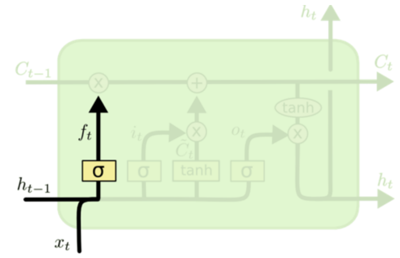
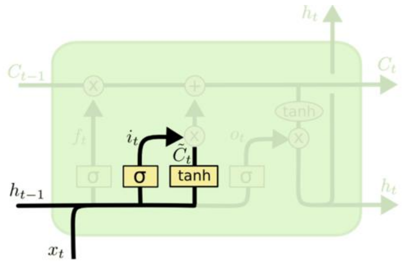
Assignment 3

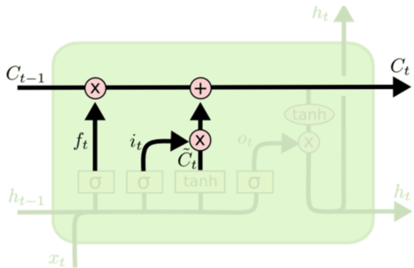
Part 1 Differentiate LSTM



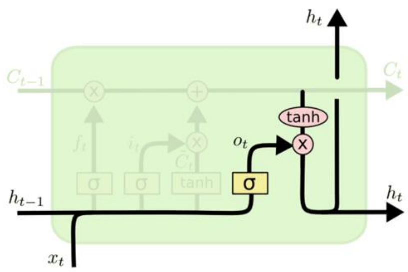
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Lemma:

1. $\sigma(z) = y = \frac{1}{1+e^{-z}}$
2. $\sigma'(z) = y(1-y)$
3. $tanh(z) = y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
4. $tanh'(z) = 1 - y^2$
5. $diag[a]X = a * X$, *where* X *is a matrix, and* a *is the vector composed of all the diag elements of* $diag[a]$
6. $a^T diag[b] = \ a * b$, *where* a *is a column vector and* b *is the vector composed of all the diag elements of* $diag[b]$

Define the loss as E, $\boldsymbol{\delta_t} \stackrel{\text{def}}{=} \frac{\partial E}{\partial h_t}$

$$\mathbf{net}_{f,t} = \boldsymbol{W_f}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_f} = \mathbf{W_{fh}}\boldsymbol{h_{t-1}} + \boldsymbol{W_{fx}}\boldsymbol{x_t} + \boldsymbol{b_f}$$
$$\mathbf{net}_{i,t} = \boldsymbol{W_i}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_f} = \mathbf{W_{ih}}\boldsymbol{h_{t-1}} + \boldsymbol{W_{ix}}\boldsymbol{x_t} + \boldsymbol{b_i}$$
$$\mathbf{net}_{\bar{c},t} = \boldsymbol{W_c}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_c} = \mathbf{W_{ch}}\boldsymbol{h_{t-1}} + \boldsymbol{W_{cx}}\boldsymbol{x_t} + \boldsymbol{b_c}$$
$$\mathbf{net}_{o,t} = \boldsymbol{W_o}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_o} = \mathbf{W_{oh}}\boldsymbol{h_{t-1}} + \boldsymbol{W_{ox}}\boldsymbol{x_t} + \boldsymbol{b_o}$$
$$\boldsymbol{\delta}_{\mathbf{f,t}} \stackrel{\text{def}}{=} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial net_{f,t}}}$$
$$\boldsymbol{\delta}_{\mathbf{i,t}} \stackrel{\text{def}}{=} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial net_{i,t}}}$$
$$\boldsymbol{\delta}_{\bar{c},\mathbf{t}} \stackrel{\text{def}}{=} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial net_{\bar{c},t}}}$$
$$\boldsymbol{\delta}_{\mathbf{o,t}} \stackrel{\text{def}}{=} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial net_{o,t}}}$$

Calculation starts here:

