Unspuervised learning, Long Short Term Memory (LSTM), Transformer

Wen Yu

Unsupervised learning

One object of Deep Learning: **extract features automatically.**
We have no-lable data $x_1(k) \cdots x_n(k)$, but want to find the latent (hidden features) inside $x(k)$

$$(\lambda_1 \cdots \lambda_m) = PCA[x(k)]$$

But if we use

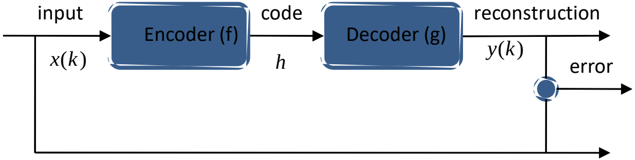
$$z = f(x), \quad \hat{x} = f^{-1}(z)$$

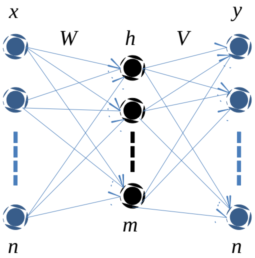
then

$$\hat{x} = f^{-1}[f(x(k))] = x(k)$$

So the maong F is regarded as a representation of the input $x(k)$

Autoencoder structure





Autoencoder structure

- encoder: input layer $\longrightarrow$ hidden layer

$$z = f(x) = \phi_f(Wx)$$

- decoder: hidden layer $\longrightarrow$ output layer

$$\hat{x} = g(z) = \phi_g(Vz)$$

ϕ_f and ϕ_g can be sigmoid functions, to simplify the computation (tied weights)

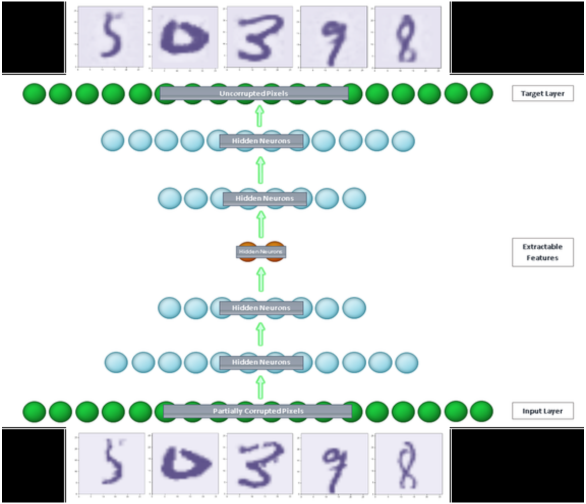
$$V = W^T$$

- decrease dimension

$$m < n$$

- When ϕ_f is a linear function, autoencoder becomes PCA

Unsupervised learning



Autoencoder training

- The object: we hope the decoded variable y can approximate the original variable x, although it is encoded into h.
- Define the reconstruction errors as

$$\begin{array}{ll} \text{squared error:} & e(k) = \left\| x(k) - \hat{x}(k) \right\|^2 \\ \text{cross entropy:} & e(k) = \sum_{i=1}^{n} \left[x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i) \right] \end{array}$$

- Loss functions

$$\text{Average:} \quad L = \frac{1}{N} \sum_{k=1}^{N} |e(k)|, \quad \min_{\theta} L = \sum_{x \in S} e(x, g[f(c)])$$

where the training data are $x(1) \cdots x(N) \in S$, $\theta = [W, p, q]$

Regularized autoencoder

Auto-encoder may perfectly construct the input without extracting any useful features. For example $f(x) = x$, or $m >> n$ (over-complete setting).
The loss functions should be modified.
L1 regularied (Lasso)

$$L = \sum_{x \in S} e\left(x, g\left[f\left(x\right)\right]\right) + \sum_{ij} |w_{ij}|$$

L2 regularied (Ridge)

$$L = \sum_{x \in S} e\left(x, g\left[f\left(x\right)\right]\right) + \lambda \sum_{ij} w_{ij}^2$$

where $W = [w_{ij}]$, λ is the weigh decay factor

Training

BP algorithm

$$\Delta W(k+1) = -\eta \frac{\partial L}{\partial W(k)}$$

can be used.
LS version

$$W(k) = \left(X^T X + \lambda I\right)^{-1} X^T Y$$

Boltzmann Machines

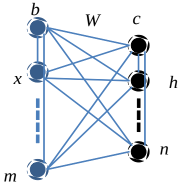
Geoffrey Hinton and Terry Sejnowski in 1985

- stochastic
- bi-direction network

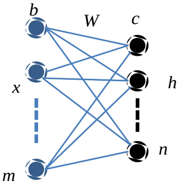
1. Boltzmann machines with unconstrained connectivity have not proven useful for practical problems in machine learning or inference
2. if the connectivity is properly constrained, the learning can be useful for practical problems

Restricted Boltzmann Machines

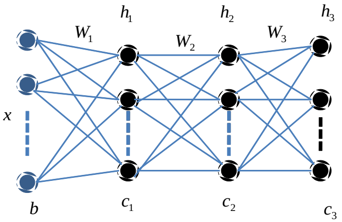
Undirected graphical model of the Restricted Boltzmann Machine (RBM)



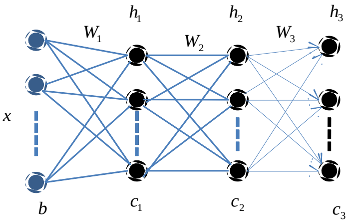
Boltzmann Machine



Restricted Boltzmann Machine



Deep Restricted Boltzmann Machines



Deep Belief Network

RBM structure

- no links between units of the same layer, only between input (or visible) units x_j and hidden units h_i. All hidden nodes are independent when the input nodes are known,

$$p\left(h \mid x\right) = \prod_{i=1}^{n} p\left(h_i \mid x\right)$$

  all visible nodes are independent when the hidden nodes are known

$$p\left(x \mid h\right) = \prod_{j=1}^{m} p\left(x_i \mid h\right)$$

- all nodes have values of 0 or 1

RBM structure

- all x and h satisfy Boltzmann distribution (Gibbs distribution):

$$p(x) \propto e^{-\frac{E}{kT}}$$

  where E is state energy, and k is the Boltzmann's constant, T is the thermodynamic temperature.
- The distribution shows that states with lower energy will always have a higher probability of being occupied than the states with higher energy (more random).
- The quantitative relationship between the probabilities of the two states being occupied

$$\frac{p_i}{p_j} = e^{-\frac{E_i - E_j}{kT}}$$

RBM encode

- Input $(x_1 \cdots x_m)$ after RBM encoded become $(h_1 \cdots h_n)$, $i-$th node in hidden layer, the probability of 1 is

$$p\left(h_i = 1 \mid x\right) = \phi\left[Wx + c\right]$$

where ϕ is a sigmoid function

- How to calculate get h_i, if $\phi\left[Wx + c\right] = a$, $a \in (0, 1)$

$$h_i = \begin{cases} 1 & a < p\left(h_i = 1 \mid x\right) \\ 0 & a \geq p\left(h_i = 1 \mid x\right) \end{cases}$$

