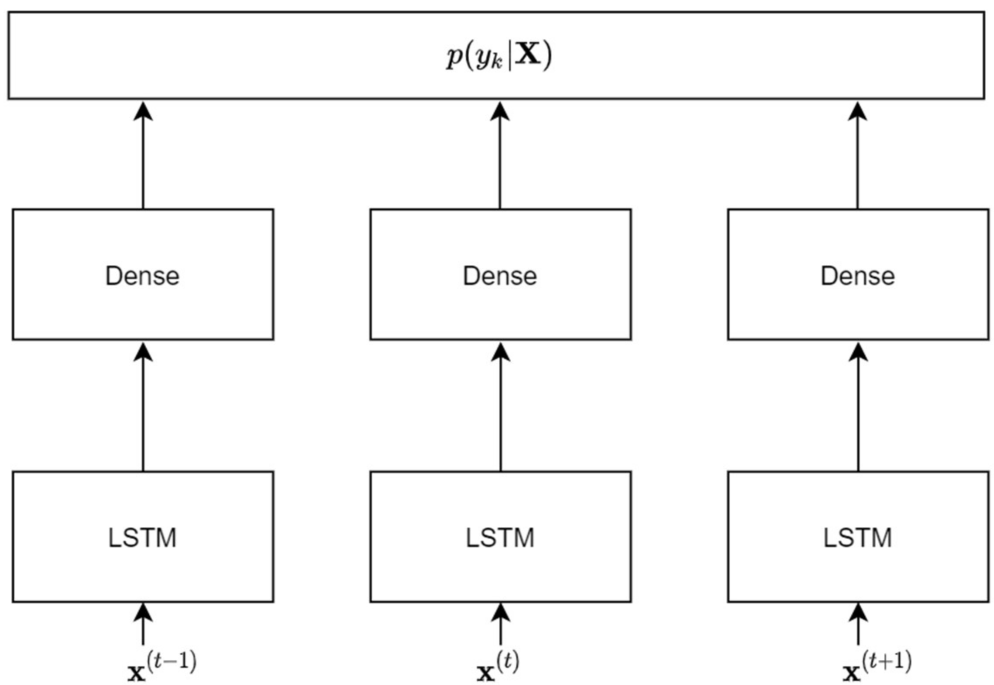
Week 6 – CRF I

EGCO467 Natural Language and Speech Processing

Sequence tagging problem

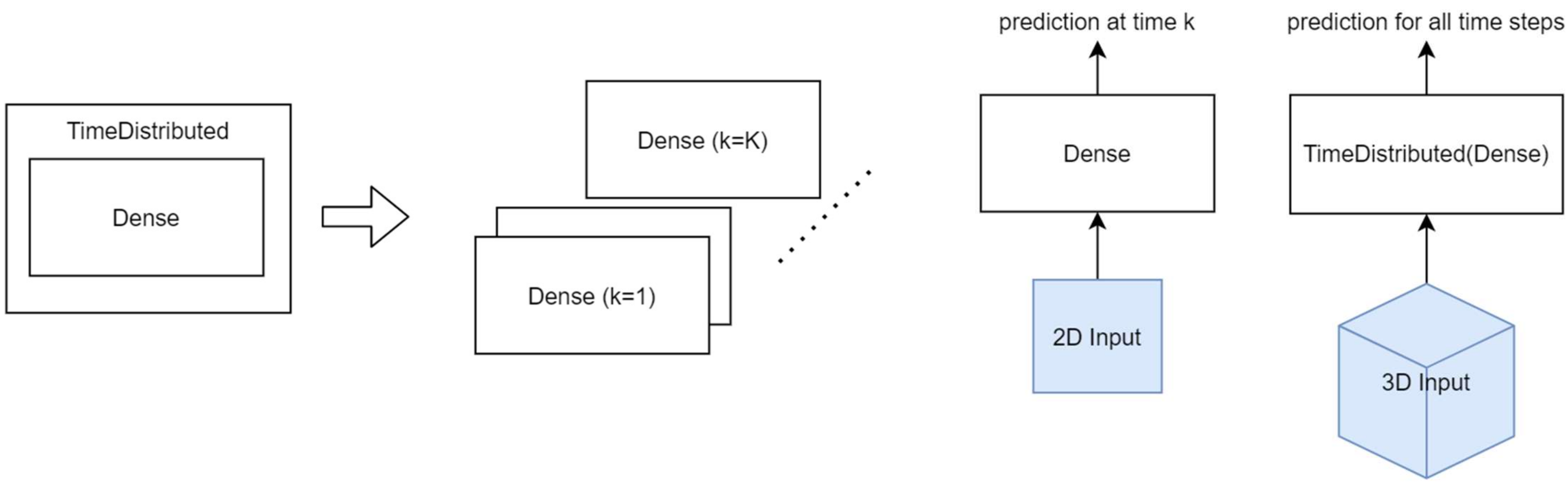
- Label each token, not the entire sequence
- Label whole sequence is classification
- Label each token is sequence tagging

One way to tag tokens



TimeDistributed

TimeDistributed(Dense(n_tags, activation="softmax"))