$$\frac{\partial \mathrm{h_t}}{\partial f_t} = \frac{\partial \mathrm{h_t}}{\partial c_t}\frac{\partial c_t}{\partial f_t} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{c_{t-1}}]$$
$$\frac{\partial \mathrm{h_t}}{\partial i_t} = \frac{\partial \mathrm{h_t}}{\partial c_t}\frac{\partial c_t}{\partial i_t} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{\bar{c}_t}]$$
$$\frac{\partial \mathrm{h_t}}{\partial \bar{C_t}} = \frac{\partial \mathrm{h_t}}{\partial c_t}\frac{\partial c_t}{\partial \bar{\mathrm{c}}_\mathrm{t}} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[i_t]$$
$$\frac{\partial \mathrm{h_t}}{\partial C_t} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]$$
$$\frac{\partial \mathrm{h_t}}{\partial C_{t-1}} = \frac{\partial \mathrm{h_t}}{\partial c_t}\frac{\partial c_t}{\partial \mathrm{c_{t-1}}} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[f_t]$$
$$\frac{\partial \mathrm{h_t}}{\partial o_t} = \mathrm{diag}[\tanh(\mathrm{c_t})]$$
$$\frac{\partial \mathrm{h_t}}{\partial h_{t-1}} = \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial net_{o,t}}\frac{\partial net_{o,t}}{\partial h_{t-1}} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial net_{f,t}}\frac{\partial net_{f,t}}{\partial h_{t-1}} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial net_{i,t}}\frac{\partial net_{i,t}}{\partial h_{t-1}} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial \bar{c}_t}\frac{\partial \bar{c}_t}{\partial net_{\bar{c}_t,t}}\frac{\partial net_{\bar{c}_t,t}}{\partial h_{t-1}}$$
$$= \mathrm{diag}[\tanh(\mathrm{c_t})]\mathrm{diag}[\mathrm{o_t} * (1 - \mathrm{o_t})]\mathrm{W_{oh}} + \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{c_{t-1}}]\mathrm{diag}[\mathrm{f_t} * (1 - \mathrm{f_t})]\mathrm{W_{fh}}$$
$$+ \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{\bar{c}_t}]\mathrm{diag}[\mathrm{i_t} * (1 - \mathrm{i_t})]\mathrm{W_{ih}}$$
$$+ \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[i_t]\mathrm{diag}[1 - \mathrm{\bar{c}_t}^2]\mathrm{W_{ch}}$$
$$\frac{\partial \mathrm{h_t}}{\partial x_t} = \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial net_{o,t}}\frac{\partial net_{o,t}}{\partial x_t} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial net_{f,t}}\frac{\partial net_{f,t}}{\partial x_t} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial net_{i,t}}\frac{\partial net_{i,t}}{\partial x_t} + \frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial \bar{c}_t}\frac{\partial \bar{c}_t}{\partial net_{\bar{c}_t,t}}\frac{\partial net_{\bar{c}_t,t}}{\partial x_t}$$
$$= \mathrm{diag}[\tanh(\mathrm{c_t})]\mathrm{diag}[\mathrm{o_t} * (1 - \mathrm{o_t})]\mathrm{W_{ox}} + \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{c_{t-1}}]\mathrm{diag}[\mathrm{f_t} * (1 - \mathrm{f_t})]\mathrm{W_{fx}}$$
$$+ \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{\bar{c}_t}]\mathrm{diag}[\mathrm{i_t} * (1 - \mathrm{i_t})]\mathrm{W_{ix}}$$
$$+ \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[i_t]\mathrm{diag}[1 - \mathrm{\bar{c}_t}^2]\mathrm{W_{cx}}$$
$$\frac{\partial \mathrm{h_t}}{\partial W_f} = \frac{\partial \mathrm{h_t}}{\partial f_t}\frac{\partial f_t}{\partial net_{f,t}}\frac{\partial net_{f,t}}{\partial w_f} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{c_{t-1}}]\mathrm{diag}[\mathrm{f_t} * (1 - \mathrm{f_t})][h_{t-1}, x_t]$$
$$\frac{\partial \mathrm{h_t}}{\partial W_i} = \frac{\partial \mathrm{h_t}}{\partial i_t}\frac{\partial i_t}{\partial net_{i,t}}\frac{\partial net_{i,t}}{\partial w_i} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{\bar{c}_t}]\ \mathrm{diag}[\mathrm{i_t} * (1 - \mathrm{i_t})][h_{t-1}, x_t]$$
$$\frac{\partial \mathrm{h_t}}{\partial W_C} = \frac{\partial h_t}{\partial \bar{c}_t}\frac{\partial \bar{c}_t}{\partial net_{\bar{c}_t,t}}\frac{\partial net_{\bar{c}_t,t}}{\partial W_c} = \mathrm{diag}[\mathrm{o_t} * (1 - \tanh(\mathrm{c_t})^2)]\mathrm{diag}[i_t]\mathrm{diag}\left[1 - \mathrm{\bar{c}_t}^2\right][h_{t-1}, x_t]$$
$$\frac{\partial \mathrm{h_t}}{\partial W_o} = \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial net_{o,t}}\frac{\partial net_{o,t}}{\partial W_o} = \mathrm{diag}[\tanh(\mathrm{c_t})]\mathrm{diag}[\mathrm{o_t} * (1 - \mathrm{o_t})][h_{t-1}, x_t]$$

$$\frac{\partial \mathrm{h_t}}{\partial b_f}=\frac{\partial \mathrm{h_t}}{\partial f_t}\frac{\partial f_t}{\partial net_{f,t}}\frac{\partial net_{f,t}}{\partial b_f}= \mathrm{diag}[\mathrm{o_t}*(1-\tanh(\mathrm{c_t})^2)]\mathrm{diag}[\mathrm{c_{t-1}}]\mathrm{diag}[\mathrm{f_t}*(1-\mathrm{f_t})]$$

$$\frac{\partial \mathrm{h_t}}{\partial b_i}=\frac{\partial \mathrm{h_t}}{\partial i_t}\frac{\partial i_t}{\partial net_{i,t}}\frac{\partial net_{i,t}}{\partial b_i}= \mathrm{diag}[\mathrm{o_t}*(1-\tanh(\mathrm{c_t})^2)]\mathrm{diag}[\bar{\mathrm{c}}_\mathrm{t}]\ \mathrm{diag}[\mathrm{i_t}*(1-\mathrm{i_t})]$$

$$\frac{\partial \mathrm{h_t}}{\partial b_C}=\frac{\partial h_t}{\partial \bar{c}_t}\frac{\partial \bar{c}_t}{\partial net_{\bar{c}_t,t}}\frac{\partial net_{\bar{c}_t,t}}{\partial b_c}= \mathrm{diag}[\mathrm{o_t}*(1-\tanh(\mathrm{c_t})^2)]\mathrm{diag}[i_t]\mathrm{diag}\left[1-\bar{\mathrm{c}}_\mathrm{t}{}^2\right]$$

$$\frac{\partial \mathrm{h_t}}{\partial b_o}=\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial net_{o,t}}\frac{\partial net_{o,t}}{\partial b_o}= \mathrm{diag}[\tanh(\mathrm{c_t})]\mathrm{diag}[\mathrm{o_t}*(1-\mathrm{o_t})]$$

