- ***building the model***
  - -> batching and shuffling the data
  - ***-> building the model***
    - -> he's defined a function which is building the model
    - -> we are training the model on batches of play
    - -> we are texting it on batches of one

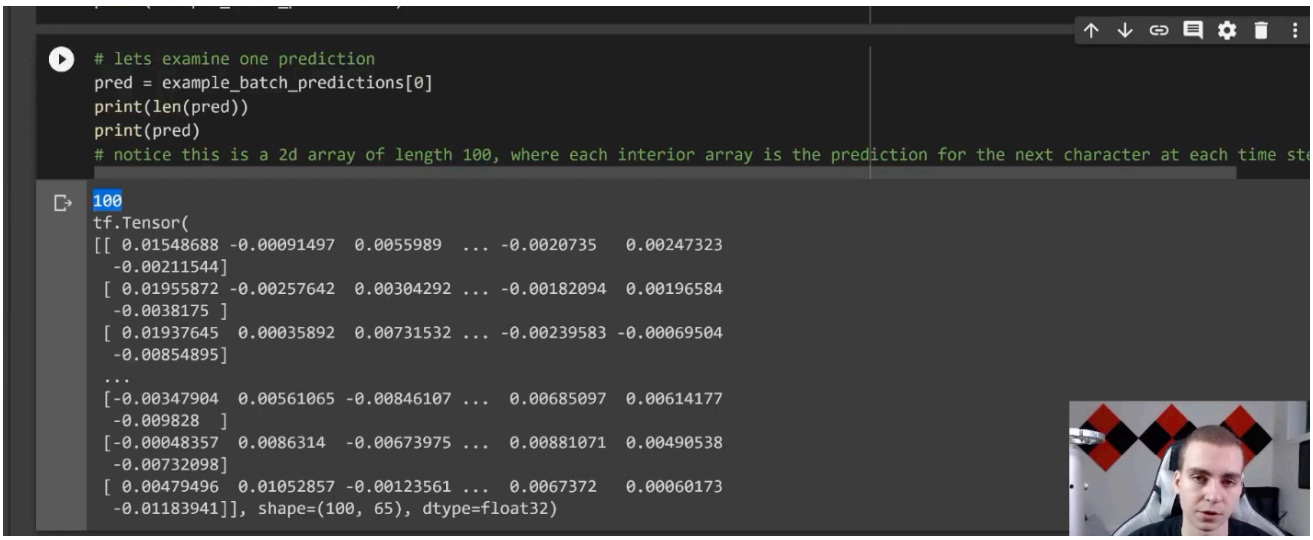
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
  model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              batch_input_shape=[batch_size, None]),
    tf.keras.layers.LSTM(rnn_units,
                         return_sequences=True,
                         stateful=True,
                         recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(vocab_size)
  ])
  return model

model = build_model(VOCAB_SIZE,EMBEDDING_DIM, RNN_UNITS, BATCH_SIZE)
model.summary()
```

- -> this includes the embedding dimension
- -> the batch size <- there are 64 entries per batch - and we don't know how long each one will be
  - -> we do this because we don't know how long the input string will be
- ***-> then making a long term short term memory layer***
  - -> using the model to make predictions
- ***-> returning the sequences***
  - -> we want to look at what the model is doing while it's training
  - -> we want the output at every single time step (after every word it processes)
- -> then recurrent_initialiser <- this is set to the default value for tensor flow
- ***-> the amount of vocabulary sized nodes***
  - -> we want the number of nodes in the final layer to be equal to the number of characters in the input string
  - -> so each of those nodes will represent the probability that that word comes next in the string
  - -> so summing them together should give 1
    - -> a predictive layer
- ***-> then printing out a model summary***

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (64, None, 256)           16640
_________________________________________________________________
lstm_1 (LSTM)                (64, None, 1024)          5246976
_________________________________________________________________
dense_1 (Dense)              (64, None, 65)            66625
=================================================================
Total params: 5,330,241
Trainable params: 5,330,241
Non-trainable params: 0
_________________________________________________________________
```

        - -> the rows are for the initial embedding layer, the LSTM and then if the model is dense or not
            - -> LSTM - aka the long term short term memory of the model - its output at each stage of training
            - -> it's a recurrence neural network - which means it's for natural language processing where for an entire layer of the network one word is processed at a time
            - -> this is a record for the value of the model each time those calculations is done
            - -> 65 <- the length of the vocabulary
- ***Creating a loss function***
    - -> this is the function whose value we are minimising
    - -> the function stores the accuracy of the model while it's being trained
    - ***-> the input is of length 64***
        - -> we are giving the model 64 training examples (sections of text to train on), and each is 100 words long
            - so the final node contains 65 nodes - each represents a word and the output is the probability that the next word in the sequence is one of those words
            - -> it's returning a length 64 tensor and each element in that array is the probability that the next word is the word which is represented at the element in the array



    - ***-> so we have a build model function***
        - -> using the parameters which we've trained
        - -> we can expect a different input shape -> depending on the length of the input string
        - -> you give it the first batch and pass it to the model
    - -> so the model is being ran for each word, and each time it's ran there are 64 different words which it could be
        - -> so we end up with an array which is 64x64 (in this case where we are training it)
    - ***-> getting a tensor of length 65***
        - -> what is outputted from the model
        - -> you need to be able to make your own loss function in tensor flow because the array which is being returned is in a certain form and specific to this context
    - ***-> at every time step, the prediction is sampled***
        - -> np.reshape
        - -> we are sampling and not just taking the maximum /mean probability
            - -> the mean probability exists in its own distribution
            - -> so you need to sample them

```
# If we want to determine the predicted character we need to sample the output distribution (pick a value based on probabillit
sampled_indices = tf.random.categorical(pred, num_samples=1)

# now we can reshape that array and convert all the integers to numbers to see the actual characters
sampled_indices = np.reshape(sampled_indices, (1, -1))[0]
predicted_chars = int_to_text(sampled_indices)

predicted_chars  # and this is what the model predicted for training sequence 1
```

- ***-> keras has a builtin loss function***
  - -> computing a loss
  - -> the goal of the algorithm is to reduce the loss

```
def loss(labels, logits):
  return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)
```