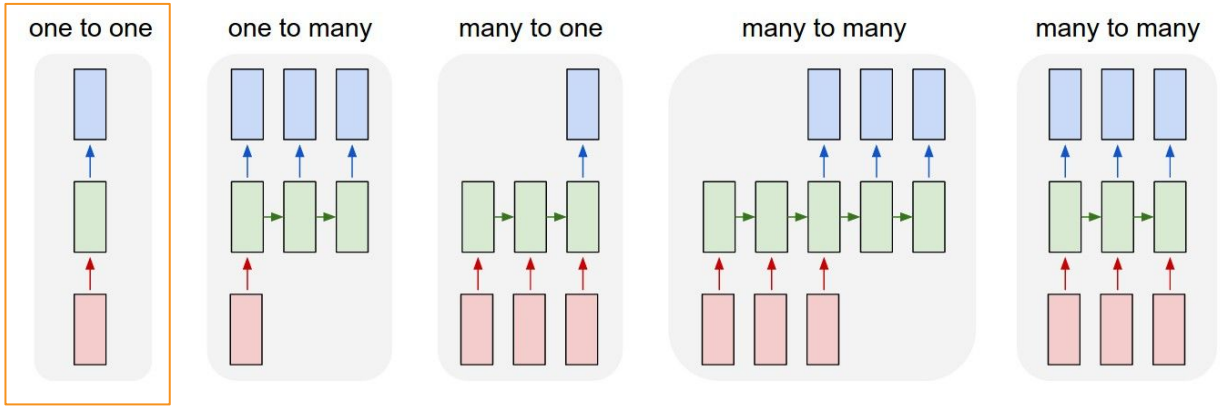
Applied Deep Learning

Lecture 8 • April 4th, 2019

Sequences



DNN: a tensor in, a tensor out (image classification).

Most examples in the following slides have complete code you can experiment with.

Diagram shamelessly borrowed from this excellent article.

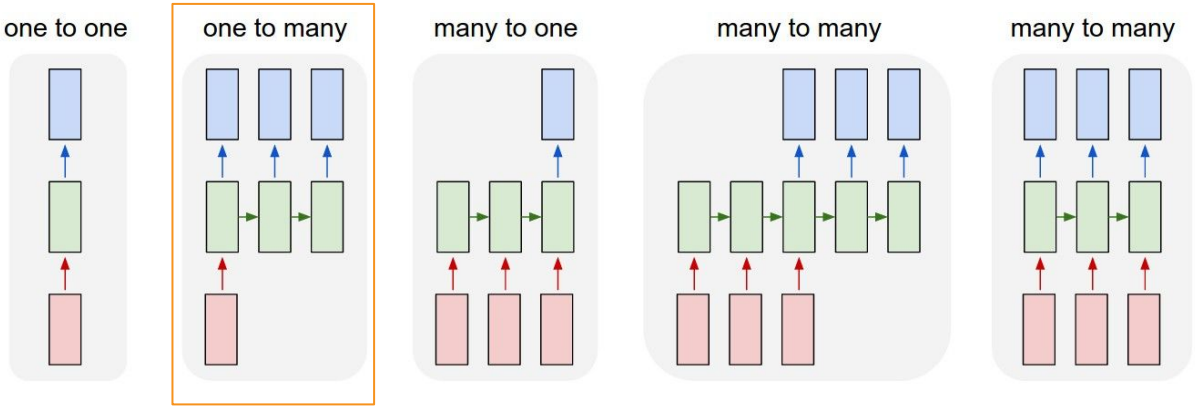


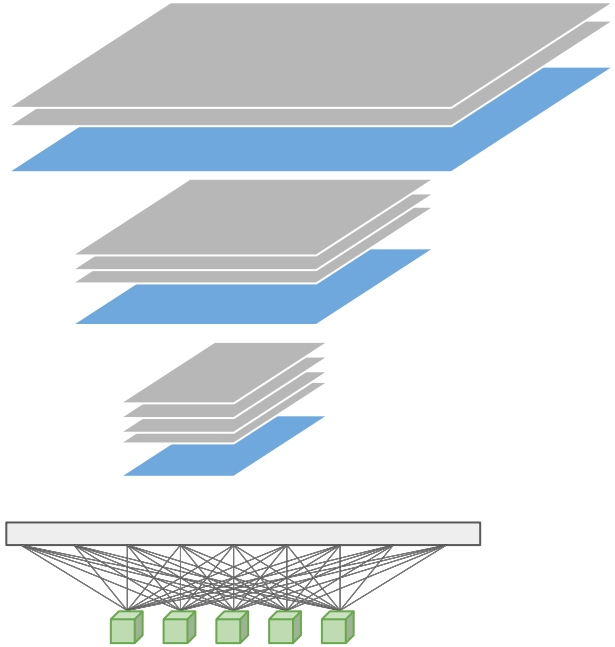
Image captioning

Complete code 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.

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.

"a surfer riding on a wave"



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!

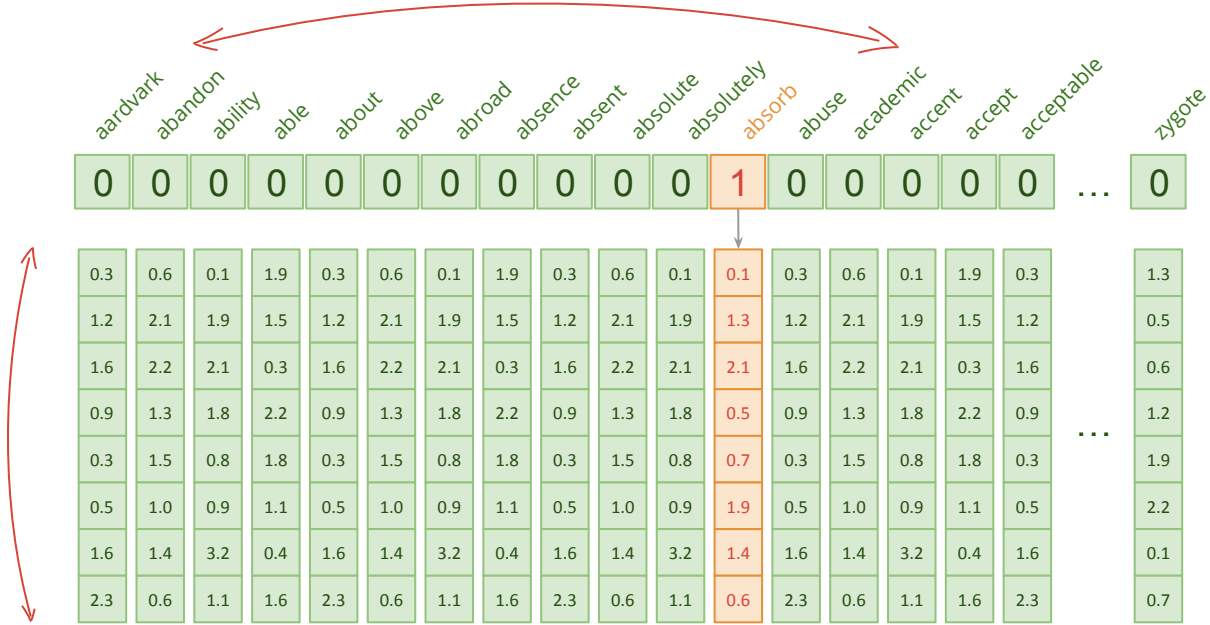
(How do we know which layer works best? Experiment).

one-hot encoding, 100,000 words

"Absorb"

=>

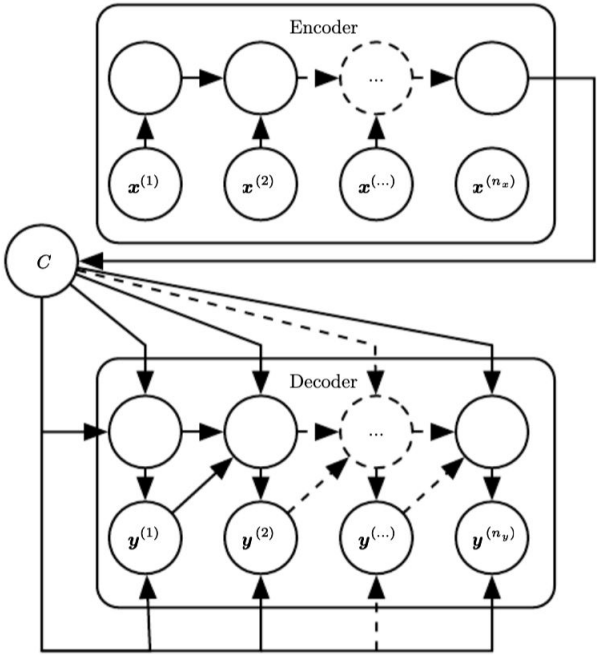
8-dimensional embedding



We've looked at embeddings before. This concept works for images (for captioning), sentences (for translation), and so on.

Encoder-Decoder Architectures

- Contrast with phrase-based SMT
- We can learn a vector that encodes the “meaning” of a sentence, or image, or video.
- We can learn an interlingual representation (multiple languages mapped to the same encoding).



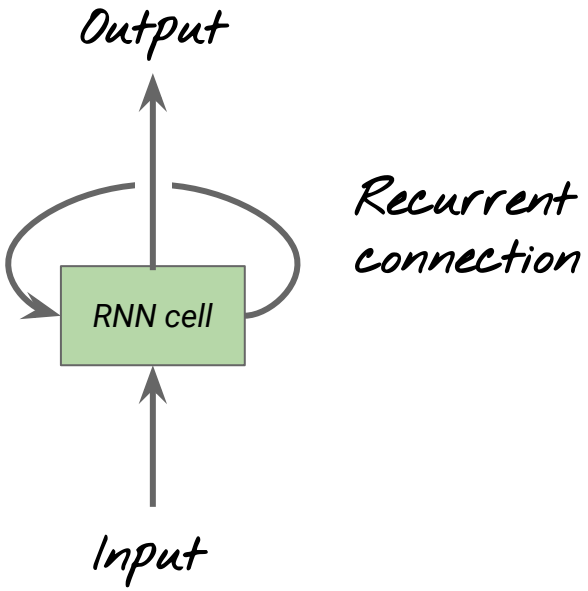
https://www.deeplearningbook.org/contents/rnn.html 10.4

How do we generate text?

Given that we know how to produce an image embedding.

RNNs

Graphical view of a “SimpleRNN”



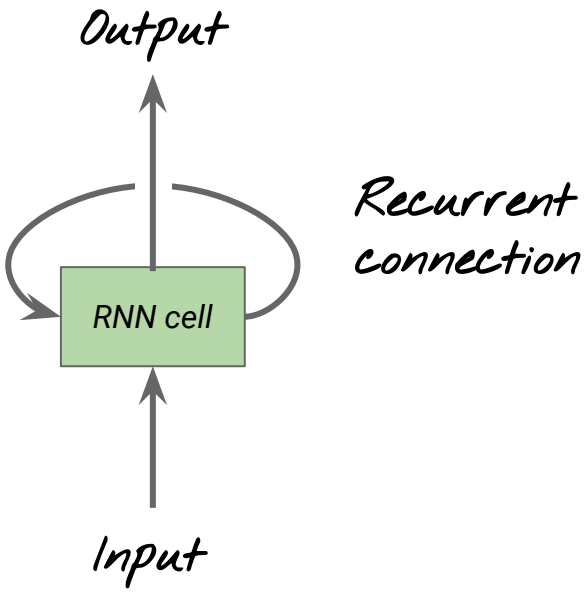
A “SimpleRNN” corresponds to fig 10.4 from Deep Learning, and the SimpleRNN class from Keras.

Different courses and books use slightly different designs for the first cell they introduce.

MIT uses this formulation, Stanford’s cs231 uses a “Vanilla RNN” that corresponds to fig 10.3 from Ian’s book.

Not a big deal, the core concepts are the same.

An API from a programmer's perspective



Recurrent connection

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

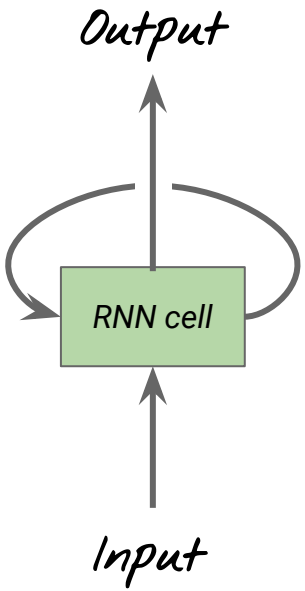
Instantiate a RNN.

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.

Good news: although there are many types of RNNs (of varying complexity), they all have the same API.

Process a sequence one element at a time



Recurrent connection

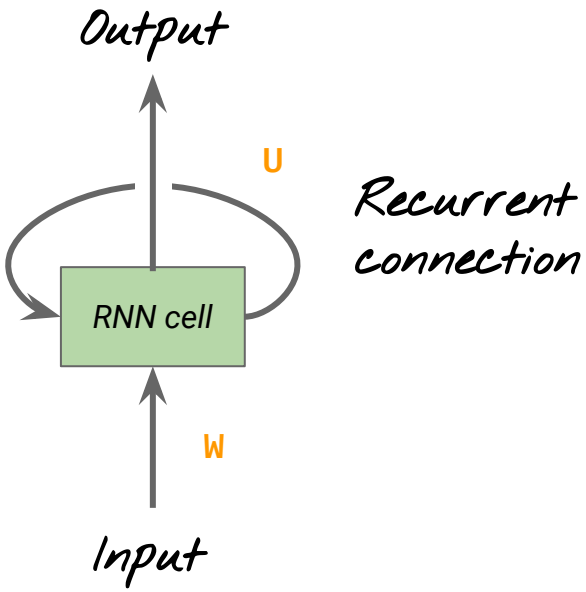
```
state = 0   Initialize the hidden state.
for input in input_sequence:
  output = f(input, state)
  state = output
```

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

This RNN simply remembers its previous state.

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

Process a sequence one element at a time

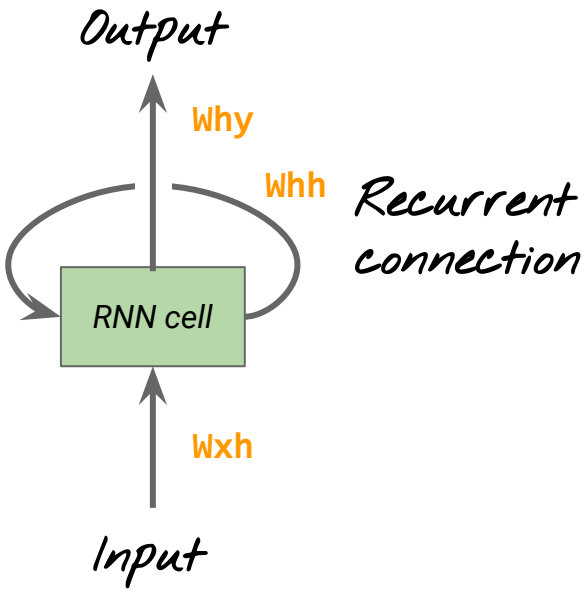


Recurrent connection

```
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



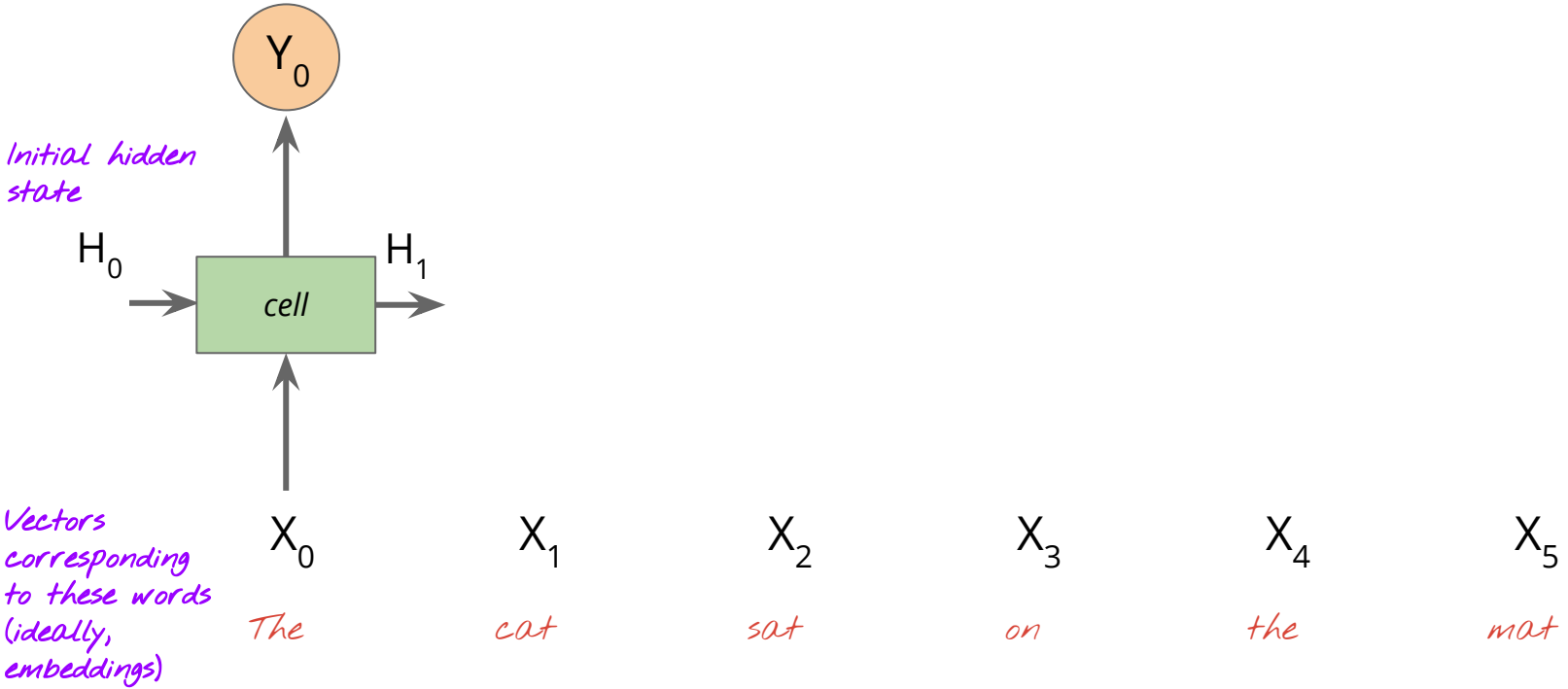

