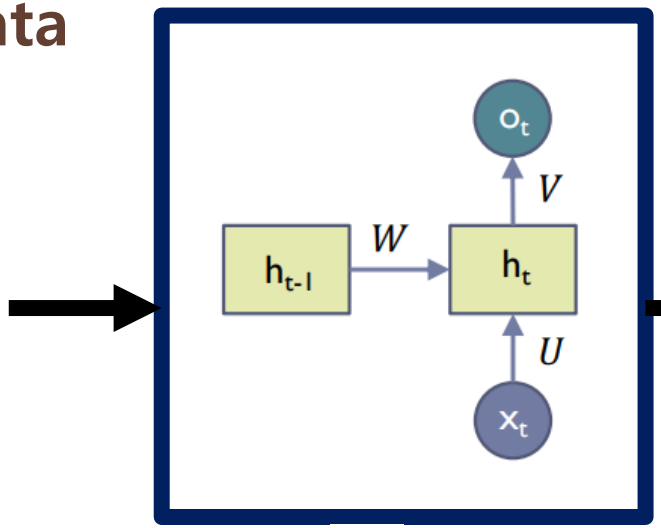
Exercise 5

RNN Basics

- **Sequential data**

1., 2., 3., 4., 5.,
6., 7., 8., 9., 10.,
11., 12., 13., 14., 15.



O_t

$$h_t = f(Ux_t + Wh_{t-1})$$
$$o_t = g(Vh_t)$$

$$\tanh\left(W_{D_h \times D_h} \times h_{t-1,\ D_h \times 1} + U_{D_h \times d} \times x_{t,\ d \times 1} + b_{D_h \times 1}\right) = h_{t,\ D_h \times 1}$$

$$\tanh\left(h_{t,\ D_h \times 1} \times V_{D_h \times 1}\right) = O_t$$

RNN Basics

```
[4]  X = seq.view(batch_size, seq_length, input_size)

     print(X.size())

     torch.Size([3, 5, 1])
```

```
[11] X

     tensor([[[ 1.],
              [ 2.],
              [ 3.],
              [ 4.],
              [ 5.]],

             [[ 6.],
              [ 7.],
              [ 8.],
              [ 9.],
              [10.]],

             [[11.],
              [12.],
              [13.],
              [14.],
              [15.]]])
```

```
[10]  [ (name, param.shape ) for name, param in singleRNN.named_parameters()]

      [('weight_ih_l0', torch.Size([10, 1])),
       ('weight_hh_l0', torch.Size([10, 10])),
       ('bias_ih_l0', torch.Size([10])),
       ('bias_hh_l0', torch.Size([10]))]
```

```
singleRNN = nn.RNN(
    input_size=input_size,
    hidden_size=hidden_size,
    num_layers=num_layers,
    nonlinearity='tanh',
    batch_first=True,
    dropout=0,
    bidirectional=False
)
```

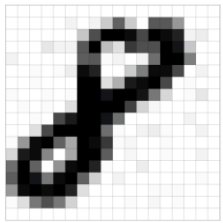
```
y, h = singleRNN(X)

print(y.size())    # (batch_size, seq_length, hidden_size * num_directions)
print(h.size())    # (num_layers * num_directions, batch_size, hidden_size)
```

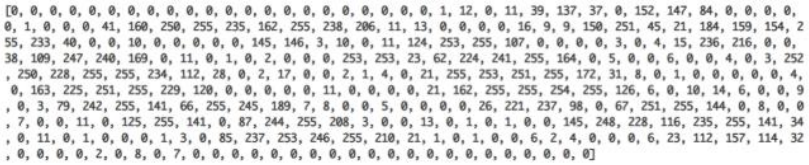
```
torch.Size([3, 5, 10])
torch.Size([1, 3, 10])
```

