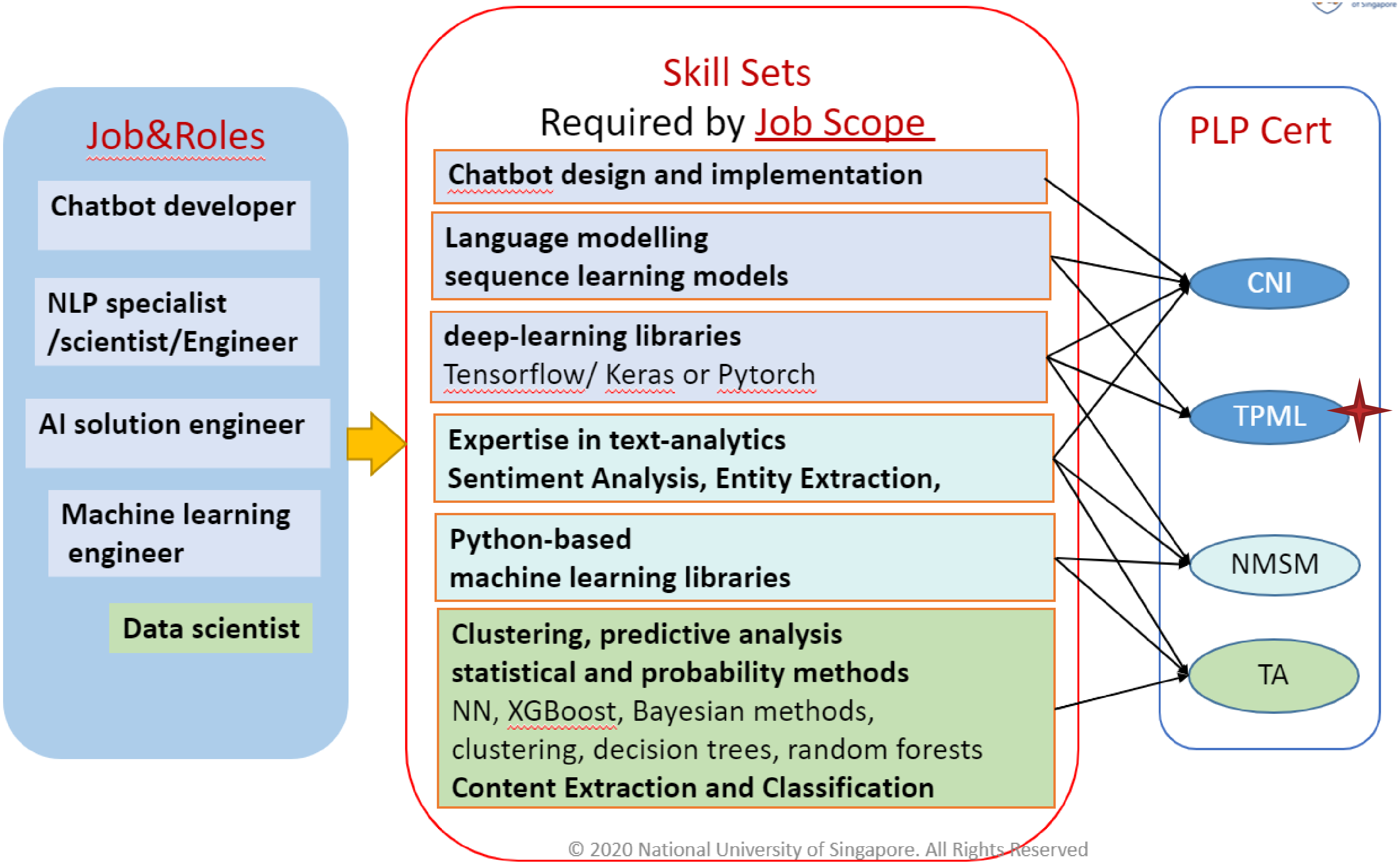
Text Processing Using Machine Learning

Neural Nets for Text Processing

Dr Wang Aobo
aobo.wang@nus.edu.sg

PLP Cert



Job&Roles
Chatbot developer
NLP specialist /scientist/Engineer
AI solution engineer
Machine learning engineer
Data scientist
Skill Sets
Required by Job Scope
Chatbot design and implementation
Language modelling
sequence learning models
deep-learning libraries
Tensorflow/ Keras or Pytorch
Expertise in text-analytics
Sentiment Analysis, Entity Extraction,
Python-based
machine learning libraries
Clustering, predictive analysis
statistical and probability methods
NN, XGBoost, Bayesian methods,
clustering, decision trees, random forests
Content Extraction and Classification
PLP Cert
CNI
TPML
NMSM
TA

Learning Outcomes

- **Recent** techniques for NLP
  - Deep learning
- Getting a basic grasp of deeplearning library for NLP
  - Knowledge can be applied to any tensor libraries
- Understanding the **underlying** techniques and knowing how to implement and integrate them.

Agenda

- **Overview of NLP**
  - Applications
  - Statistical Modelling vs. Deep Neural Nets
- Deep Learning Basics for NLP
  - MLP
  - Deep Learning Training Routine (quiz)
  - Workshop: Basic NN on Colab
- Word2Vec & DL Specific
  - CBOW and SkipGram
  - ActivationFunction/LossFunction
  - Optimiser/Learning Rate
  - Workshop: Word2Vec from Scratch

What can NLP do?

2 basic Use Cases:

1. **Automatically put text into categories- Classification**
   - Sentiment detection
   - Spam email detection
   - Emotion detection

2. **Extract specific information from the text- Extraction**
   - Named Entity extraction from sentence

What can NLP do?

Other *Fancier* Use Cases:

1. **Object Classification/Clustering & Recommendation**
2. **Search Engine**
3. **Question Answering System**
4. **Voice Assistant**
5. **Machine Translation**
6. **Grammar Error Correction & Language Learning**
7. **Chatting Robots**
8. **"*Fake Articles*" Generation & Detection**
9. **....**

How does NLP Work

- The whole task here is…

Documents

Lost glamor

High tea at Raffles!

Not what it was, but still

Amazing service
Rated 5 by travel-gini on Feb 26, 2013

Great location with a little bit of history, the staff make this hotel though

smart

Doc/Feature Matrix

	amazing	service	lost	glamour	...
Doc1	1.5	2.1	0	0	
Doc2	0	0	3.1	1.5	
Doc3	0	0	0	1.9	
Doc4	0	0	0	0	
...					

ML + CPU
Supervised
Unsupervised

Classic NLP vs. Deep learning

- **Frequency TF-IDF and PPMI vectors are**
  - long   (|V| > 100,000)
  - sparse (lots of zero)
  - efficient for simple tasks with reasonably large of dataset
  - interpretable as designed by human intelligence
  - **difficult to capture contextual dependency**

	he	drink	hold	…	*tfidf* drink apple	*tfidf* hold apple	*tfidf* apple juice	…	*PPMI* drink apple	*PPMI* hold apple	*PPMI* apple juice
sent0	0.01	0.38	0.00	…	0.87	0.00	0.92	…	4.23	0.00	8.90
sent1	0.01	0.00	0.28	…	0.00	0.87	0.00	…	0.00	2.45	0.00

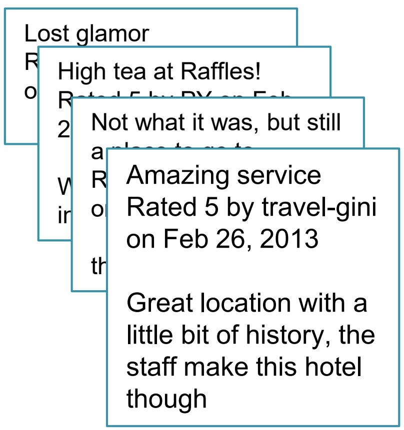
Statistical modelling

- **All models are wrong, but some are useful**
  - for Document/Sentence Classification
    - SVM/KNN/Decision Tree/RandomForest/Naïve Bayes/MaxEnt
  - for Sequence Labeling
    - HMM/CRF
  - for Language modeling
    - N-gram/RandomForest/MaxEnt
  - for Machine Translation
    - Phrase/Tree-based Model + beam search

How does NLP Work

- The whole task here is…

Documents



smart →

Doc/Feature Matrix

	amazing	service	lost	glamour	…
Doc1	1.5	2.1	0	0	
Doc2	0	0	3.1	1.5	
Doc3	0	0	0	1.9	
Doc4	0	0	0	0	
…					

\+ ML + CPU
Supervised
Unsupervised

- Able to generate this Matrix automatically ?
- Able to generate this Matrix task-independently ?
- Yes! Learn to generate from data

How does NLP Work

- The whole task here is…

Documents

Lost glamor

High tea at Raffles!

Not what it was, but still

Amazing service
Rated 5 by travel-gini on Feb 26, 2013

Great location with a little bit of history, the staff make this hotel though

smart

Doc/Feature Matrix

	amazing	service	lost	glamour	...
Doc1	1.5	2.1	0	0	
Doc2	0	0	3.1	1.5	
Doc3	0	0	0	1.9	
Doc4	0	0	0	0	
…					

ML + CPU
Supervised
Unsupervised

naive

Word Indexing Matrix

	amazing	service	lost	glamour	...
amazing	1	0	0	0	
service	0	1	0	0	
lost	0	0	1	0	
glamour	0	0	0	1	
…					

DNN + GPUs
Supervised

=

	?	?	?	?	?
Doc1	11.5	2.1	5.70	-30.2	
Doc2	-3.40	0.34	3.1	1.5	
Doc3	5.8	0.560	5.9	1.9	
…					

Classic NLP vs. Deep learning

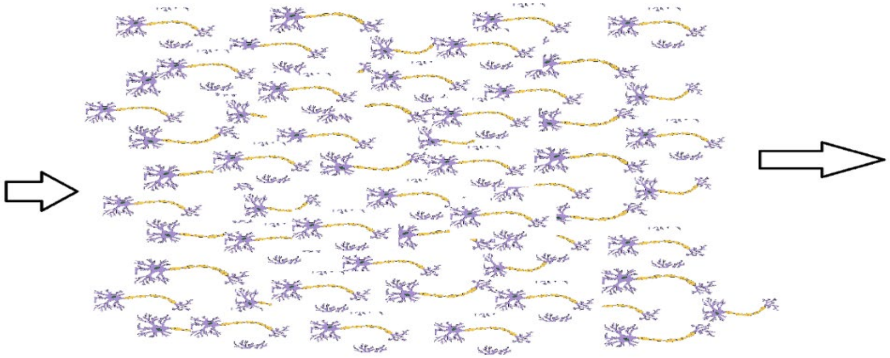
- **Deep learning can create feature vectors that are**
  - short (often fixed-sized <2000, decided empirically)
  - dense (most are non-zeros)
  - non-Interpretable as decided empirically without human intelligence
  - able to capture contextual dependency
  - beneficial to all tasks (classification/sequence labeling/Translation/QA)

Agenda

- Overview of NLP
- **Deep Learning Basics for NLP**
  - MLP
  - Deep Learning Training Routine (quiz)
- Workshop: Deep Learning from Scratch
- Word2Vec & DL Specific
  - CBOW and SkipGram
  - ActivationFunction/LossFunction
  - Optimiser/Learning Rate
  - Workshop: Word2Vec from Scratch

Deep Learning Basics

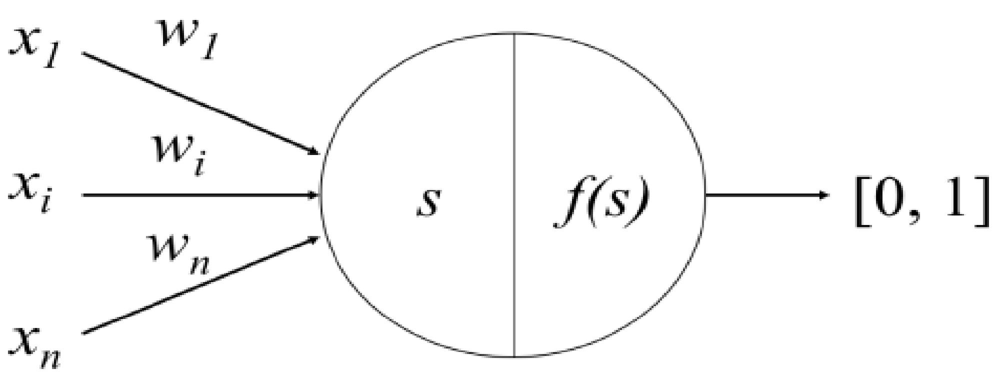
- Perceptron
  - "*system that depends on **probabilistic** rather than deterministic principles* for its operation, gains its reliability from the properties of statistical measurements obtain from a large population of elements“ - Frank Rosenblatt (1957)
  - 100 billion **perceptron** in our brain
  - BERT ~ 110 million to 17 billion **para**
  - GPT ~ 1.5 billion to 175 billion **para**

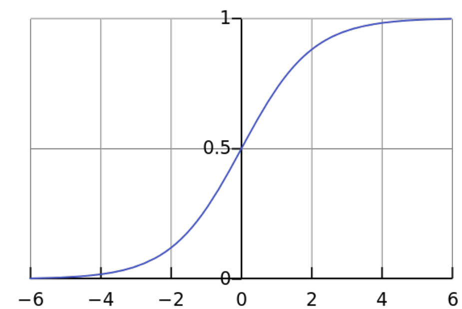


(*Image from Akshay Chandra Lagandula’s blog)

Recap Perceptron

- Given a **set of inputs x**, perceptron
  - learns ***w vector*** to map the inputs to a real-value output between [0,1]
  - through the summation of the dot product of the ***w·x***
  - with a transformation function (aka. **activation function**)





Summation

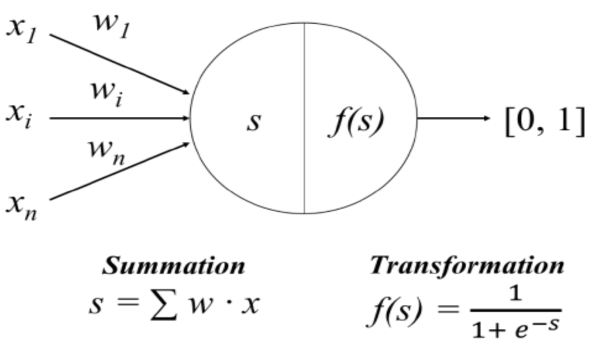
$$s = \sum w \cdot x$$

Transformation

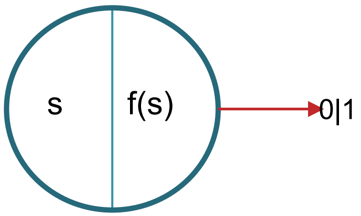
$$f(s) = \frac{1}{1 + e^{-s}}$$

Perceptron

- Word level classification
  - Positive=1;Negative=0
  - let *n=5*

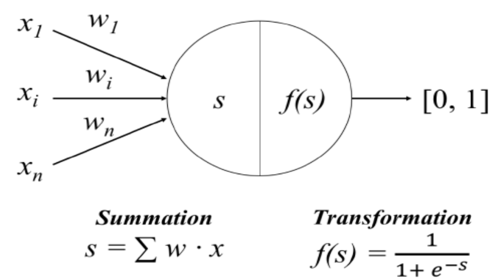


	Like x1	Hate x2	Good x3	Enjoy x4	Bad x5
Like	1	0	0	0	0
Hate	0	1	0	0	0
Good	0	0	1	0	0
Enjoy	0	0	0	1	0
Bad	0	0	0	0	1