If $\phi\left[Wx + c\right] = p\left(h_i = 1 \mid x\right) = 0.85$ means the probability of $h_i = 1$ is 0.85.
Or when x is a random number between 0 and 1, it has 85% in $(0, 0.85)$. So when the random number is in $(0, 0.85)$, i.e., $a < p\left(h_i = 1 \mid x\right)$, the event occurs, $h_i = 1$

Energy-based models

The energy of RBM is defined as

$$E\left(x,h\right)=-\sum_{i=1}^{n}\sum_{j=1}^{m}w_{ij}x_jh_i-\sum_{j=1}^{m}b_jx_j-\sum_{i=1}^{n}c_ih_i$$

w_{ij} are the weights associated with the connection between hidden unit h_i and visible unit x_j
b_j are the bias weights for the visible units x_j
c_i are the bias weights for the hidden units h_i
The energy depends on each node and connection

Energy-based models

The joint probability of visual node and the hidden node is

$$p(x, h) = \frac{e^{-E(x,h)}}{Z} \leq 1$$

where the normalizing factor Z is called the partition function

$$Z = \sum_{x,h} e^{-E(x,h)}$$

It is Boltzmann distribution.

Training error

The distance (difference) from the probability distribution of RBM $p(x)$, to the probability distribution of the input $q(x)$ is defined as

$$KL(p,q)=\sum_x q(x)\log\frac{q(x)}{p(x)}$$

WHERE $KL(p,q)$ is the the Kullback-Liebler divergence, so

$$KL(p,q)=\sum_x q(x)\log q(x)-\sum_x q(x)\log p(x)$$

the first term $\sum_x q(x)\log q(x)$ is entropy of x,

Training error

The second term cannot be obtained directly, should be estimated by Monte Carlo method, or estimated by

$$\sum_{x} q(x) \log p(x) \approx \frac{1}{l} \sum_{x \in X(l)} \log p(x)$$

where l is selected samples.
Now

$$\min KL(p,q) = \min \left[\sum_{x} q(x) \log q(x) - \sum_{x} q(x) \log p(x) \right] \rightarrow \max \sum_{x} \log p$$

If RBM have some random states (x,h), the probability in visual node arrives maximum from h to x process, i.e., the error arrive minimum

$$\frac{\partial L(\theta)}{\partial \theta} = \sum_{v} \frac{\partial \log p(v)}{\partial \theta}$$

$$\theta(k+1) = \theta(k) - \eta \frac{\partial \left[-\log p(\mathbf{x})\right]}{\partial \theta(k)}$$

where $\theta = [W, b, c]$

Training

$$\frac{\partial L(\theta)}{\partial w_{ij}} = \sum_v p\left(h_i = 1 \mid v\right) v_j - \sum_v \frac{1}{l} \sum_{k=1}^{l} p\left(h_i = 1 \mid v_{y_k}\right) v_{y_k,j}$$
$$\frac{\partial L(\theta)}{\partial b_j} = \sum_v v_j - \sum_v \frac{1}{l} \sum_{k=1}^{l} v_{y_k,j}$$
$$\frac{\partial L(\theta)}{\partial c_i} = \sum_v p\left(h_i = 1 \mid v\right) - \sum_v \frac{1}{l} \sum_{k=1}^{l} p\left(h_i = 1 \mid v_{y_k}\right)$$

- $\sum_v v_j$ and $\sum_v p\left(h_i = 1 \mid v\right)$ mean to summarize all input samples
- $\sum_v \frac{1}{l} \sum_{k=1}^{l} v_{y_k,j}$ means to summarize l output samples

Long-Term Dependencies

- RNNs can learn to use the past information, where the gap between the relevant information and the place are small.
- In theory, RNNs are absolutely capable of handling such "long-term dependencies" by picking parameters.
- In practice, RNNs don't seem to be able to learn them.
- The vanishing gradient problem for RNNs

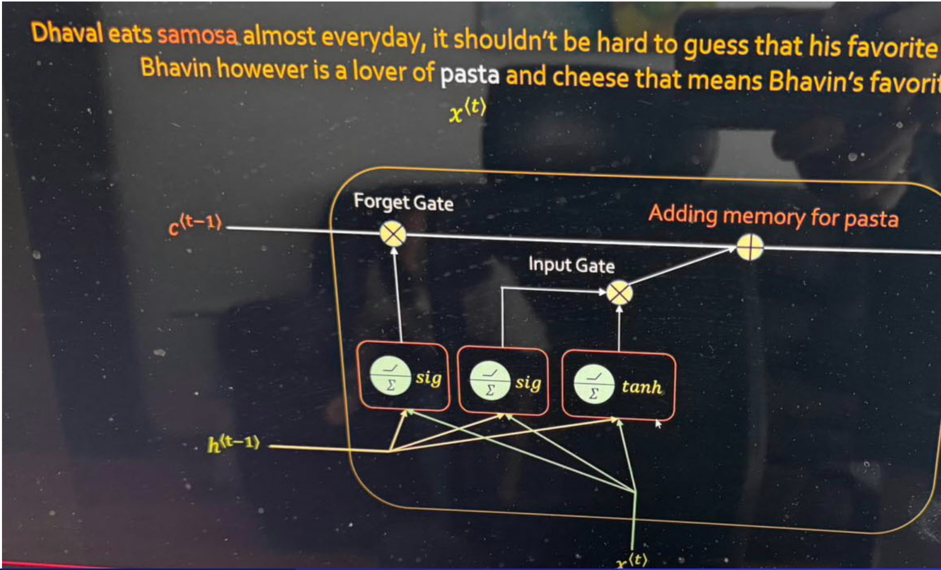
Long Short Term Memory networks

LSTM (Long Short Term Memories) is a special type of recurrent neural network structure, its hidden layer has a special structure. By introducing a gating mechanism to solve the gradient vanishing problem of RNN, it can learn long-distance dependencies.