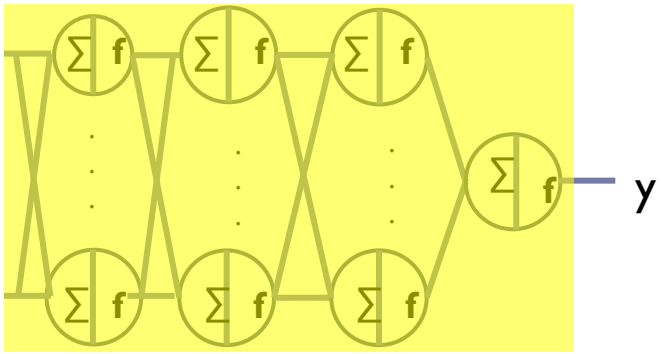
nn.Linear

RECAP

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```



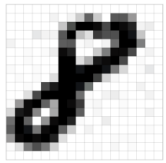


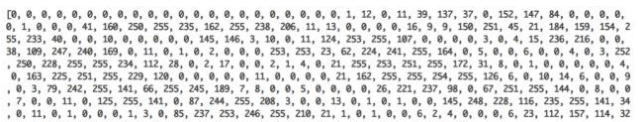


RNN

```
[ ]  class ImageRNN(nn.Module):
        def __init__(self, batch_size, seq_length, input_size, hidden_size, num_layers, num_classes):
            super().__init__()
            self.batch_size = batch_size
            self.seq_length = seq_length
            self.input_size = input_size
            self.hidden_size = hidden_size
            self.num_layers = num_layers
            self.num_classes = num_classes

            self.rnn = nn.RNN(self.input_size, self.hidden_size, self.num_layers, batch_first=True)
            self.fc = nn.Linear(self.hidden_size * self.seq_length, self.num_classes)

        def forward(self, x, h0):
            x = x.view(-1, 28, 28)    # (batch_size, channel, width, height) --> (batch_size, width as seq_length, height * channel as feature)
            out, _ = self.rnn(x, h0)    # (batch_size, seq_length, num_directions * hidden_size)
            out = out.reshape(-1, (self.seq_length * self.hidden_size))    # (batch, seq_length * num_directions * hidden_size)
            outputs = self.fc(out)    # (batch_size, num_classes)
            return outputs
```

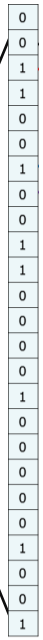
```
seq_length = 28
input_size = 28
hidden_size = 50
num_layers = 1
num_classes = 10
```





RNN

0	0	1	1	0
1	3	0	1	1
1	0	2	1	1
1	1	0	0	1



Linear

0.0125
-0.0664
0.0881
0.178
0.1551
-0.1078
0.0188
-0.2199
-0.1829
0.1543

[1000, 1, 28, 28]

[1000, 28, 50][1000, 28*50]

Transformer

- **dataset**

tokenizer

전처리 객체

데이터셋 생성

정의한 객체(Field)에 단어 집합 만들기

```
spacy_ger = spacy.load("de")
spacy_eng = spacy.load("en")

def tokenize_ger(text):
    return [tok.text for tok in spacy_ger.tokenizer(text)]

def tokenize_eng(text):
    return [tok.text for tok in spacy_eng.tokenizer(text)]


german = Field(tokenize=tokenize_ger, lower=True, init_token="<sos>", eos_token="<eos>")

english = Field(
    tokenize=tokenize_eng, lower=True, init_token="<sos>", eos_token="<eos>"
)

train_data, valid_data, test_data = Multi30k.splits(
    exts=(".de", ".en"), fields=(german, english)
)

german.build_vocab(train_data, max_size=10000, min_freq=2)
english.build_vocab(train_data, max_size=10000, min_freq=2)
```

tokenizer : https://spacy.io/
dataset: Multi30k
torchtext:https://wikidocs.net/60314
http://jalammar.github.io/illustrated-transformer/
https://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer

- **dataset**

```
# Model hyperparameters
src_vocab_size = len(german.vocab)
trg_vocab_size = len(english.vocab)
embedding_size = 512
num_heads = 8
num_encoder_layers = 3
num_decoder_layers = 3
dropout = 0.10
max_len = 100
forward_expansion = 4
src_pad_idx = english.vocab.stoi["<pad>"]

# Tensorboard to get nice loss plot
writer = SummaryWriter("runs/loss_plot")
step = 0

train_iterator, valid_iterator, test_iterator = BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size=batch_size,
    sort_within_batch=True,
    sort_key=lambda x: len(x.src),
    device=device,
)
```

