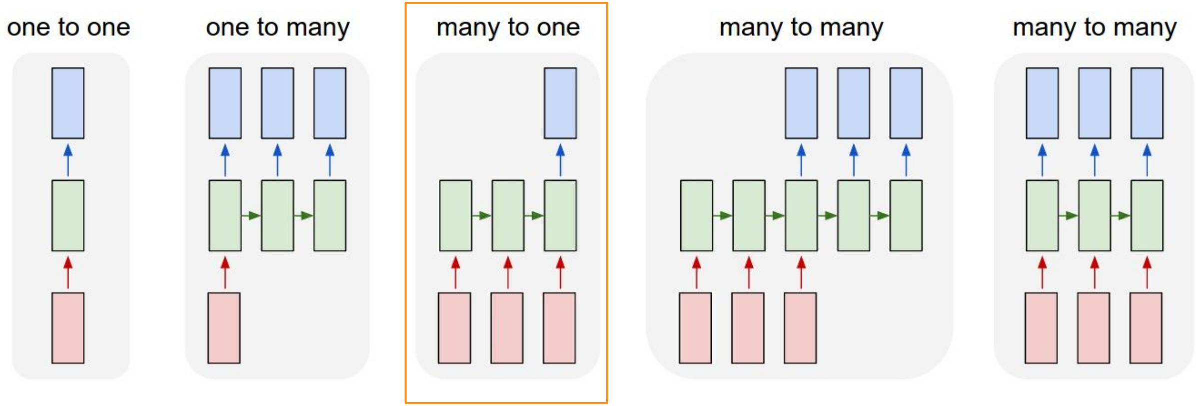


Sketch-RNN

Similar idea

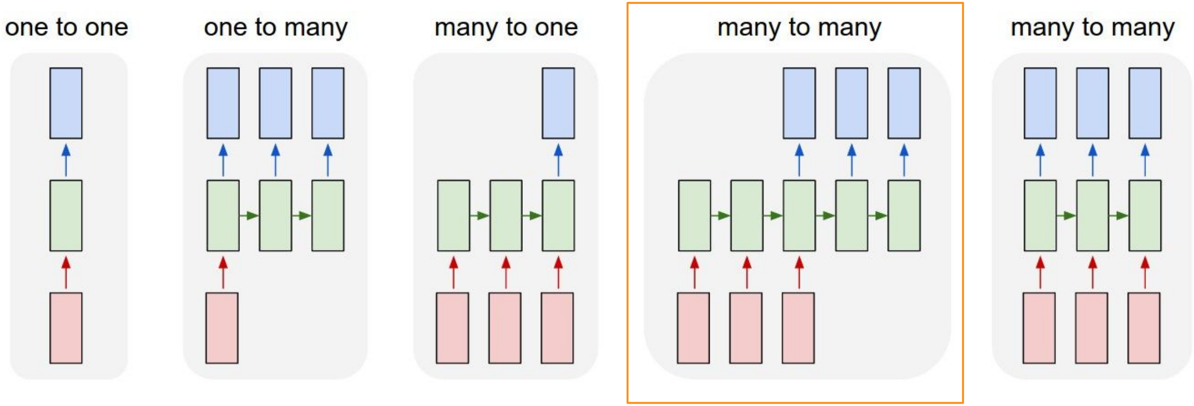


magenta.tensorflow.org/sketch-rnn-demo



Timeseries forecasting, visual question answering, sentiment analysis.

Example [here](#).



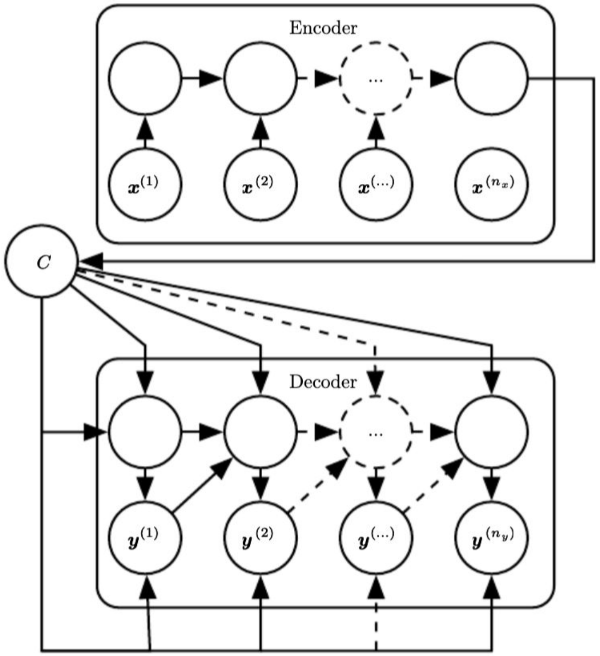
Machine translation

Example [here]

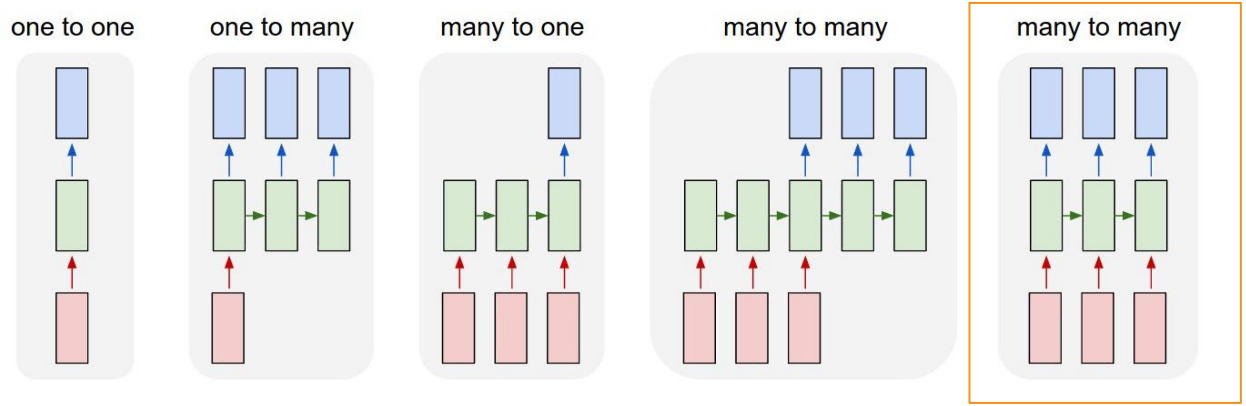
[Sequence to Sequence Learning with Neural Networks]

Encoder-Decoder

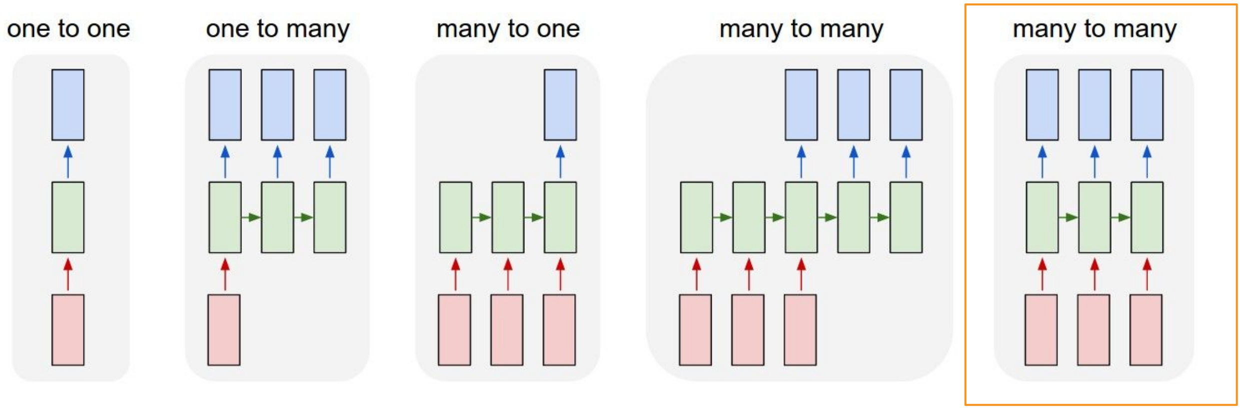
- Contrast with phrase-based SMT
- Compress a sentence into a vector that encodes the "meaning".
- Interlingual representations (multiple languages can be mapped to the same encoding).



https://www.deeplearningbook.org/contents/rnn.html 10.4


one to one
one to many
many to one
many to many
many to many

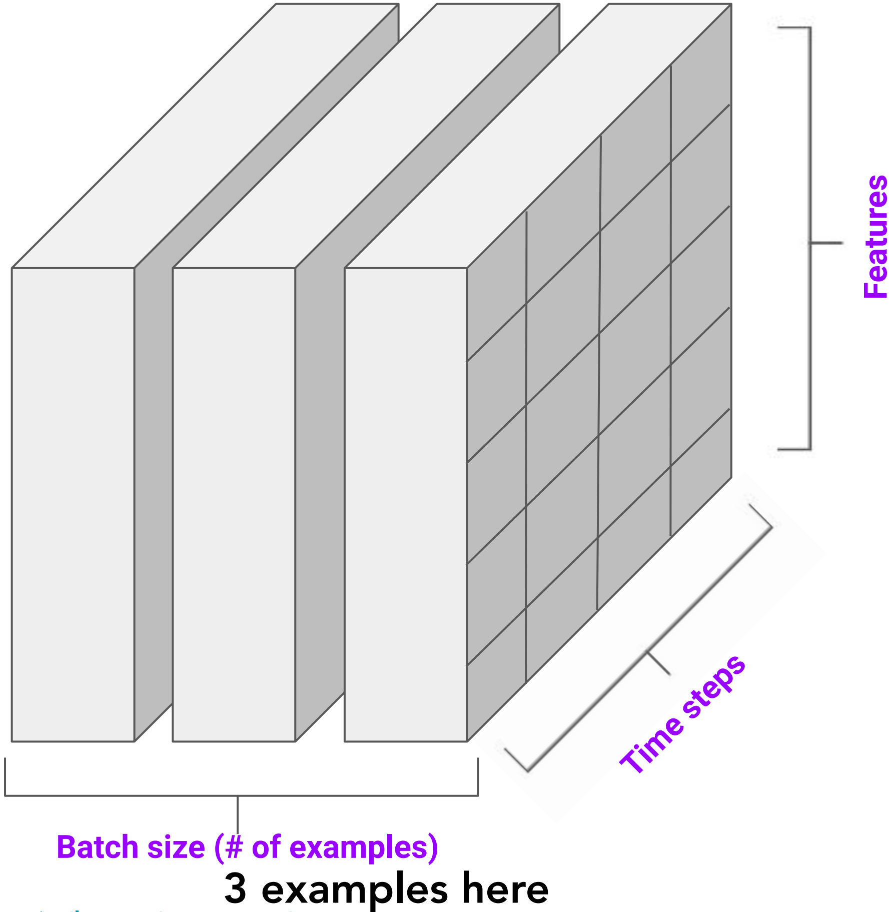
?



