In [ ]:

```
import sys
sys.path.append('.')
from common import config
from common.time_layers import *
from common.base_model import BaseModel

class Grulm(BaseModel):
    def __init__(self, vocab_size=10000, wordvec_size=100, hidden_size=100):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        # initialize weights
        embed_W = (rn(V, D) / 100).astype('f')
        gru_Wx = (rn(D, 3 * H) / np.sqrt(D)).astype('f')
        gru_Wh = (rn(H, 3 * H) / np.sqrt(H)).astype('f')
        gru_b = np.zeros(3 * H).astype('f')
        affine_W = (rn(H, V) / np.sqrt(H)).astype('f')
        affine_b = np.zeros(V).astype('f')

        # generate layers
        self.layers = [
            TimeEmbedding(embed_W),
            TimeGRU(gru_Wx, gru_Wh, gru_b, stateful=True),
            TimeAffine(affine_W, affine_b)
        ]
        self.loss_layer = TimeSoftmaxWithLoss()
        self.gru_layer = self.layers[1]

        # collect all the weights and gradients into a list
        self.params, self.grads = [], []
        for layer in self.layers:
            self.params += layer.params
            self.grads += layer.grads

    def predict(self, xs):
        for layer in self.layers:
            xs = layer.forward(xs)
        return xs

    def forward(self, xs, ts):
        score = self.predict(xs)
        loss = self.loss_layer.forward(score, ts)
        return loss

    def backward(self, dout=1):
        dout = self.loss_layer.backward(dout)
        for layer in reversed(self.layers):
            dout = layer.backward(dout)
        return dout

    def reset_state(self):
        self.gru_layer.reset_state()


# TRAINING
from common.optimizer import SGD
from common.trainer import RnnlmTrainer
from common.util import eval_perplexity
```

```
from dataset import ptb

# set hyperparameters
batch_size = 20
wordvec_size = 100
hidden_size = 100
time_size = 35
lr = 20.0
max_epoch = 1
max_grad = 0.25

# read training data
corpus, word_to_id, id_to_word = ptb.load_data('train')
corpus_val, _, _ = ptb.load_data('val')
corpus_test, _, _ = ptb.load_data('test')

vocab_size = len(word_to_id)
xs = corpus[:-1]
ts = corpus[1:]

model = Grulm(vocab_size, wordvec_size, hidden_size)
optimizer = SGD(lr)
trainer = RnnlmTrainer(model, optimizer)

best_ppl = float('inf')
for epoch in range(max_epoch):
    trainer.fit(xs, ts, max_epoch=1, batch_size=batch_size,
                time_size=time_size, max_grad=max_grad)

    trainer.plot()

    model.reset_state()
    ppl = eval_perplexity(model, corpus_val)
    print(f'Perplexity at epoch {epoch + 1} is {ppl}')

    if best_ppl > ppl:
        best_ppl = ppl
        model.save_params()
    else:
        lr /= 4.0
        optimizer.lr = lr

    model.reset_state()
    print('-' * 50)


# evaluate based on validation data
model.reset_state()
ppl_test = eval_perplexity(model, corpus_test)
print('test perplexity: ', ppl_test)
```