

# APPLICATIONS



## OF DATA SCIENCE

# Recurrent Neural Networks

## Applications of Data Science - Class 18

Giora Simchoni

[gsimchoni@gmail.com](mailto:gsimchoni@gmail.com) and add #dsapps in subject

Stat. and OR Department, TAU

2022-12-26

APPLICATIONS



OF DATA SCIENCE

# Time keeps moving on

(Janis Joplin)

APPLICATIONS



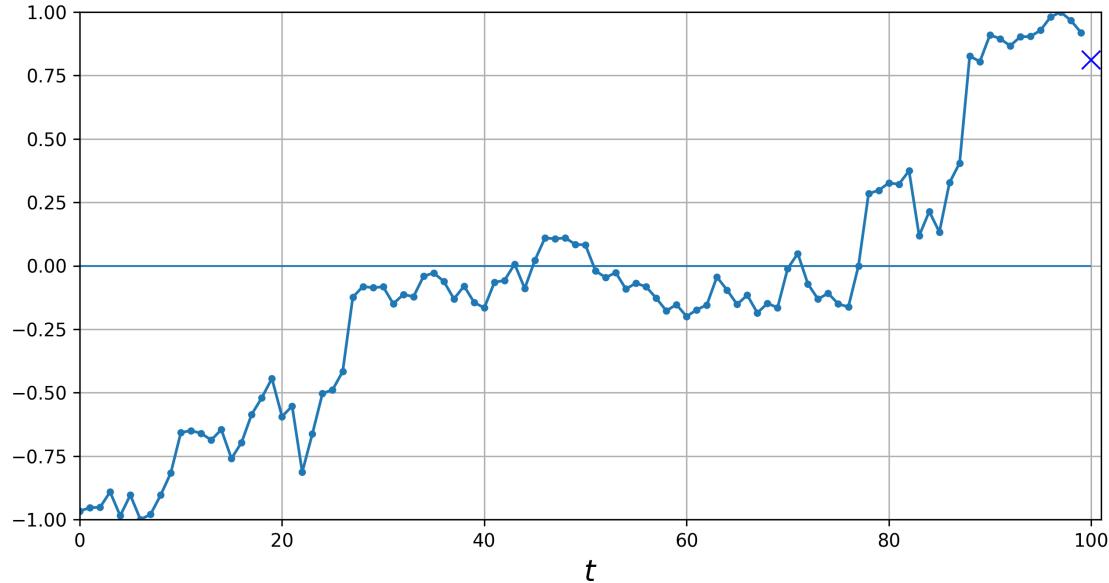
OF DATA SCIENCE

# What would Simone do next?



# What would be the price of GOOG tomorrow?

```
df = pd.read_csv("../data/fortune500_100_days.csv")
google = df[df['Fortune500'] == 'GOOG'].values[0][1:]
plot_series(google[:-1], 100, google[-1], y_label='')
plt.show()
```



# What is he going to say next?



כמazz וمتמיד, נוכחותנו  
על הקרקע, שליטתנו  
במקומות האסטרטגיים,  
ונחישותנו בהגנתם היא  
ש

# Simple RNN

APPLICATIONS

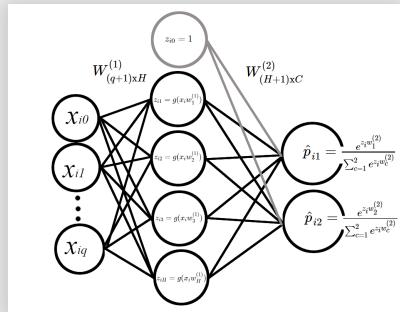


OF DATA SCIENCE

# The Setting

Suppose we have a univariate time series  $(x_{i1}, x_{i2}, \dots, x_{iT})$  of  $T$  time steps, where  $i = 1, \dots, N$  (e.g. Fortune  $N = 500$  stock price, the last  $T = 100$  days), and we want to predict  $y_i$ , which could be:

- The next day price (regression setting)
- Positive or negative outcome (classification setting)
- Simone Biles next move (???)



What are the disadvantages of a regular network in this setting?

# Detour: Time Series Analysis

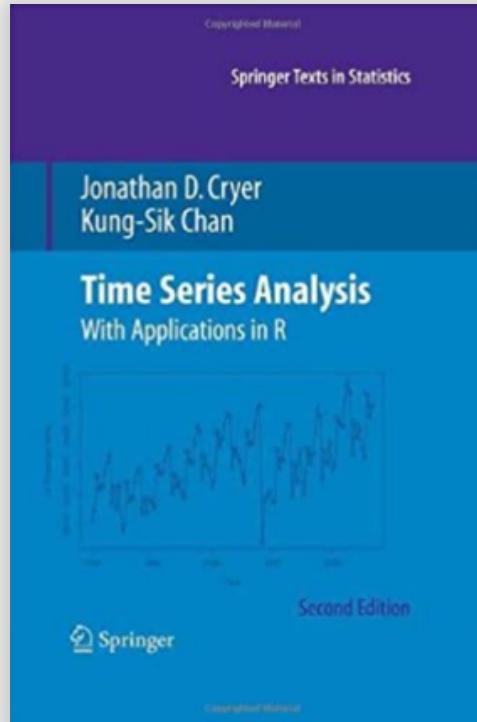
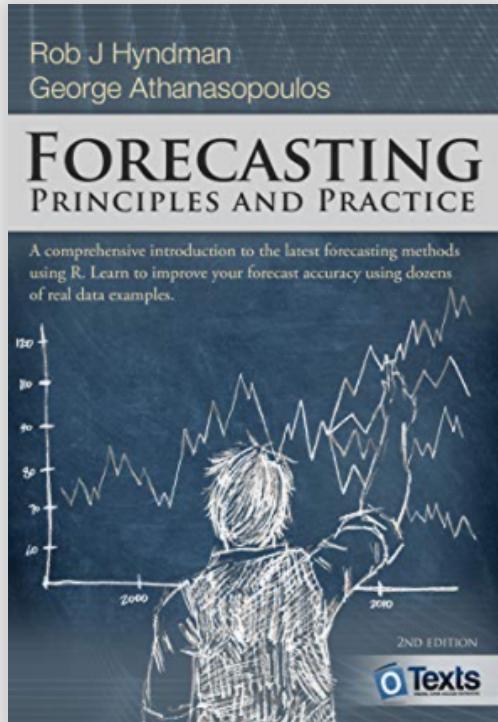
APPLICATIONS



OF DATA SCIENCE

# Don't invent the wheel!

Time Series Analysis is a big deal in Statistics.