To compute BPTT, first we need to compute $\delta_{t-1}^T=\frac{\partial \mathrm{E}}{\partial h_{t-1}}=\frac{\partial E}{\partial h_t}\frac{\partial \mathrm{h_t}}{\partial h_{t-1}}=\delta_t^T\frac{\partial \mathrm{h_t}}{\partial h_{t-1}}$

$$\delta_t^T\frac{\partial \mathrm{h_t}}{\partial h_{t-1}}=\delta_t^T\frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial net_{o,t}}\frac{\partial net_{o,t}}{\partial h_{t-1}}+\delta_t^T\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial f_t}\frac{\partial f_t}{\partial net_{f,t}}\frac{\partial net_{f,t}}{\partial h_{t-1}}+\delta_t^T\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial i_t}\frac{\partial i_t}{\partial net_{i,t}}\frac{\partial net_{i,t}}{\partial h_{t-1}}+\delta_t^T\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial \bar{c}_t}\frac{\partial \bar{c}_t}{\partial net_{\bar{c}_t,t}}\frac{\partial net_{\bar{c}_t,t}}{\partial h_{t-1}}$$

$$=\delta_{\mathrm{o,t}}^T\frac{\partial net_{o,t}}{\partial h_{t-1}}+\delta_{\mathrm{f,t}}^T\frac{\partial net_{f,t}}{\partial h_{t-1}}+\delta_{\mathrm{i,t}}^T\frac{\partial net_{i,t}}{\partial h_{t-1}}+\delta_{\bar{c}_t,t}^T\frac{\partial net_{\bar{c}_t,t}}{\partial h_{t-1}}=\delta_{\mathrm{o,t}}^T W_{oh}+\delta_{\mathrm{f,t}}^T W_{fh}+\delta_{\mathrm{i,t}}^T W_{ih}+\delta_{\bar{c}_t,t}^T W_{ch}$$

So finally we get:

$$\delta_{t-1}^T=\delta_t^T*[\tanh(c_t)*o_t*(1-o_t)W_{oh}+o_t*(1-\tanh(c_t)^2*c_{t-1}*f_t*(1-f_t)W_{fh}+o_t*(1-\tanh(c_t)^2*\bar{\mathrm{c}}_\mathrm{t}*i_t*(1-i_t)W_{ih}+o_t*(1-\tanh(c_t)^2*i_t*(1-\bar{\mathrm{c}}_\mathrm{t}{}^2)]$$

At time t,

$$\frac{\partial \mathrm{E}}{\partial W_{oh,t}}=\delta_{o,t}h_{t-1}^T,\frac{\partial \mathrm{E}}{\partial W_{fh,t}}=\delta_{f,t}h_{t-1}^T,\frac{\partial \mathrm{E}}{\partial W_{ih,t}}=\delta_{i,t}h_{t-1}^T,\frac{\partial \mathrm{E}}{\partial W_{ch,t}}=\delta_{\bar{c},t}h_{t-1}^T$$

Add all the gradients in each time step:

$$\frac{\partial \mathrm{E}}{\partial W_{oh}}=\sum_{j=1}^{t}\delta_{o,j}h_{j-1}^T,\frac{\partial \mathrm{E}}{\partial W_{fh}}=\sum_{j=1}^{t}\delta_{f,j}h_{j-1}^T,\frac{\partial \mathrm{E}}{\partial W_{ih}}=\sum_{j=1}^{t}\delta_{i,j}h_{j-1}^T,\frac{\partial \mathrm{E}}{\partial W_{ch}}=\sum_{j=1}^{t}\delta_{\bar{c},j}h_{j-1}^T$$

$$\frac{\partial \mathrm{E}}{\partial W_{ox}}=\delta_{o,t}x_t^T,\frac{\partial \mathrm{E}}{\partial W_{fx}}=\delta_{f,t}x_t^T,\frac{\partial \mathrm{E}}{\partial W_{ix}}=\delta_{i,t}x_t^T,\frac{\partial \mathrm{E}}{\partial W_{cx}}=\delta_{\bar{c},t}x_t^T$$

$$\frac{\partial \mathrm{E}}{\partial b_o}=\sum_{j=1}^{t}\delta_{o,j},\frac{\partial \mathrm{E}}{\partial b_i}=\sum_{j=1}^{t}\delta_{i,j},\frac{\partial \mathrm{E}}{\partial b_f}=\sum_{j=1}^{t}\delta_{f,j},\frac{\partial \mathrm{E}}{\partial b_c}=\sum_{j=1}^{t}\delta_{\bar{c},j}$$

Part 2 Autograd Training of LSTM

Dataset: 《全唐诗》, fetched from *https://github.com/chinese-poetry/chinese-poetry*, 10k poems with length ranged from 10 to 63. The dataset is split into training dataset and development dataset (80%:20%) (Please include the json file folder of this github repository in the project as source of data.)

Dictionary： all words in training set, EOS, OOV, START, PAD.

Hyperparameters and training setting:

- Vocabulary size, |V|: 5031
- Batch size, bs: 128
- Sentence length, sl: 64
- Hidden size, hs: 256
- Input size, is: 256

Data preparation:

1. Fetch the data, data cleaning.
   Download the dataset from github, decode the json file and extract the poems with proper length(10-63). The data cleaning procedures include deleting all the content in (),[],{},《》 and all the numbers or strange symbols.
2. Construct vocabulary dictionary, and save it as file for future reference.

```
def wordDic(self, data):  # 增加padding
    words = sorted(set([character for sent in data for character in sent]))
    words.extend(['<EOS>', '<OOV>', '<START>', '<PAD>'])
    word_to_idx = {word: idx for idx, word in enumerate(words)}
    idx_to_word = {idx: word for idx, word in enumerate(words)}
    # save dict
    with open('wordDic', 'wb') as f:
        pickle.dump(word_to_idx, f)
    return word_to_idx, idx_to_word
```

3. Map sentences to sequences of integers.
   Append EOS to the end of the poem. Add paddings to extend the poem to standard length. Mask unknown words in development data with OOV.

