Question 5:
(a)
According to the experiments in Question 1 and Question 2, RNN (2 hidden layers) with Relu gate, ClippedGradientDescentOptimizer and Sparse_softmax_cross_entropy_with_logits loss function gets better results than other models in terms of bits per character loss. According to the experiments in Question 3, though RNN (2 hidden layers) with tanh gate, ClippedGradientDescentOptimizer and Sparse_softmax_cross_entropy_with_logits loss function gets the best result, RNN (2 hidden layers) with tanh gate, ClippedGradientDescentOptimizer and sparse_kl_divergence_ml loss function also gets nice result in terms of misclassification error rate. Misclassification error rate is not the same as the bits per character loss, but I think if I combine RNN (2 hidden layers) with Relu gate, ClippedGradientDescentOptimizer
and sparse_kl_divergence_ml loss function, I may can get better result in terms of bits per character loss and it worths try. However, one problem is that if I pass sparse_kl_divergence_ml loss function into model_rnn class, the result will only show the loss which is defined by sparse_kl_divergence_ml loss function instead of bits per character loss, therefore, I changed the code a little bit:
in class model_rnn() and def __init__(self, name, cell_type, dimensions, gate_fun, loss_fun):
I add these codes:

```
show_loss_fun = loss_kl_base2
self.show_train_loss = tf.reduce_mean(show_loss_fun(z_hat, self.y))
```

In def methoddef(name, color, model, optimizer, xdata, ydata, data_train, data_valid, data_test):

I changed

```
meas.meas_rnnloss(model.x, model.y,
                                  model.zero_state, model.init_state,
                                  model.final_state,
                                  data_valid, model.train_loss, "valid_loss",
                                  BATCH, STEPS,
                                  axes=[0.0, np.inf, 0.0, YMAX_VALID]),
```

to

```
meas.meas_rnnloss(model.x, model.y,
```

```
model.zero_state, model.init_state,
model.final_state,
data_valid, model.show_train_loss, "valid_loss",
BATCH, STEPS,
axes=[0.0, np.inf, 0.0, YMAX_VALID]),
```

By changing these codes, the model can train the model based on sparse_kl_divergence_ml loss function, but when it prints the result, it will show bits per character loss instead of the loss defined by sparse_kl_divergence_ml loss function.

In conclusion, in this question, I combine RNN (2 hidden layers) with Relu gate, ClippedGradientDescentOptimizer and sparse_kl_divergence_ml loss function to see whether it can get better result and changed the code a little bit.

(d)

It's very sad that the new model, RNN (2 hidden layers) with Relu gate, ClippedGradientDescentOptimizer and sparse_kl_divergence_ml loss function doesn't beat the previous model. The bits per character loss gets by the new model is really large, the test loss is about 4.5, while the test loss got by the previous best model is only around 1.2. What surprise me is that though the new model doesn't get good result in terms of bits per character loss, the new model do really good job in terms of Misclassification error rate. The test error gained by the new model is below 0.3, which is better than the previous best model. It's really strange the new model perform terribly in terms of bits per character loss, while it does good job in terms of Misclassification error rate. What I think before this experiment is that the model which can get low bits per character loss can also get low misclassification error because I notice RNN (2 hidden layers) with tanh, ClippedGradientDescentOptimizer and Sparse_softmax_cross_entropy_with_logits loss function gets good result in terms of bits per character loss as well as Misclassification error rate, and that's why I combine the model which do well in Question 3 and the model which do well in Question 1 and 2. Now I know it was not necessarily right.