Connectionist Temporal Classification

Classifying Sequences with Neural Networks

August 25, 2013

Training a Neural Network

The process for training a (already initialized) neural network is

- take a new training example $(\boldsymbol{x}, \boldsymbol{z})$
- Let the network produce an output $\boldsymbol{y} = NN(\boldsymbol{x})$
- Compute the error $E(\boldsymbol{y}, \boldsymbol{z})$
- Compute the derivative of the error wrt. to every weight $\delta w = \frac{\partial E(\boldsymbol{x},\boldsymbol{y})}{\partial w}$
- Change each weight to reduce the error $w \leftarrow w - \eta \cdot \delta w$

Training a Neural Network

The process for training a (already initialized) neural network is

- take a new training example $(\boldsymbol{x}, \boldsymbol{z})$
- Let the network produce an output $\boldsymbol{y} = NN(\boldsymbol{x})$
- **Compute the error** $E(\boldsymbol{y}, \boldsymbol{z})$
  - Or, alternatively, just the output
- Compute the derivative of the error wrt. to every weight $\delta w = \frac{\partial E(\boldsymbol{x},\boldsymbol{y})}{\partial w}$
- Change each weight to reduce the error $w \leftarrow w - \eta \cdot \delta w$

Outline

CTC

Connectionist Temporal Classification

What is CTC

- It provides the interface between the raw network output and final string

$$(y_{tc})_{t=1..T,c=1..C} \leftrightarrow \boldsymbol{w} \in \Sigma^{\leq T}$$

$\rightarrow$ Transform network output to output string

$\leftarrow$ Computes the error function given an (input, target output) pair

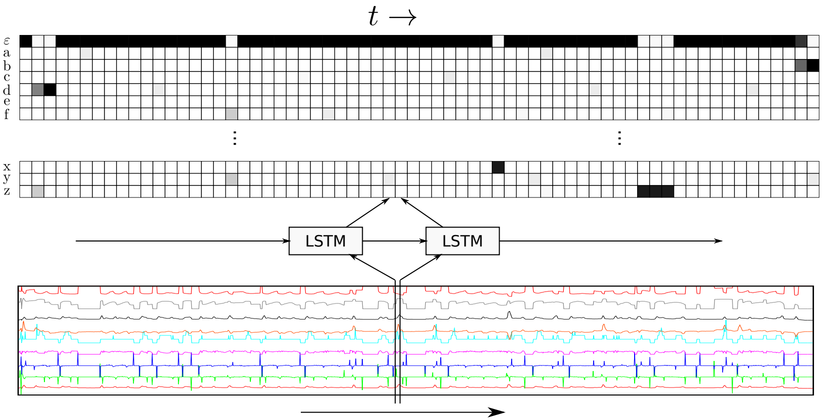
- Comparable to Viterbi algorithm for HMMs

Overview Text Recognition

The network output is a posterior character probability

$$Y = (y_{tc})$$

$y_{tc} = p(\text{output character at time } t \text{ is } c)$



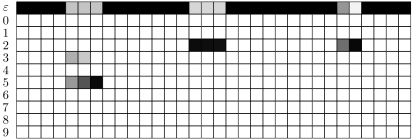
Blank node

Or ε-node

- The network output is usually contains one one for each character
- In addition, an extra node is given, the blank or ε-node
  - Not a requirement, but useful
  - Simplifies to distinguish between succeeding observations of the same class
  - Default output of the network in case no clear character can be detected

Recognition without dictionary

$\xrightarrow{NN}$



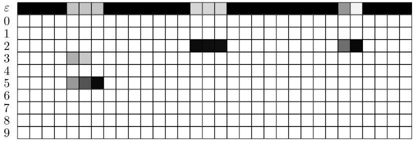
Without a dictionary, the recognition can be done as

- Find path $\boldsymbol{p} = p_1 p_2 \ldots p_T$ of the best characters $p_t = max_c y_{tc}$
- Apply reduction operator $\mathcal{B}$ to p. Operator $\mathcal{B}$:
  1. remove recurrent occurrences of the same class
  2. remove ε outputs

Example

Operator $\mathcal{B}$

$\xrightarrow{NN}$



1. Best path

$$\varepsilon\varepsilon\varepsilon555\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon222\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon\varepsilon22\varepsilon\varepsilon\varepsilon\varepsilon$$