Perceptron

- Word level classification
  - Positive=1;Negative=0
  - let *n=5 = vocabulary_size*

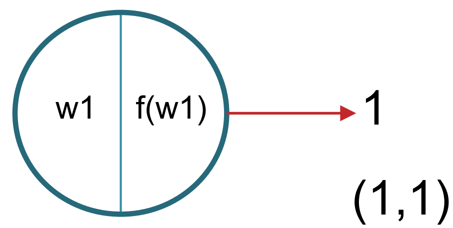


	Like x1	Hate x2	Good x3	Enjoy x4	Bad x5
Like	1	0	0	0	0
Hate	0	1	0	0	0
Good	0	0	1	0	0
Enjoy	0	0	0	1	0
Bad	0	0	0	0	1

(1,5)

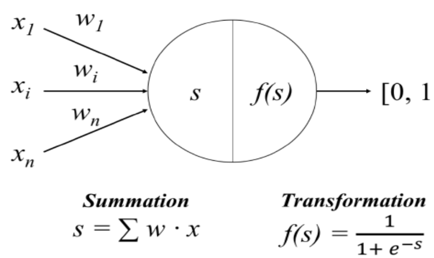
weight
w1
w2
w3
w4
w5

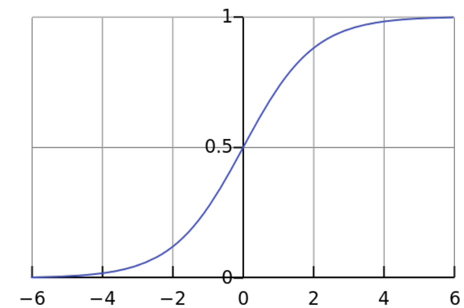
(5,1)



Perceptron

- Word level classification
  - Positive=1;Negative=0
  - let *n=5 = vocabulary_size*
  - *Batch_size = 1*



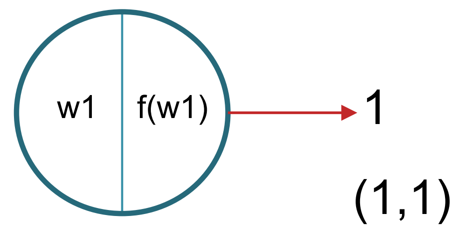


	Like x1	Hate x2	Good x3	Enjoy x4	Bad x5
Like	1	0	0	0	0
Hate	0	1	0	0	0
Good	0	0	1	0	0
Enjoy	0	0	0	1	0
Bad	0	0	0	0	1

(1,5)

weight
6
-6
6
6
-6

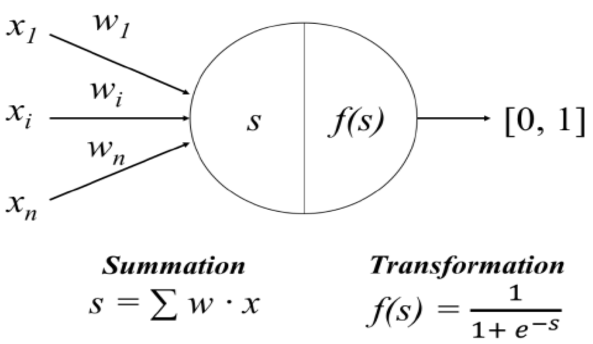
(5,1)



- How about unknown words?

Perceptron

- Word level classification
  - Positive=1;Negative=0
  - let $n=26 << vocabulary\_size$

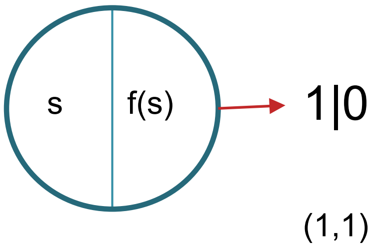


	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	1	1	0	…	0

(1,26)

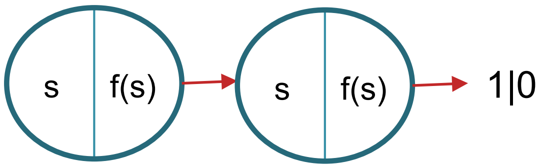
weight
w1
w2
w3
…
w26

(26,1)



Multi-Layer Perceptron

- Word level classification
  - let *n=26 << vocabulary_size*
  - More parameters
  - More layers



	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

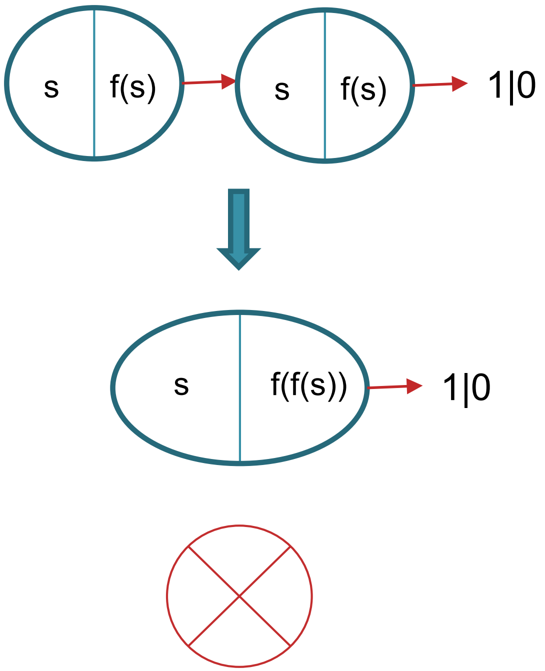
weight
w1
w2
w3
…
w26

Multi-Layer Perceptron

- Word level classification
  - let *n=26 << vocabulary_size*
  - More parameters
  - More layers

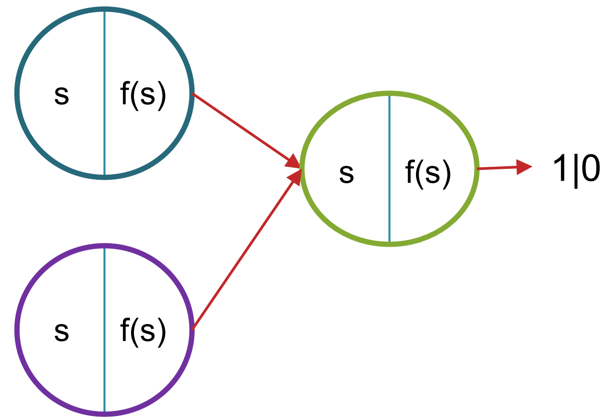
	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

weight
w1
w2
w3
…
w26



Multi-Layer Perceptron

- Word level classification
  - let *n=26 << vocabulary_size*
  - More layers
  - More parameters



	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

(1,26)

weight	weight
wb1	wp1
wb2	wp2
wb3	wp3
…	…
wb26	wp26

(26,2)

weight
wg1
wg2

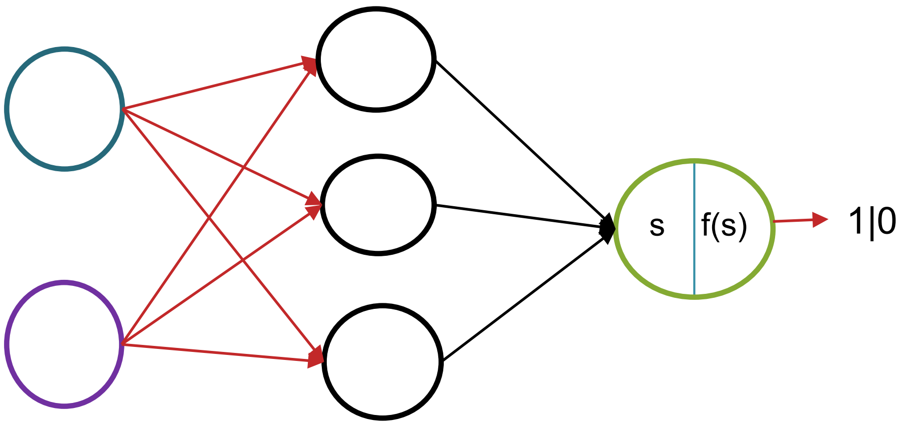
(2,1)

1|0

(1,1)

Multi-Layer Perceptron

- Word level classification
  - let *n=26 << vocabulary_size*
  - More parameters
  - *Even More Layers*



	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

(1,26)

weight	weight
wb1	wp1
wb2	wp2
wb3	wp3
…	…
wb26	wp26

(26,2)

weight	weight	weight
w	w	w
w	w	w

(2,3)

weight
wg1
wg2
wg3

(3,1)

→ Y_pred

(1,1)

Deep leaning Training Routine

Repeat the following until desired

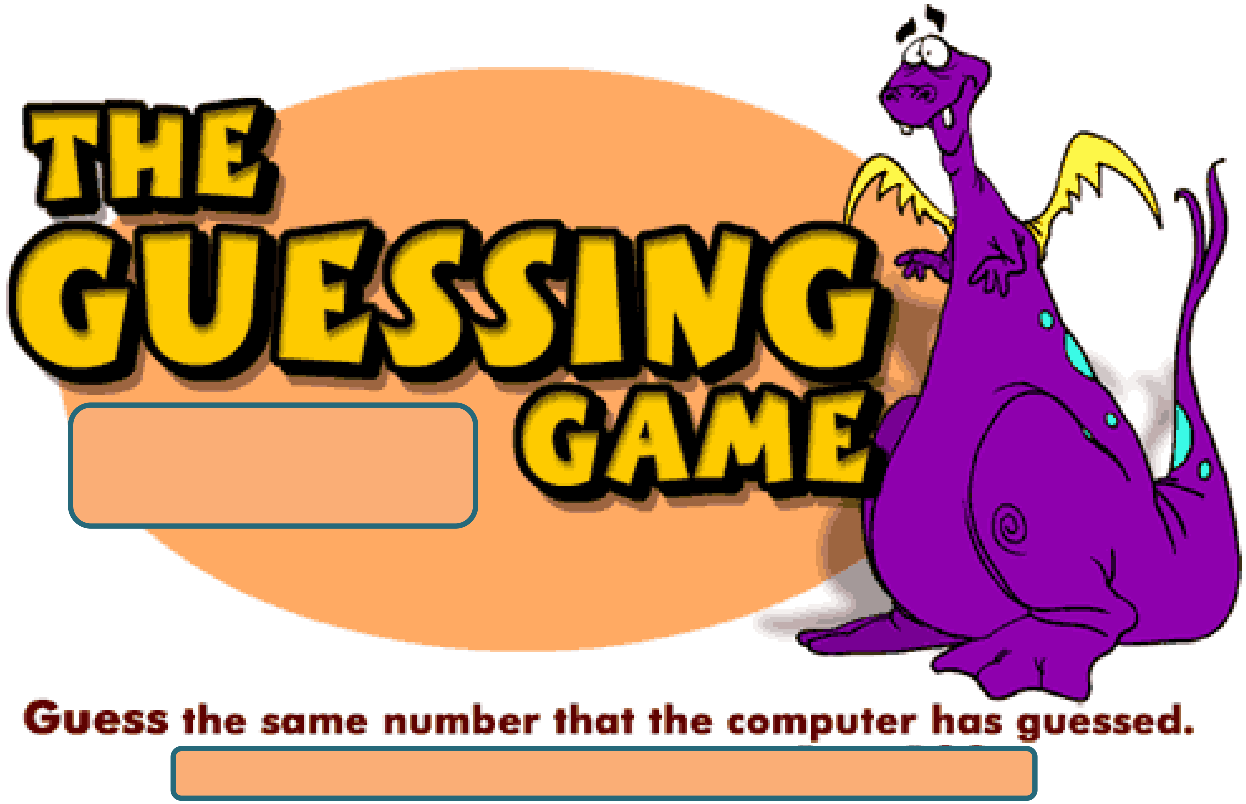
- **Initialize weights vector**
  - Random
  - One-hot encoding
- **Forward Propagation**
- Compute and log the loss
- Back Propagation
- **Optimizer**

Deep leaning Training Routine

- **Optimizer** (**searching** action with a **Strategy**)

Deep leaning Training Routine

- **Optimizer** ("brute force" **searching** action)

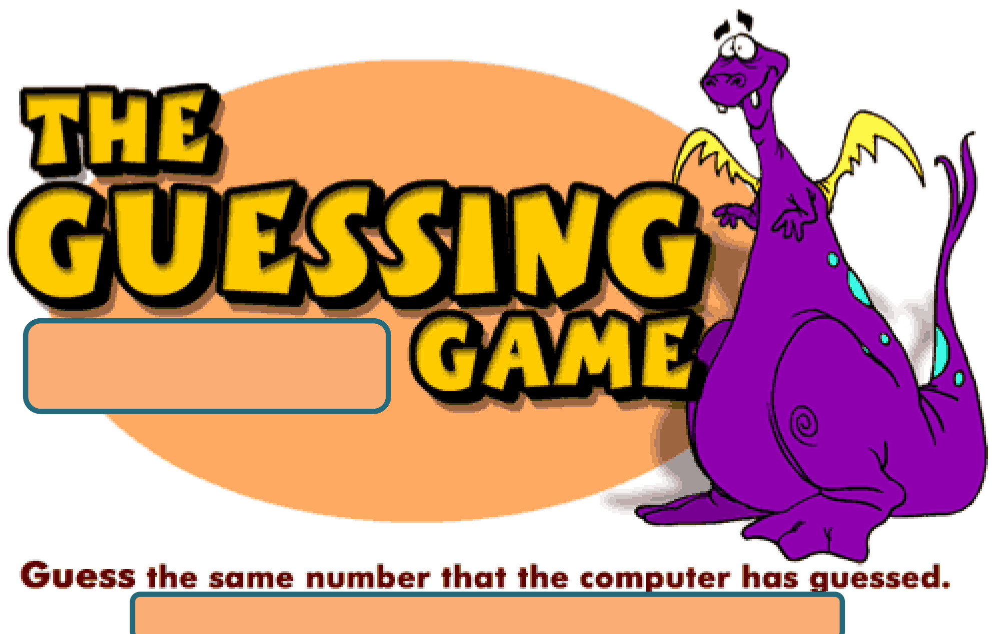


Deep leaning Training Routine

Repeat the following until desired

- ...
- **Optimizer** ("brute force" **searching** action + Strategy)
  - Gradient Descent and Delta rule

