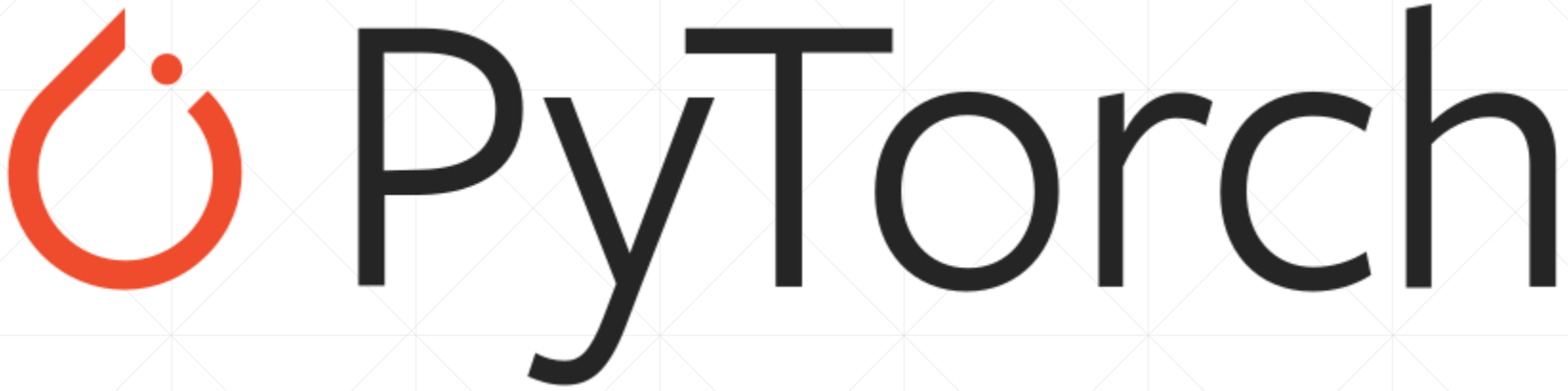
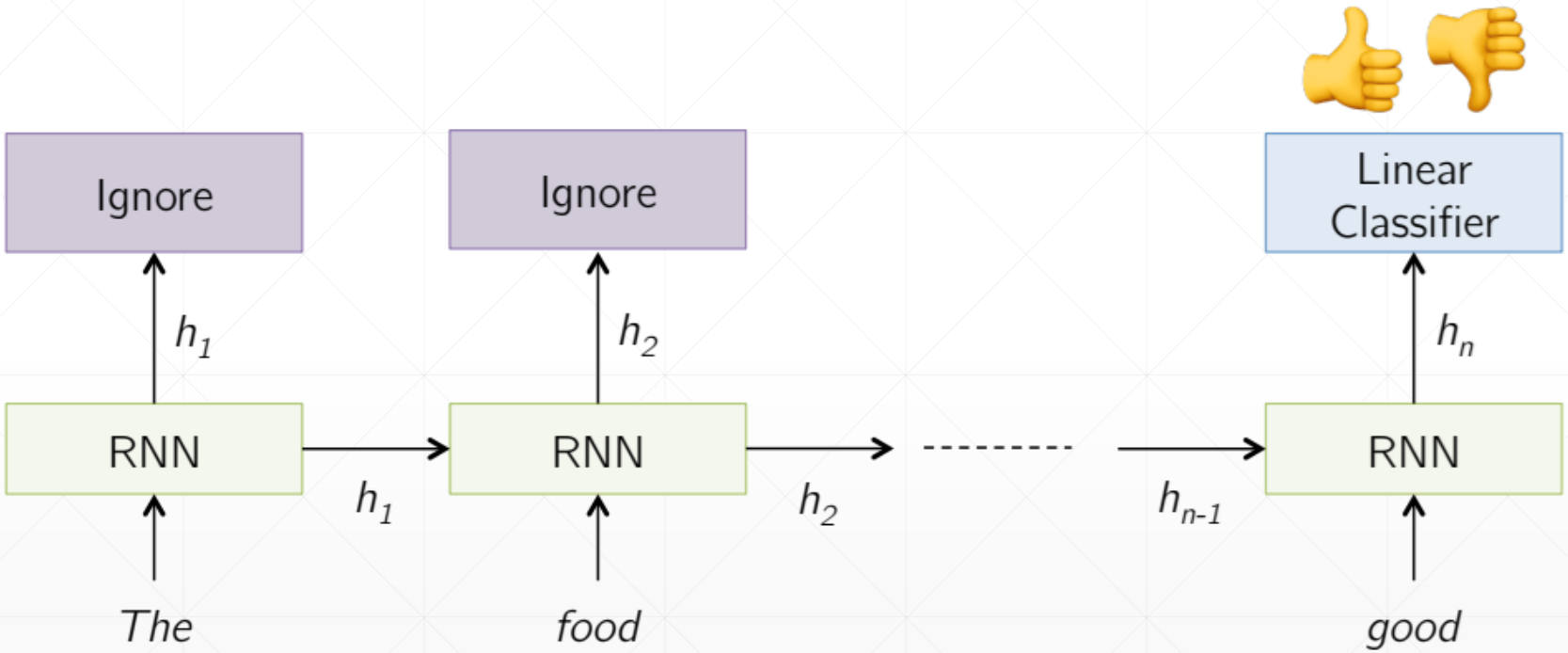


