# AI/ML Applications on Gadi
## - Natural Language Processing

NCI Training

Zhuochen Wu

# Outline

Lectures

- Introduction to Machine Learning and Deep Learning
- Text Processing
- Recurrent Neural Networks
- Transformers
- Topic Modeling

# NLP – Natural Language Processing

- The techniques for computer software to classify, understand and generate human language.
- Applications:
  - Machine translation(Google Translate)
  - Natural language generation
  - Information retrieval
  - Spam filters
  - Sentiment Analysis
  - Chatbots
  - Linguistic analysation
  - Social science analysation

# Machine Learning and Deep Learning

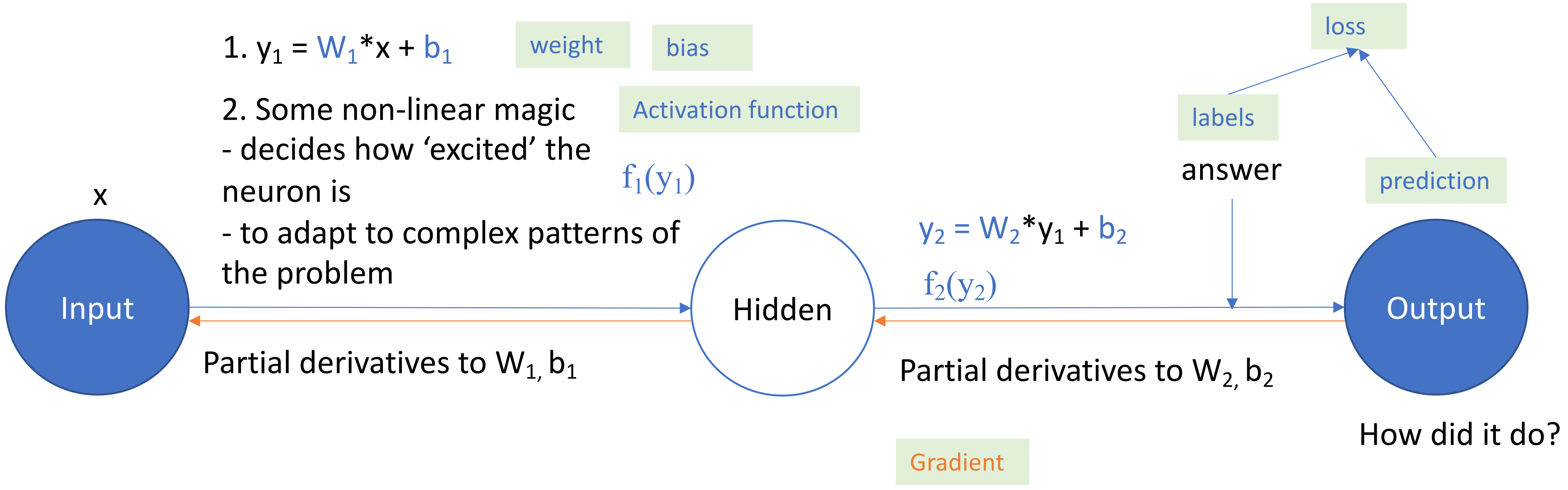
- Machine learning
- Deep learning
  - Neural networks
  - Matrix
  - Loss function
  - Gradient and backpropagation
  - SGD and learning rate
  - Example code

# Machine learning

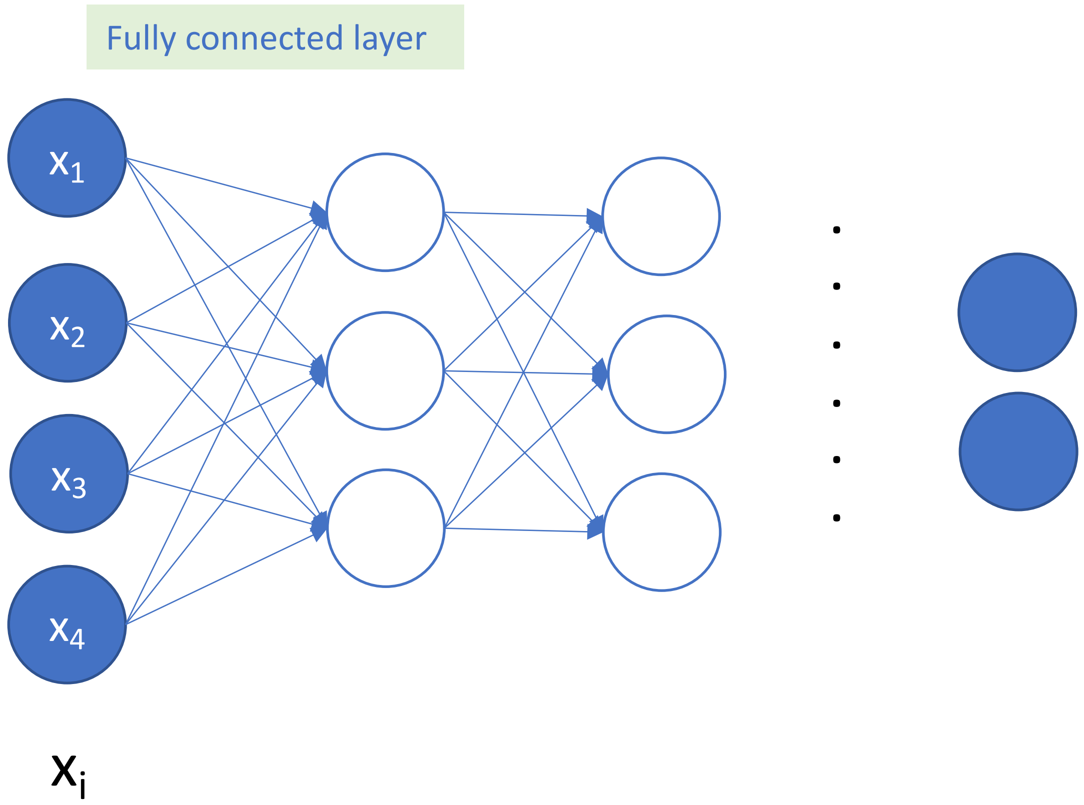
| | | | |
|---|---|---|---|
| Learning methods | Supervised | Unsupervised | Reinforced |
| Components | Dataset | Algorithm | Features |
| Examples | Numbers, text, images, speech… | Linear regression, Decision Tree, SVM…… | Attributes in data |

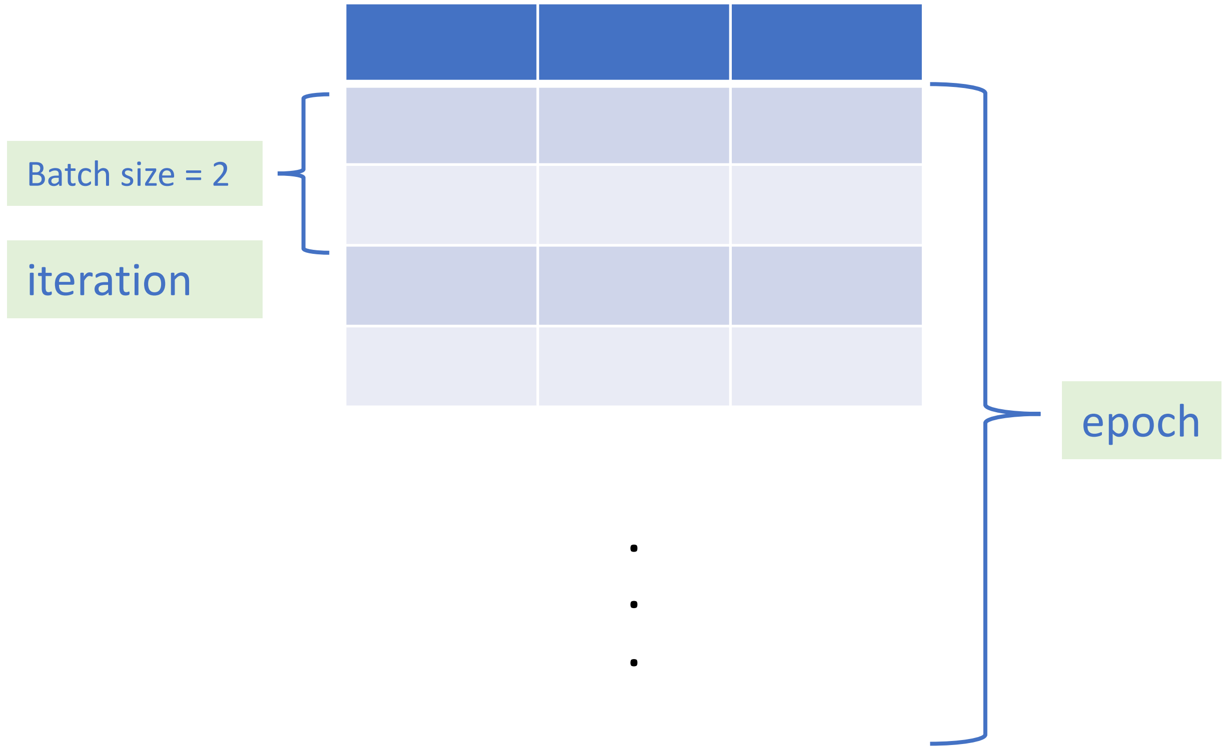
# Deep learning – Neural Networks

## Given input x, to predict the answer



# Now repeat it... And more data!



NCI AUSTRALIA

# Matrix

- Element-wise multiplication

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \quad \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = \begin{bmatrix}5 & 12\\ 21 & 32\end{bmatrix}$$