In theory, RNNs can indeed connect long-term dependencies and solve such problems. But unfortunately in practice, RNNs cannot solve this problem. Hochreiter (1991) and Bengio, et al. (1994) have studied this problem in depth and found that RNNs are indeed difficult to solve this problem.

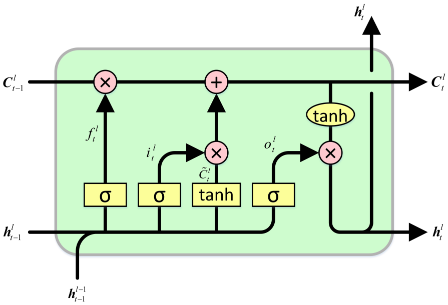
S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, Vol.9, No.8, 1997

LSTM


Dhaval eats samosa almost everyday, it shouldn't be hard to guess that his favorite
Bhavin however is a lover of pasta and cheese that means Bhavin's favorit
$x^{\langle t \rangle}$
Forget Gate
Adding memory for pasta
$c^{\langle t-1 \rangle}$
Input Gate
sig
sig
tanh
$h^{\langle t-1 \rangle}$
$x^{\langle t \rangle}$

LSTM-feedforward

Finally

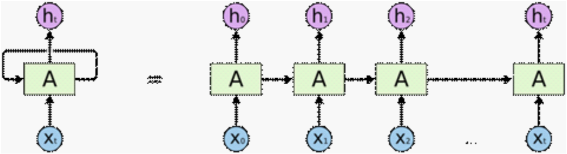


$$y_k = \sigma\left(W_{O,k}\left[y_{k-1}, u_k\right]\right)\tanh\left(x_k\right)$$

where

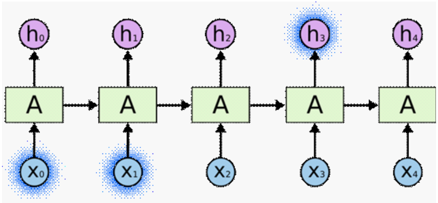
$$x_k = \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right)x_{k-1} + \sigma\left(W_{I,k}\left[y_{k-1}, u_k\right]\right)\tanh\left(W_{X,k}\left[y_{k-1}, u_k\right]\right)$$

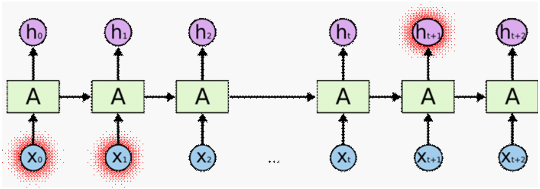
- LSTM is a recurrent neural network (RNN)
- A RNN can be thought of as multiple copies of the same network, each passing a message to a successor
- RNNs are related to sequences and lists. The architecture of RNN is suitable for such data.



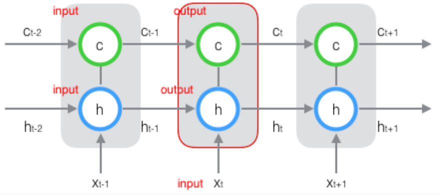
RNN

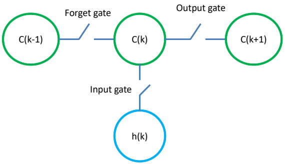
Connecting previous information to the present task depends many factors: gap length.
As the gap grows, RNNs become unable to learn to connect the information



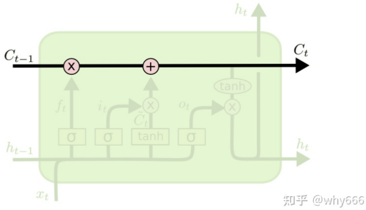


1) Cell state c (keep LongTerm Memory),while RNN only has hidden state h (keep Short Term Memory)
2) Gate is a way to let information through. LSTM has three gates to protect and control the cell state: forget gate, input gate, output gate



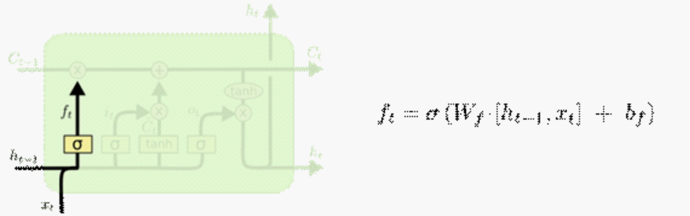


LSTM



The transfer of cell state is like a conveyor belt, with vectors passing through the entire cell, with only a few linear operations. This structure makes it easy to pass information through the entire cell without changing it. (Achieving long-term memory retention)

It looks at h_{k-1}, x_k , and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
1 represents "completely keep this" while 0 represents "completely get rid of this."

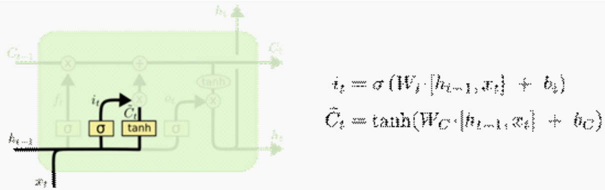


$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right]\right),$$

Input gate

For information store, to decide what new information we're going to store in the cell state. This has two parts

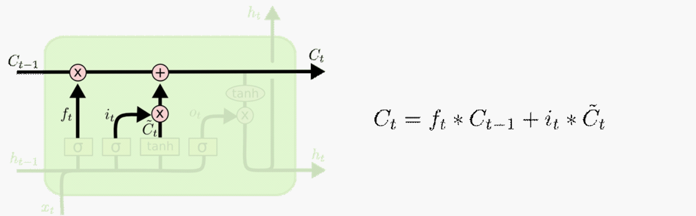
1. a sigmoid layer called the "input gate layer" decides which values we'll update.
2. a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.