Video translation (frame by frame)

Trained using unpaired data. Why unpaired?

Practical perspective



keras.io/layers/recurrent/

```
model = Sequential()
model.add(LSTM(32,input_shape=(5,4))
```

32 units, 5 timesteps, 4 features per timestep (batch size is inferred)

temperature pressure humidity density

	T	P	H	D
12pm				
1pm				
2pm				
3pm				
5pm				

RNNs shapes

Input

- RNNs take (batch_size, timesteps, input_features)

Output

- (batch_size, timesteps, output_features) -- if return_sequences=True
- (batch_size, output_features) - if return_sequences=False

Use return_sequences=True for all RNN layers except the last

You’ll often see embeddings used as an input to RNNs

One-hot encoding

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

8-dimensional embedding

0.3	0.6	0.1	1.9	0.3	0.6	0.1	1.9	0.3	0.6	0.1	0.1	0.3	0.6	0.1	1.9	0.3
1.2	2.1	1.9	1.5	1.2	2.1	1.9	1.5	1.2	2.1	1.9	1.3	1.2	2.1	1.9	1.5	1.2
1.6	2.2	2.1	0.3	1.6	2.2	2.1	0.3	1.6	2.2	2.1	2.1	1.6	2.2	2.1	0.3	1.6
0.9	1.3	1.8	2.2	0.9	1.3	1.8	2.2	0.9	1.3	1.8	0.5	0.9	1.3	1.8	2.2	0.9
0.3	1.5	0.8	1.8	0.3	1.5	0.8	1.8	0.3	1.5	0.8	0.7	0.3	1.5	0.8	1.8	0.3
0.5	1.0	0.9	1.1	0.5	1.0	0.9	1.1	0.5	1.0	0.9	1.9	0.5	1.0	0.9	1.1	0.5
1.6	1.4	3.2	0.4	1.6	1.4	3.2	0.4	1.6	1.4	3.2	1.4	1.6	1.4	3.2	0.4	1.6
2.3	0.6	1.1	1.6	2.3	0.6	1.1	1.6	2.3	0.6	1.1	0.6	2.3	0.6	1.1	1.6	2.3

Here, we’re creating 8 features for each word in our sequence

Text preprocessing

TF2 has several ways to do this - for now, the Keras utilities are the best (all of them work in TF2)

Workflow

Tokenize -> vectorize -> pad -> embed

Tokenize and vectorize

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences


s = "Alice in wonderland."
s2 = "When suddenly Alice saw a White Rabbit."
x_train = [s, s2]
```

Tokenize and vectorize

```
max_words = 1000 # limits vocab to n most common tokens
t = Tokenizer(num_words=max_words)
t.fit_on_texts(x_train)

vectorized = t.texts_to_sequences([s])
print(vectorized)
[[1, 2, 3]]
```

Notes

Fit your tokenizer on the training data only. You can pass an OOV (out of vocabulary) token to represent previously unseen words in validation / test, otherwise they'll be dropped.

```
>>> t.word_index

{'a': 7,
 'alice': 1,
 'in': 2,
 'rabbit': 9,
 'saw': 6,
 ...
```

Sorted by frequency

```
>>> dir(t)
[ ...
 'num_words',
 'oov_token',
 'sequences_to_matrix',
 'split',
 'texts_to_matrix',
 'texts_to_sequences',
 'word_counts',
 ...
```

A useful way to see what properties are available on a Python object (especially when the API docs are sparse)

Note: source for the preprocessing utilities lives here (can be difficult to tell from the API docs; they're generated)

Pad

```
max_len = 10 # pad sentences if shorter than this, trim otherwise
padded = pad_sequences(vectorized, maxlen=max_len, padding='pre')
print(padded)


[[0 0 0 0 0 0 0 1 2 3]]
```

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense, Embedding, SimpleRNN
from keras import Sequential
maxwords, maxlen = 10000, 500
(x_train, y_train), (_, _) = imdb.load_data(num_words=maxwords)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)

model = Sequential()
model.add(Embedding(maxwords, 32))
model.add(SimpleRNN(16))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10, validation_split=0.2)
```

A complete program for sentiment analysis on IMDB

Never use SimpleRNN in practice. Prefer GRUs.

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense, Embedding, SimpleRNN
from keras import Sequential
maxwords, maxlen = 10000, 500
(x_train, y_train), (_, _) = imdb.load_data(num_words=maxwords)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
```

Now a deep RNN

```python
model = Sequential()
model.add(Embedding(maxwords, 32))
model.add(SimpleRNN(16, return_sequences=True))
model.add(SimpleRNN(16))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10, validation_split=0.2)
```

Set this parameter when stacking.

Now using GRUs

```
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense, Embedding, GRU
from keras import Sequential
maxwords, maxlen = 10000, 500
(x_train, y_train), (_, _) = imdb.load_data(num_words=maxwords)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
```

```
model = Sequential()
model.add(Embedding(maxwords, 32))
model.add(GRU(16, return_sequences=True))
model.add(GRU(16))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10, validation_split=0.2)
```

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense, Embedding, LSTM
from keras import Sequential
maxwords, maxlen = 10000, 500
(x_train, y_train), (_, _) = imdb.load_data(num_words=maxwords)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
```

Now using LSTMs

```python
model = Sequential()
model.add(Embedding(maxwords, 32))
model.add(LSTM(16, return_sequences=True))
model.add(LSTM(16))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10, validation_split=0.2)
```

Note: there are differences between LSTMs and GRUs hidden by this API (LSTMs will return two hidden state objects when called directly).

Time series forecasting

Terminology

Univariate time series

- Single feature at each step (temperature)

Multivariate time series

- Multiple features at each step (temperature, pressure, humidity)

Code walkthrough

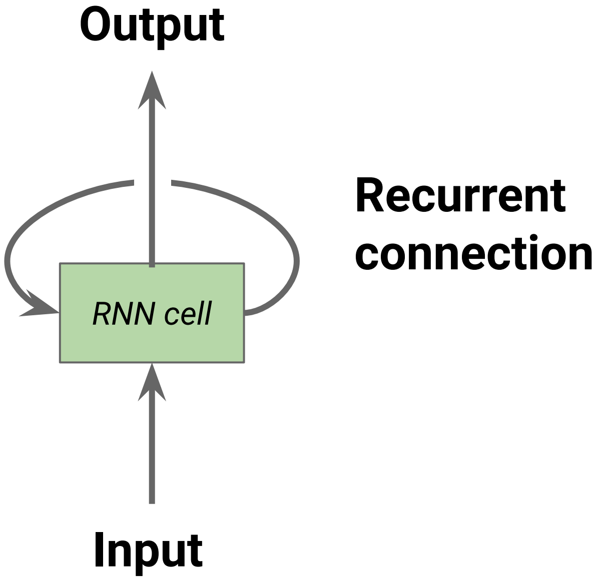
- Published this summer by a Columbia student (Arjun Dcunha - thank you!)

Break

Try the time series forecasting tutorial, it's pretty cool.

tensorflow.org/tutorials/structured_data/time_series

Graphical view of a SimpleRNN



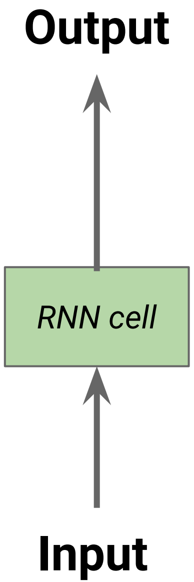
Notes

A “SimpleRNN” corresponds to fig 10.4 from Deep Learning, and the SimpleRNN class from Keras.

Various courses and books use slightly different versions of it. E.g. MIT’s Intro to Deep Learning uses the one here. Stanford’s cs231 uses a “Vanilla RNN” that corresponds to fig 10.3 from Ian’s book.

Not a big deal, concepts are the same.

From a programmer's perspective




```
rnn = RNN()
y = rnn.call(x)
```

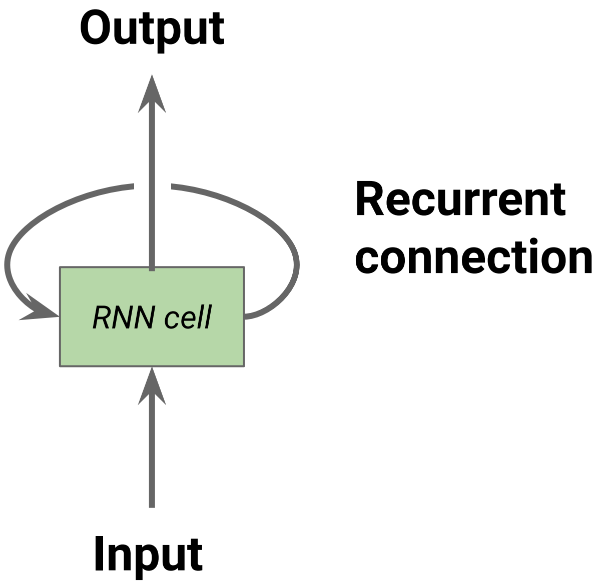
Instantiate a RNN.

Call the RNN on some input data (x), to produce an output (y).

The complexity comes from how the RNN updates its internal state using information from x, y, and its previous state.

A few types of RNN cells (all the built-in ones have the same API in TF2).

Process a sequence one element at a time




```
state = 0
for input in input_sequence:
  output = f(input, state)
  state = output
```

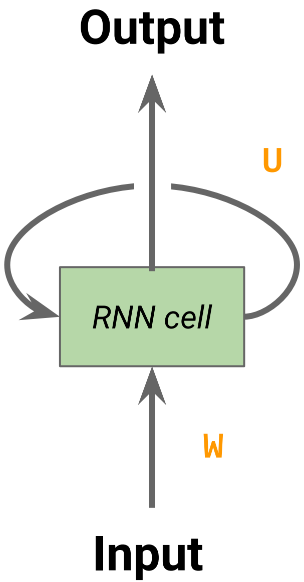
Initialize the hidden state.

Compute the current output as a function of the current input and state.

This RNN simply remembers its previous state.

An RNN is a <u>for loop</u> that reuses quantities computed during the previous iteration of the loop.

