ITE4052 Computer Vision

# Programming Assignment3

**2020095178 최윤선**

## 1. Code implementation and inference

Validation loss 를 추출하기 위해 captioning_solver.py 와 captionings_solver_transformer.py 의 _step 함수를 수정하였다.

captioning_solver.py

```python
def _step(self):
    """
    Make a single gradient update. This is called by train() and should not
    be called manually.
    """
    # Make a minibatch of training data
    minibatch = sample_coco_minibatch(
        self.data, batch_size=self.batch_size, split="train"
    )
    captions, features, urls = minibatch

    # Compute loss and gradient
    loss, grads = self.model.loss(features, captions)
    self.train_loss_history.append(loss)

    # Perform a parameter update
    for p, w in self.model.params.items():
        dw = grads[p]
        config = self.optim_configs[p]
        next_w, next_config = self.update_rule(w, dw, config)
        self.model.params[p] = next_w
        self.optim_configs[p] = next_config

    # Make a minibatch of validation data
    val_minibatch = sample_coco_minibatch(
        self.data, batch_size=self.batch_size, split="val"
    )
    val_captions, val_features, val_urls = val_minibatch

    # Compute validation loss and gradient
    val_loss, val_grads = self.model.loss(val_features, val_captions)
    self.val_loss_history.append(val_loss)
```

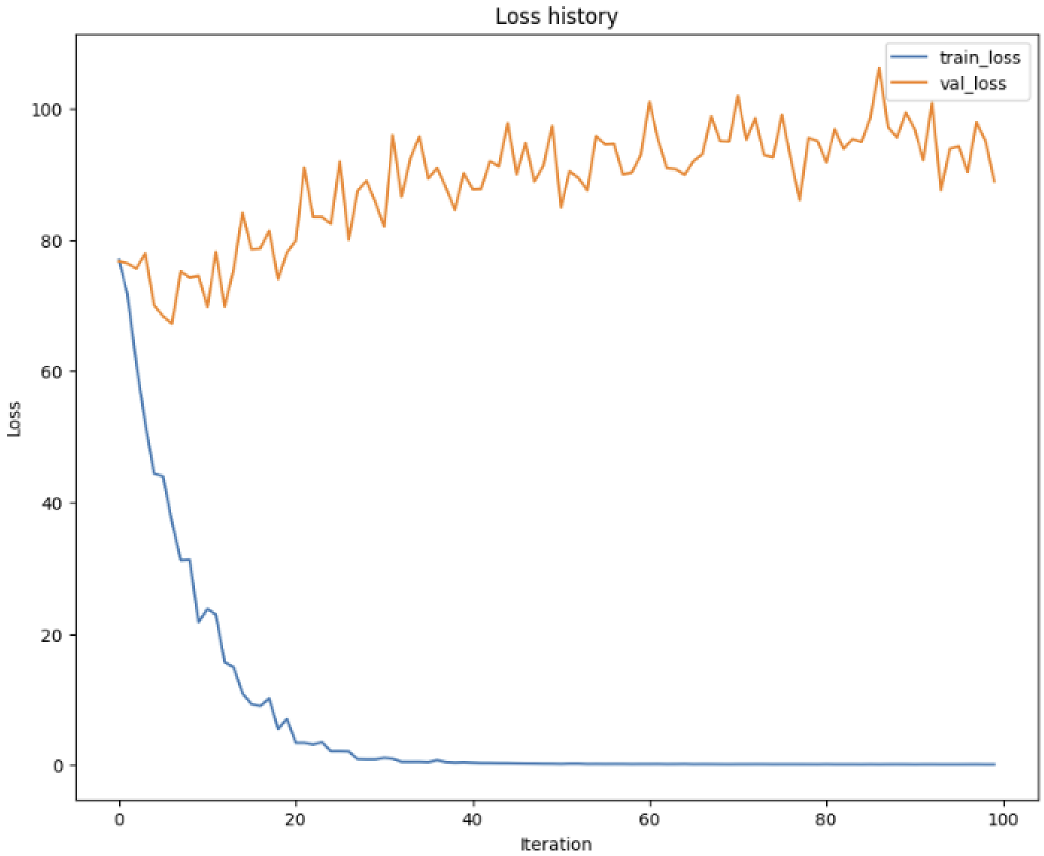
captioning_solver_transformer.py

```python
def _step(self):
    """
    Make a single gradient update. This is called by train() and should not
    be called manually.
    """
    # Make a minibatch of training data
    minibatch = sample_coco_minibatch(
        self.data, batch_size=self.batch_size, split="train"
    )
    captions, features, urls = minibatch

    captions_in = captions[:, :-1]
    captions_out = captions[:, 1:]

    mask = captions_out != self.model._null

    t_features = torch.Tensor(features)
    t_captions_in = torch.LongTensor(captions_in)
    t_captions_out = torch.LongTensor(captions_out)
    t_mask = torch.LongTensor(mask)
    logits = self.model(t_features, t_captions_in)

    loss = self.transformer_temporal_softmax_loss(logits, t_captions_out, t_mask)
    self.train_loss_history.append(loss.detach().numpy())
    self.optim.zero_grad()
    loss.backward()
    self.optim.step()

    # Make a minibatch of validation data
    val_minibatch = sample_coco_minibatch(
        self.data, batch_size=self.batch_size, split="val"
    )
    val_captions, val_features, val_urls = val_minibatch

    val_captions_in = val_captions[:, :-1]
    val_captions_out = val_captions[:, 1:]

    val_mask = val_captions_out != self.model._null

    val_t_features = torch.Tensor(val_features)
    val_t_captions_in = torch.LongTensor(val_captions_in)
    val_t_captions_out = torch.LongTensor(val_captions_out)
    val_t_mask = torch.LongTensor(val_mask)
    val_logits = self.model(val_t_features, val_t_captions_in)

    val_loss = self.transformer_temporal_softmax_loss(val_logits, val_t_captions_out, val_t_mask)
    self.val_loss_history.append(val_loss.detach().numpy())
```