APPLICATIONS



# End of Detour

APPLICATIONS

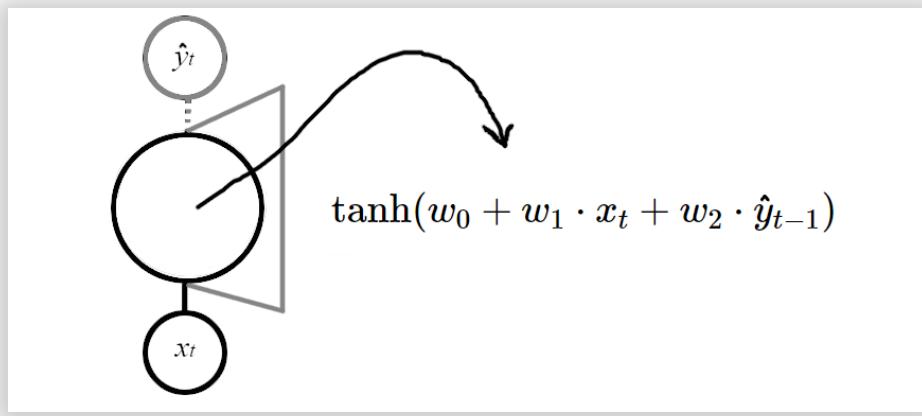


OF DATA SCIENCE

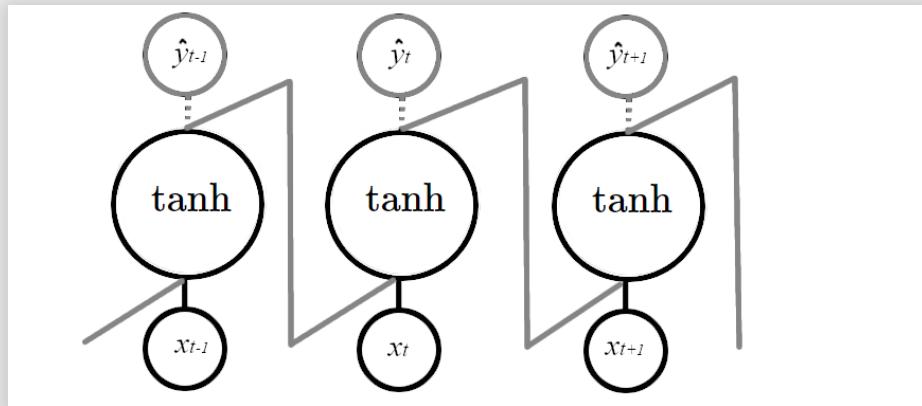
# A Single Neuron RNN

The most basic Single-Neuron RNN would:

- take  $x_t$ , learn from it and output  $\hat{y}_t$
- then take both the next input  $x_{t+1}$  and previous output  $\hat{y}_t$ , learn from them
- by learning we mean forward and backward propagating at each stage with some loss  $L$  (e.g. MSE)



# Unrolling a Neuron



💡 But how many parameters are we actually learning?

What important NN principle is demonstrated here?

# If you build it, they will come.

Remember we built a Logistic Regression NN? Guess what!

At high level nothing changes!

```
def single_rnn(X, y, epochs, alpha):
    w = np.array([1, 1, 1])
    ls = np.zeros(epochs)
    for i in range(epochs):
        l, w = optimize(X, y, alpha, w)
        ls[i] = l
    return ls, w

def optimize(X, y, alpha, w):
    y_pred_arr, l = forward(X, y, w)
    grad = backward(X, y, y_pred_arr, w)
    w = gradient_descent(alpha, w, grad)
    return l, w

def gradient_descent(alpha, w, grad):
    return w - alpha * grad
```

# Forward Propagation

$$\hat{y}_1 = \tanh(w_0 + w_1 \cdot x_1 + w_2 \cdot y_0)$$

⋮

$$\hat{y}_T = \tanh(w_0 + w_1 \cdot x_T + w_2 \cdot \hat{y}_{T-1})$$

$$L = MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_{iT})^2$$

```
def forward(X, y, w):
    N, T = X.shape
    y_pred_arr = np.zeros((N, T + 1))
    y_pred = np.zeros(N)
    y_pred_arr[:, 0] = y_pred
    for t in range(T):
        y_pred = np.tanh(w[0] + X[:, t] * w[1] + y_pred * w[2])
        y_pred_arr[:, t + 1] = y_pred
    l = np.mean((y - y_pred)**2)
    return y_pred_arr, l
```

# Backward Propagation

$$\frac{\partial \hat{y}_{i1}}{\partial w_0} = \frac{\partial \tanh(o_{i1})}{\partial o_{i1}} \frac{\partial o_{i1}}{\partial w_0} = [1 - \tanh^2(o_{i1})] \cdot 1 = 1 - \hat{y}_{i1}^2$$

$$\frac{\partial \hat{y}_{i2}}{\partial w_0} = \frac{\partial \tanh(o_{i2})}{\partial o_{i2}} \frac{\partial o_{i2}}{\partial w_0} = (1 - \hat{y}_{i2}^2)(1 + w_2 \frac{\partial \hat{y}_{i1}}{\partial w_0})$$

⋮

$$\frac{\partial L}{\partial w_0} = \sum_{i=1}^N \frac{\partial L}{\partial \hat{y}_{iT}} \frac{\partial \hat{y}_{iT}}{\partial w_0} = \sum_{i=1}^N -\frac{2}{N}(y_i - \hat{y}_{iT}) \frac{\partial \tanh(o_{iT})}{\partial o_{iT}} \frac{\partial o_{iT}}{\partial w_0} =$$

$$= \sum_{i=1}^N -\frac{2}{N}(y_i - \hat{y}_{iT})(1 - \hat{y}_{iT}^2)[1 + w_2 \frac{\partial \hat{y}_{iT-1}}{\partial w_0}]$$