Process a sequence one element at a time

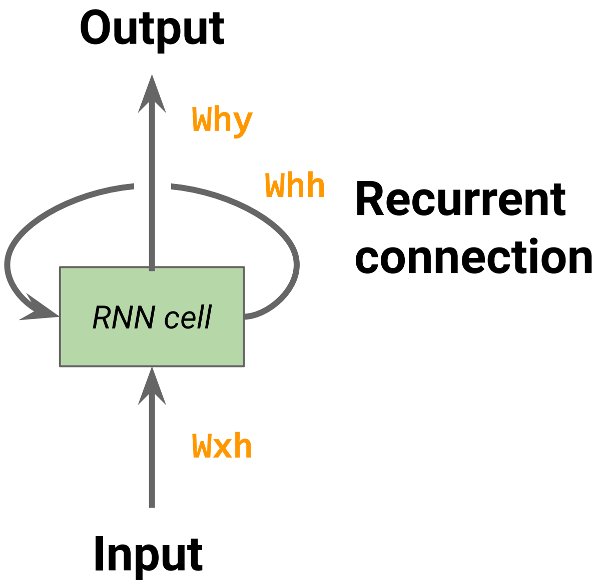


Recurrent connection

```
state = 0
for input in input_sequence:
  output = activation(dot(W, input) +
                      dot(U, state) + b)
  state = output
```

Compute the current output as a function of the current input and state.

The “Vanilla RNN” you may encounter



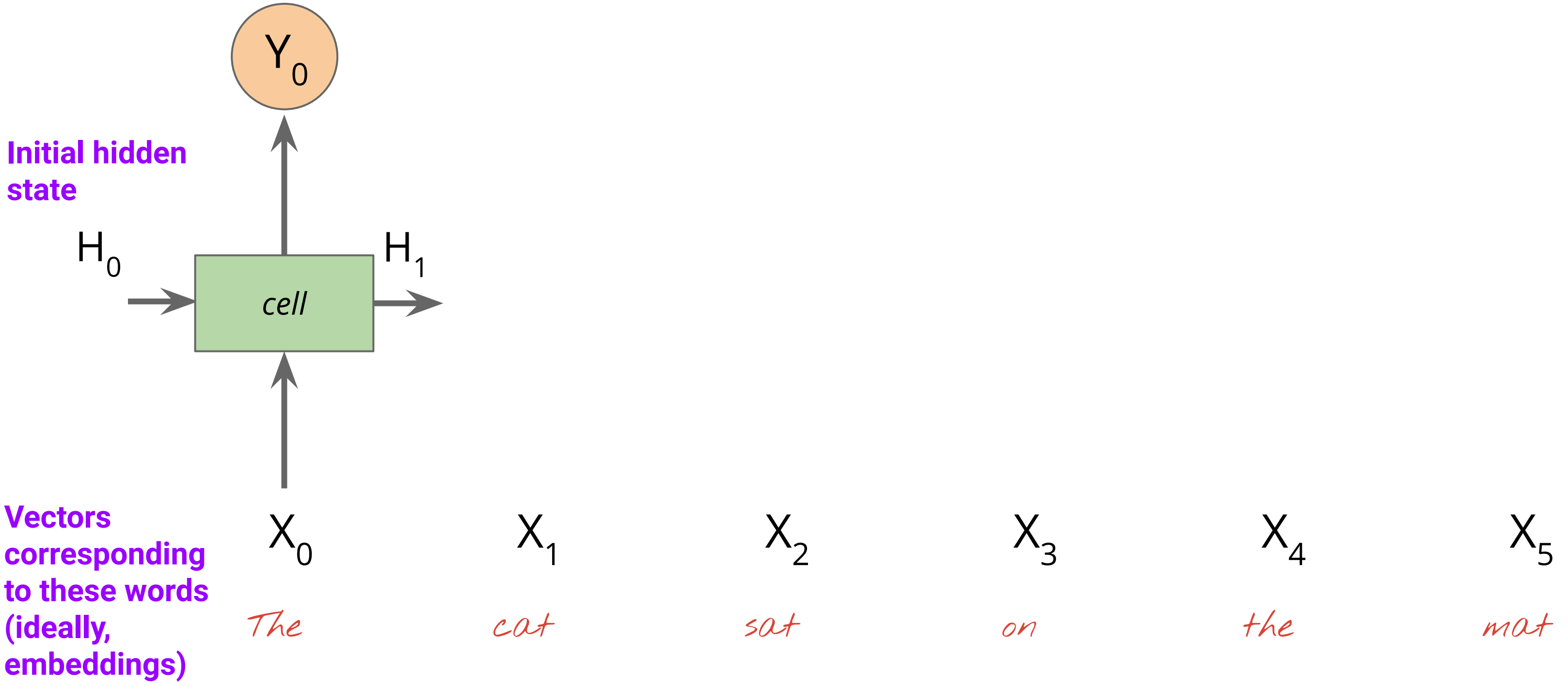

```
state = 0
for input in input_sequence:
  state = activation(dot(Wxh, input) +
                     dot(Whh, state) + b)
  output = dot(Why, state)
```

Three weight matrices now

- **Input -> hidden**
- **Hidden -> hidden**
- **Hidden -> output**

Many other configurations are possible. In practice, it’s fairly easy to experiment with different alternatives and find the type that works best for your problem.

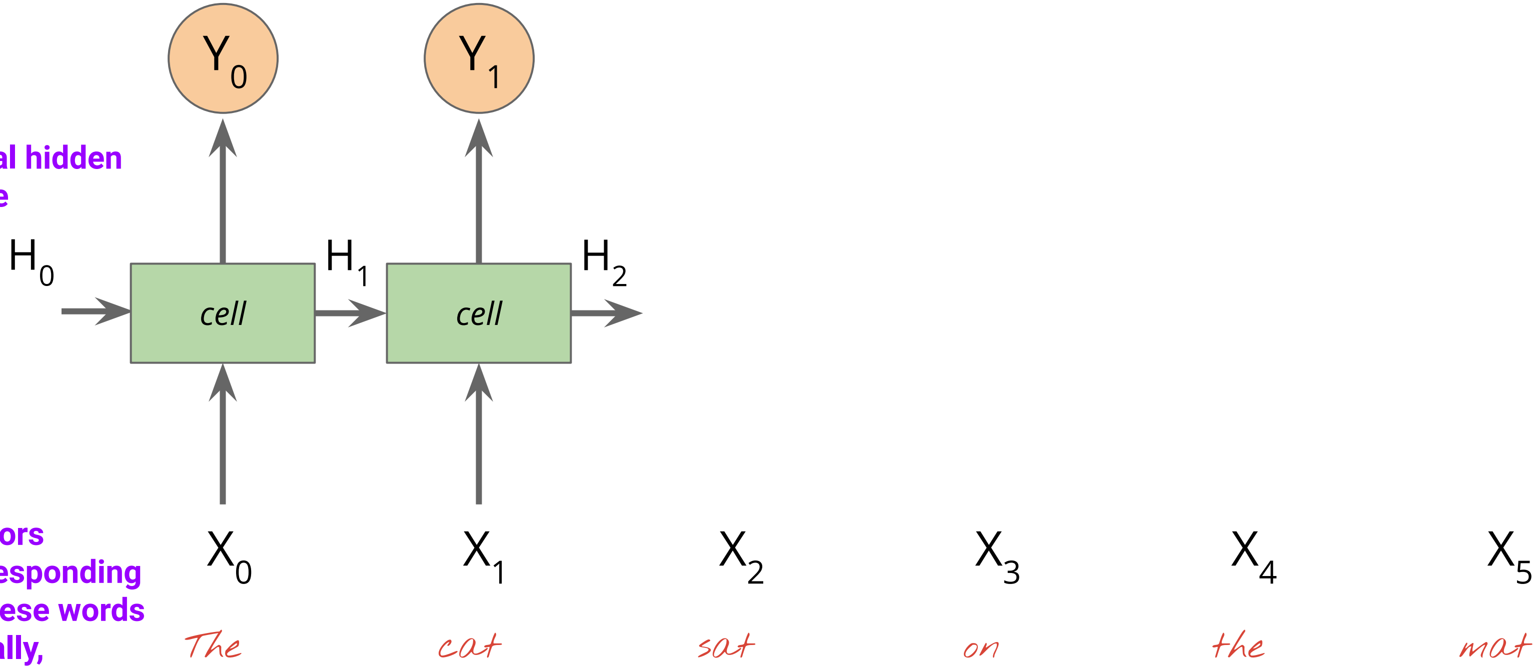
Unfolding



Initial hidden state
Y_0
H_0
H_1
cell
Vectors corresponding to these words (ideally, embeddings)
X_0
X_1
X_2
X_3
X_4
X_5
The
cat
sat
on
the
mat

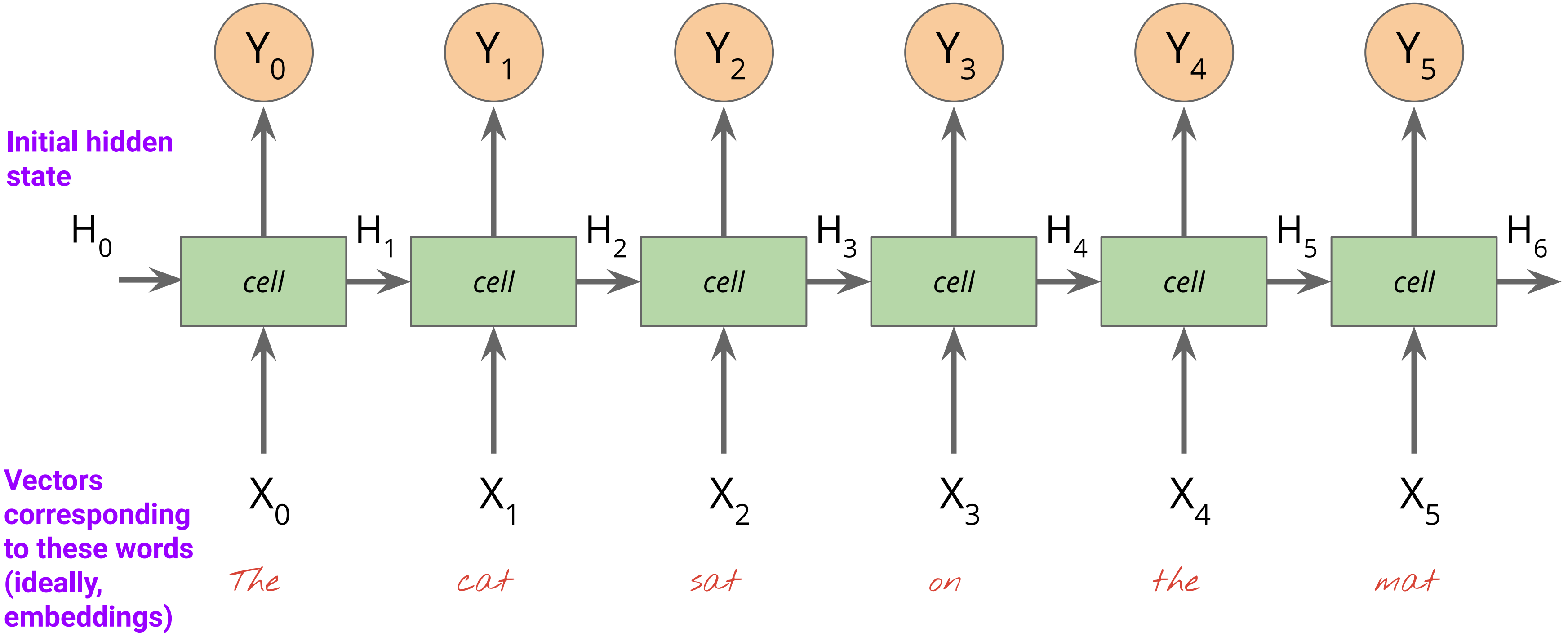
Unfolding

Initial hidden state



Vectors corresponding to these words (ideally, embeddings)