**y = tensor * tensor**

- Inner product

$1\times5 + 2\times7 = 19$

$$\begin{bmatrix}\boxed{1, 2}\\ 3, 4\end{bmatrix} \quad \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = \begin{bmatrix}19 & 22\\ 43 & 50\end{bmatrix}$$

**tensor2 = tensor.matmul(tensor1)**

- Matrix shape

$$A_{nxm} \quad B_{mxh} = C_{nxh}$$
$$C_{nxh} \rightarrow C^T{}_{hxn}$$

x $[x_{11}, x_{12}, x_{13}, \dots x_{1n}]$
$[x_{21}, x_{22}, x_{23}, \dots x_{2n}]$
……
$[x_{i1}, x_{i2}, x_{i3}, \dots x_{in}]$

Input feature dimension = n

w $[w_{11}, w_{12}, w_{13}, \dots w_{1i}]$
$[w_{21}, w_{22}, w_{23}, \dots w_{2i}]$
……
$[w_{m1}, w_{n2}, w_{n3}, \dots w_{ni}]$

Output number = m

b $[b_1, b_2, b_3, \dots b_n]$

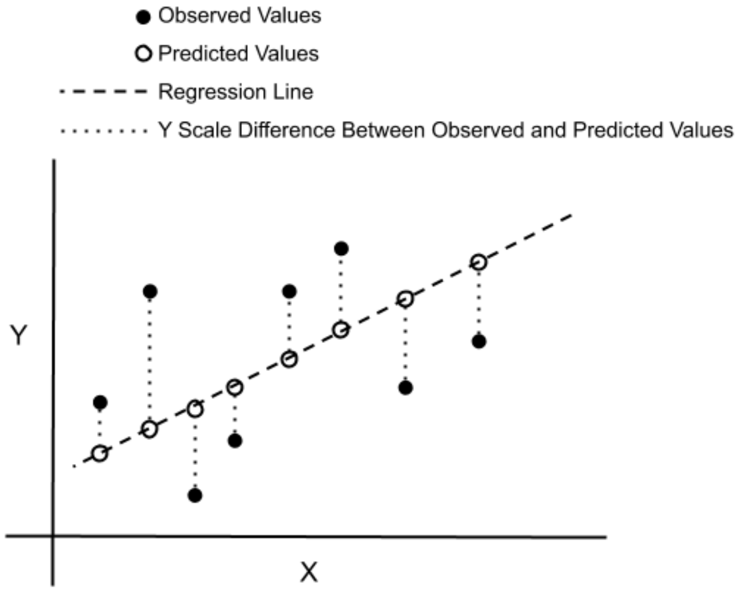
Broadcasting to mxn

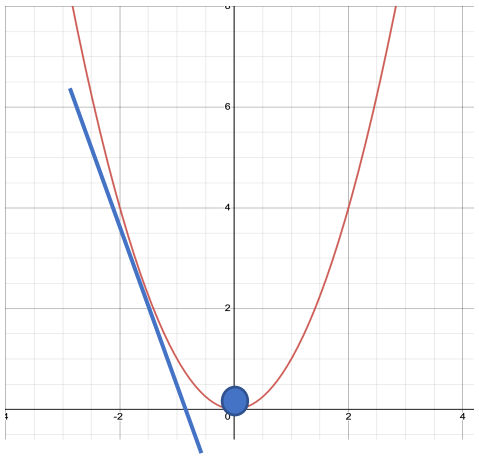
# Loss function

- Example: Mean Square Error

$$L(\mathbf{w},b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w},b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\overbrace{\mathbf{w}^\top \mathbf{x}^{(i)} + b}^{\text{Predicted value}} - y^{(i)}\right)^2$$

Training goal: $$\mathbf{w}^*, b^* = \operatorname*{argmin}_{\mathbf{w},b} L(\mathbf{w},b).$$



$Y = x^2$

$Y' = 2x$

Derivative -> value change direction

# Gradient and Backpropagation

Partial derivatives $\frac{\partial L(w,b)}{\partial w}, \quad \frac{\partial L(w,b)}{\partial b}$

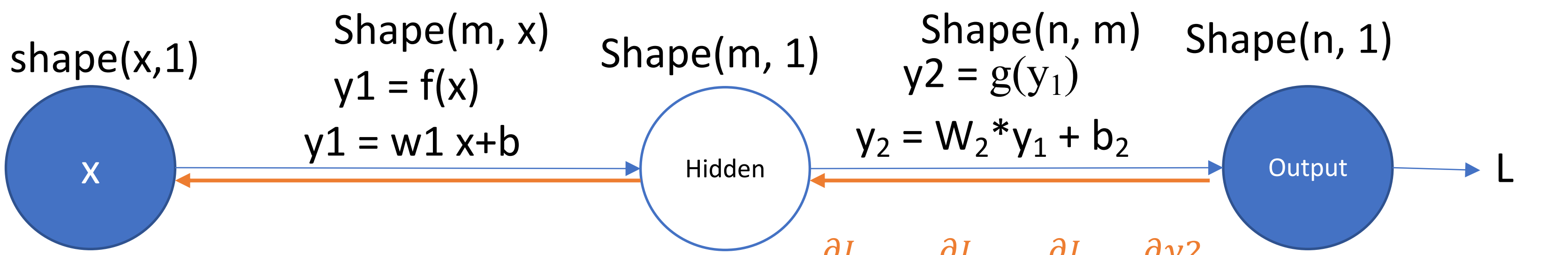
Shape: $\text{shape}(\frac{\partial z}{\partial x}) = \text{shape}(x)$

Chain rule: $y = f(x), z = g(y)$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Back propagation:

- From loss function to input direction.
- Store value for each step for quick read;
- Using the parameters from forwarding



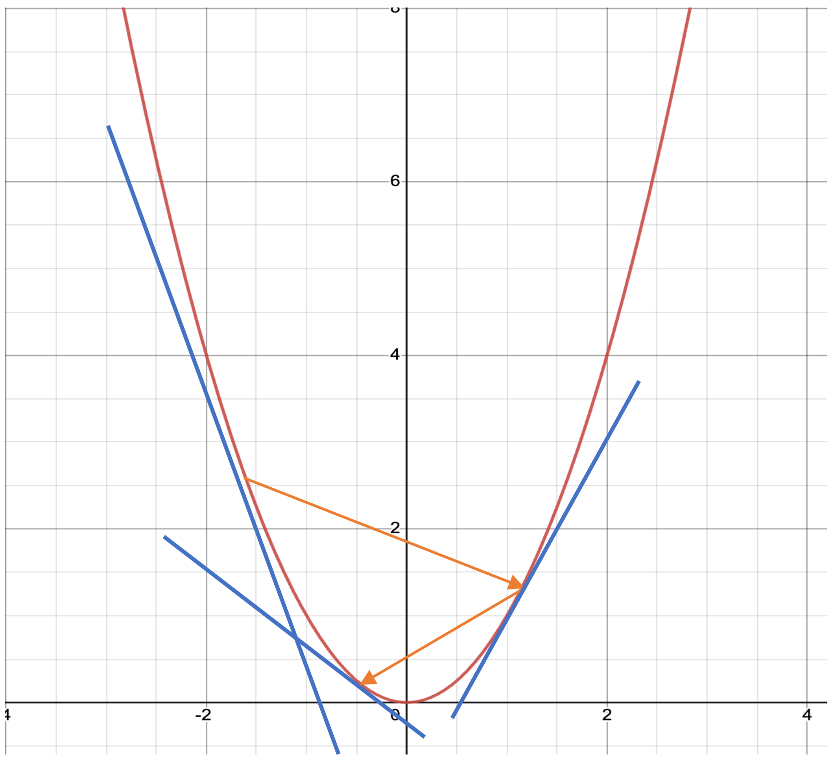

$\frac{\partial L}{\partial y1} = \frac{\partial L}{\partial y2} \times \frac{\partial y2}{\partial y1} \quad \frac{\partial L}{\partial W1} = \frac{\partial L}{\partial y1} \times \frac{\partial y1}{\partial W1} \quad \frac{\partial L}{\partial y2} \quad \frac{\partial L}{\partial W2} = \frac{\partial L}{\partial y2} \times \frac{\partial y2}{\partial W2}$

