Convolutional Neural Network, Recurrent Neural Network & Connectionist Temporal Classification

Yu Tian
School of Electronics Engineering & Computer Science
Peking University, 100871, Beijing, China
tianyu_eecs@pku.edu.cn

Outline

➢Neural Network

- Back propagation
- Convolution Neural Network

➢Recurrent Neural Network

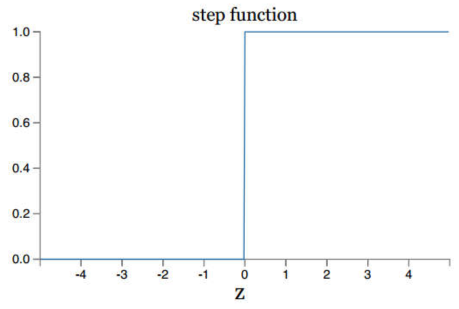
- LSTM

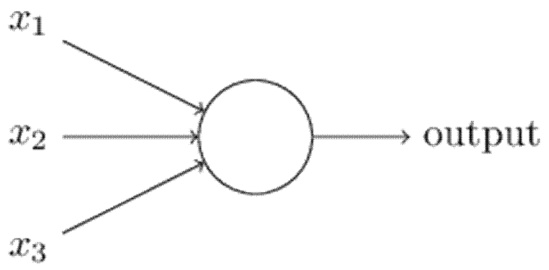
➢Connectionist Temporal Classification

- Theoretical derivation and realization

Neural Network

Perceptrons



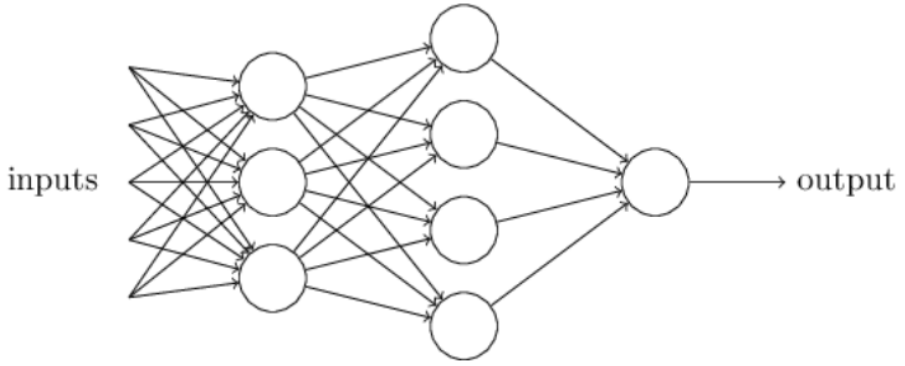


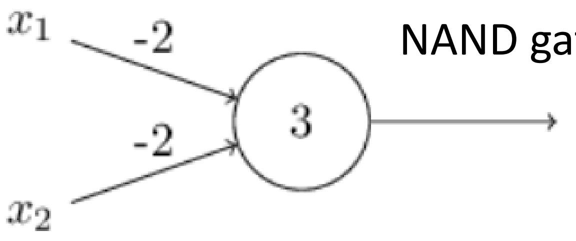
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } \underset{\uparrow}{w} \cdot x + \underset{\uparrow}{b} > 0 \end{cases}$$

weight **bias**

Weights: expressing the importance of the respective inputs to the output

Bias: measuring the degree of laziness of neurons





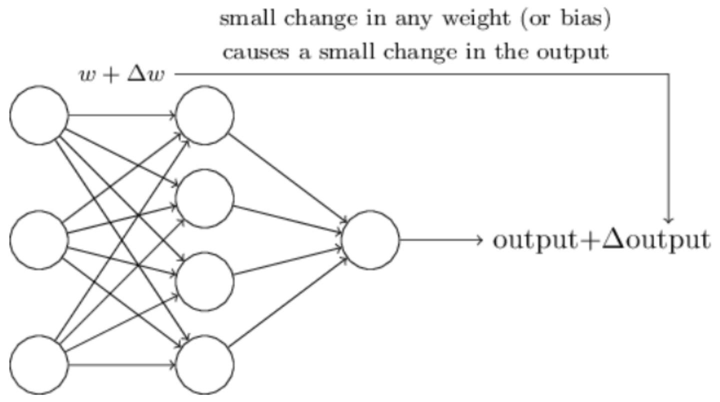
NAND gate

- First layer is making three very simple decisions, by weighing the input evidence.
- Second layer can make a decision at a more complex and more abstract level

- We can use networks of perceptrons to compute any logical function at all.

Neural Network

The key to learning - Sigmoid Neural


small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

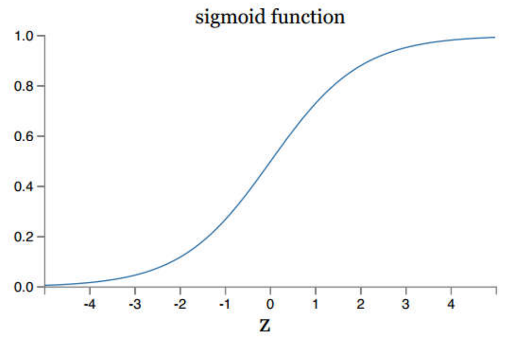
- Small change in weight to cause only a small corresponding change in the output from the network.

Smoothness differentiable

$$\Delta\text{output} \approx \sum_j \frac{\partial\,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial\,\text{output}}{\partial b}\Delta b,$$

- This linearity makes it easy to choose small changes in the weights and biases to achieve any desired small change in the output.

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$



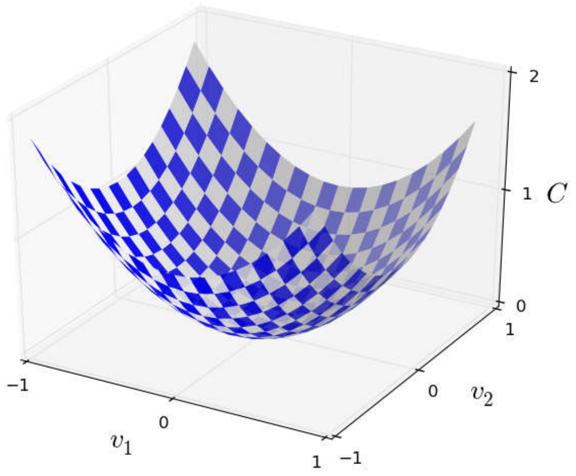
- Output of sigmoid neuron represents the log odds of whether or not a feature is detected.

Neural Network

The key to learning – Loss Function

- ***Requirement***
- Quantify how well we're learning
- Positive
- Is not a smooth function of the weights and biases in the network
- Can be written as a function of the outputs from the neural network
- Can be written as an average over cost functions for individual training examples

Required by BP



$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

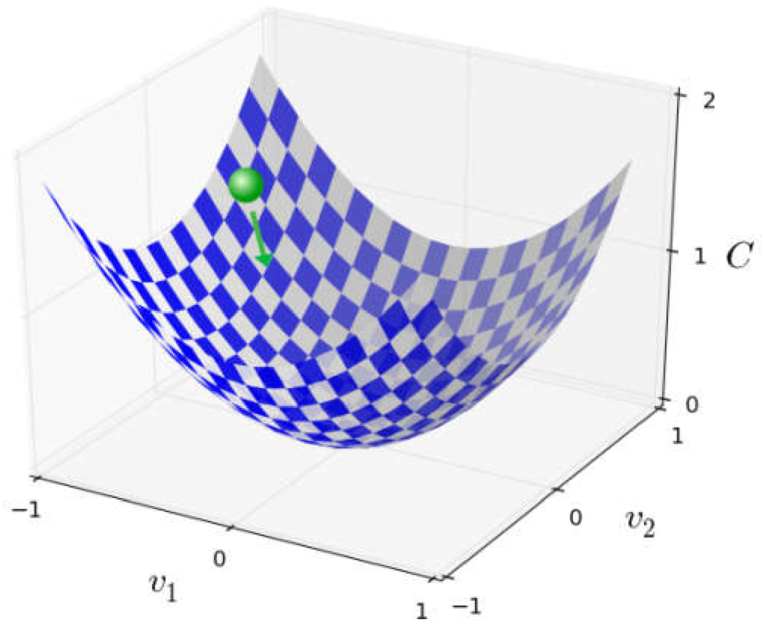
Neural Network

Gradient Descent

$$\Delta C \approx \nabla C \cdot \Delta v.$$

$$\Delta v = -\eta \nabla C,$$

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$



C will always decrease never increase.

$$v \rightarrow v' = v - \eta \nabla C$$

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Neural Network

Back Propagation (BP) algorithm

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w+\epsilon e_j)-C(w)}{\epsilon}$$

Too Complicated

- The back propagation provide us with a way of computing the gradient of the cost function. compute all the partial derivatives using just one forward pass through the network, followed by one backward pass through the network.

- Error Signal

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

- Weighted input

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

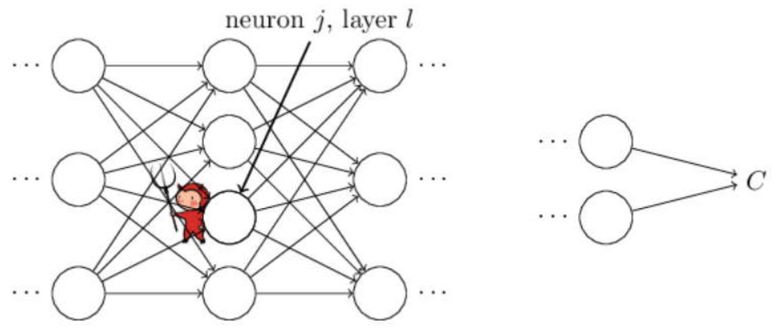
Perspective One

is a measure of the error in the neuron.

最优点偏导数应该为0

Perspective Two

Dynamic programing-**Computation efficient**



Neural Network

Error Signal

- ***Output layer***

$$\delta_j^L = \sum_k \sigma'(z_j^l) \frac{\partial C}{\partial a_k^L}$$

Weighted input

Activation

- ***Hidden layer***

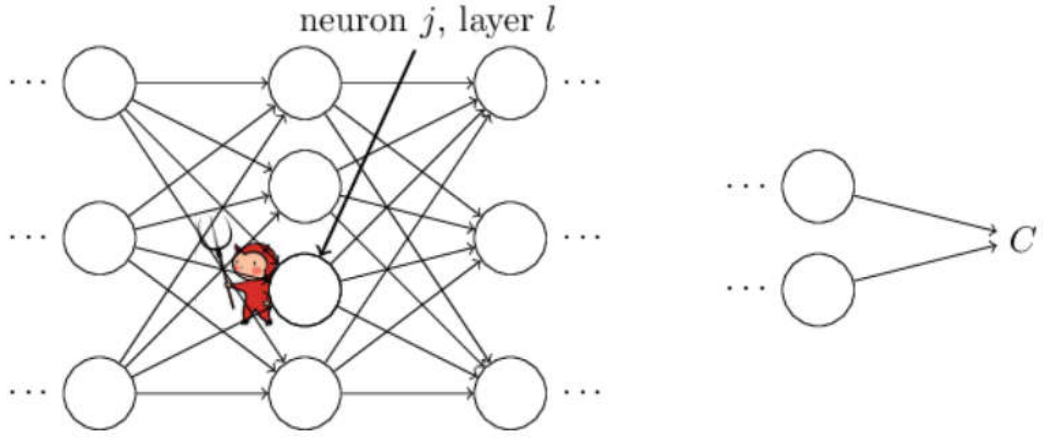
$$\delta_j^l = \sum_k \sigma'(z_j^l)\, w_{kj}^{l+1}\, \delta_k^{l+1}$$