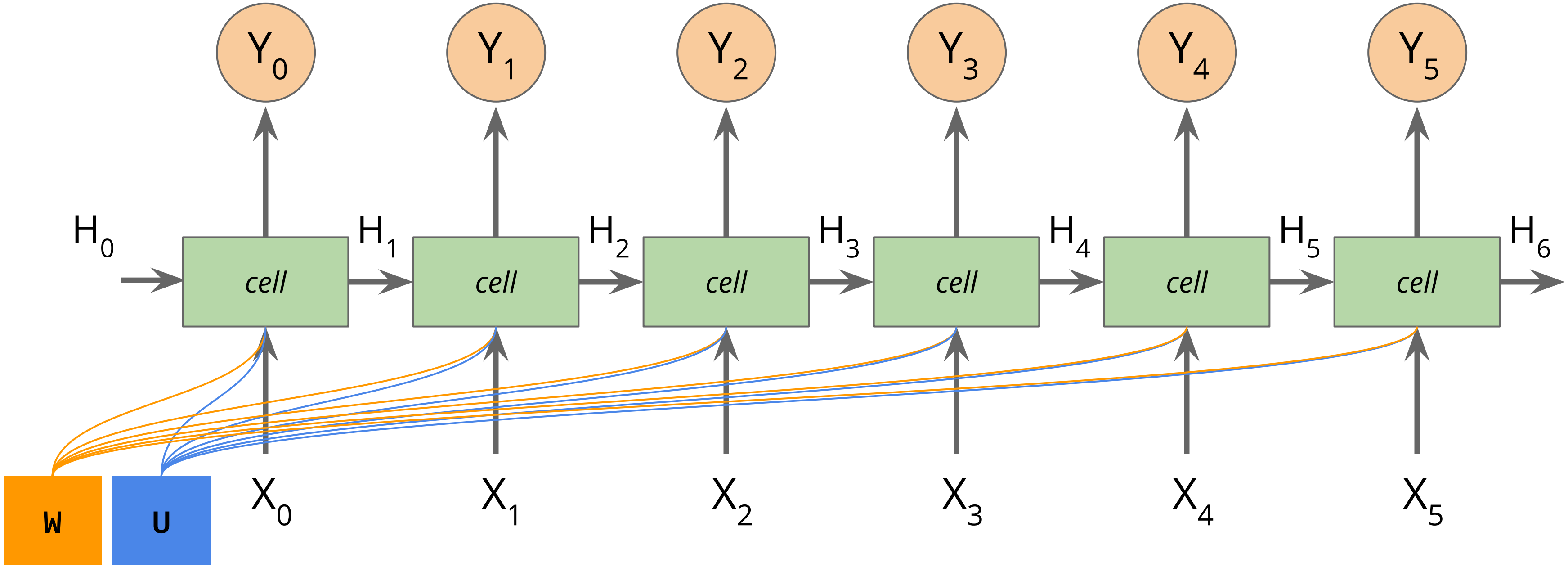
Unfolding



Initial hidden state
Vectors corresponding to these words (ideally, embeddings)
H0
H1
H2
H3
H4
H5
H6
cell
cell
cell
cell
cell
cell
Y0
Y1
Y2
Y3
Y4
Y5
X0
X1
X2
X3
X4
X5
The
cat
sat
on
the
mat

Unfolding

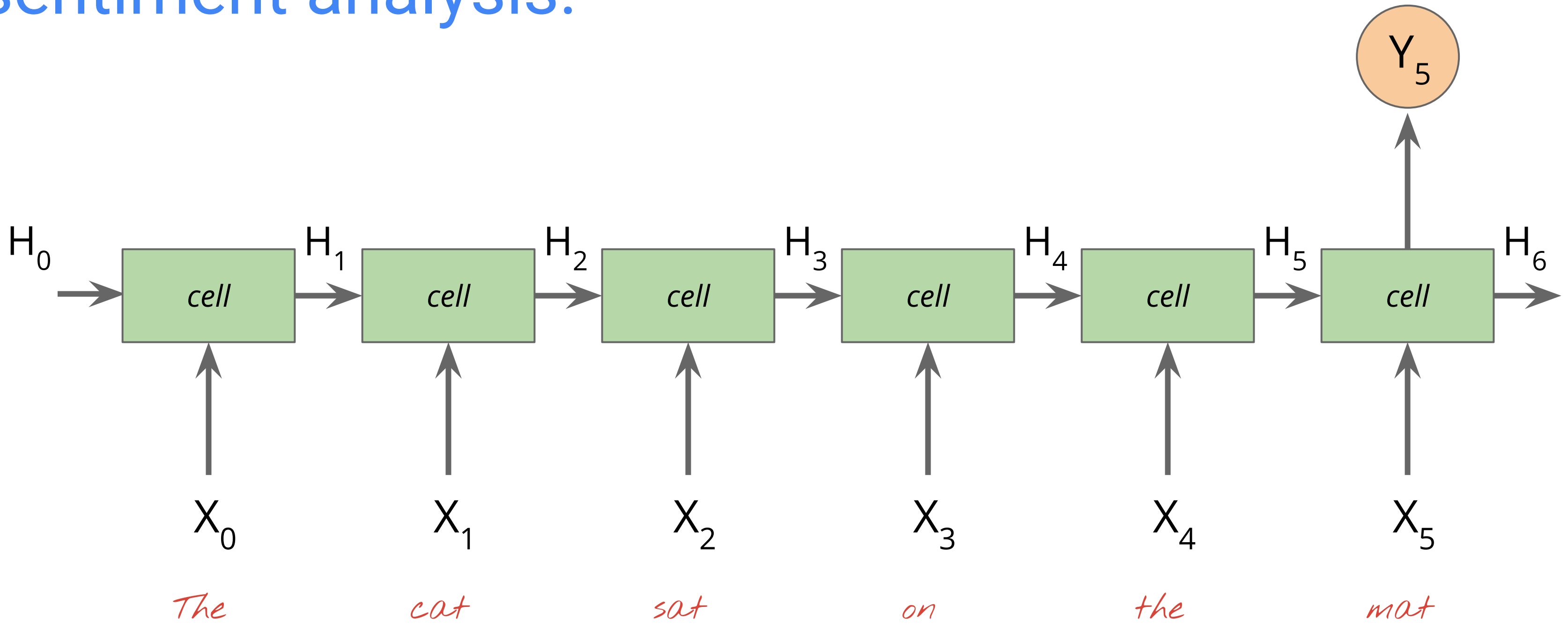
Parameters are shared over time. This entire sequence contains just two weight matrices: W, U.



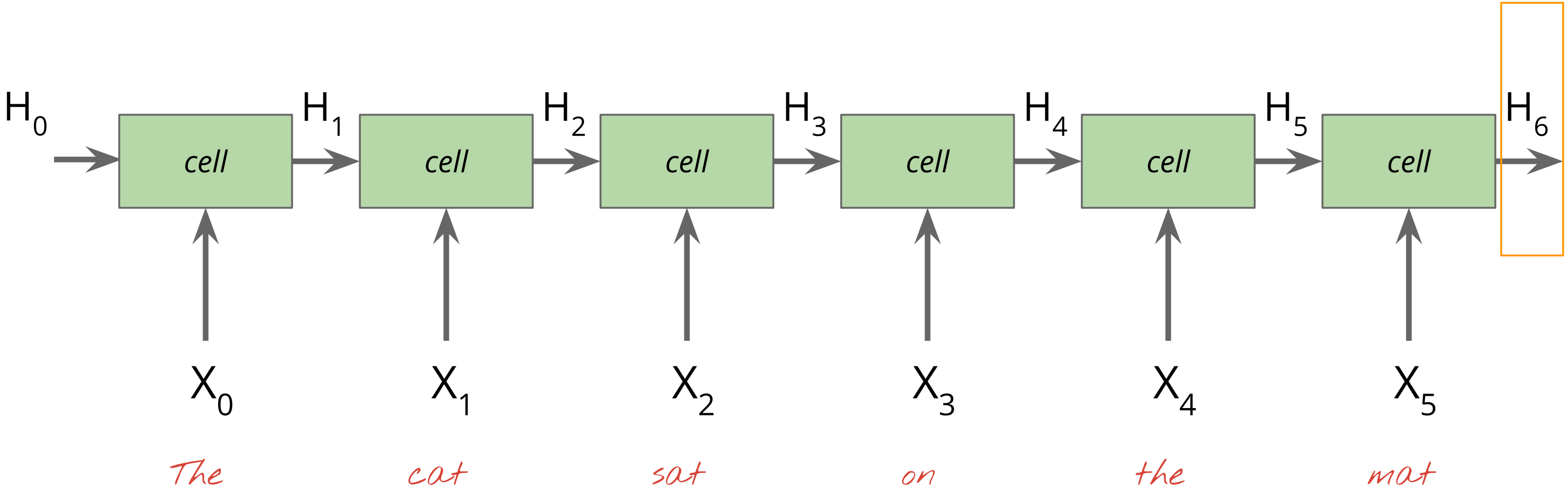
Parameter sharing

- Similar idea to convolution
- Imagine a small 3x3 edge detection filter in a CNN
- This can find edges anywhere on an image
- Likewise, weights inside an RNN can detect features anywhere in a sequence.

