In [1]:
```
from torch import optim
import numpy as np
import torch

import utils
from utils import get_chunks
from config import device
import config as cfg
from data2index_ver2 import train_data, test_data, index2slot_dict
from model import *
```

```
Number of training samples:  4978
Number of test samples:  893
Number of words:  754
Number of intent labels:  18
Number of slot labels 130
```

In [2]:
```
epoch_num = cfg.total_epoch

slot_model = Slot().to(device)
intent_model = Intent().to(device)

print(slot_model)
print(intent_model)
```

```
Slot(
  (enc): slot_enc(
    (embedding): Embedding(754, 300)
    (lstm): LSTM(300, 200, num_layers=2, batch_first=True, bidirectional=Tr
ue)
  )
  (dec): slot_dec(
    (lstm): LSTM(1000, 200)
    (fc): Linear(in_features=200, out_features=130, bias=True)
  )
)
Intent(
  (enc): intent_enc(
    (embedding): Embedding(754, 300)
    (lstm): LSTM(300, 200, num_layers=2, batch_first=True, dropout=0.2, bid
irectional=True)
  )
  (dec): intent_dec(
    (lstm): LSTM(800, 200, batch_first=True)
    (fc): Linear(in_features=200, out_features=18, bias=True)
  )
)
```

In [*]:

```
slot_optimizer = optim.Adam(slot_model.parameters(), lr=cfg.learning_rate)
intent_optimizer = optim.Adam(intent_model.parameters(), lr=cfg.learning_rate

best_correct_num = 0
best_epoch = -1
best_F1_score = 0.0
best_epoch_slot = -1
for epoch in range(epoch_num):
    slot_loss_history = []
    intent_loss_history = []
    for batch_index, data in enumerate(utils.get_batch(train_data)):

        # Preparing data
        sentence, real_len, slot_label, intent_label = data

        mask = utils.make_mask(real_len).to(device)
        x = torch.tensor(sentence).to(device)
        y_slot = torch.tensor(slot_label).to(device)
        y_slot = utils.one_hot(y_slot).to(device)
        y_intent = torch.tensor(intent_label).to(device)
        y_intent = utils.one_hot(y_intent, Num=18).to(device)

        # Calculate compute graph
        slot_optimizer.zero_grad()
        intent_optimizer.zero_grad()

        hs = slot_model.enc(x)
        slot_model.share_memory = hs.clone()

        hi = intent_model.enc(x)
        intent_model.share_memory = hi.clone()


        slot_logits = slot_model.dec(hs, intent_model.share_memory.detach())
        log_slot_logits = utils.masked_log_softmax(slot_logits, mask, dim=-1)
        slot_loss = -1.0*torch.sum(y_slot*log_slot_logits)
        slot_loss_history.append(slot_loss.item())
        slot_loss.backward()
        torch.nn.utils.clip_grad_norm_(slot_model.parameters(), 5.0)
        slot_optimizer.step()

        # Asynchronous training
        intent_logits = intent_model.dec(hi, slot_model.share_memory.detach()
        log_intent_logits = F.log_softmax(intent_logits, dim=-1)
        intent_loss = -1.0*torch.sum(y_intent*log_intent_logits)
        intent_loss_history.append(intent_loss.item())
        intent_loss.backward()
        torch.nn.utils.clip_grad_norm_(intent_model.parameters(), 5.0)
        intent_optimizer.step()

        # Log
        if batch_index % 100 == 0 and batch_index > 0:
            print('Slot loss: {:.4f} \t Intent loss: {:.4f}'.format(sum(slot_
                sum(intent_loss_history[-100:])/100.0))

    # Evaluation
```

```
    total_test = len(test_data)
    correct_num = 0
    TP, FP, FN = 0, 0, 0
    for batch_index, data_test in enumerate(utils.get_batch(test_data, batch_
        sentence_test, real_len_test, slot_label_test, intent_label_test = da
        # print(sentence[0].shape, real_len.shape, slot_label.shape)
        x_test = torch.tensor(sentence_test).to(device)

        mask_test = utils.make_mask(real_len_test, batch=1).to(device)
        # Slot model generate hs_test and intent model generate hi_test
        hs_test = slot_model.enc(x_test)
        hi_test = intent_model.enc(x_test)

        # Slot
        slot_logits_test = slot_model.dec(hs_test, hi_test)
        log_slot_logits_test = utils.masked_log_softmax(slot_logits_test, mas
        slot_pred_test = torch.argmax(log_slot_logits_test, dim=-1)
        # Intent
        intent_logits_test = intent_model.dec(hi_test, hs_test, real_len_test
        log_intent_logits_test = F.log_softmax(intent_logits_test, dim=-1)
        res_test = torch.argmax(log_intent_logits_test, dim=-1)


        if res_test.item() == intent_label_test[0]:
            correct_num += 1
        if correct_num > best_correct_num:
            best_correct_num = correct_num
            best_epoch = epoch
            # Save and load the entire model.
            torch.save(intent_model, 'model_intent_best.ckpt')
            torch.save(slot_model, 'model_slot_best.ckpt')

        # Calc slot F1 score

        slot_pred_test = slot_pred_test[0][:real_len_test[0]]
        slot_label_test = slot_label_test[0][:real_len_test[0]]

        slot_pred_test = [int(item) for item in slot_pred_test]
        slot_label_test = [int(item) for item in slot_label_test]

        slot_pred_test = [index2slot_dict[item] for item in slot_pred_test]
        slot_label_test = [index2slot_dict[item] for item in slot_label_test]

        pred_chunks = get_chunks(['O'] + slot_pred_test + ['O'])
        label_chunks = get_chunks(['O'] + slot_label_test + ['O'])
        for pred_chunk in pred_chunks:
            if pred_chunk in label_chunks:
                TP += 1
            else:
                FP += 1
        for label_chunk in label_chunks:
            if label_chunk not in pred_chunks:
                FN += 1

    F1_score = 100.0*2*TP/(2*TP+FN+FP)
    if F1_score > best_F1_score:
        best_F1_score = F1_score
```

```
            best_epoch_slot = epoch
        print('*'*20)
        print('Epoch: [{}/{}], Intent Val Acc: {:.4f} \t Slot F1 score: {:.4f}'.f
        print('*'*20)

        print('Best Intent Acc: {:.4f} at Epoch: [{}]'.format(100.0*best_correct_
        print('Best F1 score: {:.4f} at Epoch: [{}]'.format(best_F1_score, best_e
```

```
Epoch: [12/500], Intent Val Acc: 96.9765         Slot F1 score: 94.9920
********************
Best Intent Acc: 97.2004 at Epoch: [9]
Best F1 score: 95.0194 at Epoch: [9]
Slot loss: 0.3163        Intent loss: 0.0773
Slot loss: 0.2780        Intent loss: 0.0779
Slot loss: 0.5873        Intent loss: 0.0800
********************
Epoch: [13/500], Intent Val Acc: 96.7525         Slot F1 score: 94.8020
********************
Best Intent Acc: 97.2004 at Epoch: [9]
Best F1 score: 95.0194 at Epoch: [9]
Slot loss: 0.4932        Intent loss: 0.1178
Slot loss: 0.4382        Intent loss: 0.1064
Slot loss: 0.4011        Intent loss: 0.2413
********************
Epoch: [14/500], Intent Val Acc: 96.9765         Slot F1 score: 95.2061
********************
Best Intent Acc: 97.2004 at Epoch: [9]
Best F1 score: 95.2061 at Epoch: [14]
```