$$i_k = \sigma\left(W_i\left[h_{t-1}, x_t\right]\right)$$
$$\tilde{C}_t = \tanh\left(W_c\left[h_{t-1}, x_t\right]\right)$$

Cell update

We multiply the old state C_{t-1} by f_t , forgetting the things we decided to forget earlier. Then we add $i_t \tilde{C}_t$. T



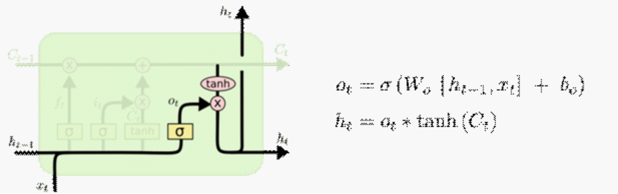
$$C_t = f_t C_{t-1} + i_t \tilde{C}_t,$$

Output gate

This output will be based on our cell state, but will be a filtered version.

1. we run a sigmoid layer which decides what parts of the cell state we're going to output.
2. we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate,

We only output the parts we decided to.



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

LSTM

$$
\begin{aligned}
f_t &= \sigma\left(W_f\left[h_{t-1}, x_t\right]\right)\\
i_k &= \sigma\left(W_i\left[h_{t-1}, x_t\right]\right)\\
\tilde{C}_t &= \tanh\left(W_c\left[h_{t-1}, x_t\right]\right)\\
C_t &= f_t C_{t-1} + i_t \tilde{C}_t\\
o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right]\right)\\
h_t &= o_t \tanh\left(C_t\right)
\end{aligned}
$$

So

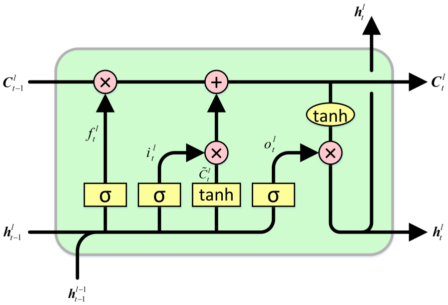
$$
h_t = \sigma\left(W_o\left[h_{t-1}, x_t\right]\right)\tanh\left(C_t\right)
$$

where

$$
\begin{aligned}
C_t &= f_t C_{t-1} + i_t \tilde{C}_t\\
&= \sigma\left(W_f\left[h_{t-1}, x_t\right]\right) C_{t-1} + \sigma\left(W_i\left[h_{t-1}, x_t\right]\right)\tanh\left(W_c\left[h_{t-1}, x_t\right]\right)
\end{aligned}
$$

LSTM

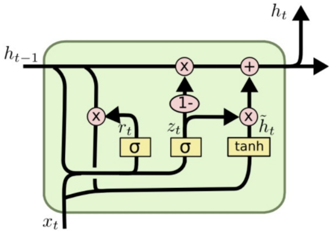
Finally



$$y_k = \sigma\left(W_{O,k}\left[y_{k-1}, u_k\right]\right) \tanh\left(x_k\right)$$

where

$$x_k = \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right) x_{k-1} + \sigma\left(W_{I,k}\left[y_{k-1}, u_k\right]\right) \tanh\left(W_{X,k}\left[y_{k-1}, u_k\right]\right)$$



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

知乎 @why666

Gated Recurrent Unit (GRU), Cho, et al. (2014).
It combines the forget gate and the input gate into a single "update gate."

GRU

1) Forget gate:

$$F_k = \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right)$$

2) Input gate:

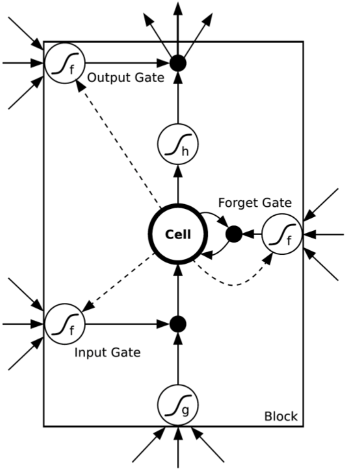
$$\begin{aligned} I_k &= \sigma\left(W_{I,k}\left[y_{k-1}, u_k\right]_I\right) \\ \tilde{x}_k &= \tanh\left(W_{X,k}\left[y_{k-1}, u_k\right]\right) \\ x_k &= F_k x_{k-1} + I_k \tilde{x}_k \end{aligned}$$

So

$$y_k = y_{k-1} + \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right)\left(\tanh\left(W_{X,k}\left[I_k y_{k-1}\right]\right) - y_{k-1}\right)$$

It merges the cell state and hidden state. GRU is simpler than standard LSTM models.

LSTM



LSTM modified

Use coupled forget and input gates. We only forget when we're going to input something in its place.
We only input new values to the state when we forget something older.

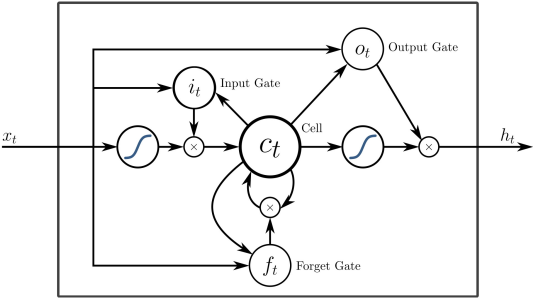
$$x_k = F_k x_{k-1} + (1 - F_k)\, \tilde{x}_k$$

LSTM-modified

Depth Gated RNNs by Yao, et al. (2015)
Clockwork RNNs by Koutnik, et al. (2014).

- Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same.
- Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs

LSTM-pinhole model

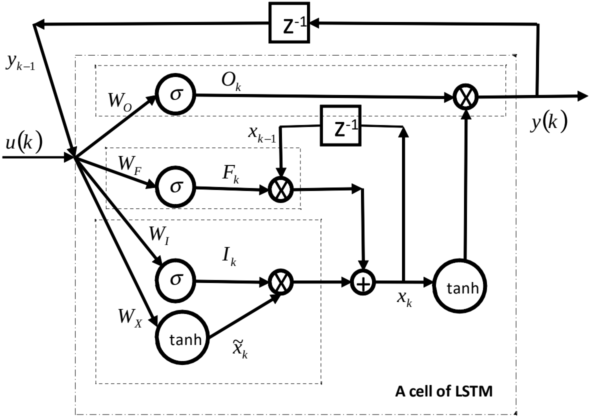


LSTM-modified

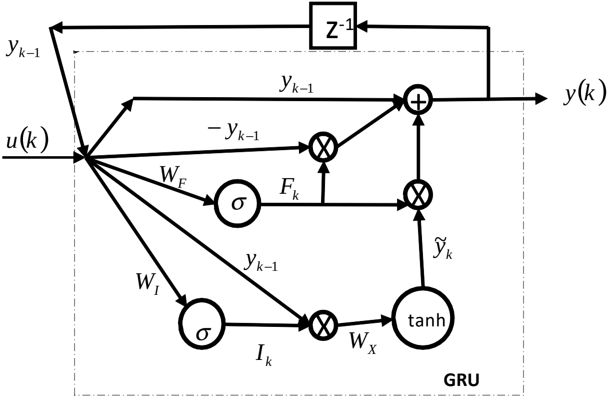
"peephole connections: the gate layers receive **cell state**

$$
\begin{aligned}
F_k &= \sigma\left(W_{F,k}\left[x_{k-1}, y_{k-1}, u_k\right] + b_f\right) \\
I_k &= \sigma\left(W_{I,k}\left[x_{k-1}, y_{k-1}, u_k\right] + b_I\right) \\
O_k &= \sigma\left(W_{O,k}\left[x_{k-1}, y_{k-1}, u_k\right] + b_O\right) \\
\tilde{x}_k &= \tanh\left(W_{X,k}\left[y_{k-1}, u_k\right] + b_x\right)
\end{aligned}
$$

LSTM in the form of system

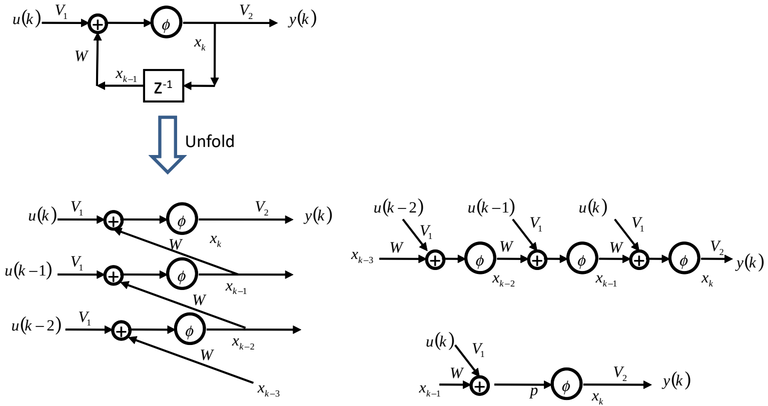


GRU



Training of RNN

Recurrent NN -> feedforward NN



Unfoled

Recurrent cell

$$
\begin{aligned}
y(k) &= V_2(k)\, x(k) \\
x(k) &= \phi\left[W(k)\, x(k-1) + V_1(k)\, u(k)\right]
\end{aligned}
$$

It can be unfoled into m steps

$$
\begin{aligned}
&\vdots \\
y(k-m) &= V_2(k)\, x(k-m) \\
x(k-m) &= \phi\left[W(k)\, x(k-m-1) + V_m(k)\, u(k-m)\right]
\end{aligned}
$$

Graident

Backpropagation Through Time (BPTT)

$y^*(k)$ is the desired output, the instantaneous of the squared output errors is

$$J(k) = \frac{1}{2}\left[y(k) - y^*(k)\right]^2 = e_o^2(k)$$

The gradient decent is

$$w(k+1) = w(k) - \eta \frac{\partial J(k)}{\partial w(k)}$$
$$\frac{\partial J}{\partial w} = \sum_{k=1}^{m} \frac{\partial J}{\partial w(k)}$$

So

$$V_2(k+1) = V_2(k) - \eta \frac{\partial J(k)}{\partial V_2(k)}$$

Because $y(k) = V_2(k)\, x(k)$

$$\frac{\partial J(k)}{\partial V_2(k)} = \frac{\partial J(k)}{\partial e_o(k)} \frac{\partial e_o(k)}{\partial y(k)} \frac{\partial y(k)}{\partial V_2(k)}$$
$$= e_o(k) \times 1 \times x(k)$$

$$V_2(k+1) = V_2(k) - \eta e_o x(k)$$

Training of input layer

So

$$V_1(k+1) = V_1(k) - \eta \frac{\partial J(k)}{\partial V_1(k)}$$

Because $y(k) = V_2(k)\,x(k)$, $p = V_1(k)\,u(k) + Wx(k-1)$

$$\frac{\partial J(k)}{\partial V_1(k)} = \frac{\partial J(k)}{\partial e_o(k)} \frac{\partial e_o(k)}{\partial y(k)} \frac{\partial y(k)}{\partial x(k)} \frac{\partial x(k)}{\partial p(k)} \frac{\partial p(k)}{\partial V_1(k)}$$
$$= e_o(k) \times 1 \times V_2(k) \times \phi' \times u(k)$$

$$V_1(k+1) = V_1(k) - \eta e_o(k)\, V_2(k)\, \phi' u(k)$$

Training of hiden ayer

$$\frac{\partial J(k)}{\partial W} = \frac{\partial J(k)}{\partial e_o(k)} \frac{\partial e_o(k)}{\partial y(k)} \frac{\partial y(k)}{\partial x(k)} \frac{\partial x(k)}{\partial p(k)} \frac{\partial p(k)}{\partial W}$$
$$= e_o(k) \times 1 \times V_2(k) \times \phi' \times x(k-1)$$

$$W(k+1) = W(k) - \eta e_o(k) V_2(k) \phi' x(k-1)$$

Training of varios inputs

Similar

$$
\begin{aligned}
e_1(k) &= e_o(k)\, V_2(k)\, \phi' \\
V_1(k+1) &= V_1(k) - \eta e_1(k)\, u(k) \\
W(k+1) &= W(k) - \eta e_1(k)\, x(k-1)
\end{aligned}
$$

$$
\begin{aligned}
e_2(k) &= e_1(k)\, W(k)\, \phi' \\
V_1(k+1) &= V_1(k) - \eta e_2(k)\, u(k-1) \\
W(k+1) &= W(k) - \eta e_2(k)\, x(k-2)
\end{aligned}
$$