And you are cordially invited to do the same for  $\frac{\partial L}{\partial w_1}$  and  $\frac{\partial L}{\partial w_2}$ .

```

def backward_t(X, y_pred_arr, w, grads, t, N):
    y_t = y_pred_arr[:, t]
    if t == 0:
        grads_w0 = np.ones((N, ))
        grads_w1 = X[:, t]
        grads_w2 = y_t
    else:
        dot_dyprev = w[2]
        dyprev_doprev = 1 - y_t ** 2
        grads_w0 = np.ones(N) + dot_dyprev * dyprev_doprev * grads[:, 0]
        grads_w1 = X[:, t] + dot_dyprev * dyprev_doprev * grads[:, 1]
        grads_w2 = y_t + dot_dyprev * dyprev_doprev * grads[:, 2]
    return np.stack([grads_w0, grads_w1, grads_w2], axis=1)

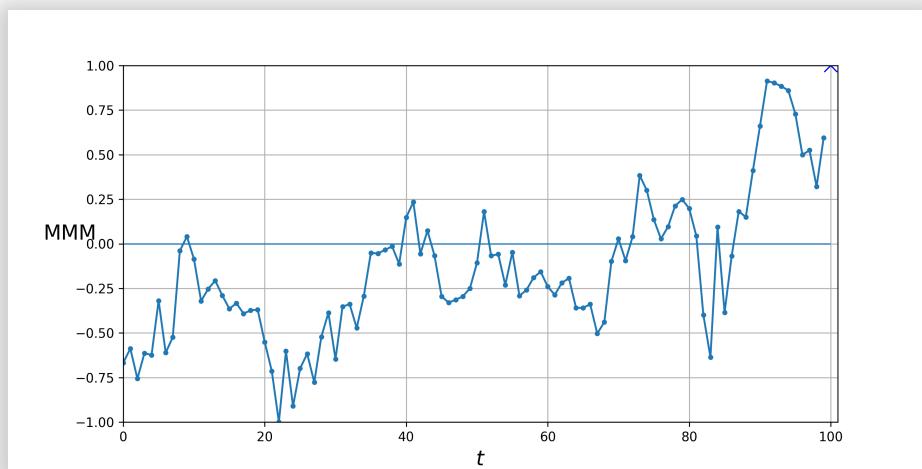
def backward(X, y, y_pred_arr, w):
    N, T = X.shape
    y_pred = y_pred_arr[:, -1]
    dl_dypred = -2 * (y - y_pred) / N
    dypred_dot = 1 - y_pred ** 2
    grads = np.zeros((N, 3))
    for t in range(T):
        grads = backward_t(X, y_pred_arr, w, grads, t, N)
    for j in range(3):
        grads[:, j] *= dl_dypred * dypred_dot
    final_grads = grads.sum(axis=0)
    return final_grads

```

# The Fortune500 Stocks

```
df = pd.read_csv("../data/fortune500_100_days.csv")
X = df.iloc[:, 1: ].values
y = X[:, -1]
X = X[:, :-1]

plt.clf()
plot_series(X[1, :], 100, y[1], y_label="MMM")
plt.show()
```



# What would be a good MSE?

Predicting with the mean of  $y$  (the 101st day mean stock price)

```
np.mean((y - y.mean())**2)
```

```
## 0.3516210190443937
```

Predicting with the last column of  $X$  (the 100th day stock price)

```
np.mean((y - X[:, -1])**2)
```

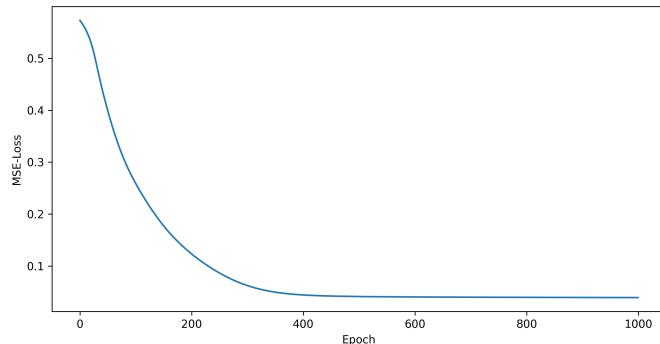
```
## 0.033289150367058463
```

# Training with our neuron

```
mses, w = single_rnn(X, y, epochs=1000, alpha=0.01)
print(w)
```

```
## [-0.07161043  1.15476269  0.25800392]
```

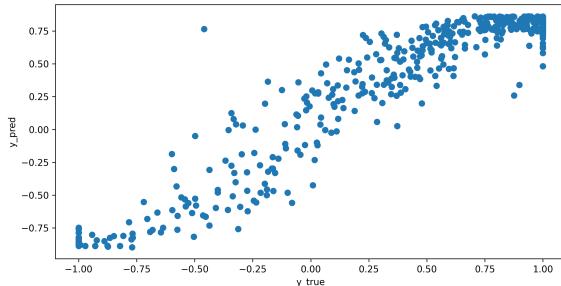
```
plt.plot(mses)
plt.ylabel('MSE-Loss'); plt.xlabel('Epoch')
plt.show()
```



```
y_pred_arr, mse = forward(X, y, w)
y_pred = y_pred_arr[:, -1]
print(mse)
```

```
## 0.03900926352398098
```

```
plt.scatter(y, y_pred)
plt.ylabel('y_pred'); plt.xlabel('y_true')
plt.show()
```



Are you surprised? How could we easily improve?

What would be a better approach for this simple dataset?

# Finally, Keras

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import SimpleRNN
from tensorflow.keras.optimizers import Adam

model = Sequential([
    SimpleRNN(1, input_shape=(None, 1))
])
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse')

X = X[:, :, np.newaxis]
y = y[:, np.newaxis]

print(X.shape)

## (502, 100, 1)

print(y.shape)

## (502, 1)

history = model.fit(X, y, epochs=50, verbose=0)
```

```

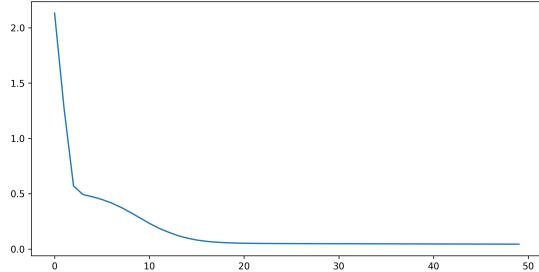
print(model.get_weights() [2], model.get_weights() [0], model.get_weights() [1])
## [-0.05025314] [[0.89638436]] [[0.4493832]]

y_pred = model.predict(X, verbose=0)
print(np.mean((y - y_pred) ** 2))

