**Exercise**: Implement the sample function below to sample characters. You need to carry out 4 steps:

* **Step 1**: Input the "dummy" vector of zeros x⟨1⟩=0⃗ x⟨1⟩=0→.
  + This is the default input before we've generated any characters. We also set a⟨0⟩=0⃗ a⟨0⟩=0→
* **Step 2**: Run one step of forward propagation to get a⟨1⟩a⟨1⟩ and ŷ ⟨1⟩y^⟨1⟩. Here are the equations:

hidden state:

a⟨t+1⟩=tanh(Waxx⟨t+1⟩+Waaa⟨t⟩+b)(1)(1)a⟨t+1⟩=tanh⁡(Waxx⟨t+1⟩+Waaa⟨t⟩+b)

activation:

z⟨t+1⟩=Wyaa⟨t+1⟩+by(2)(2)z⟨t+1⟩=Wyaa⟨t+1⟩+by

prediction:

ŷ ⟨t+1⟩=softmax(z⟨t+1⟩)(3)(3)y^⟨t+1⟩=softmax(z⟨t+1⟩)

* Details about ŷ ⟨t+1⟩y^⟨t+1⟩:
  + Note that ŷ ⟨t+1⟩y^⟨t+1⟩ is a (softmax) probability vector (its entries are between 0 and 1 and sum to 1).
  + ŷ ⟨t+1⟩iy^i⟨t+1⟩ represents the probability that the character indexed by "i" is the next character.
  + We have provided a softmax() function that you can use.

**Additional Hints**

* x⟨1⟩x⟨1⟩ is x in the code. When creating the one-hot vector, make a numpy array of zeros, with the number of rows equal to the number of unique characters, and the number of columns equal to one. It's a 2D and not a 1D array.
* a⟨0⟩a⟨0⟩ is a\_prev in the code. It is a numpy array of zeros, where the number of rows is nana, and number of columns is 1. It is a 2D array as well. nana is retrieved by getting the number of columns in WaaWaa (the numbers need to match in order for the matrix multiplication Waaa⟨t⟩Waaa⟨t⟩ to work.
* [numpy.dot](https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html)
* [numpy.tanh](https://docs.scipy.org/doc/numpy/reference/generated/numpy.tanh.html)

**Using 2D arrays instead of 1D arrays**

* You may be wondering why we emphasize that x⟨1⟩x⟨1⟩ and a⟨0⟩a⟨0⟩ are 2D arrays and not 1D vectors.
* For matrix multiplication in numpy, if we multiply a 2D matrix with a 1D vector, we end up with with a 1D array.
* This becomes a problem when we add two arrays where we expected them to have the same shape.
* When two arrays with a different number of dimensions are added together, Python "broadcasts" one across the other.
* Here is some sample code that shows the difference between using a 1D and 2D array.

**Step 3**: Sampling:

* Now that we have y⟨t+1⟩y⟨t+1⟩, we want to select the next letter in the dinosaur name. If we select the most probable, the model will always generate the same result given a starting letter. To make the results more interesting, we will use np.random.choice to select a next letter that is *likely*, but not always the same.
* Pick the next character's **index** according to the probability distribution specified by ŷ ⟨t+1⟩y^⟨t+1⟩.
* This means that if ŷ ⟨t+1⟩i=0.16y^i⟨t+1⟩=0.16, you will pick the index "i" with 16% probability.
* Use [np.random.choice](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.choice.html).

Example of how to use np.random.choice():

np.random.seed(0)

probs = np.array([0.1, 0.0, 0.7, 0.2])

idx = np.random.choice(range(len((probs)), p = probs)

* This means that you will pick the index (idx) according to the distribution:

P(index=0)=0.1,P(index=1)=0.0,P(index=2)=0.7,P(index=3)=0.2P(index=0)=0.1,P(index=1)=0.0,P(index=2)=0.7,P(index=3)=0.2.

* Note that the value that's set to p should be set to a 1D vector.
* Also notice that ŷ ⟨t+1⟩y^⟨t+1⟩, which is y in the code, is a 2D array.
* Also notice, while in your implementation, the first argument to np.random.choice is just an ordered list [0,1,.., vocab\_len-1], it is *Not* appropriate to use char\_to\_ix.values(). The *order* of values returned by a python dictionary .values() call will be the same order as they are added to the dictionary. The grader may have a different order when it runs your routine than when you run it in your notebook.