Shape(n, 1), shape(n, m) = shape(n,1) . Shape(m, 1) $y_1^T$

Shape(m, 1) = shape(n, 1) x shape(n, m), shape(m, x) = shape(m, 1) x shape(x, 1)

# Stochastic Gradient Descent and Learning Rate

L = w²




$$\underbrace{(\mathbf{w},b) \leftarrow (\mathbf{w},b) - \frac{\eta}{|\mathcal{B}|} \overbrace{\sum_{i \in \mathcal{B}} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w},b)}^{\text{gradient}}.}_{\text{Optimizer}}$$

η learning rate

B mini-batch size

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w},b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_b l^{(i)}(\mathbf{w},b) = b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

# Training Process – from scratch

```
# Initialize parameters
w = torch.normal(0, 0.01, size=(2,1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# Define the model:
def linreg(X, w, b):
        return torch.matmul(X, w) + b

# Define loss function:
def squared_loss(y_hat, y):
        return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2

# Define optimizer
def sgd(params, lr, batch_size):
        with torch.no_grad():
                for param in params:
                    param -= lr * param.grad / batch_size
                    param.grad.zero_()
```

# Training Process (Cont.)

```
# hyperparameters
lr = 0.03
num_epochs = 3
net = linreg
loss = squared_loss
# train the model
for epoch in range(num_epochs):
        for X, y in data_iter(batch_size, features, labels):
                l = loss(net(X, w, b), y)
                l.sum().backward()
                sgd([w, b], lr, batch_size)
        with torch.no_grad():
                train_l = loss(net(features, w, b), labels)
```

# Text Processing

- Text cleaning
- Co-occurance
- Word2vec
  - CBOW

Data Cleaning → Tokenisation → Stopwords Removal → Lemmatization/ Stemming

- Data structure
- Data source
- Common dirty strings:
  - HTML tags
  - Human typos
  - Data encoding
  - Punctuations

➢"<b>A touching movie!!\n</b> It is full of emotions and wonderful acting\n\n\n.<br> I could have sat through it a second time.”

Python string functions
Regular expression – Regex

➢“A touching movie It is full of emotions and wonderful acting I could have sat through it a second time”

Data Cleaning → Tokenisation → Stopwords Removal → Lemmatization. Stemming

- “A touching movie It is full of emotions and wonderful acting I could have sat through it a second time”

Python string functions
Libraries: Nltk, WordNet, Spacy …

- [“a”, “touching”, “movie”, “it”, "is”, “full”, “of”, “emotions”, “and”, “wonderful”, “acting”, “I”, “could”, “have”, “sat”, “through”, “it”, “a”, “second”, “time”]

Data Cleaning → Tokenisation → Stopwords Removal → Lemmatization/ Stemming

- [“a”, “touching”, “movie”, “it”, "is”, “full”, “of”, “emotions”, “and”, “wonderful”, “acting”, “I”, “could”, “have”, “sat”, “through”, “it”, “a”, “second”, “time”]

Python string functions (customization)
Libraries: Nltk, WordNet, Spacy …

- [“touching”, “movie”, “full”, “emotions”, “wonderful”, “acting”, “have”, “sat”, “second”, “time”]

Data Cleaning → Tokenisation → Stopwords Removal → Lemmatization/ Stemming

- ["touching", "movie", "full", "emotions", "wonderful", "acting", "have", "sat", "second", "time"]

Libraries: Nltk, WordNet, Spacy …

- ["touching", "movie", "full", "emotion~~s~~", "wonderful", "act~~ing~~", "have", "s~~a~~it", "second", "time"]
- ["touch~~ing~~", "movi~~e~~", "full", "emot~~ions~~", "wonder~~ful~~", "act~~ing~~", "have", "s~~a~~it", "second", "time"]

# How to represent words so that computer can understand?

"stand on his head" "easily"

car ⟷ auto | automobile | motorcar

- Thesaurus
- Co-occurrence
- **word2vec**
- **CBOW**

# Co-occurrence

- Represent the words
- Keep the context
  - Window size = 1

Window size

➢ [“I”, “eat”, ‘burger”, “and”, “you”, “eat”, “salad”]

Corpus

word_to_id

id_to_word

➢ [1, 2, 3, 4, 5, 2, 6]

Vocabulary

➢ {“I”:1, “eat”:2, “burger”:3, “and”:4, “you”:5, “salad”: 6}

# Co-occurrence Matrix

| | I | eat | burger | and | you | salad |
|---|---|---|---|---|---|---|
| burger | 0 | 1 | 0 | 1 | 0 | 0 |

← Unique words

← Co-occurrence count

Vector for “burger” = [0, 1, 0, 1, 0, 0, 0]

Repeat for all words in our sentence …

| | |
|---|---|
| I | [0, 1, 0, 0, 0, 0] |
| Eat | [1, 0, 1, 0, 1, 1] |
| Burger | [0, 1, 0, 1, 0, 0] |
| And | [0, 0, 1, 0, 1, 0] |
| You | [0, 1, 0, 1, 0, 0] |
| Salad | [0, 1, 0, 0, 0, 0] |

# Word2vec – a prediction problem

["I", "eat", '???", "and", "you", "eat", "salad"]

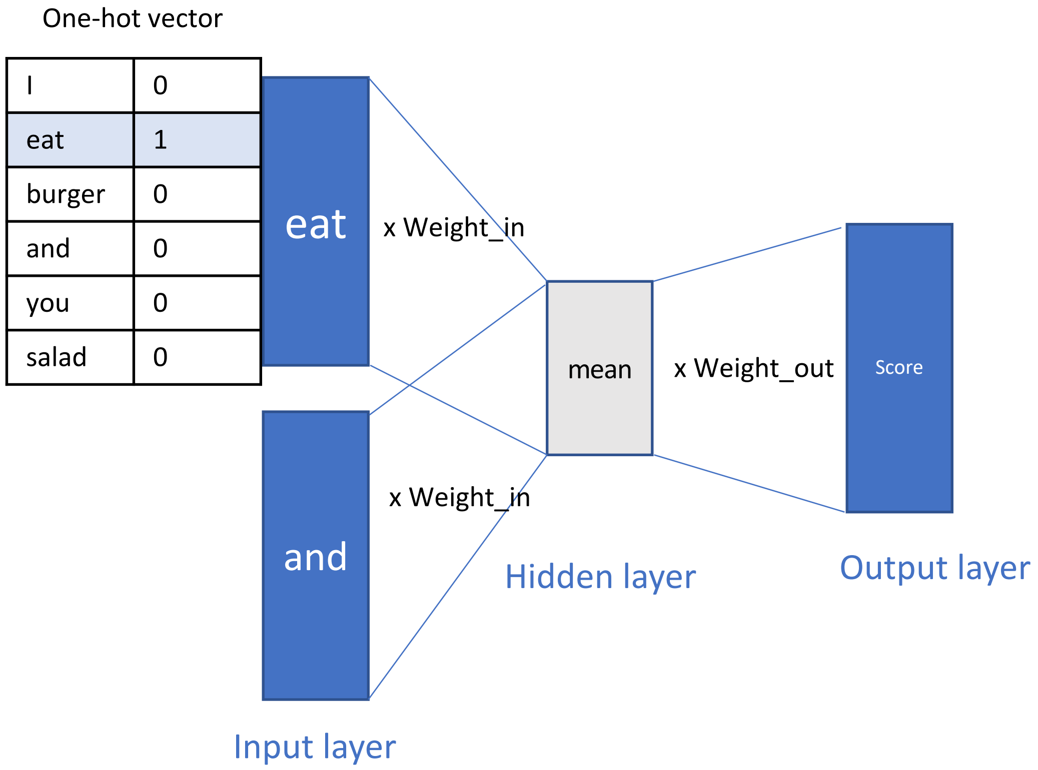
Context: "eat" "and" → Model → Probability distribution

- One-hot vector:

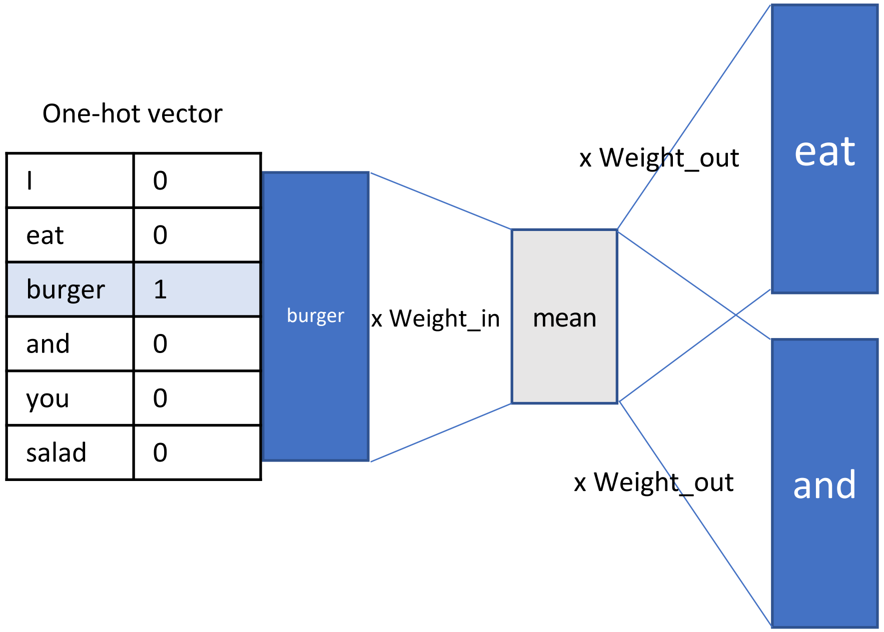
| ID | Word | I | eat | burger | and | you | salad |
|---|---|---|---|---|---|---|---|
| 2 | eat | 0 | 1 | 0 | 0 | 0 | 0 |

# Word2vec

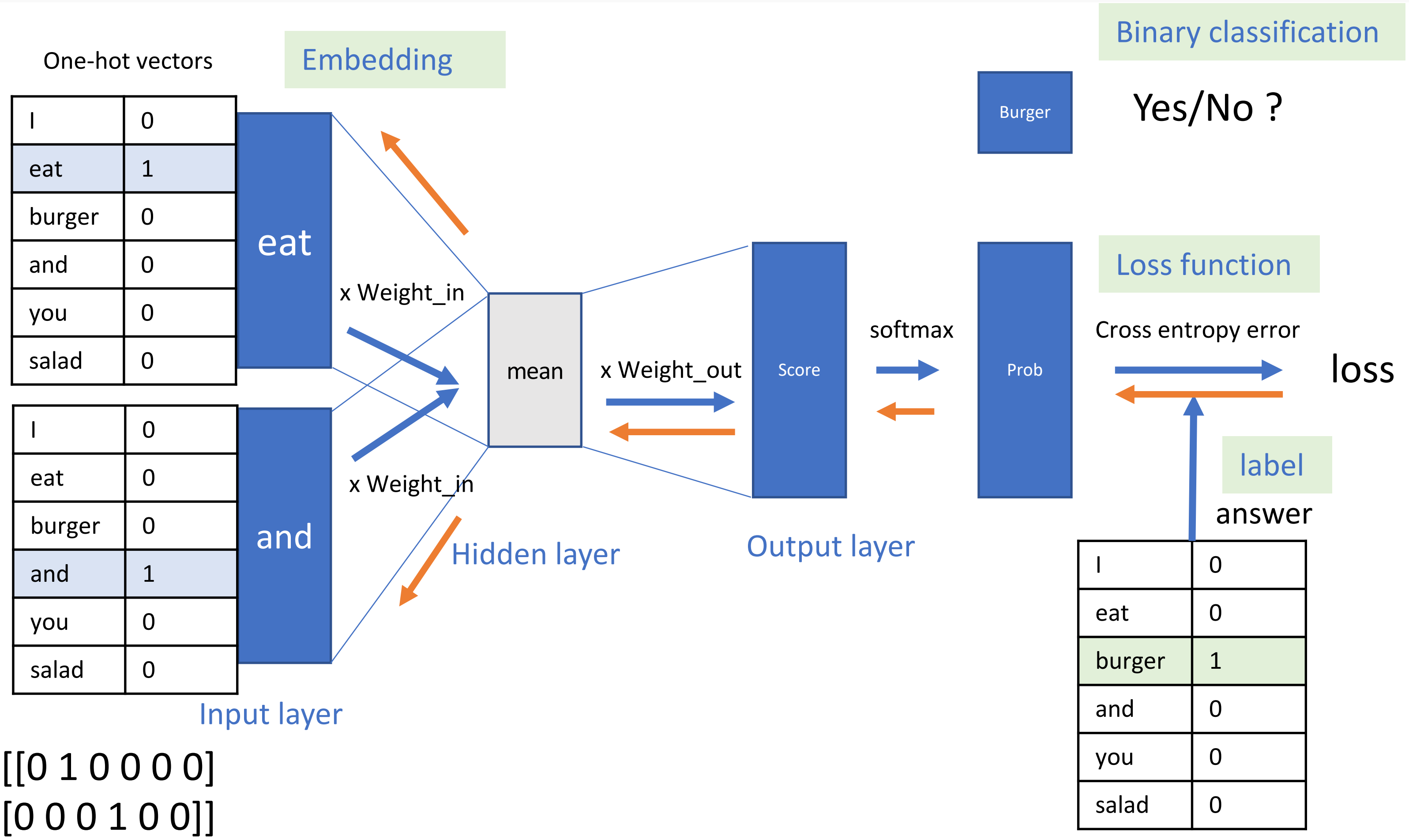
## CBOW




## Skip-gram



# CBOW Learning Process




Softmax – get probability

$$\mathrm{softmax}(\mathbf{X})_{ij} = \frac{\exp(\mathbf{X}_{ij})}{\sum_k \exp(\mathbf{X}_{ik})}.$$

Cross-entropy loss

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

# CBOW Result – Distributional Representation

- Weight_in matrix to represent words meaning
  - Syntax – plurals, past tenses…
  - Semantics
    - "king – men + women = queen"

```
[analogy] king:man = queen:?
 woman: 5.161407947540283
 veto: 4.928170680999756
 ounce: 4.689689636230469
 earthquake: 4.633471488952637
 successor: 4.6089653968811035
```

```
[analogy] take:took = go:?
 went: 4.548568248748779
 points: 4.248863220214844
 began: 4.090967178344727
 comes: 3.9805688858032227
 oct.: 3.9044761657714844
```

```
[analogy] car:cars = child:?
 children: 5.217921257019043
 average: 4.725458145141602
 yield: 4.208011627197266
 cattle: 4.18687629699707
 priced: 4.178797245025635
```

# Language Model

- Language model: the probability of a sequence of words.

$$P(w_1, \cdots, w_m) = P(w_m \mid w_1, \cdots, w_{m-1}) P(w_{m-1} \mid w_1, \cdots, w_{m-2})$$

$$\cdots P(w_3 \mid w_1, w_2) P(w_2 \mid w_1) P(w_1)$$

$$= \prod_{t=1}^{m} P(w_t \mid w_1, \cdots, w_{t-1}) \text{①}$$

$$P(A,B) = P(A|B)P(B)$$

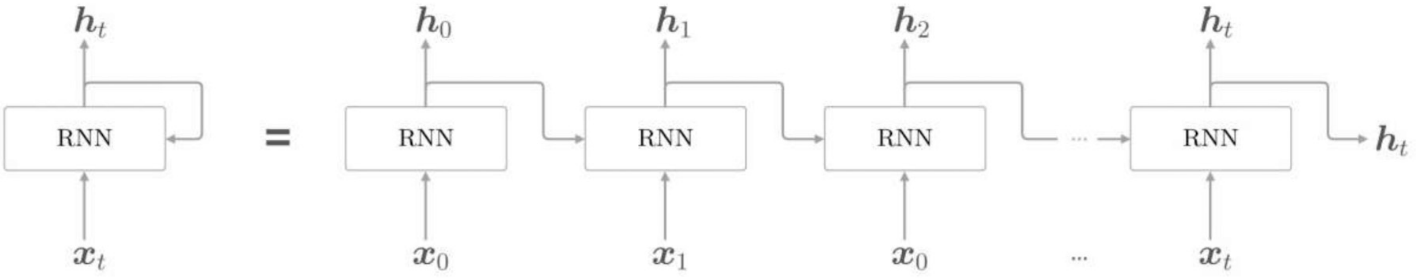
# Recurrent Neural Networks

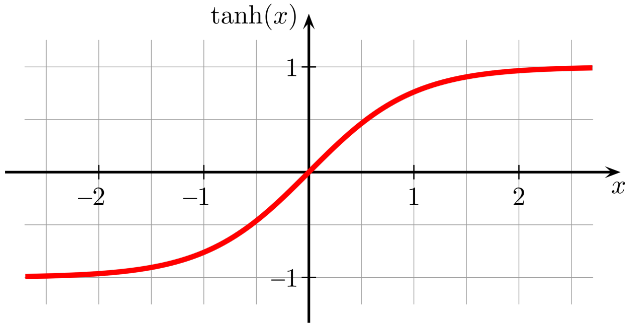
- Simple RNN
- LSTM
- Seq2seq
- Attention
- Transformer