2. No repetitions

$$\varepsilon5\varepsilon2\varepsilon2\varepsilon$$

3. No ε

$$522$$

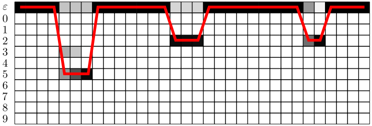
Recognition Score of a given target sequence

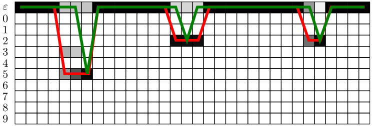
A word w is recognized with score

$$s(w) = \max_{\boldsymbol{p} \in \mathcal{B}^{-1}(w)} s(\boldsymbol{p})$$

Where score $s(\boldsymbol{p})$ of a path $\boldsymbol{p}$ is the product of all output activations

$$s(\boldsymbol{p}) = \prod_t y_{tp_t}$$



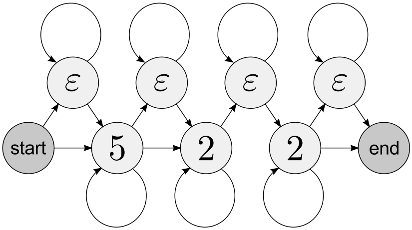


Recognition score of a given target sequence

- How do we find $s(w) = \max_{\boldsymbol{p} \in \mathcal{B}^{-1}(w)} s(\boldsymbol{p})$?
  The number of all path's that can represent a word grows exponentially

Recognition score of a given target sequence

- How do we find $s(w) = \max_{\boldsymbol{p} \in \mathcal{B}^{-1}(w)} s(\boldsymbol{p})$?
  The number of all path's that can represent a word grows exponentially
- Dynamic programming!
- To see the similarity to HMM we can transform a word into a finite state machine, or *transition graph*
- All paths that represent the sequence 522 are given by

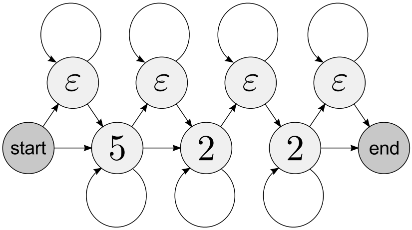


Recognition score of a given target sequence

- How do we find $s(w) = \max_{\boldsymbol{p} \in \mathcal{B}^{-1}(w)} s(\boldsymbol{p})$?
  The number of all path's that can represent a word grows exponentially

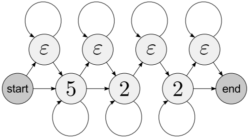
Recognition score of a given target sequence

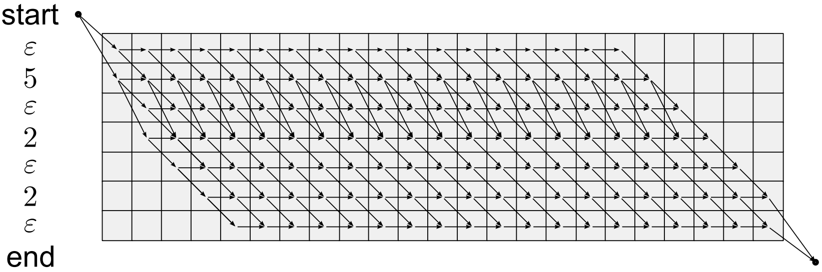
- How do we find $s(w) = \max_{\boldsymbol{p} \in \mathcal{B}^{-1}(w)} s(\boldsymbol{p})$?
  The number of all path's that can represent a word grows exponentially
- Dynamic programming!
- To see the similarity to HMM we can transform a word into a finite state machine, or *transition graph*
- All paths that represent the sequence 522 are given by



Dynamic Programming

not unlike Viterbi





`for each cell` $\vartheta(t,s)$ `with character` $\vartheta(t,s).c$ `and value` $\vartheta(t,s).v$

$$\vartheta(t,s).v \quad \leftarrow \quad y_{t\vartheta(t,s).c} \cdot \max_{\vartheta' \in \text{ precessors}} \vartheta'.v$$

Training

Using the CTC algorithm for training

Training

The process for training a (already initialized) neural network is

- take a new training example $(\boldsymbol{x}, \boldsymbol{z})$
- Let the network produce an output $\boldsymbol{y} = NN(\boldsymbol{x})$
- **Compute the error** $E(\boldsymbol{y}, \boldsymbol{z})$
- Compute the derivative of the error wrt. to every weight $\delta w = \frac{\partial E(\boldsymbol{x}, \boldsymbol{y})}{\partial w}$
- Change each weight to reduce the error $w \leftarrow w - \eta \cdot \delta w$