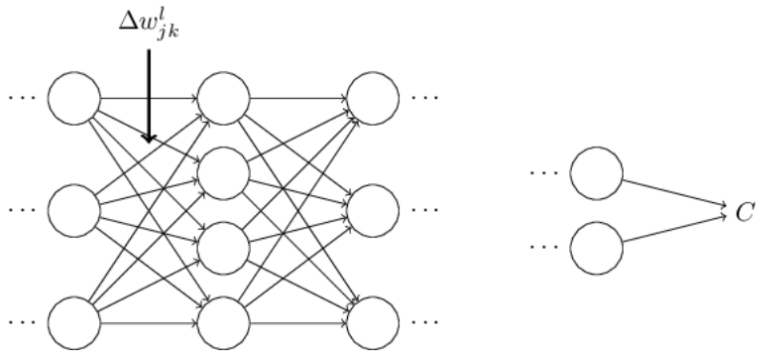
Weighted input

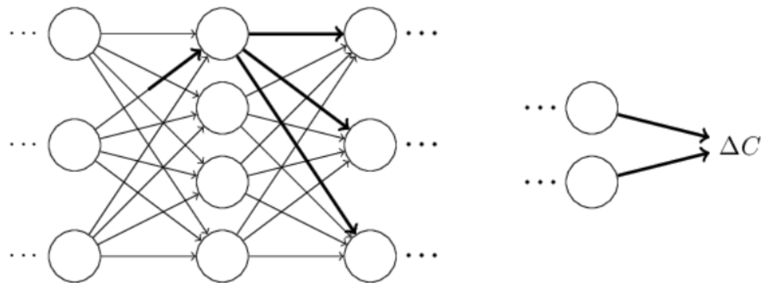
Activation

Error of last layer

- ***What's BP really doing***

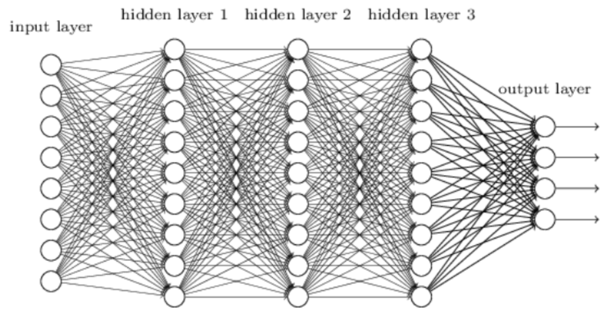






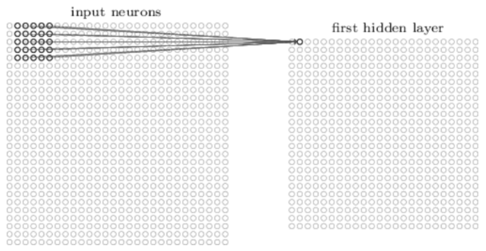
Computing $\frac{\partial C}{\partial w_{jk}^l}$ is to carefully track how a small change in w_{jk}^l propagates to cause a small change in *C*.

Convolutional Neural Network

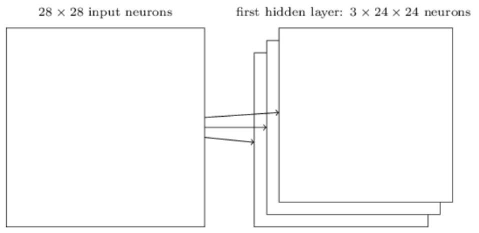


architecture

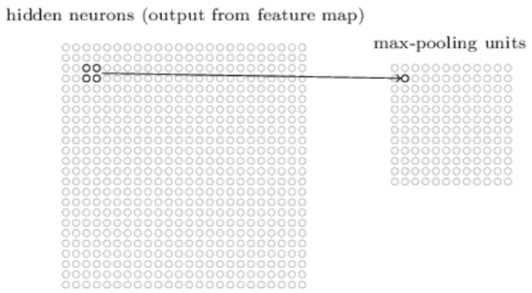
- Local receptive Fields and convolution

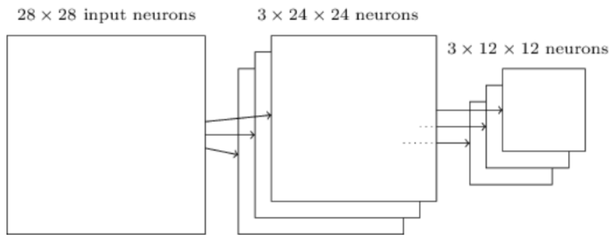


$$\sigma\left(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

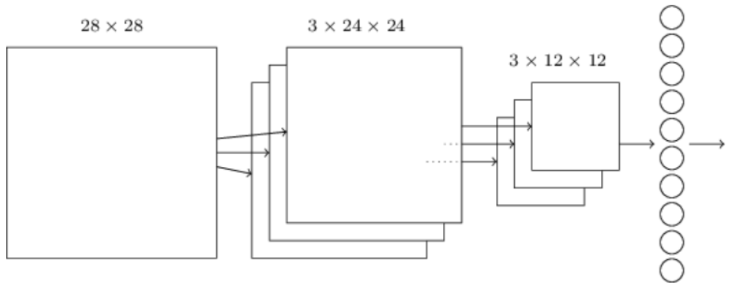


- Pooling layer



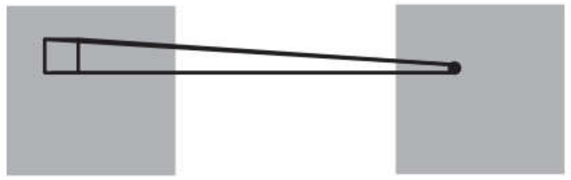


- One Layer CNN

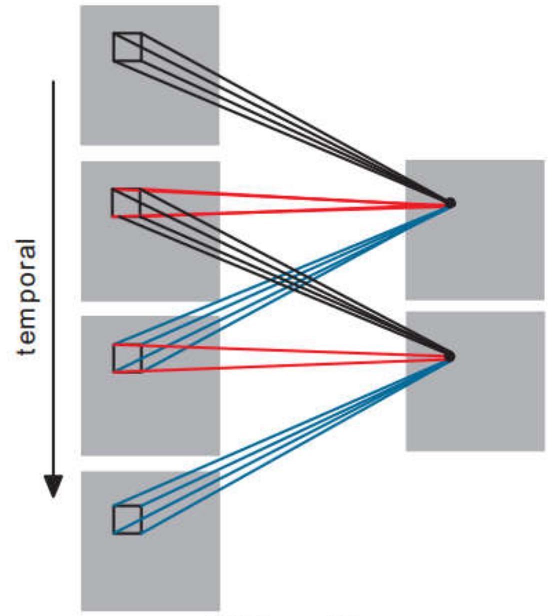


Convolutional Neural Network

2D CNN & 3D CNN & Deep CNN

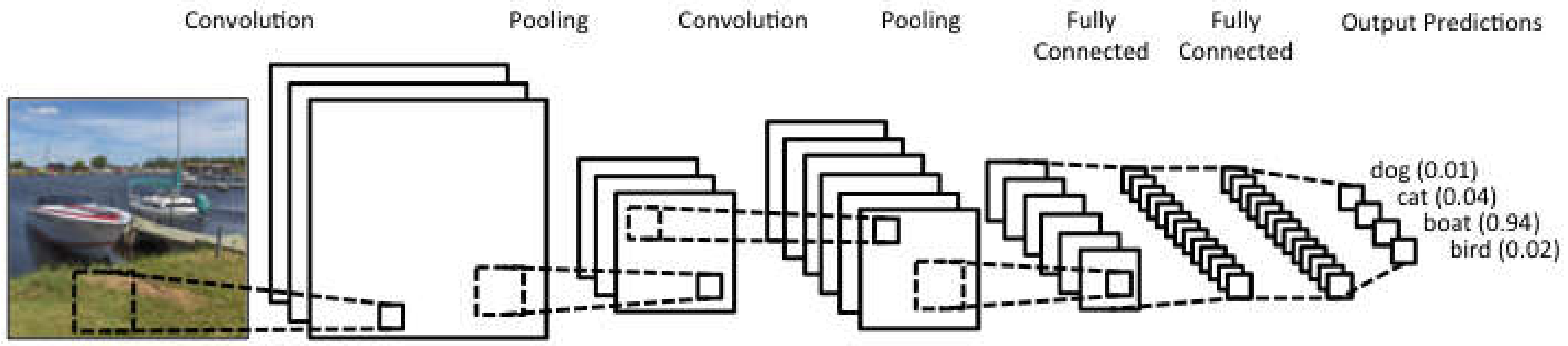

(a) 2D convolution

$$v_{ij}^{xy} = \tanh\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)}\right)$$



(b) 3D convolution

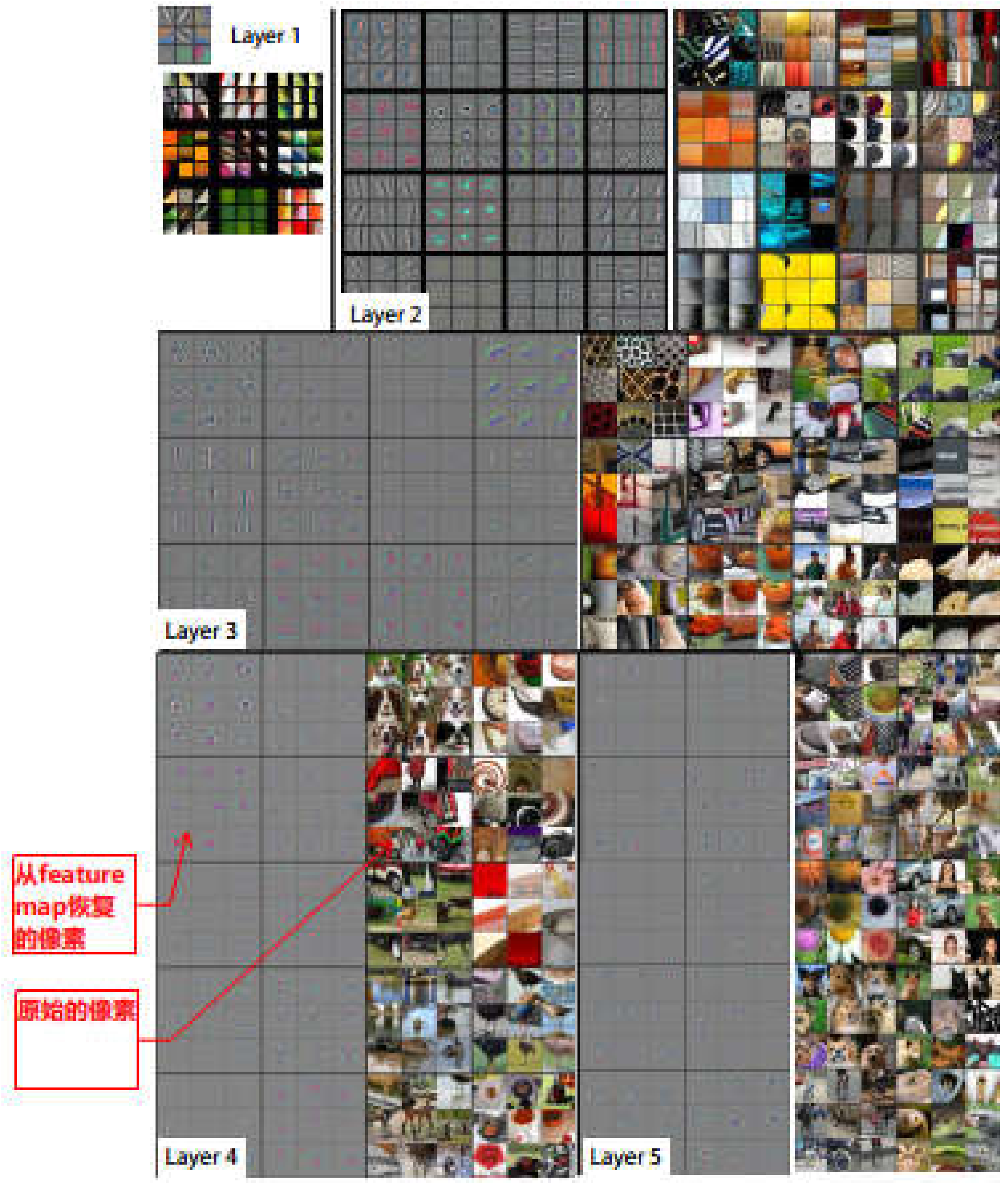
$$v_{ij}^{xyz} = \tanh\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right),$$



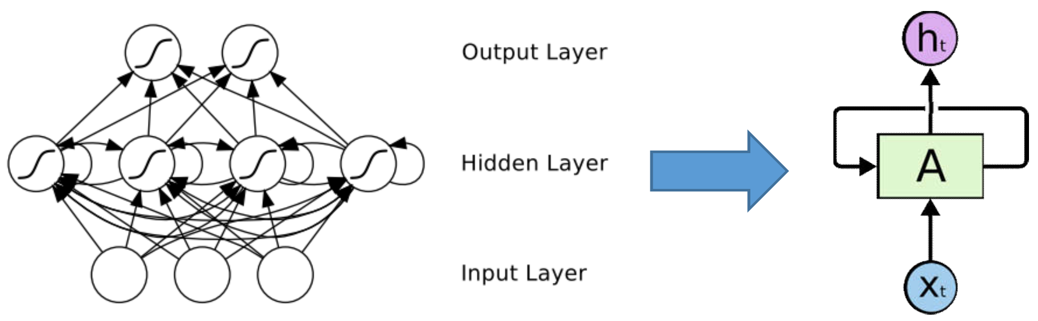
- ➢ The strong grouping within each feature map
- ➢ The projections from each layer show the hierarchical nature of the features in the network.