# Simple RNN

Hidden state

Truncated BPTT – Truncated Backpropagation Through Time

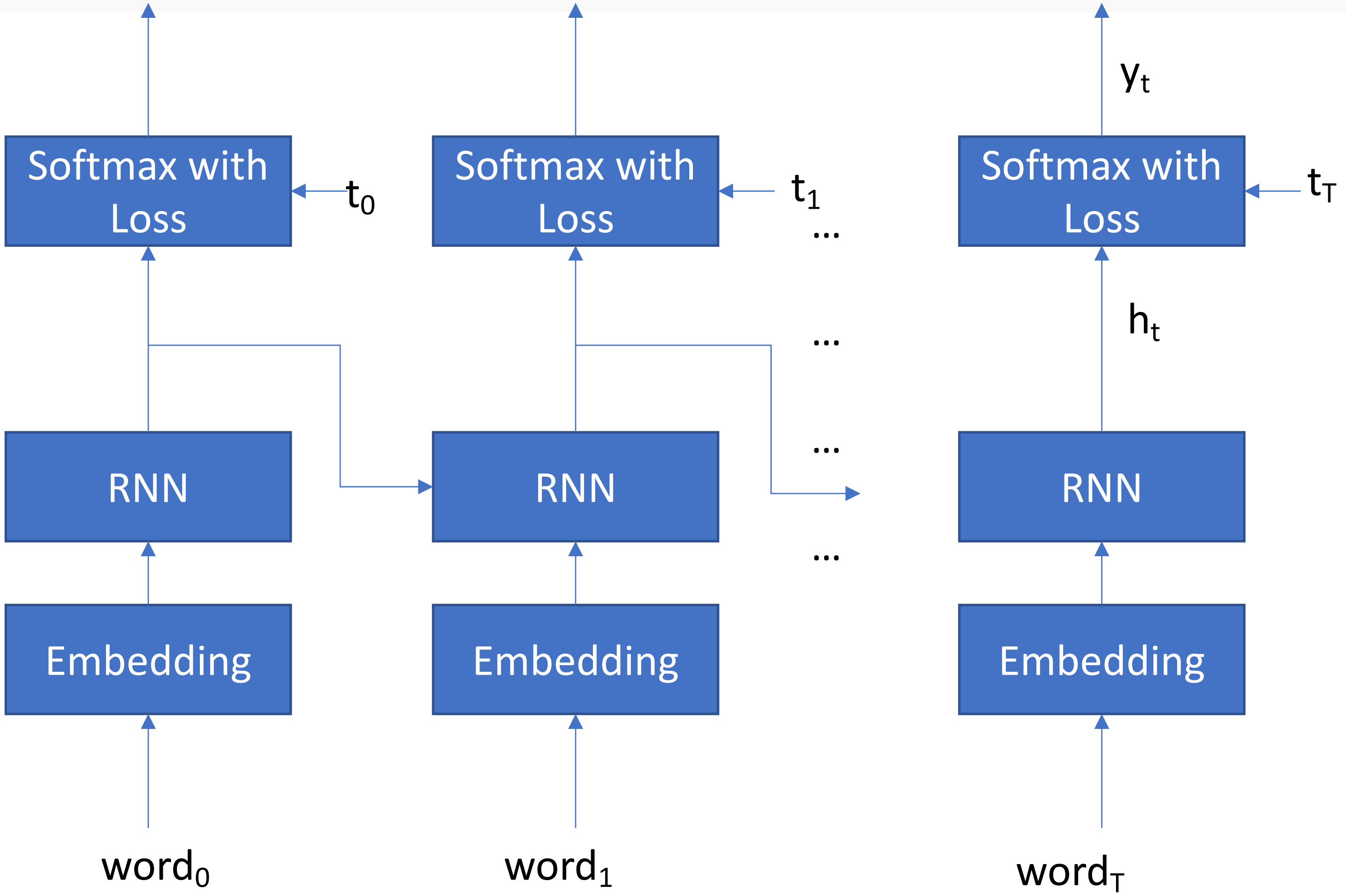



$$h_t = tanh(h_{t-1}W_h + x_tW_x + b)$$

$$\text{Output} = W_h * h_t$$

# Simple RNN

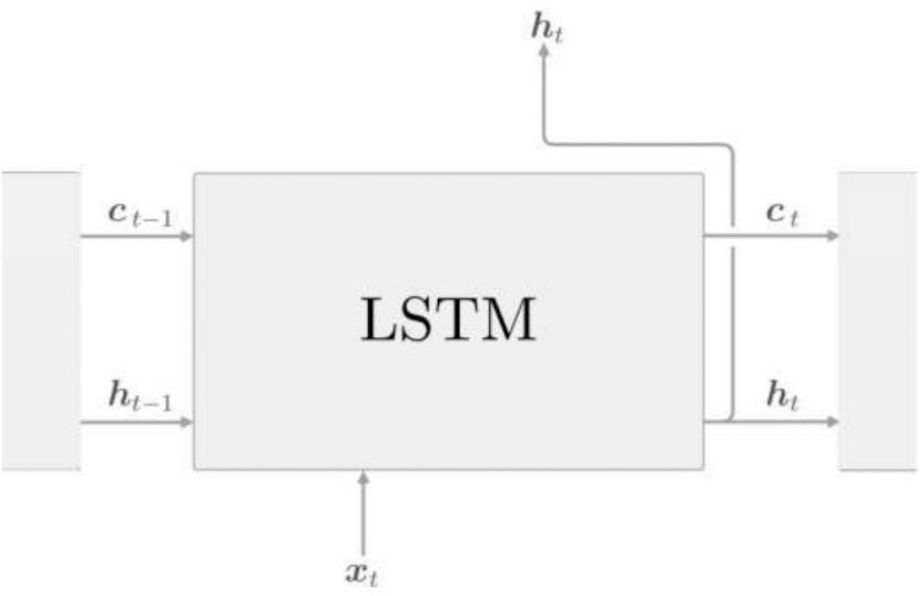


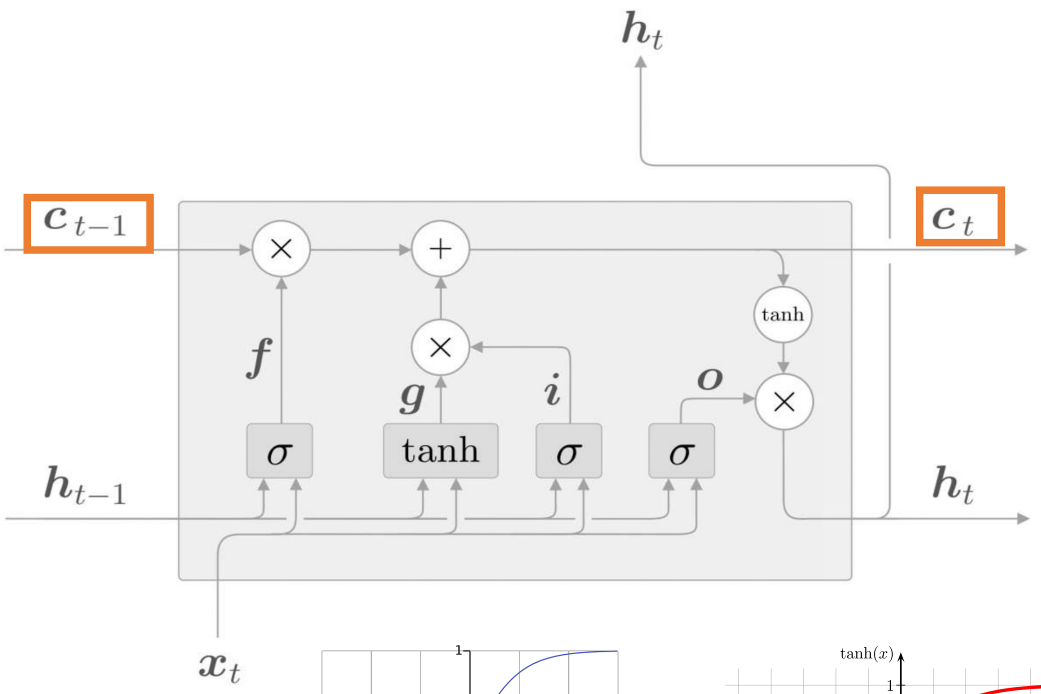

$$L = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

$$\text{Perplexity} = e^L$$

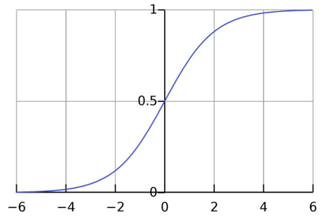
Perplexity: the number of possible candidates