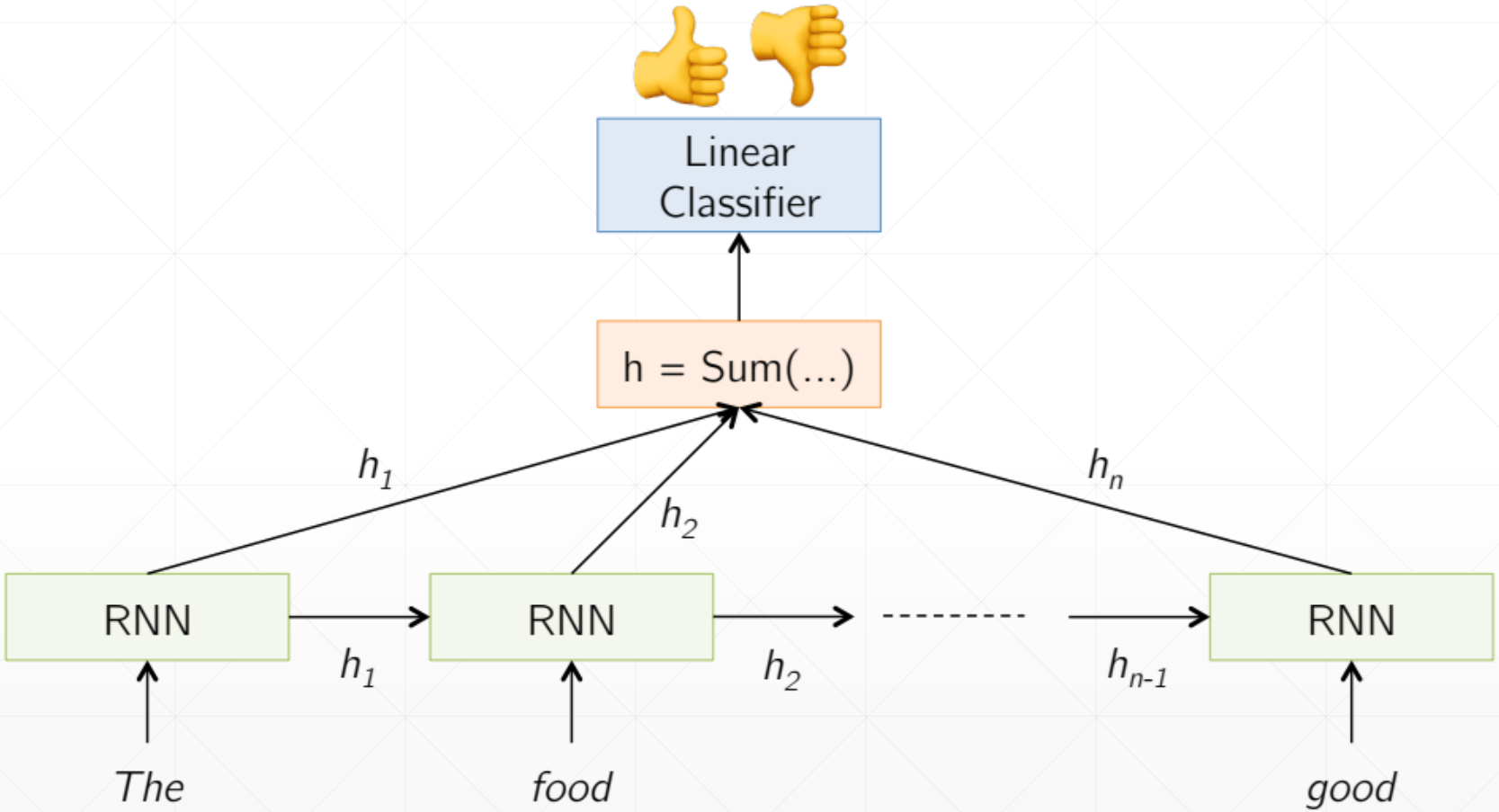
情感分类实战

主讲人：龙良曲

Sentiment Classification



Sentiment Classification



Google CoLab

- Continuous 12 hours
- free K80 for GPU
- no need to cross GFW

Load Dataset

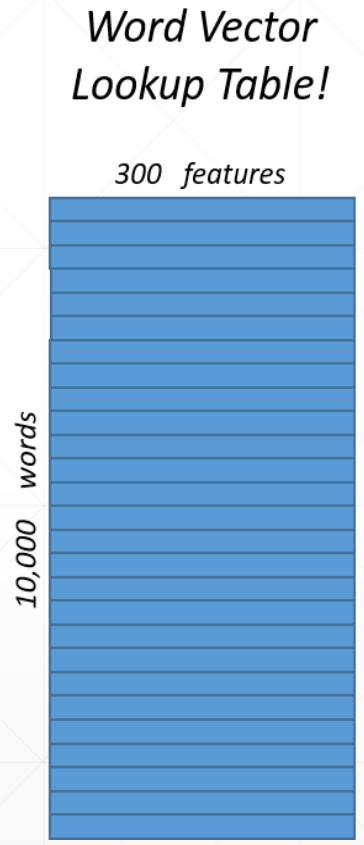
```
1 TEXT = data.Field(tokenize='spacy')
2 LABEL = data.LabelField(dtype=torch.float)
3 train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
4
5 print('len of train data:', len(train_data))
6 print('len of test data:', len(test_data))
7 len of train data: 25000
8 len of test data: 25000
9
10 print(train_data.examples[15].text)
11 print(train_data.examples[15].label)
12 ['I', 'loved', 'this', 'film', '.', 'I', 'thought', 'it', 'would', 'be', 'easy', 'to', 'watch',
   ',', 'and', 'easy', 'to', 'forget', '.', 'I', 'ran', 'out', 'after', 'watching', 'this', 'to',
   'buy', 'the', 'DVD', ',', 'obv', 'not', 'easily', 'forgotten!<br', '/><br', '/>The', 'script',
   'is', 'brilliant', ',', 'and', 'the', 'casting', 'could', "n't", 'be', 'more', 'perfect', '.',
   'Each', 'character', 'has', 'their', 'moment', ',', 'and', 'I', 'laughed', 'hard', 'throughout',
   'this', 'film', ',', 'comedic', 'timing', 'was', 'spot', '-', 'on.<br', '/><br', '/', '>']
13 pos
```

Network

```python
class RNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(RNN, self).__init__()
        # [0-10001] => [100]
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        # [100] => [256]
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers=2,
                           bidirectional=True, dropout=0.5)
        # [256*2] => [1]
        self.fc = nn.Linear(hidden_dim*2, 1)
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        # [seq, b, 1] => [seq, b, 100]