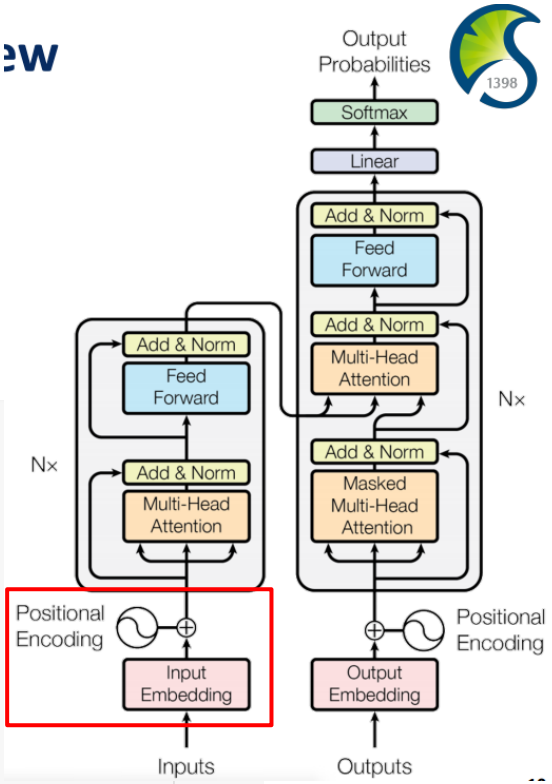
데이터로더 생성

Transformer

- **model**

```python
class Transformer(nn.Module):
    def __init__(
        self,
        embedding_size,
        src_vocab_size,
        trg_vocab_size,
        src_pad_idx,
        num_heads,
        num_encoder_layers,
        num_decoder_layers,
        forward_expansion,
        dropout,
        max_len,
        device,
    ):
        super(Transformer, self).__init__()
        self.src_word_embedding = nn.Embedding(src_vocab_size, embedding_size)
        self.src_position_embedding = nn.Embedding(max_len, embedding_size)
        self.trg_word_embedding = nn.Embedding(trg_vocab_size, embedding_size)
        self.trg_position_embedding = nn.Embedding(max_len, embedding_size)

        self.device = device
        self.transformer = nn.Transformer(
            embedding_size,
            num_heads,
            num_encoder_layers,
            num_decoder_layers,
            forward_expansion,
            dropout,
        )
        self.fc_out = nn.Linear(embedding_size, trg_vocab_size)
        self.dropout = nn.Dropout(dropout)
        self.src_pad_idx = src_pad_idx
```

```python
def forward(self, src, trg):
    src_seq_length, N = src.shape
    trg_seq_length, N = trg.shape

    src_positions = (
        torch.arange(0, src_seq_length)
        .unsqueeze(1)
        .expand(src_seq_length, N)
        .to(self.device)
    )

    trg_positions = (
        torch.arange(0, trg_seq_length)
        .unsqueeze(1)
        .expand(trg_seq_length, N)
        .to(self.device)
    )

    embed_src = self.dropout(
        (self.src_word_embedding(src) + self.src_position_embedding(src_positions))
    )
    embed_trg = self.dropout(
        (self.trg_word_embedding(trg) + self.trg_position_embedding(trg_positions))
    )

    src_padding_mask = self.make_src_mask(src)
    trg_mask = self.transformer.generate_square_subsequent_mask(trg_seq_length).to(
        self.device
    )

    out = self.transformer(
        embed_src,
        embed_trg,
        src_key_padding_mask=src_padding_mask,
        tgt_mask=trg_mask,
    )
    out = self.fc_out(out)
    return out
```



Transformer

- **train**

```
for batch_idx, batch in enumerate(train_iterator):
    # Get input and targets and get to cuda
    inp_data = batch.src.to(device)
    target = batch.trg.to(device)

    # Forward prop
    output = model(inp_data, target[:-1, :])

    output = output.reshape(-1, output.shape[2])
    target = target[1:].reshape(-1)

    optimizer.zero_grad()

    loss = criterion(output, target)
    losses.append(loss.item())

    # Back prop
    loss.backward()

    # Clip to avoid exploding gradient issues, makes sure grads are
    # within a healthy range
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)

    # Gradient descent step
    optimizer.step()

    # plot to tensorboard
    writer.add_scalar("Training loss", loss, global_step=step)
    step += 1

mean_loss = sum(losses) / len(losses)
scheduler.step(mean_loss)
```

torchtext의 데이터 로더

```
type(batch)
```

```
torchtext.data.batch.Batch
```

```
-> inp_data = batch.src.to(device)
(Pdb) type(batch)
<class 'torchtext.data.batch.Batch'>
(Pdb) batch

[torchtext.data.batch.Batch of size 32 from MULTI30K]
        [.src]:[torch.cuda.LongTensor of size 17x32 (GPU 0)]
        [.trg]:[torch.cuda.LongTensor of size 22x32 (GPU 0)]
```

Transformer example

- **test**

```
def bleu(data, model, german, english, device):
    targets = []
    outputs = []

    for example in data:
        src = vars(example)["src"]
        trg = vars(example)["trg"]

        prediction = translate_sentence(model, src, german, english, device)
        prediction = prediction[:-1]  # remove <eos> token

        targets.append([trg])
        outputs.append(prediction)

    return bleu_score(outputs, targets)
```

```
# running on entire test data takes a while
score = bleu(test_data[1:100], model, german, english, device)
print(f"Bleu score {score * 100:.2f}")
```

Question and Answer