LSTM model:

```
class LSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(LSTM, self).__init__()
        # embedding layer
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.hidden_dim = hidden_dim
        # weights for inputs
        self.weight_ih = Parameter(torch.Tensor(embedding_dim, hidden_dim*4))
        # weights for hidden states
        self.weight_hh = Parameter(torch.Tensor(hidden_dim, hidden_dim*4))
        self.bias = Parameter(torch.Tensor(hidden_dim*4))
        self.init_weights()
        # linear transformation
        self.linear = nn.Linear(self.hidden_dim, vocab_size)
        # logsoftmax, for loss calculation
        self.softmax = nn.LogSoftmax()

    def init_weights(self):
        for p in self.parameters():
            if p.data.ndimension() >= 2:
                nn.init.xavier_uniform_(p.data)
            else:
                nn.init.zeros_(p.data)
```

```
    def forward(self, input, hidden=None):
        # input(batch, seq_length, input_size)
        length = input.size()[1]
        batch_size = input.size()[0]
        embeds = self.embeddings(input).view((batch_size, length, -1))
        bs, seq_sz, _ = embeds.size()
        hidden_seq = []
        if hidden is None:
            h_t, c_t = (torch.zeros(self.hidden_dim).to(embeds.device), torch.zeros(self.hidden_dim).to(embeds.device))
        else:
            h_t, c_t = hidden

```

```
        for t in range(seq_sz):
            x_t = embeds[:, t, :]
            gates = x_t @ self.weight_ih + h_t @ self.weight_hh + self.bias
            i_t, f_t, g_t, o_t = gates.chunk(4, 1)
            i_t = torch.sigmoid(i_t)
            f_t = torch.sigmoid(f_t)
            g_t = torch.tanh(g_t)
            o_t = torch.sigmoid(o_t)
            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)
            hidden_seq.append(h_t.unsqueeze(0))

        hidden_seq = torch.cat(hidden_seq, dim=0)
        # resize to shape(batch, seq_len, input_size)
        hidden_seq = hidden_seq.transpose(0, 1)

        output = F.relu(self.linear(hidden_seq))
        if output.size()[1] == 1:  # seq_length=1, 会导致softmax出问题 [1,1,V] 要先改成[1,V]
            output = output[:, 0, :]
            output = self.softmax(output)
            output = output.view(1, 1, -1)
            return output, (h_t, c_t)
        output = self.softmax(output)
        return output, (h_t, c_t)
```

Train & Test:

```
D = data_processing.Dataset_batched_padding()
src = './model_padding_10k/'  # 存放模型

model = LSTM(len(D.word_to_ix), 256, 256)
optimizer = optim.Adam(model.parameters(), lr=0.01, weight_decay=0.0001)
# together with the softmax layer in the model, compute CrossEntrophy loss
criterion = nn.NLLLoss()


epochNum = 50
TRAINSIZE = len(D.train)
TESTSIZE = len(D.develop)
batch = 128

def compute_perplexity(output, target):  # 计算每个句子的困惑度
    N = len(target)
    return 2**(-1/N*sum([output[j][target[j]] for j in range(len(target))]))

```

```python
def test():
    model.eval()
    loss = 0
    perplexity = 0

    for batchIndex in range(int(TESTSIZE / batch)):
        if batchIndex*batch >= TESTSIZE:
            continue
        t, o = D.makeForOneBatch(D.develop[batchIndex*batch:min((batchIndex+1)*batch, TESTSIZE)])
        output, hidden = model(t)
        for i in range(o.size()[0]):
            loss += criterion(output[i, :, :], o[i, :])
            perplexity += compute_perplexity(output[i, :, :], o[i, :])

    loss = loss / TESTSIZE
    perplexity = perplexity / TESTSIZE
    print("=====", loss.item())
    print("+++++", perplexity.item())
    return loss.item()


```

```python
print("start training")
for epoch in range(epochNum):
    model.train()
    for batchIndex in range(int(TRAINSIZE / batch)):
        if batchIndex*batch >= TRAINSIZE:
            continue
        model.zero_grad()
        t, o = D.makeForOneBatch(D.train[batchIndex*batch:min((batchIndex+1)*batch, TRAINSIZE)])
        output, hidden = model(t)
        loss = 0
        for i in range(o.size()[0]):
            loss += criterion(output[i, :, :], o[i, :])
        loss = loss / o.size()[0]
        loss.backward()
        print(epoch, loss.item())
        optimizer.step()

    test_loss = test()
    torch.save(model, src+'poetry-gen-epoch%d-loss%f.pt' % (epoch+1, test_loss))
```

Poetry generation:

```python
def generate(startWord='<START>', max_length=95):
    model = torch.load('./model_padding_10k/poetry-gen-epoch38-loss4.066066.pt')
    with open('wordDic', 'rb') as f:
        word_to_ix = p.load(f)
    ix_to_word = invert_dict(word_to_ix)
    input = make_index(startWord, word_to_ix)
    poem = ""
    if startWord != '<START>':
        poem = startWord
    hidden = None
    for i in range(max_length):
        output, hidden = model(input, hidden)
        topvs, topis = output[0].data.topk(2)
        topi = topis[0][0].item()
        topi1 = topis[0][1].item()
        w = ix_to_word[topi]
        w1 = ix_to_word[topi1]
        if w == '<EOS>': # generate new sentence by random sampling
            w = ix_to_word[numpy.random.randint(low=2, high=len(ix_to_word)-4)]
            hidden = None
        if w != '<EOS>':
            poem += w
        input = make_index(w, word_to_ix)
    return poem
```

Initialization: why we shouldn't initialize all the parameters to zero?

