No.

Date

Why GBDT ?

The new tree corrects the error of previous trees. but not fully.

$$l(y_i, \hat{y}_i^{k-1} + f_k(x_i)) \simeq l(y_i, \hat{y}_i^{k-1}) + g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i)$$

previous loss correction of the new tree.

only find local min in every tree

decrease the loss.

$$\hat{y}_i^{\,1} = \hat{y}_i^{\,0} + \text{something}$$

$$\hat{y}_i^{\,1} = \hat{y}_i^{\,0} + \eta \frac{\partial L(y_i, \hat{y}_i^{\,0})}{\partial \hat{y}_i^{\,0}}$$

we fit a new model to generate g_i

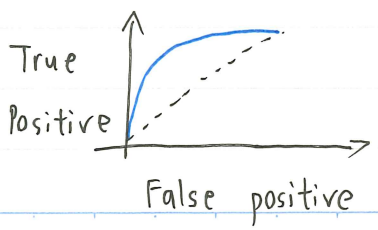
$$\therefore \hat{y}_i^{\,k} = \hat{y}_i^{\,0} + f_1(x) + f_2(x) + \dots + f_K(x)$$

$$w_j^k = \frac{-G_j}{H_j + \lambda}$$

It's essentially the average of g_i's inside every leaf

Second order term can be treated as a scaling factor that impact how far we go in the gradient direction based on curvature

ROC curve (receiver operating characteristic)



A good classifier stays as far away from diagonal line as possible

No.

Date

Deep Neural Network (DNN)

$$\frac{\partial L}{\partial w_{ij}^{[2]}} = \frac{\partial L}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w_{ij}^{[2]}}$$

$(a^{[3]} - y)$ $W^{[3]}$ $\text{diag}(g'(z^{[2]}))$ $a_j^{[1]} e_i$

1×1 1×2 2×2 2×1

$= (a^{[3]} - y)\, W^{[3]} \circ g'(z^{[2]})\, a_j^{[1]} e_i$

$= [(a^{[3]} - y)\, W^{[3]} \circ g'(z^{[2]})]_i a_j^{[1]}$

hadamard product (elementwise product)

In matrix form:

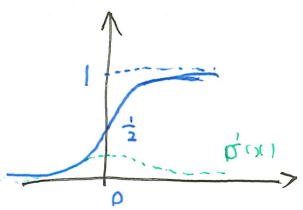
$$f(x) = \sigma^{[L]}(W^{[L]}\sigma^{[L-1]}(W^{[L-1]}\sigma^{[L-2]}(W^{[L-2]}\dots\sigma^{[1]}(W^{[1]}x + b^{[1]})\dots + b^{[L-2]}) + b^{[L-1]}) + b^{[L]})$$

a linear transformation followed by a non-linear transformation

if activation function is linear, then $f(x) = W'x + b$ is a linear model

Activation Functions

sigmoid



✓ maps to probability

✓ Logistic regression

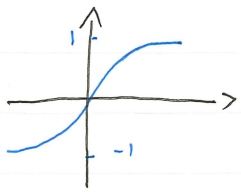
- not zero centered — bias
- saturate locally at extremes (grad = 0), no flow and vanishing gradients
- Verdict: don't use in non-output layers

No.

Date

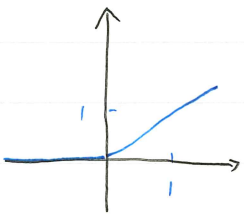
Tangens hyperbolics (tanh)

- ✗ Maps to probability
- ✗ logistic regression
- ✗ saturates (grad = 0), no flow
- ✓ zero-centered



Verdict · can use in non-output layers, but not recommended

ReLU



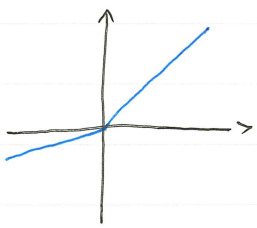
- · non-linear due to clipping
- · no expensive functions
- · doesn't saturate on right side
- ✗ not zero-centered
- ✗ neurons can get stuck · requires increased tuning of learning rate

Verdict: default

Leaky ReLU:

$g(z) = \max(\varepsilon z, z)$

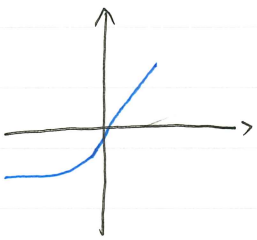
with $\varepsilon \ll 1$



ELU

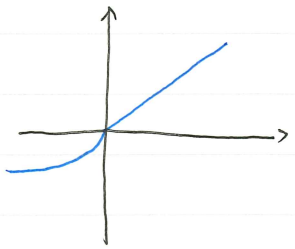
$g(z) = \max(\alpha(e^z - 1), z)$

with $\alpha \ll 1$



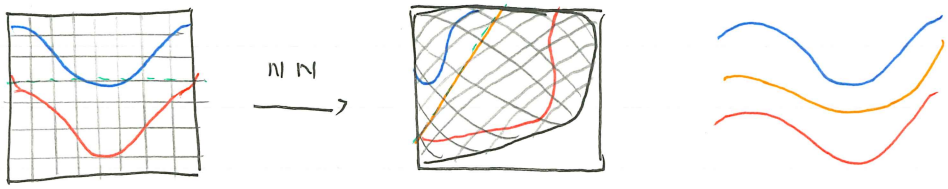
SELU

$$\begin{cases} f(x) = \lambda x & x > 0 \\ f(x) = \lambda \alpha (e^x - 1) & x \leq 0 \end{cases}$$



No.

Date



Universality Theorem

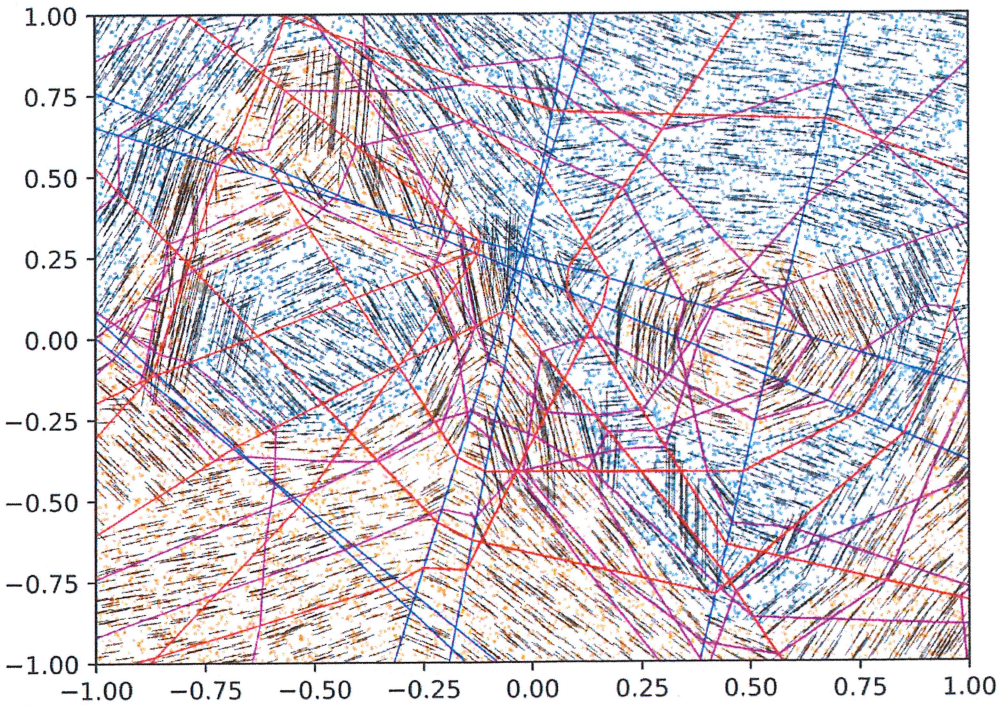
Any continuous function f

$f : R^N \rightarrow R^M$

can be realized by a network with one hidden layer

given enough hidden neurons

ReU DNNs Partitions Input Space 3 layers x 8 neurons



No.