```
state = 0
for input in input_sequence:
  state = activation(dot(Wxh, input) +
                     dot(Whh, state) + b)
  output = dot(Why, state)
```

Three weight matrices now

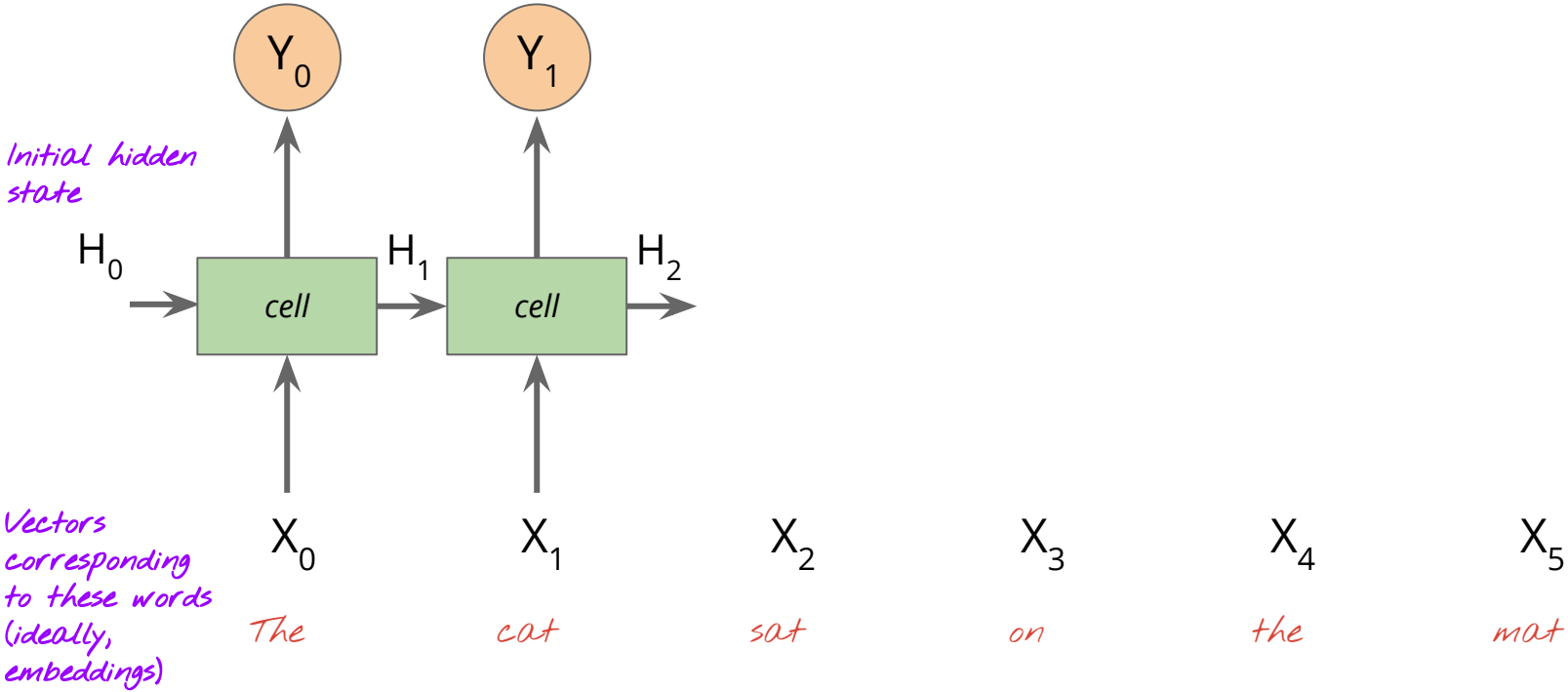
- *Input -> hidden*
- *Hidden -> hidden*
- *Hidden -> output*

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

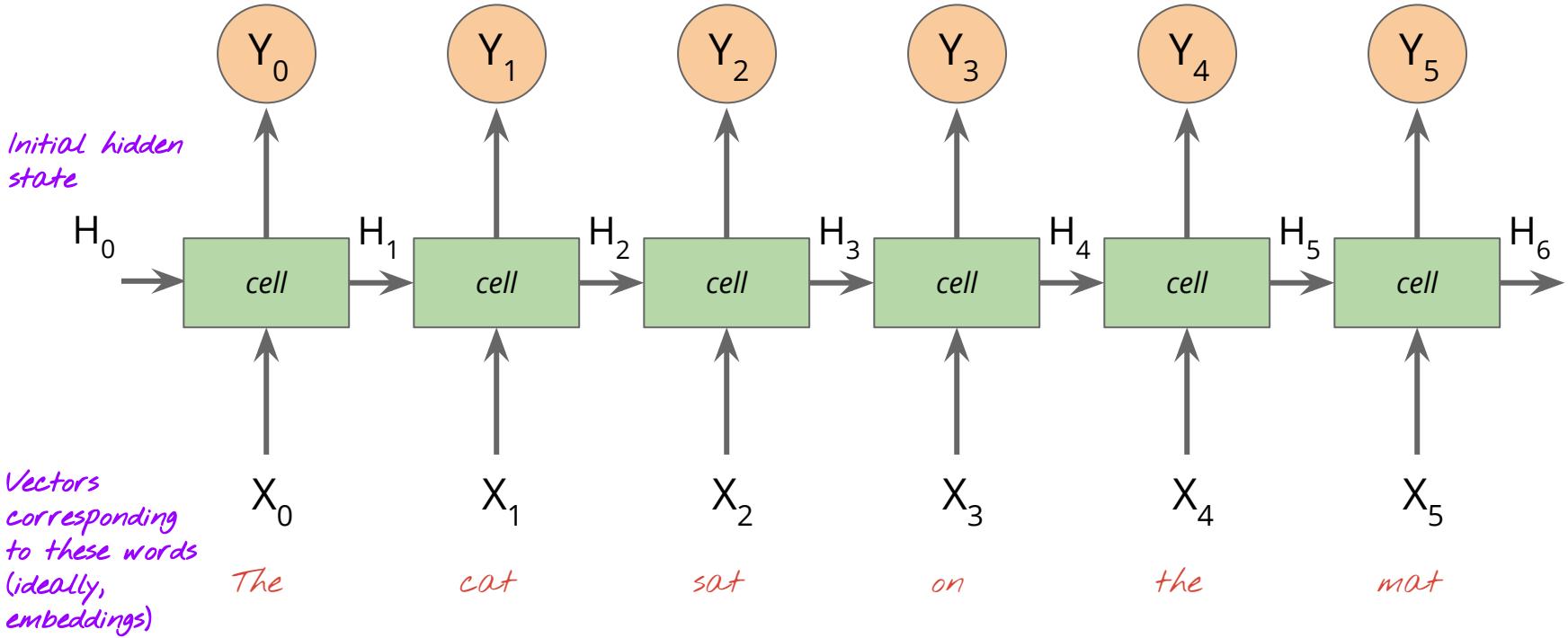
Unfolding


Y_0
Initial hidden state
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

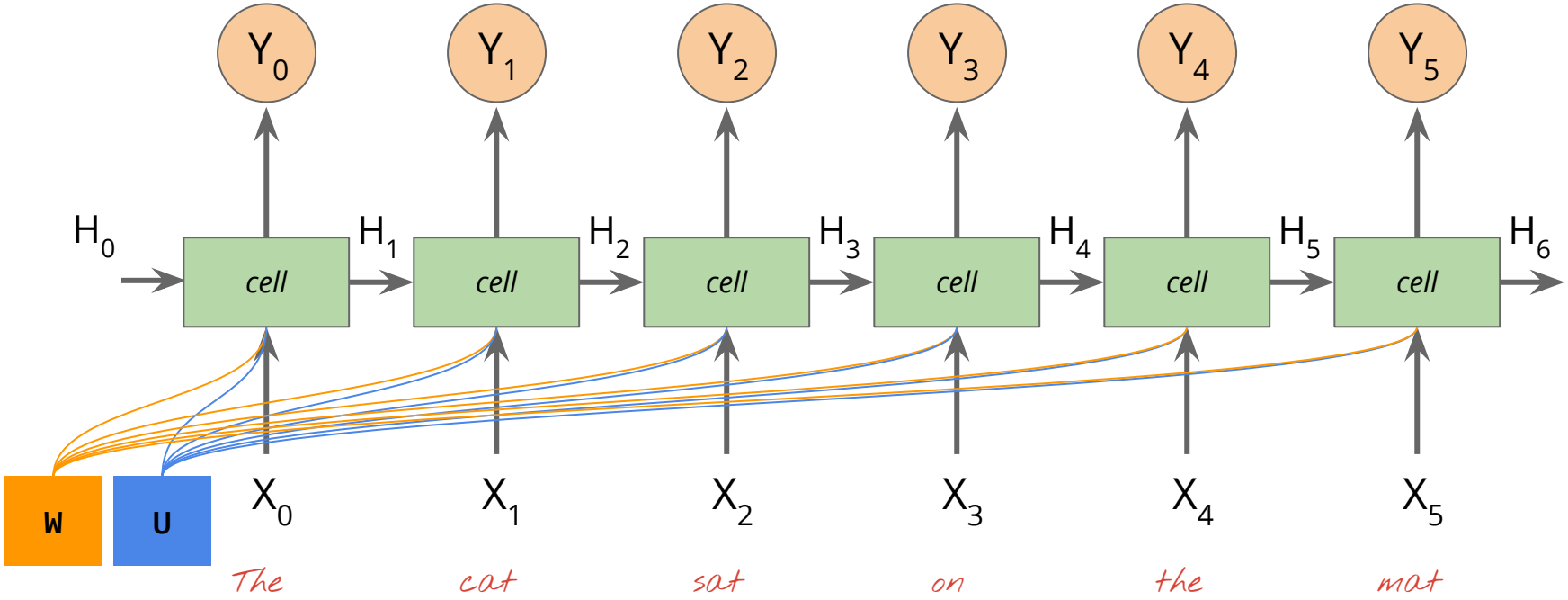

Y0
Y1
Initial hidden state
H0
H1
H2
cell
cell
Vectors corresponding to these words (ideally, embeddings)
X0
X1
X2
X3
X4
X5
The
cat
sat
on
the
mat

Unfolding


Y0
Y1
Y2
Y3
Y4
Y5
Initial hidden state
H0
H1
H2
H3
H4
H5
H6
cell
cell
cell
cell
cell
cell
Vectors corresponding to these words (ideally, embeddings)
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.

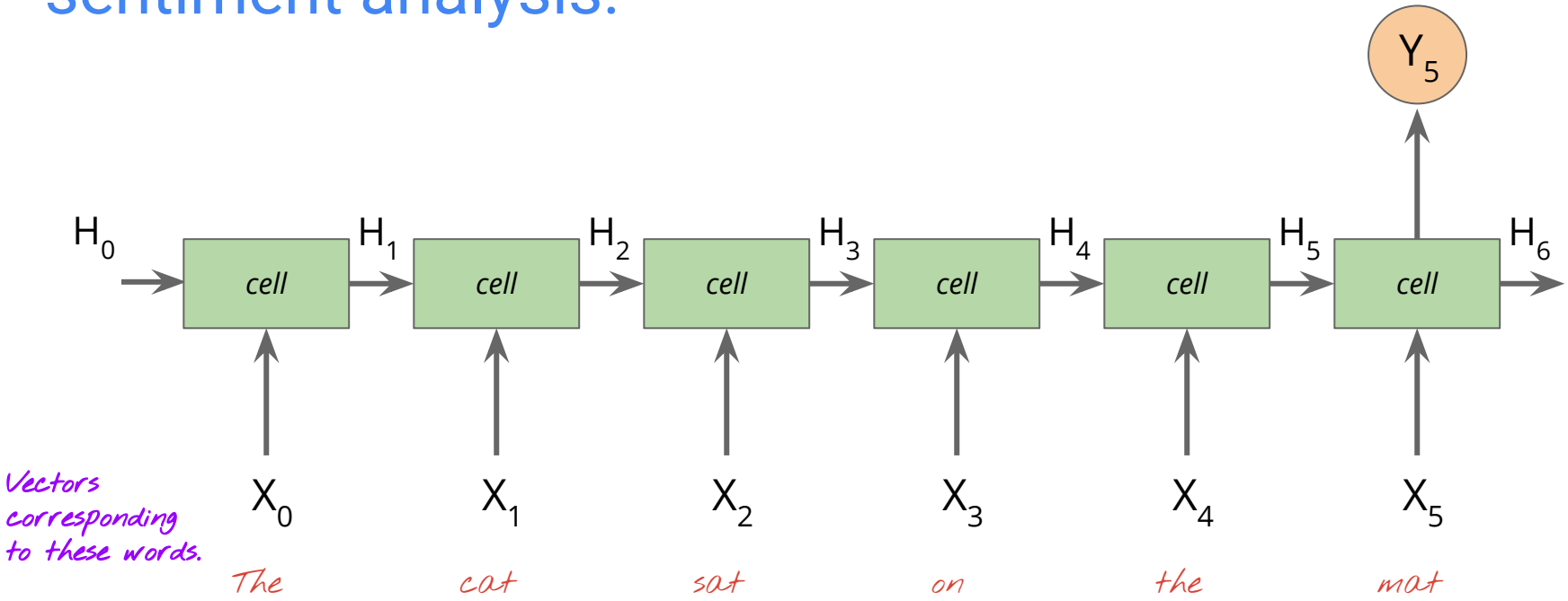


Similar idea to convolution

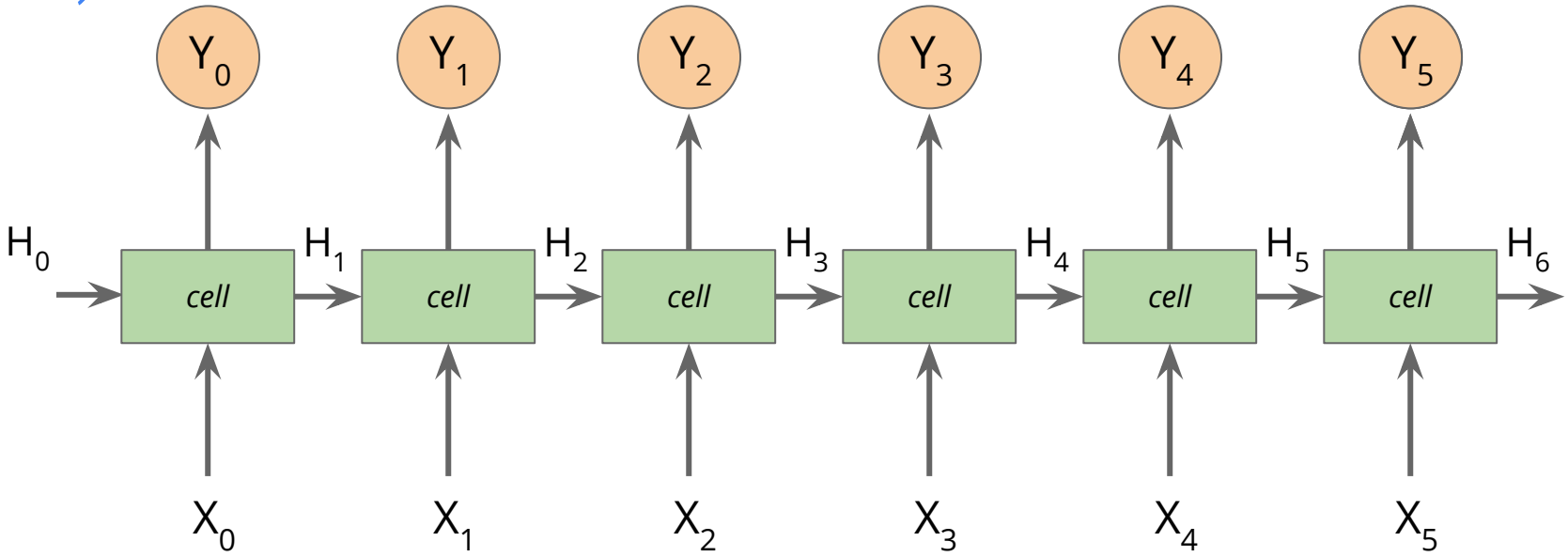
Weights for filters in a CNN are shared (e.g., using 9 weights, you can detect lines anywhere in an image).

Likewise, weights learned by an RNN can detect features anywhere in a sentence (e.g. the presence of a comma, or whether we are “inside” quotes).

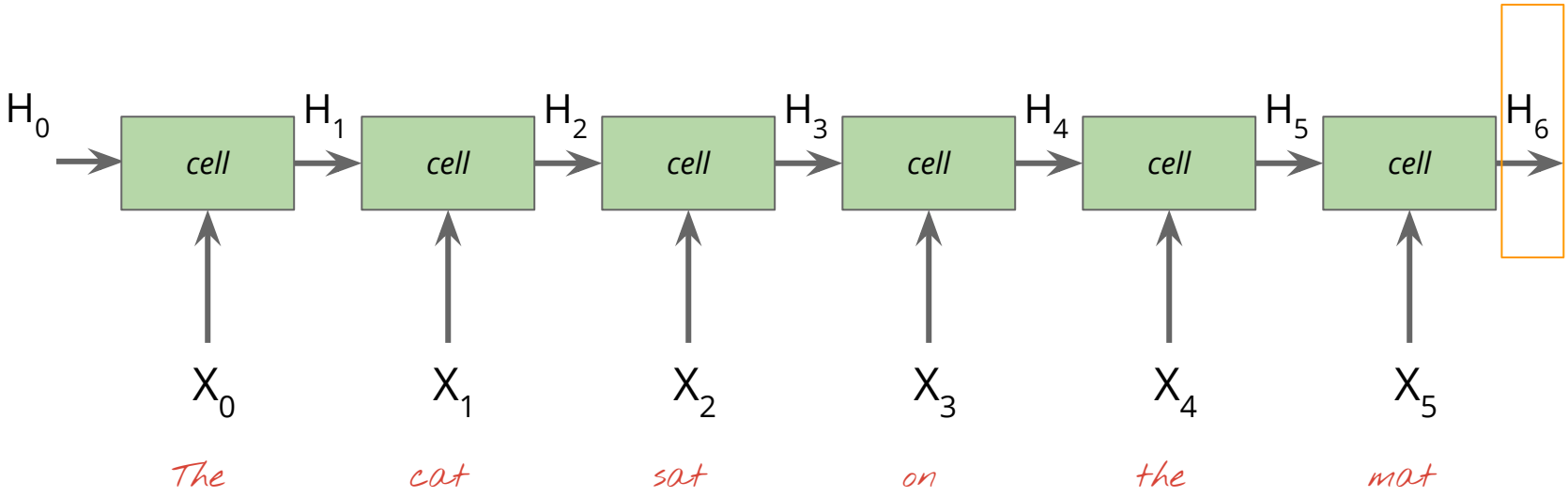
We may only be interested in the last output, say, in sentiment analysis.