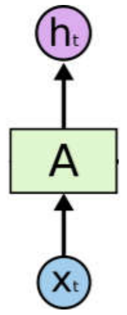
- Layer 2 responds to corners and other edge/color conjunctions.
-

  Layer 3 has more complex invariances, capturing similar textures (e.g. mesh patterns (Row 1, Col 1); text (R2,C4)).
- Layer 4 shows significant variation, and is more class-specific: dog faces (R1,C1); bird’s legs (R4,C2).
- Layer 5 shows entire objects with significant pose variation, e.g. keyboards (R1,C11) and dogs (R4).



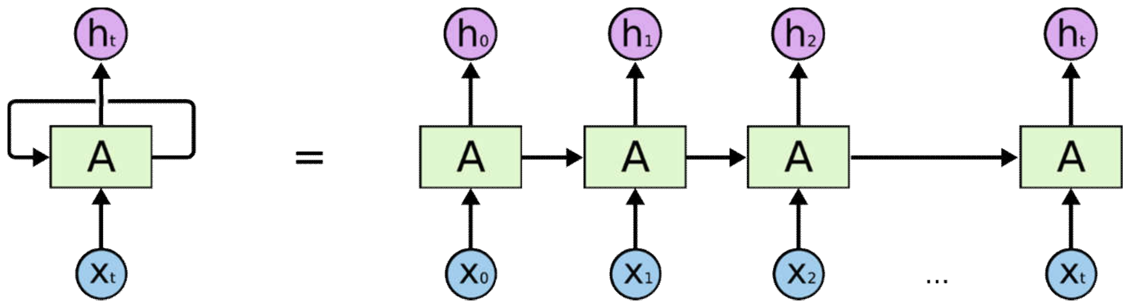
Recurrent Neural Network (RNN) Architecture





RNN *Feedforward Neural Network (FNN)*

- *Unfolding: A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop.*

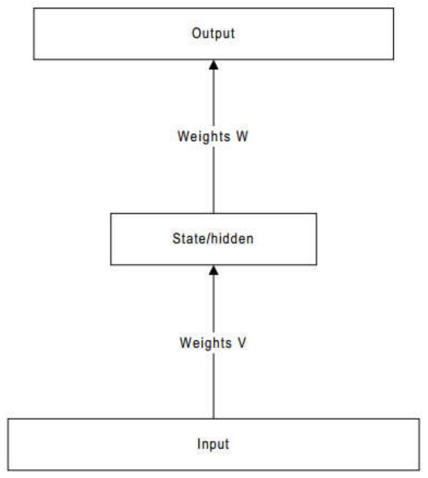


- *RNN not only operate on an input space but also on an internal state space – a trace of what already has been processed by the network.*

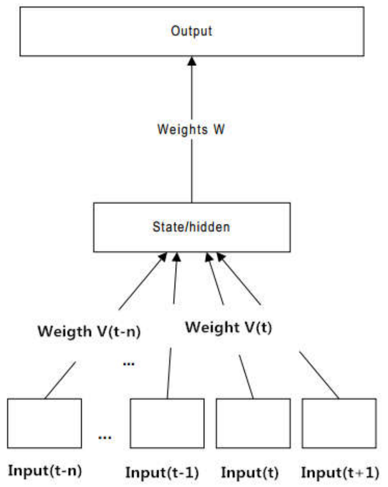
Hammer, 2000 : RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy.

Benefits of RNN over Tapped Delay Line

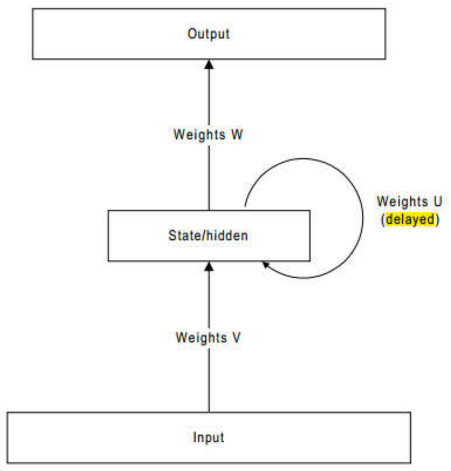
- *The easiest way to incorporate temporal or sequential information is to make the temporal domain spatial and use a feedforward architecture.*



feedforward



tapped delay line

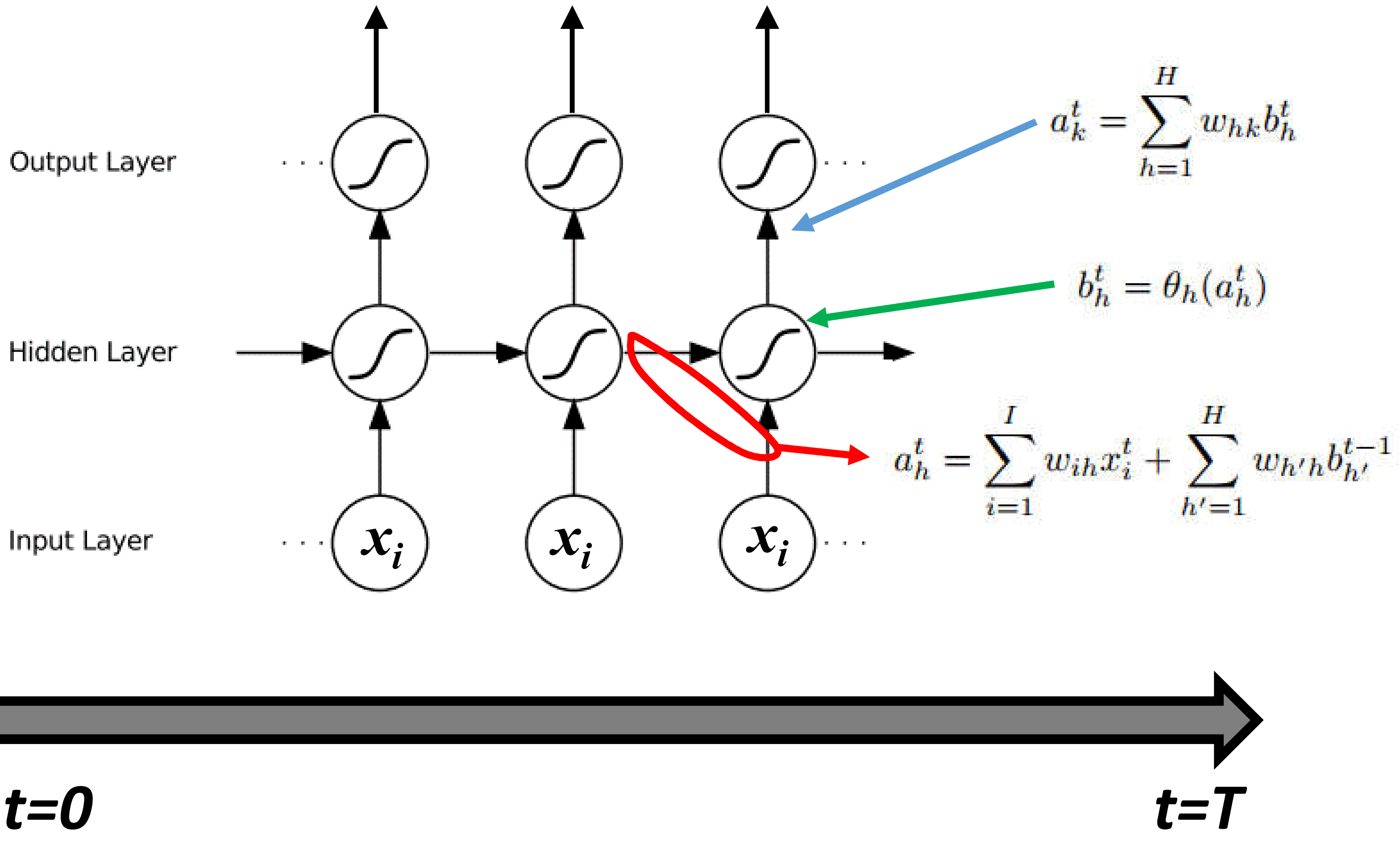


RNN

- *Disadvantages: include that the user has to select the maximum number of time steps which is useful to the network.*

- *In addition, the large number of weights requires a larger set of examples to avoid over-specialization.*

RNN Implementation – forward pass



$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

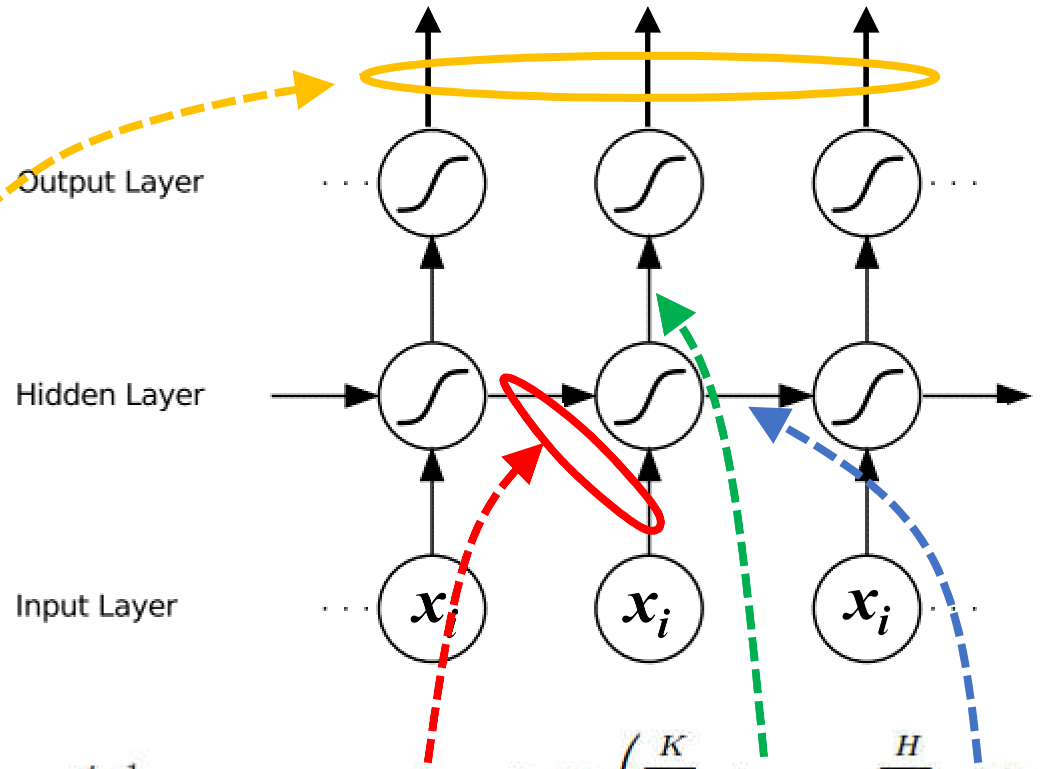
Same weights are reused at every timestep

RNN Implementation – backward pass

$$\boxed{= \frac{1}{2}\sum_{p}^{n} \sum_{k}^{m} (d_{pk} - y_{pk})^2}$$

Error signal

$$\delta_j^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial a_j^t}$$



$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right)$$

t=0 **t=T**

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

RNN Implementation – backward pass

Other perspectives

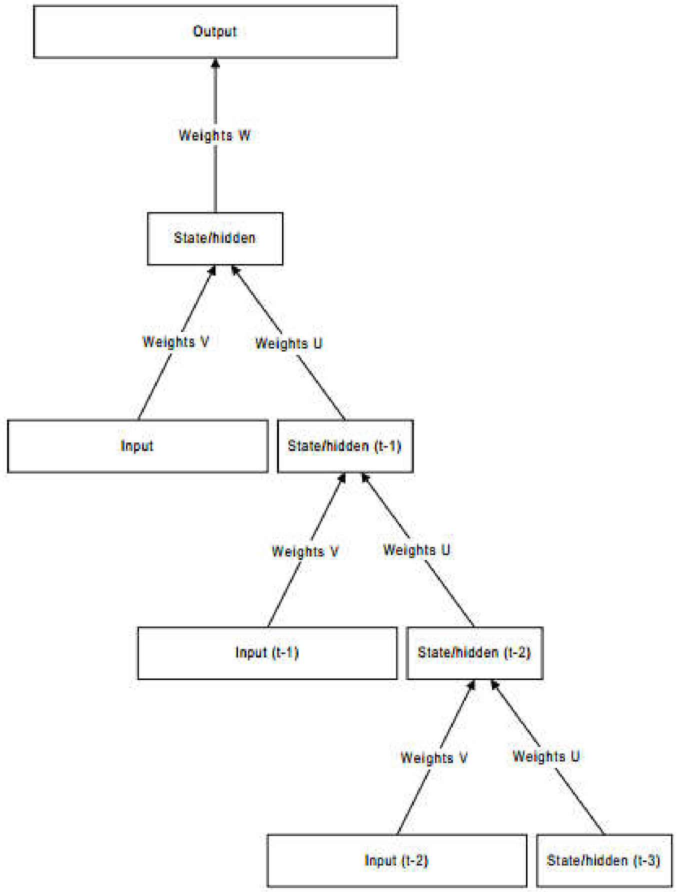


Figure 5: The effect of unfolding a network for BPTT ($\tau = 3$).