Date

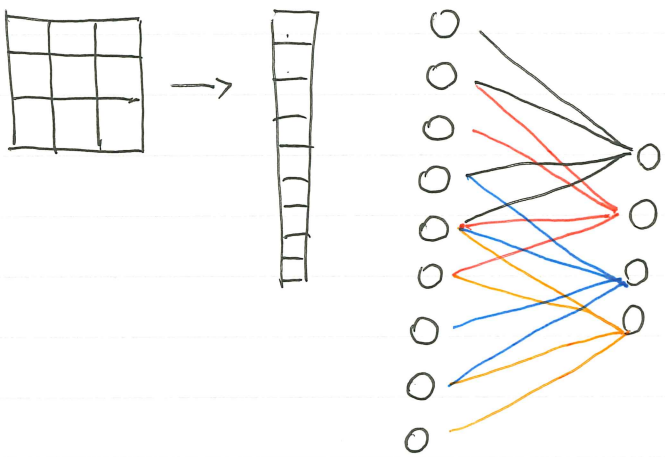
Blue: Linear regions (convex polyhedrons) generated by 1st layer weights

Red: 2nd

Purple: 3rd

Convolutional Layer:

CNN is a sparsely connected DNN.



2x2 Filter

output size $\frac{N+2P-F}{\text{stride}} + 1$

Image → CONV → Max Pooling → Flatten → FC

(CONV → Max Pooling) × N

In CNN: · first layers usually acts as edge detectors

· deeper feature maps encode high level features

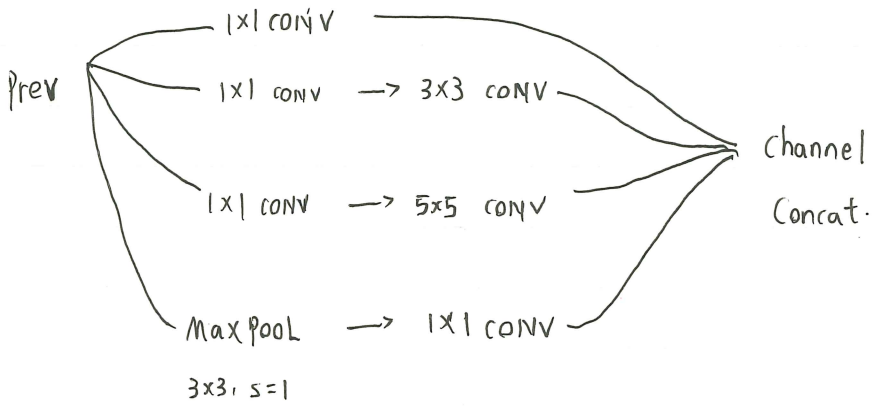
No.

Date

VGG Net

Smaller filters are
- Cheapers.
- Stack of 3×3 filters ×3 has same receptive field as 7×7

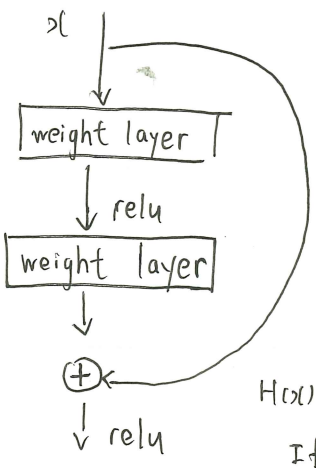
Inception Net.



ResNet - Motivation

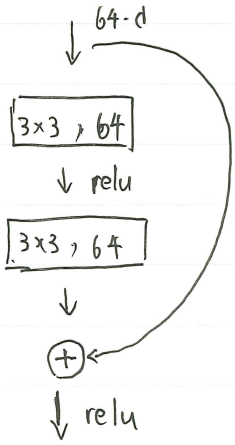
Degradation problem: adding more layers to a suitably deep model → higher training error

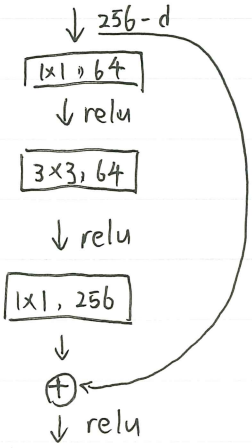
Optimization Difficulties: Optimizers can't find the solution when going deeper



$H(x) = F(x) + x$

If identity were optimal, easy to set weights as 0





all 3×3 ← similar complexity → bottleneck for Res 50/101/152

Useful tricks

· LR scheduler : (multi)stepLR, cosine AnnealLR

Dropout: During training, randomly remove neurons, by a factor $\frac{1}{1-p}$

During inference ① use all neurons

② still dropout, but do ensembling over multiple runs

prevent co-adaptatiton of neurons, (similar to random forest)

Data Augmentation

- Mixup

Batchnorm

Address covariance shift: Layers' scales change based on previous layers.

