MSBA 6460: Advanced AI for Business Applications

Summer 2022, Mochen Yang

Recurrent Neural Network (RNN)

Table of contents

Setup: Import Data and Preprocess Text

In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

We will use a sentiment classification dataset on UCI. Download this dataset and import it. Create two numpy arrays to store the texts and labels separately.

In [2]:
```
text = []
label = []
for line in open("../datasets/sentiment.txt"):
    line = line.rstrip('\n').split('\t')
    text.append(line[0])
    label.append(int(line[1]))
text = np.array(text)
label = np.array(label)
```

Here we use the `TextVectorization()` function in `keras` to complete basic text processing tasks. See the function documentation here. In particular, you can control the following things:

- tokenization (split)
- lowercasing and remove puctuation (standardize)
- optionally generate ngrams
- turn into integer representation (output_mode = 'int')
- whether to pad texts of different length to the same sequence length (output_sequence_length)

In [3]:
```
vectorize_layer = keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens = None,
    standardize = 'lower_and_strip_punctuation',
    split = 'whitespace',
    ngrams = None,
    output_mode = 'int',
    output_sequence_length = None
)
```

In [4]:
```
# apply it to the text data with "adapt"
vectorize_layer.adapt(text)
```

In [5]:
```
# check preprocessing results, such as vocabulary,
vectorize_layer.get_vocabulary()
```