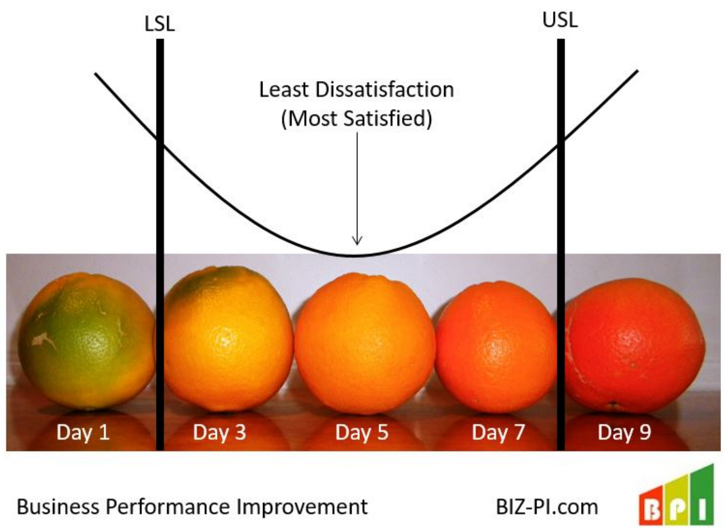
    New weight = Old weight – Derivative Rate * Learning rate
  - learning rate a constant (usually very small)
  - to avoid big steps



Deep leaning Training Routine

"brute force" **searching** action with **strategy** **until desired**

- **Compute and keep the Cost/Loss**
  - Define and compute the ***loss***$(Y_{pred}, Y_{correct})$
- **Back Propagation**
  - Compute the **partial derivatives rate** of ***Loss*** *function* ***wrt*** ***W*** *for all layers*
  - Chain rule applies



- **Optimizer**

New weight = Old weight – Derivative Rate * Learning rate

Optimization

- Example
  - $\boldsymbol{y} = \boldsymbol{wx}$ with *n=5* training examples (forward)
  - Loss function: $L = \frac{\sum(y_p - y_t)}{n} = \frac{\sum(wx - y_t)}{n}$
  - when w randomly initialized as 3 (guessing)

$$\frac{dL}{dw} = \sum_{i=0}^{n} x/\text{n} = 2$$

$$w^{new} = w^{old} - \eta \frac{dL}{dw}$$

$$\eta = 0.5 \quad w_{init} = 3$$

x	*y_pred* (***w=3***) *y=3x*	*y_true*	$y_{pred} - y_{true}$
0	0	0	0
1	3	2	1
2	6	4	2
3	9	6	3
4	12	8	4
Loss	-	-	2

$1st\ iter$ $\quad w^{new} = 3 - 0.5 * 2 = 2$

$\boldsymbol{y} = \boldsymbol{2x}$ Loss function=0

Chain Rule

$$loss(Y_{pred}\ ,\ Y_{correct}) = Y_{pred} - Y_{correct} = x * w_{bp} \quad * w \quad * w_g\ -\ 1$$

$$f_1(x, Wbp) = y_1$$

$$f_2(y_1, W) = y_2$$

$$f_3(y_2, w_g) = y_3$$

$$\frac{\partial Loss}{\partial w_{bp}} = \frac{\partial(y_3)}{\partial y_2}\frac{\partial(y_2)}{\partial y_1}\frac{\partial(y_1)}{\partial w_{bp}}$$

	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

(1,26)

weight	weight
wb1	wp1
wb2	wp2
wb3	wp3
…	…
wb26	wp26

(26,2)

weight	weight	weight
w	w	w
w	w	w

(2,3)

weight
wg1
wg2
wg3

(3,1)

→ y

(1,1)

$$Wbp^{new} = Wbp^{old} - \eta \frac{\partial Loss}{\partial Wbp}$$

Deep leaning Training Routine

Repeat the following until desired

- Initialize weights vector ***W*** for all layers
- **Forward Propagation**
  - *reaching the final layer to get* $\boldsymbol{Y_{pred}}$
- **Compute and keep the cost/loss**
  - Define and compute the ***loss***$(Y_{pred}, Y_{correct})$
- Back Propagation
  - **partial derivatives rate** of ***Loss*** *function* ***wrt W*** *for all layers*
- Optimizer
  - ***New weight = Old weight – Derivative Rate * Learning rate***

Dress Rehearsal

Agenda

- Overview of NLP
- Deep Learning Basics for NLP
- **Workshop: NN Basics on Colab**
- Word2Vec & DL Specifics
  - CBOW and SkipGram
  - ActivationFunction/LossFunction
  - Optimiser/Learning Rate
- Workshop: Word2Vec from Scratch

Word2Vec &DL Specifics

Dr Wang Aobo
aobo.wang@nus.edu.sg

Features from Word Vectors

– **Count from Data**
  – Word Co-occurrence + SVD
  – Count-based model

NMSM
Day 2

– **Learn from Data**
  – CBOW and SKIPGRAM
  – NN Methods
  – Predictive Model

TPML
Day 1

– **Count and Learn from Data**
  – GLOVE: Global Vectors for Word Representation
  – Count + SGD

One-Hot Encoding (Sparse Representation)

Vocabulary of the corpus (big enough)

	give	she	at	talk	have	ramen	a	drink	
give	1	0	0	0	0	0	0	0	
talk	0	0	0	1	0	0	0	0	**v(‘*talk*’)**
have	0	0	0	0	1	0	0	0	
drink	0	0	0	0	0	0	0	1	
a	0	0	0	0	0	0	1	0	
at	0	0	1	0	0	0	0	0	

Lookup Function

v(‘*talk*’) | | v(‘*talk*’) Embedded

$$\begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}^T \bullet \begin{bmatrix}0.1 & 0.2 & -0.4 & 0.9\\ 0.2 & 0.1 & -0.3 & 0.9\\ 0.2 & -1.4 & 0.3 & -0.1\\ 0.3 & -2.0 & 0.5 & -0.5\\ 0.2 & -1.1 & 0.3 & -0.7\\ 0.9 & -1.3 & 0.4 & -0.9\\ 0.3 & -3.0 & 0.5 & -0.2\\ 0.5 & -0.1 & 0.2 & 0.1\end{bmatrix} = \begin{pmatrix}0.3\\-0.2\\-0.5\\-0.5\end{pmatrix}^T$$

One-Hot Encoding | **Word Embeddings** | **Input**

1 x |V| | |V| x d | 1 x d

Learn the Matrix through Making Prediction

39

Word2Vec (CBOW)

Learn the Matrix through "classification" task

Sentence: the bulk of linguistic questions concern the dis- tinction between a and m. a linguistic account of phenomenon …

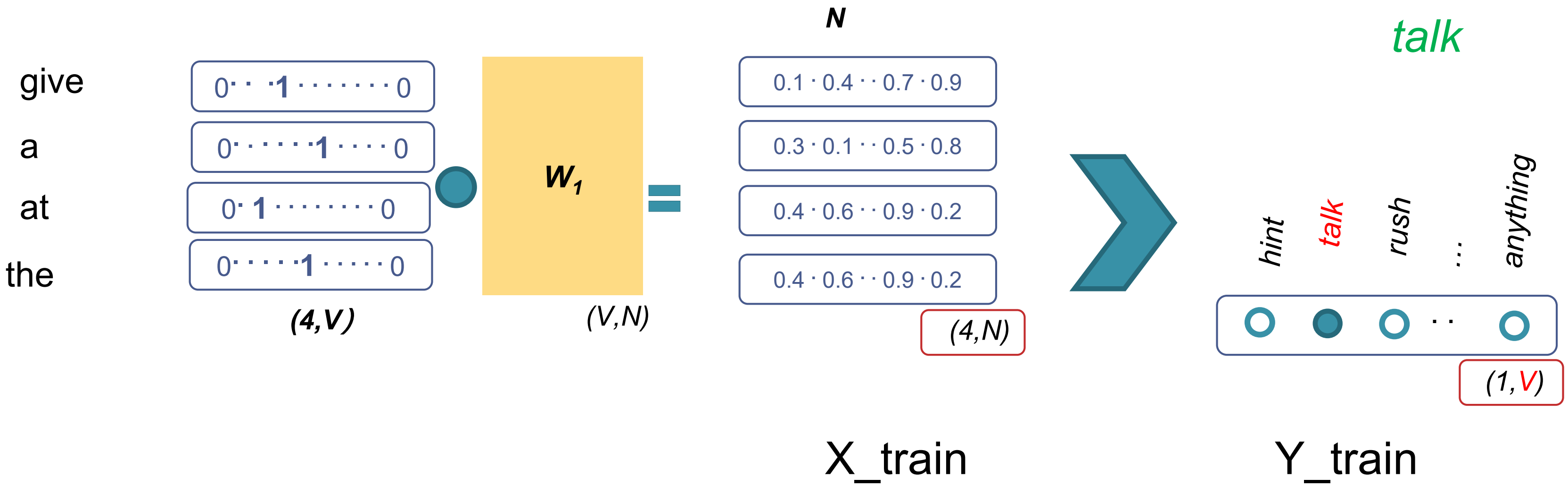
of	the bulk ______ linguistic questions
linguistic	bulk of ______ questions concern
questions	of linguistic ______ concern the
concern	linguistic questions ______ the dis-
the	questions concern ______ dis- tinction
dis-	concern the ______ tinction between
tinction	the dis- ______ between a
between	dis- tinction ______ a and
a	tinction between ______ and m.
and	between a ______ m. a
m.	a and ______ a linguistic
a	and m. ______ linguistic account
linguistic	m. a ______ account of
account	a linguistic ______ of a
of	linguistic account ______ a phenomenon
a	account of ______ phenomenon gen-
phenomenon	of a ______ gen- erally

window_size = 2

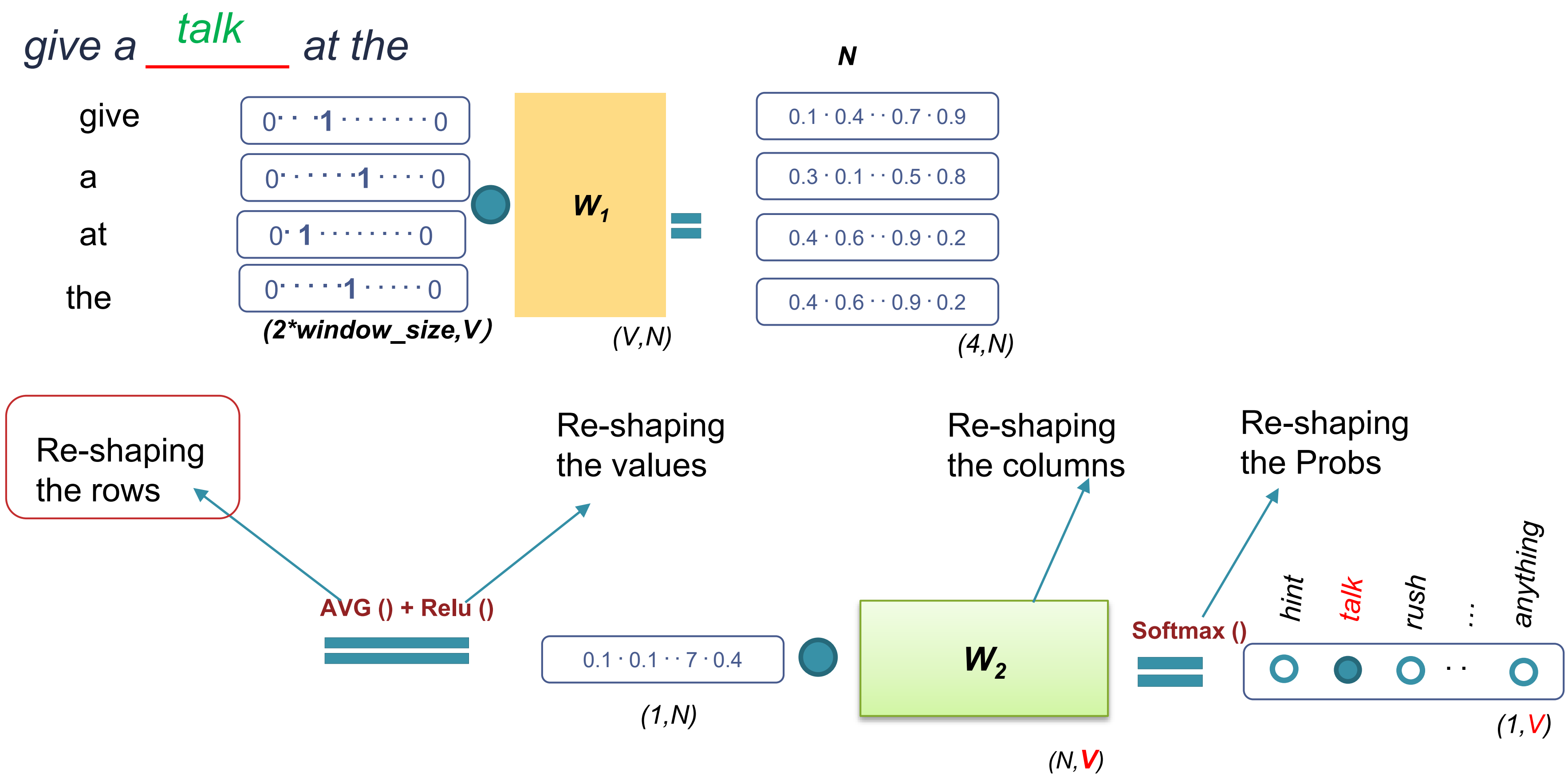
Word2Vec (CBOW)

Task: Iterate through every word with a given window; learn W such the models can predict what's the word given only the context words as inputs.

give a ______ at the

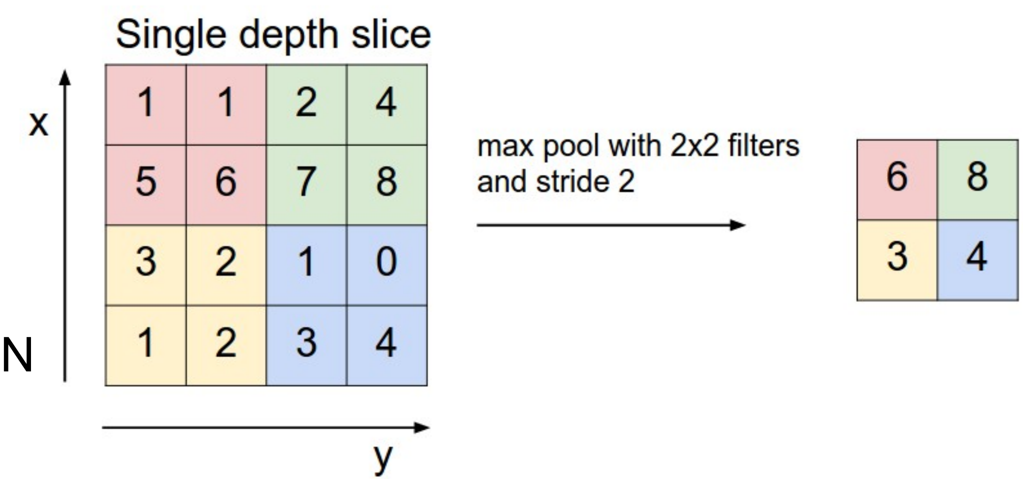


Word2Vec (CBOW)