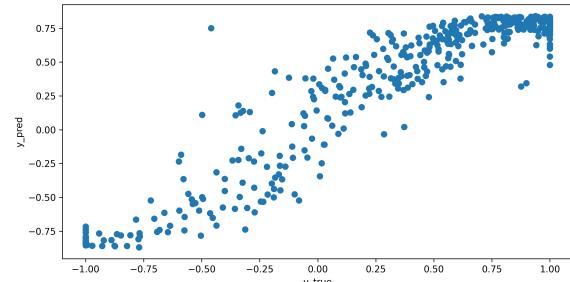
## 0.0437715170171821

plt.plot(history.history['loss'])
plt.show()

```



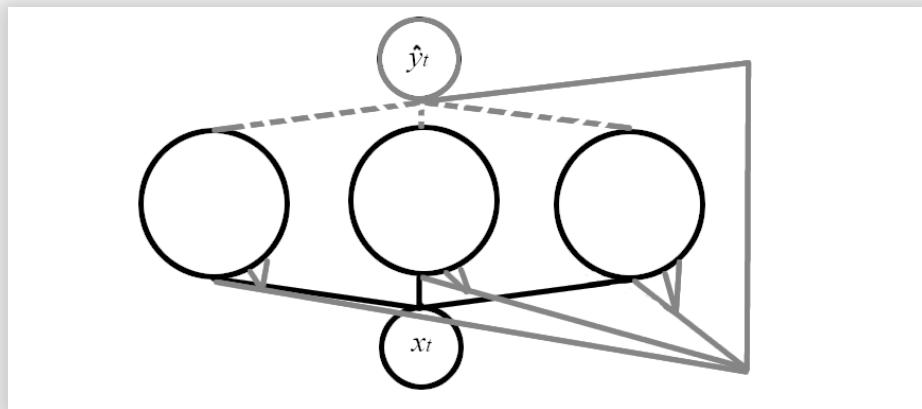
```
plt.scatter(y, y_pred)
plt.ylabel('y_pred'); plt.xlabel('y_true')
plt.show()
```



# Add inputs, Add neurons

$$\hat{Y}_t = \tanh(W_0 + X_t \cdot W_1 + \hat{Y}_{t-1} \cdot W_2)$$

- $X_t$  is  $n$  batch size X  $m$  inputs
- $W_1$  is  $m$  inputs X  $p$  neurons
- $\hat{Y}_t$  is  $n$  batch size X  $p$  neurons
- $W_2$  is  $p$  neurons X  $p$  neurons
- $W_0$  is  $p$  X 1 bias vector



# Add layers

```
model = Sequential([
    SimpleRNN(10, return_sequences=True, input_shape=[None, 1]),
    SimpleRNN(5, return_sequences=True),
    SimpleRNN(1)
])

model.compile(optimizer=Adam(learning_rate=0.01), loss='mse')
```

```
model.summary()
```

```
## Model: "sequential_1"
##
##   Layer (type)          Output Shape       Param #
##   ========
##   simple_rnn_1 (SimpleRNN)    (None, None, 10)        120
##   simple_rnn_2 (SimpleRNN)    (None, None, 5)         80
##   simple_rnn_3 (SimpleRNN)    (None, 1)            7
##
## Total params: 207
## Trainable params: 207
## Non-trainable params: 0
##
```

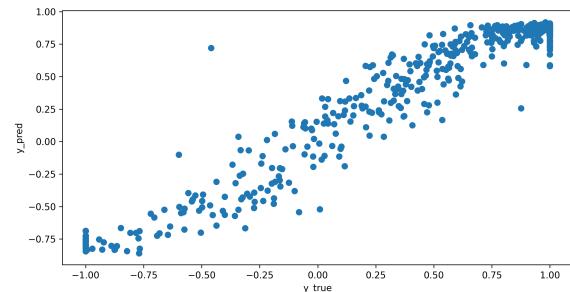
- 10 neurons each having: 1 inputs, 1 bias, all other 10 outputs in layer:  $10 * (1 + 1 + 10)$
- 5 neurons each having: 10 inputs, 1 bias, all other 5 outputs in layer:  $5 * (10 + 1 + 5)$
- 1 neurons each having: 5 inputs, 1 bias, all other 1 outputs in layer:  $1 * (5 + 1 + 1)$

```
history = model.fit(X, y, epochs=30, verbose=0)

y_pred = model.predict(X, verbose=0)
print(np.mean((y - y_pred) **2))