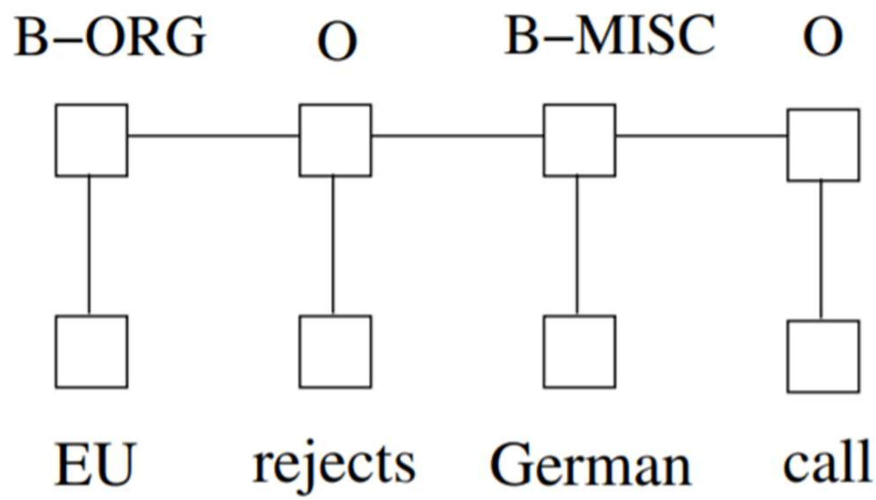


Improvement

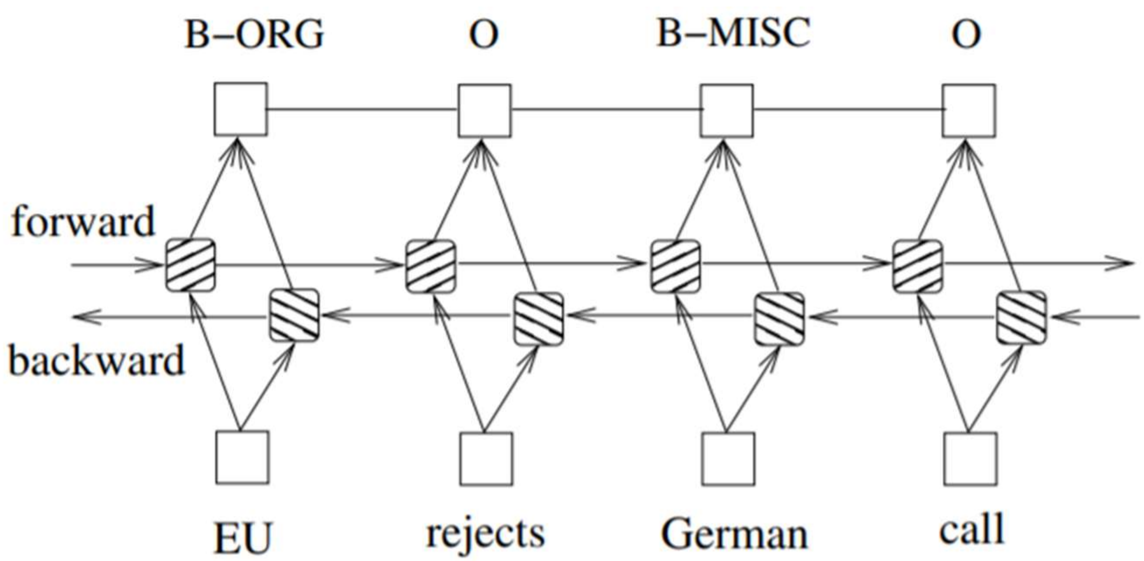
- Should be able to enforce of grammar rule. E.g. adj should be followed by N, ordinal should be followed by N, etc.
- y_k should influence y_{k+1}
- LSTM alone cannot do this

CRF - Conditional Random Field

- "weight" for each type of token c, and for each time k
- "transition weight" for changing from token type j to token type i, when time goes from k to k+1