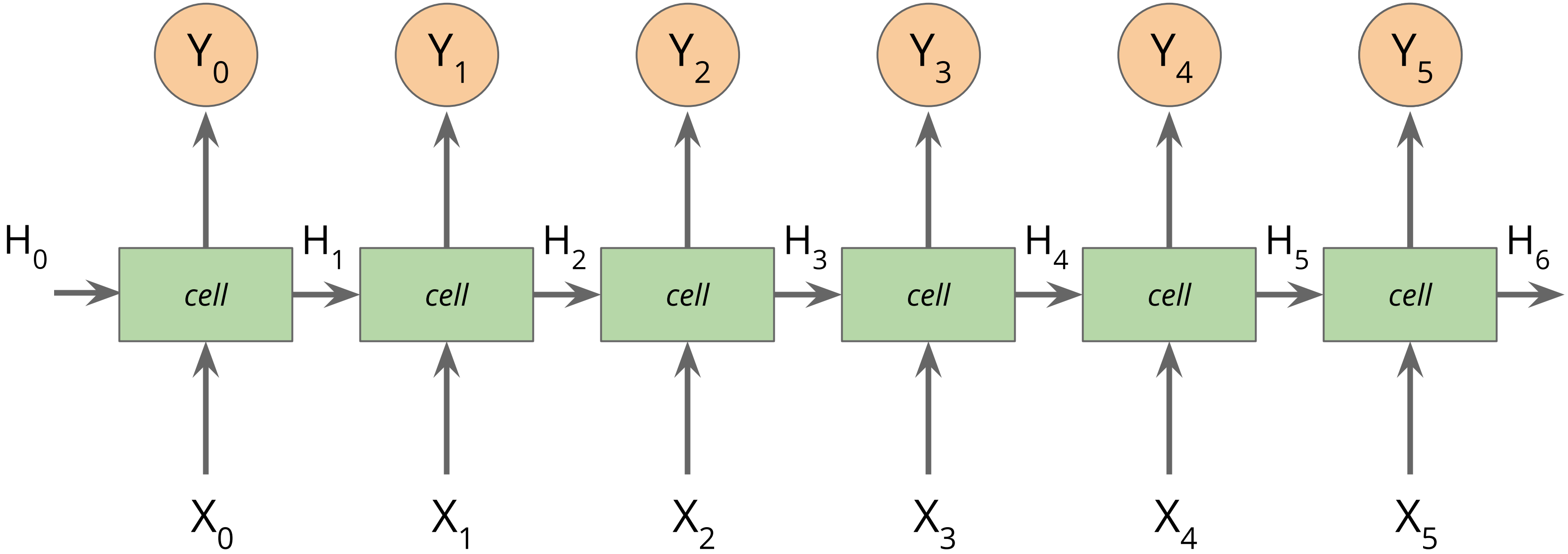
We may only be interested in the last output, say, in sentiment analysis.



Or, in the final hidden state (say, if we want a representation of the sentence).

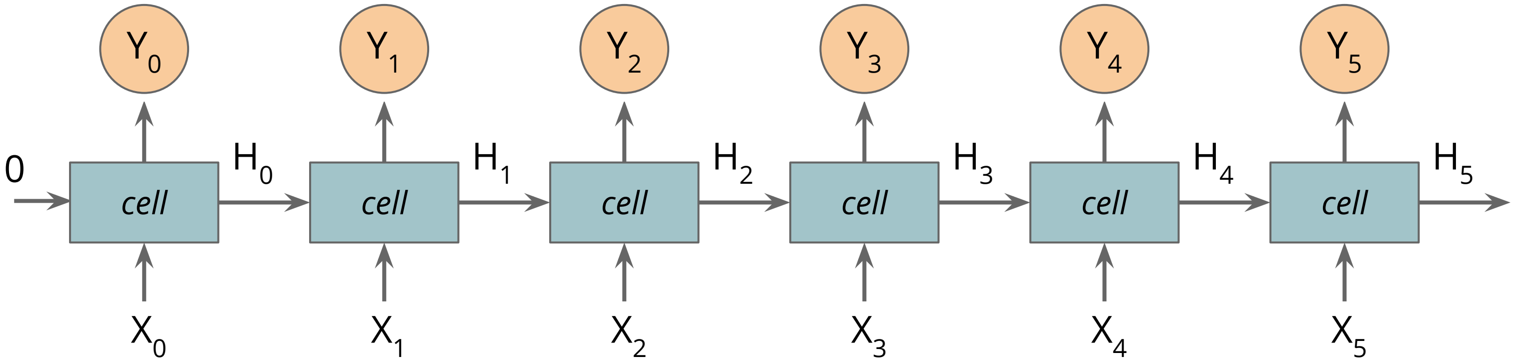


Or, outputs at every step (if this is an intermiedate layer)

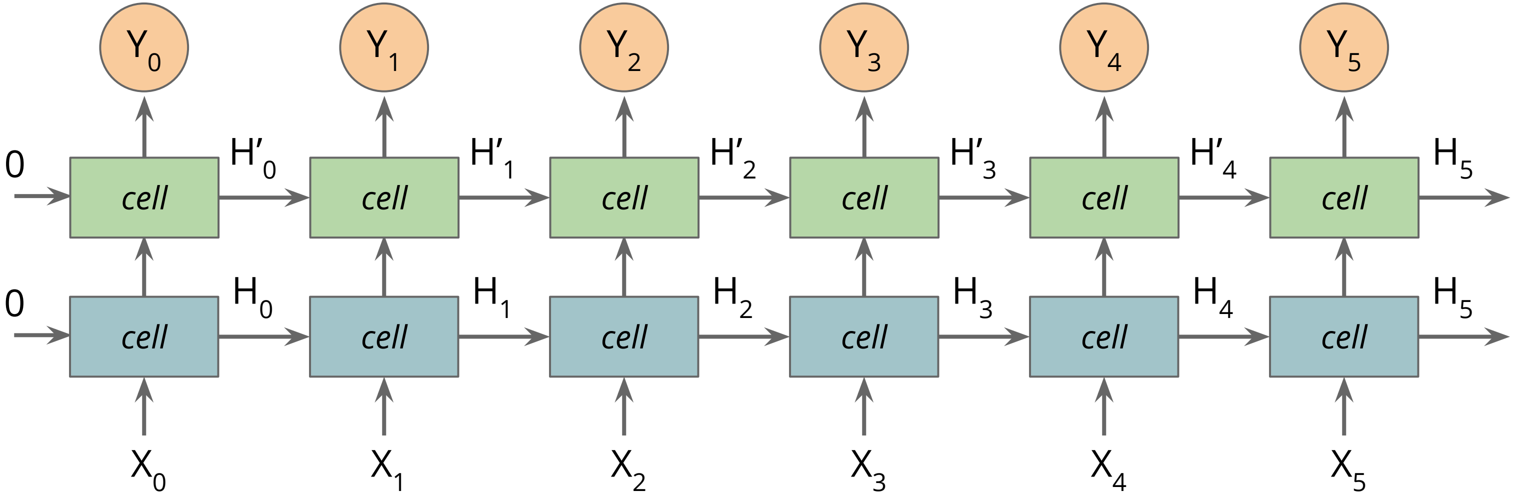


Multilayer RNNs

RNNs can be stacked like other layers



RNNs can be stacked like other layers



API for stacking RNNs

```
rnn = RNN()
rnn2 = RNN()
state1, state2 = 0, 0
for input in input_sequence:
  output1 = rnn1.call(input, state1)
  state1 = output1
  output2 = rnn2.call(output1, state2)
  state2 = output2
```

Instantiate a RNN.

It has a friend!

The second RNN takes the output of the first as input.

How many layers do you need?

A lot of people used a bunch of layers in their homework.

- Are the extra layers necessary?

About how many layers does **Google Translate** use (as of 2016?)

- Ballpark… 1? 5? 100? 1000?

[Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](#)

How many layers do you need?

A lot of people used a bunch of layers in their homework.

- Are the extra layers necessary?

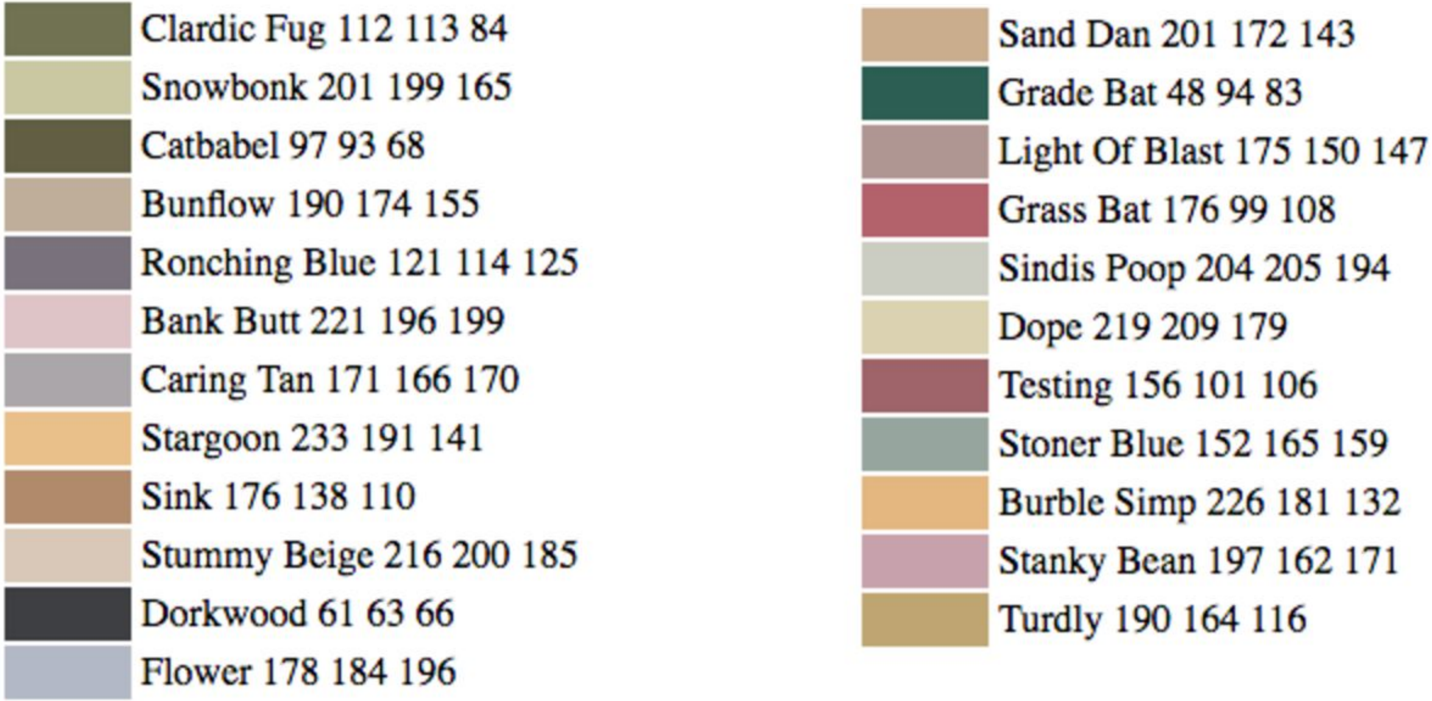
About how many layers does **Google Translate** use (as of 2016?)