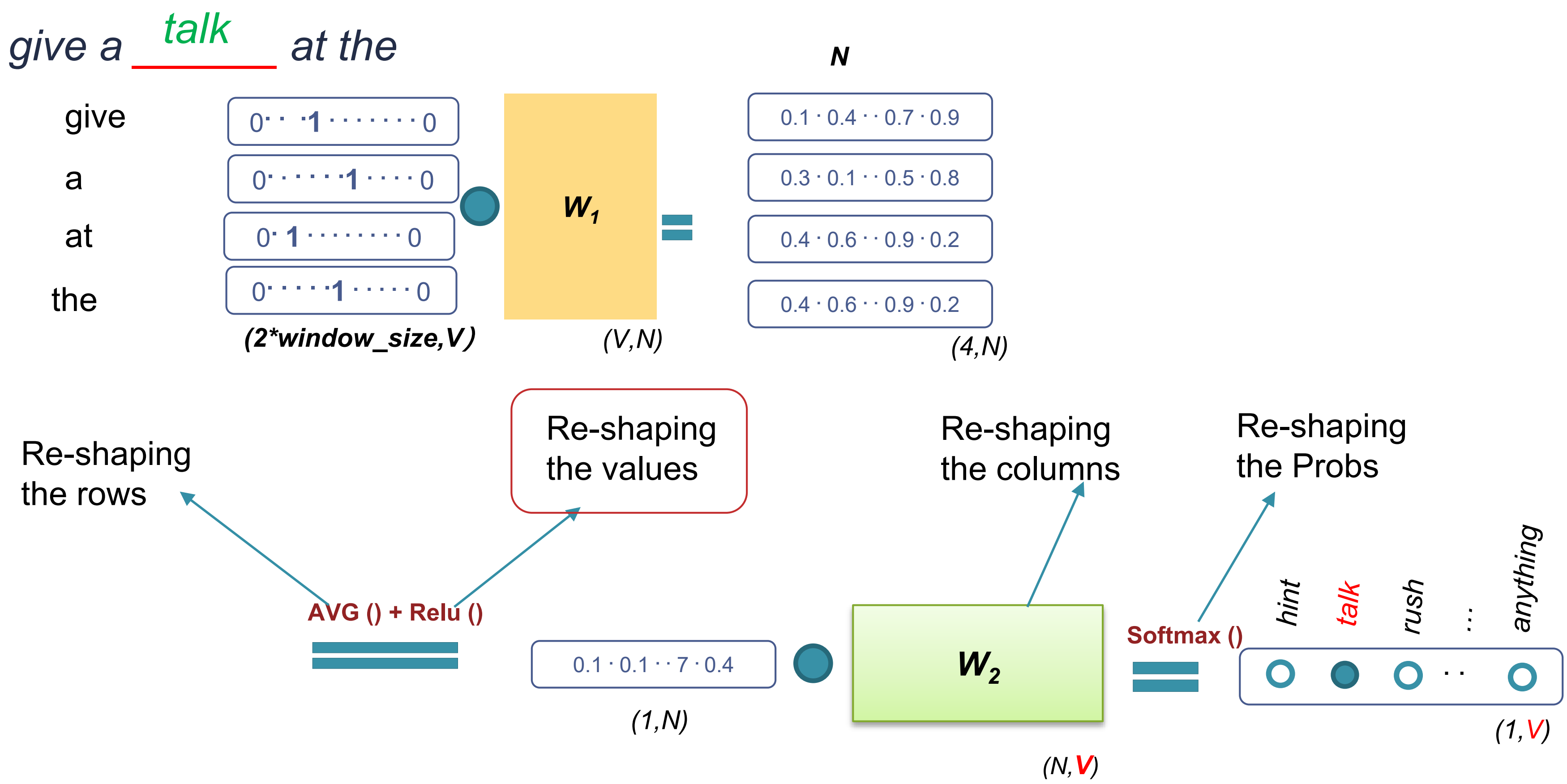
Naïve Sentence Embedding

v(‘*give*’)	0.5	-1.3	0.6	1.1
v(‘*a*’)	0.3	-0.2	0.7	0.5
v(‘*at*’)	0.3	2.3	-0.8	1.0
v(‘the’)	1.7	-0.2	-0.1	1.0
	AVG()/MAX()/MIN()/Concat()			
Sent0 (“give a at the”)	0.7	0.2	0.1	0.9



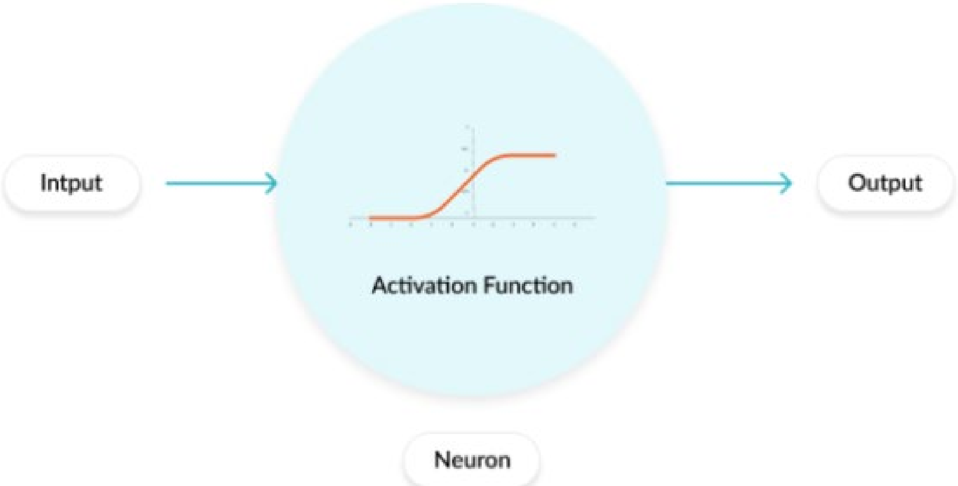
Pooling Layer in CNN

Word2Vec (CBOW)



Activation Function

- **Activation Function**
  - Smoother decision function is expected
  - Activations bound in (0,1)
  - Support backpropagation
  - Need to be non-linear

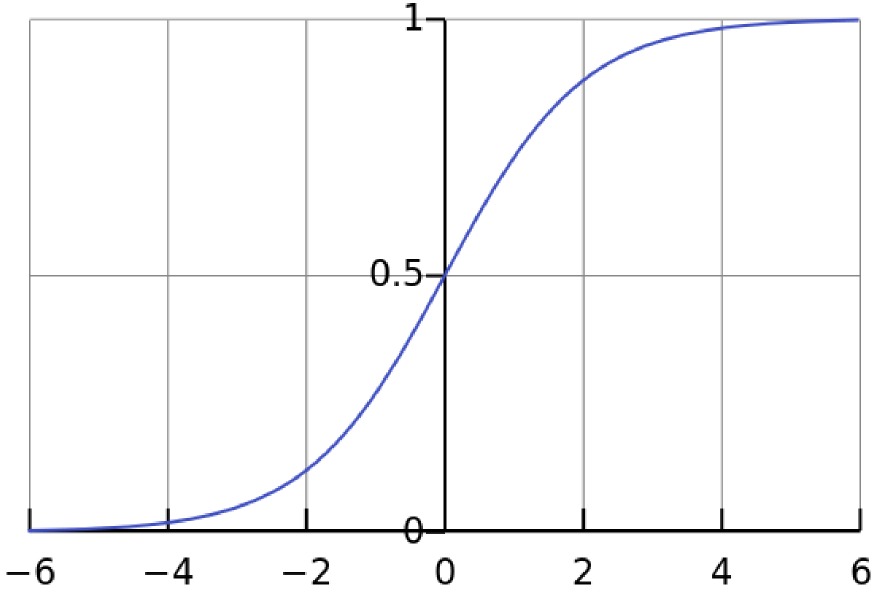


https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

Activation Function

- **Sigmoid** function
  - smooth output between 0 and 1
  - interpreted as a probability of "Yes"

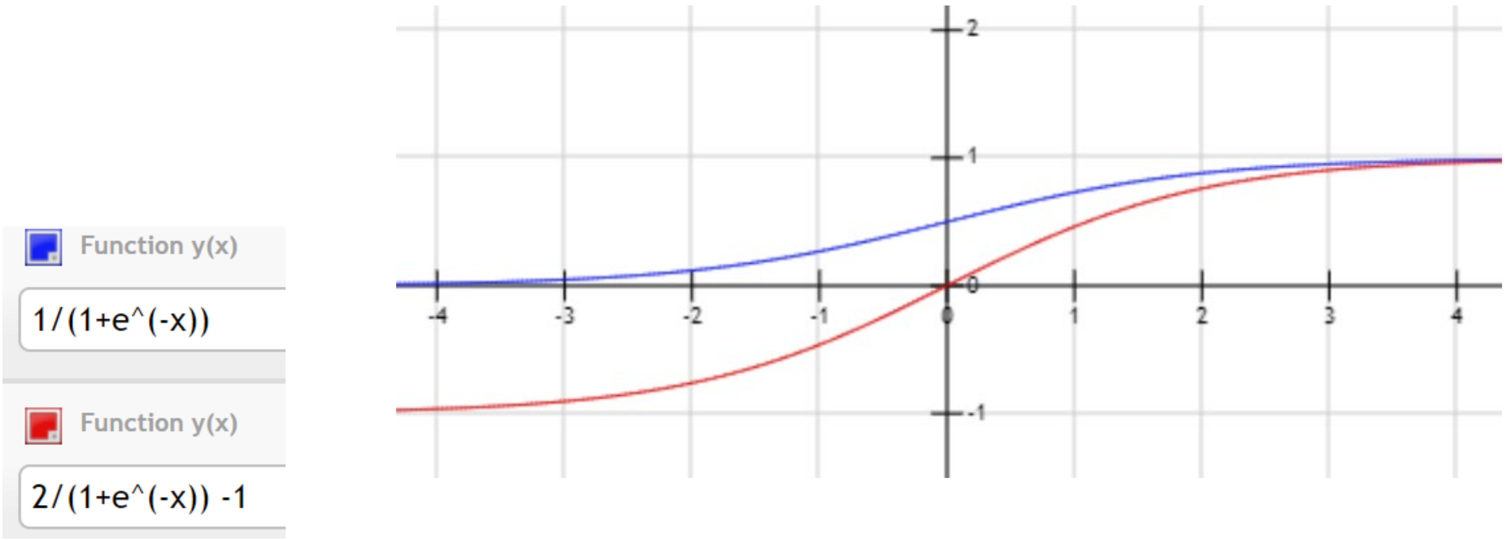
$$S(x) = \frac{1}{1 + e^{-x}}$$



(Image from Wikipedia)

Activation Function

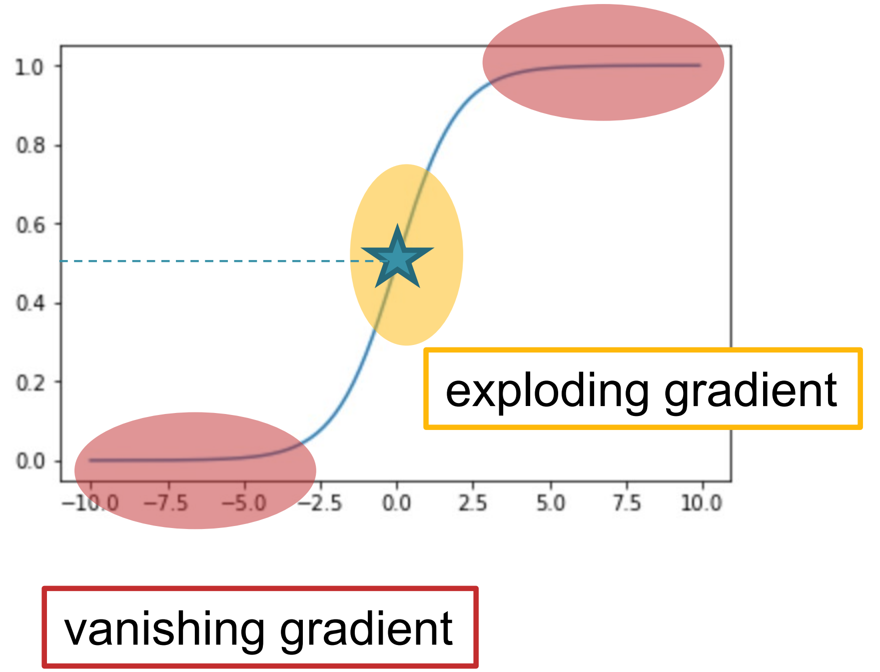
- Tanh Function
  - smooth output between -1 and 1
  - 0 centroid



$$tanh(x) = 2\, sigmoid(2x) - 1$$

Activation Function (Sigmoid)

```python
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1/(1+np.exp(-x))

# Generate points from -10 to +10,
# in steps of 0.1
x = np.arange(-10, 10, 0.1)
y = sigmoid(x)

# Plot the graph.
plt.plot(x, y)
plt.show()
```

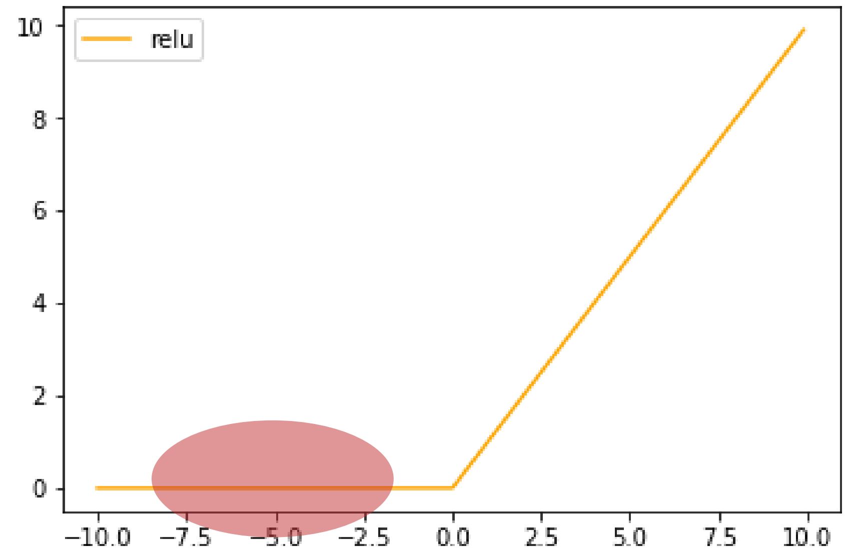


Activation Function (ReLU)