Out[5]:
```
['',
 '[UNK]',
 'the',
 'and',
 'i',
 'a',
 'is',
 'to',
 'it',
 'this',
 'of',
 'was',
 'in',
 'for',
 'not',
 'that',
 'with',
 'my',
 'very',
 'good',
 'on',
 'great',
 'you',
 'but',
 'have',
 'are',
 'movie',
 'as',
 'so',
 'phone',
 'film',
 'its',
 'be',
 'all',
 'one',
 'had',
 'at',
 'food',
 'like',
 'just',
 'place',
 'time',
 'were',
 'service',
 'an',
 'really',
 'if',
 'from',
 'there',
 'they',
 'bad',
 'we',
 'well',
 'out',
 'has',
 'dont',
 'about',
 'would',
 'your',
 'or',
 'no',
 'only',
 'by',
 'best',
```

```
 'ever',
 'even',
 'here',
 'also',
 'will',
 'back',
 'up',
 'when',
 'me',
 'than',
 'more',
 'quality',
 'go',
 'what',
 'love',
 'ive',
 'which',
 'made',
 'he',
 'can',
 'because',
 'product',
 'im',
 'how',
 'too',
 'get',
 'work',
 'their',
 'some',
 'works',
 'nice',
 'could',
 'better',
 'any',
 'excellent',
 'after',
 'never',
 'do',
 'recommend',
 'much',
 'been',
 'who',
 'use',
 'our',
 'did',
 'again',
 'sound',
 'other',
 'think',
 'his',
 'headset',
 'first',
 'battery',
 'way',
 'them',
 'see',
 'make',
 'didnt',
 'pretty',
 'acting',
 'most',
 'worst',
 'still',
 'now',
```

```
'got',
'does',
'say',
'over',
'enough',
'characters',
'two',
'little',
'everything',
'every',
'ear',
'disappointed',
'am',
'thing',
'then',
'price',
'being',
'waste',
'these',
'right',
'people',
'going',
'2',
'terrible',
'real',
'off',
'minutes',
'definitely',
'case',
'amazing',
'movies',
'money',
'look',
'new',
'know',
'experience',
'cant',
'came',
'both',
'into',
'wont',
'story',
'many',
'her',
'friendly',
'few',
'doesnt',
'worth',
'used',
'poor',
'plot',
'piece',
'films',
'far',
'us',
'seen',
'years',
'wonderful',
'while',
'want',
'restaurant',
'quite',
'nothing',
'lot',
```

```
'long',
'10',
'she',
'script',
'life',
'happy',
'always',
'wasnt',
'highly',
'give',
'found',
'delicious',
'anyone',
'watching',
'times',
'character',
'worked',
'vegas',
'take',
'should',
'probably',
'loved',
'fine',
'easy',
'car',
'bought',
'awful',
'around',
'another',
'absolutely',
'went',
'since',
'screen',
'same',
'item',
'however',
'horrible',
'funny',
'comfortable',
'camera',
'buy',
'before',
'awesome',
'where',
'watch',
'totally',
'thought',
'those',
'things',
'stars',
'staff',
'scenes',
'makes',
'impressed',
'find',
'eat',
'down',
'couldnt',
'cool',
'charger',
'big',
'though',
'talk',
'such',
```

```
 'small',
 'slow',
 'show',
 'part',
 'night',
 'music',
 'last',
 'job',
 'fantastic',
 'end',
 'come',
 'cast',
 'actors',
 'stupid',
 'said',
 'perfect',
 'ordered',
 'old',
 'must',
 'kind',
 'family',
 'day',
 'cheap',
 'bluetooth',
 'black',
 'beautiful',
 'thats',
 'sure',
 'performance',
 'overall',
 'next',
 'fresh',
 'feel',
 'chicken',
 'avoid',
 'actually',
 '5',
 'try',
 'tried',
 'sucks',
 'simply',
 'scene',
 'salad',
 'reception',
 'problems',
 'problem',
 'pizza',
 'order',
 'menu',
 'line',
 'least',
 'interesting',
 'id',
 'hard',
 'felt',
 'especially',
 'done',
 'bit',
 'between',
 'writing',
 'worse',
 'without',
 'wait',
 'taste',
```

```
 'steak',
 'special',
 'purchase',
 'man',
 'low',
 'looks',
 'liked',
 'ill',
 'hear',
 'gets',
 'fit',
 'fast',
 'expect',
 'everyone',
 'enjoyed',
 'either',
 'each',
 'customer',
 'completely',
 'coming',
 'charge',
 'cell',
 'calls',
 'away',
 'anything',
 'almost',
 '1',
 'year',
 'working',
 'whole',
 'white',
 'using',
 'through',
 'sushi',
 'short',
 'server',
 'seriously',
 'rather',
 'left',
 'hour',
 'flavor',
 'different',
 'dialogue',
 'device',
 'clear',
 'call',
 'bland',
 'watched',
 'volume',
 'unfortunately',
 'understand',
 'took',
 'tell',
 'super',
 'started',
 'side',
 'several',
 'saw',
 'return',
 'put',
 'plug',
 'play',
 'perfectly',
 'may',
```

```
 'looking',
 'incredible',
 'having',
 'getting',
 'full',
 'feeling',
 'fact',
 'extremely',
 'enjoy',
 'disappointing',
 'design',
 'deal',
 'burger',
 'buffet',
 'believe',
 'art',
 'yet',
 'truly',
 'three',
 'theres',
 'tasty',
 'sucked',
 'stay',
 'soon',
 'received',
 'phones',
 'once',
 'need',
 'motorola',
 'mess',
 'mediocre',
 'meal',
 'light',
 'less',
 'kids',
 'huge',
 'hours',
 'hot',
 'fits',
 'ending',
 'during',
 'disappointment',
 'crap',
 'certainly',
 'care',
 'barely',
 'atmosphere',
 '3',
 'wrong',
 'why',
 'wasted',
 'waited',
 'together',
 'strong',
 'selection',
 'sauce',
 'recommended',
 'prices',
 'predictable',
 'pleased',
 'played',
 'picture',
 'own',
 'original',
```

```
'none',
'months',
'kept',
'inside',
'home',
'high',
'gave',
'fun',
'friends',
'effects',
'easily',
'director',
'days',
'cold',
'boring',
'area',
'actor',
'wouldnt',
'wear',
'wanted',
'waitress',
'voice',
'turn',
'trying',
'top',
'table',
'style',
'spot',
'something',
'series',
'seems',
'second',
'sandwich',
'places',
'myself',
'meat',
'lunch',
'lost',
'keep',
'junk',
'him',
'helpful',
'hands',
'half',
'guess',
'glad',
'game',
'fries',
'face',
'dropped',
'drama',
'dishes',
'directing',
'decent',
'couple',
'company',
'clean',
'cannot',
'broke',
'breakfast',
'bar',
'amazon',
'ago',
'weak',
```

```
'waiting',
'verizon',
'value',
'unit',
'under',
'twice',
'tv',
'told',
'today',
'tasted',
'star',
'spicy',
'someone',
'simple',
'set',
'rude',
'quickly',
'quick',
'point',
'plus',
'playing',
'pictures',
'perhaps',
'pay',
'overpriced',
'others',
'oh',
'obviously',
'let',
'jabra',
'itself',
'important',
'human',
'house',
'headsets',
'hate',
'given',
'garbage',
'finally',