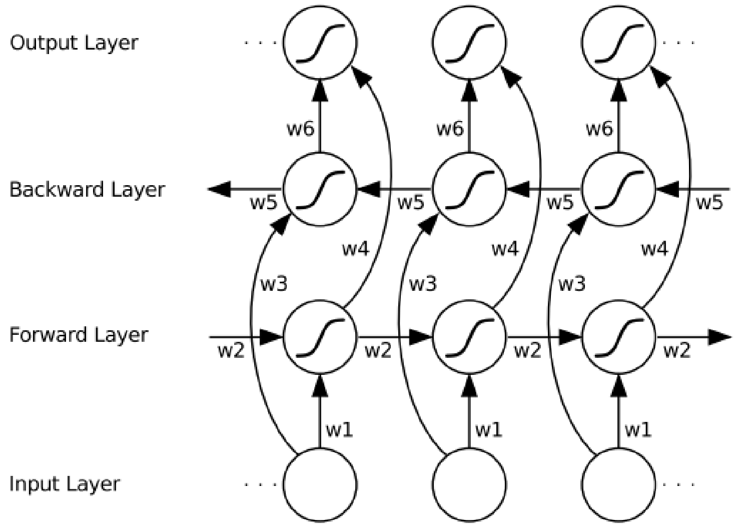
$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

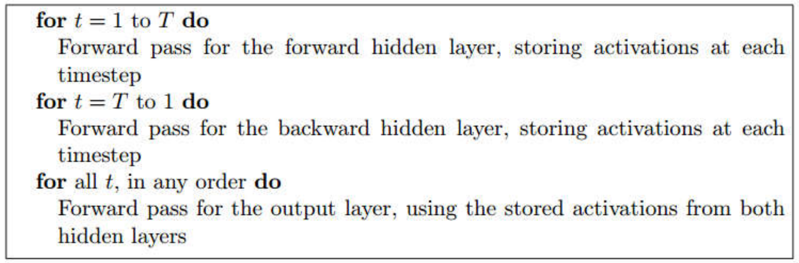
- ***Perspective One***
- 不同timestep的输入可以看成不同维度。
- 最终权重的更新要考虑所有维度要求下降的梯度。
- ***Perspective Two***
- 隐藏层之间的连接刻画了状态转移的规则。

Bidirectional RNN (BRNN)

- *Standard RNNs process sequences in temporal order, they ignore future context.*

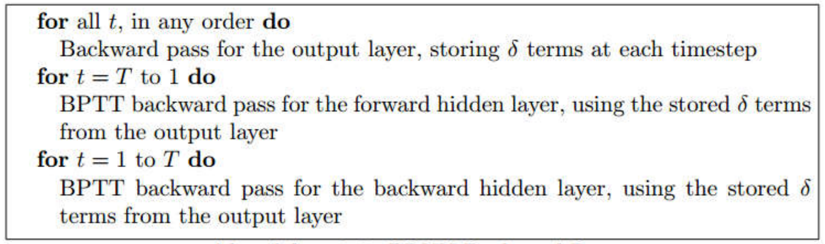
- *BRNN present each training sequence forwards and backwards to two separate recurrent hidden layers, both of which are connected to the same output layer.*





```
for $t = 1$ to $T$ do
    Forward pass for the forward hidden layer, storing activations at each timestep
for $t = T$ to 1 do
    Forward pass for the backward hidden layer, storing activations at each timestep
for all $t$, in any order do
    Forward pass for the output layer, using the stored activations from both hidden layers
```

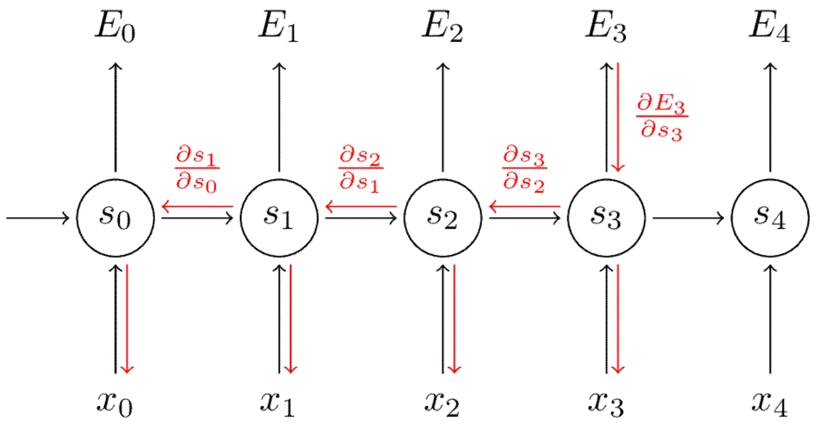
Algorithm 3.1: BRNN Forward Pass

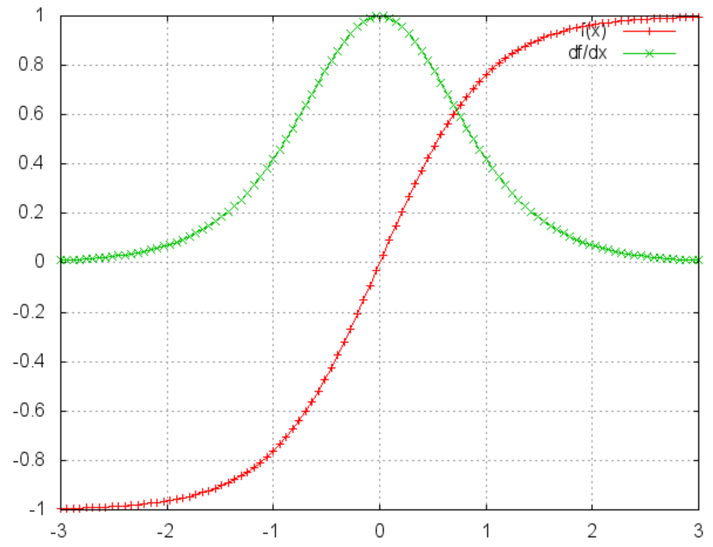


```
for all $t$, in any order do
    Backward pass for the output layer, storing $\delta$ terms at each timestep
for $t = T$ to 1 do
    BPTT backward pass for the forward hidden layer, using the stored $\delta$ terms from the output layer
for $t = 1$ to $T$ do
    BPTT backward pass for the backward hidden layer, using the stored $\delta$ terms from the output layer
```

Algorithm 3.2: BRNN Backward Pass

RNN - Vanishing Gradient Problem


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4



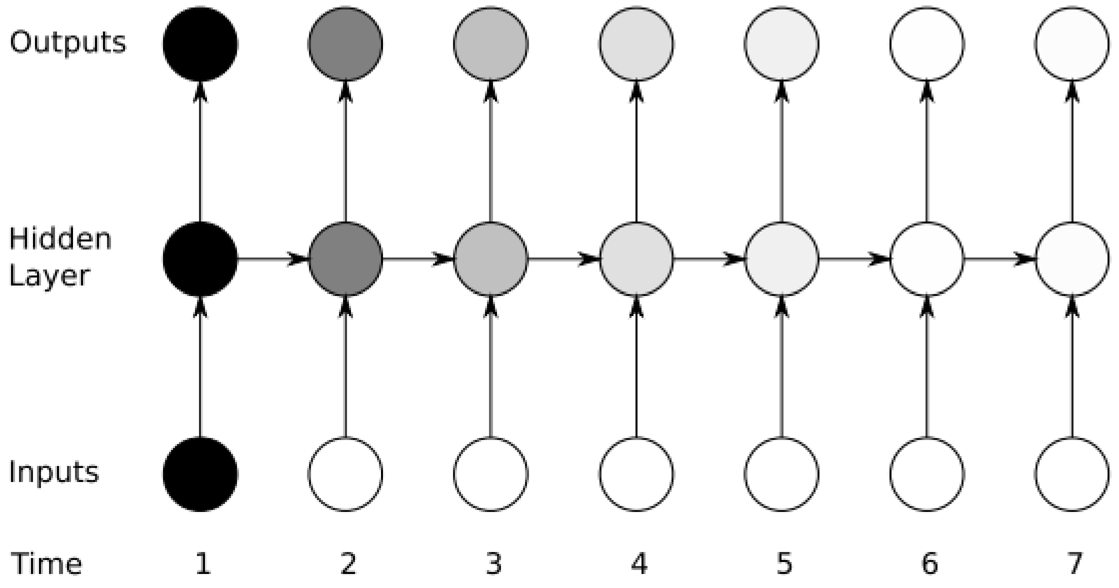
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

幸运的是，已经有一些方法解决了梯度消失问题。合适的初始化矩阵W可以减小梯度消失效应，正则化也能起作用。更好的方法是选择ReLU而不是sigmoid和tanh作为激活函数。ReLU的导数是常数值0或1，所以不可能会引起梯度消失。更通用的方案时采用长短项记忆（LSTM）或门限递归单元（GRU）结构。LSTM在1997年第一次提出，可能是目前在NLP上最普遍采用的模型。GRU，2014年第一次提出，是LSTM的简化版本。这两种RNN结构都是为了处理梯度消失问题而设计的，可以有效地学习到长距离依赖。

RNN – Not suitable for long term dependence



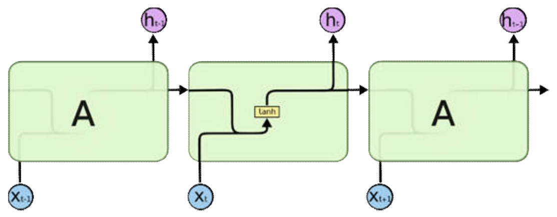
- *Reason one: The gradient is decaying. Therefore long term input can long used for training the network. As a result, RNN can not learn long term dependence.*

- *Reason two:* he sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

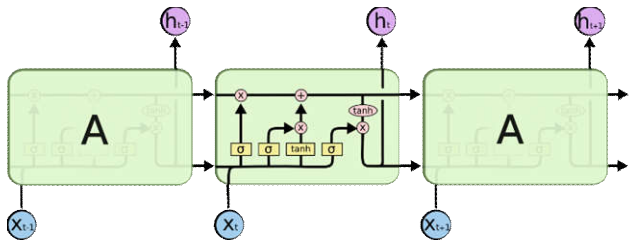
Long Short Temporal Memory (LSTM)

- *LSTMs were designed to combat vanishing gradients through a gating mechanism.*