# LSTM – Long Short-Term Memory (Gated RNN)

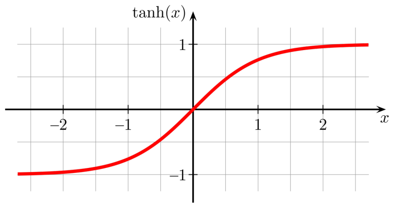



Controls gates



Sigmoid(x)



Tanh(x)

# LSTM – Long Short-Term Memory (Gated RNN)

- Output gate

$$\boldsymbol{o} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{o})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{o})} + \boldsymbol{b}^{(\mathrm{o})})$$

- Forget gate

$$\boldsymbol{f} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{f})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{f})} + \boldsymbol{b}^{(\mathrm{f})})$$

- Memory gain

$$\boldsymbol{g} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{g})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{g})} + \boldsymbol{b}^{(\mathrm{g})})$$

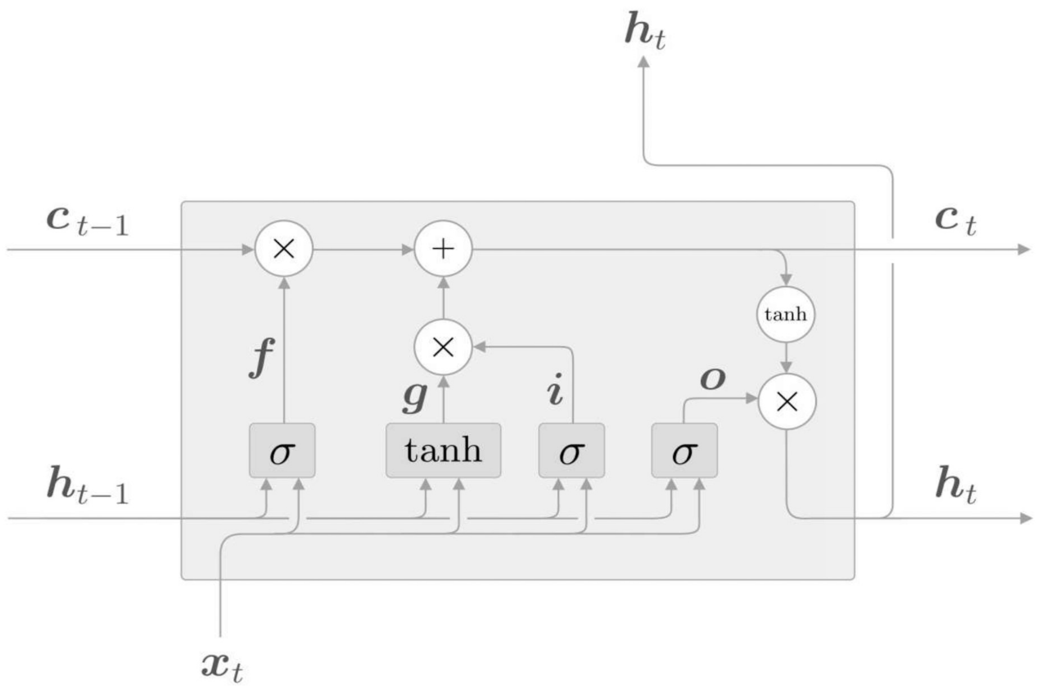
- Input gate

$$\boldsymbol{i} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^{(\mathrm{i})} + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^{(\mathrm{i})} + \boldsymbol{b}^{(\mathrm{i})})$$

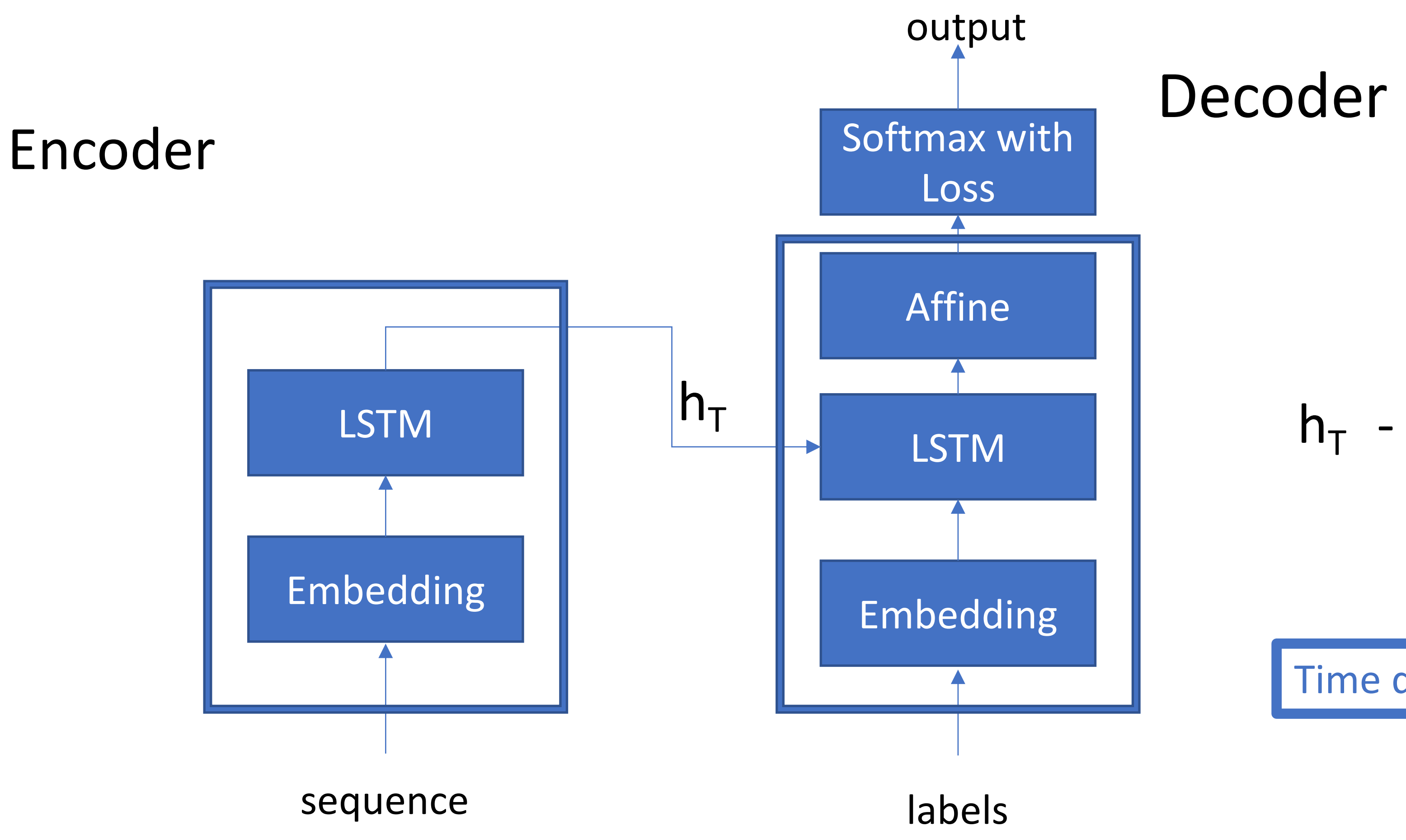
Element-wise multiplication

$$\boldsymbol{h}_t = \boldsymbol{o} \odot \tanh(\boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \boldsymbol{f} \odot \boldsymbol{c}_{t-1} + \boldsymbol{g} \odot \boldsymbol{i}$$



