- ***compiling the model***
  - -> with the optimiser and the loss function
  - -> then creating checkpoints
  - -> then training the model -> changing the run time type to GPU to give it more resources to train the model
    - -> the more epochs in this case the better the model (there is less chance of overfitting)

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

+ Code + Text

```
[ ]  model.compile(optimizer='adam', loss=loss)
```

```
[ ]  # Directory where the checkpoints will be saved
     checkpoint_dir = './training_checkpoints'
     # Name of the checkpoint files
     checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

     checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
         filepath=checkpoint_prefix,
         save_weights_only=True)
```

```
[ ]  history = model.fit(data, epochs=40, callbacks=[checkpoint_callback])
```

- ***loading the model***
  - -> the model is being trained on batches of 64 words
  - -> we want the model to return one word
  - -> so once the model is trained then we need to change it to accommodate one word
  - -> the finished model is taking the previous words and predicting the next one
  - -> he's changing the model to a batch size of 1

```
[69]  model = build_model(VOCAB_SIZE, EMBEDDING_DIM, RNN_UNITS, batch_size=1)
```

  - -> then telling it to rebuild the model

```
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
model.build(tf.TensorShape([1, None]))
```

```
[ ]  checkpoint_num = 10
     model.load_weights(tf.train.load_checkpoint("./training_checkpoints/ckpt_" + str(checkpoint_num)))
     model.build(tf.TensorShape([1, None]))
```

        - -> the first cell is loading the weights for the model
        - -> the directory is where the tensor flow checkpoint is saved
        - -> then the prefixes which it's being saved with
        - -> they are saved when each batch of training data is loaded into it
- ***generating text***
    - ***-> how the function works***
        - -> he enters a string and then it generates an output sequence
        - -> it's pseudo English -> it was trained on two epochs
        - -> to improve the accuracy (in this case) you would train it on more epochs
    - -> he's running the function called generate text -> which is then returning the characters
    - -> expanding the dimensions turns a [list] into a [[nested list]]
    - -> then there is a temperature parameter which you can tweak -> to make the results more surprising / predictable
        - -> inputting a value
    - ***-> using a categorical distribution***
        - -> sampling the output from the model
        - -> taking the output and adding it to the input evaluation
        - -> converting the text / integers back into a string and returning it
- ***a summary of the code***
    - -> importing the modules
    - -> loading in the file
    - -> reading the file
    - -> decoding the file into utf8
    - -> creating a vocabulaty and encoding the text which is inside the file
    - -> write a function which goes from integer to text
        - -> and from text to integers
        - -> it's a translator
    - -> then character slicing the dataset into different training datasets
        - -> bathing them in lengths of 101
        - -> splitting it into the training example
    - -> mapping the function two sequences -> applying it to two different datasets
        - -> defining the parameters for the initial dataset
        - -> define the function which makes the model
        - -> building the model
        - -> creating the loss function
        - -> compiling the model
        - -> setting the checkpoints for saving
        - -> training the model
        - -> checkpoint callback <- saving the epochs (batches of data the model is trained on)
- ***using the model with the Bee Movie script used to train it***
    - -> building the model with a batch size of 1
    - -> Romeo and Juliet is longer and a lot more predictable
        - natural language processing differs depending on the data used to train it (which text)
    - -> when the model is being trained, after each time you want the loss function to decrease - i.e the accuracy of the model to increase
- ***next***
    - -> reinforcement learning