Finally

$$V_1\left(k+1\right)=V_1\left(k\right)-\eta e_1\left(k\right)u\left(k\right)$$
$$V_1\left(k\right)=V_1\left(k-1\right)-\eta e_2\left(k\right)u\left(k-1\right)$$
$$V_1\left(k-1\right)=V_1\left(k-2\right)-\eta e_3\left(k\right)u\left(k-2\right)$$

$$e_1\left(k\right)=e_o\left(k\right)V_2\left(k\right)\phi'$$
$$e_2\left(k\right)=e_1\left(k\right)W\left(k\right)\phi'$$
$$e_3\left(k\right)=e_2\left(k\right)W\left(k\right)\phi'$$

$$W\left(k+1\right)=W\left(k\right)-\eta e_1\left(k\right)x\left(k-1\right)$$
$$W\left(k\right)=W\left(k-1\right)-\eta e_2\left(k\right)x\left(k-2\right)$$
$$W\left(k-1\right)=W\left(k-2\right)-\eta e_3\left(k\right)x\left(k-3\right)$$

Backpropagation Through Time (BPTT)

So

$$W(k+1) = W(k) - \eta \sum_{i=1}^{M} e_i(k)\, x(k-i)$$
$$V_1(k+1) = V_1(k) - \eta \sum_{i=1}^{M} e_i(k)\, u(k-i+1)$$
$$V_2(k+1) = V_2(k) - \eta e_o x(k)$$

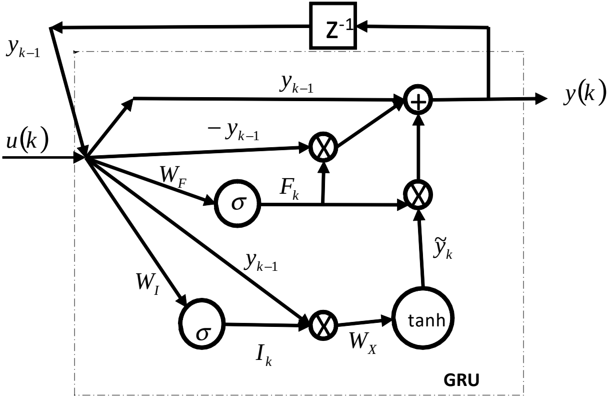
where

$$e_i(k) = e_{i-1}(k)\, W(k)\, \phi'$$

If $\|W(k)\phi'\| > 1$, it is gardient blow up. If $\|W(k)\phi'\| < 1$, it is gardient vanish. We need to add a constraint as

$$\text{when } \left\|W(k)\phi'\right\| \approx 1, \text{ the layer is actived}$$

GRU


z^{-1}
y_{k-1}
y_{k-1}
$y(k)$
$u(k)$
$-y_{k-1}$
W_F
σ
F_k
$\tilde{y}_k$
y_{k-1}
W_I
tanh
σ
W_X
I_k
GRU

GRU Training

Recurrent part

$$y_k = y_{k-1} - F_k y_{k-1} + F_k \tilde{y}_k$$

Feedforward part

$$\begin{aligned} F_k &= \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right) \\ \tilde{y}_k &= \phi\left(W_{X,k}\left[I_k y_{k-1}, u_k\right]\right) \\ I_k &= \sigma\left(W_{I,k}\left[y_{k-1}, u_k\right]\right) \end{aligned}$$

In state space

$$\begin{aligned} x_k = x_{k-1} - \sigma\left(W_{F,k}\left[x_{k-1}, u_k\right]\right) x_{k-1} \\ +\sigma\left(W_{F,k}\left[x_{k-1}, u_k\right]\right) \phi\left(W_{X,k}\left[\sigma\left(W_{I,k}\left[x_{k-1}, u_k\right]\right) x_{k-1}\right]\right) \end{aligned}$$

Training-recurrent NN

$$J(k) = \frac{1}{2}\left[y(k) - d(k)\right]^2 = e_i^2(k)$$
$$W(k+1) = W(k) - \eta \frac{\partial J(k)}{\partial W(k)}$$

If

$$y(k) = \phi\left[W(k)\, y(k-1) + V_1(k)\, u(k)\right]$$

BPTT is

$$W(k+1) = W(k) - \eta \sum_{i=1}^{M} e_i(k)\, y(k-i)$$
$$V_1(k+1) = V_1(k) - \eta \sum_{i=1}^{M} e_i(k)\, u(k-i+1)$$
$$e_i(k) = e_{i-1}(k)\, W(k)\, \phi'$$

Training-GRU

For $W_{F,k}$, because

$$y_k = y_{k-1} - \sigma\left(W_{F,k}\left[y_{k-1}, u_k\right]\right) y_{k-1} + F_k \tilde{y}_k$$

BPTT

$$\begin{array}{l} W_F\left(k+1\right) = W_F\left(k\right) - \eta \sum_{i=1}^{M} e_F\left(k\right)\left[y_{k-i}, u_{k-i}\right] \\ e_F\left(k\right) = e_{F-1}\left(k\right) W_F\left(k\right) \sigma' y_{k-i} \end{array}$$

or

$$W_F\left(k+1\right) = W_F\left(k\right) - \eta \sigma' \sum_{i=1}^{M} e_{F-1}\left(k\right) W_F\left(k\right)\left[y_{k-i}, u_{k-i}\right]$$

Training-GRU

For W_X and W_I,

$$y_k = y_{k-1} - F_k y_{k-1} + F_k \phi \left(W_{X,k} \left[\sigma \left(W_{I,k} \left[x_{k-1}, u_k \right] \right) x_{k-1} \right] \right)$$