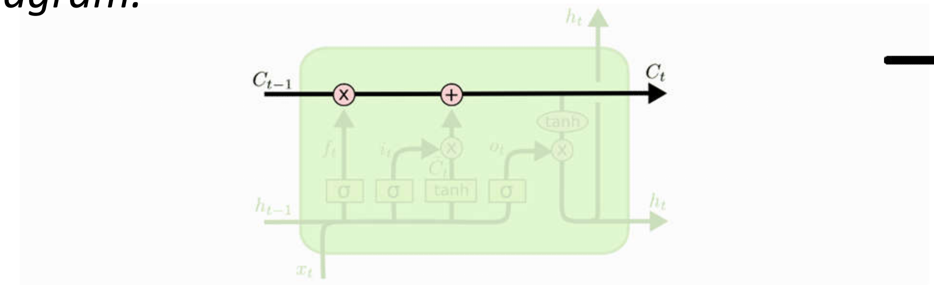
Conventional RNN

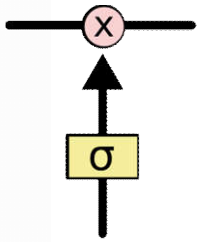


LSTM



- *The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.*





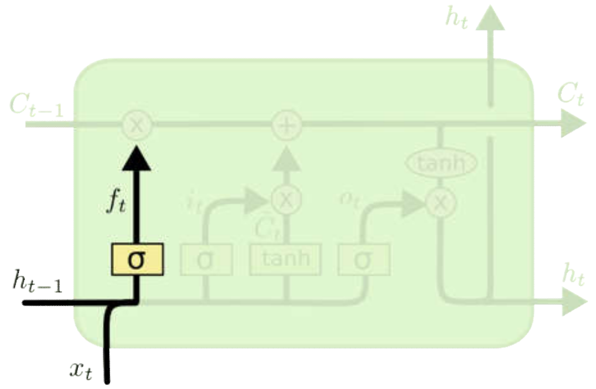
Elementwise Multiplying

- *The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.*

- *The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.*

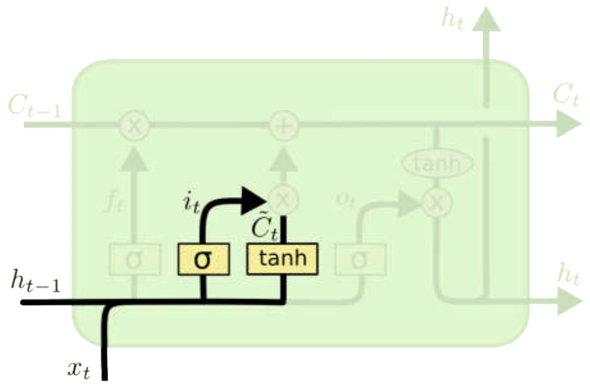
Forward of LSTM

Step1： *Decide what information we're going to throw away from the cell state.*



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

Step2： *Decide what new information we're going to store in the cell state.*

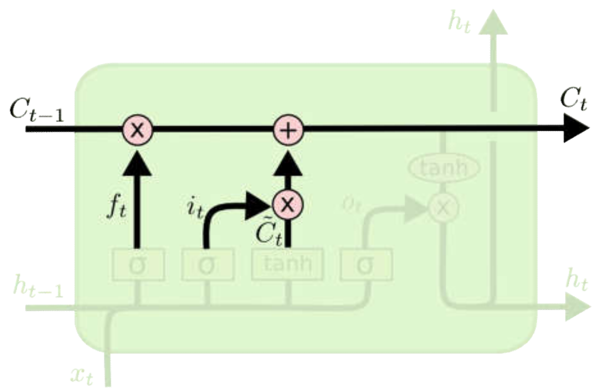


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

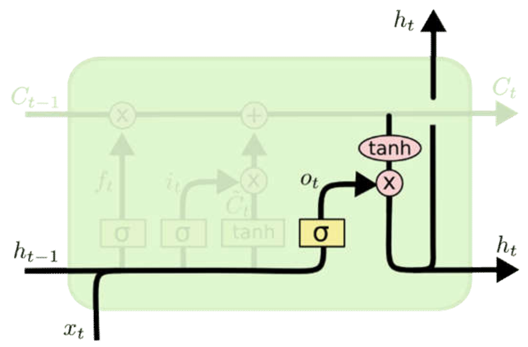
Forward of LSTM

Step3： *Update the old cell state.*



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

Step4： *Decide what we're going to output*



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

backward of LSTM

reverse the arrow

$$\epsilon_c^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1} \tag{4.11}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \tag{4.12}$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{4.13}$$

Cells Input

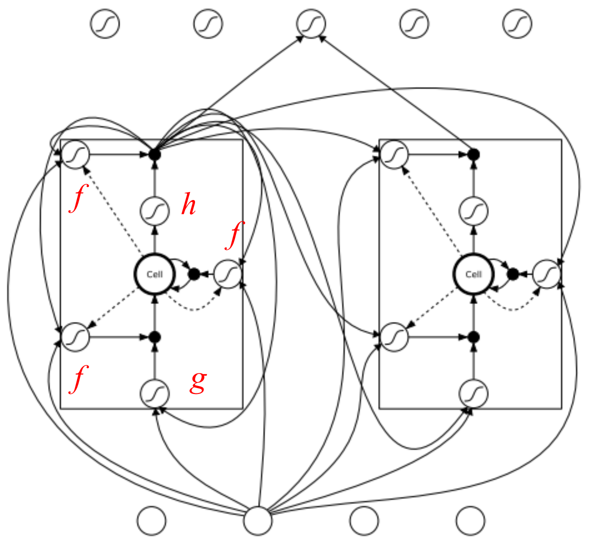
$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t \tag{4.14}$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{4.15}$$

Input Gates

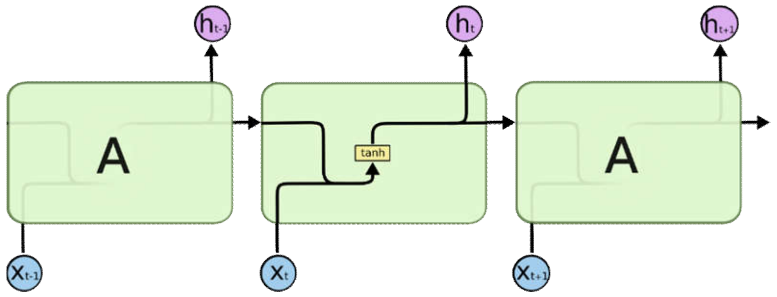
$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{4.16}$$



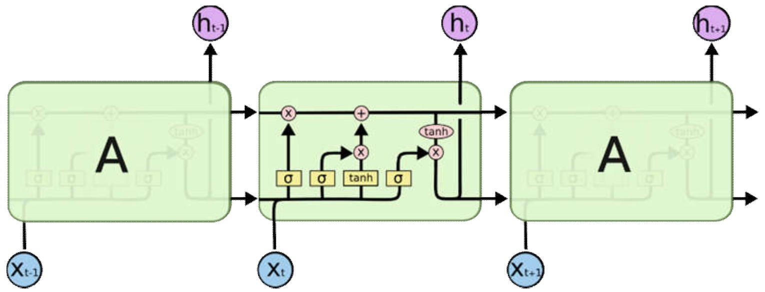
Why LSTM can handle long term dependence

Reason 1: Combat gradient vanishing

Conventional RNN



LSTM



$$S_t = f(S_{t-1}, x_t)$$

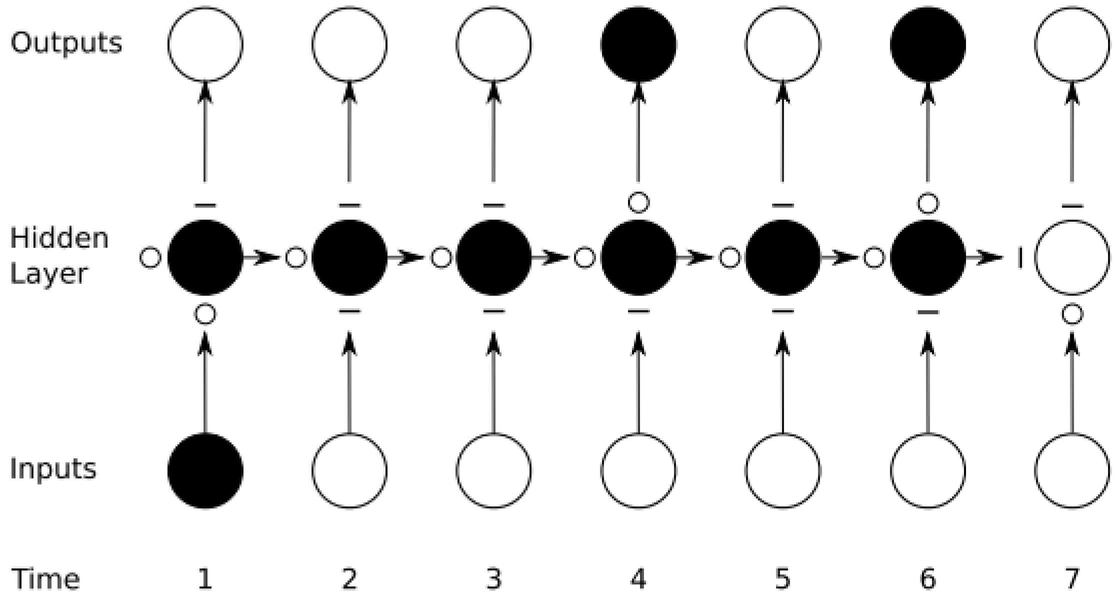
$$S_t = \sum_{\tau=1}^{t} \Delta S_\tau$$

- 传统的RNN总是用“覆写”的方式计算状态$S_t = f(S_{t-1}, x_t)$，其中$f(\cdot)$ 表示仿射变换外面在套一个Sigmoid，x_t表示输入序列在时刻t 的值。根据求导的链式法则，这种形式直接导致梯度被表示为连成积的形式，以致于造成梯度消失——粗略的说，很多个小于1的项连乘就很快的逼近零。

- 现代的RNN（包括但不限于使用LSTM单元的RNN）使用“累加”的形式计算状态：$S_t = \sum_{\tau=1}^{t} \Delta S_\tau$ ，其中的ΔS_τ显示依赖序列输入x_t. 稍加推导即可发现，这种累加形式导致导数也是累加形式，因此避免了梯度消失。

Why LSTM can handle long term dependence

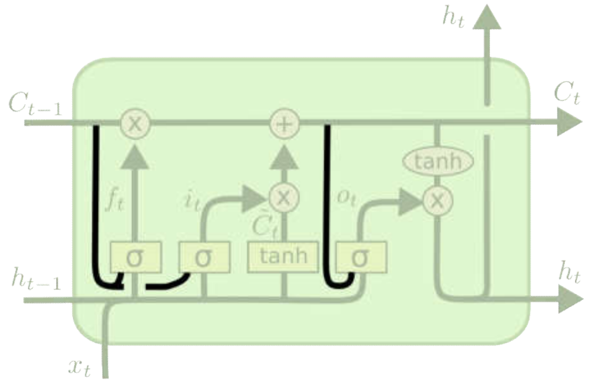
Reason 2: Combat activation overwrite



The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

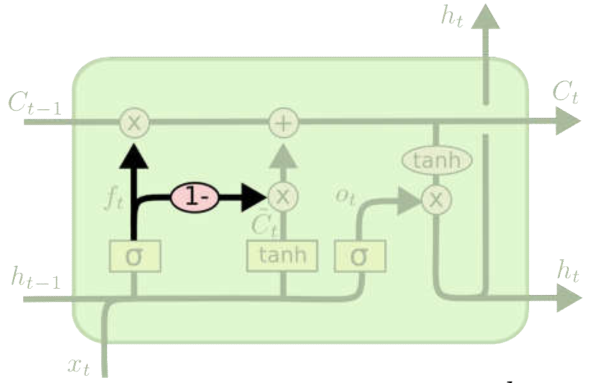
Variation of LSTM

Peephole Connections



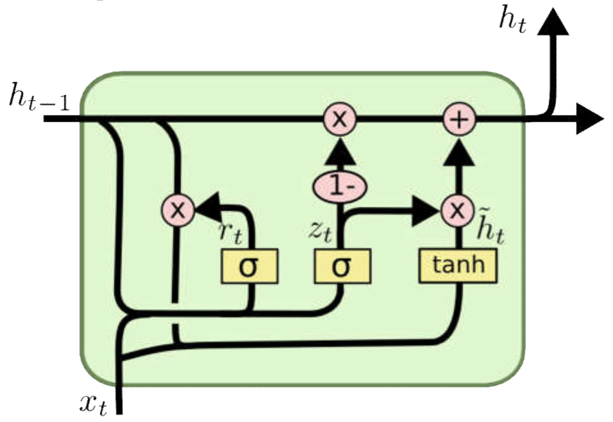
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Coupled Gates



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

GRU



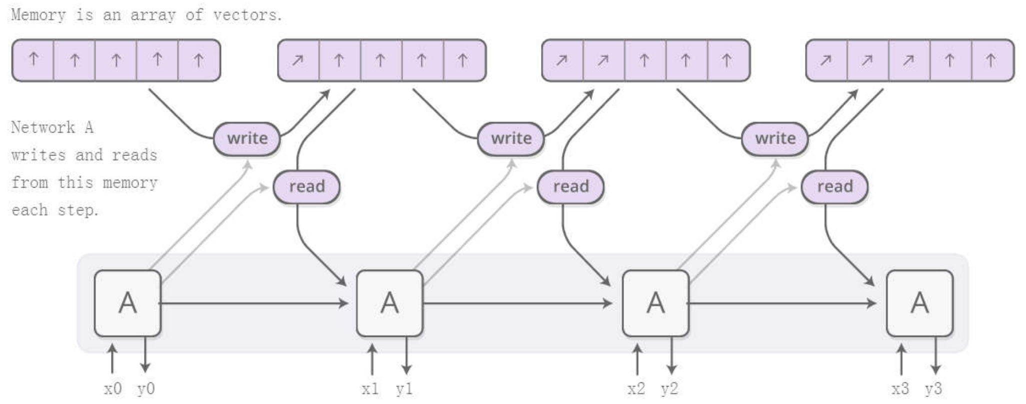
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Extensions of RNN – Attention

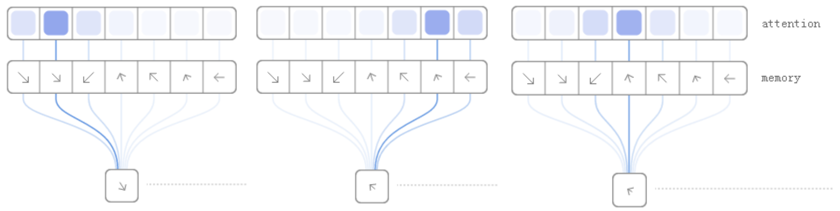
focusing on part of a subset of the information they're given at different step

Neural Turing Machines (NTM)

- Combine a RNN with an external memory bank.
- Every step, read and write every memory to different extents.