'fails',
'expected',
'entire',
'else',
'eating',
'dish',
'dining',
'comes',
'cinematography',
'cinema',
'check',
'buttons',
'beer',
'average',
'audio',
'ask',
'arrived',
'amount',
'although',
'20',
'written',
'world',
'word',
'within',
'whatever',
```

```
            'week',
            'wall',
            'waiter',
            'useless',
            'town',
            'tender',
            'suspense',
            'superb',
            'subtle',
            'store',
            'solid',
            'software',
            'single',
            'sick',
            'served',
            'room',
            'review',
            'reasonable',
            'priced',
            'possible',
            'outside',
            'nokia',
            'needed',
            'mostly',
            'mind',
            'might',
            'mention',
            'maybe',
            'location',
            'literally',
            'lines',
            'later',
            'joy',
            'john',
            'isnt',
            'horror',
            'hope',
            'heart',
            'girl',
            'front',
            'free',
            'feels',
            'editing',
            'drive',
            'cover',
            'course',
            'cooked',
            'comedy',
            'color',
            'classic',
            'chips',
            'charm',
            'cable',
            'bring',
            'brilliant',
            'beyond',
            'authentic',
            'attentive',
            'ambiance',
            'action',
            '30',
            '12',
            'zero',
            'youre',
```

```
'wow',
'wife',
'whether',
'whatsoever',
'warm',
'walked',
'until',
'unless',
'throughout',
'thin',
'theyre',
'terrific',
'sweet',
'songs',
'sometimes',
'signal',
'shrimp',
'setting',
'seemed',
'seem',
'seeing',
'sat',
'running',
'ridiculous',
'reviews',
'rest',
'rent',
'reason',
'rating',
'rare',
'range',
'potato',
'portrayal',
'poorly',
'player',
'plastic',
'pho',
'performances',
'particular',
'note',
'mistake',
'management',
'making',
'lovely',
'loud',
'lots',
'looked',
'longer',
'leave',
'large',
'lacks',
'lacking',
'joke',
'internet',
'instead',
'idea',
'holes',
'hold',
'hit',
'history',
'hilarious',
'headphones',
'happened',
'gives',
```

```
'flick',
'fish',
'features',
'feature',
'extra',
'exactly',
'elsewhere',
'ears',
'dinner',
'difficult',
'despite',
'dead',
'damn',
'costs',
'considering',
'clever',
'casting',
'cases',
'business',
'bother',
'book',
'believable',
'become',
'basically',
'bars',
'annoying',
'above',
'Â—',
'youll',
'youd',
'wish',
'weeks',
'water',
'utterly',
'usual',
'unbelievable',
'turned',
'trip',
'trash',
'total',
'tom',
'thumbs',
'themselves',
'thai',
'tasteless',
'tables',
'sturdy',
'stuff',
'soundtrack',
'silent',
'sides',
'shots',
'shot',
'shipping',
'servers',
'seated',
'seafood',
'satisfied',
'samsung',
'sad',
'roles',
'recently',
'recent',
'reasonably',
```

```
 'reading',
 'razr',
 'rate',
 'provided',
 'portions',
 'please',
 'plain',
 'period',
 'pathetic',
 'pasta',
 'party',
 'particularly',
 'owners',
 'owned',
 'options',
 'ok',
 'often',
 'offers',
 'minute',
 'memorable',
 'mean',
 'living',
 'level',
 'leather',
 'lead',
 'lame',
 'lacked',
 'involved',
 'incredibly',
 'impressive',
 'imagination',
 'ice',
 'holds',
 'hitchcock',
 'havent',
 'hand',
 'graphics',
 'goes',
 'generally',
 'games',
 'fried',
 'form',
 'forever',
 'favorite',
 'fans',
 'fan',
 'fall',
 'eyes',
 'etc',
 'ended',
 'empty',
 'embarrassing',
 'eaten',
 'dry',
 'dirty',
 'direction',
 'dessert',
 'deserves',
 'data',
 'cult',
 'consider',
 'connection',
 'computer',
 'close',
```

```
'clarity',
'choice',
'child',
'charging',
'buying',
'budget',
'break',
'bread',
'blue',
'belt',
'below',
'beef',
'beat',
'bargain',
'bacon',
'audience',
'asked',
'arent',
'anytime',
'along',
'ability',
'8',
'40',
'4',
'yummy',
'yourself',
'young',
'yes',
'worthless',
'words',
'wonderfully',
'wings',
'wine',
'wind',
'weird',
'website',
'wasting',
'visual',
'visit',
'vibe',
'vegetables',
'unreliable',
'type',
'turns',
'true',
'trouble',
'treo',
'towards',
'touch',
'torture',
'tool',
'tmobile',
'thriller',
'thoroughly',
'third',
'thinking',
'theater',
'ten',
'tea',
'tale',
'takes',
'tacos',
'support',
'strip',
```

```
'storyline',
'stories',
'steaks',
'station',
'starts',
'space',
'soup',
'song',
'son',
'somewhat',
'shows',
'showed',
'share',
'sets',
'serious',
'sense',
'sending',
'seller',
'says',
'salmon',
'run',
'rolls',
'role',
'rocks',
'ringtones',
'rice',
'results',
'replace',
'remember',
'red',
'ready',
'ray',
'rated',
'purchased',
'pull',
'production',
'previous',
'pretentious',
'premise',
'power',
'pork',
'pleasant',
'plays',
'plantronics',
'phoenix',
'person',
'passed',
'parts',
'palm',
'pair',
'paid',
'ones',
'occasionally',
'number',
'network',
'needs',
'nearly',
'nasty',
'moving',
'mouth',
'missed',
'mic',
'mexican',
'massive',
```

```
 'market',
 'manager',
 'main',
 'loves',
 'live',
 'list',
 'likes',
 'lightweight',
 'lg',
 'lasts',
 'killer',
 'keyboard',
 'italian',
 'issues',
 'intelligence',
 'insult',
 'instructions',
 'indeed',
 'included',
 'immediately',
 'husband',
 'honestly',
 'honest',
 'hes',
 'heard',
 'hated',
 'happier',
 'hair',
 'guy',
 'greatest',
 'gotten',
 'gone',
 'genuine',
 'gem',
 'fx',
 'forget',
 'follow',
 'folks',
 'flat',
 'five',
 ...]
```