- Ballpark… 1? 5? 100? 1000?

- “Our model consists of a deep LSTM network with **8** encoder and **8** decoder layers using attention and residual connections”

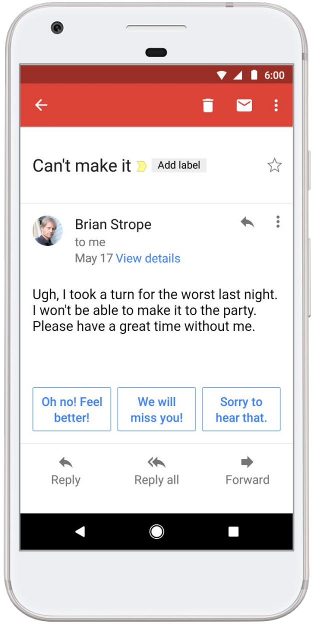
Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

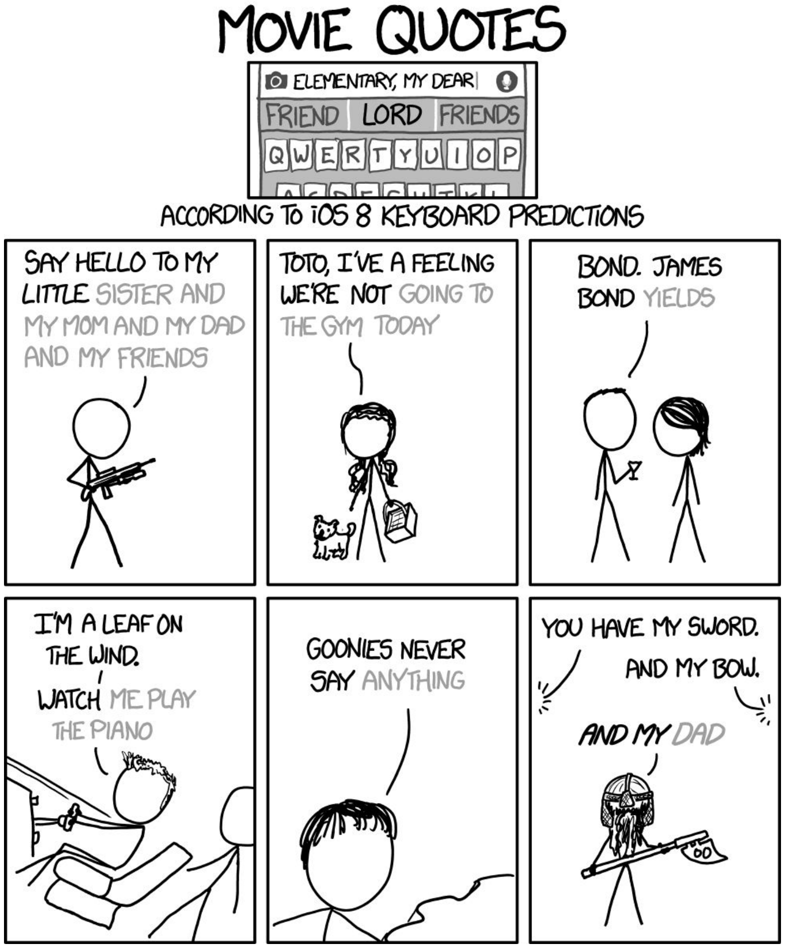
Text generation



[An AI invented a bunch of new paint colors that are hilariously wrong](#).

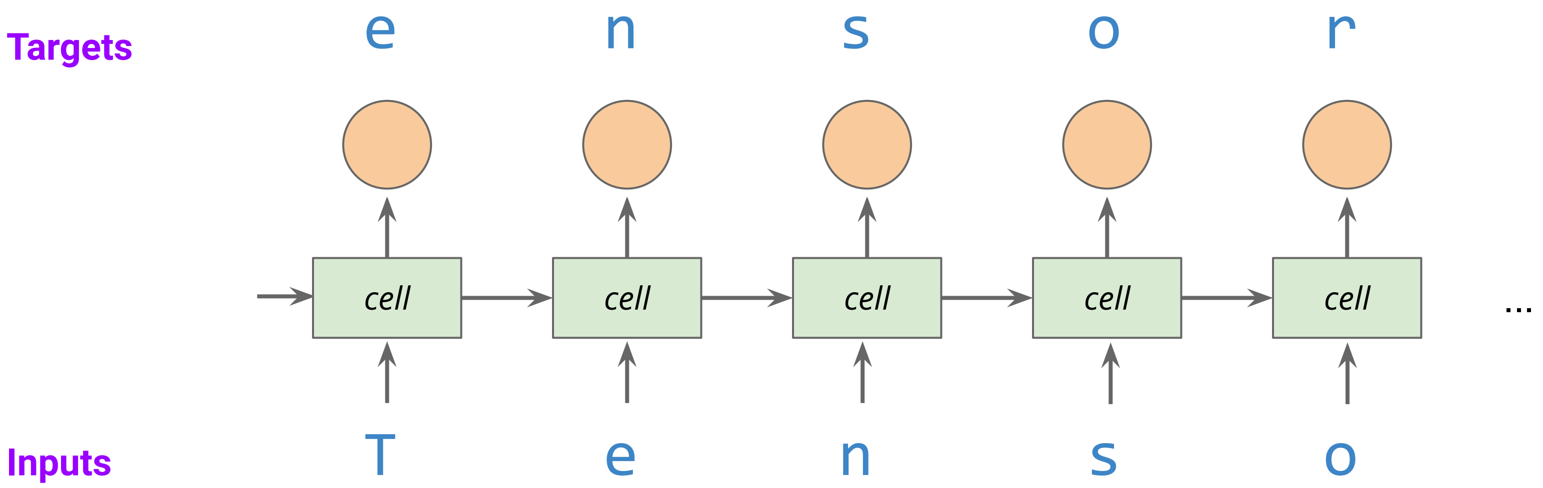
Text generation





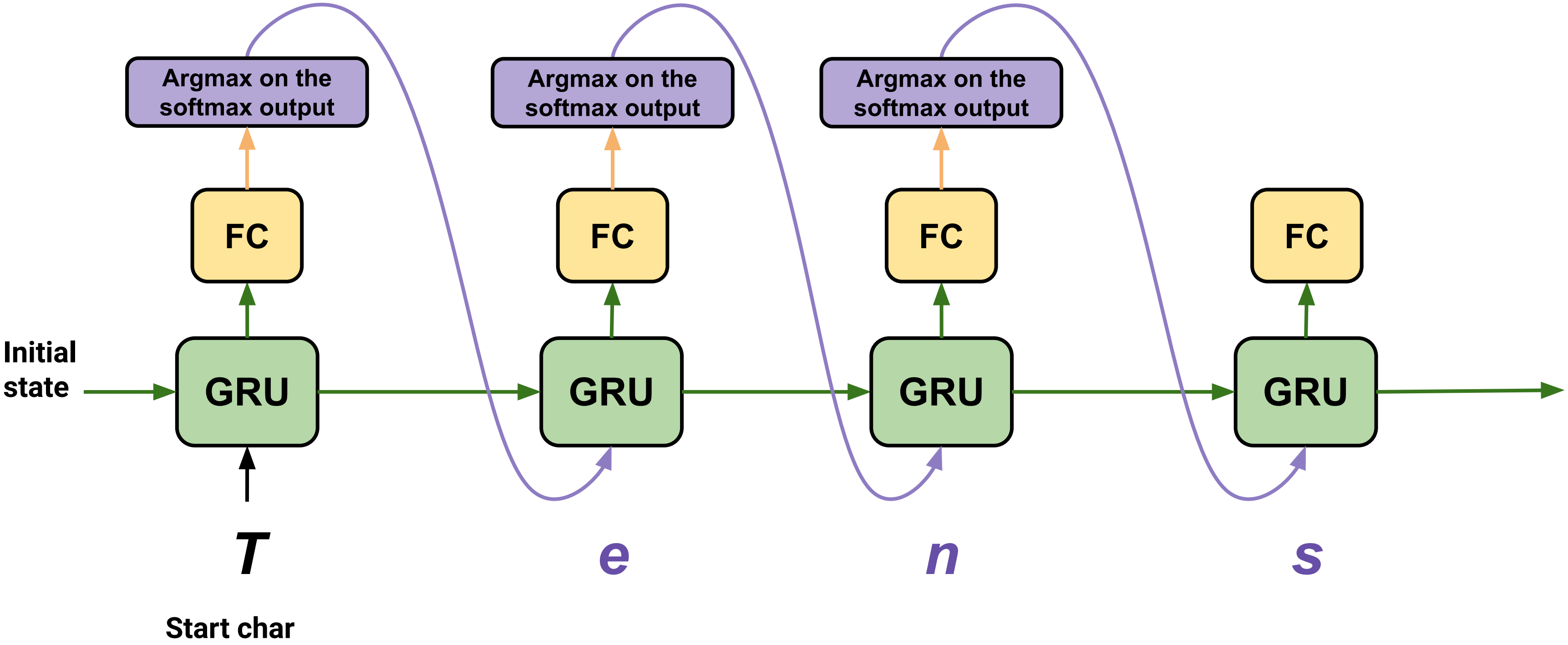
https://xkcd.com/1427/
https://ai.google/research/pubs/pub45189

Preparing training data: Tokenize, vectorize, divide into sequences (if generating free text, try 100 for a sequence length), then shift each character one to the right for each sequence to create a target.