When we are training a model, in fact we are trying to update and find the best weights so that the model can find the hidden relationship between input and output. **If all the weights are initialized to zero, then every neuron in the same layer will have the same output value. When we do back propagation, all the gradients are the same, too. Therefore, the neurons cannot learn different features, and the model is likely to fail.**

Proper way to initialize parameters:

A proper initialization should avoid two things: excessive saturation of activation functions(the gradients will not propagate well), and overly linear units. **Xavier initialization** ensures that the variances of the gradients on the weights is the same for all layers. Assume that the size of layer i is n_i, the initialization can be in the form: $W \sim U[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}]$

Final perplexity and some of the models:

```
48 4.403109550476074
48 4.105821132659912
48 4.1353888511657715
48 4.054264545440674
48 4.121681213378906
48 4.101590633392334
48 4.087003707885742
48 4.108267784118652
48 4.042253494262695
48 4.250035285949707
48 4.16510534286499
48 4.160760402679443
48 4.147387504577637
48 4.411069393157959
48 4.185972213745117
48 4.450275421142578
48 4.17116117477417
48 4.194272041320801
48 4.112797737121582
=====loss: 4.074352264404297
+++++perplexity: 18.452558517456055
```

名称	修改日期	类型
poetry-gen-epoch48-loss4.074352.pt	2019/5/21 20:57	PT 文件
poetry-gen-epoch47-loss4.056911.pt	2019/5/20 11:22	PT 文件
poetry-gen-epoch46-loss4.068842.pt	2019/5/20 10:47	PT 文件
poetry-gen-epoch45-loss4.072767.pt	2019/5/20 10:13	PT 文件
poetry-gen-epoch44-loss4.066272.pt	2019/5/20 9:38	PT 文件
poetry-gen-epoch43-loss4.063836.pt	2019/5/20 9:03	PT 文件
poetry-gen-epoch42-loss4.066301.pt	2019/5/20 0:24	PT 文件
poetry-gen-epoch41-loss4.062868.pt	2019/5/20 0:02	PT 文件
poetry-gen-epoch40-loss4.060951.pt	2019/5/19 23:40	PT 文件
poetry-gen-epoch39-loss4.059323.pt	2019/5/19 23:18	PT 文件
poetry-gen-epoch38-loss4.066066.pt	2019/5/19 22:56	PT 文件
poetry-gen-epoch37-loss4.062064.pt	2019/5/19 22:33	PT 文件
poetry-gen-epoch36-loss4.061742.pt	2019/5/19 22:11	PT 文件
poetry-gen-epoch35-loss4.067415.pt	2019/5/19 21:48	PT 文件