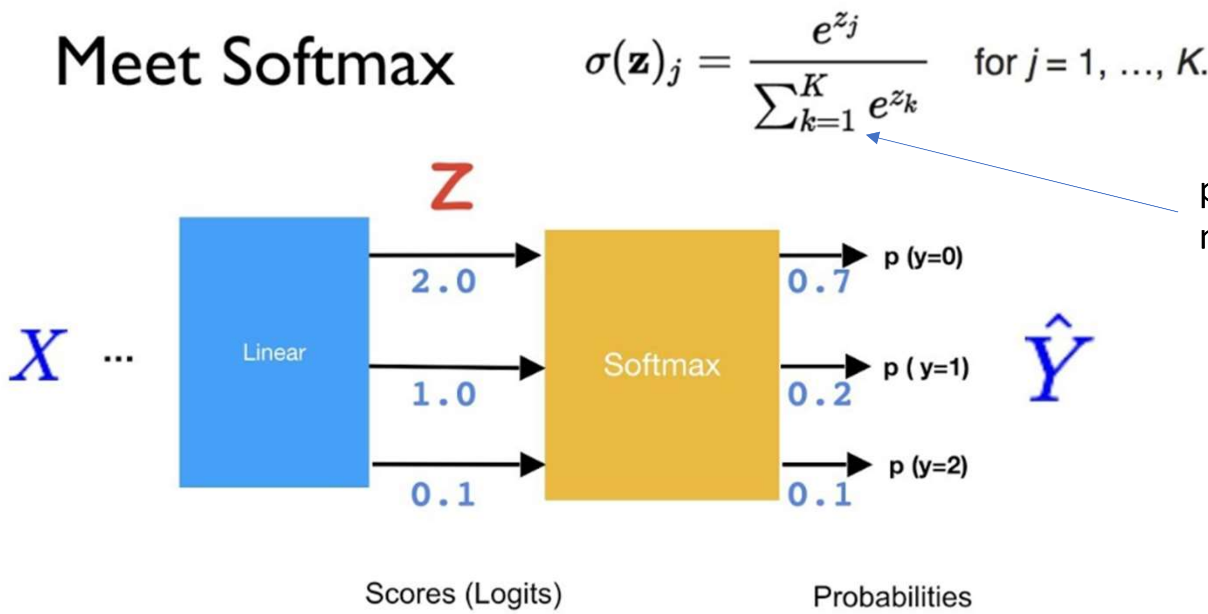


LSTM-CRF



Huang, Zhiheng, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF models for sequence tagging." *arXiv preprint arXiv:1508.01991* (2015).

Review Softmax

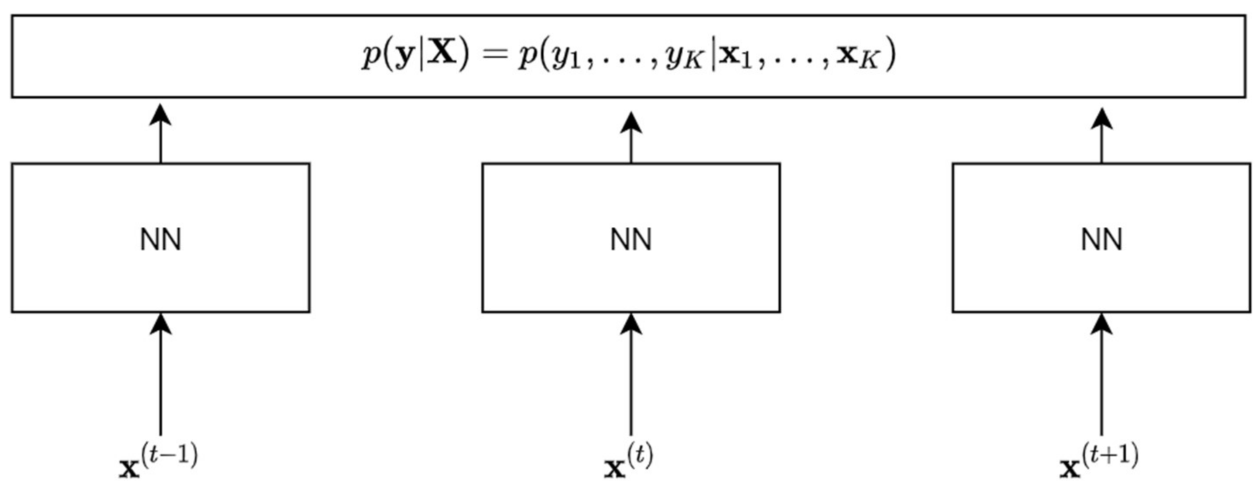


partition function
make scale become 0 to 1

Joint probability

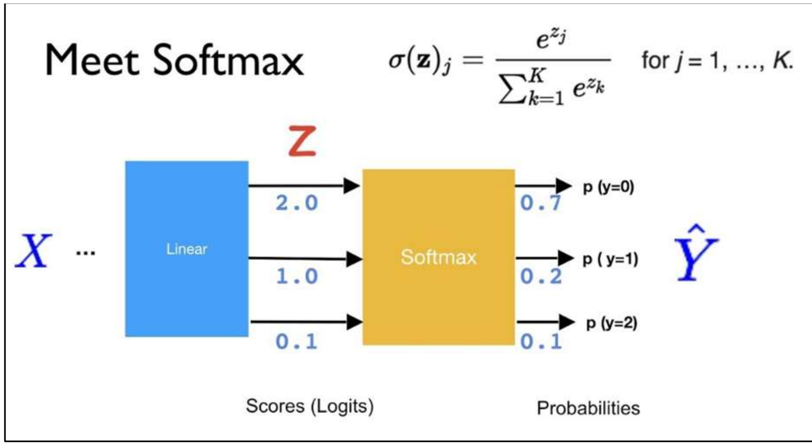
training example:

$$(\mathbf{X}, \mathbf{y})$$



Linear Chain CRF

Linear chain CRF



- Regular classification (tags are independent)

$$p(\mathbf{y}|\mathbf{X}) = \prod_k \exp(\mathrm{NN}(\mathbf{x}_k)_{y_k})/Z(\mathbf{x}_k)$$

product of exponentials = exponential of sum

$$= \exp\left(\sum_k \mathrm{NN}(\mathbf{x}_k)_{y_k}\right) / \left(\prod_k Z(\mathbf{x}_k)\right)$$

- Linear chain: assume tag k influence tag k+1

$$p(\mathbf{y}|\mathbf{X}) = \exp\left(\sum_{k=1}^{K} \mathrm{NN}(\mathbf{x}_k)_{y_k} + \sum_{k=1}^{K-1} V_{y_k, y_{k+1}}\right) / Z(\mathbf{X})$$

V = transition matrix

Transition Matrix

- V is C by C matrix, where C = number of tag types
- E.g for C=3

$$
\begin{array}{cc}
 & \begin{array}{ccc} \text{tar=t1} & \text{tar=t2} & \text{tar=t3} \end{array} \\
V = & \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2.2 \\ 1 & 2.2 & 0.5 \end{bmatrix} \begin{array}{l} \text{src=t1} \\ \text{src=t2} \\ \text{src=t3} \end{array}
\end{array}
$$