        embedding = self.dropout(self.embedding(x))
        # output: [seq, b, hid_dim*2]
        # hidden/h: [num_layers*2, b, hid_dim]
        # cell/c: [num_layers*2, b, hid_dim]
        output, (hidden, cell) = self.rnn(embedding)
        # [num_layers*2, b, hid_dim] => 2 of [b, hid_dim] => [b, hid_dim*2]
        hidden = torch.cat([hidden[-2], hidden[-1]], dim=1)
        # [b, hid_dim*2] => [b, 1]
        hidden = self.dropout(hidden)
        out = self.fc(hidden)
        return out
```

Load word embedding

```
1 rnn = RNN(len(TEXT.vocab), 100, 256)
2
3 pretrained_embedding = TEXT.vocab.vectors
4 print('pretrained_embedding:', pretrained_embedding.shape)
5 rnn.embedding.weight.data.copy_(pretrained_embedding)
6 print('embedding layer inited.')
```



Train

```python
def train(rnn, iterator, optimizer, criteon):
    avg_acc = []
    rnn.train()

    for i, batch in enumerate(iterator):
        # [seq, b] => [b, 1] => [b]
        pred = rnn(batch.text).squeeze(1)
        loss = criteon(pred, batch.label)
        acc = binary_acc(pred, batch.label).item()
        avg_acc.append(acc)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Test

```python
def binary_acc(preds, y):
    preds = torch.round(torch.sigmoid(preds))
    correct = torch.eq(preds, y).float()
    acc = correct.sum() / len(correct)
    return acc

def eval(rnn, iterator, criteon):
    avg_acc = []
    rnn.eval()
    with torch.no_grad():
        for batch in iterator:
            # [b, 1] => [b]
            pred = rnn(batch.text).squeeze(1)
            loss = criteon(pred, batch.label)
            acc = binary_acc(pred, batch.label).item()
            avg_acc.append(acc)
    avg_acc = np.array(avg_acc).mean()
    print('>>test:', avg_acc)
```

下一课时

GAN

Thank You.