In [16]:
```
len(vectorize_layer.get_vocabulary())
```

Out[16]:
```
5404
```

Note that the vocabulary contains two special tokens:

- '' is called the **padding token**. It is an empty string, correspond to index 0, that can be used to pad texts of different lengths to the same sequence length. When this text vectorization layer is applied to a set of texts, it automatically perform padding (to the longest sequence).
- '[UNK]' is called the **Out-of-Vocabulary token**. It is used to represent any word that does not appear in the vocabulary.

The next block of code is for demonstration purpose only - it is typically NOT needed in actual model building. We want to use `vectorize_layer` to process some texts and represent them as indices in vocabulary.

In [6]:
```
# now use it to process some text
```

```
input_text = [['very good movie'], ['Mochen Yang']]
vectorize_layer(input_text)
```

Out[6]:
```
<tf.Tensor: shape=(2, 3), dtype=int64, numpy=
array([[18, 19, 26],
       [ 1,  1,  0]], dtype=int64)>
```

Basic RNN Model

Why Use RNN for Language-Related Machine Learning Tasks?

1. Natural language is a sequence (of words). RNNs are specifically designed to handle sequential data;
2. It is easy to use word embeddings as inputs to RNNs, further boosting the ability to incorporate semantic information;
3. Different pieces of texts can have different lengths. RNNs are built to deal with variable-length data (via parameter sharing - discussed later);
4. Different NLP tasks require different network architectures. RNNs are versatile enough to accommodate those.

Common RNN Architectures for NLP

Many Inputs, One Output

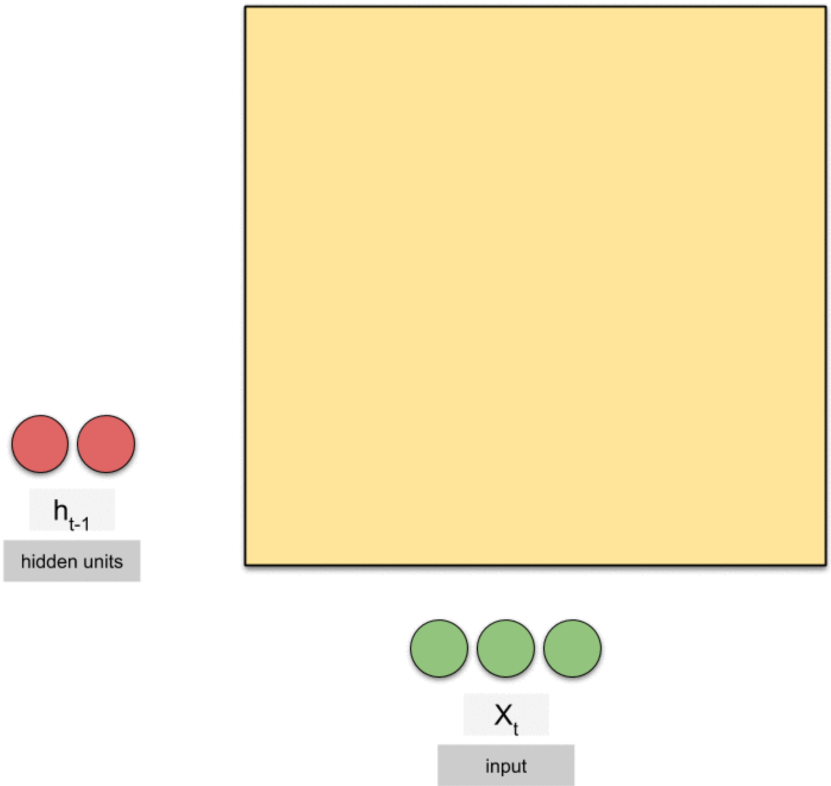
- **Suitable Task**: Classification and Numeric Prediction
- **Typical Architecture** is discussed and demonstrated in this notebook.

Many Inputs, Many Outputs

- **Suitable Tasks**:
  - Machine Translation
  - Document Summarization
  - Conversational Model (Q&A, Chatbot, etc.)
- **Typical Architecture**: encoder-decoder architecture (discussed in a different notebook).

Animated Illustration of a Simple RNN Unit

image credit: https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45



How does a Simple RNN Unit work?

The inner workings of a simple RNN unit can be represented either as a recurrent equation:

$$h_t = f(h_{t-1}, X_t, \Theta)$$

- h_t is the hidden states at time step t;
- X_t is the input (usually a vector) at time step t;
- Θ is (usually matrices) of weights to be learned. Notice that Θ does not have any subscript, which means that it is the same set of parameters for **every time step** t. This is the key idea of **parameter sharing**!. Without parameter sharing, RNNs would not be able to handle texts of different lengths.
- $f()$ is the activation function that governs how past states (h_{t-1}) combine with current information (X_t) to determine current states of the network. Typically this is the hyperbolic tangent $tanh$ function (see here for technical definition).

Importantly, the recurrent relationship discussed here is universal to RNNs. Even for more complex types of RNNs, we are still trying to characterize how the hidden states at time $t-1$, combined with input received at time t, jointly determine the hidden states at time t . In the context of NLP, you can conceptually think of this recurrent relationship as a "reading" process: the neural network is reading a piece of text word by word, and its hidden states are "updated" after the ingestion of every word.