- E.g. Assume $\{y_k=t_1, y_{k+1}=t_2\}$, then $V_{yk,yk+1} = V_{t1,t2} = 2$

Notation

- Let $a_u(y_k) = \mathrm{NN}(\mathbf{x}_k)_{y_k}$ and $a_p(y_k, y_{k+1}) = V_{y_k, y_{k+1}}$
- Then

$$p(\mathbf{y}|\mathbf{X}) = \exp\left(\sum_{k=1}^{K} a_u(y_k) + \sum_{k=1}^{K-1} a_p(y_k, y_{k+1})\right) / Z(\mathbf{X})$$

- u for unary (one y) and p for pair (two y's)

Partition function

sum over all possible sequences

$$p(\mathbf{y}|\mathbf{X}) = \exp\left(\sum_{k=1}^{K} a_u(y_k) + \sum_{k=1}^{K-1} a_p(y_k, y_{k+1})\right) / Z(\mathbf{X})$$

$$Z(\mathbf{X}) = \sum_{y'_1} \sum_{y'_2} \cdots \sum_{y'_K} \exp\left(a_u(y'_k) + a_p(y'_k, y'_{k+1})\right)$$

C tags, K length -> O(C^K) complexity

if there are 10 possible tags, then number of all possible sequences is 10^K

Forward & Backward Algorithm

Forward algorithm

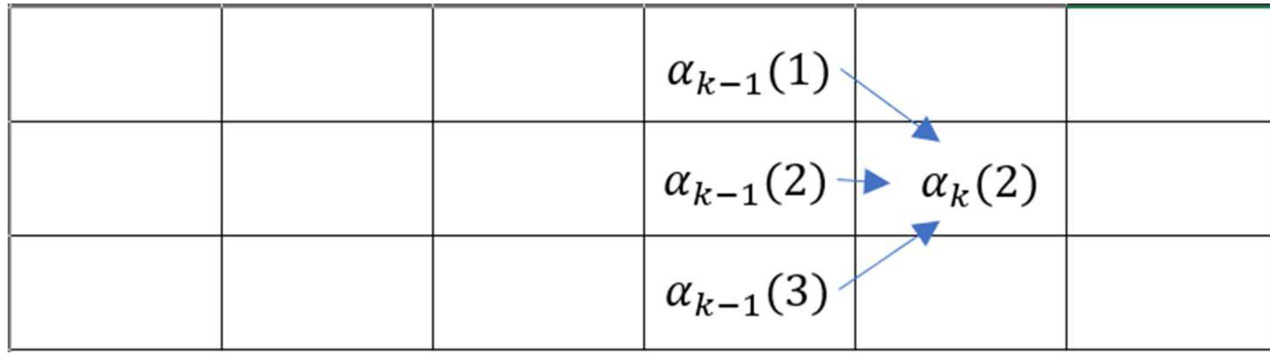
$$Z(\mathbf{X}) = \sum_{y'_1} \sum_{y'_2} \cdots \sum_{y'_K} \exp\left(a_u(y'_k) + a_p(y'_k, y'_{k+1})\right)$$

$$\begin{aligned} Z(\mathbf{X}) = {} & \exp(a_u(y'_k)) \\ & \Bigg(\sum_{y'_{K-1}} \exp(a_u(y'_{K-1}) + a_p(y'_{K-1}, y'_K)) \\ & \cdots \\ & \Bigg(\sum_{y'_2} \exp(a_u(y'_2) + a_p(y'_2, y'_3)) \\ & \Bigg(\sum_{y'_1} \exp(a_u(y'_1) + a_p(y'_1, y'_2)) \Bigg) \cdots \Bigg) \end{aligned}$$

```
for (int i = 0; i < N; i++){
    for (int j = 0; j < N; j++){
        for (int k = 0; k < N; k++){

        }
        // do something with result of k loop
    }
    // do something with result of j loop
}
```

Forward algorithm



- Computing $p(\boldsymbol{y}|\boldsymbol{X})$
- Initialize, for all values of y_2': $\alpha_1(y_2') \leftarrow \sum_{y_1'} \exp(a_u(y_1') + a_p(y_1', y_2'))$
- for k=2 to K-1, and for all values of y_{k+1}'

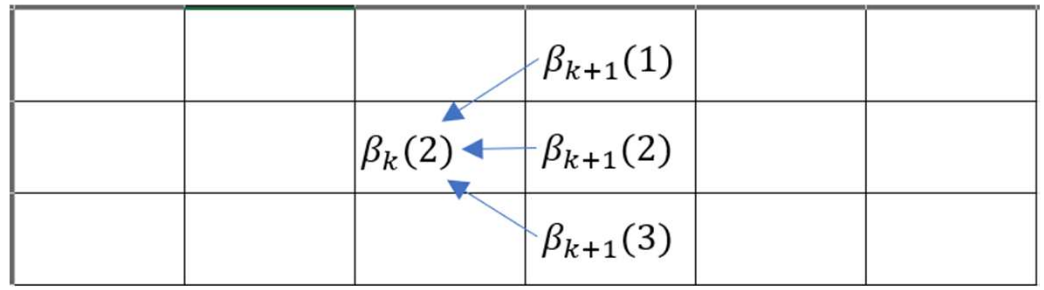
$$\alpha_k(y_{k+1}') \leftarrow \sum_{y_k'} \exp(a_u(y_k') + a_p(y_k', y_{k+1}'))\alpha_{k-1}(y_k')$$