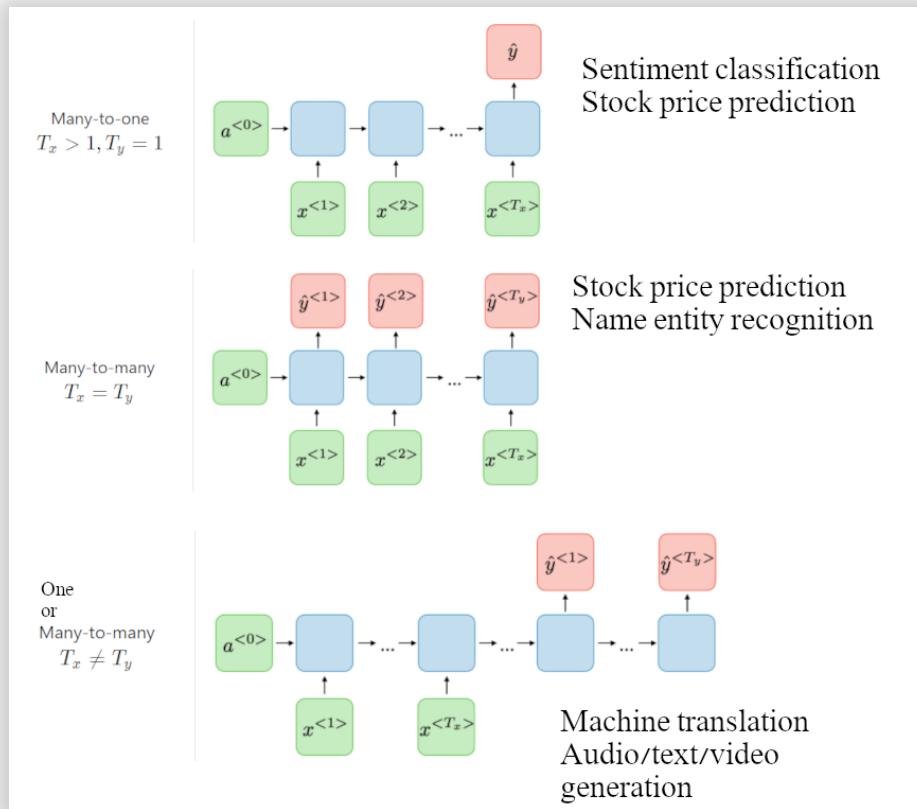
## 0.027710174771783818
```

```
plt.scatter(y, y_pred)
plt.ylabel('y_pred'); plt.xlabel('y_true')
plt.show()
```



Though ~200 params for predicting 500 numbers... sounds a bit much.

# Possibilities are endless



# Detour: Text is a Time Series!

APPLICATIONS



OF DATA SCIENCE

## But we first have to tokenize it

```
from tensorflow.keras.preprocessing import text, sequence

sentences = [
    'I love Stats so much',
    'I love ML too',
    'I love DS',
    'I love NN'
]

tokenizer = text.Tokenizer(num_words = 10)
tokenizer.fit_on_texts(sentences)

print(tokenizer.word_counts)

## OrderedDict([('i', 4), ('love', 4), ('stats', 1), ('so', 1), ('much', 1))

print(tokenizer.word_index)

## {'i': 1, 'love': 2, 'stats': 3, 'so': 4, 'much': 5, 'ml': 6, 'too': 7, 'l': 8, 'ds': 9, 'nn': 10}

print(tokenizer.index_word)

## {1: 'i', 2: 'love', 3: 'stats', 4: 'so', 5: 'much', 6: 'ml', 7: 'too', 8: 'l', 9: 'ds', 10: 'nn'}
```

## Then make it a sequence

```
text_sequences = tokenizer.texts_to_sequences(sentences)
print(text_sequences)

## [[1, 2, 3, 4, 5], [1, 2, 6, 7], [1, 2, 8], [1, 2, 9]]

X = sequence.pad_sequences(text_sequences,
    padding='post', truncating='post', maxlen=4)
print(X)

## [[1 2 3 4]
##  [1 2 6 7]
##  [1 2 8 0]
##  [1 2 9 0]]
```

## Now we can embed it

```
from tensorflow.keras.layers import Embedding

embed_layer = Embedding(input_dim= (10 + 1), output_dim=3)

print(embed_layer(np.array([1])))

## tf.Tensor([[ 0.00612465  0.00696361 -0.04569301]], shape=(1, 3), dtype=f

X_embedded = embed_layer(X)

print(X_embedded.shape)

## (4, 4, 3)

print(X_embedded)

## tf.Tensor(
## [[[ 0.00612465  0.00696361 -0.04569301]
## [-0.00974723 -0.00407351  0.03557405]
## [-0.03233683  0.00490437  0.04315286]
## [-0.03733008  0.03002742 -0.02753231]]
## [[ 0.00612465  0.00696361 -0.04569301]
```

# End of Detour

APPLICATIONS



OF DATA SCIENCE

# Yelp!

~600K (!) [text reviews](#) of shops and restaurants, polarized to negative (1-2 stars) and positive (3-4 stars), 560K in training set.

```
(ds_train, ds_test), ds_info = tfds.load('yelp_polarity_reviews',
    split=['train', 'test'], with_info=True)

df_train = tfds.as_dataframe(ds_train, ds_info)
df_test = tfds.as_dataframe(ds_test, ds_info)
df_test['text'] = df_test['text'].str.decode('utf-8')
df_train['text'] = df_train['text'].str.decode('utf-8')

print(df_train.shape)
print(df_test.shape)
print(df_test.head(3))

## (560000, 2)

## (38000, 2)

##      label                           text
## 0      0      Was not impressed, and will not return.
## 1      0      I went in to purchase overalls and was treated...
## 2      0      This place really is horrible... Every time I ...
```

## Yelp! But with sequences

```
from sklearn.model_selection import train_test_split

max_features = 10000
seq_len = 100

tokenizer = text.Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(df_train['text'])
text_sequences = tokenizer.texts_to_sequences(df_train['text'])
X = sequence.pad_sequences(text_sequences, padding='post',
    truncating='post', maxlen=seq_len)

X_train, X_test, y_train, y_test = train_test_split(X,
    df_train['label'], test_size = 0.2)

print(X_train.shape)
print(X_test.shape)

## (448000, 100)
## (112000, 100)
```

In case you're wondering, yes, there are smarter text generators, but even this 0.5M rows  $\times$  matrix is only ~220MB.

## Yelp with MLP

Remember there's nothing preventing you from using a simple NN, for (almost) everything:

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

n_cells = 10
epochs = 100
batch_size = 30
words_embed_dim = 50
callbacks = EarlyStopping(monitor='val_loss', patience=5)

mlp = Sequential([
    Embedding(max_features + 1, words_embed_dim),
    Dense(n_cells, activation='relu'),
    Dense(1, activation='sigmoid')
])

mlp.compile(loss = 'binary_crossentropy',
            optimizer='adam', metrics='accuracy')
```

```
mlp.summary()
```

```
## Model: "sequential_2"
## -----
##   Layer (type)          Output Shape       Param #
##   embedding_1 (Embedding)    (None, None, 50)        500050
##   dense (Dense)           (None, None, 10)        510
##   dense_1 (Dense)          (None, None, 1)         11
## -----
## Total params: 500,571
## Trainable params: 500,571
## Non-trainable params: 0
## -----
```

```
history = mlp.fit(X_train, y_train, validation_split=0.1, callbacks=callbacks,
                    batch_size=batch_size, epochs=epochs)
```

```
mlp.evaluate(X_test, y_test)
```

```
## [0.6664084792137146, 0.5871875286102295]
```