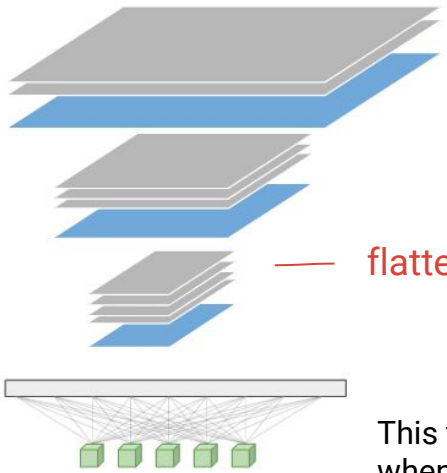


Or, outputs at every step (say, if generating a sequence of text).

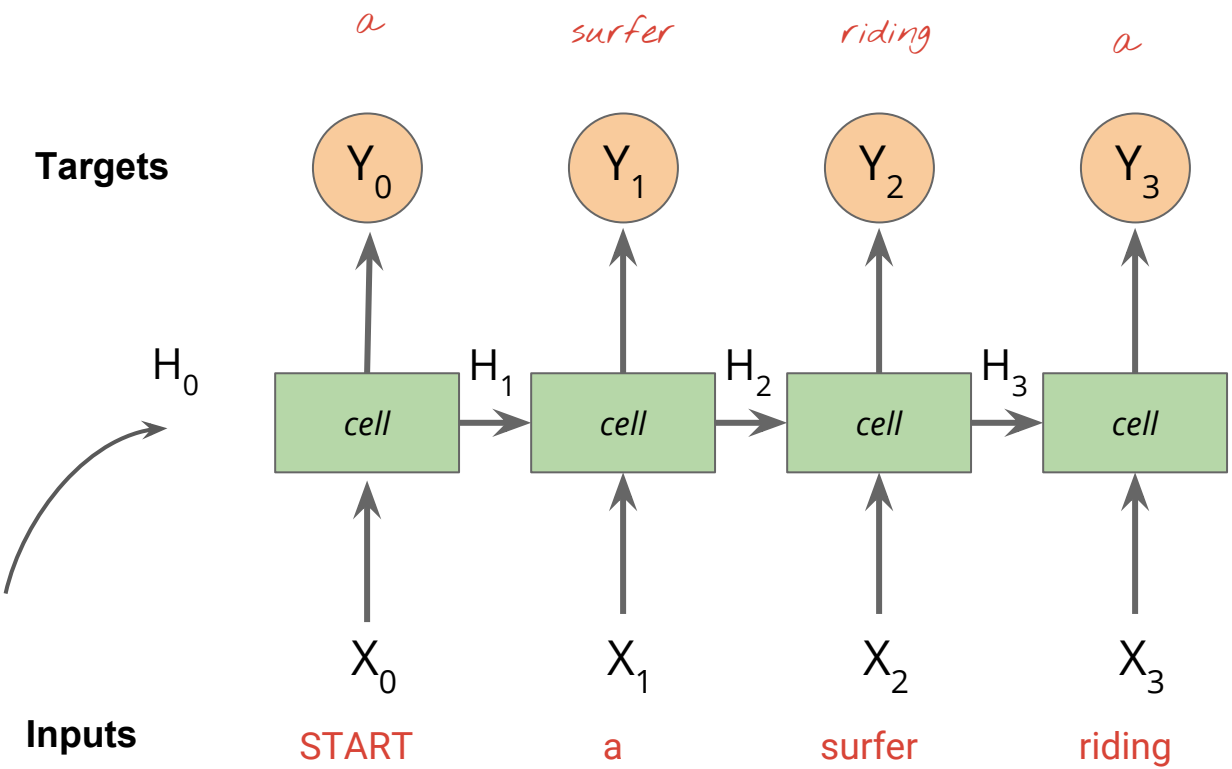


Or, in the final hidden state (say, if we want a representation of the sentence).



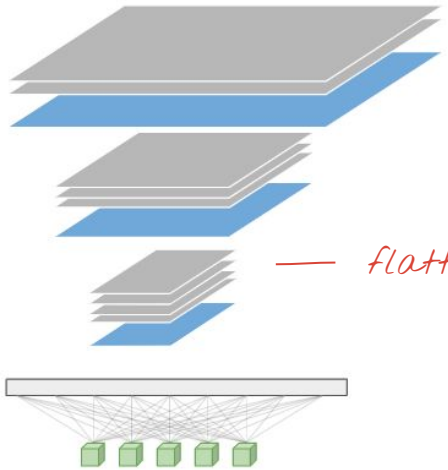


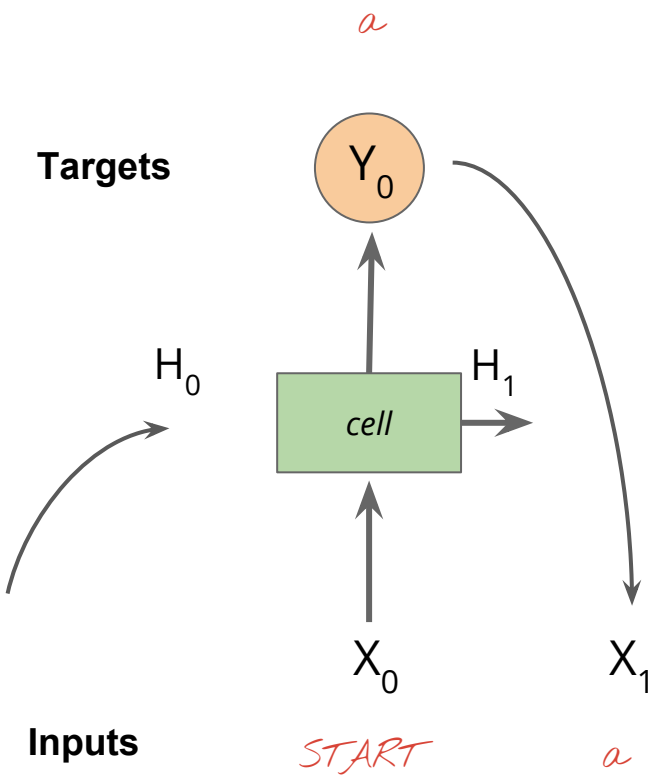
During training



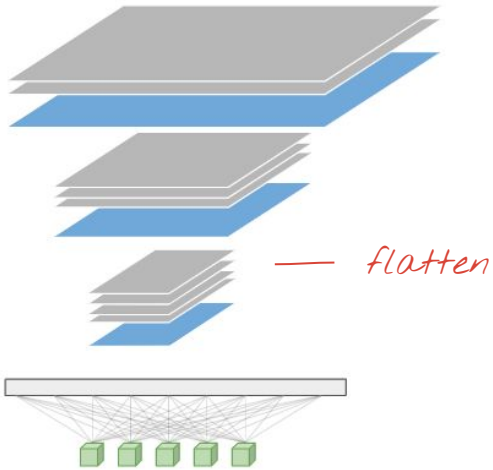
This technique is called "**teacher forcing**". We discard the actual output of the RNN (which is noisy when training), and instead feed the token it should have predicted as the next input. This can accelerate training.

During inference

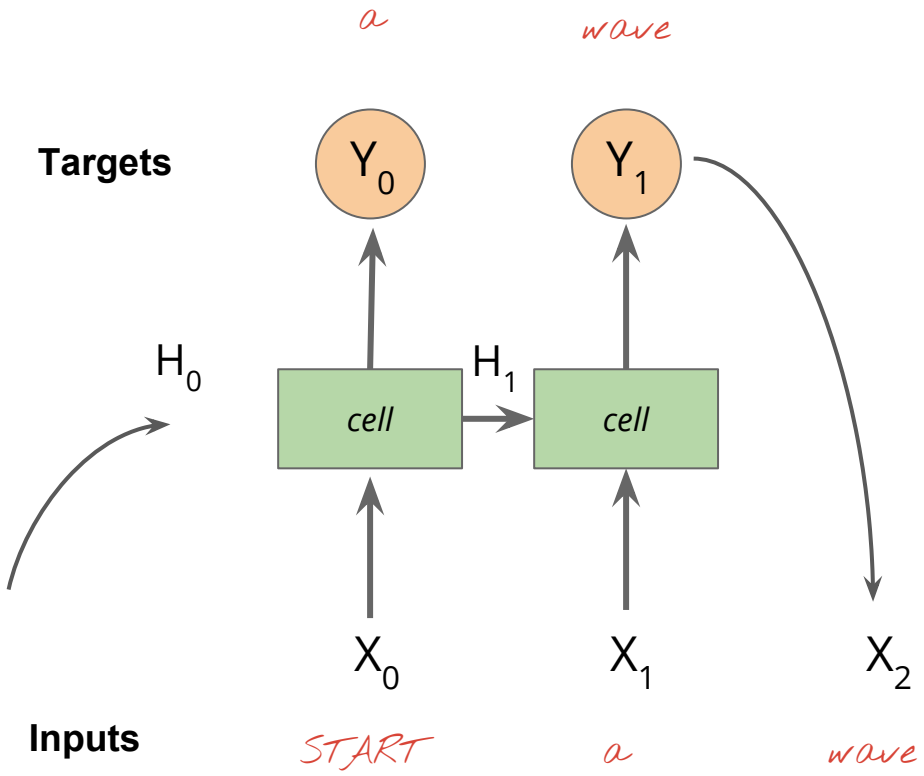


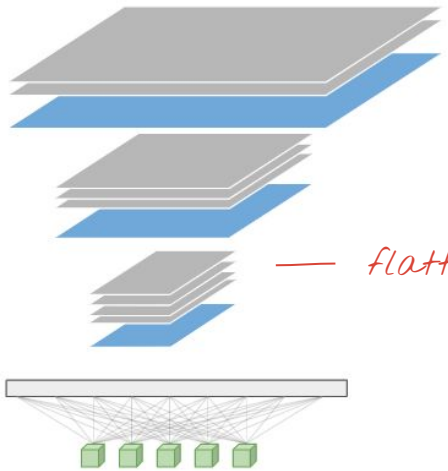


Sample from the output - more on this later. Feed that and in the cell state as inputs to the next step.



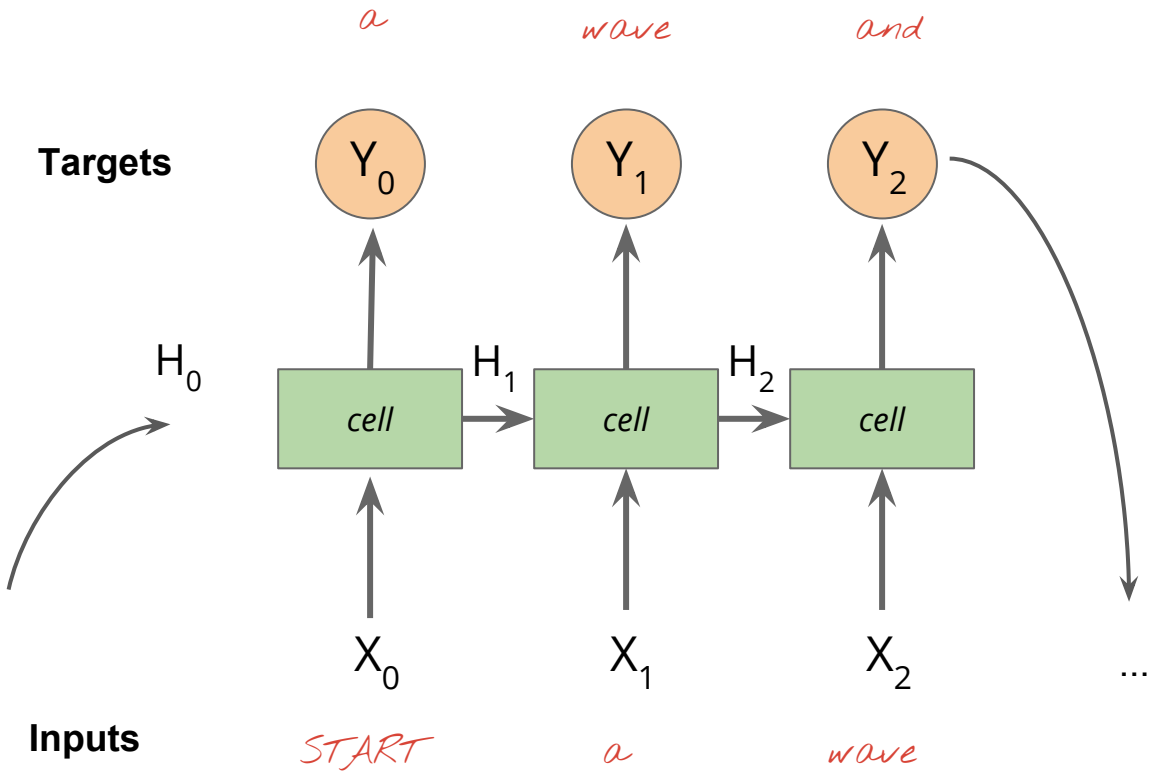
During inference





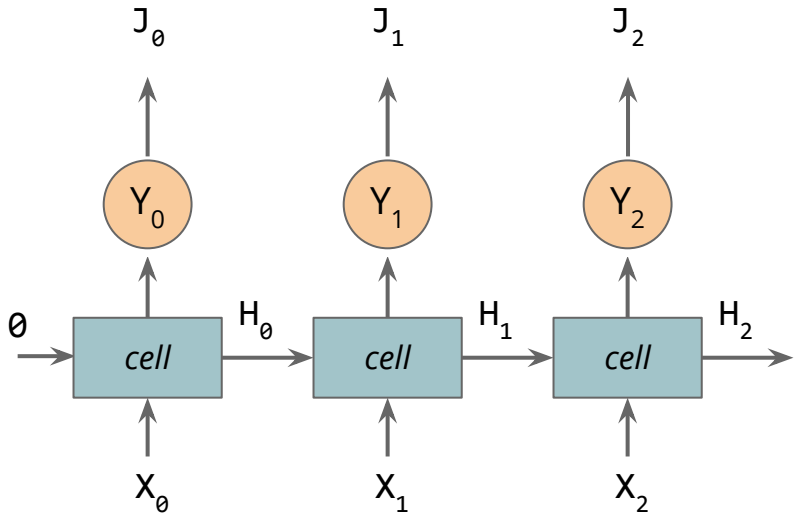
flatten

During inference



Sample until max_len or special END character reached.

Backprop through time



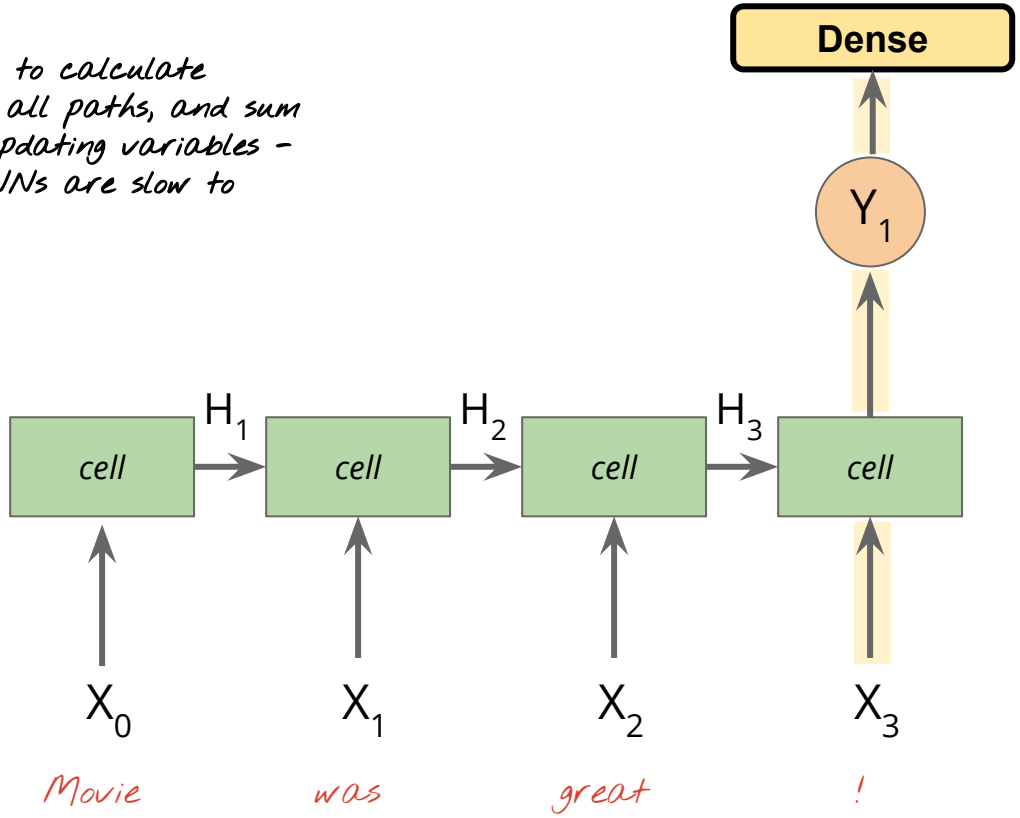
Recall: backprop

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

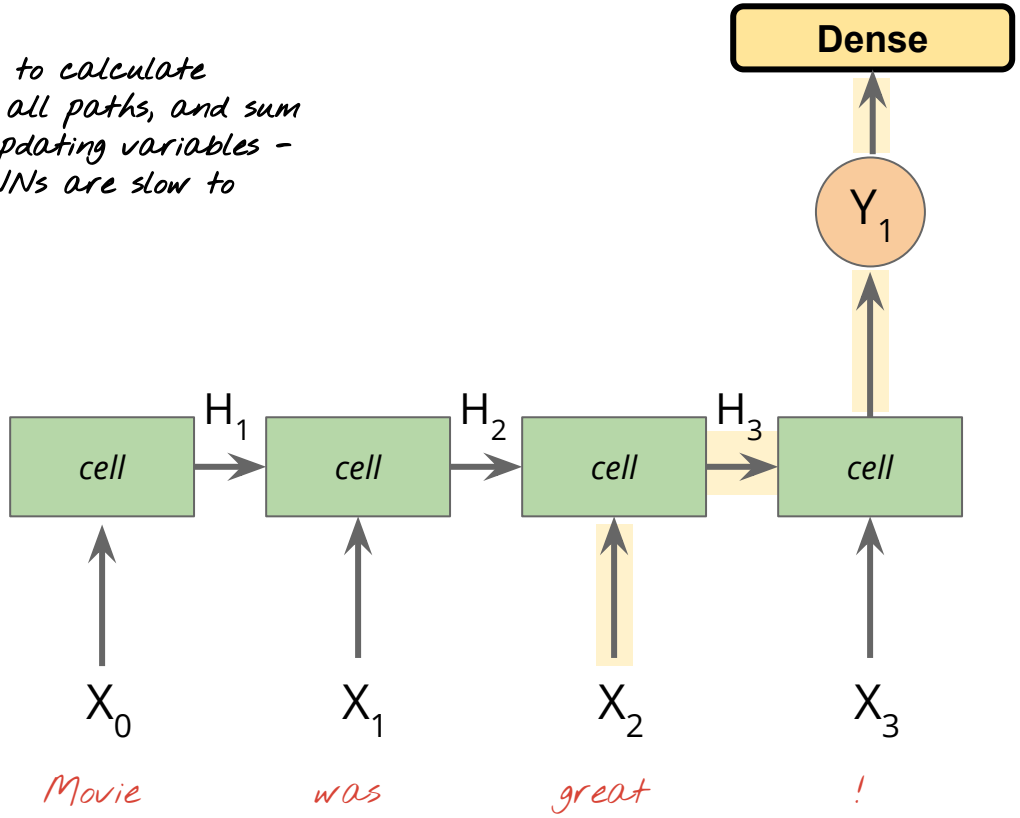
With RNNs, we can have an output at every timestep, so we can have a loss at every timestep.

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