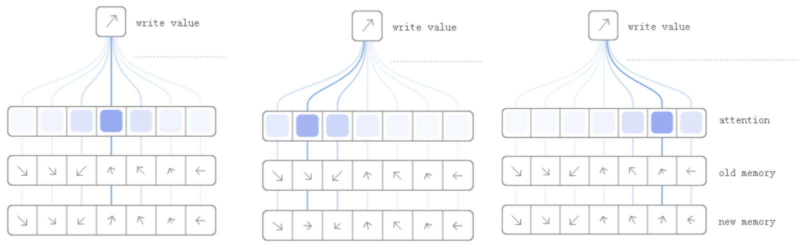

Memory is an array of vectors.
Network A writes and reads from this memory each step.
write
read
A
x0 y0
x1 y1
x2 y2
x3 y3

Read



$$r \leftarrow \sum_i a_i M_i$$

Write



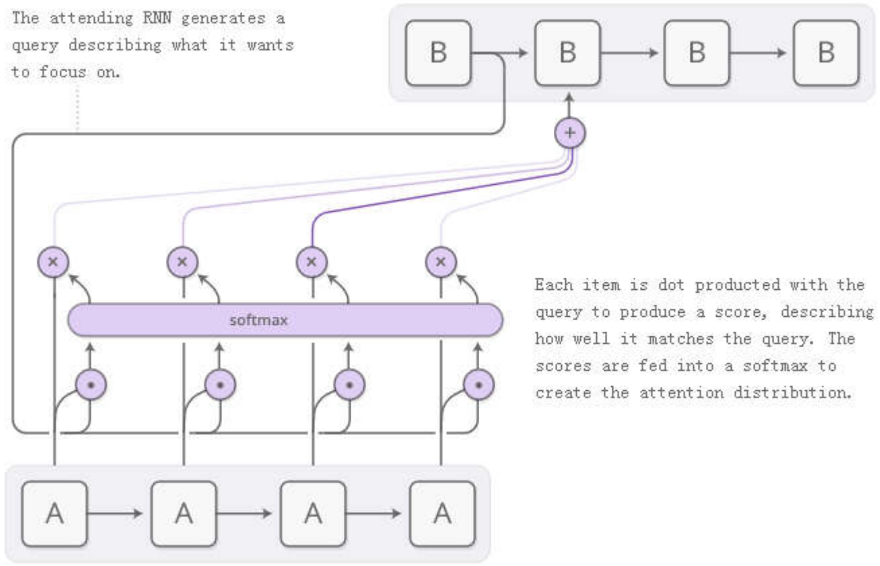
$$M_i \leftarrow a_i w + (1 - a_i) M_i$$

Extensions of RNN – Attention

focusing on part of a subset of the information they're given at different step

Attention Interface

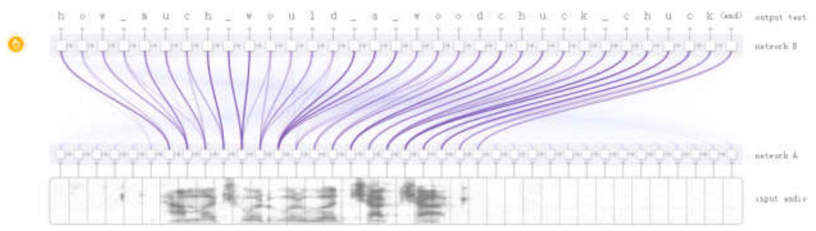
- Interaction Interface between Neural Network A with Neural Network B


The attending RNN generates a query describing what it wants to focus on.

Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

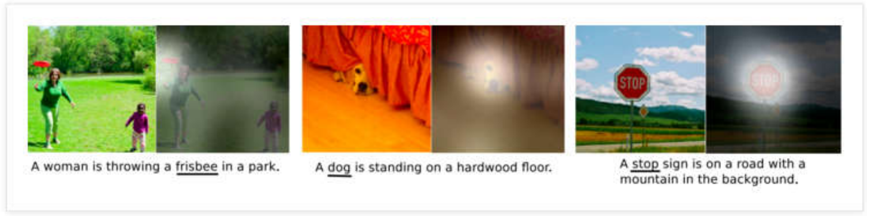
RNN + Attention + RNN

for voice recognition


h o w _ m u c h _ w o u l d _ a _ w o o d c h u c k _ c h u c k (end)
output text
network B
network A
input audio

CNN + Attention + RNN

for image description


A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

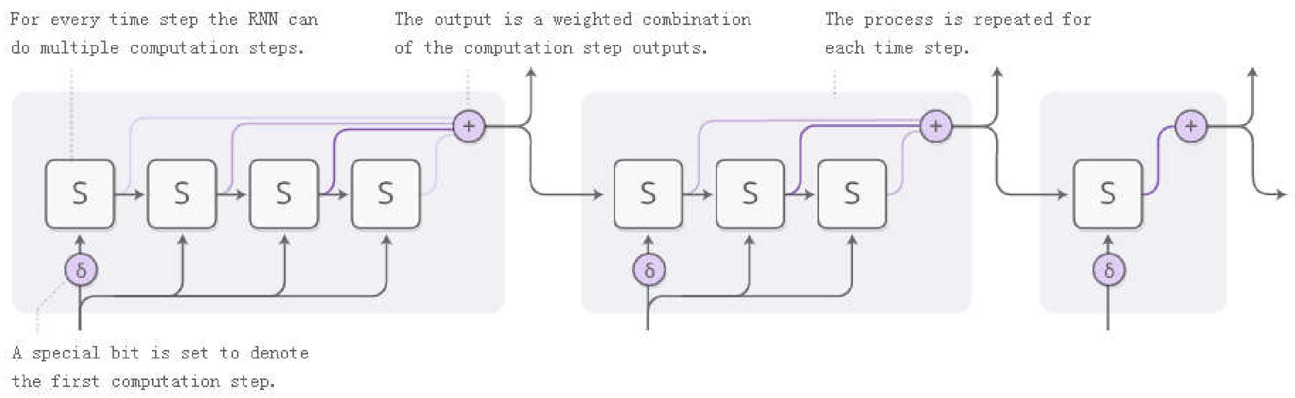
Figure from [3]

Extensions of RNN – Attention

focusing on part of a subset of the information they're given at different step

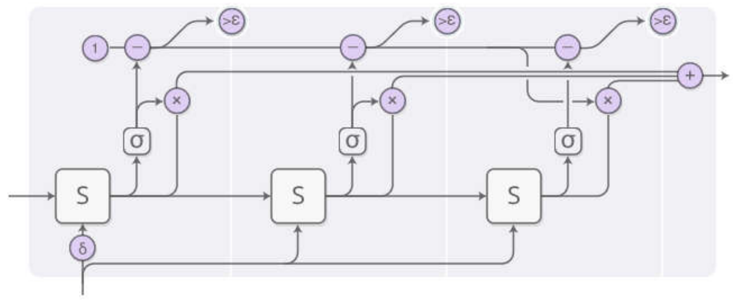
Adaptive Computation Time

- Different amounts of computation in each time step
- Allow the RNN to do multiple steps of computation for each time step



For every time step the RNN can do multiple computation steps.
The output is a weighted combination of the computation step outputs.
The process is repeated for each time step.
A special bit is set to denote the first computation step.
S
δ
+

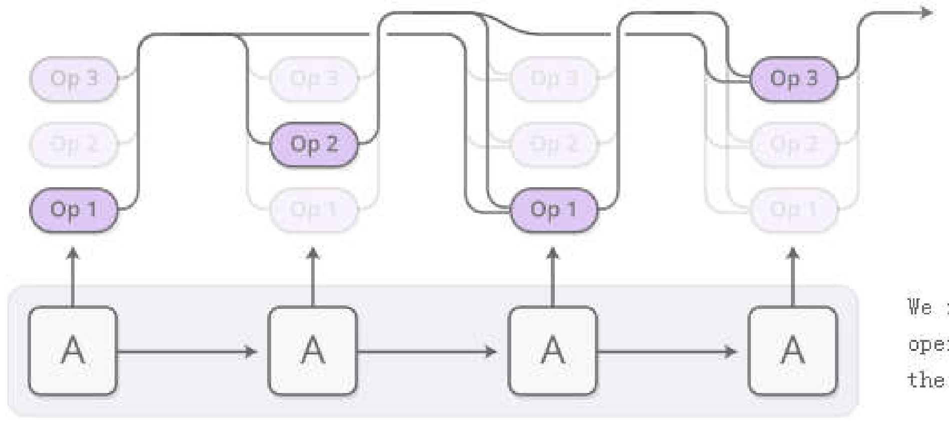
Implementation



Extensions of RNN – Attention

focusing on part of a subset of the information they're given at different step

Neural Programmer

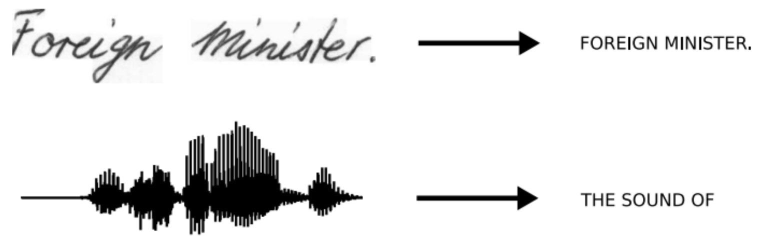


We run all of the operations and average the outputs together.

