**2.2 Keras and mini-batching**

* In this exercise, we want to train Keras using mini-batches.
* However, most deep learning frameworks require that all sequences in the same mini-batch have the **same length**.
  + This is what allows vectorization to work: If you had a 3-word sentence and a 4-word sentence, then the computations needed for them are different (one takes 3 steps of an LSTM, one takes 4 steps) so it's just not possible to do them both at the same time.

**Padding handles sequences of varying length**

* The common solution to handling sequences of **different length** is to use padding. Specifically:
  + Set a maximum sequence length
  + Pad all sequences to have the same length.

***Example of padding***

* Given a maximum sequence length of 20, we could pad every sentence with "0"s so that each input sentence is of length 20.
* Thus, the sentence "I love you" would be represented as (eI,elove,eyou,0⃗ ,0⃗ ,…,0⃗ )(eI,elove,eyou,0→,0→,…,0→).
* In this example, any sentences longer than 20 words would have to be truncated.
* One way to choose the maximum sequence length is to just pick the length of the longest sentence in the training set.

**2.3 - The Embedding layer**

* In Keras, the embedding matrix is represented as a "layer".
* The embedding matrix maps word indices to embedding vectors.
  + The word indices are positive integers.
  + The embedding vectors are dense vectors of fixed size.
  + When we say a vector is "dense", in this context, it means that most of the values are non-zero. As a counter-example, a one-hot encoded vector is not "dense."
* The embedding matrix can be derived in two ways:
  + Training a model to derive the embeddings from scratch.
  + Using a pretrained embedding

**Using and updating pre-trained embeddings**

* In this part, you will learn how to create an [Embedding()](https://keras.io/layers/embeddings/) layer in Keras
* You will initialize the Embedding layer with the GloVe 50-dimensional vectors.
* In the code below, we'll show you how Keras allows you to either train or leave fixed this layer.
* Because our training set is quite small, we will leave the GloVe embeddings fixed instead of updating them.

**Inputs and outputs to the embedding layer**

* The Embedding() layer's input is an integer matrix of size **(batch size, max input length)**.
  + This input corresponds to sentences converted into lists of indices (integers).
  + The largest integer (the highest word index) in the input should be no larger than the vocabulary size.
* The embedding layer outputs an array of shape (batch size, max input length, dimension of word vectors).
* The figure shows the propagation of two example sentences through the embedding layer.
  + Both examples have been zero-padded to a length of max\_len=5.
  + The word embeddings are 50 units in length.
  + The final dimension of the representation is (2,max\_len,50).

**Figure 4**: Embedding layer

**Prepare the input sentences**

**Exercise**:

* Implement sentences\_to\_indices, which processes an array of sentences (X) and returns inputs to the embedding layer:
  + Convert each training sentences into a list of indices (the indices correspond to each word in the sentence)
  + Zero-pad all these lists so that their length is the length of the longest sentence.

***Additional Hints***

* Note that you may have considered using the enumerate() function in the for loop, but for the purposes of passing the autograder, please follow the starter code by initializing and incrementing j explicitly.

#### Build embedding layer

* Let's build the Embedding() layer in Keras, using pre-trained word vectors.
* The embedding layer takes as input a list of word indices.
  + sentences\_to\_indices() creates these word indices.
* The embedding layer will return the word embeddings for a sentence.

**Exercise**: Implement pretrained\_embedding\_layer() with these steps:

1. Initialize the embedding matrix as a numpy array of zeros.
   * The embedding matrix has a row for each unique word in the vocabulary.
     + There is one additional row to handle "unknown" words.
     + So vocab\_len is the number of unique words plus one.
   * Each row will store the vector representation of one word.
     + For example, one row may be 50 positions long if using GloVe word vectors.
   * In the code below, emb\_dim represents the length of a word embedding.
2. Fill in each row of the embedding matrix with the vector representation of a word
   * Each word in word\_to\_index is a string.
   * word\_to\_vec\_map is a dictionary where the keys are strings and the values are the word vectors.
3. Define the Keras embedding layer.
   * Use [Embedding()](https://keras.io/layers/embeddings/).
   * The input dimension is equal to the vocabulary length (number of unique words plus one).
   * The output dimension is equal to the number of positions in a word embedding.
   * Make this layer's embeddings fixed.
     + If you were to set trainable = True, then it will allow the optimization algorithm to modify the values of the word embeddings.
     + In this case, we don't want the model to modify the word embeddings.
4. Set the embedding weights to be equal to the embedding matrix.
   * Note that this is part of the code is already completed for you and does not need to be modified.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise:** Implement Emojify\_V2(), which builds a Keras graph of the architecture shown in Figure 3.

* The model takes as input an array of sentences of shape (m, max\_len, ) defined by input\_shape.
* The model outputs a softmax probability vector of shape (m, C = 5).
* You may need to use the following Keras layers:
  + [Input()](https://keras.io/layers/core/#input)
    - Set the shape and dtype parameters.
    - The inputs are integers, so you can specify the data type as a string, 'int32'.
  + [LSTM()](https://keras.io/layers/recurrent/#lstm)
    - Set the units and return\_sequences parameters.
  + [Dropout()](https://keras.io/layers/core/#dropout)
    - Set the rate parameter.
  + [Dense()](https://keras.io/layers/core/#dense)
    - Set the units,
    - Note that Dense() has an activation parameter. For the purposes of passing the autograder, please do not set the activation within Dense(). Use the separate Activation layer to do so.
  + [Activation()](https://keras.io/activations/).
    - You can pass in the activation of your choice as a lowercase string.
  + [Model](https://keras.io/models/model/) Set inputs and outputs.

#### Additional Hints

* Remember that these Keras layers return an object, and you will feed in the outputs of the previous layer as the input arguments to that object. The returned object can be created and called in the same line.

*# How to use Keras layers in two lines of code*

dense\_object = Dense(units = ...)

X = dense\_object(inputs)

*# How to use Keras layers in one line of code*

X = Dense(units = ...)(inputs)

* The embedding\_layer that is returned by pretrained\_embedding\_layer is a layer object that can be called as a function, passing in a single argument (sentence indices).
* Here is some sample code in case you're stuck
* raw\_inputs = Input(shape=(maxLen,), dtype='int32')
* preprocessed\_inputs = ... *# some pre-processing*
* X = LSTM(units = ..., return\_sequences= ...)(processed\_inputs)
* X = Dropout(rate = ..., )(X)
* ...
* X = Dense(units = ...)(X)
* X = Activation(...)(X)

model = Model(inputs=..., outputs=...)