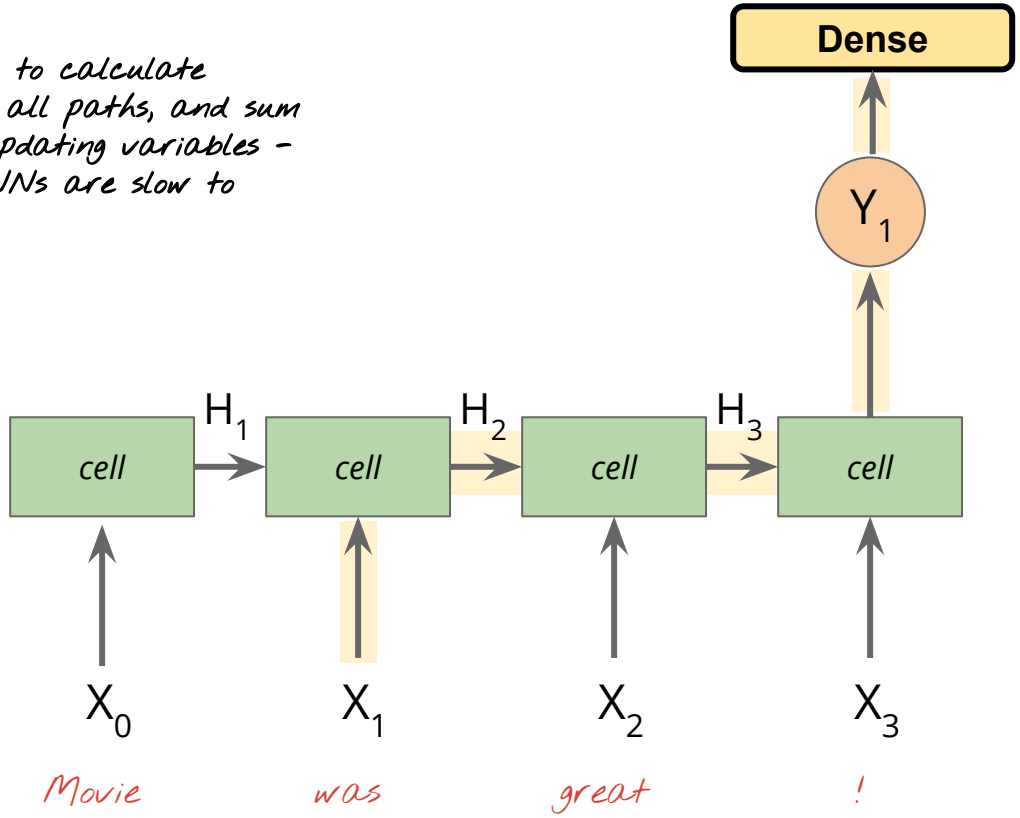
Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.



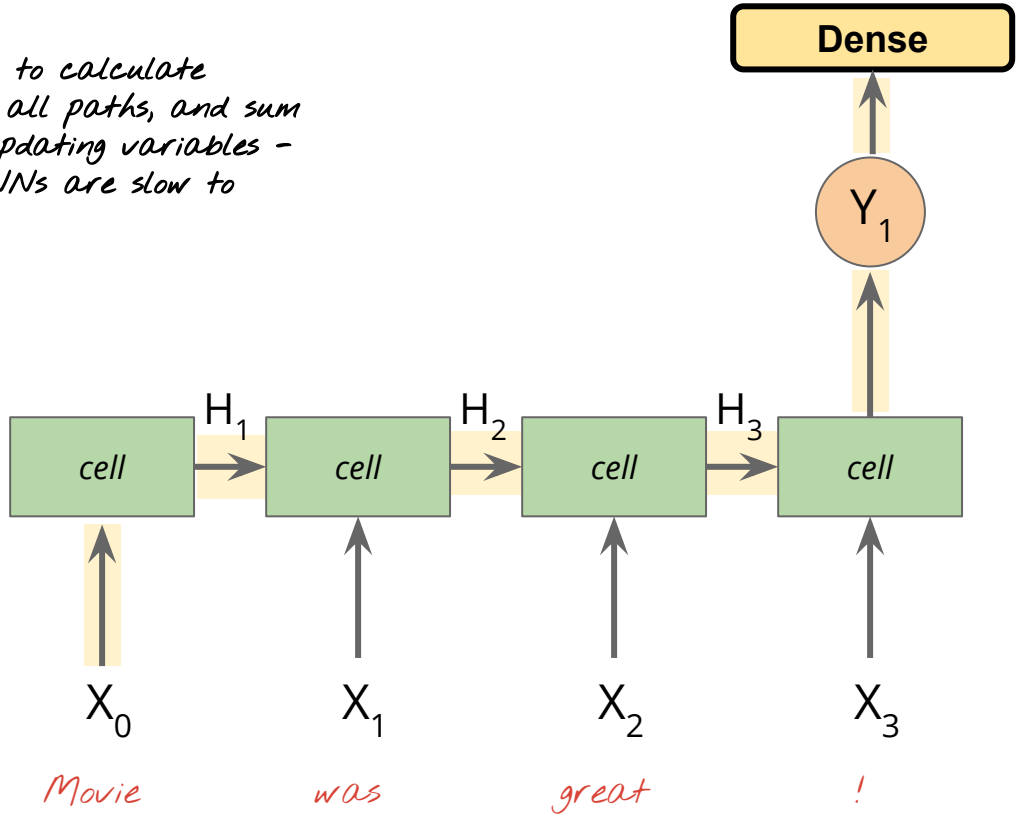
Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.



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

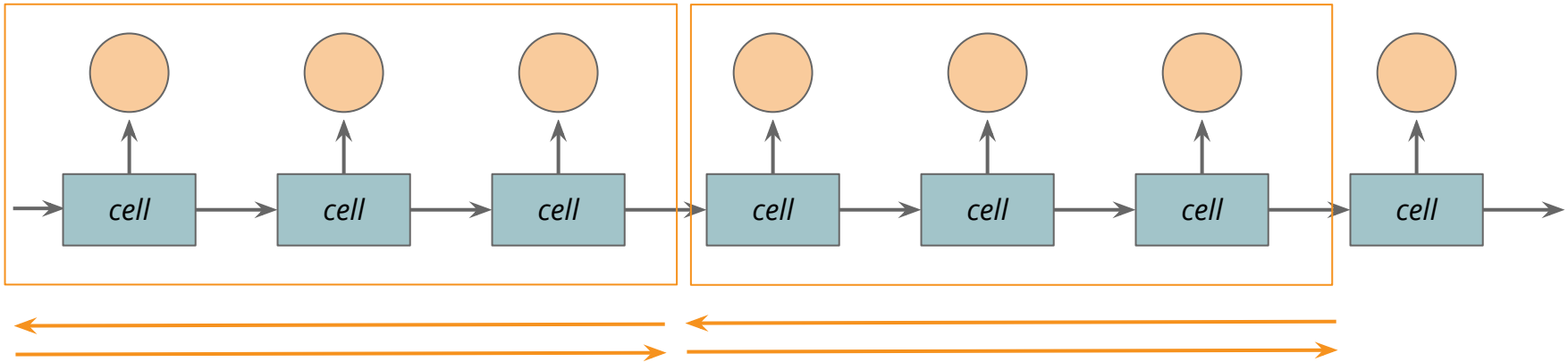


Backprop has to calculate gradients over all paths, and sum them, before updating variables - expensive - RNNs are slow to train.

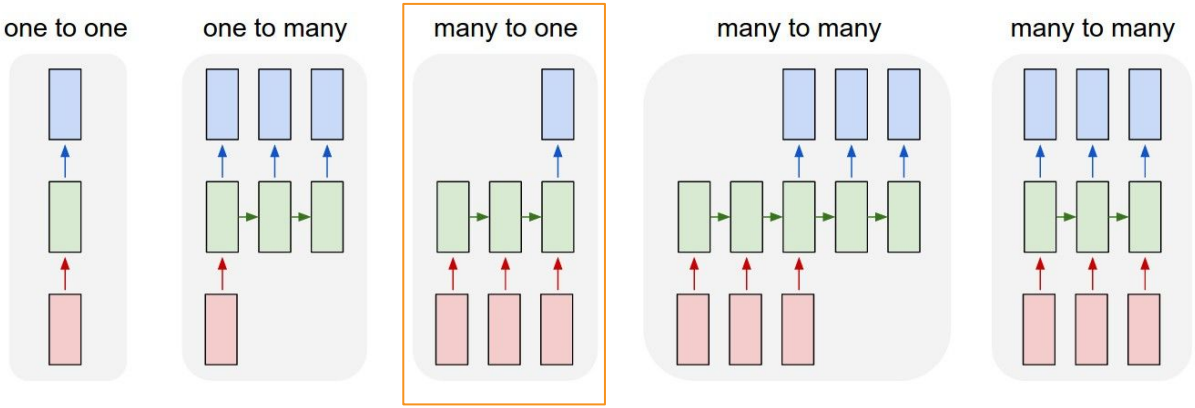


Truncated backprop through time

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

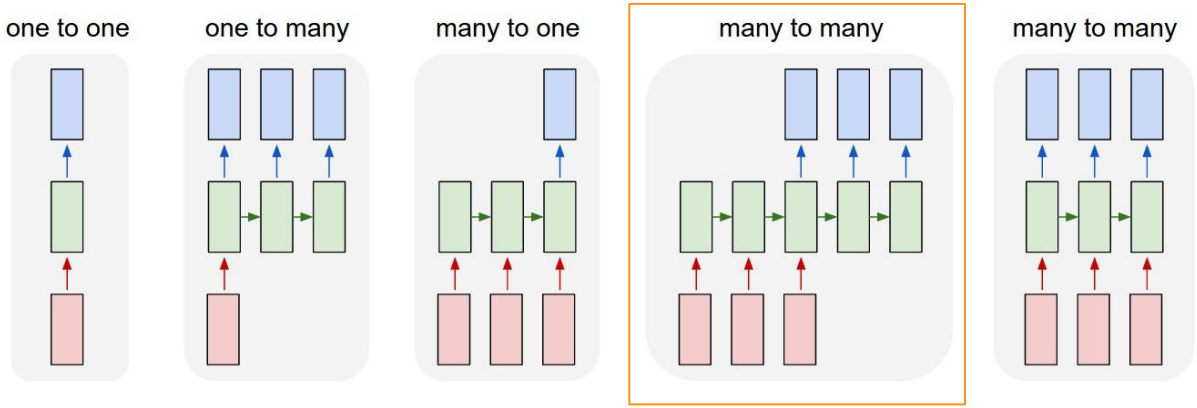


Solution: divide sequence into batches. Forward and backward one batch at a time. Noisy, but efficient, like SGD. (Hidden state propagates forward between batches).



Sentiment analysis, text classification, or timeseries classification / regression

Examples for all three are in Deep Learning with Python.



Machine translation

Complete code here.

Sequence to Sequence Learning with Neural Networks (for the core idea)

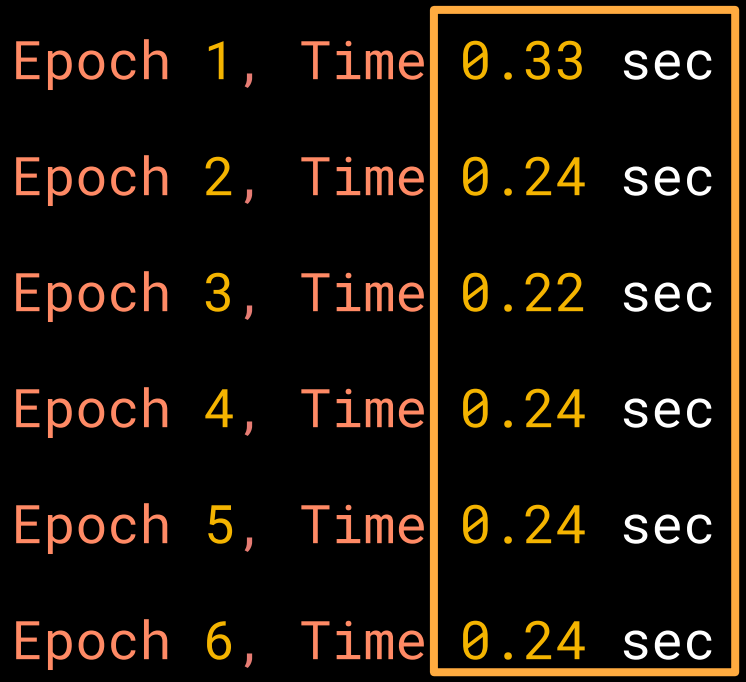
Code walkthrough

Lab: try it!

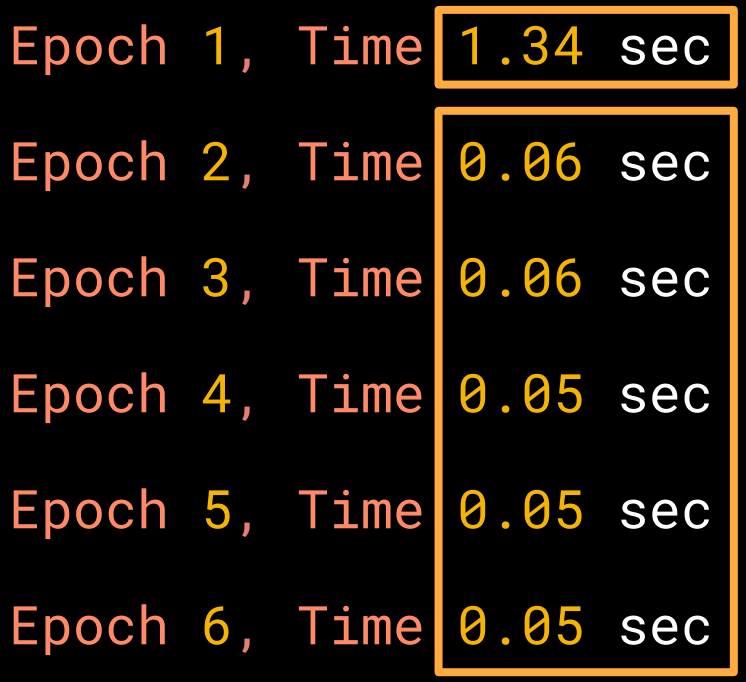
- The best way to understand this code it to go step by step, and focus on the shapes. E.g., what is the input shape to the encoder? What is the output shape? What do the different dimensions represent?

bit.ly/minimal-nmt