# Seq2Seq Model



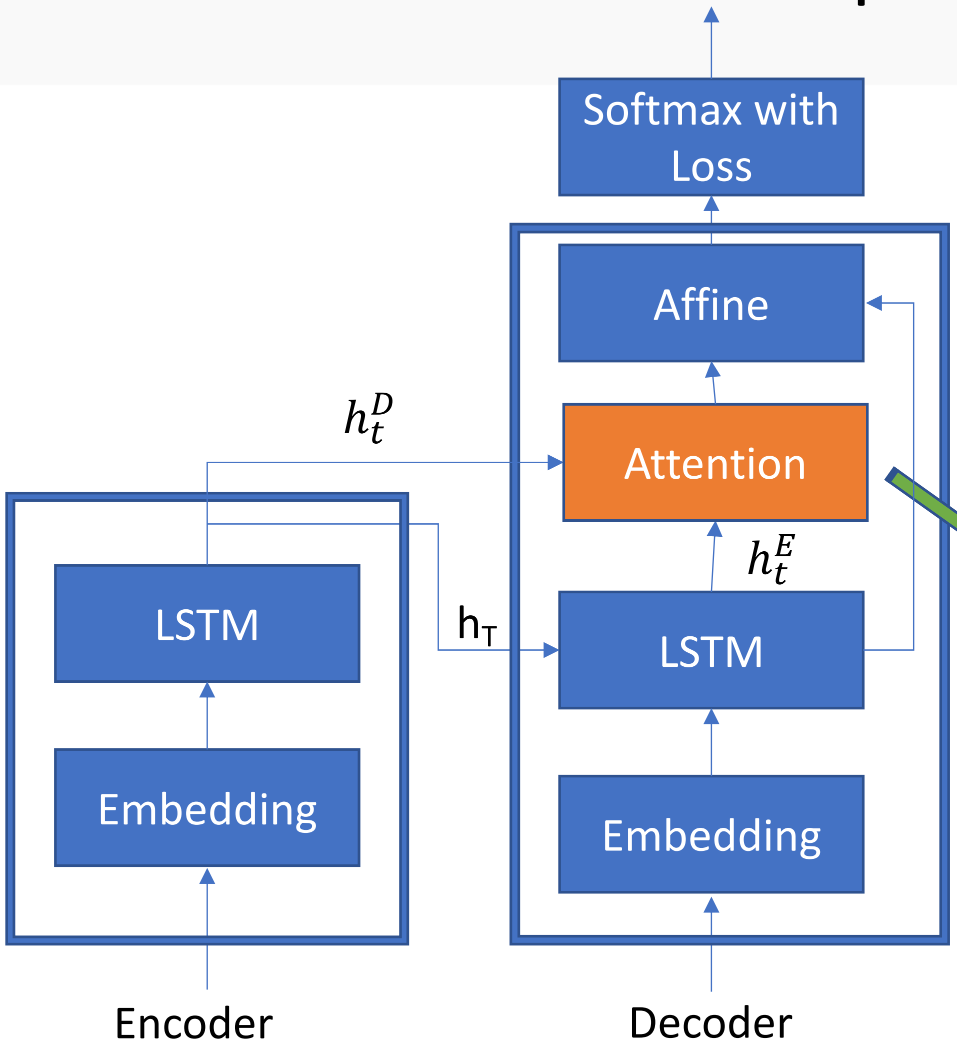

Applications:
- Translation
- Chat bot
- Summarization
- Image to caption

$h_T$ - the last row of hidden state

Time direction repetition

NCI AUSTRALIA

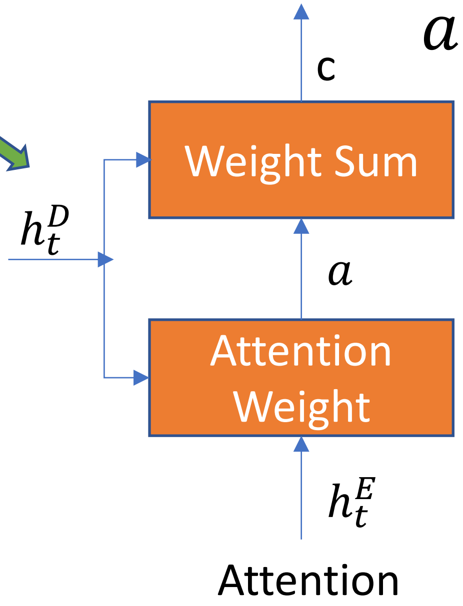
# Attention in Seq2Seq




Weight sum

$$sum(h_t^D \odot a) = \mathrm{c}$$ Context vector

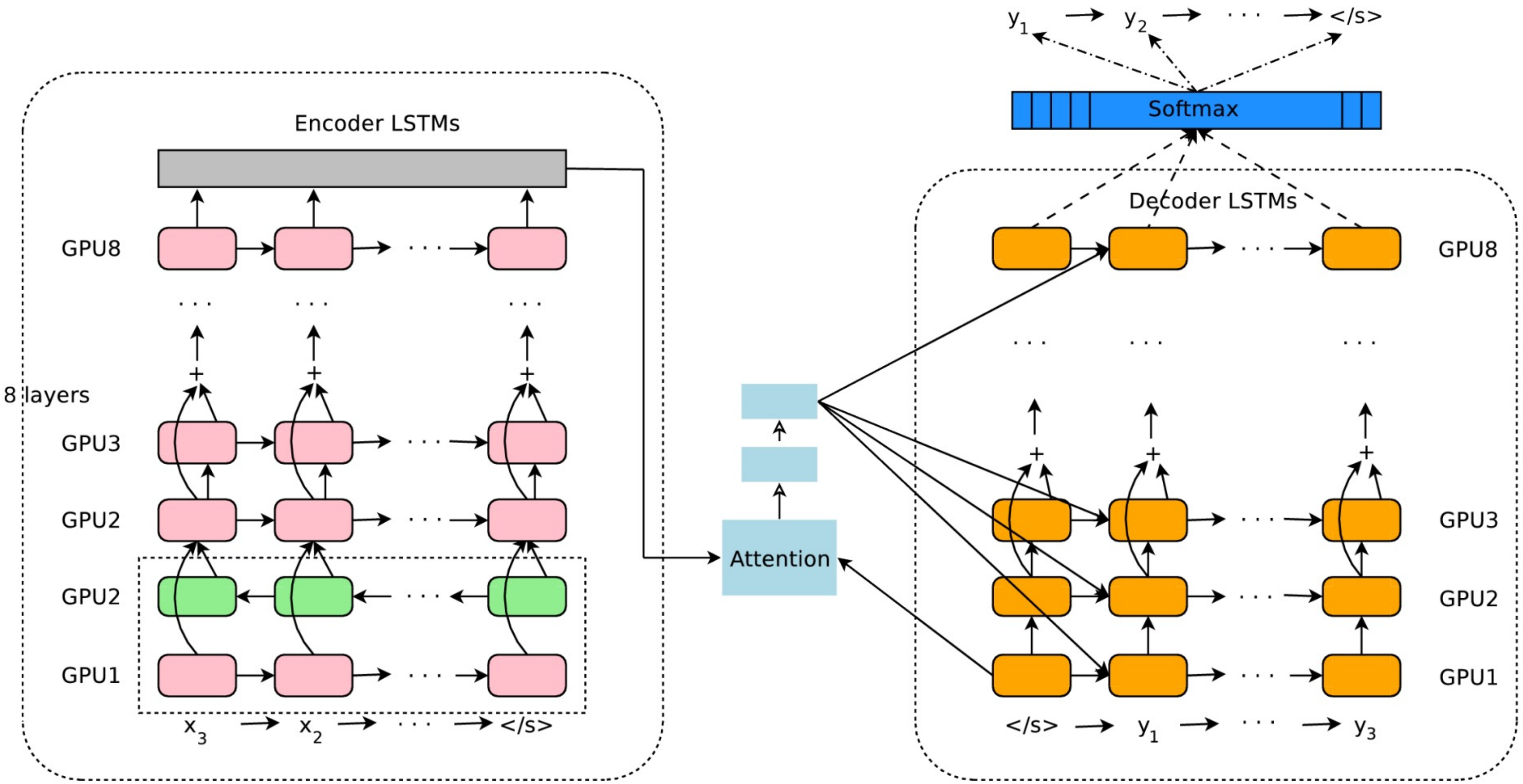
$a$ - probability distribution of each word's importance




Attention weight

$$a = \mathrm{softmax}\Big(\underbrace{sum(h_t^D \odot h_t^E)}_{\text{e.g. vector similarity, here we use dot product}}\Big)$$