##### *Additional Hints*

* [range](https://docs.python.org/3/library/functions.html#func-range)
* [numpy.ravel](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ravel.html) takes a multi-dimensional array and returns its contents inside of a 1D vector.
* arr = np.array([[1,2],[3,4]])
* print("arr")
* print(arr)
* print("arr.ravel()")
* print(arr.ravel())

Output:

arr

[[1 2]

[3 4]]

arr.ravel()

[1 2 3 4]

* Note that append is an "in-place" operation. In other words, don't do this:
* fun\_hobbies = fun\_hobbies.append('learning') *## Doesn't give you what you want*
* **Step 4**: Update to x⟨t⟩x⟨t⟩
  + The last step to implement in sample() is to update the variable x, which currently stores x⟨t⟩x⟨t⟩, with the value of x⟨t+1⟩x⟨t+1⟩.
  + You will represent x⟨t+1⟩x⟨t+1⟩ by creating a one-hot vector corresponding to the character that you have chosen as your prediction.
  + You will then forward propagate x⟨t+1⟩x⟨t+1⟩ in Step 1 and keep repeating the process until you get a "\n" character, indicating that you have reached the end of the dinosaur name.

##### *Additional Hints*

* In order to reset x before setting it to the new one-hot vector, you'll want to set all the values to zero.
  + You can either create a new numpy array: [numpy.zeros](https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html)
  + Or fill all values with a single number: [numpy.ndarray.fill](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.fill.html)

**def** rnn\_forward(X, Y, a\_prev, parameters):

""" Performs the forward propagation through the RNN and computes the cross-entropy loss.

It returns the loss' value as well as a "cache" storing values to be used in backpropagation."""

....

return loss, cache

**def** rnn\_backward(X, Y, parameters, cache):

""" Performs the backward propagation through time to compute the gradients of the loss with respect

to the parameters. It returns also all the hidden states."""

...

return gradients, a

**def** update\_parameters(parameters, gradients, learning\_rate):

""" Updates parameters using the Gradient Descent Update Rule."""

...

return parameters

Recall that you previously implemented the clip function:

**3.2 - Training the model**

* Given the dataset of dinosaur names, we use each line of the dataset (one name) as one training example.
* Every 2000 steps of stochastic gradient descent, you will sample several randomly chosen names to see how the algorithm is doing.

**Exercise**: Follow the instructions and implement model(). When examples[index] contains one dinosaur name (string), to create an example (X, Y), you can use this:

***Set the index idx into the list of examples***

* Using the for-loop, walk through the shuffled list of dinosaur names in the list "examples".
* For example, if there are n\_e examples, and the for-loop increments the index to n\_e onwards, think of how you would make the index cycle back to 0, so that we can continue feeding the examples into the model when j is n\_e, n\_e + 1, etc.
* Hint: n\_e + 1 divided by n\_e is zero with a remainder of 1.
* % is the modulus operator in python.

***Extract a single example from the list of examples***

* single\_example: use the idx index that you set previously to get one word from the list of examples.

***Convert a string into a list of characters: single\_example\_chars***

* single\_example\_chars: A string is a list of characters.
* You can use a list comprehension (recommended over for-loops) to generate a list of characters.
* str = 'I love learning'
* list\_of\_chars = [c **for** c **in** str]
* print(list\_of\_chars)

['I', ' ', 'l', 'o', 'v', 'e', ' ', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g']

***Convert list of characters to a list of integers: single\_example\_ix***

* Create a list that contains the index numbers associated with each character.
* Use the dictionary char\_to\_ix
* You can combine this with the list comprehension that is used to get a list of characters from a string.

***Create the list of input characters: X***

* rnn\_forward uses the **None** value as a flag to set the input vector as a zero-vector.
* Prepend the list [**None**] in front of the list of input characters.
* There is more than one way to prepend a value to a list. One way is to add two lists together: ['a'] + ['b']

***Get the integer representation of the newline character ix\_newline***

* ix\_newline: The newline character signals the end of the dinosaur name.
  + get the integer representation of the newline character '\n'.
  + Use char\_to\_ix

***Set the list of labels (integer representation of the characters): Y***

* The goal is to train the RNN to predict the next letter in the name, so the labels are the list of characters that are one time step ahead of the characters in the input X.
  + For example, Y[0] contains the same value as X[1]
* The RNN should predict a newline at the last letter so add ix\_newline to the end of the labels.
  + Append the integer representation of the newline character to the end of Y.
  + Note that append is an in-place operation.
  + It might be easier for you to add two lists together.