Build Simple RNN in Keras

Now, let's actually build a basic RNN model, by stacking together the text processing layer, an embedding layer, and an RNN layer. Documentation to RNN layer is here.

In [11]:
```
model_rnn = keras.Sequential()

model_rnn.add(vectorize_layer)

model_rnn.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 64,
    mask_zero = True
))

model_rnn.add(keras.layers.SimpleRNN(128)) # see note below

model_rnn.add(keras.layers.Dense(1, activation = 'sigmoid'))
```

For classification task, because we only need a single output at the end of the entire RNN, you only need to specify the total number of RNN units in the `SimpleRNN` function. You might be wondering - don't we need the input_size parameter? It is not necessary here because the preceding embedding layer automatically informs the RNN layer that each input text will have variable length (padded with special value 0 to max sequence length, and padding value 0 is automatically masked) and each word is represented by 64 dimensions in this example (i.e., the output dimension of the word embedding).

Question: Looking at the embedding layer, which method is it using to train the word embeddings (e.g., skip-gram, continuous bag-of-words, both, neither)? This method/strategy of training a neural network model has a special name (you learned about it in Yicheng's class), what is it?

In [12]:
```
# configure training / optimization
model_rnn.compile(loss = keras.losses.BinaryCrossentropy(),
                  optimizer='adam',
                  metrics=['accuracy'])
```

In [13]:
```
# training with 20% validation and 10 epochs.
model_rnn.fit(x = text, y = label, validation_split = 0.2,
              epochs=10, batch_size = 32)
```

```
Epoch 1/10
75/75 [==============================] - 1s 11ms/step - loss: 0.6897 - accur
acy: 0.5579 - val_loss: 0.6677 - val_accuracy: 0.6150
Epoch 2/10
75/75 [==============================] - 1s 8ms/step - loss: 0.4860 - accura
cy: 0.8350 - val_loss: 0.5263 - val_accuracy: 0.7500
Epoch 3/10
75/75 [==============================] - 1s 8ms/step - loss: 0.1925 - accura
cy: 0.9413 - val_loss: 0.4953 - val_accuracy: 0.7850
Epoch 4/10
75/75 [==============================] - 1s 7ms/step - loss: 0.0818 - accura
cy: 0.9792 - val_loss: 0.5727 - val_accuracy: 0.7783
Epoch 5/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0389 - accura
cy: 0.9925 - val_loss: 0.6064 - val_accuracy: 0.7783
Epoch 6/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0156 - accura
cy: 0.9983 - val_loss: 0.7091 - val_accuracy: 0.7867
Epoch 7/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0104 - accura
cy: 0.9996 - val_loss: 0.7453 - val_accuracy: 0.7767
Epoch 8/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0048 - accura
cy: 1.0000 - val_loss: 0.8131 - val_accuracy: 0.7683
Epoch 9/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0033 - accura
cy: 1.0000 - val_loss: 0.8540 - val_accuracy: 0.7717
Epoch 10/10
75/75 [==============================] - 1s 8ms/step - loss: 0.0024 - accura
cy: 1.0000 - val_loss: 0.8836 - val_accuracy: 0.7733
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x1b128e90280>
```

In [15]:
```
model_rnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization_1 (TextVe (None, None)              0
_________________________________________________________________
embedding (Embedding)        (None, None, 64)          345856
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 128)               24704
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 370,689
Trainable params: 370,689
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
# try to make some predicitons
model_rnn.predict([['I hate this meal!'], ['I love this restaurant']])
```

Out[14]:
```
array([[0.03613052],
       [0.999736  ]], dtype=float32)
```

The Long-Term Dependency Problem

So Why Do We Need Anything More than Simple

RNN?

Here is a highly simplified (non-rigorous) derivation to help you understand. Suppose there is only 1 parameter, w, to learn, and the activitaion function, $f()$, is linear, and there is a constant input value of 0. Considering realistic inputs, non-linear activiation functions, and parameter matrices would make the derivation too hard to track but the underlying intuitions are the same.

Think about how the hidden states of an RNN unit change over time:

$$h_t = w \cdot h_{t-1}$$

in other words,

$$h_t = w^t \cdot h_0$$

Now, the gradient of h_t with respect to parameter w is

$$\frac{dh_t}{dw} = tw^{t-1}h_0$$

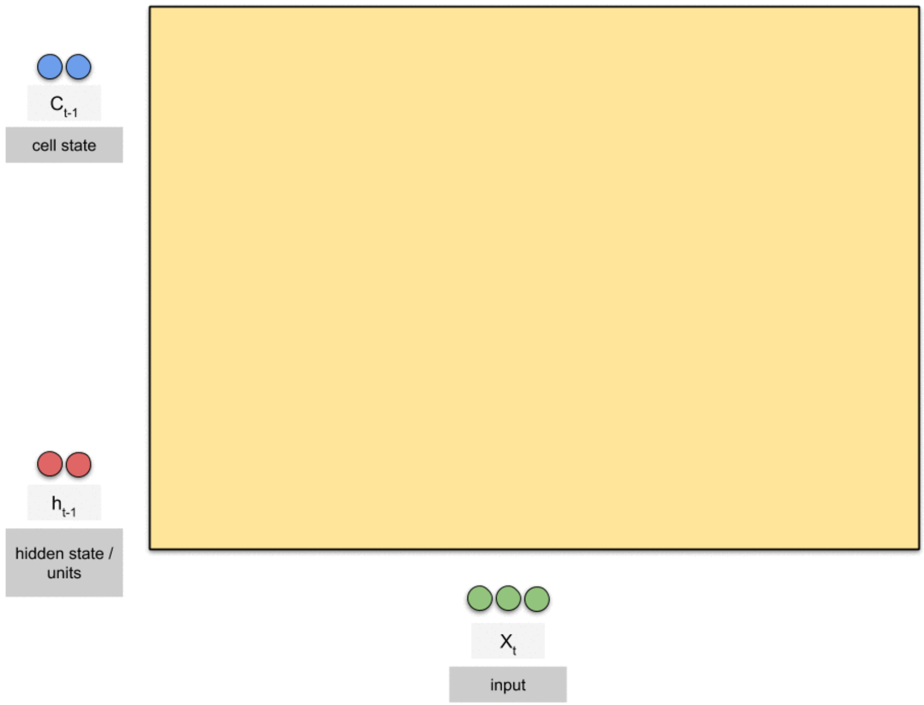
For a long sequence (i.e., t is large), the **gradient will explode** (go to ∞) even if w is just slightly larger than 1, and the **gradient will vanish** (go to 0) even if w is just slightly smaller than 1. This makes training RNN to learn from long sequences very hard. If the gradient is extremely large, gradient descent with any regular learning rate will be highly unstable. If the gradient is extremely small, gradient descent won't "descent" much at all.

Question: is this a problem for traditional, feed-forward neural networks with many hidden layers? If not, why not?

Long Short-Term Memory (LSTM) Model

Animated Illustration of a Single LSTM Unit

image credit: https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45



How does a Single LSTM Unit/Cell Work?

Here is a simplified version of how LSTM works to convey the key intuitions. A more technical description can be found here.

A LSTM unit has takes an input X_t, process it with a series of operations via **three "gates"** (respectively input gate, forget gate, and output gate), and then output h_t. A LSTM also has an **internal cell state**, C_t, which is different from the network's hidden state h_{t-1}.

- **forget gate**: $forget_t = sigmoid(X_t, h_{t-1}, \Theta_{forget})$
- **input gate**: $input_t = sigmoid(X_t, h_{t-1}, \Theta_{input})$
- **output gate**: $output_t = sigmoid(X_t, h_{t-1}, \Theta_{output})$

Notice: The inner workings of the three gates are almost identical - each one applies a sigmoid function over the combination of input X_t and previous hidden state h_{t-1}, with its own set of parameters. Therefore, the best way to think about the purpose of these gates is that they act as "weights" (between 0 and 1, due to the sigmoid function).

- **Update internal cell state**: $C_t = forget_t \cdot C_{t-1} + input_t \cdot tahn(X_t, h_{t-1}, \Theta)$

Notice: This is nothing but a weighted average! The forget gate controls how much information from LSMT's previous internal state gets passed on to current internal state, and the input gate controls how much information from the new input gets passed on to current internal state. The $tahn(X_t, h_{t-1}, \Theta)$ function, on its own, is exactly the same as in the simple RNN case. This design of internal cell state also has a fancy name - "Constant Error Carousel" (CEC).

Question: does this remind you of anything you learned from Prof. Xuan Bi's class on time series? Hint: think of $forget_t \cdot C_{t-1}$ as historical information and

$input_t \cdot tahn(X_t, h_{t-1}, \Theta)$ as new information.

- **Produce output**: $h_t = output_t \cdot tahn(C_t)$

The output gate controls how much information from the updated internal state gets passed on to current hidden state.

In summary, though it might seem complicated, the above steps are still trying to compute an updated hidden state h_t based on the previous hidden state h_{t-1} and the new input X_t. The three gates act like weights to control information flow, and the internal cell state *remembers* information from the past. This is the intuition why LSTM can mitigate the vanishing gradient problem.

However, the LSTM design does not necessarily address the exploding gradient problem. It's a bit technical to explain why, and I refer you to this blog post for more details if you are interested.

Build RNN with LSTM Units in Keras

In [6]:
```
model_lstm = keras.Sequential()

model_lstm.add(vectorize_layer)

model_lstm.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 64,
    mask_zero = True
))

model_lstm.add(keras.layers.LSTM(128))

model_lstm.add(keras.layers.Dense(1, activation = 'sigmoid'))
```

In [7]:
```
# configure training / optimization
model_lstm.compile(loss = keras.losses.BinaryCrossentropy(),
                   optimizer='adam',
                   metrics=['accuracy'])
```