- **RNN**

  - Validation loss curve

```
(Train Iteration 1 / 100) loss: 76.913487       (Validation Iteration 1 / 100) loss: 76.709664
(Train Iteration 11 / 100) loss: 23.782408      (Validation Iteration 11 / 100) loss: 69.812757
(Train Iteration 21 / 100) loss: 3.341500       (Validation Iteration 21 / 100) loss: 79.848425
(Train Iteration 31 / 100) loss: 1.076844       (Validation Iteration 31 / 100) loss: 81.969059
(Train Iteration 41 / 100) loss: 0.332487       (Validation Iteration 41 / 100) loss: 87.697004
(Train Iteration 51 / 100) loss: 0.159628       (Validation Iteration 51 / 100) loss: 84.918769
(Train Iteration 61 / 100) loss: 0.143359       (Validation Iteration 61 / 100) loss: 101.026165
(Train Iteration 71 / 100) loss: 0.118976       (Validation Iteration 71 / 100) loss: 101.915825
(Train Iteration 81 / 100) loss: 0.116376       (Validation Iteration 81 / 100) loss: 91.756397
(Train Iteration 91 / 100) loss: 0.090608       (Validation Iteration 91 / 100) loss: 96.849620
```




```
Final train loss:  0.08876050923087137
```

✓ Qualitative results of three validation images

val
from a large table of a <END>
GT:<START> sliced sandwich with tomatoes on a plate on a table <END>