- **Intuitions can be useless for NN**
  - Smoother decision function is expected
  - Activations bound in a range
  - “0” centroid

- **All you need is just**
  - Simple
  - Fast

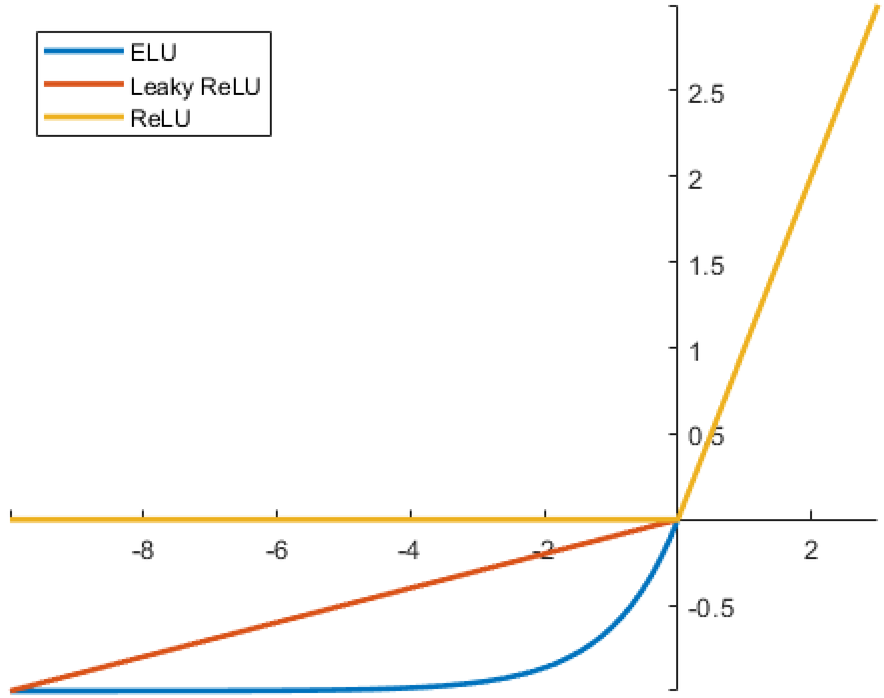
$$A(x) = \max(0,x)$$



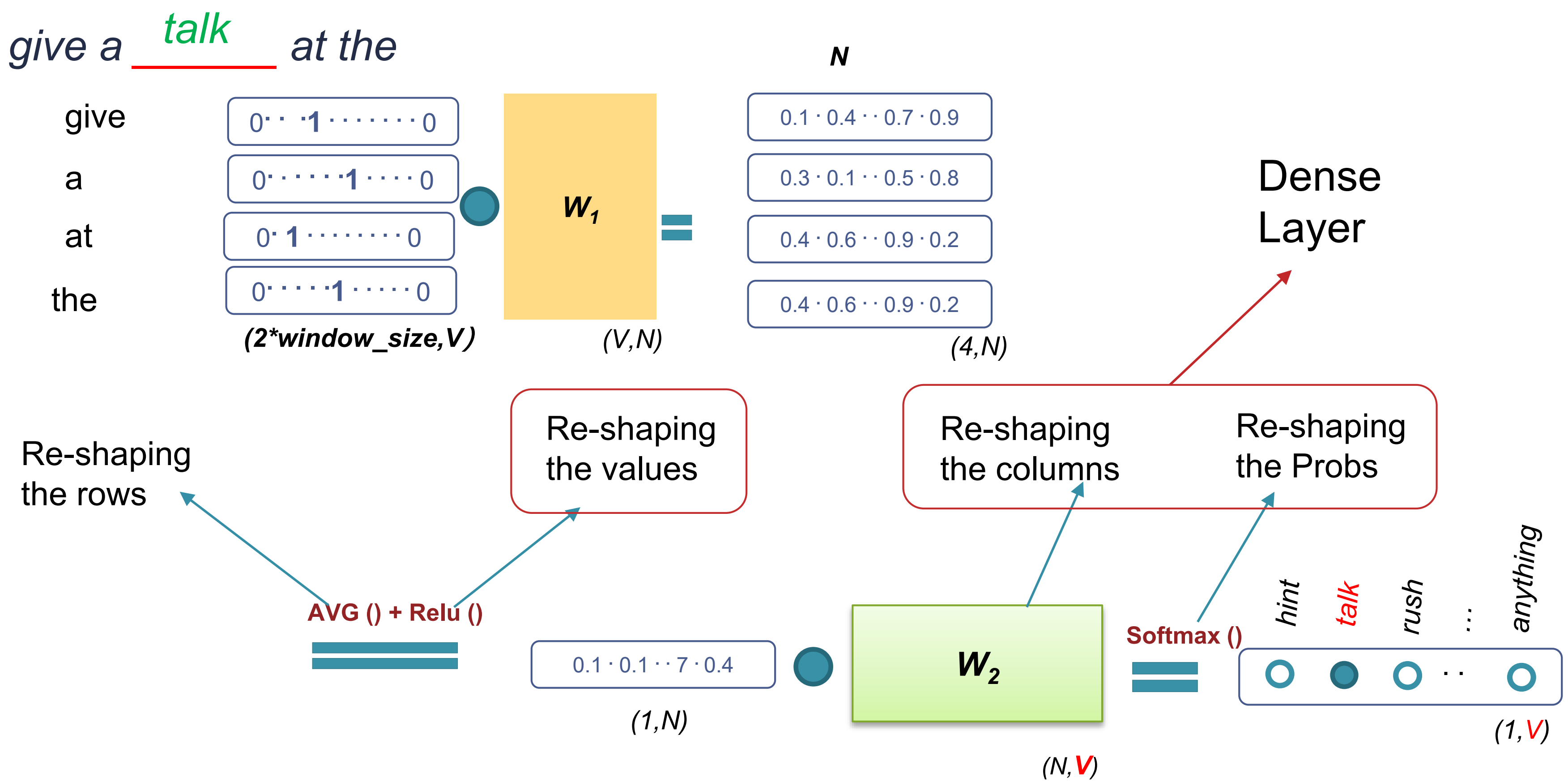
vanishing gradient

Activation Function (ReLU)

- ReLU’s Family



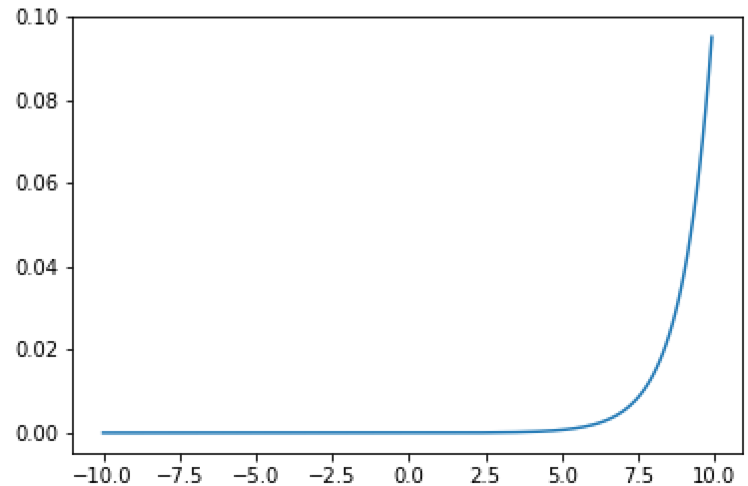
Word2Vec (CBOW)



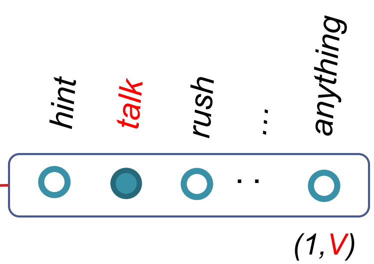
Last Layer Activation Function

- **SoftMax**

```
import numpy as np
import matplotlib.pyplot as plt

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)

x = np.arange(-10, 10, 0.1)
y = softmax(x)

plt.plot(x,y)
plt.show()
```

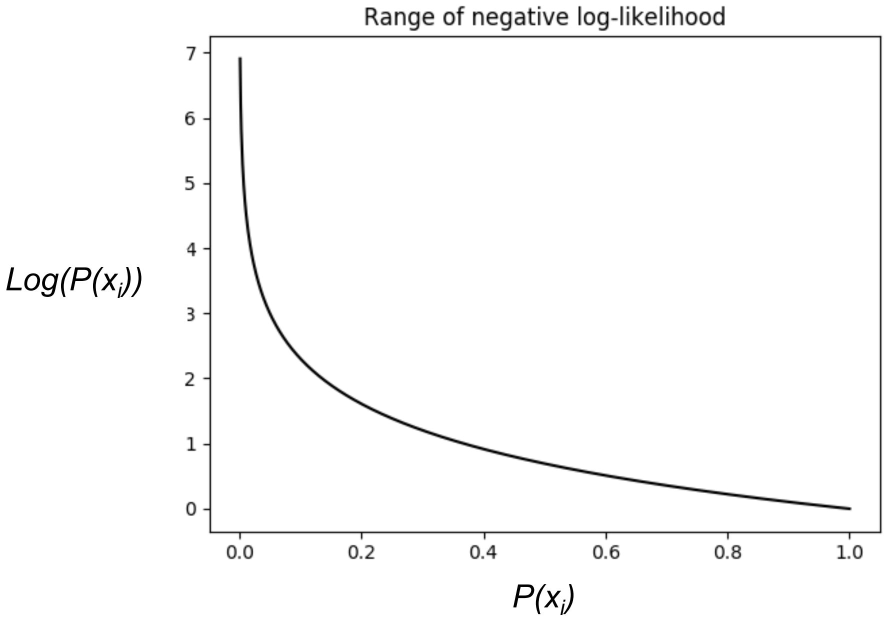


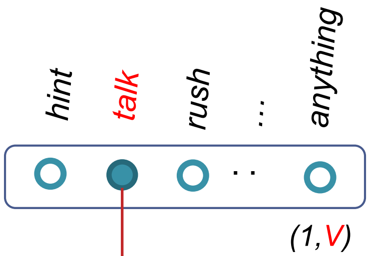
Vector of x_i as Input
Never be a single x



Last Layer Activation Function

- **NegLogSoftMax**

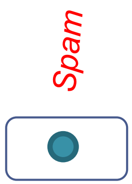




Max { $p(x_i)$ } as input
Never be a vector of $p(x_i)$

Last Layer Activation Function

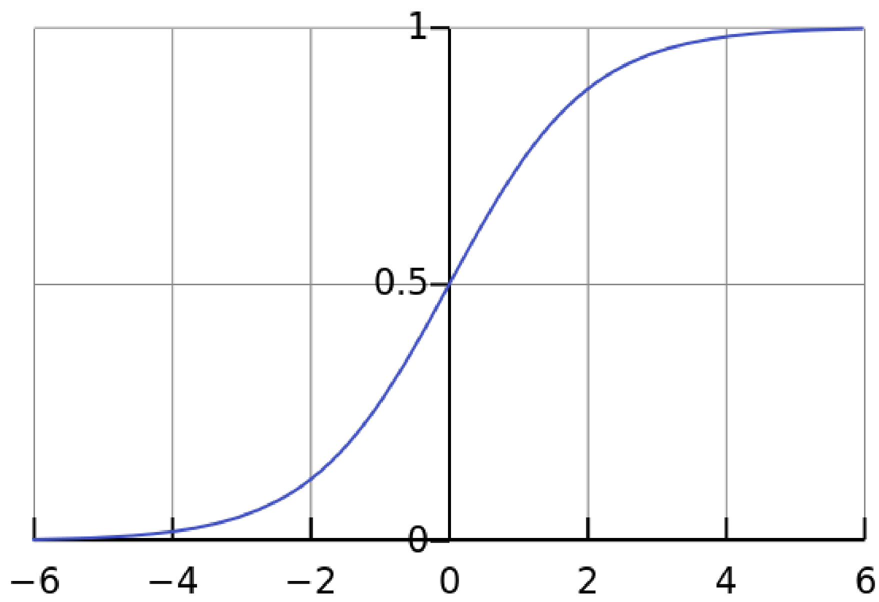
- **Sigmoid** function
  - smooth output between 0 and 1
  - interpreted as a probability of "Yes"



(1,1)

A single x as input
Never be a vector of x_i

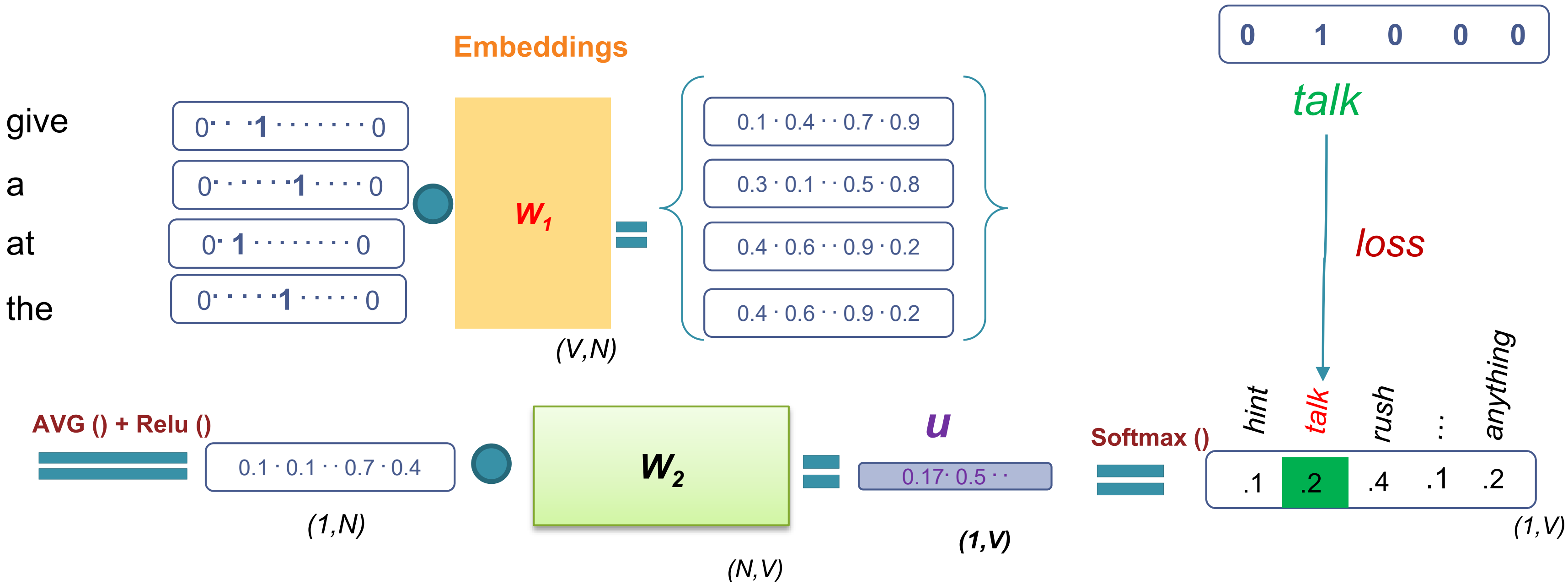
$$S(x) = \frac{1}{1+e^{-x}}$$



(Image from Wikipedia)