# Yelp with RNN

```
rnn = Sequential([
    Embedding(max_features + 1, words_embed_dim),
    SimpleRNN(n_cells, return_sequences=True),
    SimpleRNN(1, activation='sigmoid')
])

rnn.compile(loss = 'binary_crossentropy',
            optimizer='adam', metrics='accuracy')

rnn.summary()
```

```
## Model: "sequential_3"
##
##           Layer (type)        Output Shape       Param #
##           ====== ====== ======
##           embedding_2 (Embedding)   (None, None, 50)      500050
##           simple_rnn_4 (SimpleRNN)  (None, None, 10)      610
##           simple_rnn_5 (SimpleRNN)  (None, 1)          12
##           ======
##           Total params: 500,672
##           Trainable params: 500,672
##           Non-trainable params: 0
##
```



```
history = rnn.fit(X_train, y_train, validation_split=0.1, callbacks=callbacks)
batch_size=batch_size, epochs=epochs)

rnn.evaluate(X_test, y_test)

## [0.44967198371887207, 0.8164107203483582]
```

# 1D Convolution Layers

APPLICATIONS



OF DATA SCIENCE

# If you got it in 2D...

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1																				
2	0	0	1	2	1	0	0		0.2	0.6	0.2		0.2	1	1.6	1	0.2		1	1.6
3																				
4	x							w					z							
5																				

Or, in a formula:  $Z_i = b + \sum_{v=0}^{f_w-1} X_{i-\frac{f_w-1}{2}+v} \cdot W_v$

Or, in Numpy:

```
np.convolve([0,1,2,1,0], [0.2,0.6,0.2][::-1], 'same')
```

```
## array([0.2, 1., 1.6, 1., 0.2])
```

💡 Why would we want this?

```

from tensorflow.keras.layers import Conv1D

rnn_conv1d = Sequential([
    Embedding(max_features + 1, words_embed_dim),
    Conv1D(filters=5, kernel_size=2, strides=1),
    SimpleRNN(n_cells, return_sequences=True),
    SimpleRNN(1, activation='sigmoid')
])

rnn_conv1d.compile(loss = 'binary_crossentropy',
                    optimizer='adam', metrics='accuracy')

rnn_conv1d.summary()

```

```

## Model: "sequential_4"
## -----
##   Layer (type)          Output Shape       Param #
##   ========
##   embedding_3 (Embedding)    (None, None, 50)      500050
##   conv1d (Conv1D)           (None, None, 5)       505
##   simple_rnn_6 (SimpleRNN)  (None, None, 10)      160
##   simple_rnn_7 (SimpleRNN)  (None, 1)            12
##   ========
##   Total params: 500,727
##   Trainable params: 500,727

```

```
history = rnn_conv1d.fit(X_train, y_train, validation_split=0.1, c  
batch_size=batch_size, epochs=epochs)  
  
rnn_conv1d.evaluate(X_test, y_test)  
  
## [0.4487762749195099, 0.825705349445343]
```

# LSTM

APPLICATIONS



OF DATA SCIENCE

# RNN Cell has Short Memory

I hated this bar, though the bartender was so handsome and the drink he made me was absolutely delicious.

The RNN sees [I, hated, this, bar, ..., so, handsome, ... absolutely, delicious].

What do you think it would predict?

Enter Long Short-Term Memory cells (LSTM).

LSTM keeps track of its memory, of its state  $C_t$ , by constantly updating how much it needs to:

- forget from previous state:  $f_t \cdot C_{t-1}$
- remember from current "candidate" state:  $i_t \cdot \tilde{C}_t$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Where  $i_t$  and  $f_t$  are "gates" between 0 and 1.

Finally, the state goes through  $\tanh()$  activation and another 0-1 gate  $o_t$ , and the output is:

$$\hat{Y}_t = o_t \cdot \tanh(C_t)$$

So how do we learn the gates and get  $\tilde{C}_t$ ? Don't panic:

$$f_t = \sigma(W_{0f} + X_t \cdot W_{1f} + \hat{Y}_{t-1} \cdot W_{2f})$$

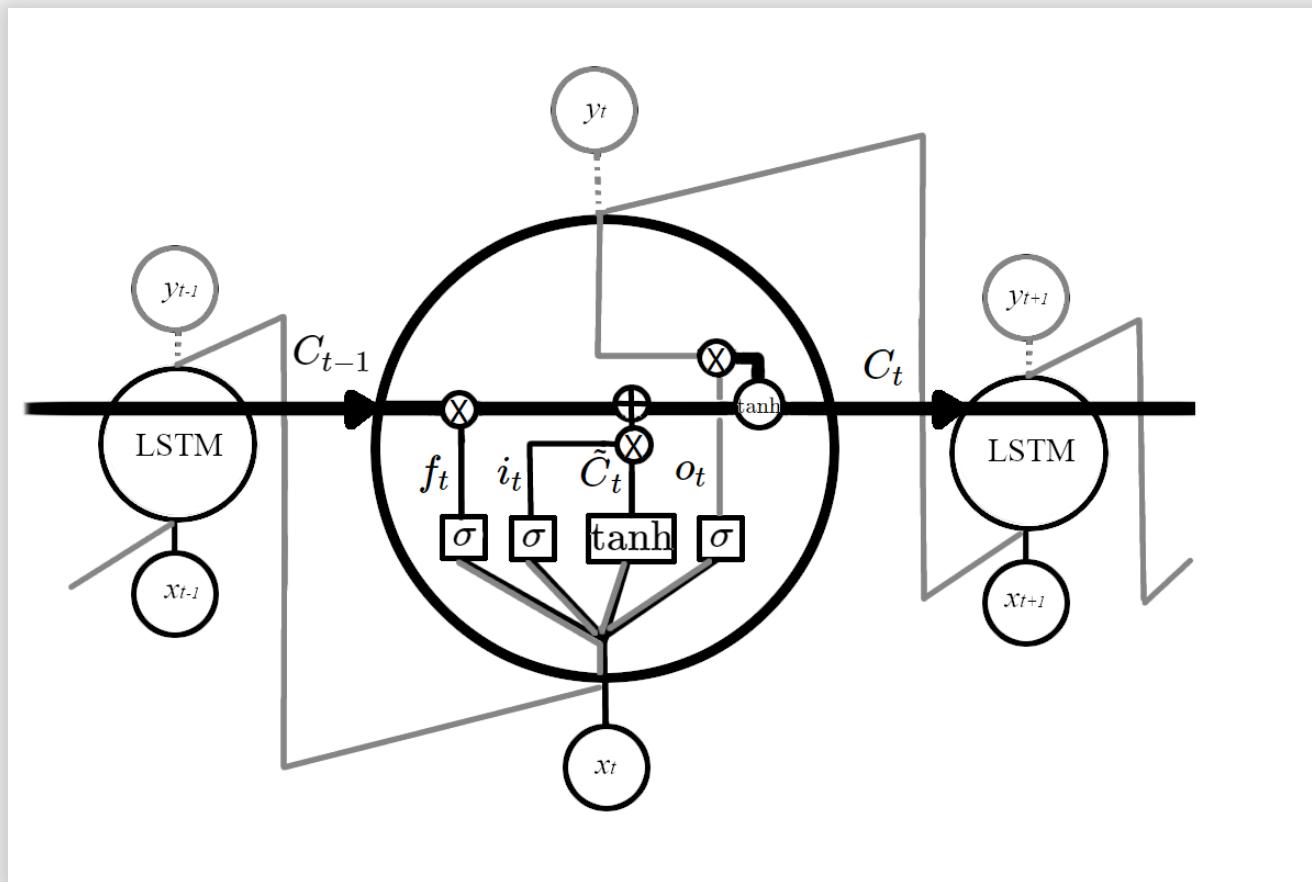
$$i_t = \sigma(W_{0i} + X_t \cdot W_{1i} + \hat{Y}_{t-1} \cdot W_{2i})$$