```
@tf.function
def train_step(source_seq, target_seq, target_labels, initial_state):
  with tf.GradientTape() as tape:
    encoder_output, encoder_state = encoder(source_seq, initial_state)
    logits, decoder_state = decoder(target_seq, encoder_state)
    loss = calc_loss(target_labels, logits)
  variables = encoder.trainable_variables + decoder.trainable_variables
  gradients = tape.gradient(loss, variables)
  optimizer.apply_gradients(zip(gradients, variables))
  return loss
```



Without @tf.function



With @tf.function (compiling on first epoch)

Okay! Break.

Then Object Detection guest lecture from Pratik (thank you!), then multilayer RNNs.

Demo

Object Detection

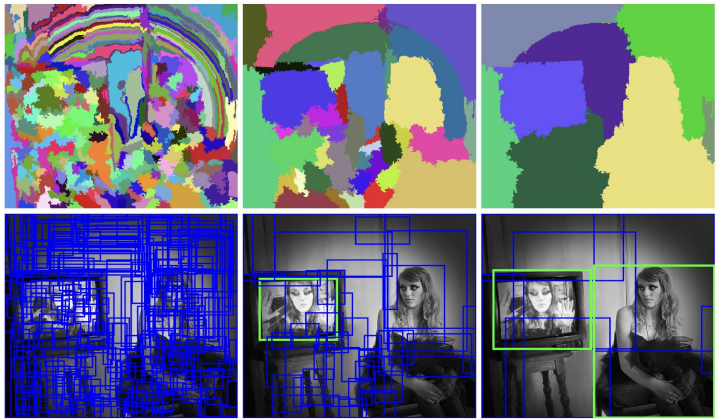
- Sometimes recognizing and classifying objects in an image is not enough and we want to precisely localize the objects.
- Applications in autonomous driving, robotics, face recognition, etc.
- Deep Neural Networks are really good at this!

Deep Neural Networks for Object Detection

Types of Object Detection Frameworks

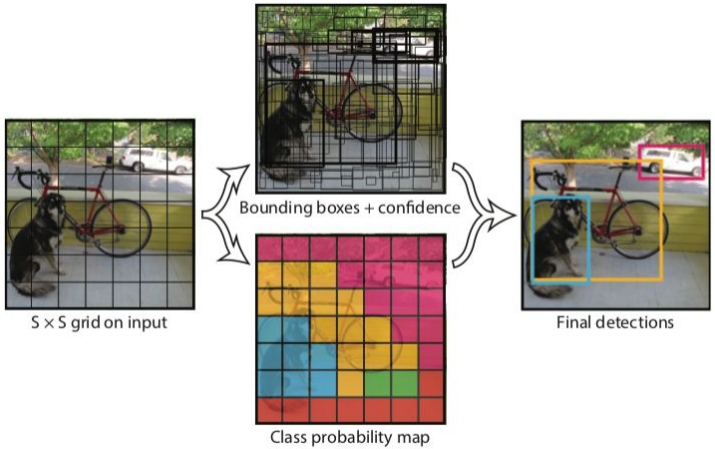
Region Proposal Based

- R-CNN
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN



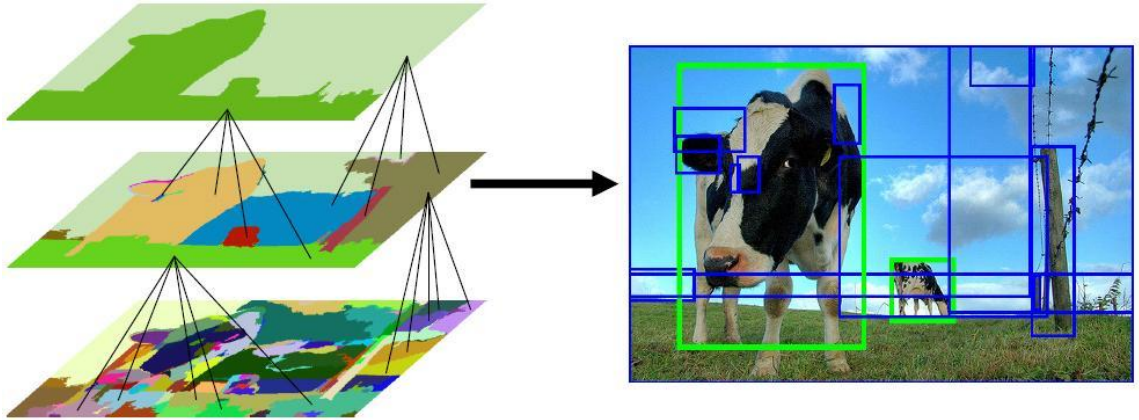
Single Pass or Regression/Classification Based

- YOLO: You Only Look Once
- Single Shot Detector (SSD)



Region Proposal

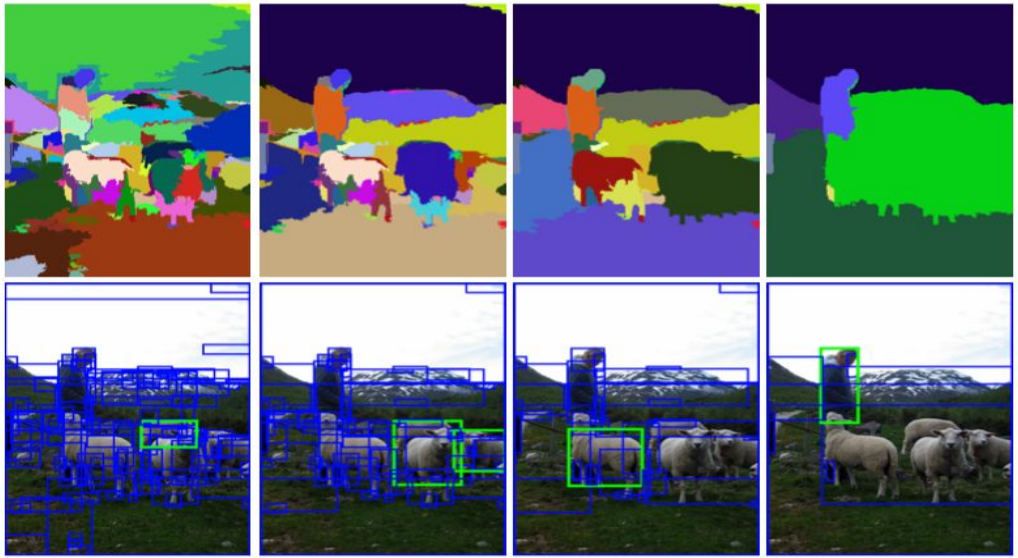
- Segmentation is the process of partitioning an image into regions which are similar to each other.
- Perform segmentation on the input image and propose regions of interest.
- Classify the proposed regions as an object. Output the class label and bounding box for it.



Selective Search for Object Detection

Region Proposal

- Region proposals are done at multiple scales.
- Selective Search is commonly used for Region Proposals.



Selective Search for Object Detection

R-CNN Family of Networks

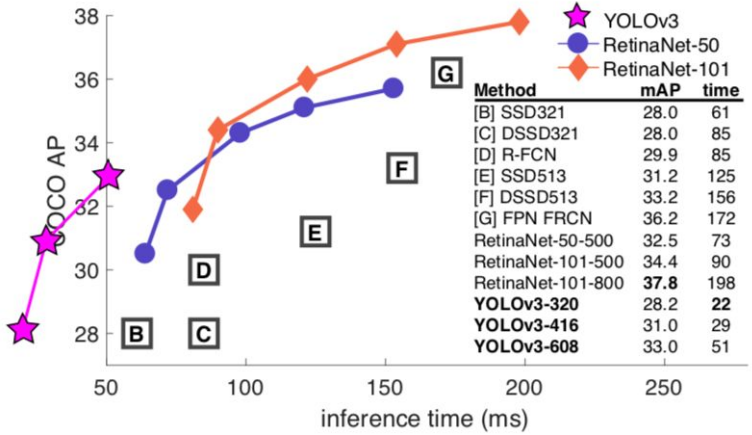
- R-CNN: Propose Regions. Classify proposed regions one at a time.
- Fast R-CNN: Propose Regions. Use a convolutional sliding window method to classify all regions.
- Faster R-CNN: Use a convolutional network to propose regions.
- Mask R-CNN: Extension of Faster R-CNN. Outputs mask for the segmented object along with bounding box.

R-CNN Family of Networks

- However, region proposal based networks are slow.
- Faster R-CNN achieves a maximum speed of 7 frames/second on a GPU!
- Not suitable for real-time purposes.

Single Pass Family of Networks

- Very fast.
- The image is passed through the network once. Multiple scales are handled inside the network itself.
- A single network performs both, classification and localization.

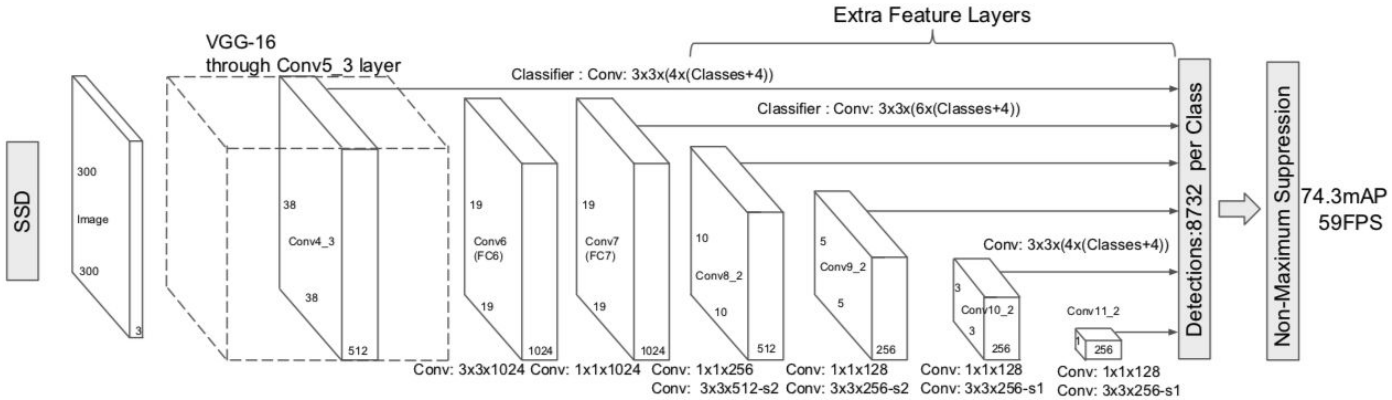


Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

YOLOv3: An Incremental Improvement

Single Shot Detector Framework

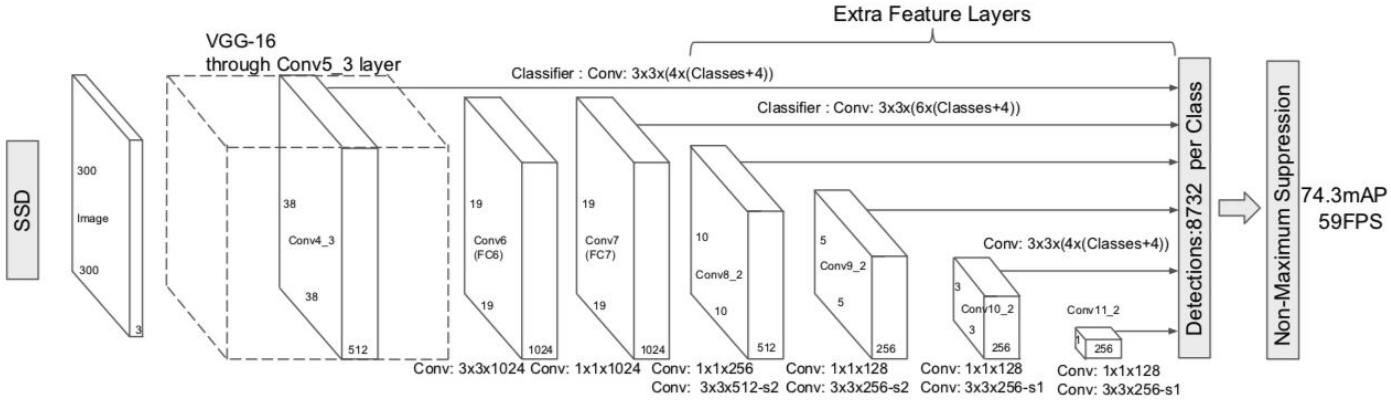
- Single Deep Neural Network.
- Completely eliminates region proposal.
- Performs detections at multiple scales and aspect ratios.
- Can have different base networks!



SSD: Single Shot MultiBox Detector

Multi-Scale Feature Maps

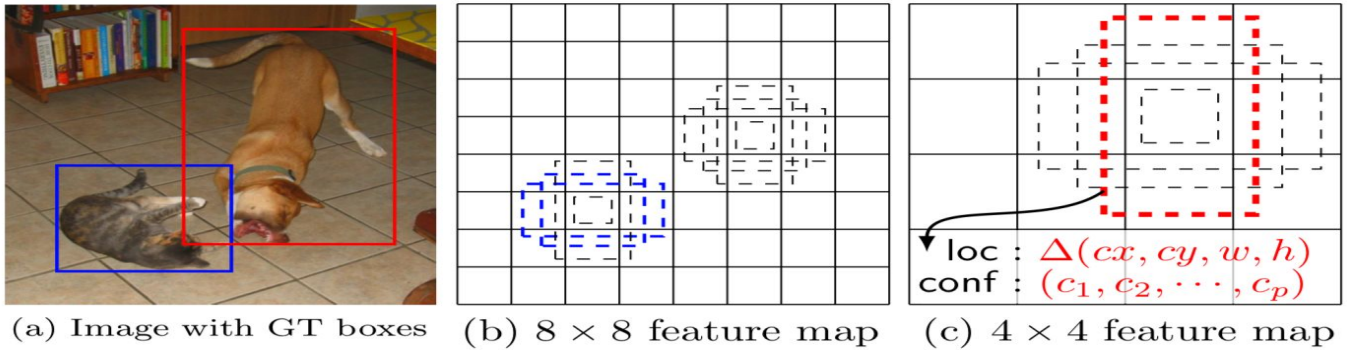
- Convolutional feature layers are added to the end of the base network.
- Layers decrease in size progressively.
- Allows predictions of detections at multiple scales.



SSD: Single Shot MultiBox Detector

Default Boxes and Aspect Ratios

- Set of default bounding boxes are associated with each feature map cell.
- At each feature map cell for each default box:
  - Predict offsets relative to the default box.
  - Per-class scores that indicate the presence of a class instance in the box.
- If we have *k* boxes, *c* classes and *4* offsets relative to the default box shape, for a $m \times n$ feature map, we yield an output of $(c + 4)kmn$.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Training Strategy

Key components of the SSD training pipeline:

- Matching Strategy
- Hard Negative Mining
- Data Augmentation

Matching Strategy

- During training, ground truth boxes need to be matched to their corresponding default boxes.
- Unlike other networks, which match one ground truth box to a default box, SSD allows a ground truth box to be matched to multiple default boxes.
- All default boxes which have an Intersection over Union (IoU) value greater than 0.5 with a ground truth box, is assigned to the ground truth box.
- Simplifies the learning problem.

Hard Negative Mining

- If a default box is matched to a ground truth box, it is considered as positive, else it is considered as negative.
- Most of the default boxes are negative, especially when we have a large number of default boxes.
- Not all negative examples are used. Ratio of negative to positive examples is at most 3:1.
- Leads to faster optimization and more stable training.

Data Augmentation

- Each training image is randomly sampled by one of the following methods:
  - Original image is used.
  - Sample a patch such that the minimum IoU with the objects is 0.1, 0.3, 0.5, 0.7 or 0.9.
  - Randomly sample a patch.
- Size of each sampled patch is [0.1, 1] of the original image size, and the aspect ratio is between [0.5, 2].
- Each sampled patch is then resized to a fixed size and horizontally flipped with a probability of 0.5. Some photometric distortions are also applied.
- Due to this extensive augmentation strategy, the results improved by 8.8%!

Off-the-shelf SSD Models

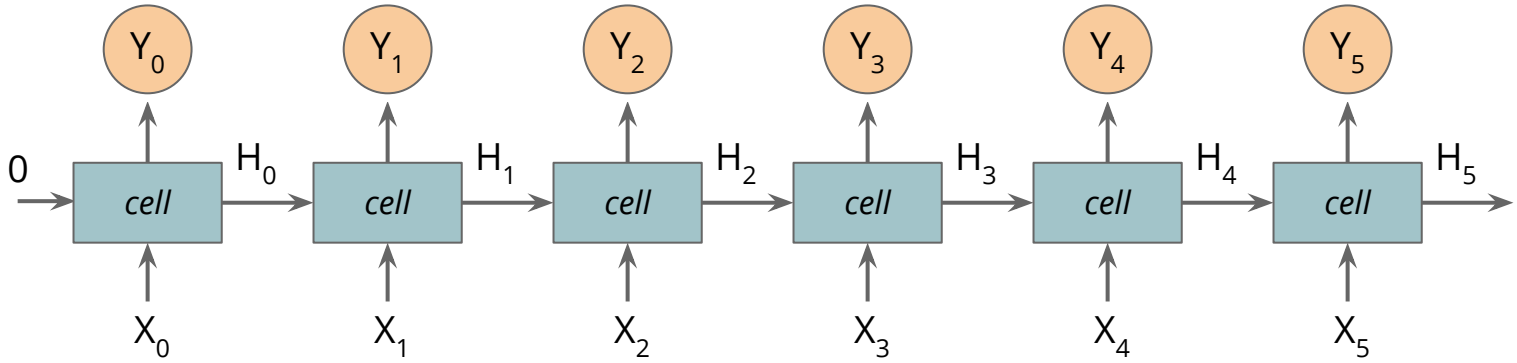
COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes

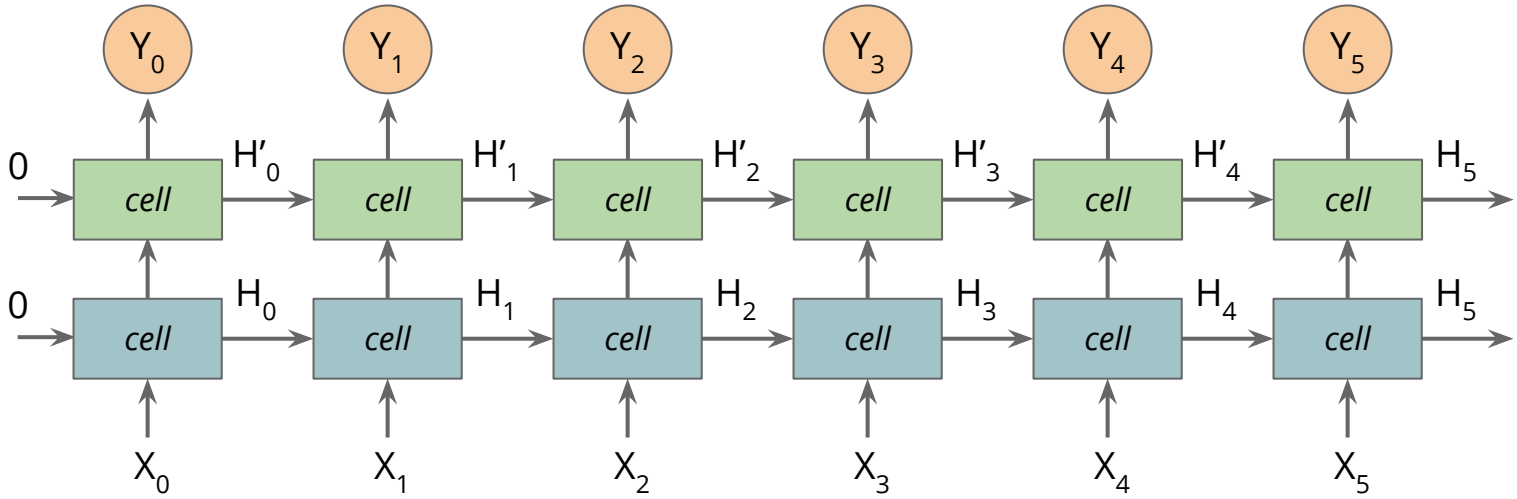
Tensorflow Detection Model Zoo

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

Practical perspective

A SimpleRNN

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(SimpleRNN(16, input_shape=(1,32,)))
model.summary()
```