Overview Training

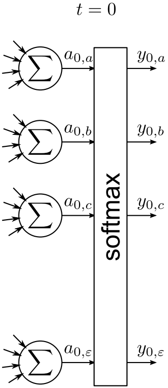
The other task of the CTC algorithm is to estimate the error gradient

$$\frac{\partial}{\partial y_{tc}} E\left(NN(x), z\right)$$

respectively

$$\frac{\partial}{\partial a_{tc}} E\left(NN(x), z\right)$$

where a_{tc} is the input into the output layer



Objective Function

Training a neural network requires a differentiable error function $E(y, z)$, with $\boldsymbol{y} = NN(\boldsymbol{x})$

- We define the error function to the negative log probability of correctly labeling the training set

$$E = -\log\left(\prod_{(\boldsymbol{x},\boldsymbol{z})\in S} p(\boldsymbol{z}|\boldsymbol{x})\right) = -\sum_{(\boldsymbol{x},\boldsymbol{z})\in S} \log p(\boldsymbol{z}|\boldsymbol{x})$$

- In the training step, we consider one training sequence at a time

$$E = -\log p(\boldsymbol{z}|\boldsymbol{x})$$

Training

For the training, we need to differentiate the error function wrt. to the nodes of the output layer

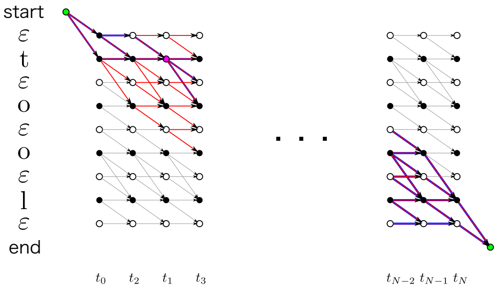
$$\frac{\partial E}{\partial y_{tc}} = -\frac{\partial \log p(z|\mathbf{x})}{\partial y_{tc}} = -\frac{1}{p(z|\mathbf{x})}\frac{\partial p(z|\mathbf{x})}{\partial y_{tc}}$$

Objective Function

Estimating the derivative of the log probability

For any given character, time combination (c, t) we can write the probability $p(z|x)$ as

$$p(z|x) \quad = \quad p(z|x \wedge y_t = c) + p(z|x \wedge y_t \neq c)$$



$$p(z|x) = \sum_{\substack{p \in \mathcal{B}^{-1}(z) \\ p_t = c}} \prod_t y_{tp_t} + \sum_{\substack{p \in \mathcal{B}^{-1}(z) \\ p_t \neq c}} \prod_t y_{tp_t}$$

Objective Function

Estimating the derivative of the log probability

The derivative is therefore

$$\begin{aligned}
\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_{tc}} &= \frac{\partial p(\mathbf{z}|\mathbf{x} \wedge y_t = c)}{\partial y_{tc}} + \frac{\partial p(\mathbf{z}|\mathbf{x} \wedge y_t \neq c)}{\partial y_{tc}} \\
&= \frac{\partial}{\partial y_{tc}} \sum_{\substack{p \in \mathcal{B}^{-1}(\mathbf{z}) \\ p_t = c}} \prod_t y_{tp_t} + 0 \\
&= \frac{p(\mathbf{z}|\mathbf{x} \wedge y_t = c)}{y_{tc}}
\end{aligned}$$

since

$$\frac{\partial}{\partial a_i} \prod_j a_j = \frac{1}{a_i} \prod_j a_j$$

Objective Function

And the sought-after error gradient is

$$\begin{aligned}\frac{\partial E}{\partial y_{tc}} &= -\frac{\partial \log p(\mathbf{z}|\mathbf{x})}{\partial y_{tc}} = -\frac{1}{p(\mathbf{z}|\mathbf{x})}\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_{tc}} \\ &= -\frac{p(\mathbf{z}|\mathbf{x} \wedge y_t = c)}{p(\mathbf{z}|\mathbf{x}) \cdot y_{tc}}\end{aligned}$$

The next step is to compute the derivative wrt. the inputs that go into the output layer a_{tc} for each time t and character c