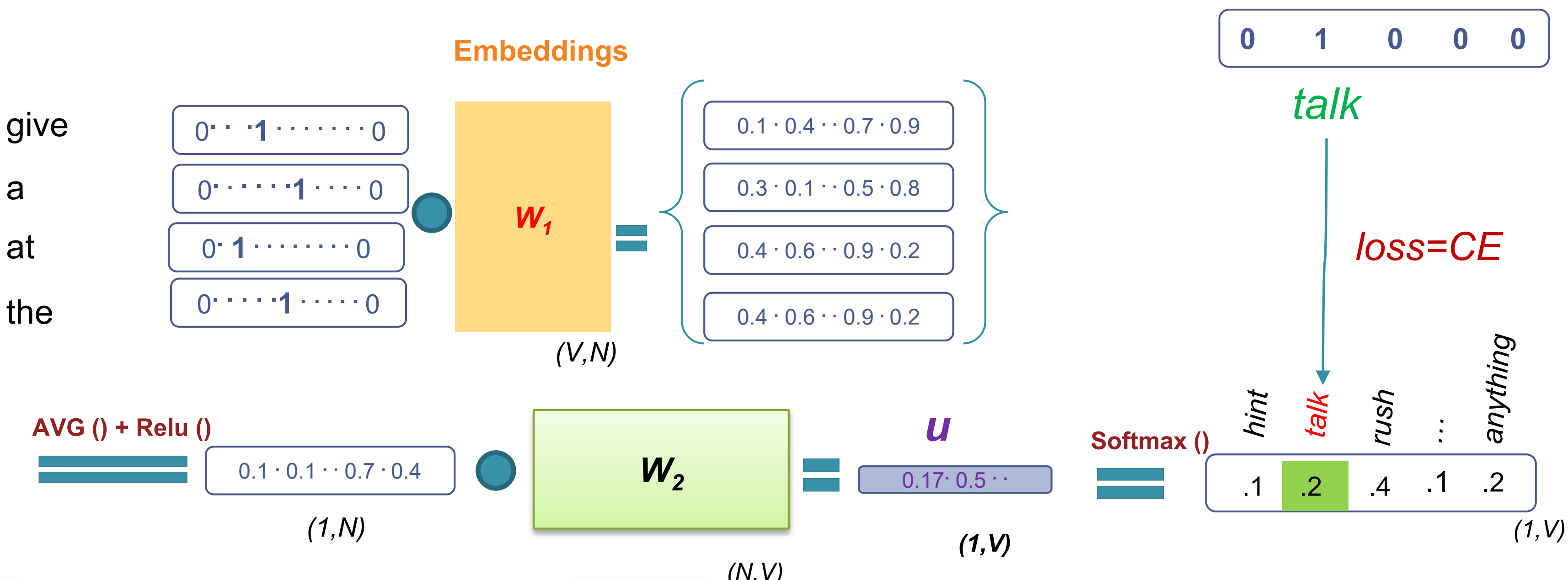
Word2Vec (CBOW)

Loss Function



Loss Function

(Categorical) Cross Entropy Loss Function



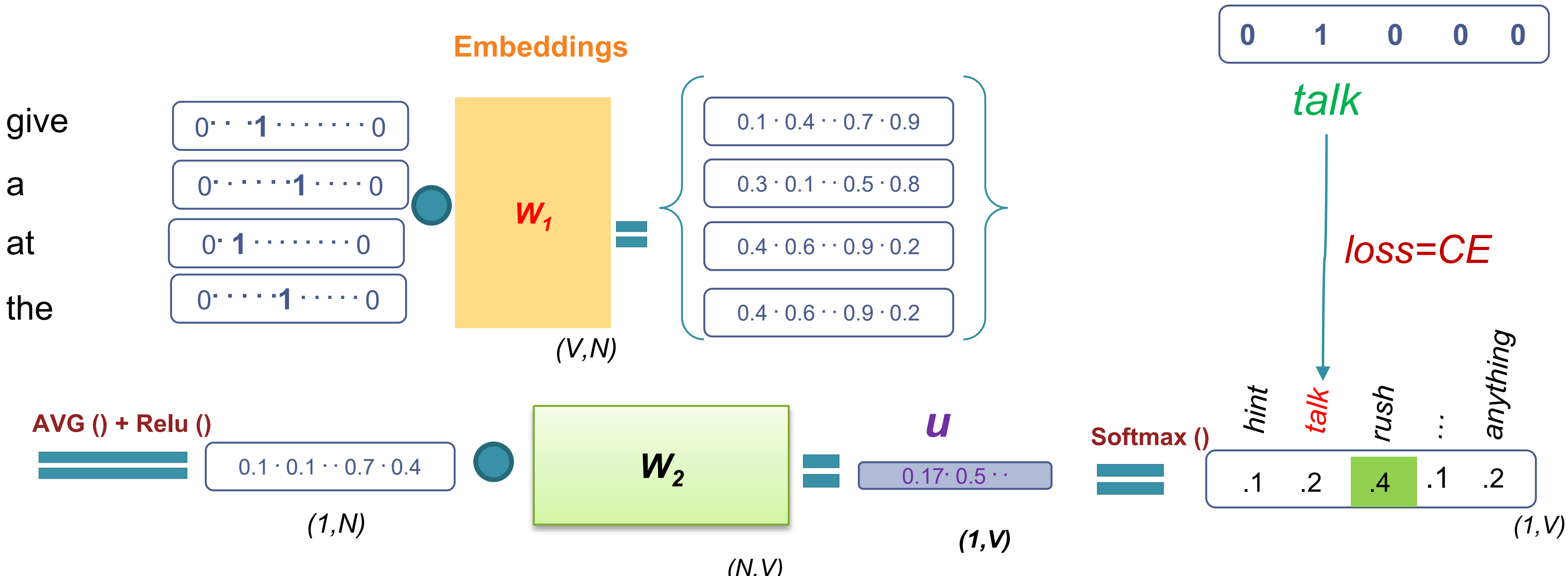
$$\textbf{Loss} = -\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Talk
- 0*log(0.1) -1*log(0.2) - 0*log(0.4) – 0*log(0.1) – 0*log(0.2)

(Categorical) Cross Entropy Loss Function



$$\mathbf{Loss} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Note

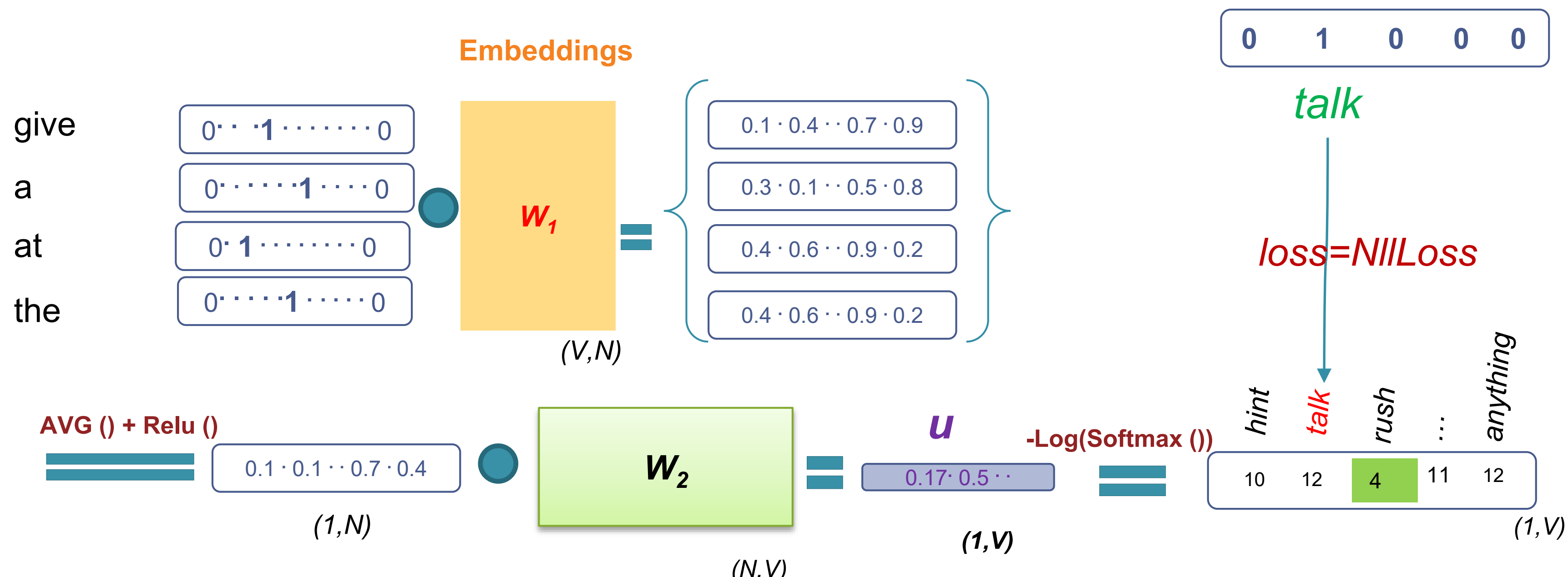
- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

$- \sum \{$

Talk	give	rush	....
-1*log(0.2)	-1*log(0.2)	-1*log(0.2)	.
-1*log(0.3)	-1*log(0.3)	-1*log(0.3)	.
-1*log(0.1)	-1*log(0.0)		.
-1*log(0.0)	-1*log(0.1)		
	-1*log(0.1)		
	-1*log(0.1)		

$\}$

(Categorical) Negative Log Loss Function



$$\text{Loss} = -\sum_{c=1}^{M} y_{o,c} l_{o,c}$$

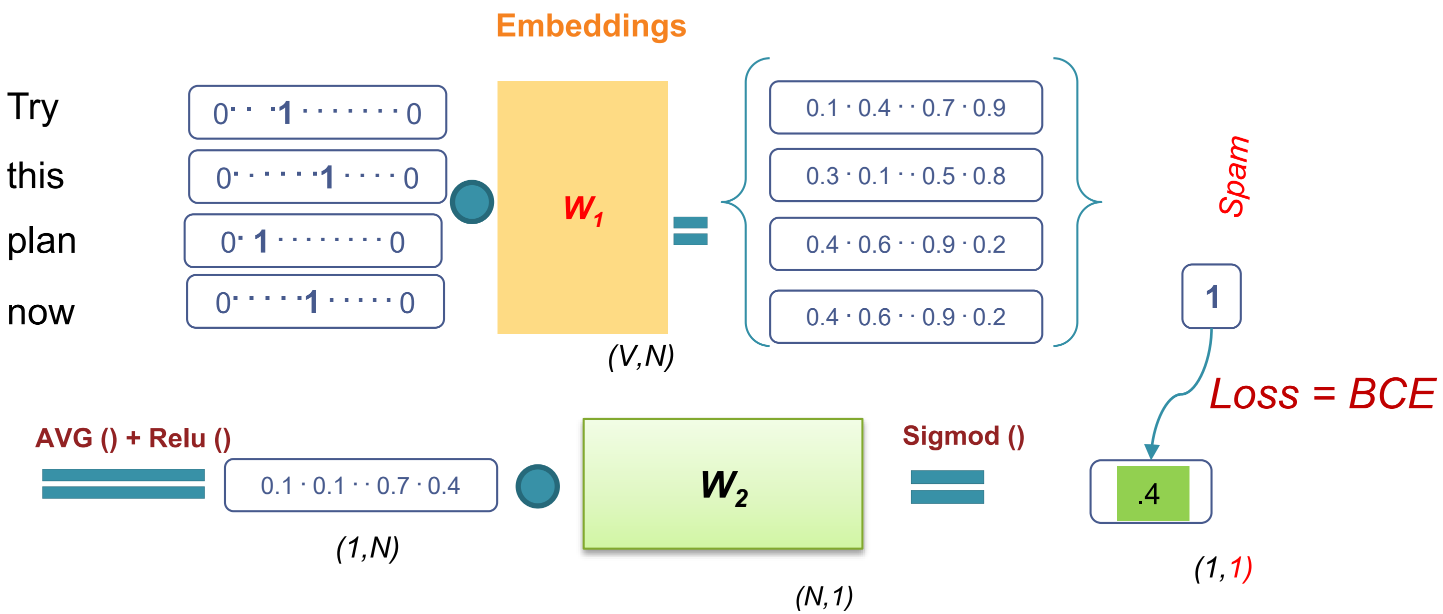
Note

- M - number of classes (dog, cat, fish)
- ~~log - the natural log~~
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

$$-\sum \left\{ \begin{array}{cccc} \text{Talk} & \text{give} & \text{rush} & \cdots \\ -1^*\log(0.2) & -1^*\log(0.2) & -1^*\log(0.2) & \vdots \\ -1^*\log(0.3) & -1^*\log(0.3) & -1^*\log(0.3) & \\ -1^*\log(0.1) & -1^*\log(0.0) & & \\ -1^*\log(0.0) & -1^*\log(0.1) & & \\ & -1^*\log(0.1) & & \\ & -1^*\log(0.1) & & \end{array} \right\}$$