Shape: 1 row, 32 features.

How many parameters in a SimpleRNN?

W = (32, 16) U = (16, 16) bias = 16

32 * 16 + 16 * 16 + 16

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_15 (SimpleRNN)    (None, 16)                784
=================================================================
Total params: 784
Trainable params: 784
Non-trainable params: 0
_________________________________________________________________
```

```
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.

```
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

```
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)
```

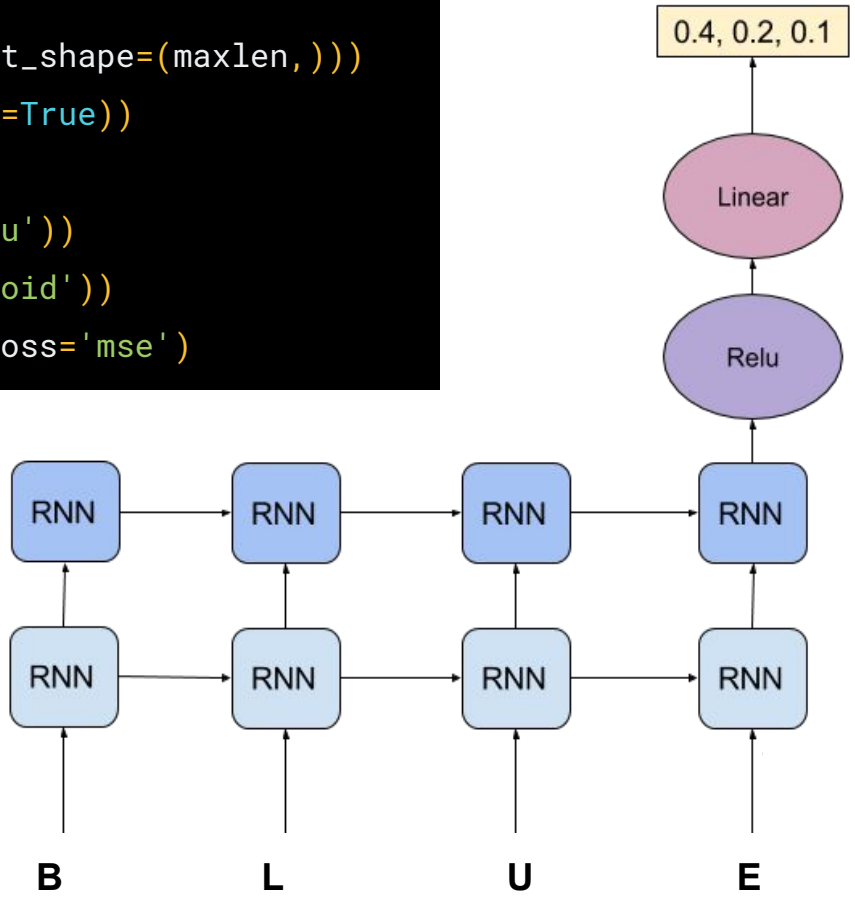
Now using LSTMs

```
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)
```

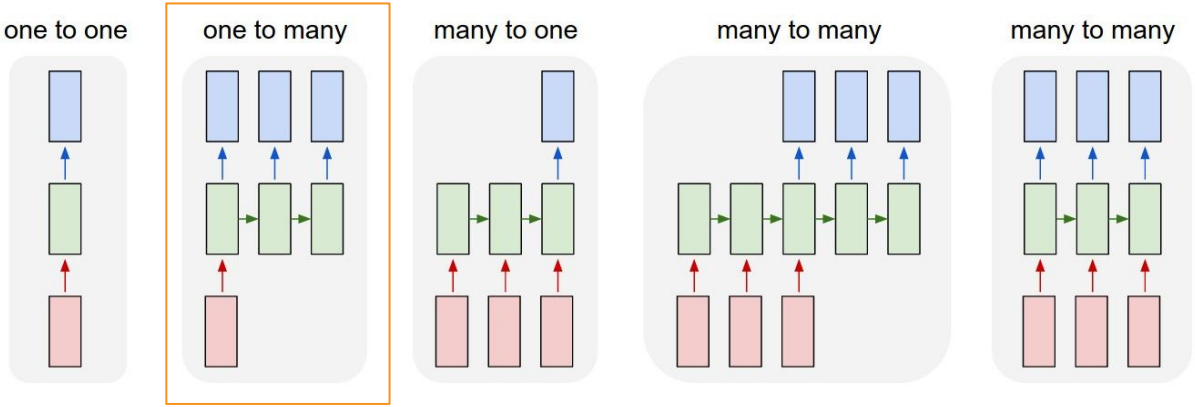
```
model = Sequential()
model.add(Embedding(maxwords, 32))
model.add(LSTM(16, return_sequences=True))
model.add(LSTM(16))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs=10, validation_split=0.2)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