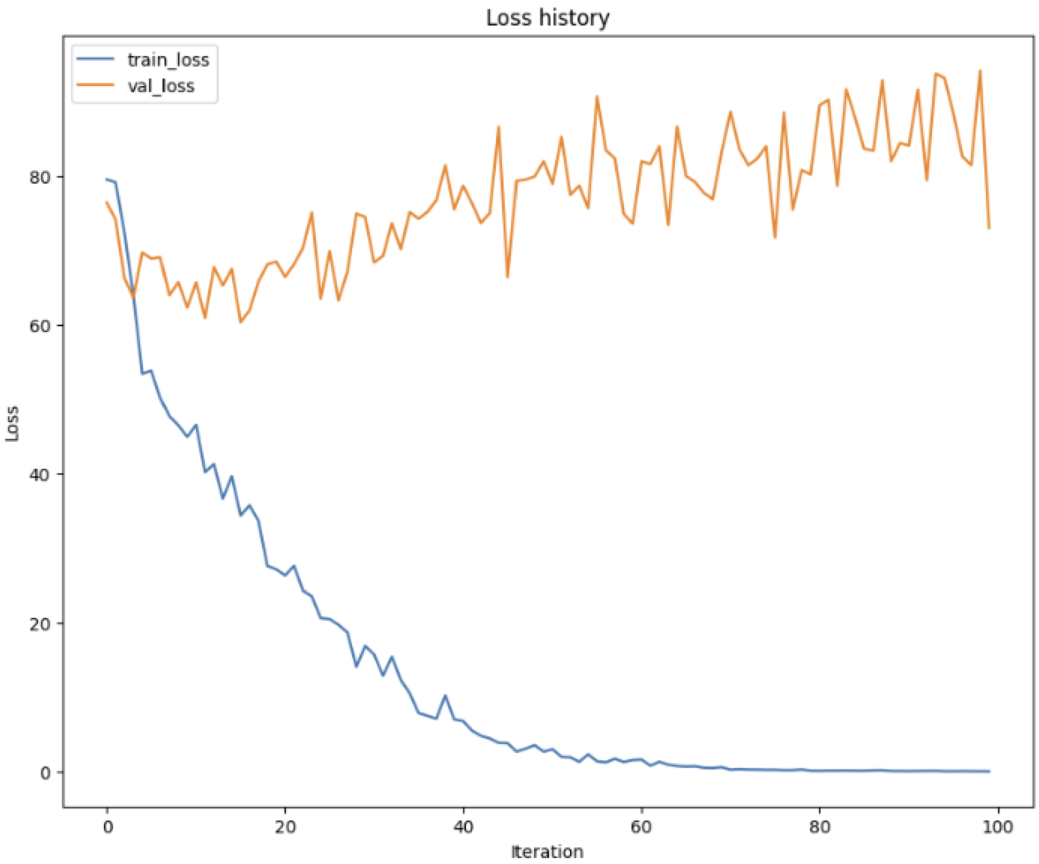
val
a plane that is <UNK> the in coming suitcases <END>
GT:<START> a very nice looking fighter type plane in the grass <END>


val
winter watches <END>
GT:<START> some green chairs a laptop on a table some plants and trees <END>

## ➢ LSTM

✓ Validation loss curve

```
(Train Iteration 1 / 100) loss: 79.551150        (Validation Iteration 1 / 100) loss: 76.450083
(Train Iteration 11 / 100) loss: 46.586122       (Validation Iteration 11 / 100) loss: 65.723861
(Train Iteration 21 / 100) loss: 26.353119       (Validation Iteration 21 / 100) loss: 66.468134
(Train Iteration 31 / 100) loss: 15.793431       (Validation Iteration 31 / 100) loss: 68.411234
(Train Iteration 41 / 100) loss: 6.853037        (Validation Iteration 41 / 100) loss: 78.721156
(Train Iteration 51 / 100) loss: 3.064186        (Validation Iteration 51 / 100) loss: 78.949645
(Train Iteration 61 / 100) loss: 1.671999        (Validation Iteration 61 / 100) loss: 81.996636
(Train Iteration 71 / 100) loss: 0.322624        (Validation Iteration 71 / 100) loss: 88.657692
(Train Iteration 81 / 100) loss: 0.161874        (Validation Iteration 81 / 100) loss: 89.516888
(Train Iteration 91 / 100) loss: 0.125630        (Validation Iteration 91 / 100) loss: 84.096376
```




```
Final train loss:  0.08816709030385614
```

✓ Qualitative results of three validation images

val
a man is a bite of his picture taken <END>
GT:<START> a yellow wall and a white bed in a room <END>