Not For CBOW

Binary Cross Entropy Loss Function

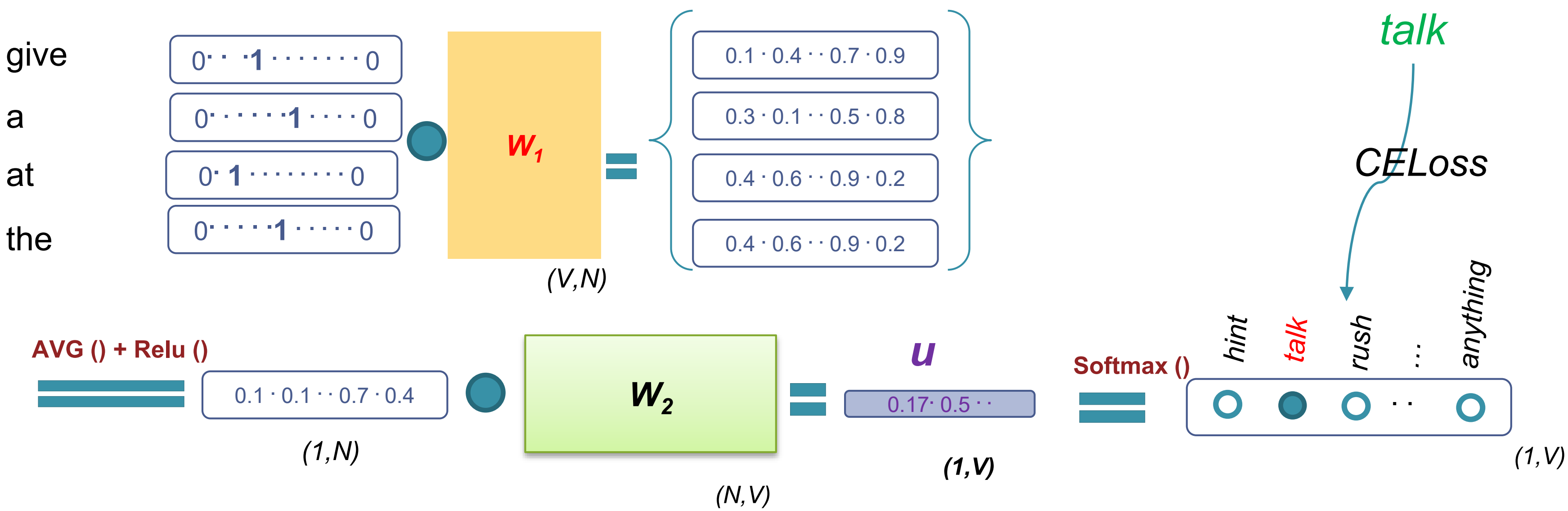


$$BCE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) - (1-t_1) log(1-s_1)$$

$$-\sum \left\{ \begin{array}{ll} \text{spam} & \text{not spam} \\ -1*\log(0.4) & -(1-0.4)*\log(1-0.4) \\ -1*\log(0.3) & -(1-0.3)*\log(1-0.3) \\ -1*\log(0.1) & -(1-0.1)*\log(1-0.1) \\ -1*\log(0.0) & -(1-0.0)*\log(1-0.0) \end{array} \right\}$$

Word2Vec (CBOW)

Backpropagation SGD



$$W_1(new) = W_1(old) - \frac{\partial lost}{\partial w_1} * lr$$

$$W_2(new) = W_2(old) - \frac{\partial lost}{\partial w_2} * lr$$

Backpropagation

SGD vs Adam

Faster
but sometimes
not converging

```
# Vanilla SGD
x += - learning_rate * dx
```

x is a vector of parameters and
dx is the gradient

Well generalised
but slower
Used together with
momentum

```
# Adam
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m /
        (np.sqrt(v) + eps)
```

x is a vector of parameters and
dx is the gradient
m is the smoothen gradient
v is the 'cache' used to normalize **x**
eps is smoothing term (1e-4 to 1e-8)
beta1,beta2 are hypers (0.9, 0.999)

Backpropagation

- **Learning Rate**
  - Fixed Learning Rate based on Experience
    - Vanilla model: MLP/RNN/CNN **10^{-2} ~10^{-3}**
    - Typical Models: LSTM/CNN **10^{-3}**
    - Complex Models: BERT **10^{-5}**

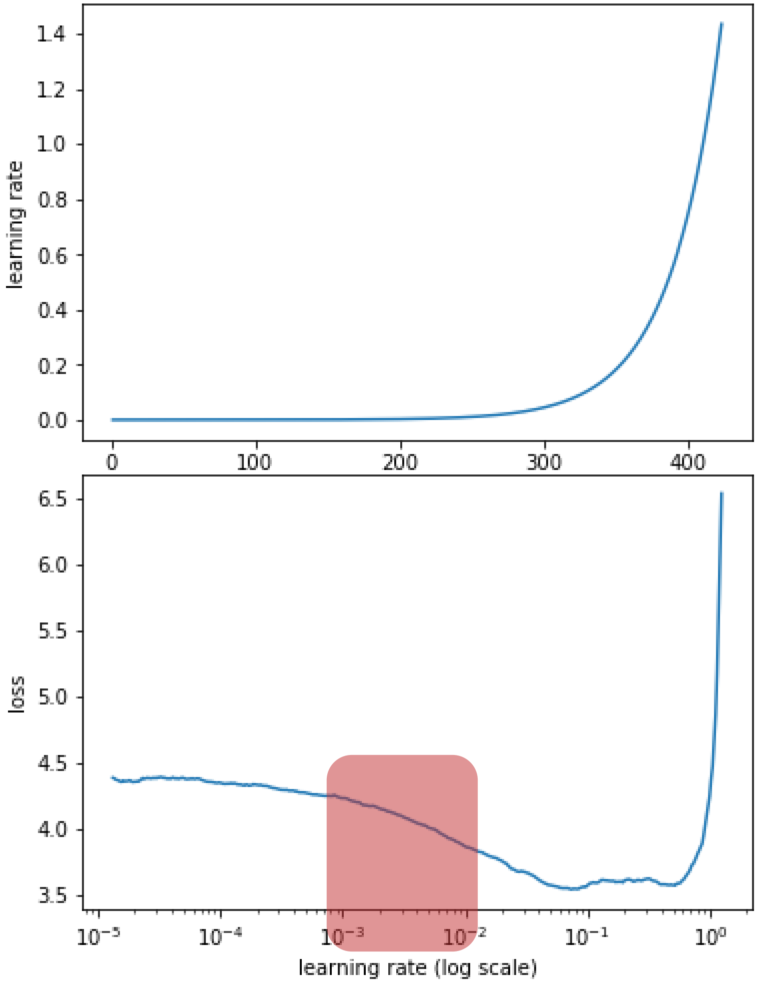
Backpropagation

- **Fixed Learning Rate**
  - Estimate Fixed Learning Rate

Increase ***lr*** as iterations going up

save and plot the *loss* as per *lr*

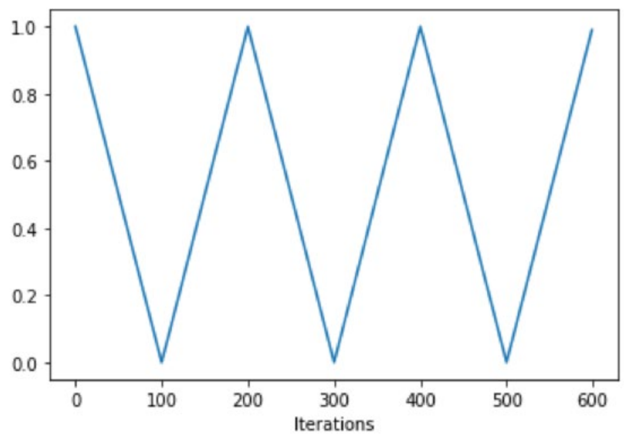
get the ***lr***
fastest decrease in the loss



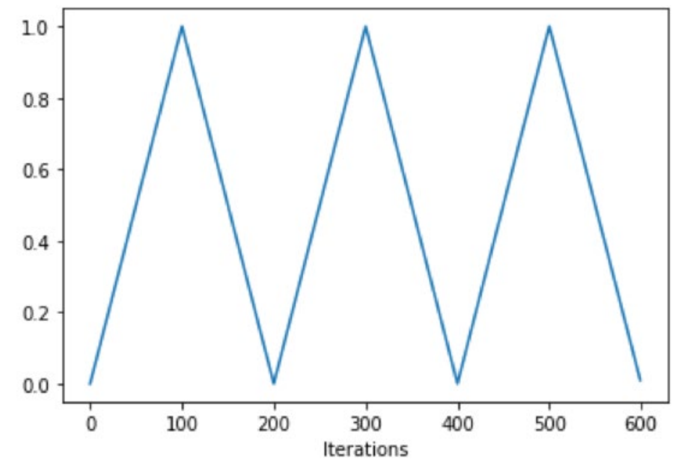
Backpropagation

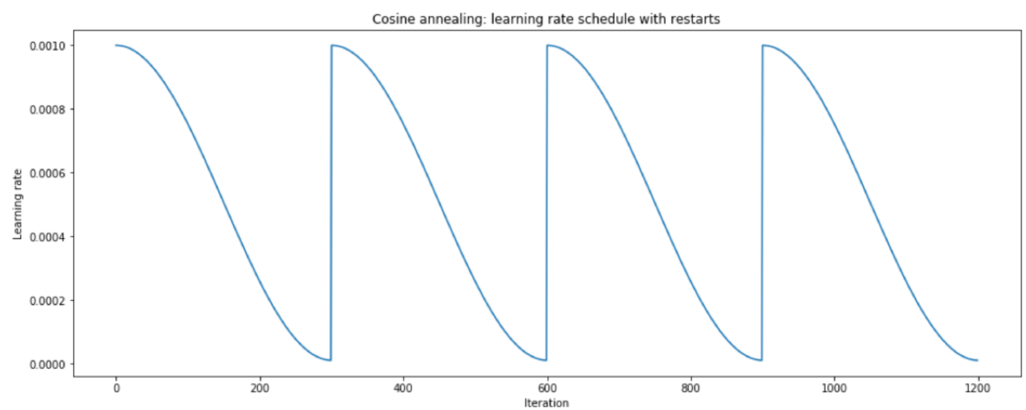
- **Dynamic Learning Rate**
  - Estimate Initial Learning Rate
  - Program Dynamic Learning Rate

Function: $|\frac{iterations}{step\_size} - 2(cycle) + 1|$



Function: $1 - |\frac{iterations}{step\_size} - 2(cycle) + 1|$

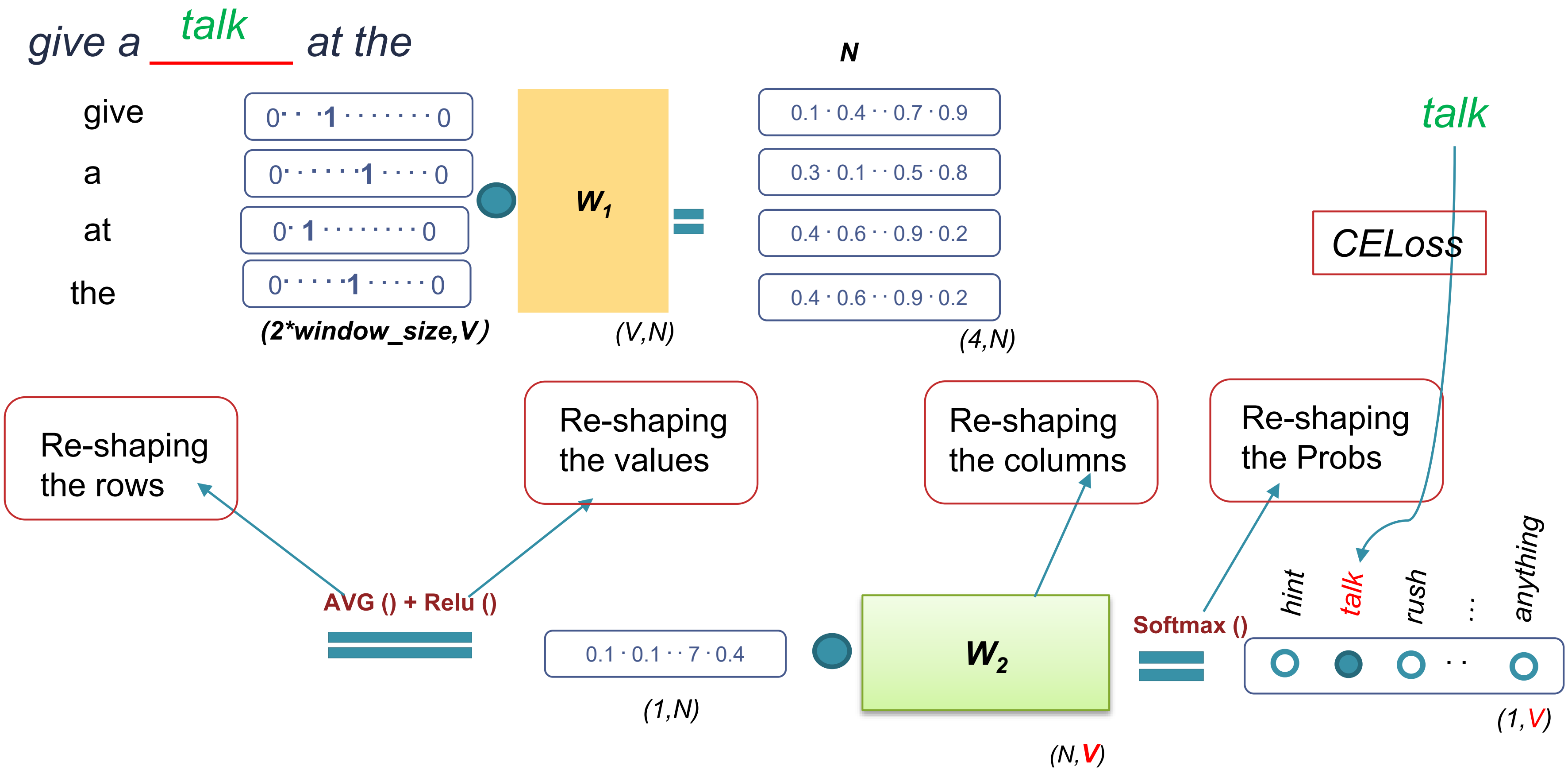




$$\eta_t = \eta^i_{\min} + \frac{1}{2}(\eta^i_{\max} - \eta^i_{\min})\left(1 + \boxed{\cos\left(\frac{T_{current}}{T_i}\pi\right)}\right)$$

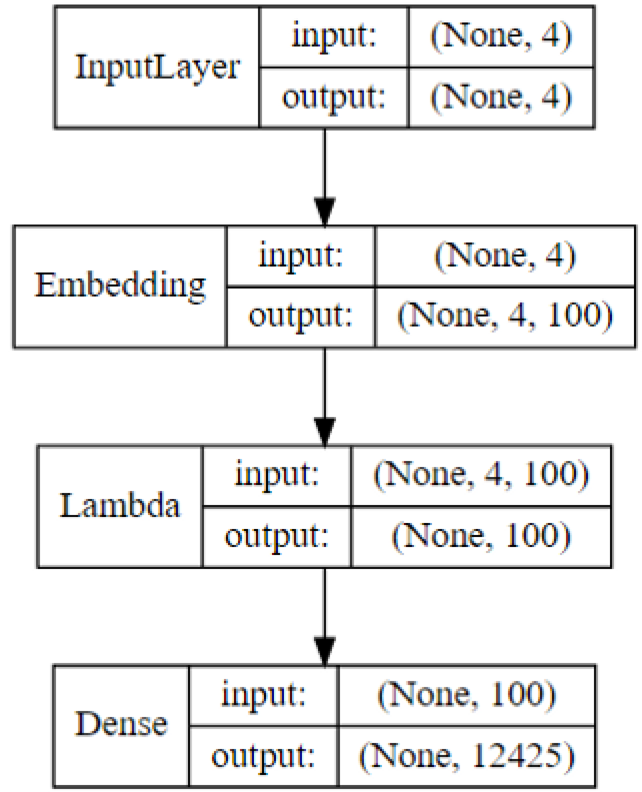
Annealing, i.e. taking a partial step

Word2Vec (CBOW)