BPTT

$$\begin{array}{l} W_X \left(k+1 \right) = W_X \left(k \right) - \eta \sum_{i=1}^{M} e_X \left(k \right) \left[I_k y_{k-1}, u_k \right] \\ e_X \left(k \right) = e_{X-1} \left(k \right) W_X \left(k \right) \phi' \sigma \left(W_{F,k} \left[y_{k-1}, u_k \right] \right) \\ W_I \left(k+1 \right) = W_I \left(k \right) - \eta \sum_{i=1}^{M} e_I \left(k \right) y_{k-1} \\ e_I \left(k \right) = e_{I-1} \left(k \right) W_X \left(k \right) \phi' \sigma' \sigma W_I \left(W_{F,k} \left[y_{k-1}, u_k \right] \right) \end{array}$$

LSTM for nonlinear system identification

$$y(k) = \Phi\left[y\left(k-1\right), \cdots, y\left(k-n_y\right), u\left(k\right), \cdots, u\left(k-n_u\right)\right] \tag{1}$$

Simulation model,

$$\hat{y}(k) = F\left[u\left(k\right), \cdots, u\left(k-n\right)\right] \tag{2}$$

$$\hat{y}(k) = N[\hat{y}(k-1), \cdots, \hat{y}(k-m), u\left(k\right), \cdots, u\left(k-n\right)] \tag{3}$$

It is the parallel identification model.

Structure

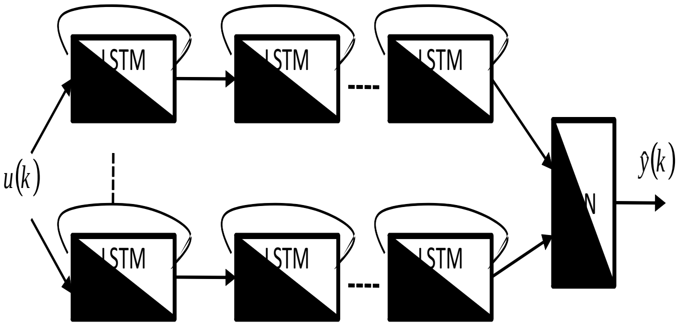
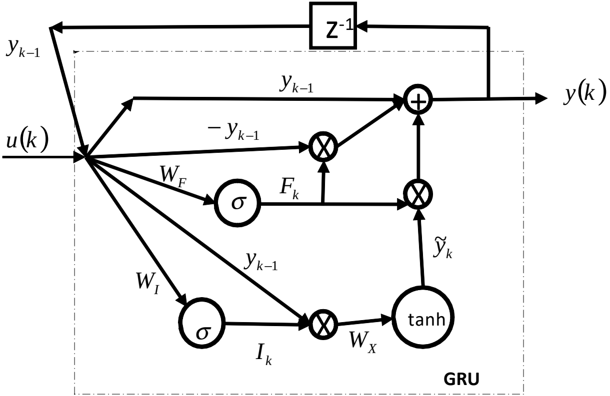


Figure: LSTM for nonlinear system modeling

GRU



z^{-1}
y_{k-1}
y_{k-1}
$y(k)$
$u(k)$
$-y_{k-1}$
W_F
σ
F_k
$\tilde{y}_k$
y_{k-1}
W_I
σ
tanh
I_k
W_X
GRU

Structure

$$e(k) = \hat{y}(k) - y(k)$$

The dynamic of the GRU is

$$x_{k+1} = x_k - \sigma(W_1 x_k + W_u u_k) x_k + \sigma(W_1 x_k + W_u u_k) \phi(W_2 [\sigma(W_3 x_k) x_k])$$

The GRU can be transformed into two parts: RNN and FNN (feedforward NN)

$$\begin{array}{ll} \text{RNN:} & x(k+1) = Ax(k) - \sigma[W_1 x(k) + W_u u(k)] \tilde{u}(k) \\ \text{FNN:} & \tilde{u}(k) = \phi(W_2 \sigma[W_3 x(k)] x(k)) + x(k) \end{array} \quad (4)$$

Structure

Theorem

For the feedforward part of GRU, FNN, the following backpropagation-like algorithm can make identification error $e_F(k)$ bounded

$$\begin{array}{l} W_{2,k+1} = W_{2,k} - \eta_k e_F(k) \phi' x^2(k) \sigma \\ W_{3,k+1} = W_{3,k} - \eta_k e_F(k) \phi' \sigma' W_{2,k} x^2(k) \end{array} \quad (5)$$

where $\eta_k = \dfrac{\eta}{1 + \left\|\phi' x^2(k) \sigma\right\|^2 + \left\|\phi' \sigma' W_{2,k} x^2(k)\right\|^2}$. *The average of the identification error satisfies*

$$J = \limsup_{T\to\infty} \frac{1}{T} \sum_{k=1}^{T} e_F^2(k) \leq \frac{\eta}{\pi} \bar{\delta}_1 \quad (6)$$

where $\pi = \dfrac{\eta}{1+\kappa}\left[1 - \dfrac{\kappa}{1+\kappa}\right] > 0,$

$\kappa = \max_k \left(\left\|\phi' x^2(k) \sigma\right\|^2 + \left\|\phi' \sigma' W_{2,k} x^2(k)\right\|^2\right), \ \bar{\delta}_1 = \max_k \left[\delta_1^2(k)\right],$

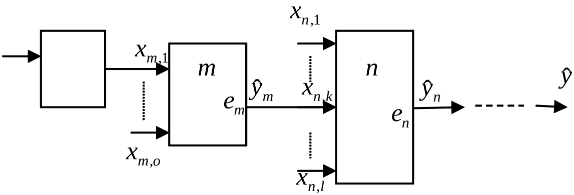


Figure: Error backpropagation

The performance index is defined as

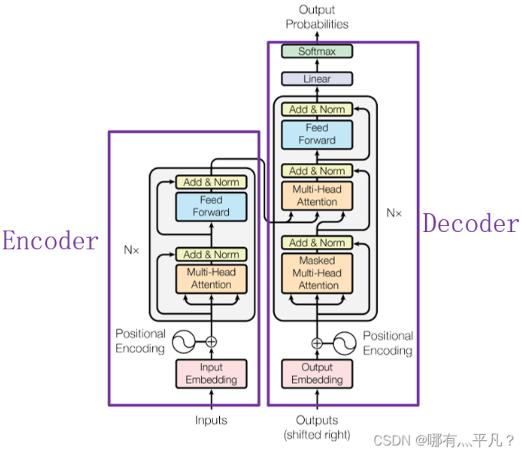
$$J = \frac{1}{2} e_0^2 \quad e_o = \hat{y} - y, \quad \hat{y} = \phi \left[W_{N,k} \hat{y}_i \right]$$

$$e_m = \frac{\partial \sigma}{\partial t} w_u e_n \tag{7}$$

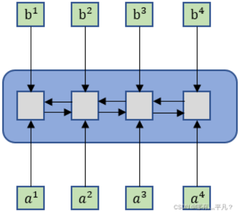
the training law is the gradient descent

$$W_N \left(k+1 \right) = W_N \left(k \right) - \eta \hat{y}_{i,q} \left(k \right) e_o \left(k \right) \tag{8}$$