The poems generated by the model:（at different epochs）

```
C:\Myprogram\python3.6.7\python.exe "C:/Myprogram/Pycharm Projects/Autograd_lstm/sample_batched.py"
C:\Myprogram\Pycharm Projects\Autograd_lstm\model_batched.py:107: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  output = self.softmax(output)
日暮雲霞雪，天明月下天。凍衣裳落日，不得此時時。湛露寒雲下，風風動雪聲。芰天涯遠望，天子是誰家。蒲葉落花落，江南風柳風。補風吹雪雪，江上見天風。褪君不見此，不得不知君。殢風吹落葉，江水不飛天。
紅燭滿堂花，風風動白雲。童子子家家，天涯不得來。又逢君子不，不得此時時。舊國風吹落，天風風動來。賚君不見此，不得不知君。胤君不見此，不得不知君。撫君王子，不得其天。陵路遠山水，青山不可飛。掉首
山川水上天，天子不能聞。瀉君不見此，不得不知君。救君不見此，不得不知君。凋風吹落葉，落葉不成風。堤上天高望，天涯不得歸。真子不知此，我心不知君。挹君此去，不得其然。玫君不見此，不得不知君。誠知
夜月明明月，天明不見人。驎君不見此，不得不知君。鬟風吹落葉，江水不飛天。彤城南望天，天子不能聞。印衣裳上，天子不成。攤子風吹，不復天天。窠花落花落，不得風風吹。懿君王子，不得天子。伯馬行人不，
湖上天涯遠，天涯此日來。隮風吹雪雪，江上見天風。違君不見此，不得不能聞。曜有詩人事，不知此時時。概然此去，誰復不知。責君不見此，不得不知君。伴有詩人事，誰家不得時。吸日暮雲去，天涯不可飛。暢心
海上天涯遠，天涯此日來。慧子風吹，天子其然。獯君不見此，不得不知君。蒙詩書自得，我士不能言。證道風流落，天明月下天。畝中有所思，我有此時時。盱君不見此，不得不知君。振衣裳上，天子不成。窸聲落日
月明天子，天子是天。迪君不見此，不得不知君。批君不見此，不得不知君。馳道風流水，天涯不見人。竭風吹落葉，江水不飛天。面下天高望，天明月下來。歙風吹落葉，江水不飛天。猱子風吹，天子其然。孰知此時
春風吹柳色，江上見春風。袖中有雪下，不得一聲聲。坦衣裳上，天子不成。組不知君不得，誰家不得知君。倪君不見此，不得不知君。源中有所見，我有此時時。圖書自有所，我士不知君。驥子風吹，天子其然。晦日
花落花開樹，風風柳上風。元君不見此，不得不能聞。宙上天高望，天涯不得歸。狹下天高望，天明月下來。簾馬行人不見君，不知天子不能聞。沌風吹落葉，江水不飛天。黜落花開處，風風柳上風。弘者不知此，天子
秋風吹落葉，江水不飛天。磬聲聲落，天子不成。建章天子，天子是天。透簾開翠露，天子不能聞。沃陵風景已，天子月明明。鷗馬行人不見君，不知天子不能聞。枚君不見此，不得不知君。葡君不見此，不得不知君。

Process finished with exit code 0
```

```
C:\Myprogram\python3.6.7\python.exe "C:/Myprogram/Pycharm Projects/Autograd_lstm/sample_batched.py"
C:\Myprogram\Pycharm Projects\Autograd_lstm\model_batched.py:107: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  output = self.softmax(output)
日暮雲霞雪，風風動雪聲。贊風吹落葉，江水不悠悠。皆風吹落葉，江水不悠悠。鬃子風吹露，風風動此聲。溱風吹落葉，江水不悠悠。阮家家子，無事不知。崐風吹落葉，江水不悠悠。萌馬行人不，此時不可憐。驍子
紅燭滿堂中，風風動玉衣。楠花落葉落花落，不得風聲入玉樓。菽中有所思，我我不知此。冊君不見，我心不知。惻君王子，不得其然。故人間有所，我事不知君。揚得知君子，不知此去年。愚子不知，我其其然。貴君
山川風吹雪，江上月明明。邂啼花落落，不得風風吹。睟風吹落葉，江水不悠悠。彰君不見，我心不知。縫衣裳上，天子不成。惑子風吹，不見天子。昇天子子，我我其然。笥風吹落葉，江水不悠悠。綫子風吹露，風風
夜月明明月，風風動玉衣。彝馬行人不，此時不可憐。小人心自有，我此不知君。礪君不見，我心不知。橘花開處處，風柳半天風。搖風吹落葉，江水不悠悠。碾雪中天下，天風動雪飛。漢陽城下有餘風，天子無聲不得
湖上天涯遠，天涯不可憐。輅下天涯遠，天南天子歸。腕君不見，我心不知。伐風吹落葉，江水不悠悠。孫子不知，我其其然。鵨風吹落葉，江水不悠悠。蘅風吹落葉，江水不悠悠。澱衣裳上雪，天子不知人。璧衣裳落
海上風吹雪，江南風柳風。激君不知，我心不知。掉首望鄉處，不知此去時。揣君不見，我心不知。諧君不見，我心不知。助有餘年少，不知此去來。庵下有餘風，天子不得來。亘子風吹露，風風動此聲。耽酒不知，我
月明明月照，江上天風風。嫔下天涯遠，天涯不可憐。袋風吹落葉，江水不悠悠。匝君不見，我心不知。鋋風吹雪雪，江上雪天秋。躑聲聲遠，天子不知。西南望鄉路，不得一聲聲。麞風吹落葉，江水不悠悠。檝城南望
春風吹柳色，江水綠花開。淺不知君不得，不知此去不知。薢花落葉落花落，不得風聲入玉樓。劃風吹落葉，江水不悠悠。頎子風吹露，風風動此聲。厭君王子，不得其然。譏風吹落葉，江水不悠悠。瘗玉琴聲，不見天
花落花開盡，風風柳色飛。鸁風吹落葉，江水不悠悠。俊人不見，我心不知。勉君不知，我心不知。鞮風吹落葉，江水不悠悠。挂衣裳上，風吹雪天。着子君無事，我心不得知。珠簾下翠露，天子不知人。瑀君不見，我
秋風吹落葉，江水不悠悠。稽風吹落葉，江水不悠悠。疇君不見，我心不知。遭年年少少，不得不能歸。湄風吹落葉，江水不悠悠。謨下天涯遠，天涯不可憐。復聞聲聲，不見天子。閭門外人不，此日不能歸。鸖聲不可

Process finished with exit code 0
```