In [8]:
```
# training with 20% validation and 10 epochs.
model_lstm.fit(x = text, y = label, validation_split = 0.2,
               epochs=10, batch_size = 32)
```

```
Epoch 1/10
75/75 [==============================] - 11s 75ms/step - loss: 0.6773 - accu
racy: 0.5966 - val_loss: 0.5655 - val_accuracy: 0.7583
Epoch 2/10
75/75 [==============================] - 2s 25ms/step - loss: 0.4064 - accur
acy: 0.8712 - val_loss: 0.4680 - val_accuracy: 0.7917
Epoch 3/10
75/75 [==============================] - 2s 27ms/step - loss: 0.1905 - accur
acy: 0.9502 - val_loss: 0.5194 - val_accuracy: 0.7967
Epoch 4/10
75/75 [==============================] - 2s 26ms/step - loss: 0.0885 - accur
acy: 0.9784 - val_loss: 0.5099 - val_accuracy: 0.8083
Epoch 5/10
75/75 [==============================] - 2s 25ms/step - loss: 0.0625 - accur
acy: 0.9840 - val_loss: 0.5844 - val_accuracy: 0.8167
Epoch 6/10
75/75 [==============================] - 2s 26ms/step - loss: 0.0340 - accur
acy: 0.9939 - val_loss: 0.6914 - val_accuracy: 0.8067
Epoch 7/10
75/75 [==============================] - 2s 25ms/step - loss: 0.0150 - accur
acy: 0.9981 - val_loss: 0.8738 - val_accuracy: 0.7833
Epoch 8/10
75/75 [==============================] - 2s 25ms/step - loss: 0.0144 - accur
acy: 0.9974 - val_loss: 1.0673 - val_accuracy: 0.7833
Epoch 9/10
75/75 [==============================] - 2s 26ms/step - loss: 0.0246 - accur
acy: 0.9925 - val_loss: 0.8362 - val_accuracy: 0.8083
Epoch 10/10
75/75 [==============================] - 2s 26ms/step - loss: 0.0111 - accur
acy: 0.9968 - val_loss: 0.8860 - val_accuracy: 0.8033
```

Out[8]:

```
<tensorflow.python.keras.callbacks.History at 0x179a9615a00>
```

In [36]:

```python
# try to make some predicitons
model_lstm.predict([['I hate this meal!'], ['I love this restaurant']])
```

Out[36]:

```
array([[0.02249685],
       [0.99999833]], dtype=float32)
```

In [10]:

```python
model_lstm.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization (TextVect (None, None)              0
_________________________________________________________________
embedding (Embedding)        (None, None, 64)          345856
_________________________________________________________________
lstm (LSTM)                  (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 444,801
Trainable params: 444,801
Non-trainable params: 0
_________________________________________________________________
```

Gated Recurrent Unit (GRU) Model

Animated Illustration of a Single GRU

image credit: https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

How does a Single GRU Work?

A GRU has two gates: an update gate and a reset gate.

- **Update gate**: $update_t = sigmoid(X_t, h_{t-1}, \Theta_{update})$
- **Reset gate**: $reset_t = sigmoid(X_t, h_{t-1}, \Theta_{reset})$
- **Produce output**: $h_t = update_t \cdot h_{t-1} + (1 - update_t) \cdot tahn(X_t, reset_t \cdot h_{t-1}, \Theta)$

Notice: Just like in LSTM, the two gates of GRU are weights. The update gate controls how much information from previous hidden state h_{t-1} gets passed on to current hidden state h_t, and the reset gate controls how much information from previous hidden state gets to be combined with current input X_t.

Interestingly, even though GRU seems simpler and works basically as well as LSTM, it was proposed later than LSTM.

Build RNN with GRUs in Keras

In [11]:
```
model_gru = keras.Sequential()

model_gru.add(vectorize_layer)

model_gru.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 64,
    mask_zero = True
```

```
))

model_gru.add(keras.layers.GRU(128))

model_gru.add(keras.layers.Dense(1, activation = 'sigmoid'))
```

In [12]:

```
# configure training / optimization
model_gru.compile(loss = keras.losses.BinaryCrossentropy(),
                  optimizer='adam',
                  metrics=['accuracy'])
```

In [13]:

```
# training with 20% validation and 10 epochs.
model_gru.fit(x = text, y = label, validation_split = 0.2,
              epochs=10, batch_size = 32)
```

```
Epoch 1/10
75/75 [==============================] - 4s 31ms/step - loss: 0.6808 - accur
acy: 0.5511 - val_loss: 0.5101 - val_accuracy: 0.7767
Epoch 2/10
75/75 [==============================] - 2s 22ms/step - loss: 0.3536 - accur
acy: 0.8766 - val_loss: 0.4265 - val_accuracy: 0.8083
Epoch 3/10
75/75 [==============================] - 2s 22ms/step - loss: 0.1402 - accur
acy: 0.9627 - val_loss: 0.4923 - val_accuracy: 0.8150
Epoch 4/10
75/75 [==============================] - 2s 22ms/step - loss: 0.0488 - accur
acy: 0.9879 - val_loss: 0.5205 - val_accuracy: 0.8167
Epoch 5/10
75/75 [==============================] - 2s 22ms/step - loss: 0.0434 - accur
acy: 0.9908 - val_loss: 0.5580 - val_accuracy: 0.8017
Epoch 6/10
75/75 [==============================] - 2s 22ms/step - loss: 0.0416 - accur
acy: 0.9917 - val_loss: 0.6560 - val_accuracy: 0.8100
Epoch 7/10
75/75 [==============================] - 2s 21ms/step - loss: 0.0210 - accur
acy: 0.9951 - val_loss: 0.9197 - val_accuracy: 0.7967
Epoch 8/10
75/75 [==============================] - 2s 21ms/step - loss: 0.0132 - accur
acy: 0.9985 - val_loss: 0.8774 - val_accuracy: 0.7967
Epoch 9/10
75/75 [==============================] - 2s 21ms/step - loss: 0.0082 - accur
acy: 0.9985 - val_loss: 1.0786 - val_accuracy: 0.7883
Epoch 10/10
75/75 [==============================] - 2s 21ms/step - loss: 0.0182 - accur
acy: 0.9941 - val_loss: 1.0432 - val_accuracy: 0.7717
```