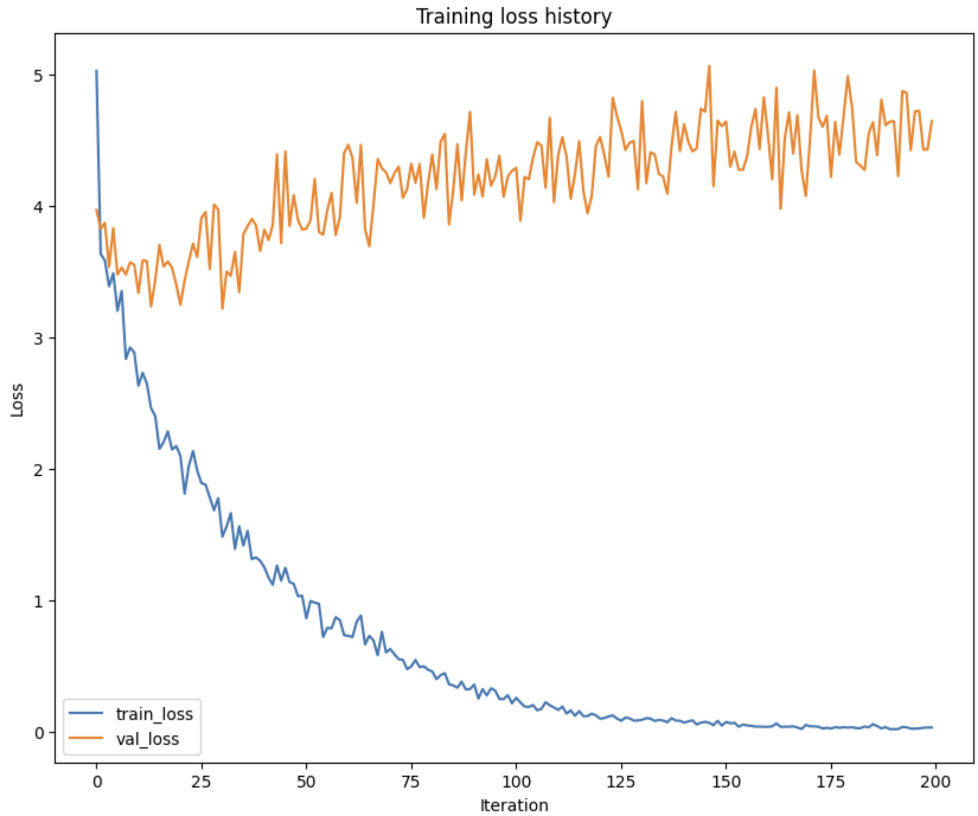
val
a man is <UNK> with a donut in his hand <END>
GT:<START> three girls hanging out together on a nice day <END>

val
a cat sitting with a <UNK> <END>
GT:<START> two elephants in a pen with their <UNK> <UNK> <END>