Features →

$$\mu_i = \frac{1}{m}\sum A_i \qquad \sigma_i = \sqrt{\frac{1}{m}\sum (A_i-\mu)^2}$$

$$\rightarrow \hat{A}_i = \frac{A_i-\mu_i}{\sigma_i} \rightarrow \tilde{BN}_i = \gamma \odot \hat{A}_i + \beta \rightarrow$$

Moving Average

$$\mu_{movi} = \alpha\,\mu_{movi} + (1-\alpha)\,\mu_i$$

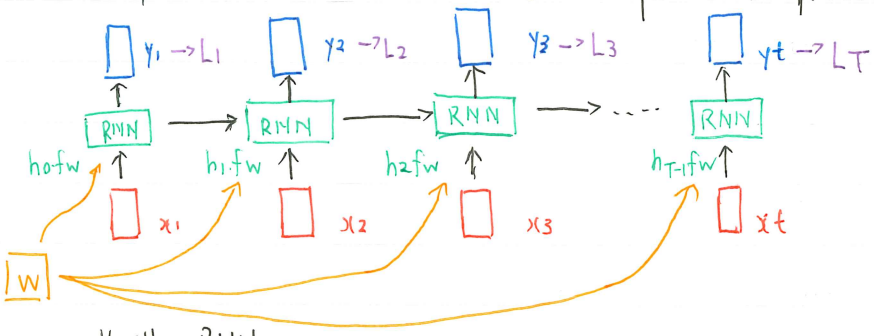
$$\sigma_{movi} = \alpha\,\sigma_{movi} + (1-\alpha)\,\sigma_i$$

No.

Date

Recurrent Neural Network

update an internal state as a sequence is processed



Vanilla RNN:

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single vector

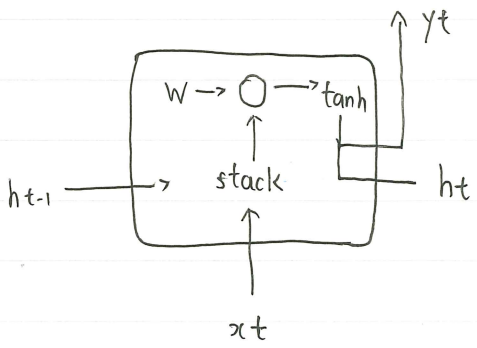
sequence 2 sequence

Advantage:

- Can process any length input.
- Computation for step t can use information from many steps back
- Model size doesn't increase for larger input.

Disadvantages:

- Recurrent computation is slow.



Backpropagation through time

$$\frac{\partial J_T}{\partial w} = \sum_{i=1}^{T} \frac{\partial J_T}{\partial w}\Big|_{(i)}$$

$$\because h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$\therefore \frac{\partial h_t}{\partial h_{t-1}} = \mathrm{diag}[\tanh'(W_{hh} h_{t-1} + W_{xh} x_t)] W_{hh}$$

$$\therefore \frac{\partial J_T}{\partial h_t} = \frac{\partial J_T}{\partial h_T} W_{hh}^{T-t} \prod_{t<i\le T} \mathrm{diag}[\tanh'(W_{hh} h_{i-1} + W_{xh} x_{(i)})]$$