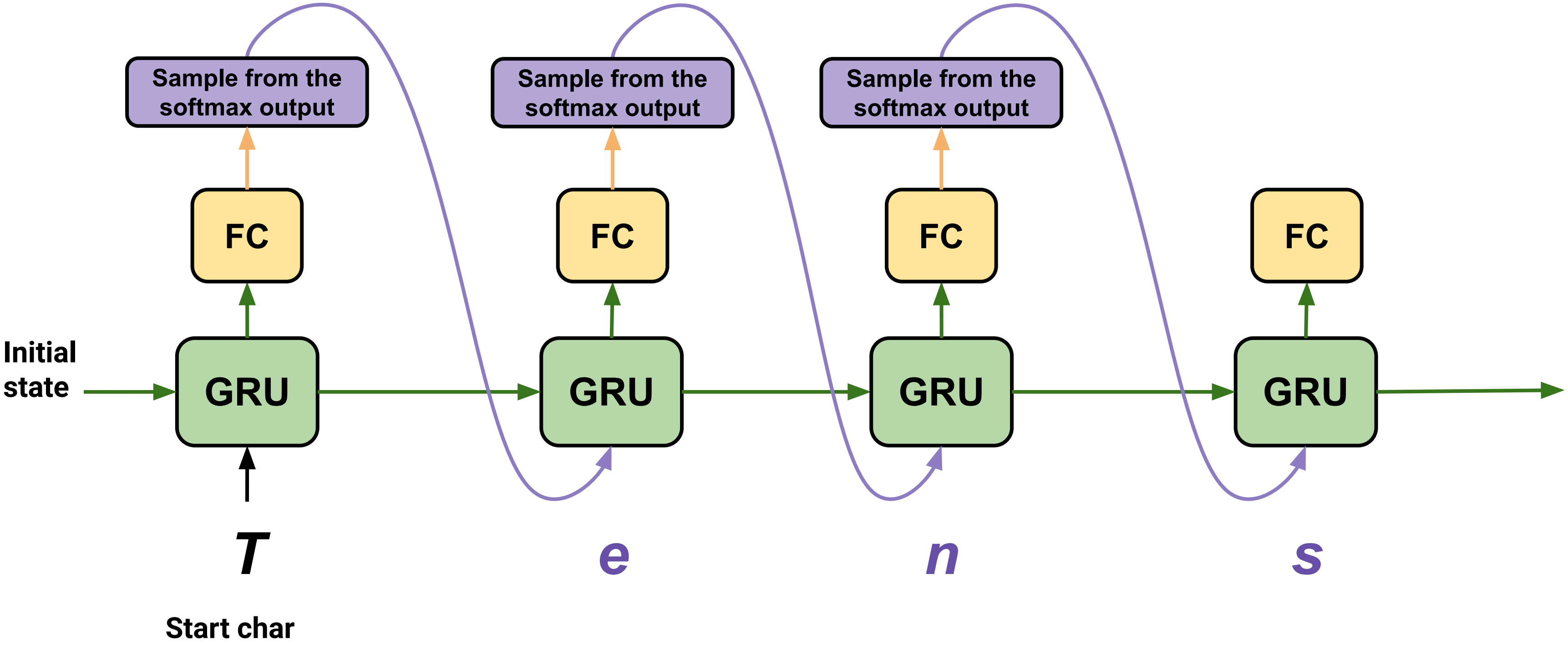
Sampling strategy 1

Problem: we'll generate the same sentence every time given a start character. No fun.

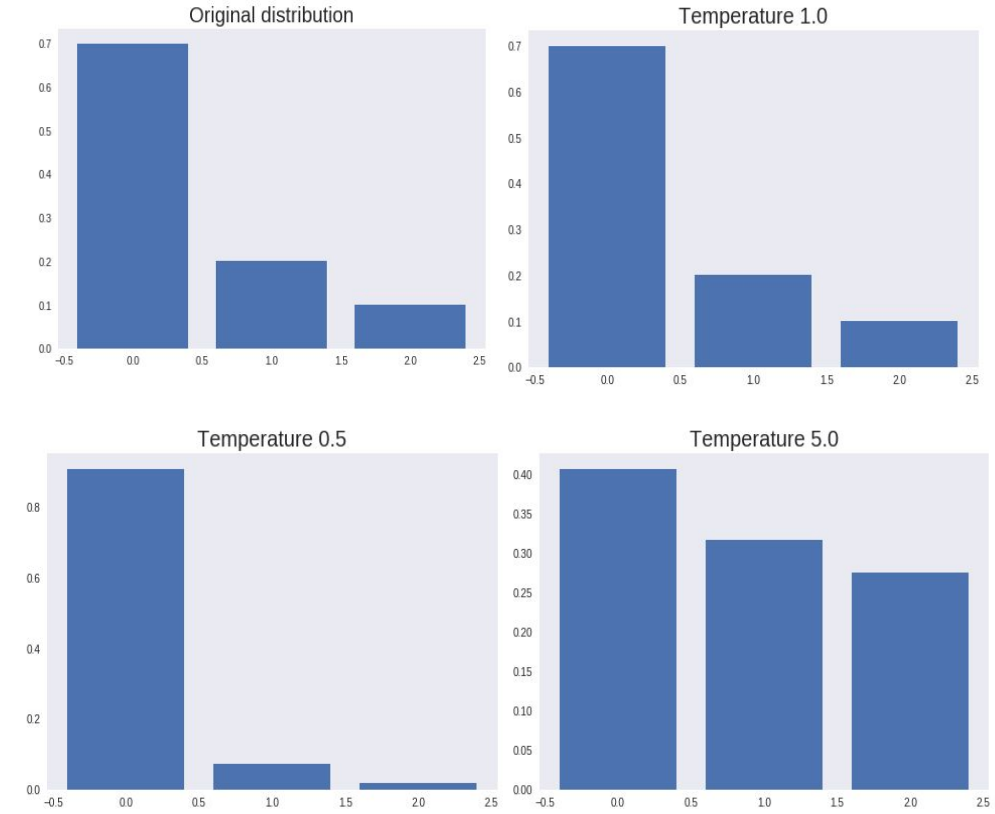


Sampling strategy 2

Instead, we can choose the next character with probability proportional to the softmax output. Leads to more surprising text, but, we have no control over how surprisingly we want the output to be.



Temperature



A bit of math to reweight the softmax output before sampling the next character.

Higher temperatures result in more surprising text, lower temperatures in more predictable text. Experiment in practice to find the "best" value.

```
import numpy as np
def reweight(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    return preds
softmax_output = [0.7, 0.2, 0.1]
reweight(softmax_output)
```

Effect of high and low temperature

Temperature 0.0001

QUEEENES:
The more than the more of the more than a man that he would have meet me to the more than a man that he would have meet me

Repeating

Temperature 5.0

QUuNs,LOUuw?ahf-ciy.
bYoPiTcyFOF
Y:!I-ShQH3OLR:
'luzoprO.!
NeeoxksIiJCt;-kiUUMNNY&FWlio?
haxTWaJh:
DMyf loesur?NAkvuslrox

Effect of high and low temperature

Temperature 1.0

BIANCA:
Fol, lead; he may drum!
Wear-bloos here, that where she buses
To that shampered as I am here?

Temperature 0.5

ARIEL:
I am the trumpet and soldiers have a not to the greater,
The grieve the queen should have too remember for the more.

Experiment to find "the best" value in practice.

Start sequence for all of these is Q.

Break

Try the text generation tutorial, it's great.

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.1-text-generation-with-lstm.ipynb

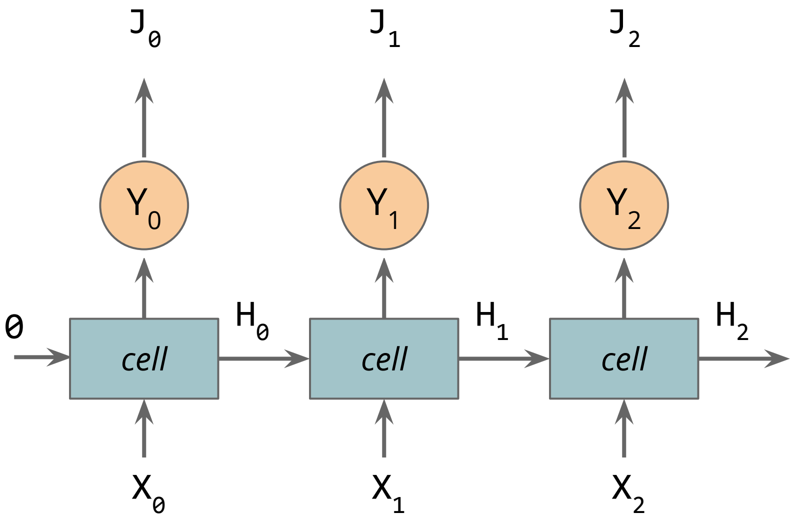
FYI, Convolution works well for sequences, too.

```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense, Embedding, Conv1D, MaxPooling1D, GlobalMaxPooling1D
from keras import Sequential
max_features = 10000
max_len = 500
(x_train, y_train), (_, _) = imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=max_len)


model = Sequential()
model.add(Embedding(max_features, 128, input_length=max_len))
model.add(Conv1D(32, 7, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Conv1D(32, 7, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10)
```

Extras

Backprop through time



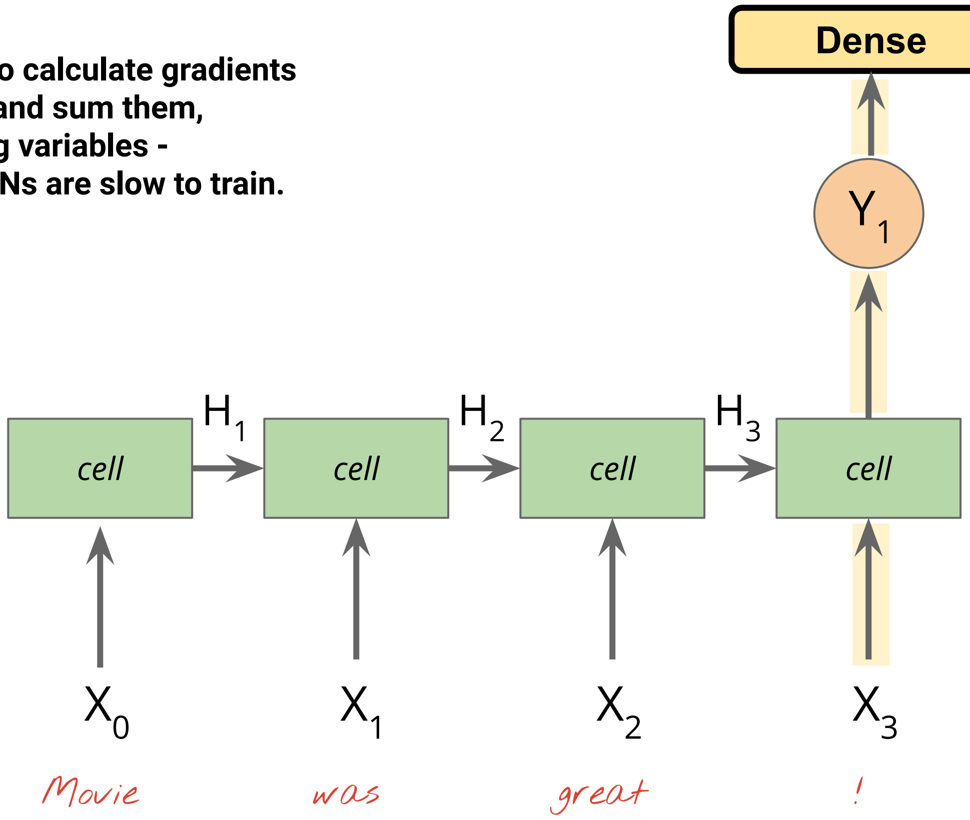
Recall: backprop

- **Find the gradient of the loss w.r.t. each weight**
- **Nudge weights in opposite directions to minimize loss.**