## ➢ Transformer

### ✓ Validation loss curve

```
(Train Iteration 1 / 200) loss: 5.023862        (Validation Iteration 1 / 200) loss: 3.968018
(Train Iteration 11 / 200) loss: 2.631999       (Validation Iteration 11 / 200) loss: 3.335799
(Train Iteration 21 / 200) loss: 2.097307       (Validation Iteration 21 / 200) loss: 3.244662
(Train Iteration 31 / 200) loss: 1.483391       (Validation Iteration 31 / 200) loss: 3.216268
(Train Iteration 41 / 200) loss: 1.250284       (Validation Iteration 41 / 200) loss: 3.817843
(Train Iteration 51 / 200) loss: 0.863107       (Validation Iteration 51 / 200) loss: 3.824898
(Train Iteration 61 / 200) loss: 0.729774       (Validation Iteration 61 / 200) loss: 4.461519
(Train Iteration 71 / 200) loss: 0.629557       (Validation Iteration 71 / 200) loss: 4.171115
(Train Iteration 81 / 200) loss: 0.458553       (Validation Iteration 81 / 200) loss: 4.387319
(Train Iteration 91 / 200) loss: 0.358953       (Validation Iteration 91 / 200) loss: 4.084745
(Train Iteration 101 / 200) loss: 0.259114      (Validation Iteration 101 / 200) loss: 4.288295
(Train Iteration 111 / 200) loss: 0.166151      (Validation Iteration 111 / 200) loss: 4.384060
(Train Iteration 121 / 200) loss: 0.099862      (Validation Iteration 121 / 200) loss: 4.519814
(Train Iteration 131 / 200) loss: 0.091569      (Validation Iteration 131 / 200) loss: 4.794100
(Train Iteration 141 / 200) loss: 0.070163      (Validation Iteration 141 / 200) loss: 4.618903
(Train Iteration 151 / 200) loss: 0.074946      (Validation Iteration 151 / 200) loss: 4.639798
(Train Iteration 161 / 200) loss: 0.037240      (Validation Iteration 161 / 200) loss: 4.535690
(Train Iteration 171 / 200) loss: 0.041797      (Validation Iteration 171 / 200) loss: 4.499037
(Train Iteration 181 / 200) loss: 0.034865      (Validation Iteration 181 / 200) loss: 4.752665
(Train Iteration 191 / 200) loss: 0.020196      (Validation Iteration 191 / 200) loss: 4.641896
```




```
Final train loss:  0.032513067
```

- ✓ Qualitative results of three validation images

val
half a <UNK> near a <UNK> of a apples <END>
GT:<START> three animals on the side of a <UNK> rocky hill <END>

val
a plane truck parked to the ground in the <UNK> in a landing <END>
GT:<START> some animals that are standing in the grass together <END>

val
half there is walk large wave in the water <END>
GT:<START> a man is surfing a small wave on a board <END>

## 2. Implementation of evaluation matric (CIDEr score)

```
!pip install pycocoevalcap
from pycocoevalcap.cider.cider import Cider
```

'pycocoevalcap' 패키지를 설치하여 CIDEr score 를 계산하였다. 다음 github 을 참고하여 구현하였다.

https://github.com/salaniz/pycocoevalcap

### ➢ RNN

먼저 train 과 validation dataset 의 gt_caption 과 smaple_caption 형태를 확인하였다.

```
print(train_gt_caption)
print(train_sample_caption)
print(val_gt_caption)
print(val_sample_caption)
```

```
[['<START> there is a male surfer coming out of the water <END>', '<START> a group of people that are sitting near
[['there is a male surfer coming out of the water <END>', 'a group of people that are sitting near train tracks <E
[['<START> sliced sandwich with tomatoes on a plate on a table <END>', '<START> a very nice looking fighter type p
[['from a large table of a <END>', 'a plane that is <UNK> the in coming suitcases <END>', 'winter watches <END>']]
```

Cider() 라이브러리의 compute_score 함수를 사용하기 위해 caption 의 형태를 list 에서 dictionary 형태로 변형해 주었다.

```
{'img1': ['<START> there is a male surfer coming out of the water <END>'], 'img2': ['<START> a group of people that are sitting near trai
{'img1': ['there is a male surfer coming out of the water <END>'], 'img2': ['a group of people that are sitting near train tracks <END>']

{'img1': ['<START> sliced sandwich with tomatoes on a plate on a table <END>'], 'img2': ['<START> a very nice
{'img1': ['from a large table of a <END>'], 'img2': ['a plane that is <UNK> the in coming suitcases <END>'],
```

RNN 모델의 CIDEr score 는 다음과 같다.

```
scorer = Cider()
score, _ = scorer.compute_score(train_gt_caption_dict, train_sample_caption_dict)
print('Training CIDEr score: ', score)
```

```
Training CIDEr score:  9.446182157629071
```

```
scorer = Cider()
score, _ = scorer.compute_score(val_gt_caption_dict, val_sample_caption_dict)
print('Validating CIDEr score: ', score)
```

```
Validating CIDEr score:  0.297708996448075
```