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

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Reshape((1, maxlen), input_shape=(maxlen,)))
model.add(tf.keras.layers.GRU(128, return_sequences=True))
model.add(tf.keras.layers.GRU(128))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(3, activation='sigmoid'))
model.compile(optimizer=tf.train.AdamOptimizer(), loss='mse')
```



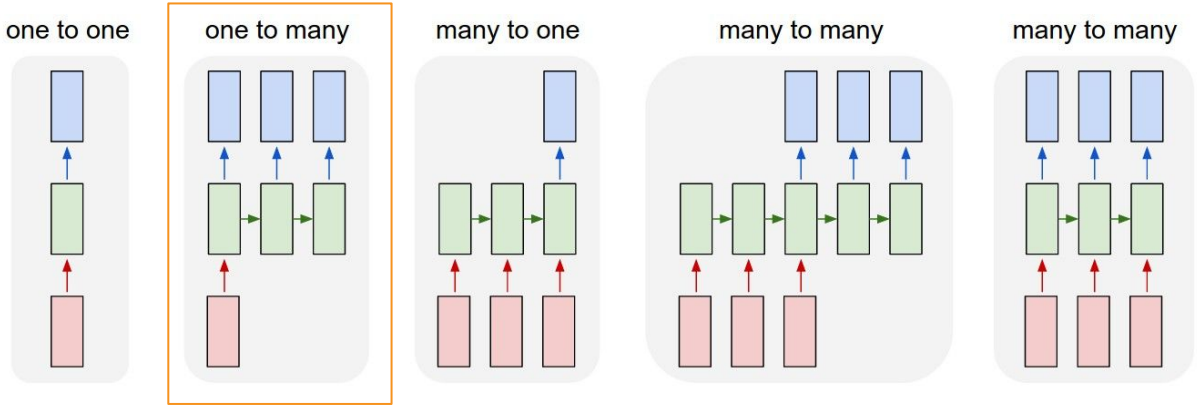
A bit more on generating sequences



Text generation

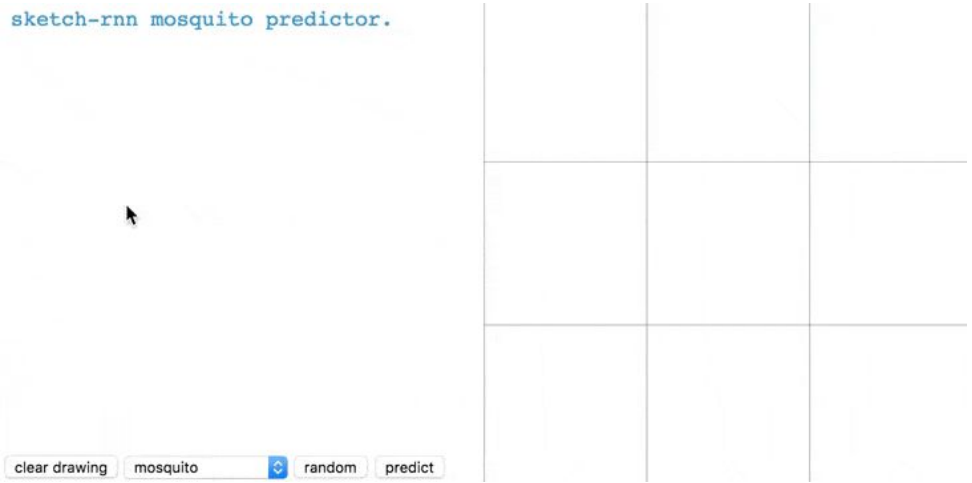
Complete code here (written verbosely...), and here (written concisely). Also see the decoder in the translation example.

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



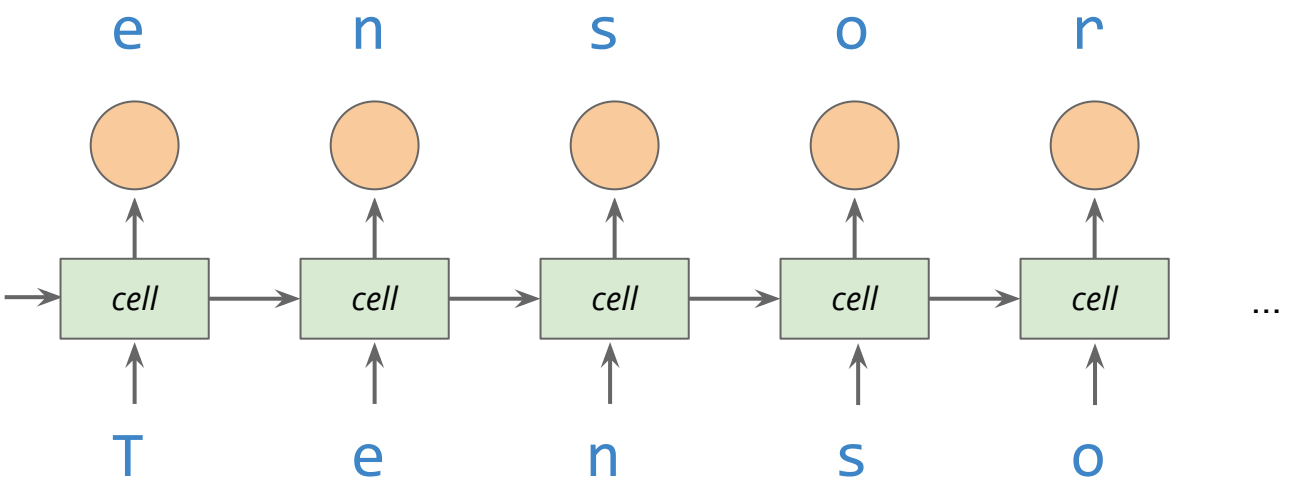
Sketch-RNN

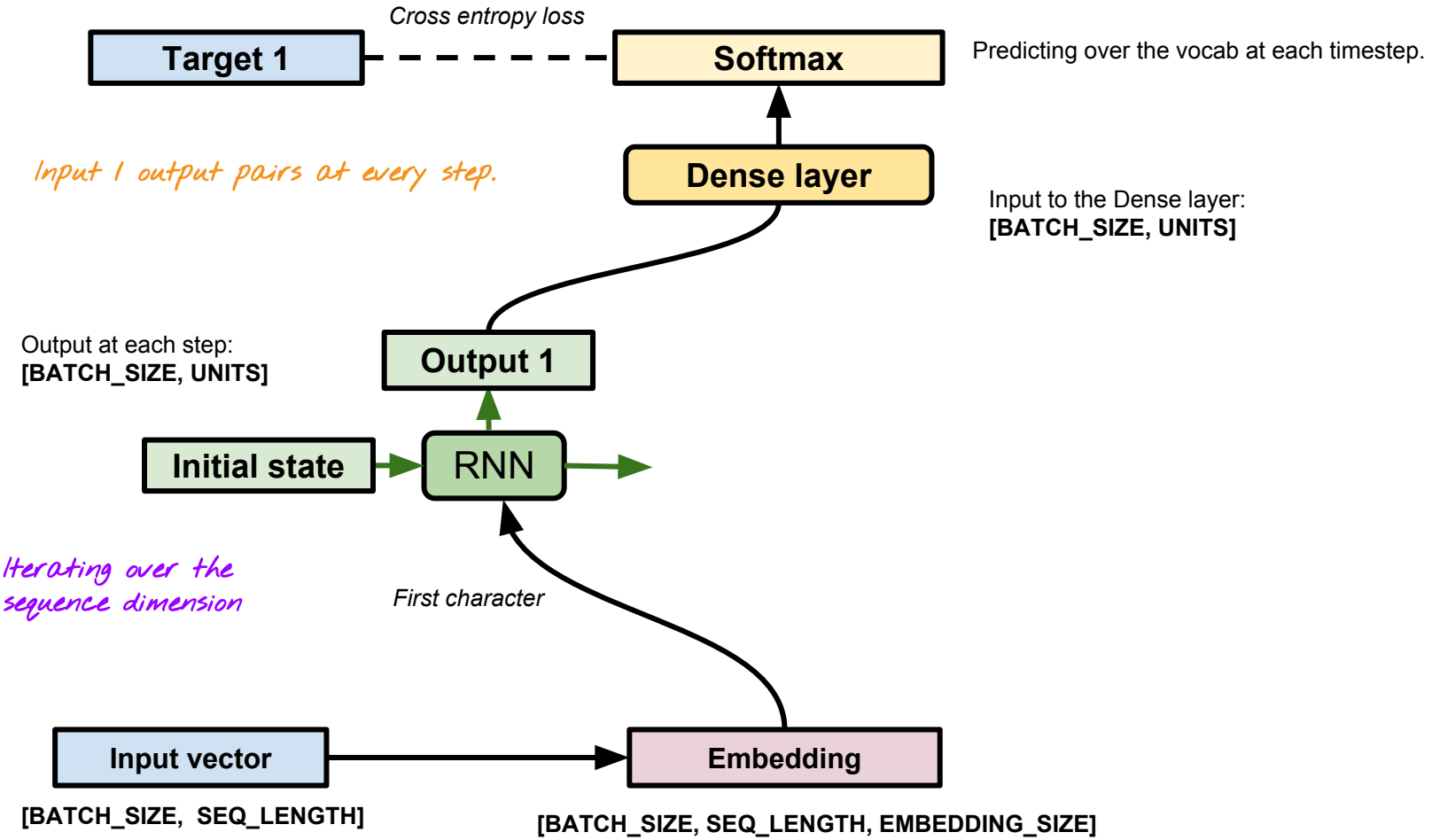
Same idea

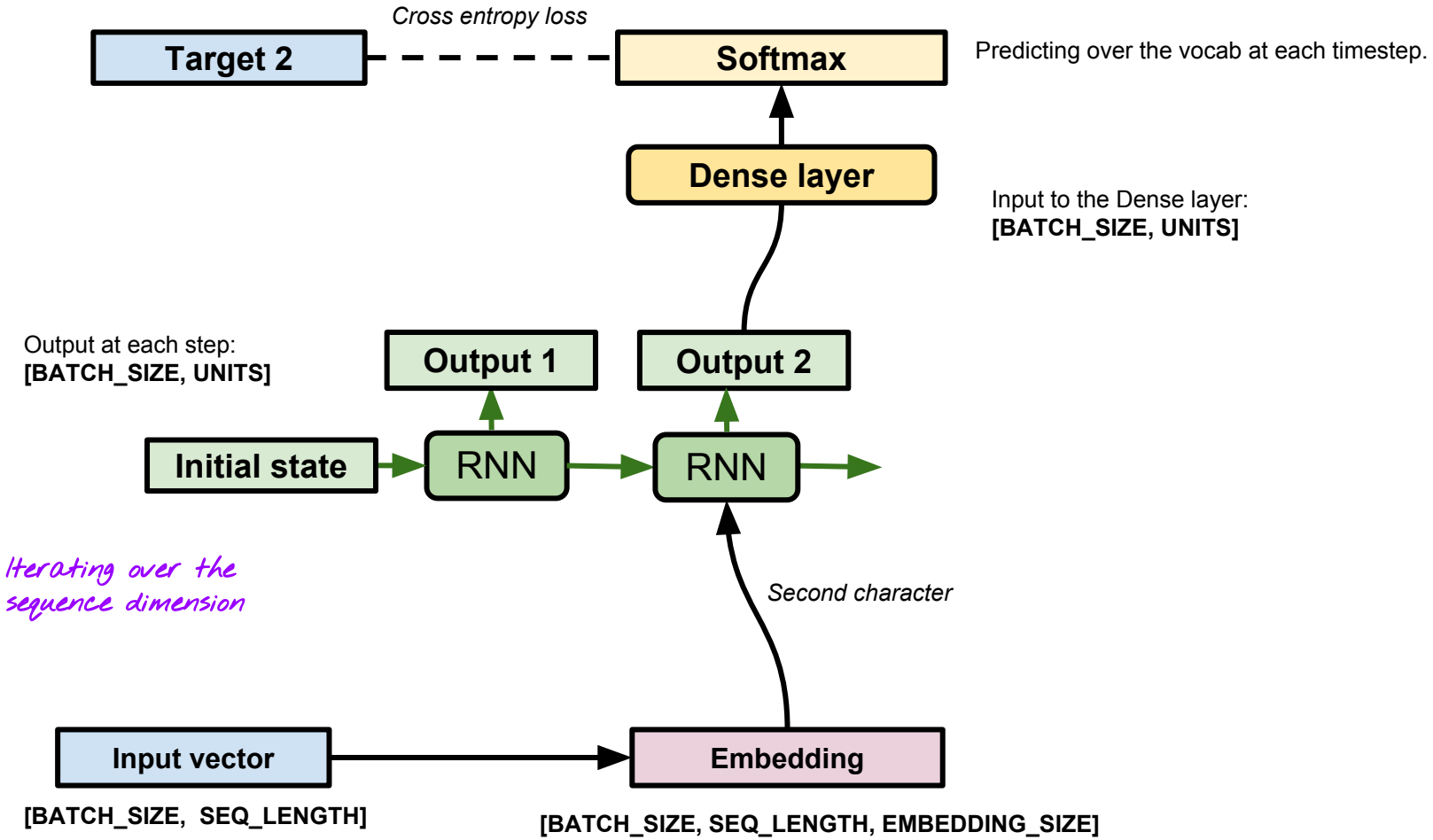


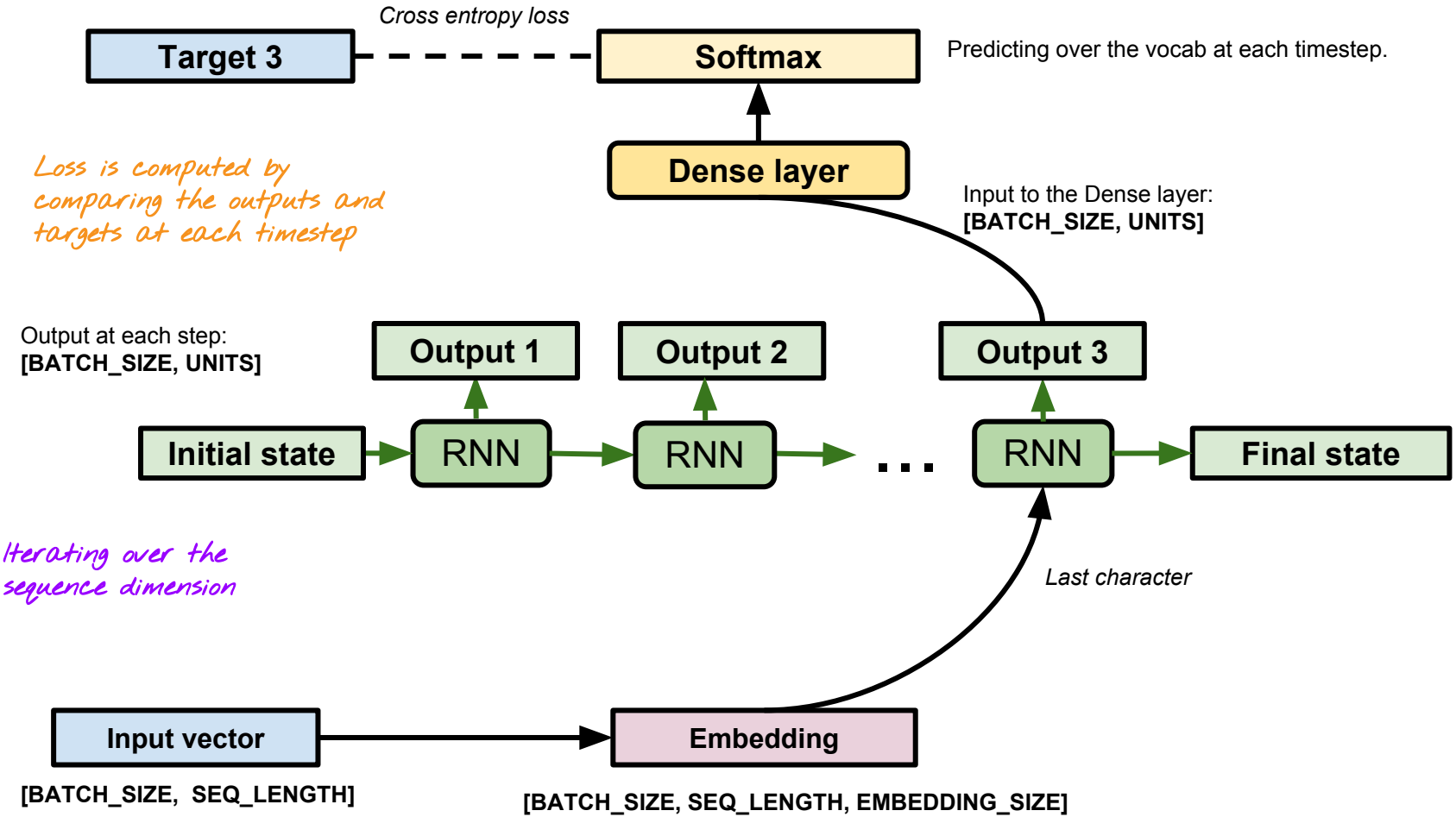
magenta.tensorflow.org/sketch-rnn-demo

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.



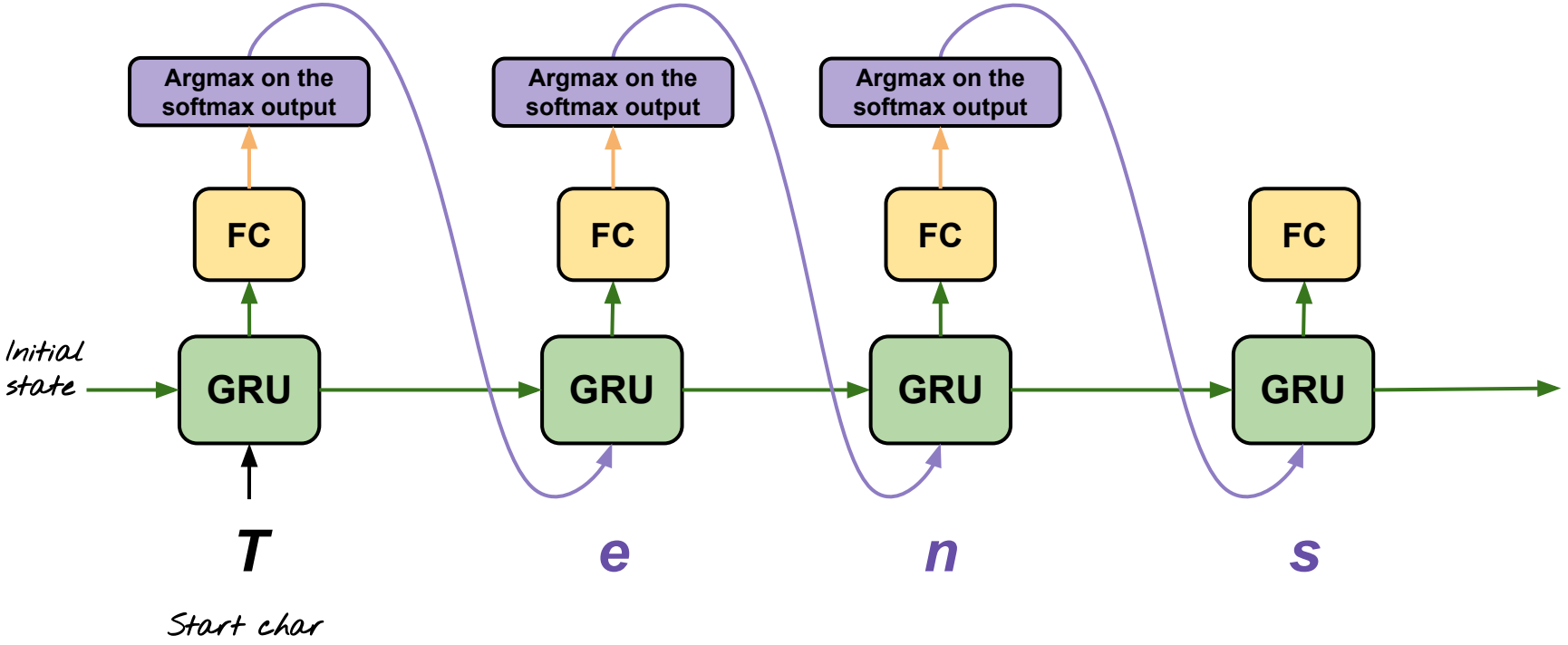

Cross entropy loss
Target 1
Softmax
Predicting over the vocab at each timestep.
Input / output pairs at every step.
Dense layer
Input to the Dense layer:
[BATCH_SIZE, UNITS]
Output at each step:
[BATCH_SIZE, UNITS]
Output 1
Initial state
RNN
Iterating over the sequence dimension
First character
Input vector
Embedding
[BATCH_SIZE, SEQ_LENGTH]
[BATCH_SIZE, SEQ_LENGTH, EMBEDDING_SIZE]


Cross entropy loss
Target 2
Softmax
Predicting over the vocab at each timestep.
Dense layer
Input to the Dense layer:
[BATCH_SIZE, UNITS]
Output at each step:
[BATCH_SIZE, UNITS]
Output 1
Output 2
Initial state
RNN
RNN
Iterating over the sequence dimension
Second character
Input vector
Embedding
[BATCH_SIZE, SEQ_LENGTH]
[BATCH_SIZE, SEQ_LENGTH, EMBEDDING_SIZE]


Cross entropy loss
Target 3
Softmax
Predicting over the vocab at each timestep.
Loss is computed by comparing the outputs and targets at each timestep
Dense layer
Input to the Dense layer:
[BATCH_SIZE, UNITS]
Output at each step:
[BATCH_SIZE, UNITS]
Output 1
Output 2
Output 3
Initial state
RNN
RNN
...
RNN
Final state
Iterating over the sequence dimension
Last character
Input vector
Embedding
[BATCH_SIZE, SEQ_LENGTH]
[BATCH_SIZE, SEQ_LENGTH, EMBEDDING_SIZE]

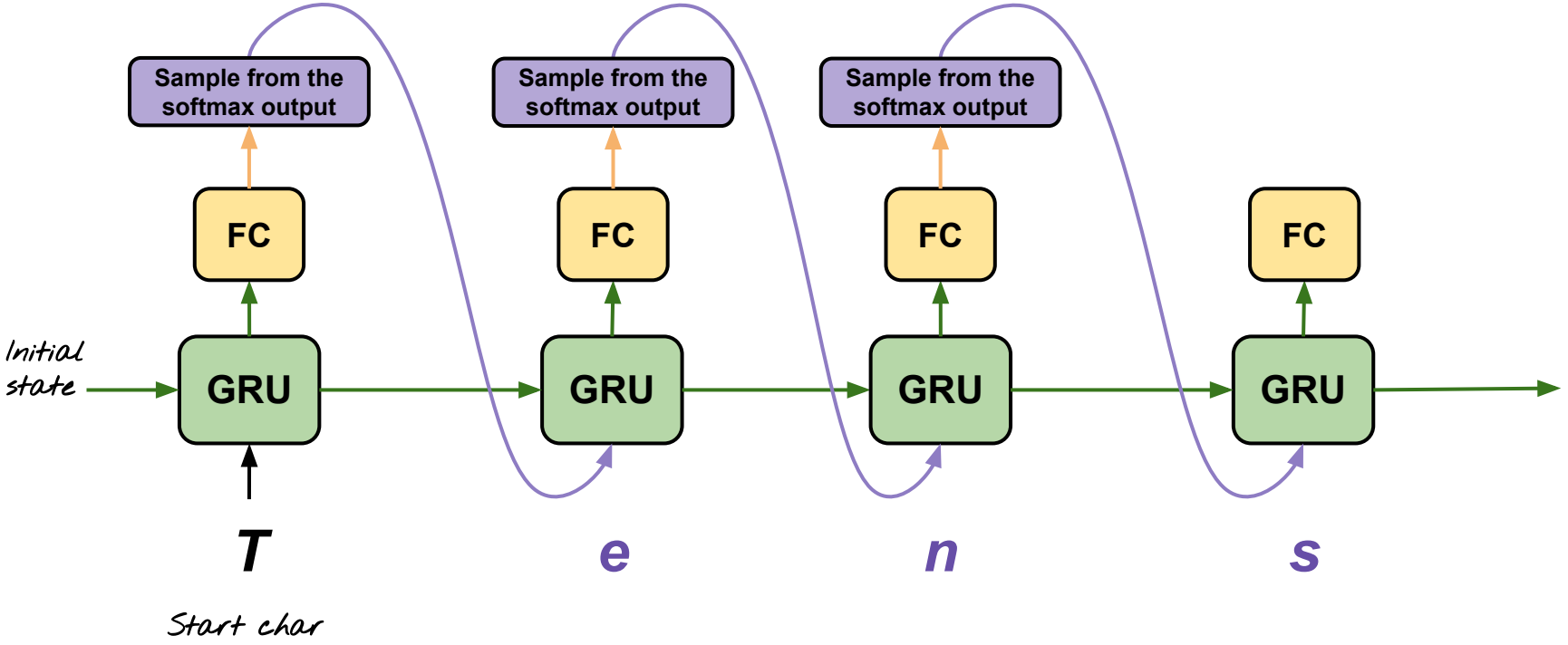
Sampling strategy 1

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

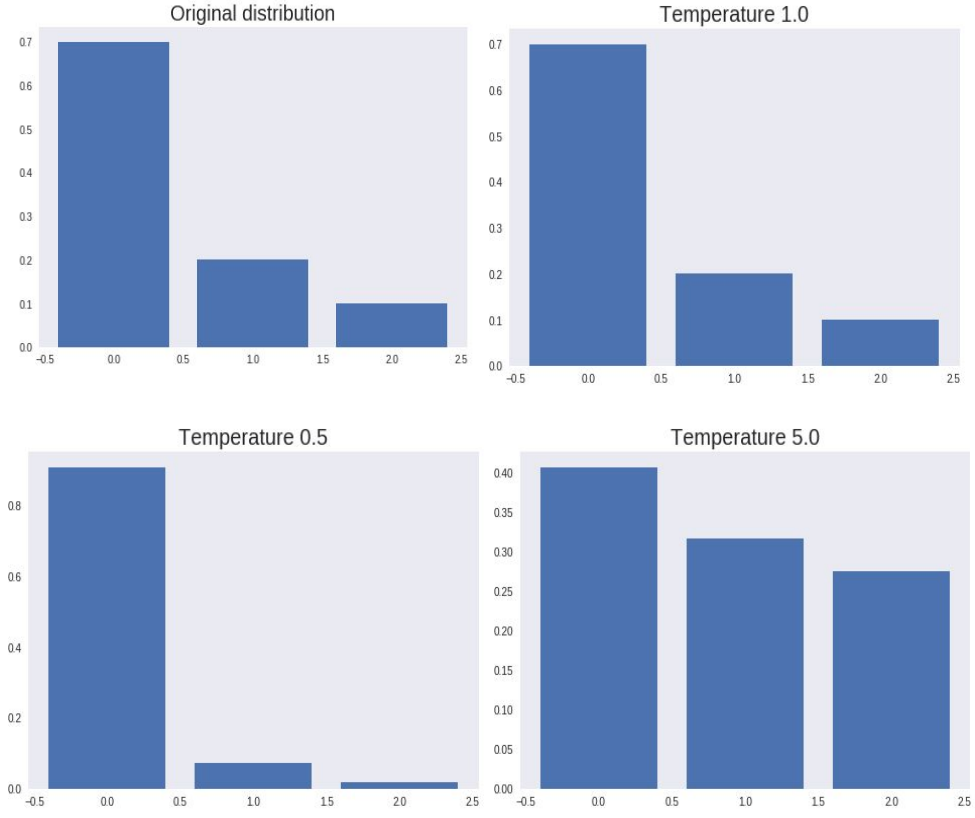


Sampling strategy 2

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 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.

Effect of high and low temperature

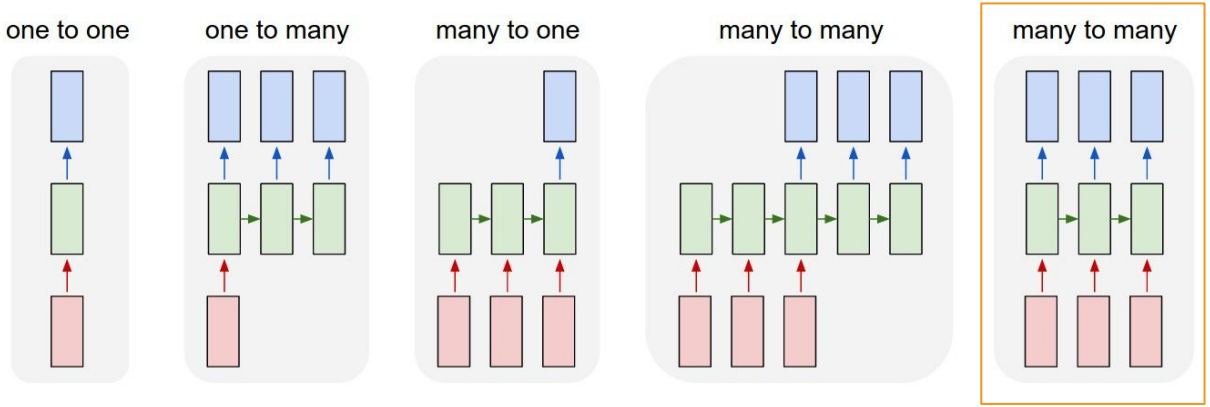
Temperature 0.0001

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



Video analysis

You want to make a prediction at each frame, conditioned on past context. We haven't implemented an example of this yet, but the Keras functional API guide sketches the idea.

Preprocessing

Sentence
“The cat sat on the mat.”

Vector
2, 7, 5, 8, 2, 3, 4

Tokens
“the”, “cat”, “sat”, “on”, “the”, “mat”, “.”

Embedding
(0.1. 1.1, 2.2), (1.9, 0.5, 0.6), ...