To prevent gradient explosion: Gradient Clipping

Before SGD update scale down gradient if its norm is greater than some threshold

if $\|g\| \ge$ threshold then

$$g \leftarrow \frac{\text{threshold}}{\|g\|} g.$$

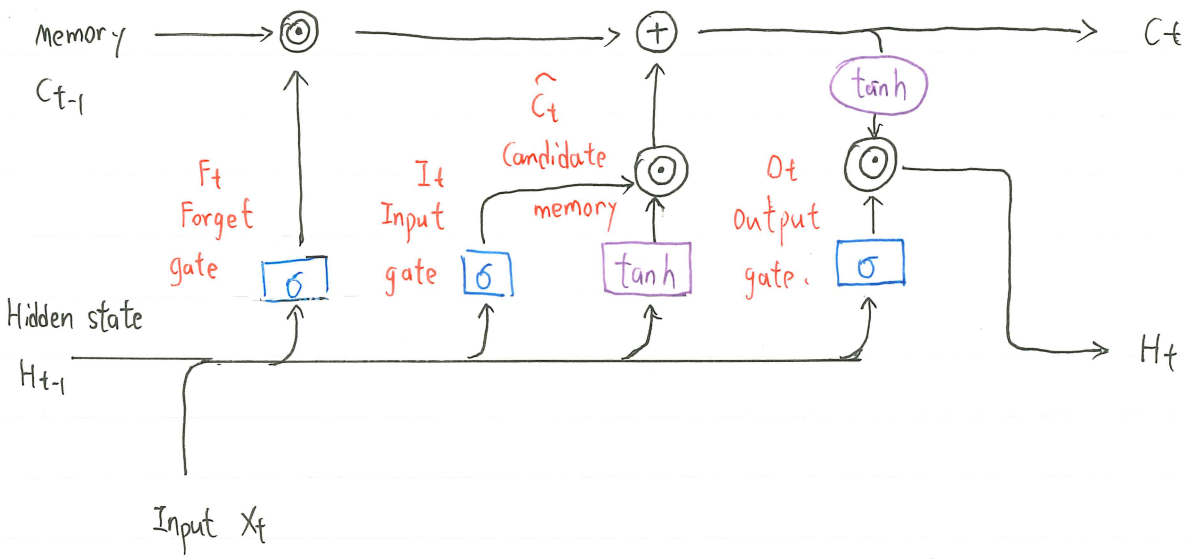
Simple RNNs have bad long-term memory

Solution: LSTM. GRU (gated recurrent unit)

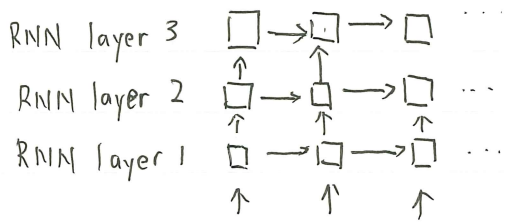
No.

Date

Long short-term memory (LSTM)



Muti-layer RNNs



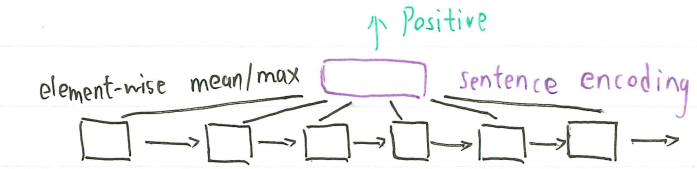
High-performing RNNs are often multi layer (not as deep as CNN)

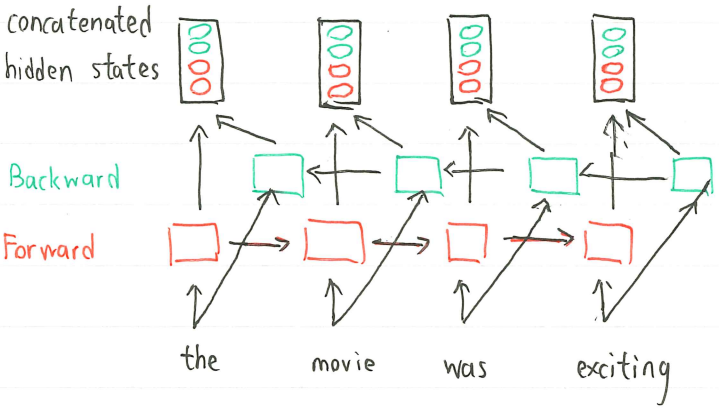
However, this only contain information about the left context

No.

Date

Bidirectional RNN :





Forward RNN $\overrightarrow{h_t} = RNN_{FW}(\overrightarrow{h_{t-1}}, x_t)$

$\overleftarrow{h_t} = RNN_{BW}(\overleftarrow{h_{t-1}}, x_t)$

$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$

Encoder —hidden state→ Decoder

Bottleneck Problem —— solved by attention

No.