### ➢ LSTM

RNN_Captioning 과 똑같이 코드를 구현했으며, captions 의 dictionary 형태는 다음과 같다.

```
print(train_gt_caption)
print(train_sample_caption)
print(val_gt_caption)
print(val_sample_caption)
```

```
[['<START> many people standing near boxes of many apples <END>', '<START> a man standing at home plate preparing to bat <END>', '<STA
[['many people standing near boxes of many apples <END>', 'a man standing at home plate preparing to bat <END>', 'a cute dog standing
[['<START> a yellow wall and a white bed in a room <END>', '<START> three girls hanging out together on a nice day <END>', '<START> tw
[['a man is a bite of his picture taken <END>', 'a man is <UNK> with a donut in his hand <END>', 'a cat sitting with a <UNK> <END>']]
```

```
{'img1': ['<START> many people standing near boxes of many apples <END>'], 'img2': ['<START> a man standing at home plate preparing to
{'img1': ['many people standing near boxes of many apples <END>'], 'img2': ['a man standing at home plate preparing to bat <END>'], 'in

{'img1': ['<START> a yellow wall and a white bed in a room <END>'], 'img2': ['<START> three girls hanging out together
{'img1': ['a man is a bite of his picture taken <END>'], 'img2': ['a man is <UNK> with a donut in his hand <END>'], 'in
```

LSTM 모델의 CIDEr score 는 다음과 같다.

```
scorer = Cider()
score, _ = scorer.compute_score(train_gt_caption_dict, train_sample_caption_dict)
print('Training CIDEr score: ', score)
```

```
Training CIDEr score:  9.488565900393462
```

```
scorer = Cider()
score, _ = scorer.compute_score(val_gt_caption_dict, val_sample_caption_dict)
print('Validating CIDEr score: ', score)
```

```
Validating CIDEr score:  0.4212701105390278
```

### ➢ Transformer

captions 의 dictionary 형태는 다음과 같다.

```
print(train_gt_caption)
print(train_sample_caption)
print(val_gt_caption)
print(val_sample_caption)
```

```
[['<START> a group of men riding in a boat across a lake <END>', '<START> a kitchen with cement walls and a curtain over the <UNK>
[['a group of men riding in a boat across a lake <END>', 'a kitchen with cement walls and a curtain over the <UNK> way of the door
[['<START> three animals on the side of a <UNK> rocky hill <END>', '<START> some animals that are standing in the grass together <I
[['half a <UNK> near a <UNK> of a apples <END>', 'a plane truck parked to the ground in the <UNK> in a landing <END>', 'half there

{'img1': ['<START> a group of men riding in a boat across a lake <END>'], 'img2': ['<START> a kitchen with cement wall:
{'img1': ['a group of men riding in a boat across a lake <END>'], 'img2': ['a kitchen with cement walls and a curtain (
{'img1': ['<START> three animals on the side of a <UNK> rocky hill <END>'], 'img2': ['<START> some animals that are stanc
{'img1': ['half a <UNK> near a <UNK> of a apples <END>'], 'img2': ['a plane truck parked to the ground in the <UNK> in a
```

Transformer 모델의 CIDEr score 는 다음과 같다.

```
scorer = Cider()
score, _ = scorer.compute_score(train_gt_caption_dict, train_sample_caption_dict)
print('Training CIDEr score: ', score)
```

```
Training CIDEr score:  9.64346822618489
```

```
scorer = Cider()
score, _ = scorer.compute_score(val_gt_caption_dict, val_sample_caption_dict)
print('Validating CIDEr score: ', score)
```

```
Validating CIDEr score:  0.7173601839769567
```

### ➢ Conclusion

| CIDEr Score | RNN | LSTM | Transformer |
|---|---|---|---|
| **Training** | 약 9.45 | 약 9.5 | 약 9.64 |
| **Validating** | 약 0.3 | 약 0.42 | 약 0.72 |

세 모델의 CIDEr score 를 비교해 보니, 발전된 모델일수록 CIDEr score 가 더 잘 나오는 것을 확인할 수 있었다.