Seq2seq


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Decoder
Encoder
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Positional
Encoding
Output
Embedding
Inputs
Outputs
(shifted right)
CSDN @哪有,,,平凡？

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia PolosukhinAttention Is All You Need, *Neural Information Processing Systems*, 2017

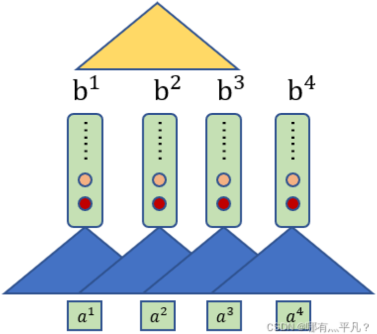


The problem of RNN

Input series $\{a_1, a_2, a_3, a_4\}$, the output series $\{b_1, b_2, b_3, b_4\}$. If we want b_3 we need to input $\{a_1, a_2, a_3\}$, we cannot obtain all b_1, b_2, b_3, b_4 at the same time.

Parallel

One solution is to us filters (CNN)



Parallel serires $\{b_1, b_2, b_3, b_4\}$ with $\{a_1, a_2\}$, $\{a_1, a_2, a_3\}$, $\{a_2, a_3, a_4\}$, $\{a_3, a_4\}$

But we lost long term memory

Self attention

Each input x_i is mutiplied by W, we have (q, k, v)

$$q_i = W^q a_i, \quad k_i = W^k a_i, \quad v_i = W^v a_i$$

where

$$a_i = W x_i$$

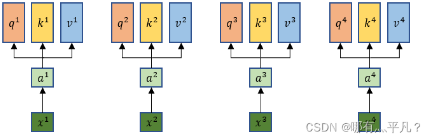
then we calculate dot product

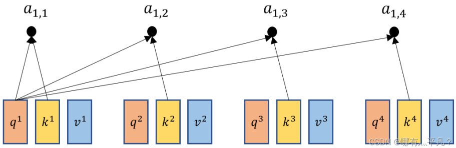
$$a_{1,i} = \frac{1}{\sqrt{d}} q_1 k_i$$

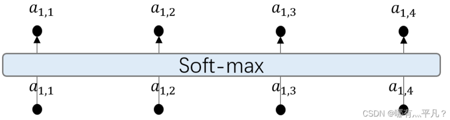
then we allpy soft-max

$$\hat{a}_{1,i} = \frac{e^{a_{1,i}}}{\sum_j e^{a_{1,j}}}$$

Self attention





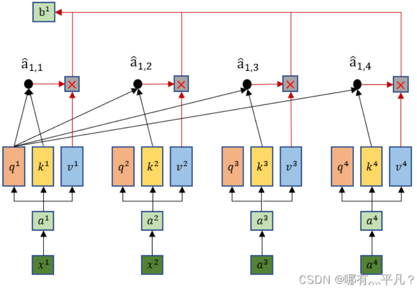


Self attention

$\hat{a}_{1,i}$ is mutiplied by v_i

$$b_1 = \sum_i \hat{a}_{1,i} v_i$$

so b_1 have all information of x_i



Also: if b_1 only pay attention on $\{x_1, x_2\}$, then we can set $\hat{a}_{1,3} = \hat{a}_{1,4} = 0$

Parallel

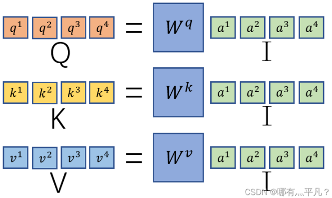
If we use vector form for,

$$q_i = W^q a_i, \quad k_i = W^k a_i, \quad v_i = W^v a_i, a_i = Wx_i$$

$$Q = [q_1, \cdots q_4], K = [k_1, \cdots k_4], V = [v_1, \cdots v_4]$$

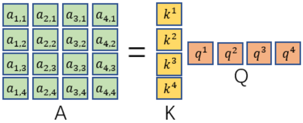
The input is X, the output

$$A = XW, Q = W^q A, K = W^k A, V = W^v A$$

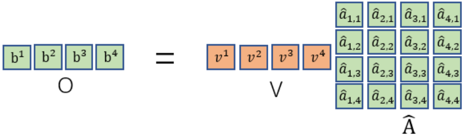


Parallel

$$\hat{A} = KQ, \quad \hat{A} = soft\max\left(\hat{A}\right)$$

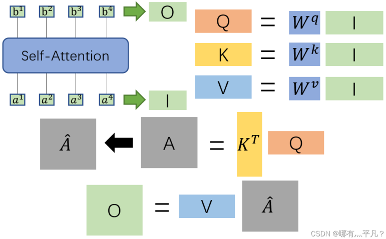


$$B = V\hat{A},$$

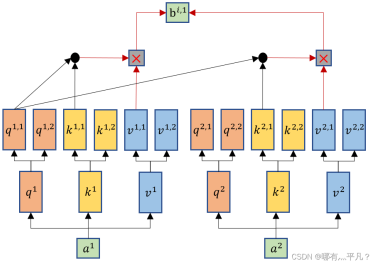


So

$$O = V\hat{A},$$


b1
b2
b3
b4
O
Self-Attention
a1
a2
a3
a4
I
Q
=
W^q
I
K
=
W^k
I
V
=
W^v
I
$\hat{A}$
A
=
K^T
Q
O
=
V
$\hat{A}$
CSDN @哪有,,,平凡？

(q, k, v) is divided into several, then each of them are applied to self attention



Positional Encoding

Positions information: each element should has its own position information e_i