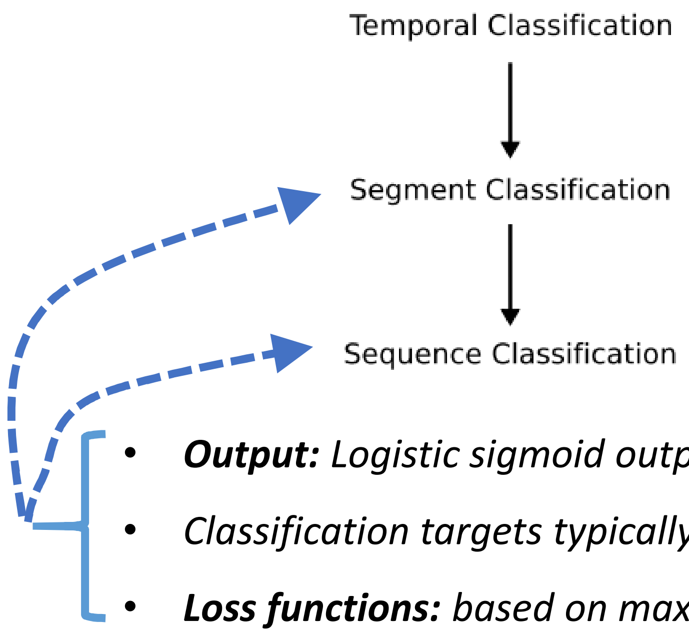
Connectionist temporal classification (CTC)

output layer for recurrent neural networks

- *The goal of sequence labelling is to assign **sequences of labels**, drawn from a fixed alphabet, to **sequences of input data**.*



- *Three levels of sequence labelling*



locations of the target labels are unknown
RNN + Markov or RNN + CTC

locations of the target labels are known in advance
Context aids inferring

the label sequences are constrained to be length 1
standard pattern classification

- ***Output:** Logistic sigmoid output (two classes) and softmax output (multiple classes)*
- *Classification targets typically presented at the ends of the sequences or segments.*
- ***Loss functions:** based on maximum-likelihood* $\mathcal{L}(x,z) = -\sum_{k=1}^{K} z_k \ln y_k$ z_k *is one-hot*

Connectionist temporal classification (CTC)

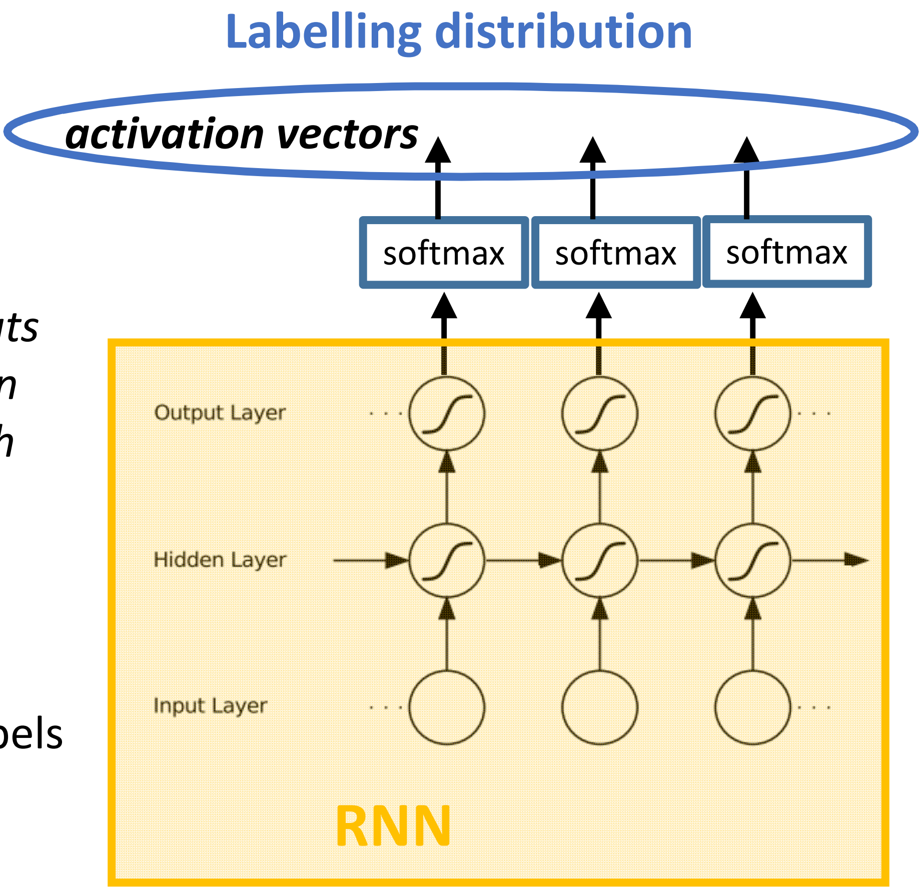
output

- *Assume that the labels are drawn from an alphabet* ***A****,CTC consists of a softmax output layer with one more unit than there are labels in* ***A*** *(blank label or no label).*

- *The activation vector is the probabilities of the corresponding labels occur at particular times.*

- *The complete sequence of network outputs vector is then used to define a distribution over all possible label sequences of length up to that of the input sequence.*

- *Notation*
  <u>label: </u>element in **A'** (**A** + blank label)
  <u>labelling/label sequence: </u>sequence of labels



Connectionist temporal classification (CTC)

path & labelling

- 由于被打断/动作持续时间不同, 同一个labelling可能有不同的表现形式.
- 这些不同的表现形式导致的不同network output 称为路径path
- 因此CTC允许同时对不同label之间、非空label与空label之间、相同label之间的转换过程进行建模。
- it allows the network to predict the labels without knowing in advance *where* they occur.

- Map paths (**π**) to labellings (**l**)

$$\text{many-to-one function } \mathcal{F} : A'^T \mapsto A^{\leq T}$$

$$\mathcal{F}(a - ab-) = \mathcal{F}(-aa - -abb) = aab$$

- Together, these outputs define the probabilities of all possible ways of aligning all possible label sequences with the input sequence.
  Given input sequence **x**, the possibility of labelling **l** is given by

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{F}^{-1}(\mathbf{l})} \boxed{p(\pi|\mathbf{x})}$$

Connectionist temporal classification (CTC)

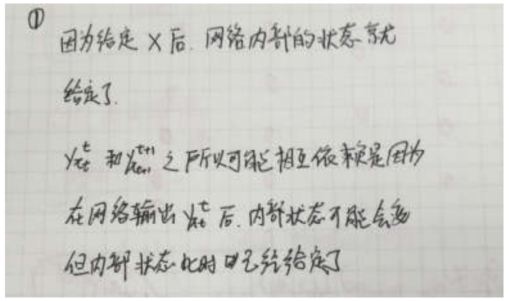
Critical Assumption of p(π|x)

- Assume the output probabilities at each timestep to be **independent** of those at other timesteps

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}$$

网络在t时刻的输出，即估计的在t时刻出现label π_t 概率

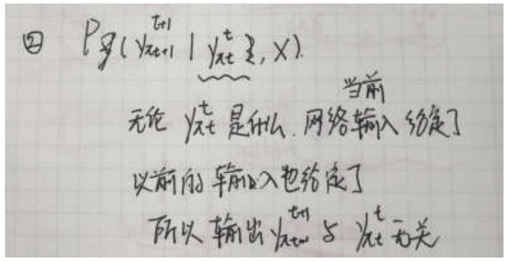
- 对这一假设的三种理解



① 因为给定 X 后. 网络内部的状态就给定了.

$y^t_{\pi_t}$ 和 $y^{t+1}_{\pi_{t+1}}$ 之所以可能相互依赖是因为在网络输出 $y^t_{\pi_t}$ 后. 内部状态可能会变

但内部状态此时已经给定了

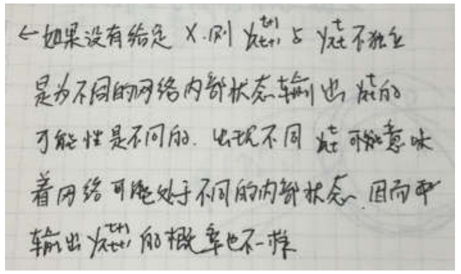


② $P(y^{t+1}_{\pi_{t+1}} \mid \underbrace{y^t_{\pi_t}}\}, X)$.

无论 $y^t_{\pi_t}$ 是什么. 网络输入给定了 当前

以前的输入也给定了

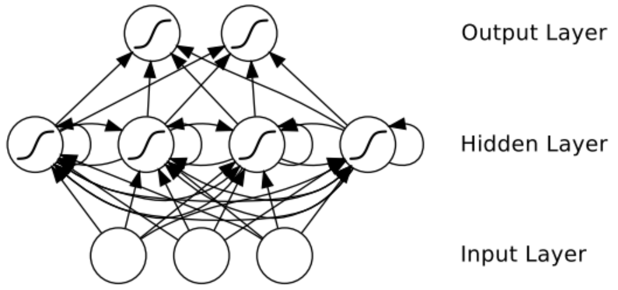
所以输出 $y^{t+1}_{\pi_{t+1}}$ 与 $y^t_{\pi_t}$ 无关



← 如果没有给定 X. 则 $y^{t+1}_{\pi_{t+1}}$ 与 $y^t_{\pi_t}$ 不独立

是为不同的网络内部状态输出 $y^t_{\pi_t}$ 的可能性是不同的. 出现不同 $y^t_{\pi_t}$ 可能意味着网络可能处于不同的内部状态. 因而输出 $y^{t+1}_{\pi_{t+1}}$ 的概率也不一样

- This is ensured by requiring that no feedback connections exist from the output layer to itself or the network.



CTC - Forward-Backward Algorithm

modify label sequence to traverse all possible paths

- Modified label sequence **l’**, with blanks added to the beginning and the end of **l**, and inserted between every pair of consecutive labels. If the length of **l** is U, the length of **l’** is therefore U’= 2U + 1.

- Example: **l** = { c a t }, **l’** = { - c – a – t - }, -stands for blank label.

- *Purpose*: In calculating the probabilities of prefixes of **l**′ we allow all transitions between blank and non-blank labels, those between any pair of distinct non-blank labels, and stay on the same label.

CTC - Forward-Backward Algorithm

Forward variable A Dynamic Programming Perspective

- ***Definition:*** α(t,u) is the summed probability of all length t paths that (a) are mapped by F onto the length u/2 prefix of **l,** (b) reach the u-th state in **l'**.

Timestep index of path

Index of state in **l'**

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y^i_{\pi_i} \qquad V(t,u) = \{\pi \in A'^t : \mathcal{F}(\pi) = \mathbf{l}_{1:u/2}, \pi_t = l'_u\}$$

$$p(t, \mathbf{l}_{1:u/2}|\mathbf{x}) = a(t, u) + a(t, u-1) \xrightarrow{t=T} p(\mathbf{l}|\mathbf{x}) = \alpha(T, U') + \alpha(T, U'-1)$$

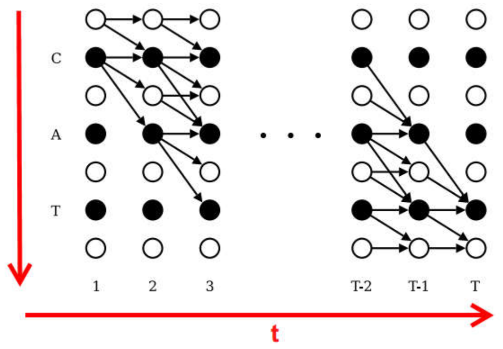
- ***Iteration rules***

Transfer probability

Transferred from where

$$\alpha(t,u) = y^t_{l'_u} \sum_{i=f(u)}^{u} \alpha(t-1, i)$$

$$f(u) = \begin{cases} u-1 & \text{if } l'_u = \text{blank or } l'_{u-2} = l'_u \\ u-2 & \text{otherwise} \end{cases}$$



CTC - Forward-Backward Algorithm

Forward variable A Dynamic Programming Perspective

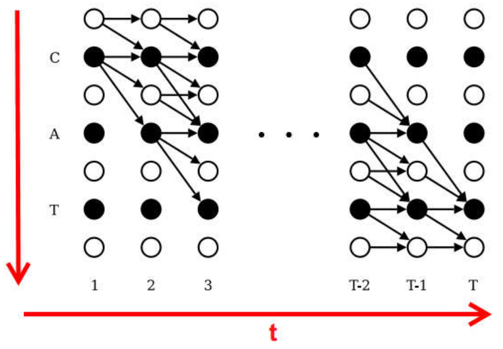
- ***Boundary conditions***

$$\alpha(t,0) = 0\ \forall t$$

$$\alpha(1,1) = y_b^1$$

$$\alpha(1,2) = y_{l_1}^1$$

$$\alpha(1,u) = 0,\ \forall u > 2$$



间插blank后labelling的总长度

剩余时间内最多可跨越的label个数
最乐观情况一个timestep跨过两个状态

$$\alpha(t,u) = 0\ \forall u < \underbrace{U' - \boxed{2(T-t)} - 1}$$

最后一个label是blank可以不经历

在timestep t时，至少需要经历的label个数

CTC - Forward-Backward Algorithm

backward variable

- ***Definition:*** The *backward variables* $\beta(t, u)$ are defined as the summed probabilities of all paths starting at $t + 1$ that complete **l** when appended to any path contributing to $\alpha(t, u)$.