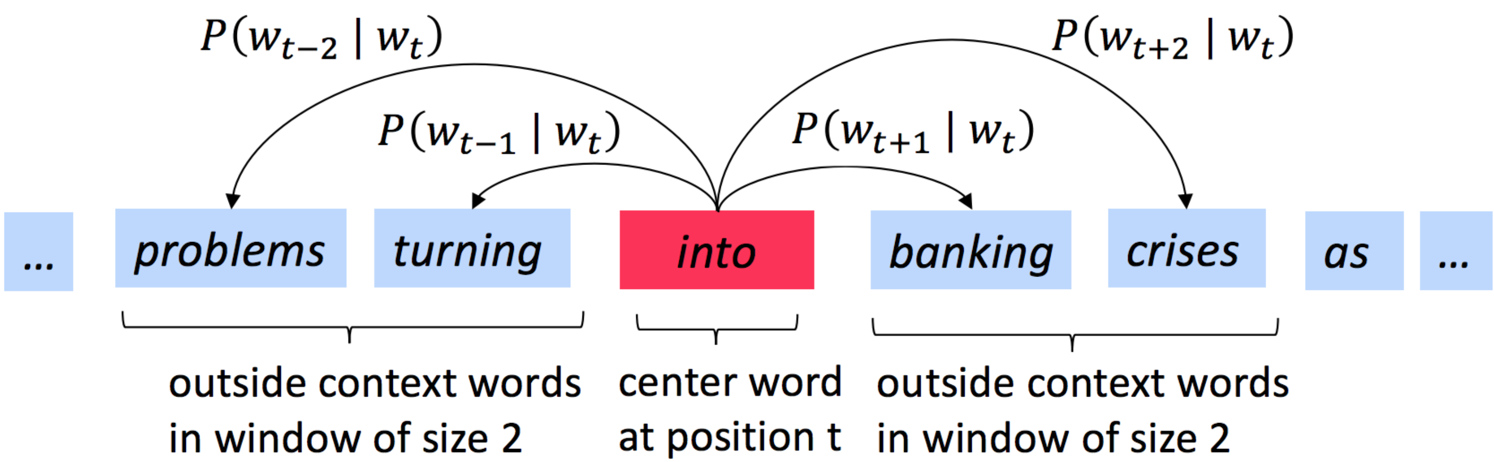
CBOW Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            1242500
_________________________________________________________________
lambda_1 (Lambda)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 12425)             1254925
=================================================================
Total params: 2,497,425
Trainable params: 2,497,425
Non-trainable params: 0
_________________________________________________________________
```



Word2Vec (Skipgram)

Task: Iterate through each word with a given window; for each word predict the context words within the window



(E.g. from Manning (2018) Stanford cs224n course)

Word2Vec (Skipgram)

Sentence: `language users never choose words randomly , and language is essentially non-random .`

In-/Outputs:

```
[
    ('never',    ['language', 'users', 'choose', 'words']),
    ('choose',   ['users', 'never', 'words', 'randomly']),
    ('words',    ['choose', 'words', ',', 'and']),
    ('randomly',['words', 'randomly', 'and', 'language'])
]
```

Word2Vec (Skipgram)

Sentence: language users never choose words randomly , and language is essentially non-random .

Windows:

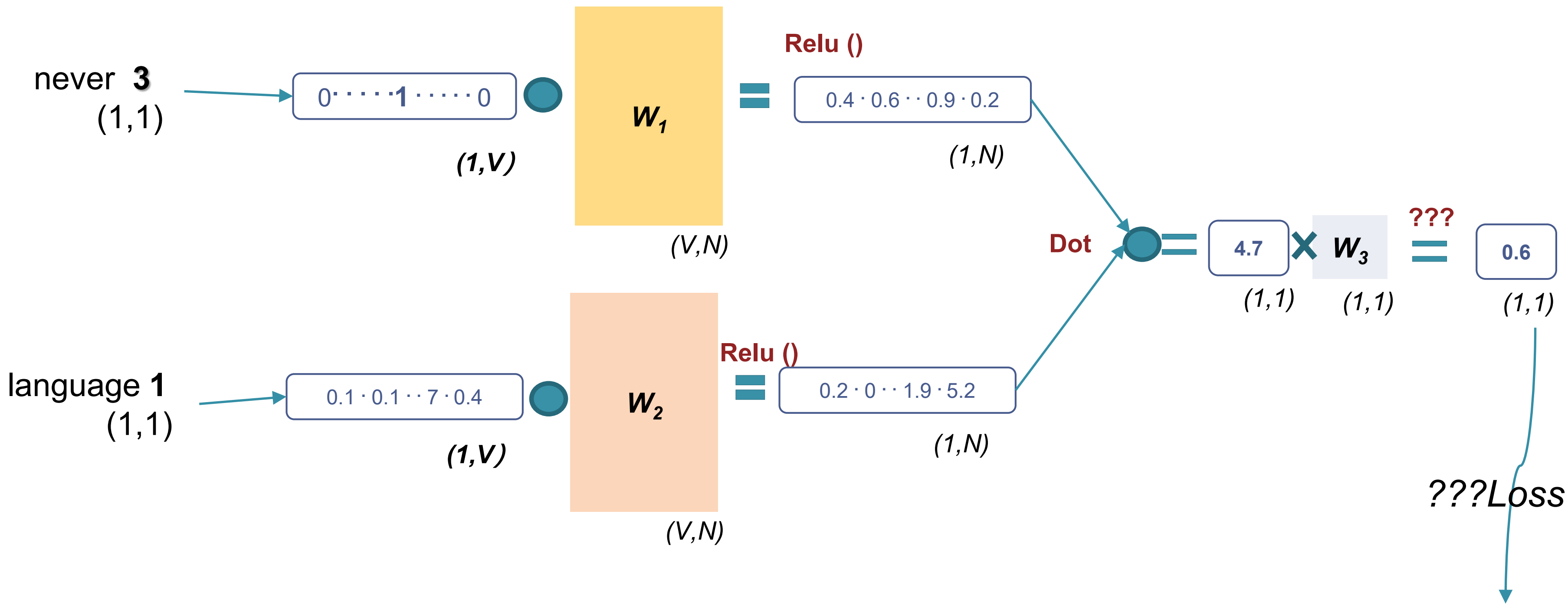
```
['language', 'users', 'never', 'choose', 'words']

('never', 'language', 1),
('never', 'users', 1),
('never', 'choose', 1),
('never', 'words', 1),
('never', ',', 0),
('never', 'non-random', 0),
('never', 'is', 0),
('never', 'is', 0)
```

aka. negative sampling

Word2Vec (Skipgram)

Sentence: language (1) users (2) never (3) choose (4) words (5) randomly (6) , (7) and (8) language (1) is (10) essentially (11) non-random (12) . (13)

['language', 'users', 'never', 'choose', 'words']

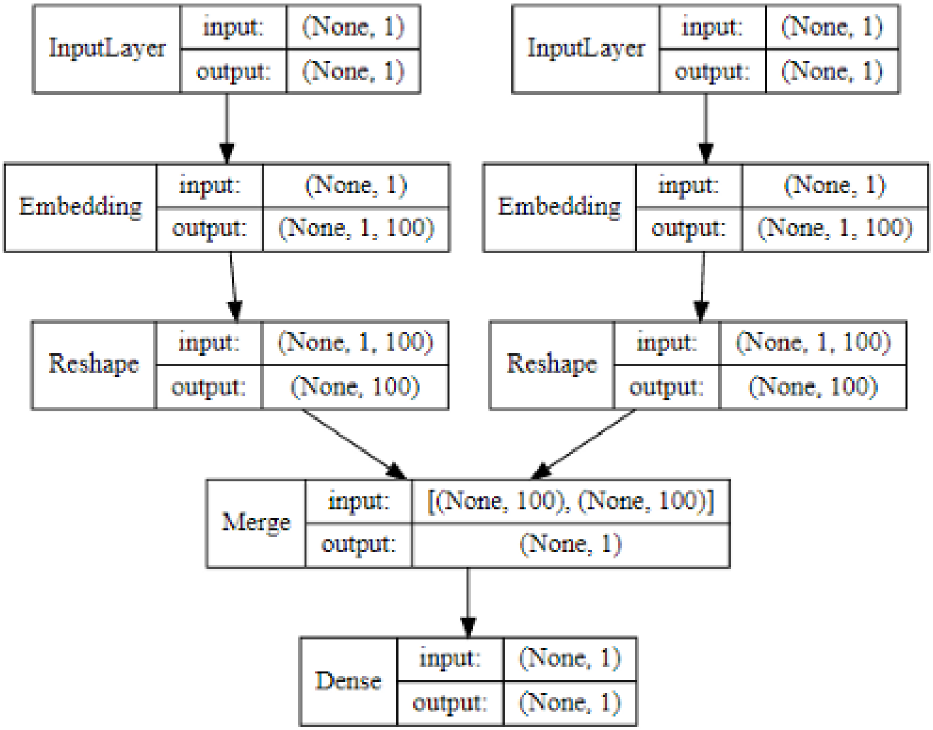


('never', 'language', +1)

Word2Vec (Skipgram)

- **Skipgram Model**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
merge_2 (Merge)              (None, 1)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 2
=================================================================
Total params: 2,485,002
Trainable params: 2,485,002
Non-trainable params: 0
_________________________________________________________________
```



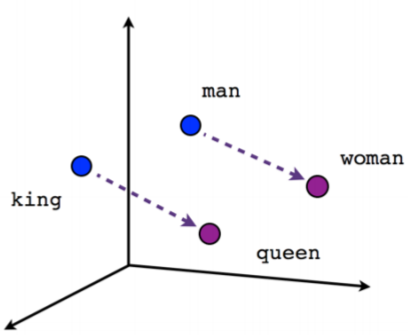
How to Choose Context?

- **Different contexts lead to different embeddings**

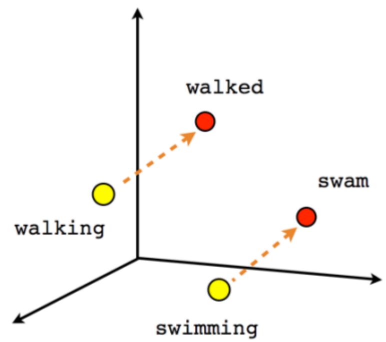
- **Small context window:** more syntax related
  - *I like…*
  - *She likes …*

- **Large context window:** more semantics related
  - ***stackoverflow*** *great* ***website*** *for* ***programmers***

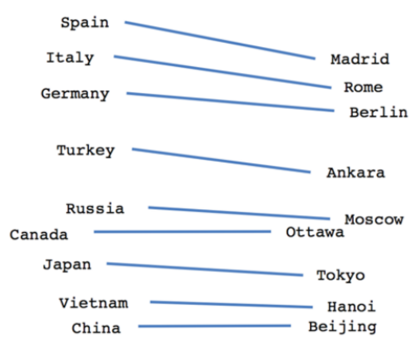
Properties of Word Embeddings



Male-Female



Verb tense



Country-Capital

Ingredients

Corpus of text	As large as possible
Annotations	0
Initialize weights (aka Embeddings)	1x per word
Deep Learning Model	1x
Cost Function	Appropriately
GPU	Lotsa of it

When to use pre-trained embeddings?

- Generally, when you don’t have much training/annotated data
- **Useful:** Use as inputs to model for classification task, e.g. tagging, parsing, ranking (based on similarity)
- **Less Useful:** Machine Translation / Sequence generating tasks
- **Not Useful:** Generic Language Modeling, for those, we have sentence embedings…

Limitations

- **Sensitive to "tokens"** (cat vs cats)
- **Inconsistent across space**, embeddings for the same words trained with different data are different
- **Can encode bias** (stereotypical gender roles, racial bias)
- **Not interpretable**

Embedding Bias

$$\overrightarrow{\mathrm{man}} - \overrightarrow{\mathrm{woman}} \approx \overrightarrow{\mathrm{king}} - \overrightarrow{\mathrm{queen}}$$

$$\overrightarrow{\mathrm{man}} - \overrightarrow{\mathrm{woman}} \approx \overrightarrow{\mathrm{computer\ programmer}} - \overrightarrow{\mathrm{homemaker}}.$$

	"He" Occupations	"She" Occupations
Cosine Similarity	*["retired", "doctor", "teacher", "student", "miller", "assistant", "lawyer", "baker", "judge", "governor", "butler"]*	*["doctor", "**teacher**", "**nurse**", "actress", "student", "miller", "reporter", "retired", "lawyer", "actor", "artist"]*
Inner Product Similarity	*["cleric", "photographer", "**skipper**", "chaplain", "accountant", "inspector", "rector", "investigator", "psychologist", "treasurer", "supervisor"]*	*["**librarian**", "**housekeeper**", "**nanny**", "accountant", "sheriff", "envoy", "tutor", "salesman", "butler", "footballer", "solicitor"]*

Summary Word2Vec

Steps

1. Define task that we want to predict
2. Go through each sentence and create the task's in-/outputs
3. Iterate through task's I/O, put the inputs through the embeddings and models to create predictions
4. Measure cost of the predicted and expected output
5. Update embedding weights accordingly (*backprop)
6. Repeat Step 3-5 until desired.

Neither **GloVe** or **Word2Vec** has been shown to provide definitively better results rather they should both be evaluated for a given dataset.

Train Word2Vec from “scratch”

CBOW AND SKIPGRAM

References

1. Role, Franois, and Mohamed Nadif. "Handling the impact of low frequency events on co-occurrence based measures of word similarity." Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR-2011). Scitepress. 2011.
2. Yogan, Jaya Kumar, et al. "A review on automatic text summarization approaches." Journal of Computer Science12.4 (2016): 178-190
3. Peters, Matthew E., et al. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018)
4. Radford, Alec, et al. "Improving language understanding by generative pre-training." URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf (2018).
5. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.
6. Howard, Jeremy, and Sebastian Ruder. "Universal language model fine-tuning for text classification." arXiv preprint arXiv:1801.06146 (2018).
7. Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.