```
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]
```

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

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

As always, fit your tokenizer on the training data. A special OOV (out of vocabulary) token will be used to represent new words in validation / test.

```
>>> 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)

The source for the preprocessing utilities lives in this repo (can be difficult to tell from the API docs, due to the way they're generated)

```
>>> t.word_index

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

Sorted by frequency

- For A4 part 1, you'll need to export the word index as well when deploying your model to the web. Make sure the tokenization happens identically in JavaScript.
- Notice the index starts from 1, 0 is reserved (later used for padding).

```
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]]
```

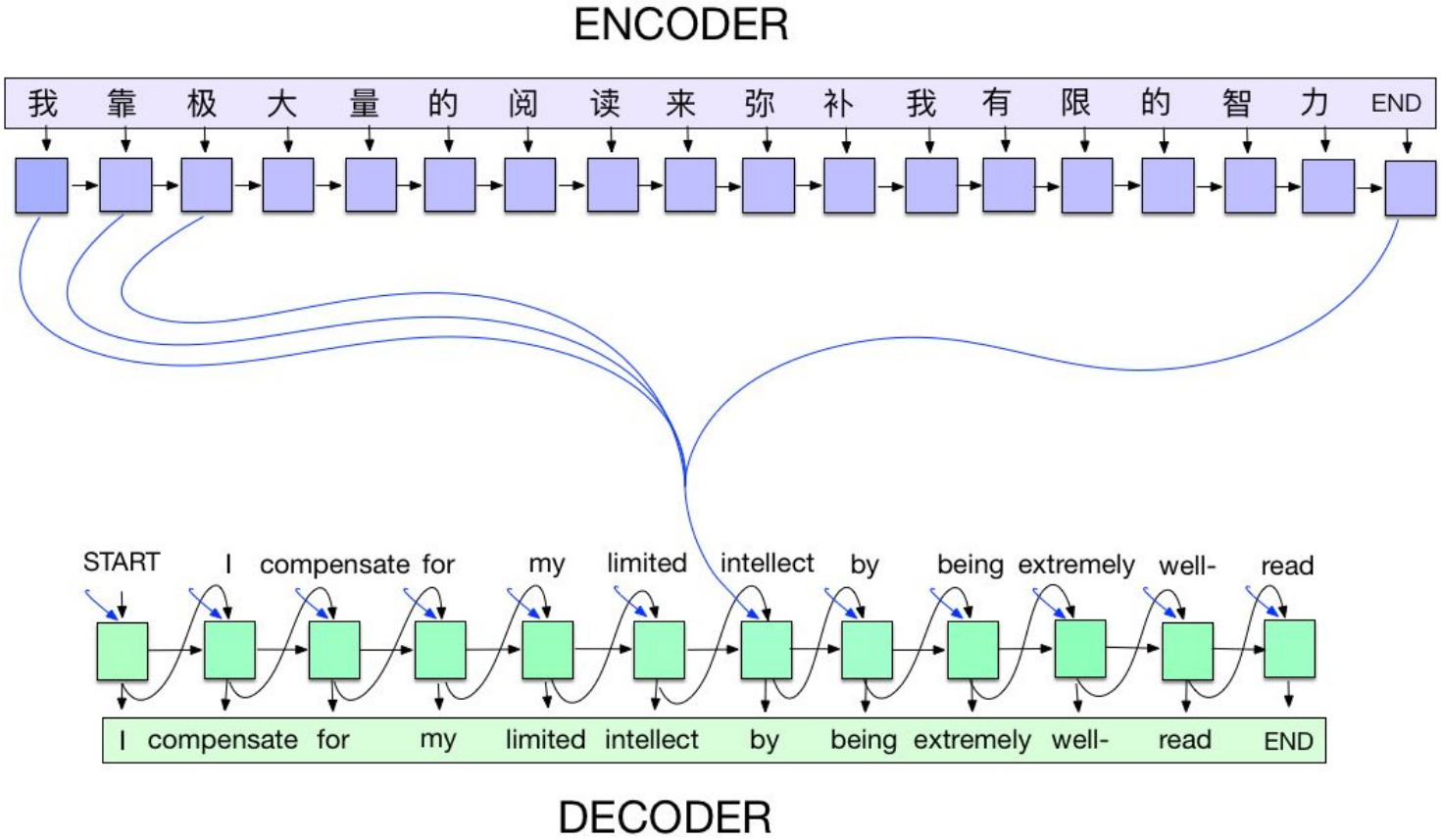
FYI, Convolution works well for sequences, too.

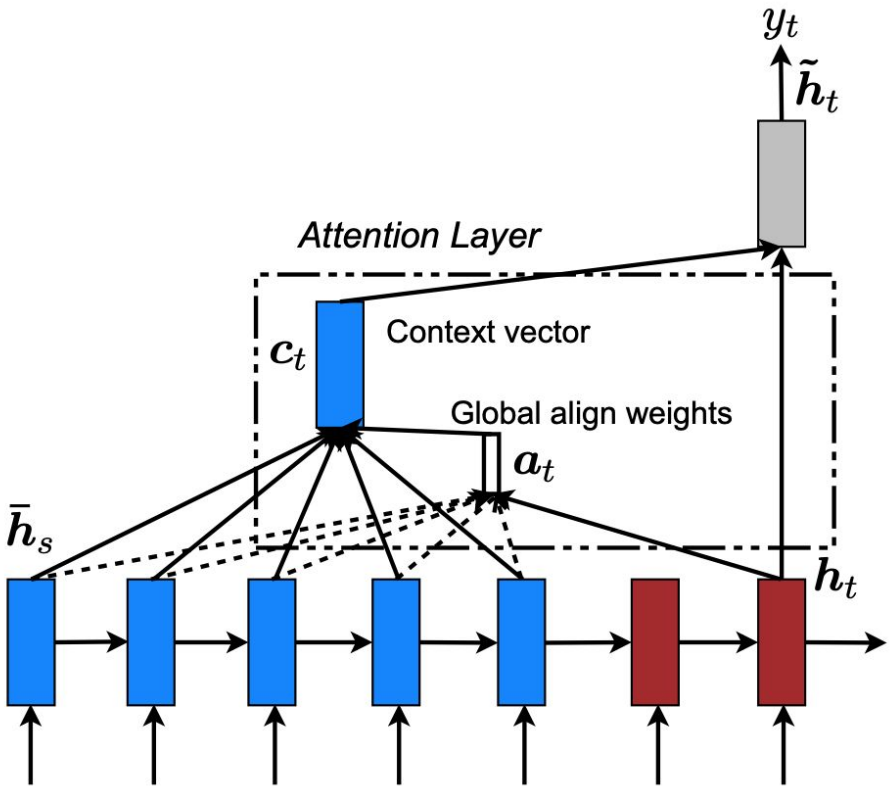
More efficient than RNNs

```
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))
```

Worth trying as a strong baseline.


ENCODER
我 靠 极 大 量 的 阅 读 来 弥 补 我 有 限 的 智 力 END
START I compensate for my limited intellect by being extremely well- read
I compensate for my limited intellect by being extremely well- read END
DECODER


y_t
$\tilde{h}_t$
Attention Layer
c_t
Context vector
Global align weights
a_t
$\bar{h}_s$
h_t

Reading

- Sequence to Sequence Learning with Neural Networks (2014)
- Deep Learning with Python chapter 6 (RNNs in practice)
- Deep Learning chapter 10 (background on RNNs)

Assignments

- A5 will involve machine translation. If you'd like to play with code early, check out the reference implementation, and this helpful blog post with a minimal example (best bet: start w/ the blog post, and try to understand the code for the plain seq2seq model, without attention).

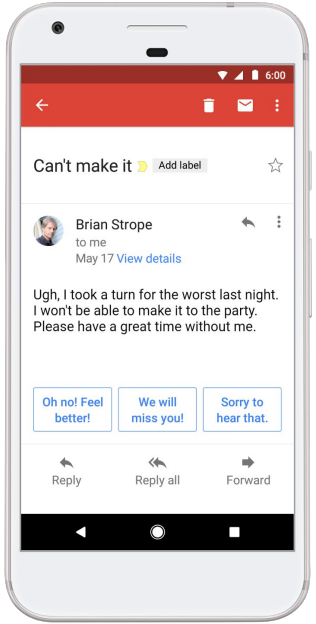
Reading cont'd

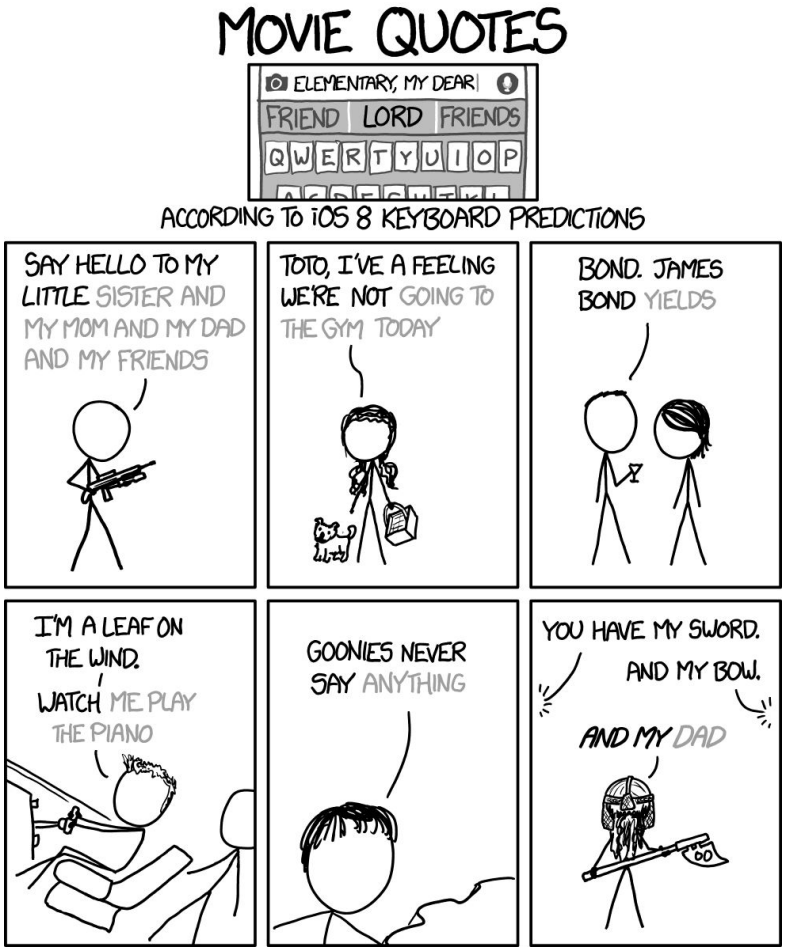
If you'd like to read ahead.

- Effective Approaches to Attention-based Neural Machine Translation (2015)
- Attention Is All You Need (2017)

Extras

Text generation

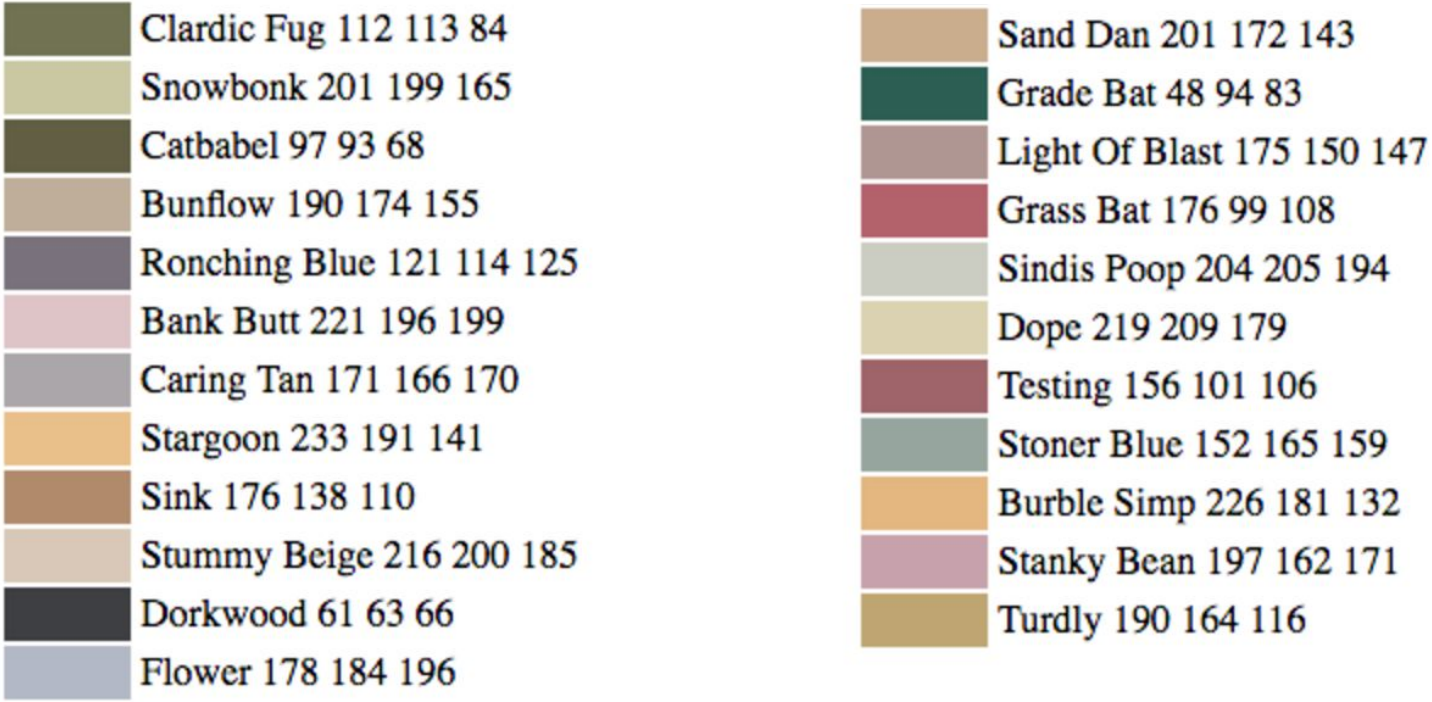




https://xkcd.com/1427/
https://ai.google/research/pubs/pub45189


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

An AI invented a bunch of new paint colors that are hilariously wrong



An AI invented a bunch of new paint colors that are hilariously wrong.