- end of recursion $Z(\mathbf{X}) \leftarrow \sum_{y_K'} \exp(a_u(y_k'))\alpha_{K-1}(y_K')$
- Complexity $O(KC^2)$

Backward algorithm

$$
\begin{aligned}
Z(\mathbf{X}) = \exp(a_u(y'_1)) \\
&\left(\sum_{y'_2} \exp(a_u(y'_1) + a_p(y'_2, y'_3))\right. \\
&\dots \\
&\left(\sum_{y'_{K-2}} \exp(a_u(y'_{K-2}) + a_p(y'_{K-2}, y'_{K-1}))\right. \\
&\left.\left(\sum_{y'_{K-1}} \exp(a_u(y'_{K-1}) + a_p(y'_{K-1}, y'_K))\right) \cdots \right)
\end{aligned}
$$

Backward Algorithm



- Computing $p(\boldsymbol{y}|\boldsymbol{X})$
- Initialize, for all values of y'_{K-1}: $\beta_K(y'_{K-1}) \leftarrow \sum_{y'_K} \exp(a_u(y'_K) + a_p(y'_{K-1}, y'_K))$
- for k = K-1 to 2 and for all values of y'_{k-1}

$$\beta_k(y'_{k-1}) \leftarrow \sum_{y'_k} \exp(a_u(y'_k) + a_p(y'_{k-1}, y'_k))\beta_{k+1}(y'_k)$$

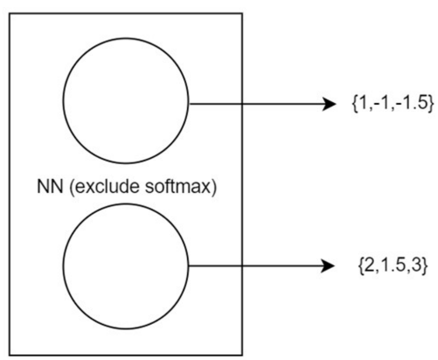
- end of recursion $Z(\mathbf{X}) \leftarrow \sum_{y'_1} \exp(a_u(y'_1))\beta_2(y'_1))$
- Complexity O(KC^2)

Example forward algorithm

	k=1	k=2	k=3
c=1	1	-1	-1.5
c=2	2	1.5	3

V	tar=1	tar=2
src=1	4	5
src=2	6	7

	alpha 1	alpha 2	alpha 3
c=1	A1	B1	C1
c=2	A2	B2	C2



	k=1	k=2	k=3
c=1	1	-1	-1.5
c=2	2	1.5	3

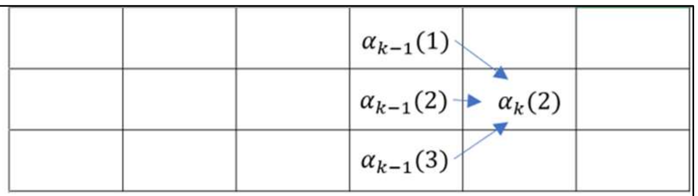
V	tar=1	tar=2
src=1	4	5
src=2	6	7

	alpha 1	alpha 2
c=1	A1	B1
c=2	A2	B2

$$\alpha_1(1) = \sum_{y'_1} \exp(a_u(y'_1) + a_p(y'_1, 1))$$
$$= \exp(a_u(c_1) + V_{c_1,1}) + \exp(a_u(c_2) + V_{c_2,1})$$
$$= \exp(1+4) + \exp(2+6)$$

$$\alpha_1(2) = \sum_{y'_1} \exp(a_u(y'_1) + a_p(y'_1, 2))$$
$$= \exp(a_u(c_1) + V_{c_1,2}) + \exp(a_u(c_2) + V_{c_2,2})$$
$$= \exp(1+5) + \exp(2+7)$$

Forward algorithm



- Computing $p(\boldsymbol{y}|\boldsymbol{X})$
- Initialize, for all values of y'_2: $\alpha_1(y'_2) \leftarrow \sum_{y'_1} \exp(a_u(y'_1) + a_p(y'_1, y'_2))$
- for k=2 to K-1, and for all values of y'_{k+1}

$$\alpha_k(y'_{k+1}) \leftarrow \sum_{y'_k} \exp(a_u(y'_k) + a_p(y'_k, y'_{k+1}))\alpha_{k-1}(y'_k)$$

- end of recursion $Z(\mathbf{X}) \leftarrow \sum_{y'_K} \exp(a_u(y'_k))\alpha_{K-1}(y'_K)$
- Complexity O(KC^2)

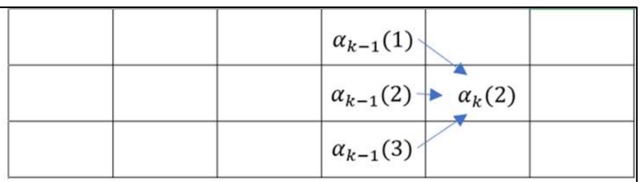
	k=1	k=2	k=3
c=1	1	-1	-1.5
c=2	2	1.5	3

V	tar=1	tar=2
src=1	4	5
src=2	6	7

	alpha 1	alpha 2
c=1	A1	B1
c=2	A2	B2

$$\alpha_2(1) = \sum_{y_2'} \exp(a_u(y_2') + a_p(y_2', 1))\alpha_1(y_2')$$
$$= \exp(a_u(c_1) + V_{c_1,1})\alpha_1(1) + \exp(a_u(c_2) + V_{c_2,1})\alpha_1(2)$$
$$= \exp(-1 + 4)\alpha_1(1) + \exp(1.5 + 6)\alpha_1(2)$$