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{\pi_i}^{t+i} \qquad W(t,u) = \{\pi \in A'^{T-t} : \mathcal{F}(\hat{\pi} + \pi) = \mathbf{l}\ \forall \hat{\pi} \in V(t,u)\}$$

- ***Iteration rules***

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{l'_i}^{t+1}$$

$$g(u) = \begin{cases} u+1 & \text{if } l'_u = \text{blank or } l'_{u+2} = l'_u \\ u+2 & \text{otherwise} \end{cases}$$

- ***Boundary conditions***

$$\beta(t, U'+1) = 0\ \forall t$$

$$\beta(T, U') = \beta(T, U'-1) = 1$$

$$\beta(T, u) = 0, \ \ \forall u < U'-1$$

$$\beta(t,u) = 0\ \forall u > \overset{\nearrow}{2t}$$

最多可跨越的label个数
最乐观情况一个timestep跨过两个状态

CTC - Forward-Backward Algorithm

loss function

- ***Definition:*** *Loss function is defined as the negative log probability of correctly labelling all the training examples in some training set S*

$$\mathcal{L}(S) = -\ln \prod_{(\mathbf{x},\mathbf{z})\in S} p(\mathbf{z}|\mathbf{x}) = -\sum_{(\mathbf{x},\mathbf{z})\in S} \ln p(\mathbf{z}|\mathbf{x})$$

$$\alpha(t,u)\beta(t,u) = \sum_{\pi\in X(t,u)} \prod_{t=1}^{T} y^t_{\pi_t} \Longrightarrow \alpha(t,u)\beta(t,u) = \sum_{\pi\in X(t,u)} p(\pi|\mathbf{x})$$

$$X(t,u) = \{\pi \in A'^T : \mathcal{F}(\pi) = \mathbf{z}, \pi_t = \mathbf{z}'_u\}$$

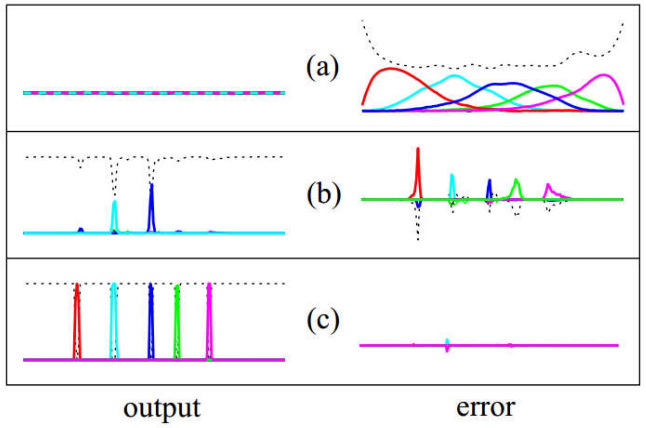
- *This is the portion of the total probability p(z|x) due to those paths going through z_u' at timestep t. For any t, we can therefore sum over all u to get*

$$p(\mathbf{z}|\mathbf{x}) = \sum_{u=1}^{|\mathbf{z}'|} \alpha(t,u)\beta(t,u)$$

CTC

loss gradient

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{\partial \ln p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})}\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t}$$



- Noting that the same label (or blank) may occur several times in a single labelling

$$\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = \frac{1}{y_k^t}\sum_{u\in B(\mathbf{z},k)} \alpha(t,u)\beta(t,u). \qquad B(\mathbf{z},k) = \{u : \mathbf{z}'_u = k\}$$

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})y_k^t}\sum_{u\in B(\mathbf{z},k)} \alpha(t,u)\beta(t,u)$$

- Error signal

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = -\sum_{k'}\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial y_{k'}^t}\frac{\partial y_{k'}^t}{\partial a_k^t}$$

$$y_k^t = \frac{e^{a_k^t}}{\sum_{k'} e^{a_{k'}^t}} \quad \Rightarrow \frac{\partial y_{k'}^t}{\partial a_k^t} = y_{k'}^t\delta_{kk'} - y_{k'}^t y_k^t$$

$$\boxed{p(\mathbf{z}|\mathbf{x}) = \sum_{u=1}^{|\mathbf{z}'|}\alpha(t,u)\beta(t,u)}$$

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})}\sum_{u\in B(\mathbf{z},k)} \alpha(t,u)\beta(t,u)$$

CTC - Decoding

Best Path Decoding

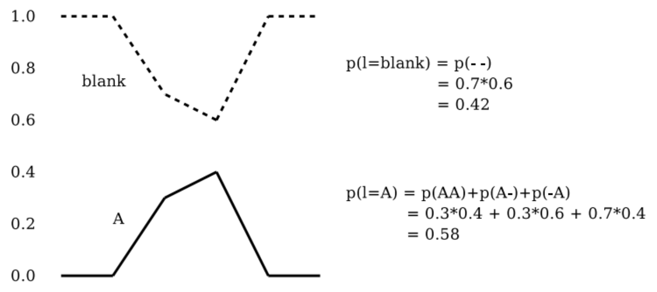
- Once the network is trained, we would ideally label some unknown input sequence x by choosing the most probable labelling **l***

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{x})$$

- The computation complexity is too high to enumerate all possible **l.**

- ***Best Path Decoding:*** the most probable path corresponds to the most probable labelling

$$\mathbf{l}^* \approx \mathcal{F}(\pi^*) \qquad \pi^* = \arg\max_\pi p(\pi|\mathbf{x}).$$



Another perspective

从单个时刻看A确实出现的概率较小 倾向于没有出现。但是结合上下文的模式分布来看 更倾向于A出现了。因为一个手势出现以前往往伴随着一些准备动作 而这些准备动作可以帮助判断A是否出现 Best Path Decoding没有利用到这些上下文信息。

CTC - Decoding

Prefix Search Decoding

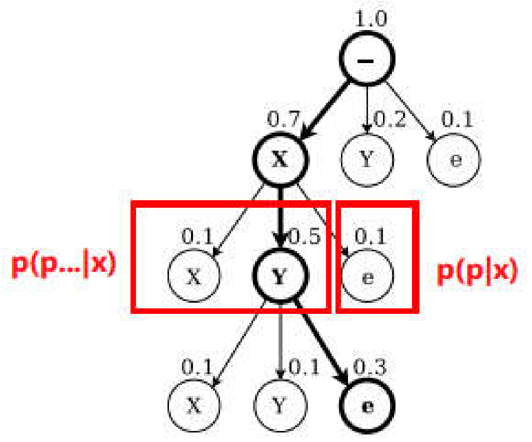
- $\gamma(\mathbf{p}_n, t)$ be the probability of the network outputting prefix **p** by time t such that a non-blank label is output at t.
- $\gamma(\mathbf{p}_b, t)$ be the probability of the network outputting prefix **p** by time t such that the blank label is output at t.

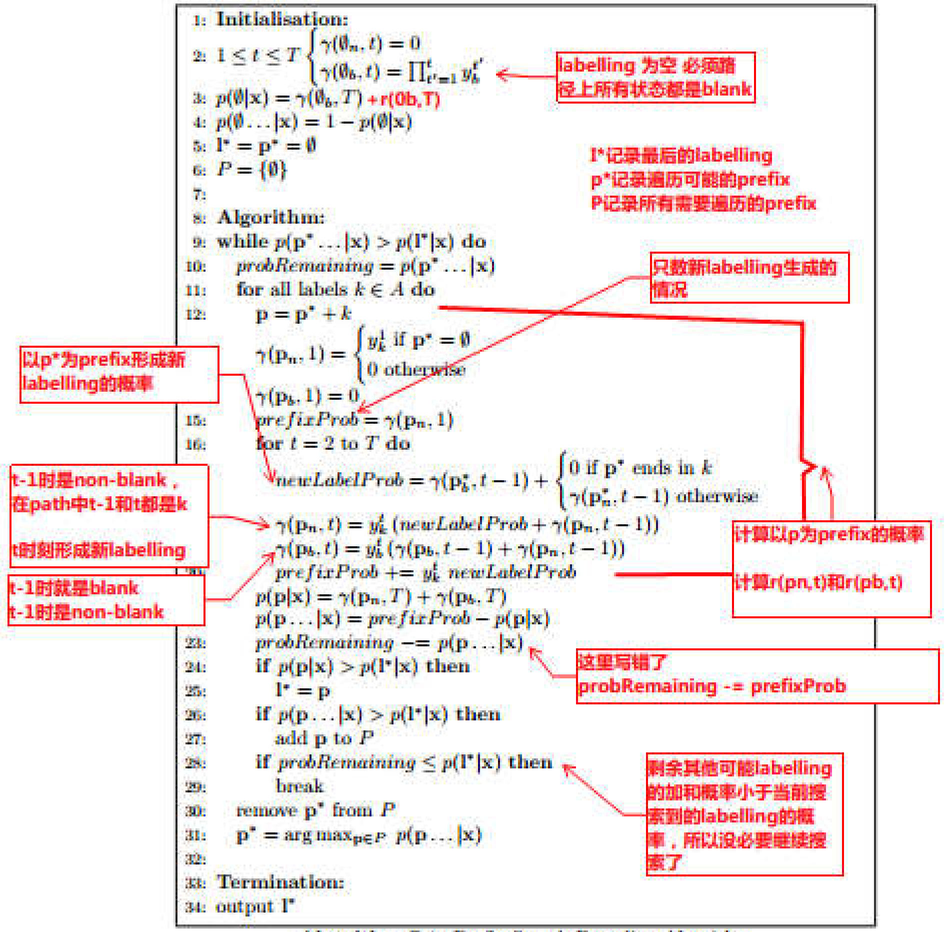
$$p(\mathbf{p}|\mathbf{x}) = \gamma(\mathbf{p}_n, T) + \gamma(\mathbf{p}_b, T)$$

$$p(\mathbf{p}\ldots|\mathbf{x}) = \sum_{\mathbf{l}\neq\emptyset} p(\mathbf{p}+\mathbf{l}|\mathbf{x})$$

p(p|x) 表示p为完整labelling的概率

p(p...|x)表示p仅为prefix的概率





```
Initialisation:
$1 \le t \le T \begin{cases} \gamma(\emptyset_n, t) = 0 \\ \gamma(\emptyset_b, t) = \prod_{t'=1}^{t} y_b^{t'} \end{cases}$
$p(\emptyset|\mathbf{x}) = \gamma(\emptyset_b, T)$ +r(0b,T)
$p(\emptyset\ldots|\mathbf{x}) = 1 - p(\emptyset|\mathbf{x})$
$\mathbf{l}^* = \mathbf{p}^* = \emptyset$
$P = \{\emptyset\}$

Algorithm:
while $p(\mathbf{p}^*\ldots|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ do
  $probRemaining = p(\mathbf{p}^*\ldots|\mathbf{x})$
  for all labels $k \in A$ do
    $\mathbf{p} = \mathbf{p}^* + k$
    $\gamma(\mathbf{p}_n, 1) = \begin{cases} y_k^1 \text{ if } \mathbf{p}^* = \emptyset \\ 0 \text{ otherwise} \end{cases}$
    $\gamma(\mathbf{p}_b, 1) = 0$
    $prefixProb = \gamma(\mathbf{p}_n, 1)$
    for $t = 2$ to $T$ do
      $newLabelProb = \gamma(\mathbf{p}_b^*, t-1) + \begin{cases} 0 \text{ if } \mathbf{p}^* \text{ ends in } k \\ \gamma(\mathbf{p}_n^*, t-1) \text{ otherwise} \end{cases}$
      $\gamma(\mathbf{p}_n, t) = y_k^t\,(newLabelProb + \gamma(\mathbf{p}_n, t-1))$
      $\gamma(\mathbf{p}_b, t) = y_b^t\,(\gamma(\mathbf{p}_b, t-1) + \gamma(\mathbf{p}_n, t-1))$
      $prefixProb \mathrel{+}= y_k^t\ newLabelProb$
    $p(\mathbf{p}|\mathbf{x}) = \gamma(\mathbf{p}_n, T) + \gamma(\mathbf{p}_b, T)$
    $p(\mathbf{p}\ldots|\mathbf{x}) = prefixProb - p(\mathbf{p}|\mathbf{x})$
    $probRemaining \mathrel{-}= p(\mathbf{p}\ldots|\mathbf{x})$
    if $p(\mathbf{p}|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ then
      $\mathbf{l}^* = \mathbf{p}$
    if $p(\mathbf{p}\ldots|\mathbf{x}) > p(\mathbf{l}^*|\mathbf{x})$ then
      add $\mathbf{p}$ to $P$
    if $probRemaining \le p(\mathbf{l}^*|\mathbf{x})$ then
      break
  remove $\mathbf{p}^*$ from $P$
  $\mathbf{p}^* = \arg\max_{\mathbf{p}\in P} p(\mathbf{p}\ldots|\mathbf{x})$

Termination:
output $\mathbf{l}^*$
```

labelling 为空 必须路径上所有状态都是blank

l*记录最后的labelling
p*记录遍历可能的prefix
P记录所有需要遍历的prefix

只数新labelling生成的情况

以p*为prefix形成新labelling的概率

t-1时是non-blank，在path中t-1和t都是k

t时刻形成新labelling

t-1时就是blank
t-1时是non-blank

计算以p为prefix的概率

计算r(pn,t)和r(pb,t)

这里写错了
probRemaining -= prefixProb

剩余其他可能labelling的加和概率小于当前搜索到的labelling的概率，所以没必要继续搜索了

Algorithm 7.1: Prefix Search Decoding Algorithm