# Example. Google Neural Machine Translation



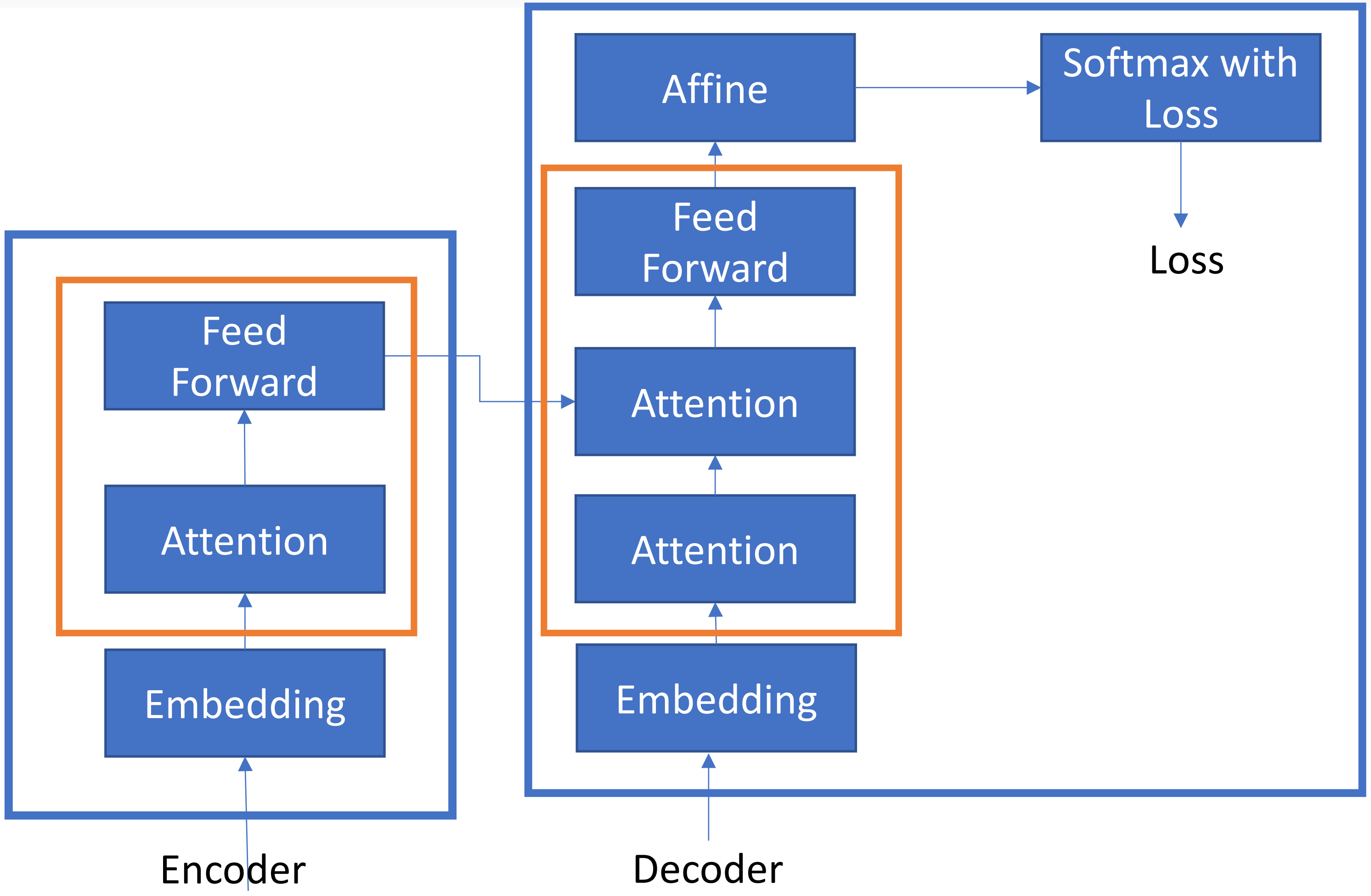



*Wu, Yonghui, et al. Google's neural machine translation system:Bridging the gap between human and machine translation[J]. arXiv preprint arXiv:1609.08144, 2016.*

# Transformers

- Transformer
- GPT
- Bert

# Transformer – Attention without RNN




- Self-attention instead of LSTM layer
- Multi-head attention

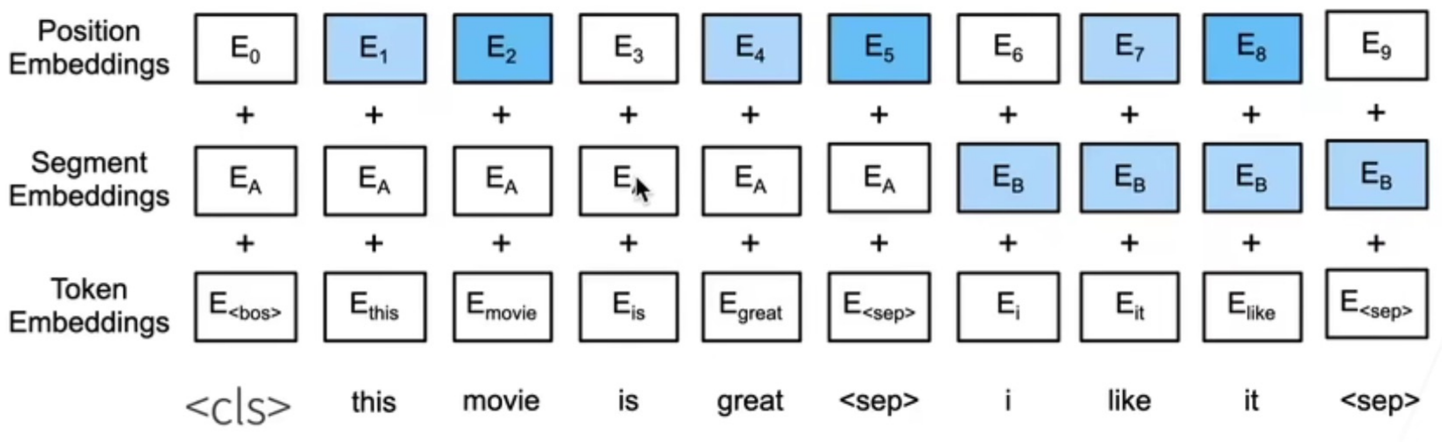
Depth repetition

# Bert – NLP Model Based on Fine Tuning!

## Bert – transformer without decoder

- Pre-trained model has extracted enough features
- Only need to replace output layer for a new task
- Original models in the paper:
  - Base: 12 (transformer encoder) blocks, hidden size = 768, 12 heads, 110M parameters
  - Large: 24 blocks, hidden size = 1024, 16 heads, 340M parameters
  - Trained on more than 3B words (whole Wikipedia and some books)

# Bert

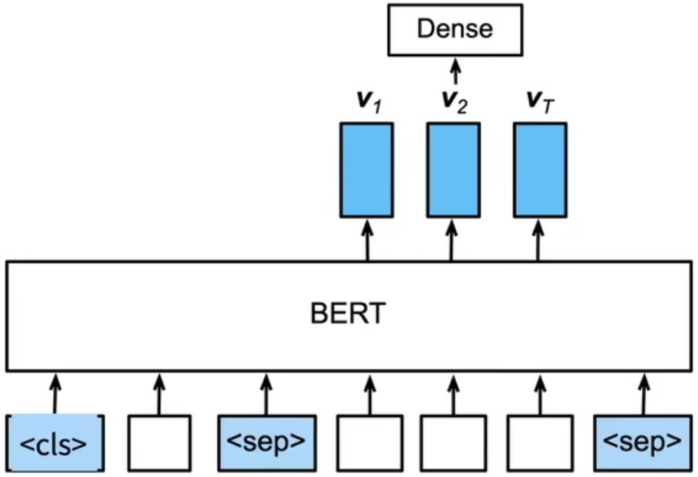



Pretraining:

- Masked language model - randomly mask some words in the sentence to predict
- Predict next sentence – 50% chance to select adjacent sentences as paired input, 50% chance to select random sentence pairs. Predict <cls> in output

1. Paired sentence input
2. Additional special tokens
3. Trainable position embedding

# Bert – Q&A




Question Paragraph

Output:

- Token is the start of the answer
- Token is the end of the answer
- Neither

Fine tuning:
Increase the learning rate for output layer
Set some of the base layer parameters so finish training faster

# References


- *Deep Learning from Scratch 2 © 2018 Koki Saitoh, O' Reilly Japan, Inc.*
- *Wu, Yonghui, et al. Google's neural machine translation system:Bridging the gap between human and machine translation[J]. arXiv preprint arXiv:1609.08144, 2016.*
- *https://www.youtube.com/watch?v=T05t-SqKArY&list=TLPQMjQxMDIwMjLirqQmTCjo-w&index=1*
- *https://www.youtube.com/watch?v=6ArSys5qHAU*

# Questions ?

NCI AUSTRALIA


## NCI Contacts

General enquiries: +61 2 6125 9800
Media enquiries: +61 2 6125 4389

Support: help@nci.org.au

Email: zhuochen.wu@anu.edu.au

## Address

NCI, ANU Building 143
143 Ward Road
The Australian National University
Canberra ACT 2601