Forward algorithm



- Computing $p(\boldsymbol{y}|\boldsymbol{X})$
- Initialize, for all values of y_2': $\alpha_1(y_2') \leftarrow \sum_{y_1'} \exp(a_u(y_1') + a_p(y_1', y_2'))$
- for k=2 to K-1, and for all values of y_{k+1}'

$$\alpha_k(y_{k+1}') \leftarrow \sum_{y_k'} \exp(a_u(y_k') + a_p(y_k', y_{k+1}'))\alpha_{k-1}(y_k')$$

- end of recursion $Z(\mathbf{X}) \leftarrow \sum_{y_K'} \exp(a_u(y_k'))\alpha_{K-1}(y_K')$
- Complexity O(KC^2)

$$\alpha_2(2) = \sum_{y_2'} \exp(a_u(y_2') + a_p(y_2', 2))\alpha_1(y_2')$$
$$= \exp(a_u(c_1) + V_{c_1,2})\alpha_1(1) + \exp(a_u(c_2) + V_{c_2,2})\alpha_1(2)$$
$$= \exp(-1 + 5)\alpha_1(1) + \exp(1.5 + 7)\alpha_1(2)$$

number of columns = K-1
number of rows = C

Marginal Probability

- $p(y_k|\mathbf{X})$
- sum over all possible sequence, excluding yk
- Example
  - tags = {N,V,A}
  - k=2 and K=3
  - possible sequences, excluding k=2=V: {N,**V**,N},{N,**V**,V},{V,**V**,A},{V,**V**,N},...,{A,**V**,A}
  - probability $p(y_{k=2}=V|X)$ can be calculated from alphas and betas

joint probability

$$p(\mathbf{y}|\mathbf{X}) = p(y_1, \ldots, y_K|\mathbf{x}_1, \ldots, \mathbf{x}_K)$$

all possible sequences, except the kth position

Marginal Probability

- Compute $p(y_k|\boldsymbol{X})$
- The α or β can be calculated by

$$p(\mathbf{y}|\mathbf{X}) = \exp\left(\sum_{k=1}^{K} a_u(y_k) + \sum_{k=1}^{K-1} a_p(y_k, y_{k+1})\right) / Z(\mathbf{X})$$

take log to cancel exp in alpha,beta formula

$$p(y_k|\mathbf{X}) = \frac{\exp(a_u(y_k) + \log \alpha_{k-1}(y_k) + \log \beta_{k+1}(y_k))}{\sum_{y'_k} \exp(a_u(y_k) + \log \alpha_{k-1}(y_k) + \log \beta_{k+1}(y_k))}$$

classification (tag sequence) option 1

- At each position k, pick y_k with the highest maginal probability $p(y_k|\mathbf{X})$
- Optimal if the CRF is the true distribution (not true)
- Given **X, V, NN**
  - calculate alpha and beta tables
  - use formula for marginal probability

Numerical Examples

	k=1	k=2	k=3
c=1	1	-1	-1.5
c=2	2	1.5	3

V	tar=1	tar=2
src=1	4	5
src=2	6	7

Example

	alpha 1	alpha 2
c=1	A1	B1
c=2	A2	B2

- K = 3, C = {N,V}
- Output of last layer of NN = [[1,2], [-1,1.5], [-1.5,3]]
- V = [[1, 0.8] ; [0.5, 1.1]]
- Calculate the alpha table

	$k=1$	$k=2$	$k=3$
$c=N$	1	-1	-1.5
$c=V$	2	1.5	3

$$V = \begin{bmatrix} 1 & 0.8 \\ 0.5 & 1.1 \end{bmatrix} \quad \begin{bmatrix} N \to N & N \to V \\ V \to N & V \to V \end{bmatrix}$$

	$k=1$	$k=2$
N	A	C
V	B	D

$$B = \alpha_1(2)$$

$$= \exp(a_u(N) + V_{N \to V}) + \exp(a_u(V) + V_{V \to V})$$

$$= \exp(1 + 0.8) + \exp(2 + 1.1)$$

$$= 28.28 \quad \text{B}$$

$$A = \alpha_1(1) = \sum_{y_1'} \exp\left(a_u(y_1') + a_p(y_1', 1)\right)$$

$$= \exp\left(a_u(N) + V_{N \to N}\right) + \exp\left(a_u(V) + V_{V \to N}\right)$$

$$= \exp(1 + 1) + \exp(2 + 0.5)$$

$$= 19.57 \quad \text{A}$$

$$\alpha_1(y_2') \leftarrow \sum_{y_1'} \exp(a_u(y_1') + a_p(y_1', y_2'))$$

$$C = \alpha_2(1) = \sum_{y_2'} \exp(a_u(y_2') + a_p(y_2', 1))\alpha_1(y_2')$$

$$= \exp(a_u(N) + V_{N \to N})\,\alpha_1(1) + \exp(a_u(V) + V_{V \to N})\,\alpha_1(2)$$