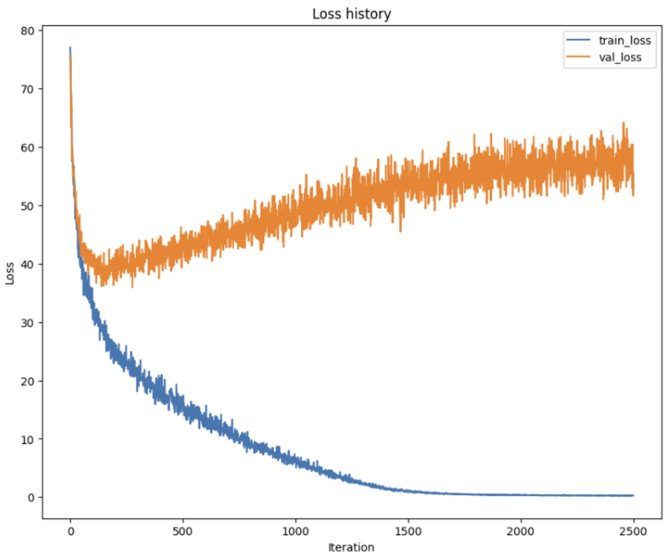
## 3. Improve the performance with your own idea

모델의 성능을 높이기 위해 max_train data 의 개수와 batch_size 를 조정해 보았다.

### ➢ RNN

```
small_data = load_coco_data(max_train=5000)

small_rnn_model = CaptioningRNN(
    cell_type='rnn',
    word_to_idx=data['word_to_idx'],
    input_dim=data['train_features'].shape[1],
    hidden_dim=512,
    wordvec_dim=256,
)

small_rnn_solver = CaptioningSolver(
    small_rnn_model, small_data,
    update_rule='adam',
    num_epochs=50,
    batch_size=100,
    optim_config={
     'learning_rate': 5e-3,
    },
    lr_decay=0.95,
    verbose=True, print_every=100,
)
```

Train data 의 개수를 50 개에서 5000 개로 늘리고, batch_size 를 25 에서 100 으로 수정하였다.



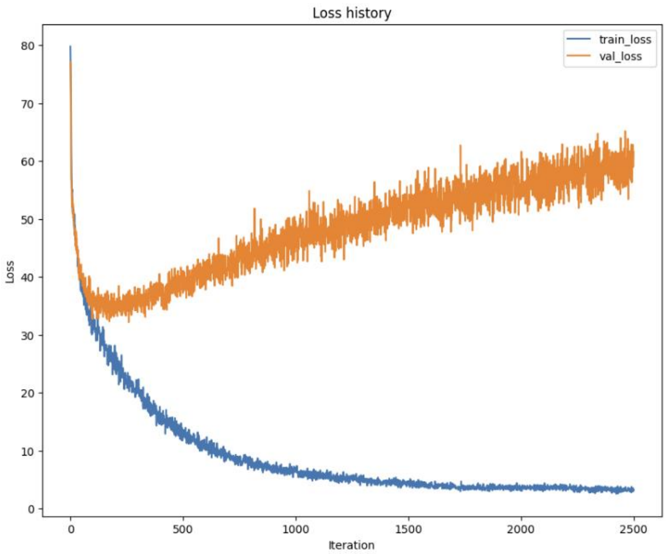

```
Final train loss:  0.28661093475877303
CIDEr score:  0.9512688596438491
```

Trian data 의 개수와 batch_size 를 늘리기 전과 후의 **validation loss 값을 비교해 보면**, 늘린 후의 loss 값이 초반에 많이 줄어드는 것을 볼 수 있다. 전체 dataset 을 학습시킨 것이 아니기 때문에 여전히 overfitting 되어 있지만 그래도 validation loss 값을 보았을 때, 이전보다 성능이 약간은 개선된 것을 확인할 수 있었다. 또한, CIDEr score 가 눈에 띄게 좋아진 것을 볼 수 있다.