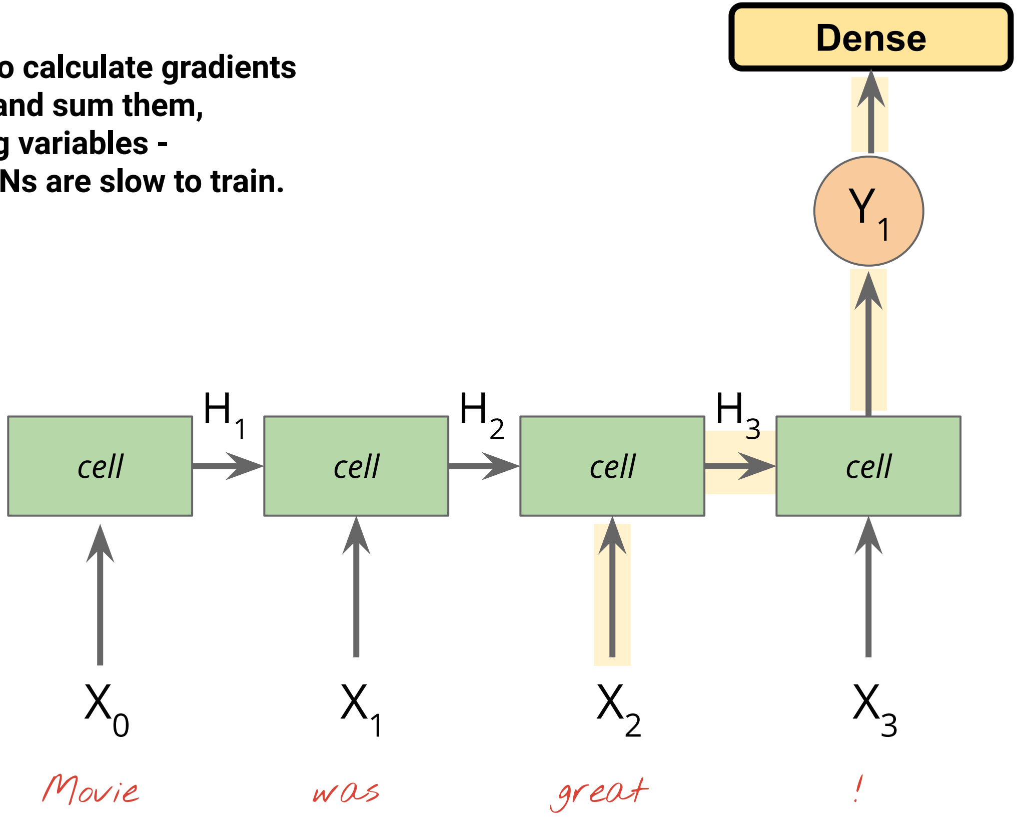
With RNNs, we can have an output at every timestep, so we can have a loss at every timestep.

- **Total loss = sum of losses at each timestep.**
- **Total gradient = sum of gradients for each weight across all timesteps.**

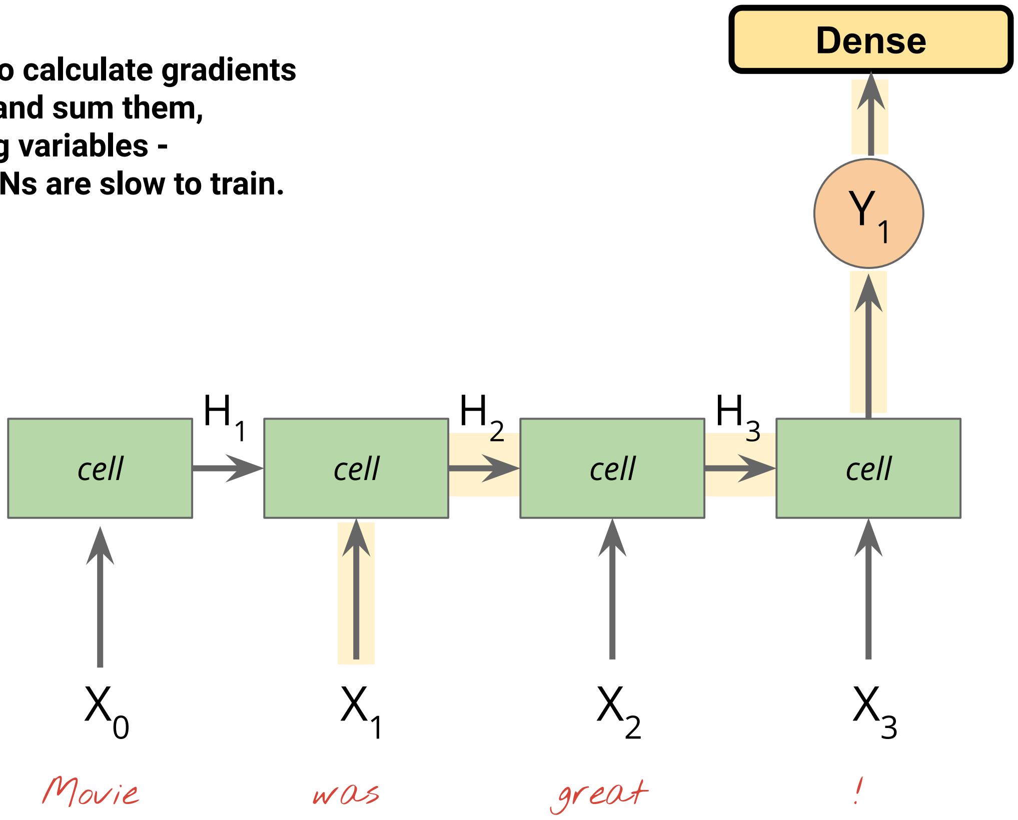
Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.



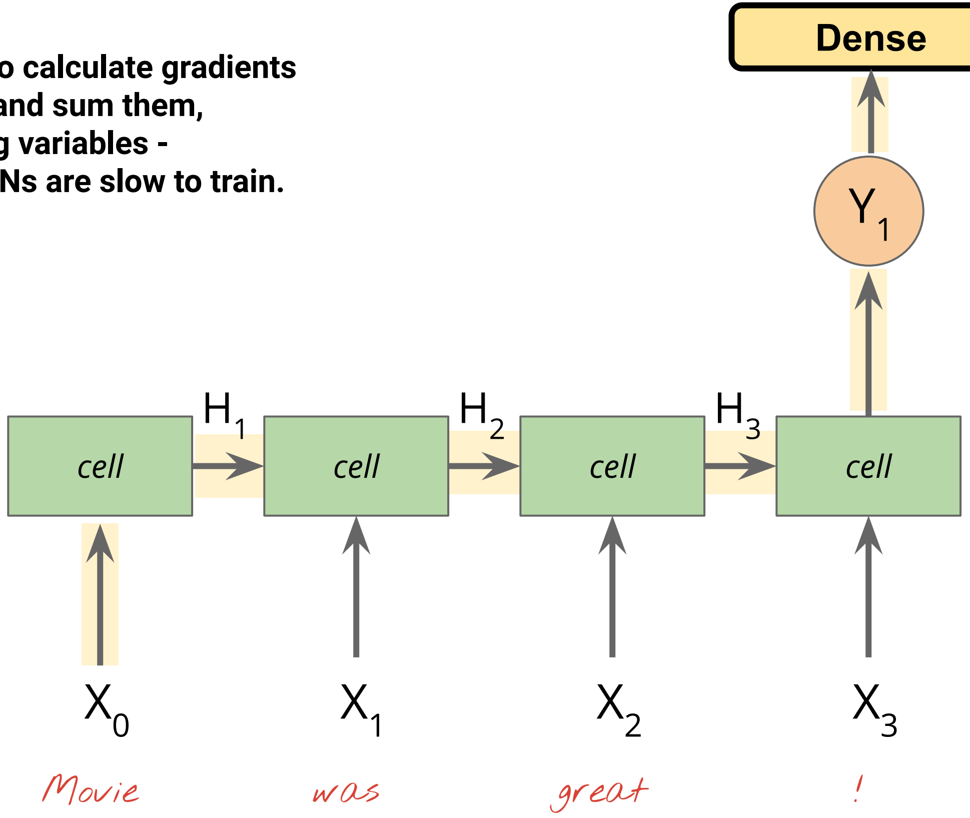
Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.



Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.

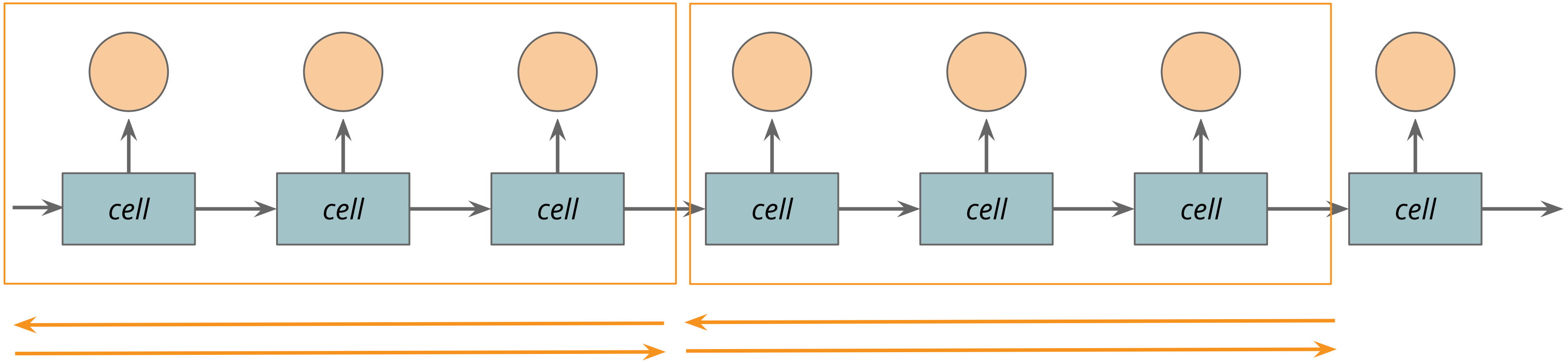


Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.



Truncated backprop through time

Imagine unrolling a network over a long sequence. Problem #1 efficiency.



Solution: divide sequence into batches. Forward and backward one batch at a time. Noisy, but efficient, like SGD. (Hidden state propagates forward between batches).

Reading

Practical examples

- DLP chapter 6 (RNNs)

Blog posts

- The Unreasonable Effectiveness of Recurrent Neural Networks

Papers

- Sequence to Sequence Learning with Neural Networks (2014)

Optional

- Deep Learning ch 10 (for a deeper dive into RNNs)