$$o_t = \sigma(W_{0o} + X_t \cdot W_{1o} + \hat{Y}_{t-1} \cdot W_{2o})$$

$$\tilde{C}_t = \tanh(W_{0c} + X_t \cdot W_{1c} + \hat{Y}_{t-1} \cdot W_{2c})$$

Where  $\sigma$  is the sigmoid function, shrinking any input to be between 0 and 1.

Or if you prefer a diagram



## Yelp with LSTM

```
from tensorflow.keras.layers import LSTM

lstm = Sequential([
    Embedding(max_features + 1, words_embed_dim),
    LSTM(n_cells, return_sequences=True),
    LSTM(1, activation='sigmoid')
])

## WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it does:

lstm.compile(loss = 'binary_crossentropy',
              optimizer='adam', metrics='accuracy')
```



So if RNN layer would have  $l$  parameters, LSTM would have...?

```
lstm.summary()

## Model: "sequential_5"
## -----
##   Layer (type)          Output Shape       Param #
##   =====            ======           =====
##   embedding_4 (Embedding)    (None, None, 50)      500050
##   lstm (LSTM)             (None, None, 10)      2440
##   lstm_1 (LSTM)            (None, 1)          48
## -----
## Total params: 502,538
## Trainable params: 502,538
## Non-trainable params: 0
## -----
```

```
history = lstm.fit(X_train, y_train, validation_split=0.1, callbacks=callbacks,
                    batch_size=batch_size, epochs=epochs)
```

```
lstm.evaluate(X_test, y_test)
```

```
## [0.22240985929965973, 0.9227678775787354]
```

# Beyond the black box

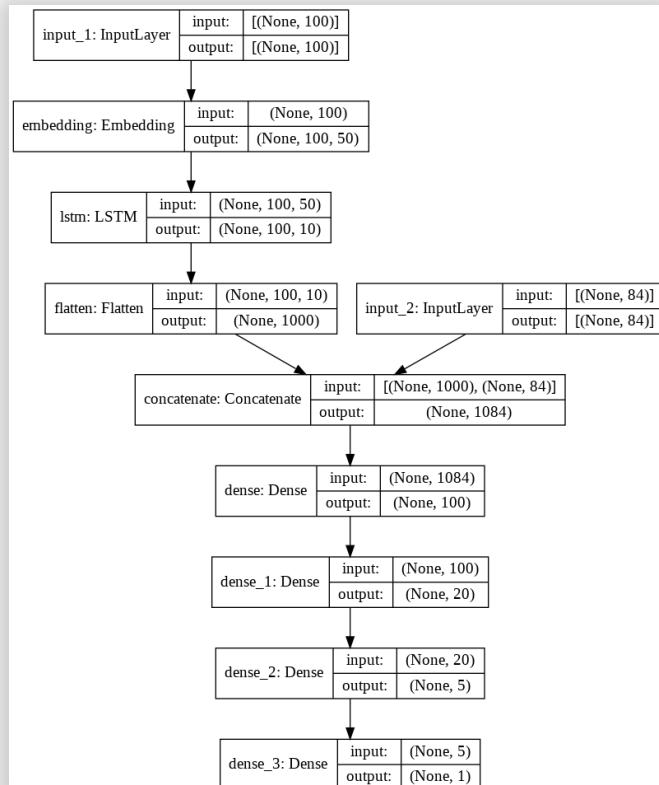
Sending "probes" can help us get what the LSTM is doing. By standardizing an LSTM neuron output to 0-1 and translating this to a cold-hot palette as was done [here](#) we can see what turns it on/off.

Here I checked the first neuron for a few test reviews (see full notebook in Colab):

the p and i ventured to his old grounds for lunch today the and on 16th st and left me with little to ask for before coming here i had a that 5 diners were it's so nice and they had misters we get two different servers bringing us stuff talk about service and i ask the one waitress for recommendations i didn't mediocre burgers if you are in the area and want a fast food burger fatburger is a better bet than wendy's but it is nothing to go out of your way for not at all impressed our server was not very happy to be there food was very sub par and it was way to crowded not the good kind i crowded where you you feel a fight may break out also if the chocolate fountain is their golden gem why is it ok for people to dip the strawberry lick off the chocolate and re i wish i would have read megan review before i decided to cancel my dinner reservations because i was offered a comp dinner at n ntwo of my girlfriend nightclub that night we wanted to go to haze and xs so we declined the promoter by the way his name is offers us a comp dinner at restaurant at crystal a large selection of food from all over the world great atmosphere and ambiance quality of food is on par with a 5 star hotel but did not have lobster an i know i know a review for subway come on but i have to say that the service at this subway is top notch the staff is very friendly and always goes out o say there but i will be back again on my lunch breaks  
we came in for a pre bachelor party madness meal and i have to say it was one of the best dining experiences i've ever had n n started with some cocktail course we split the 32 oz steak in my top five steaks of all time a split order of king crab legs and lobster tail all of it was amazing it came with a side o cheap and delicious i eat here about once a week because it fills me up for cheap price and the people who work there are really fun it's like a subway s and best in town for that price  
as good as it gets for a vegas buffet better than all the other buffets out there on the strip the weekend seafood buffet which also includes beer and wine pittsburgh bound part x n n for part go to http www yelp com biz brothers company pittsburgh q n the name is famous in pittsburgh and although their mouth filled with great mozzarella cheese i grabbed a loaf and we picked at it while we walked around my mom and i half of it without an eye n n if you a