日暮雲霞雪，天明月下天。俄有餘花落，不知此去來。

紅燭下花落，不知風露清。柳色青青草，風流不可飛。隕水無窮事，何年不得歸。漢陽城下有，我有此中來。

山川風吹雪，江上月明明。莫風吹落葉，落葉不蕭條。

夜雨深山月暮，無人不待風。煖風吹落露枝，不知不得離。

夜雨聲聲落露，不知不得離人。無事不知此，不知無所思。

湖上有餘，此心無所。荻花落盡日，風景不可飛。慘別離心不，此時無所情。

海上天涯遠，天涯此去來。船下有餘風，天明不可見。

月明朝日暮，天子不知來。荊州城下雪，不見青青天。匰風吹落日，不得一聲聲。莫問君王子，誰家不見人。

月明明月下，不見此時來。踈柳花枝落日斜，不知此日望南來。

numpy implementation:

```python
class Sigmoid(object):
    def forward(self, weighted_input):
        return 1.0 / (1.0 + np.exp(-weighted_input))

    def backward(self, output):
        return output * (1 - output)


class Tanh(object):
    def forward(self, weighted_input):
        return 2.0 / (1.0 + np.exp(-2 * weighted_input)) - 1.0

    def backward(self, output):
        return 1 - output * output


class Identity(object):
    def forward(self, weighted_input):
        return weighted_input

    def backward(self, output):
        return 1
```

```python
class LstmLayer(object):
    def __init__(self, input_size, hidden_size,
                 learning_rate):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.learning_rate = learning_rate
        # 门的激活函数
        self.gate_activator = Sigmoid()
        # 输出的激活函数
        self.output_activator = Tanh()
        # 当前时刻初始化为t0
        self.times = 0
        # 各个时刻的单元状态向量c
        self.c_list = self.init_state_list()
        # 各个时刻的输出向量h
        self.h_list = self.init_state_list()
        # 各个时刻的遗忘门f
        self.f_list = self.init_state_list()
        # 各个时刻的输入门i
        self.i_list = self.init_state_list()
        # 各个时刻的输出门o
        self.o_list = self.init_state_list()
        # 各个时刻的即时状态c~
        self.ct_list = self.init_state_list()
        # 遗忘门权重矩阵Wfh, Wfx, 偏置项bf
        self.Wfh, self.Wfx, self.bf = (
            self.init_weight_mat())
        # 输入门权重矩阵Wfh, Wfx, 偏置项bf
        self.Wih, self.Wix, self.bi = (
            self.init_weight_mat())
        # 输出门权重矩阵Wfh, Wfx, 偏置项bf
        self.Woh, self.Wox, self.bo = (
            self.init_weight_mat())
        # 单元状态权重矩阵Wfh, Wfx, 偏置项bf
        self.Wch, self.Wcx, self.bc = (
            self.init_weight_mat())

    def init_state_list(self):
        # 初始化保存状态的向量
        state_vec_list = []
        state_vec_list.append(np.zeros(
            (self.hidden_size, 1)))
        return state_vec_list

    def init_weight_mat(self):
        # 初始化权重矩阵
        Wh = np.random.uniform(-1e-4, 1e-4,
                               (self.hidden_size, self.hidden_size))
        Wx = np.random.uniform(-1e-4, 1e-4,
                               (self.hidden_size, self.input_size))
        b = np.zeros((self.hidden_size, 1))
        return Wh, Wx, b
```

```
    def forward(self, x):
        # 前向计算
        self.times += 1
        # 遗忘门
        fg = self.calc_gate(x, self.Wfx, self.Wfh,
                            self.bf, self.gate_activator)
        self.f_list.append(fg)
        # 输入门
        ig = self.calc_gate(x, self.Wix, self.Wih,
                            self.bi, self.gate_activator)
        self.i_list.append(ig)
        # 输出门
        og = self.calc_gate(x, self.Wox, self.Woh,
                            self.bo, self.gate_activator)
        self.o_list.append(og)
        # 即时状态
        ct = self.calc_gate(x, self.Wcx, self.Wch,
                            self.bc, self.output_activator)
        self.ct_list.append(ct)
        # 单元状态
        c = fg * self.c_list[self.times - 1] + ig * ct
        self.c_list.append(c)
        # 输出
        h = og * self.output_activator.forward(c)
        self.h_list.append(h)

    def calc_gate(self, x, Wx, Wh, b, activator):
        # 计算门
        h = self.h_list[self.times - 1]  # 上次的LSTM输出
        net = np.dot(Wh, h) + np.dot(Wx, x) + b
        gate = activator.forward(net)
        return gate

    def backward(self, x, delta_h, activator):
        # 反向传播，实现LSTM训练算法
        self.calc_delta(delta_h, activator)
        self.calc_gradient(x)

```

```python
    def update(self):
        # 按照梯度下降，更新权重
        self.Wfh -= self.learning_rate * self.Whf_grad
        self.Wfx -= self.learning_rate * self.Whx_grad
        self.bf -= self.learning_rate * self.bf_grad
        self.Wih -= self.learning_rate * self.Whi_grad
        self.Wix -= self.learning_rate * self.Whi_grad
        self.bi -= self.learning_rate * self.bi_grad
        self.Woh -= self.learning_rate * self.Wof_grad
        self.Wox -= self.learning_rate * self.Wox_grad
        self.bo -= self.learning_rate * self.bo_grad
        self.Wch -= self.learning_rate * self.Wcf_grad
        self.Wcx -= self.learning_rate * self.Wcx_grad
        self.bc -= self.learning_rate * self.bc_grad

    def calc_delta(self, delta_h, activator):
        # 初始化各个时刻的误差项
        self.delta_h_list = self.init_delta()  # 输出误差项
        self.delta_o_list = self.init_delta()  # 输出门误差项
        self.delta_i_list = self.init_delta()  # 输入门误差项
        self.delta_f_list = self.init_delta()  # 遗忘门误差项
        self.delta_ct_list = self.init_delta()  # 即时输出误差项

        # 保存从上一层传递下来的当前时刻的误差项
        self.delta_h_list[-1] = delta_h

        # 迭代计算每个时刻的误差项
        for k in range(self.times, 0, -1):
            self.calc_delta_k(k)

    def init_delta(self):
        # 初始化误差项
        delta_list = []
        for i in range(self.times + 1):
            delta_list.append(np.zeros(
                (self.hidden_size, 1)))
        return delta_list

```

```python
    def calc_delta_k(self, k):
        # 根据k时刻的delta_h，计算k时刻的delta_f、delta_i、delta_o、delta_ct，以及k-1时刻的delta_h
        # 获得k时刻前向计算的值
        ig = self.i_list[k]
        og = self.o_list[k]
        fg = self.f_list[k]
        ct = self.ct_list[k]
        c = self.c_list[k]
        c_prev = self.c_list[k - 1]
        tanh_c = self.output_activator.forward(c)
        delta_k = self.delta_h_list[k]

        delta_o = (delta_k * tanh_c *
                   self.gate_activator.backward(og))
        delta_f = (delta_k * og *
                   (1 - tanh_c * tanh_c) * c_prev *
                   self.gate_activator.backward(fg))
        delta_i = (delta_k * og *
                   (1 - tanh_c * tanh_c) * ct *
                   self.gate_activator.backward(ig))
        delta_ct = (delta_k * og *
                    (1 - tanh_c * tanh_c) * ig *
                    self.output_activator.backward(ct))
        delta_h_prev = (
                np.dot(delta_o.transpose(), self.Woh) +
                np.dot(delta_i.transpose(), self.Wih) +
                np.dot(delta_f.transpose(), self.Wfh) +
                np.dot(delta_ct.transpose(), self.Wch)
        ).transpose()

        # 保存全部delta值
        self.delta_h_list[k - 1] = delta_h_prev
        self.delta_f_list[k] = delta_f
        self.delta_i_list[k] = delta_i
        self.delta_o_list[k] = delta_o
        self.delta_ct_list[k] = delta_ct