$$= \exp(-1 + 1)\,19.57 + \exp(1.5 + 0.1)\,28.28$$

$$= 19.57 + 208.96 = 228.53 \quad \text{C}$$

$$\boxed{\alpha_k(y'_{k+1}) \leftarrow \sum_{y'_k} \exp(a_u(y'_k) + a_p(y'_k, y'_{k+1}))\alpha_{k-1}(y'_k)}$$

$$D = \alpha_2(2) = \exp(a_u(N) + V_{N \to V})\,\alpha_1(1)$$

$$+ \exp(a_u(V) + V_{V \to V})\,\alpha_1(2)$$

$$= \exp(-1 + 0.8)\,19.57 + \exp(1.5 + 1.1)\,28.28$$

$$= 360.54 \quad \text{D}$$

Example

- Given alpha and beta tables

α's	$\alpha_1(1) = 0.5$	$\alpha_2(1) = 1.1$
	$\alpha_1(2) = 0.7$	$\alpha_2(2) = 0.1$
β's	$\beta_4(1) = 0.2$	$\beta_5(1) = 0.6$
	$\beta_4(2) = 1.0$	$\beta_5(2) = 0.4$

- And observation (pre-activation before CRF) at k=3: {-0.5, -0.1}
- Compute $p(y_3=N|\mathbf{X})$

observation at k=3: {-0.5, -0.1}

$$p(\mathbf{y}|\mathbf{X}) = \exp\left(\sum_{k=1}^{K} a_u(y_k) + \sum_{k=1}^{K-1} a_p(y_k, y_{k+1})\right) / Z(\mathbf{X})$$

α's	$\alpha_1(1) = 0.5$	$\alpha_2(1) = 1.1$
	$\alpha_1(2) = 0.7$	$\alpha_2(2) = 0.1$
β's	$\beta_4(1) = 0.2$	$\beta_5(1) = 0.6$
	$\beta_4(2) = 1.0$	$\beta_5(2) = 0.4$

$$p(y_k|\mathbf{X}) = \frac{\exp(a_u(y_k) + \log \alpha_{k-1}(y_k) + \log \beta_{k+1}(y_k))}{\sum_{y_k'} \exp(a_u(y_k) + \log \alpha_{k-1}(y_k) + \log \beta_{k+1}(y_k))}$$

$$\frac{\exp[a_u(N) + \log \alpha_2(1) + \log \beta_4(1)]}{\exp[a_u(N) + \log \alpha_2(1) + \log \beta_4(1)] + \exp[a_u(V) + \log \alpha_2(2) + \log \beta_4(2)]}$$

$a_u(N) = -0.5$

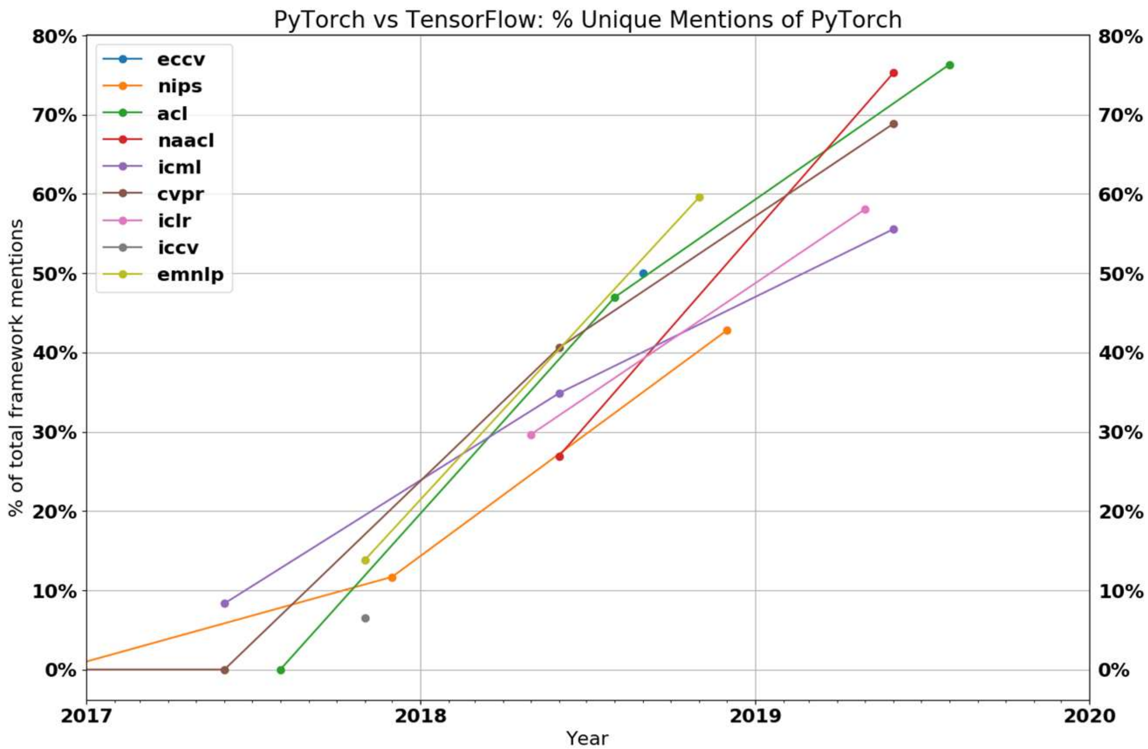
$a_u(v) = -0.1$

Pytorch

Pytorch

- MNIST in Pytorch
- IMDB in Pytorch
- Datasets

Why Pytorch?