# Play with it!

For example, combining LSTM with the old MLP to predict women's shoes price:



# Quora!

The [Quora Question Pairs](#) dataset has over 400K pairs of questions labeled for whether they're duplicate or not:

```
quora_df = pd.read_csv('../data/quora_small.csv')
```

```
quora_df[['question1', 'question2', 'is_duplicate']].head()
```

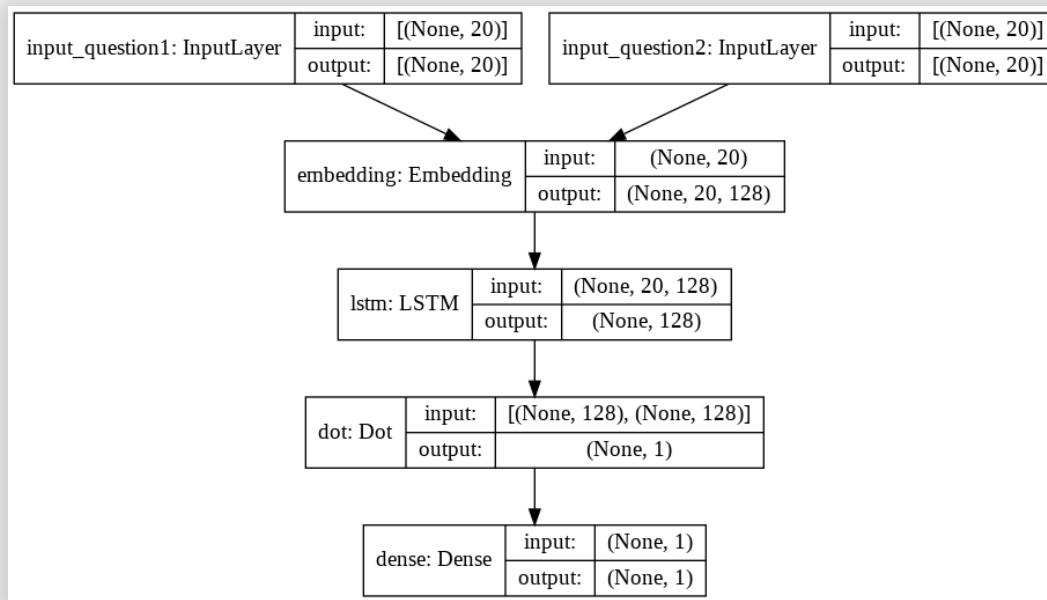
```
##                                     question1 ... is_duplicate
## 0  What is the step by step guide to invest in sh... ...      0
## 1  What is the story of Kohinoor (Koh-i-Noor) Dia... ...      0
## 2  How can I increase the speed of my internet co... ...      0
## 3  Why am I mentally very lonely? How can I solve... ...      0
## 4  Which one dissolve in water quickly sugar, salt... ...      0
##
## [5 rows x 3 columns]
```

```
quora_df['is_duplicate'].mean()
```

```
## 0.38
```

# Siamese Recurrent Architecture

Deciding whether two "things" are the same or not is a challenge to any ML system. NN are a great framework for this kind of task through a Siamese architecture, and LSTMs are great for text. We're following RStudio's [AI Blog](#).



```

from tensorflow.keras import Model
from tensorflow.keras.layers import Dot, Input
from tensorflow.keras.regularizers import l2

max_features = 50000
tokenizer = text.Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(pd.concat([quora_df['question1']], quora_df

question1 = tokenizer.texts_to_sequences(quora_df['question1']))
question2 = tokenizer.texts_to_sequences(quora_df['question2'])

seq_len = 20
question1_padded = sequence.pad_sequences(question1, maxlen=seq_len)
question2_padded = sequence.pad_sequences(question2, maxlen=seq_len)

input1 = Input(shape = (seq_len,), name = "input_question1")
input2 = Input(shape = (seq_len, ), name = "input_question2")

word_embedder = Embedding(
    input_dim = max_features + 2, output_dim = 128,
    input_length = seq_len,
    embeddings_regularizer = l2(0.0001)
)

seq_embedder = LSTM(units = 128,
    kernel_regularizer = l2(0.0001)
)

```

```
vector1 = seq_embedder(word_embedder(input1))
vector2 = seq_embedder(word_embedder(input2))

cosine_similarity = Dot(axes=1)([vector1, vector2])

output = Dense(1, activation = 'sigmoid')(cosine_similarity)

model = Model(inputs = [input1, input2], outputs = output)
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', me
```

```
# messy print, see notebook in Colab
model.summary()
```

```

train_sample, val_sample = train_test_split(quora_df.index, test_size=0.2, random_state=42)

train_question1_padded = question1_padded[train_sample,]
train_question2_padded = question2_padded[train_sample,]
train_is_duplicate = quora_df['is_duplicate'][train_sample]

val_question1_padded = question1_padded[val_sample,]
val_question2_padded = question2_padded[val_sample,]
val_is_duplicate = quora_df['is_duplicate'][val_sample]

history = model.fit(
    [train_question1_padded, train_question2_padded],
    train_is_duplicate,
    batch_size = 64,
    epochs = 100,
    callbacks = [EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    validation_data = ([val_question1_padded, val_question2_padded],
    )

model.evaluate(([val_question1_padded, val_question2_padded], val_is_duplicate))
## [0.43971124291419983, 0.8325954079627991]

```

Save the model, put this function in the most basic Shiny/Dash app, and...

```
def predict_question_pairs(model, tokenizer, q1, q2):
    q1 = tokenizer.texts_to_sequences([q1])
    q2 = tokenizer.texts_to_sequences([q2])

    q1 = sequence.pad_sequences(q1, maxlen=seq_len, value = max_f
    q2 = sequence.pad_sequences(q2, maxlen=seq_len, value = max_f

    return model.predict([q1, q2])[0][0]

predict_question_pairs(
    model,
    tokenizer,
    "What does a LSTM do?",
    "How does a LSTM work?"
)

## 0.5638776
```