Out[13]:

```
<tensorflow.python.keras.callbacks.History at 0x179b3074610>
```

In [40]:

```
# try to make some predicitons
model_gru.predict([['I hate this meal!'], ['I love this restaurant']])
```

Out[40]:

```
array([[0.01605341],
       [0.9999916 ]], dtype=float32)
```

Bidirectional RNN Models

In the context of text classification, an intuitive understanding of forward (i.e., one-directional) RNN is that it "reads" a piece of text from beginning to end, and produces a prediction. By the same analogy, a bidirectional RNN would read the text from beginning to end and also from end to beginning, then produces a prediction.

To illustrate how a bidirectional RNN works, let's consider the simple RNN units as an example. Essentially, instead of keeping one set of hidden states h_t that move forward in time, a bidirectional RNN also keep another set of hidden states g_t that move backward in time. Like this:

$$h_t = f(h_{t-1}, X_t, \Theta_h)$$

$$g_t = f(g_{t+1}, X_t, \Theta_g)$$

Under this general structure, you can replace the simple RNN units with LSTM or GRU, which would give rise to bi-LSTM and bi-GRU models.

Building Bidirectional RNN Model in Keras

Specifically, let's build a bidirectional LSTM model.

In [41]:
```
model_bilstm = keras.Sequential()

model_bilstm.add(vectorize_layer)

model_bilstm.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 64,
    mask_zero = True
))

model_bilstm.add(keras.layers.Bidirectional(keras.layers.LSTM(128)))

model_bilstm.add(keras.layers.Dense(1, activation = 'sigmoid'))
```

In [42]:
```
# configure training / optimization
model_bilstm.compile(loss = keras.losses.BinaryCrossentropy(),
                     optimizer='adam',
                     metrics=['accuracy'])
```

In [43]:
```
# training with 20% validation and 10 epochs.
model_bilstm.fit(x = text, y = label, validation_split = 0.2,
                 epochs = 10, batch_size = 32)
```

```
Epoch 1/10
75/75 [==============================] - 7s 97ms/step - loss: 0.6309 - accur
acy: 0.6704 - val_loss: 0.5082 - val_accuracy: 0.7667
Epoch 2/10
75/75 [==============================] - 3s 40ms/step - loss: 0.3136 - accur
acy: 0.9008 - val_loss: 0.4528 - val_accuracy: 0.7850
Epoch 3/10
75/75 [==============================] - 3s 42ms/step - loss: 0.1349 - accur
acy: 0.9621 - val_loss: 0.4401 - val_accuracy: 0.8117
Epoch 4/10
75/75 [==============================] - 3s 36ms/step - loss: 0.0757 - accur
acy: 0.9808 - val_loss: 0.7320 - val_accuracy: 0.7883
Epoch 5/10
75/75 [==============================] - 3s 38ms/step - loss: 0.0476 - accur
acy: 0.9871 - val_loss: 0.7814 - val_accuracy: 0.7950
Epoch 6/10
75/75 [==============================] - 3s 38ms/step - loss: 0.0322 - accur
acy: 0.9908 - val_loss: 0.7875 - val_accuracy: 0.7900
Epoch 7/10
75/75 [==============================] - 3s 37ms/step - loss: 0.0205 - accur
acy: 0.9954 - val_loss: 0.9245 - val_accuracy: 0.8133
Epoch 8/10
75/75 [==============================] - 3s 36ms/step - loss: 0.0115 - accur
acy: 0.9983 - val_loss: 1.1594 - val_accuracy: 0.8067
Epoch 9/10
75/75 [==============================] - 3s 39ms/step - loss: 0.0070 - accur
acy: 0.9992 - val_loss: 1.3125 - val_accuracy: 0.8083
Epoch 10/10
75/75 [==============================] - 3s 34ms/step - loss: 0.0043 - accur
acy: 0.9992 - val_loss: 1.4194 - val_accuracy: 0.8183
```

Out[43]:
```
<tensorflow.python.keras.callbacks.History at 0x1bcccd34a90>
```

In [44]:
```
# try to make some predicitons
model_bilstm.predict([['I hate this meal!'], ['I love this restaurant']])
```

Out[44]:
```
array([[0.00192887],
       [0.9999987 ]], dtype=float32)
```