```

```python
    def calc_gradient(self, x):
        # 初始化遗忘门权重梯度矩阵和偏置项
        self.Wfh_grad, self.Wfx_grad, self.bf_grad = (
            self.init_weight_gradient_mat())
        # 初始化输入门权重梯度矩阵和偏置项
        self.Wih_grad, self.Wix_grad, self.bi_grad = (
            self.init_weight_gradient_mat())
        # 初始化输出门权重梯度矩阵和偏置项
        self.Woh_grad, self.Wox_grad, self.bo_grad = (
            self.init_weight_gradient_mat())
        # 初始化单元状态权重梯度矩阵和偏置项
        self.Wch_grad, self.Wcx_grad, self.bc_grad = (
            self.init_weight_gradient_mat())

        # 计算对上一次输出h的权重梯度
        for t in range(self.times, 0, -1):
            # 计算各个时刻的梯度
            (Wfh_grad, bf_grad,
             Wih_grad, bi_grad,
             Woh_grad, bo_grad,
             Wch_grad, bc_grad) = (
                self.calc_gradient_t(t))
            # 实际梯度是各时刻梯度之和
            self.Wfh_grad += Wfh_grad
            self.bf_grad += bf_grad
            self.Wih_grad += Wih_grad
            self.bi_grad += bi_grad
            self.Woh_grad += Woh_grad
            self.bo_grad += bo_grad
            self.Wch_grad += Wch_grad
            self.bc_grad += bc_grad

        # 计算对本次输入x的权重梯度
        xt = x.transpose()
        self.Wfx_grad = np.dot(self.delta_f_list[-1], xt)
        self.Wix_grad = np.dot(self.delta_i_list[-1], xt)
        self.Wox_grad = np.dot(self.delta_o_list[-1], xt)
        self.Wcx_grad = np.dot(self.delta_ct_list[-1], xt)
```

```
    def init_weight_gradient_mat(self):
        # 初始化权重矩阵
        Wh_grad = np.zeros((self.hidden_size,
                            self.hidden_size))
        Wx_grad = np.zeros((self.hidden_size,
                            self.input_size))
        b_grad = np.zeros((self.hidden_size, 1))
        return Wh_grad, Wx_grad, b_grad

    def calc_gradient_t(self, t):
        # 计算每个时刻t权重的梯度
        h_prev = self.h_list[t - 1].transpose()
        Wfh_grad = np.dot(self.delta_f_list[t], h_prev)
        bf_grad = self.delta_f_list[t]
        Wih_grad = np.dot(self.delta_i_list[t], h_prev)
        bi_grad = self.delta_f_list[t]
        Woh_grad = np.dot(self.delta_o_list[t], h_prev)
        bo_grad = self.delta_f_list[t]
        Wch_grad = np.dot(self.delta_ct_list[t], h_prev)
        bc_grad = self.delta_ct_list[t]
        return Wfh_grad, bf_grad, Wih_grad, bi_grad, \
               Woh_grad, bo_grad, Wch_grad, bc_grad

    def reset_state(self):
        # 当前时刻初始化为t0
        self.times = 0
        # 各个时刻的单元状态向量c
        self.c_list = self.init_state_list()
        # 各个时刻的输出向量h
        self.h_list = self.init_state_list()
        # 各个时刻的遗忘门f
        self.f_list = self.init_state_list()
        # 各个时刻的输入门i
        self.i_list = self.init_state_list()
        # 各个时刻的输出门o
        self.o_list = self.init_state_list()
        # 各个时刻的即时状态c~
        self.ct_list = self.init_state_list()
```

Optimization: Comparing two optimizers

- Adagrad: Adagrad is an Adaptive Gradient Method that implies different adaptive learning rates for each feature. Hence it is more intuitive for especially sparse problems and it is likely to find more discriminative features. Although we provide an initial learning rate, Adagrad tunes it regarding the history of the gradients for each feature dimension. The formulation for Adagrad is as below:

  $$\mathrm{W_{t+1}} = W_t - \frac{\eta_0}{\sqrt{\sum_{t'=1}^{t}\left(g_{t',i}\right) + \epsilon}} \odot g_{t,i}$$

  The upper formula states that, for each feature dimension, learning rate s divided by all the squared root gradient history. **The main strengths of Adagrad is that we don't need to adjust the learning rate by ourselves. However, as the iteration times become larger, the learning rate will become smaller and finally approaches zero.**

- Adam: Adam is derived from adaptive moment estimation, and is very beneficial when it's used on non-convex optimization problems. Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The formulation of Adam is as below:

  $$\mathrm{m_t} = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
  $$\mathrm{v_t} = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}, \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\widehat{v_t}+\epsilon}}\widehat{m_t}$$

m_t, v_t is first and second moments of the gradients.

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

Optimizer	Strength	Weakness
Adagrad	In the context of sparse data, Adagrad can better use the information from sparse gradients, so it can converge quicker than standard SGD.	As iteration times become larger, the denominator becomes larger and larger, finally the learning rate will become too small and the gradients cannot be updated efficiently.
Adam	Adam is easy to implement and efficient in calculation. The hyperparameters have great interpretability. It can also change the learning rate automatically, so it's especially suitable for unstable target function.	

Therefore, Adam might be the best overall choice.

These two optimizers will influence our gradient calculation in that, the gradients update are not merely determined by initial learning rate, but also by other variables like moment or history weights. So when we are calculating the gradients, we have to store additional parameters.