## ➢ LSTM

```
small_data = load_coco_data(max_train=5000)

small_lstm_model = CaptioningRNN(
    cell_type='lstm',
    word_to_idx=data['word_to_idx'],
    input_dim=data['train_features'].shape[1],
    hidden_dim=512,
    wordvec_dim=256,
    dtype=np.float32,
)

small_lstm_solver = CaptioningSolver(
    small_lstm_model, small_data,
    update_rule='adam',
    num_epochs=50,
    batch_size=100,
    optim_config={
     'learning_rate': 5e-3,
    },
    lr_decay=0.995,
    verbose=True, print_every=100,
)
```

RNN 과 마찬가지로, train data 의 개수를 50 개에서 5000 개로 늘리고, batch_size 를 25 에서 100 으로 수정하였다.



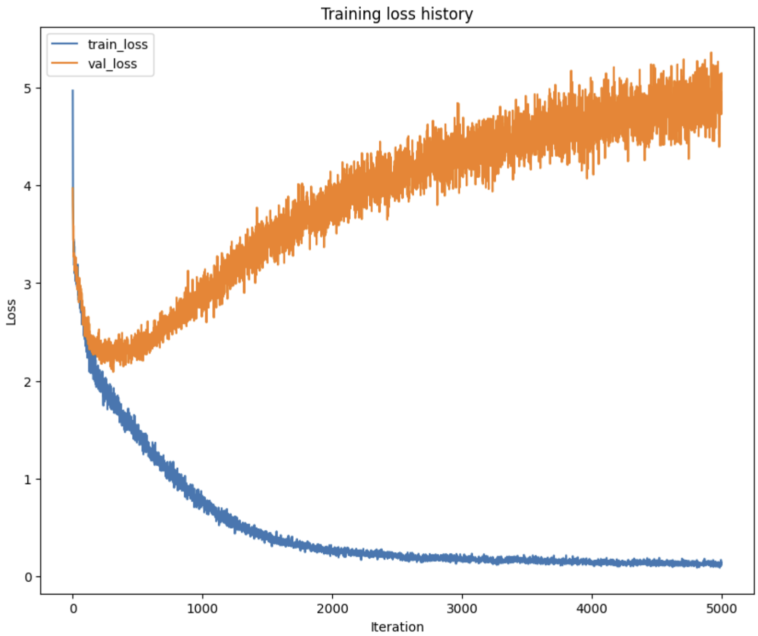

`Final train loss:  3.19336932745147`

Trian data 의 개수와 batch_size 를 늘리기 전과 후의 **validation loss 값을 비교해 보면**, 늘린 후의 loss 값이 초반에 많이 줄어드는 것을 볼 수 있다. 전체 dataset 을 학습시킨 것이 아니기 때문에 여전히 overfitting 되어 있지만 최저점을 찍은 validation loss 값을 보았을 때, 이전보다 성능이 약간은 개선된 것을 확인할 수 있었다.

## ➢ Transformer

```
data = load_coco_data(max_train=5000)

transformer = CaptioningTransformer(
          word_to_idx=data['word_to_idx'],
          input_dim=data['train_features'].shape[1],
          wordvec_dim=256,
          num_heads=2,
          num_layers=2,
          max_length=30
        )


transformer_solver = CaptioningSolverTransformer(transformer, data, idx_to_word=data['idx_to_word'],
           num_epochs=100,
           batch_size=100,
           learning_rate=0.001,
           verbose=True, print_every=100,
         )
```

Train data 개수를 50 개에서 5000 개로 늘리고, batch_size 를 25 에서 100 으로 수정하였다.




```
Final train loss:  0.14153814
```

Trian data 의 개수와 batch_size 를 늘리기 전과 후의 **validation loss 값을 비교해 보면**, 늘린 후의 loss 값이 초반에 많이 줄어드는 것을 볼 수 있다. 전체 dataset 을 학습시킨 것이 아니기 때문에 여전히 overfitting 되어 있지만 그래도 최저점을 찍은 validation loss 값을 보았을 때, 이전보다 성능이 약간은 개선된 것을 확인할 수 있었다.