Date

Attention:

score (h_t, s_k)

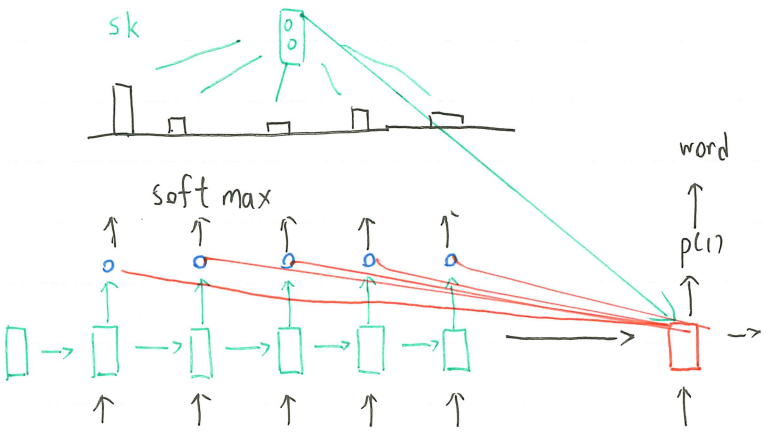
scalar ↑ out

Attention function:

How relevant is source token k for target step t ?

Decoder state at step t: h_t

Encoder state for token k: s_k



word embeddings Input output

Source sentence

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \quad (k = 1 \dots m)$$

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \dots + a_m^{(t)} s_m = \sum_{i=1}^{m} a_k^{(t)} s_k$$

Source context for decoder step t

No.

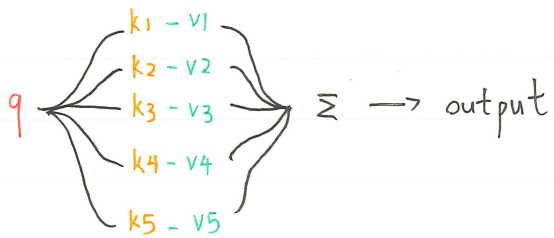
Date

Finally, concatenate with decoder state and pass on to output layer

$$\tilde{h}_t = \tanh(W_c [c_t ; h_t^{dec}]) \in \mathbb{R}^h \quad , W_c \in \mathbb{R}^{2h \cdot h}$$

Intuition: The query matches to all keys softly to a weight $\in [0,1]$

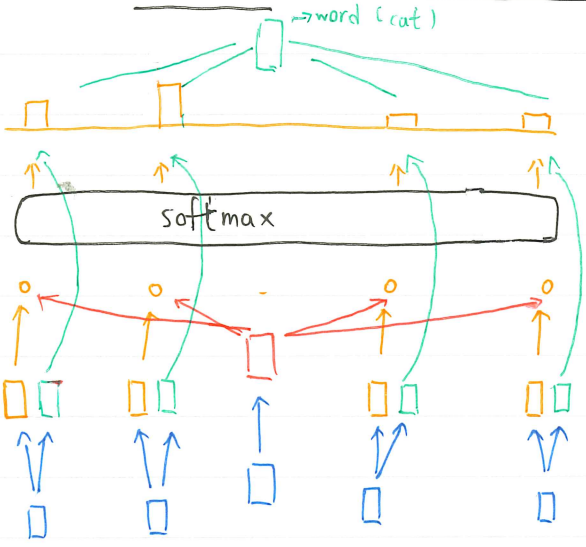
The key's value are multipled by weights and summed



Self attention:

$$\text{Attention}(q, k, V) = \overbrace{\text{softmax}\left(\frac{q k^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} V$$

From each state to All other tokens in the sequence



word.

KOKUYO

No.

Date

Recommendation Systems

A set of tuples (u, i, r, t), r_{ui}^{t}

For a linear model.

$$\text{rating}(\text{user}, \text{movie}) = f(\phi_u(u), \theta_u) + f(\phi_i(i), \theta_i)$$

To recommend a movie to a user

$\underset{i \in \text{candidates}}{\text{argmax}} : f(u) + \boxed{f(i)}$ not personalized

The key is to model interactions between users and items