Derivative assuming a softmax normalization

For the next step assume we have a softmax normalization a output layer

$$y_{tc} = \frac{\exp(a_{tc})}{\sum_{c'} \exp(a_{tc})}$$

The derivative of the error function wrt. a_{tc} is then

$$\begin{aligned} \frac{\partial E}{\partial a_{tc}} &= -\sum_{c'} \frac{\partial E}{\partial y_{tc'}} \frac{\partial y_{tc'}}{a_{tc}} \\ &= -\sum_{c'} -\frac{p(z|\mathbf{x} \wedge y_t = c')}{p(z|\mathbf{x}) \cdot y_{tc'}} \cdot \left(y_{tc'} \delta(c, c') - y_{tc} y_{tc'} \right) \end{aligned}$$

And putting it all together results in

$$\frac{\partial E}{\partial a_{tc}} = y_{tc} - \frac{p(z|\mathbf{x} \wedge y_t = c)}{p(z|\mathbf{x})}$$

Computing the derivative

The values to compute are for each time t and character c

$$\frac{\partial E}{\partial a_{tc}} = y_{tc} - \frac{p(\mathbf{z}|\mathbf{x} \wedge y_t = c)}{p(\mathbf{z}|\mathbf{x})}$$

- The first part y_{tc} is the direct output of the neural network at node c for time t
- The second part $\frac{p(\mathbf{z}|\mathbf{x} \wedge y_t = c)}{p(\mathbf{z}|\mathbf{x})}$ can be computed efficiently with the forward-backward algorithm

Forward-Backward Algorithm

Consider target word sequence $\boldsymbol{z}$ and network output $\boldsymbol{y}$.

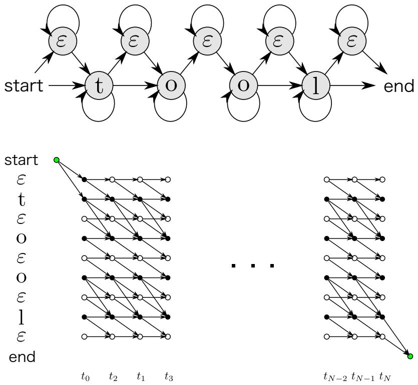
- We can estimate the probability start in $t = 0$ and arrive at y_{tc} via the forward algorithm $\alpha_t(c)$
- Similarly, the backward algorithm returns the probability of starting in y_{tc} and arriving at the end of the text line in the last character $\beta_t(c)$
- Their product is

$$\alpha_t(c)\beta_t(c) = p(\boldsymbol{z}|\boldsymbol{x} \wedge y_t = c)$$

- Both α and β can be computed efficiently using dynamic programming

Forward-Backward Algorithm

Both α and β can be computed efficiently using dynamic programming



`for each cell` $\vartheta(t,s)$ `with character` $\vartheta(t,s).c$ `and value` $\vartheta(t,s).v$

$$\vartheta(t,s).v \;\leftarrow\; y_{t\vartheta(t,s).c} \cdot \sum_{\vartheta' \in \text{ precessors}} \vartheta'.v$$

Algorithmic Overview for the training

Given training example $(\boldsymbol{x}, \boldsymbol{z})$

1. Use the NN forward pass to compute
   $Y = (\boldsymbol{y}_{tc})_{t=1\ldots T, c=1\ldots C} = NN(\boldsymbol{x})$
2. Use the forward algorithm to compute $\alpha_t(c)$ for each (c,t)
3. Use the backward algorithm to compute $\beta_t(c)$ for each (c,t)
4. Compute $\gamma_t(c) = p(\boldsymbol{z}|\boldsymbol{x} \wedge y_t = c) = \alpha_t(c) \cdot \beta_t(c)$
5. Compute $p(\boldsymbol{z}|\boldsymbol{x}) = \sum_c p(\boldsymbol{z}|\boldsymbol{x} \wedge y_t = c)$ for any arbitrary t
6. Compute the error gradient

$$\frac{\partial E}{\partial a_{tc}} = y_{tc} - \frac{p(\boldsymbol{z}|\boldsymbol{x} \wedge y_t = c)}{p(\boldsymbol{z}|\boldsymbol{x})}$$

7. Use the back-propagation algorithm to compute the error gradients for the network weights
8. Update weights

Summary

- CTC is not a specific algorithm but describes the way of doing sequence recognitions with neural networks
- In this part we have seen how the neural network connects to the domain of text recognition
- The CTC algorithm provides a way for single word recognition with and without a dictionary as well as text line recognition with bi-gram language models
- For training, we can use the forward-backward algorithm to compute the error gradient for each node
- All CTC algorithm contain a high similarity to the algorithms used for HMMs

Time for Questions