PyTorch vs TensorFlow: % Unique Mentions of PyTorch
eccv
nips
acl
naacl
icml
cvpr
iclr
iccv
emnlp
% of total framework mentions
80%
70%
60%
50%
40%
30%
20%
10%
0%
2017
2018
2019
2020
Year

Why Pytorch?

- Same level of abstraction as Tensorflow. So not directly comparable to Keras.
- Code looks almost like regular Python – easier to read/debug.

Pytorch Basics

- model is implemented as a class
- model inherit from *torch.nn.Module*

- inside a model class:
  - create the layers in the constructor *__init__()*
  - chain them together in *forward()*

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784,500)
        self.fc2 = nn.Linear(500,10)
        self.fc3 = nn.Linear

    def forward(self, x):
        x = nn.Flatten()(x)
        x = F.relu(self.fc1(x))
        x = nn.Dropout(0.2)(x)
        x = self.fc2(x) # no softmax
        return x
```

Pytorch Basics

- dataset also a class
- dataset inherit from *torch.utils.data.Dataset, torch.utils.data.DataLoader*
- dataset class implements *__len__()* and *__item__(index)*
- make data loader using *torch.utils.data.DataLoader(dataset)*

```python
kwargs = {'num_workers': 2}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=32, shuffle=True, **kwargs)
```

Pytorch Basic

- Iterate over the Dataloader object with a for loop
- Do the gradient weight update in each pass

```
for epoch in range(5):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device) # move data to GPU
        optimizer.zero_grad() # clear old gradient
        output = model(data) # run model
        loss = F.cross_entropy(output,target)
        loss.backward() # calculate gradient
        optimizer.step() # update weight
        if batch_idx % 500 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch+1, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

FNN_pytorch.ipynb

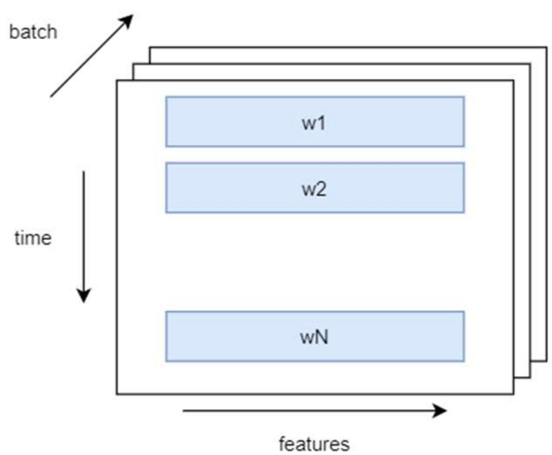
Shape of Tensors in Pytorch (Input)

- Embedding: (batch, seq_length)
- LSTM: (batch, seq_length, embedding_dim)

- Embedding: (seq_length, batch)
- LSTM: (seq_length, batch, embedding_dim)

Shape of Tensors in Pytorch (Output)

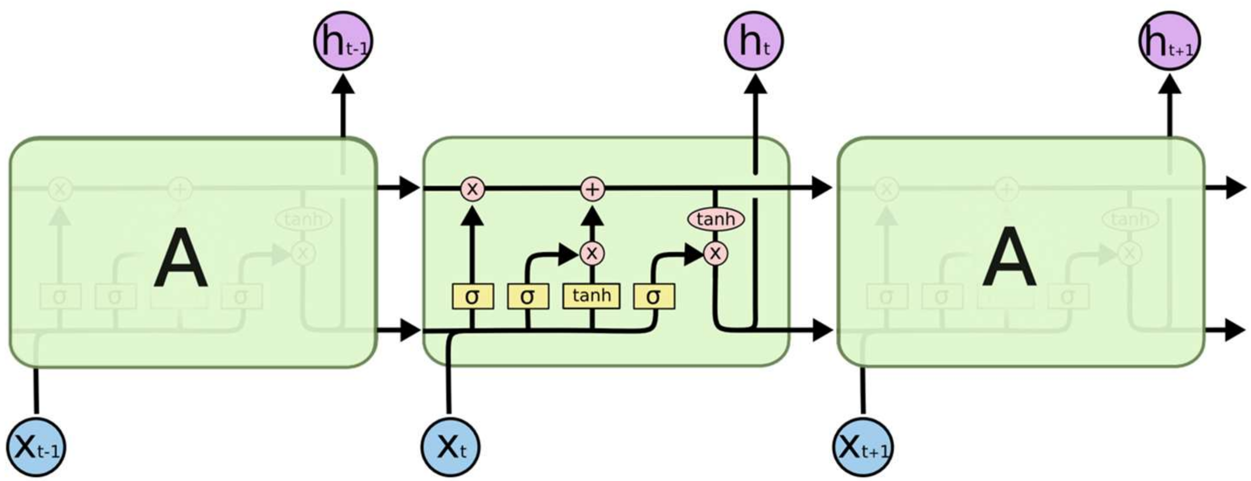
- Embedding: (batch, seq_length, embedding_dim)
- LSTM: (batch, seq_length, hidden_size)



- Embedding: (seq_length, batch embedding_dim)
- LSTM: (1, batch, hidden_size)
- (1, batch, hidden_size) -> squeeze -> (batch, hidden_size)
- (batch, hidden_size) is correct input shape for Linear (Dense) layer

Pytorch LSTM

- Outputs: output, (h_n, c_n)



IMDB_pytorch.ipynb