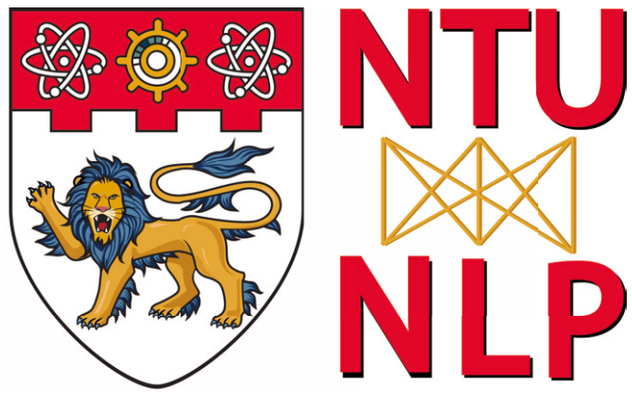
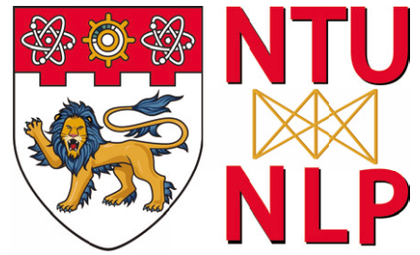
Deep Learning for Natural Language Processing

Shafiq Joty



Lecture 5: More on Word Vectors

Announcements

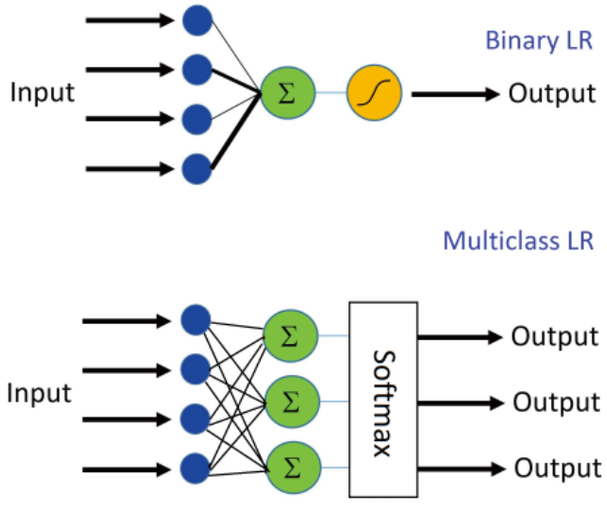


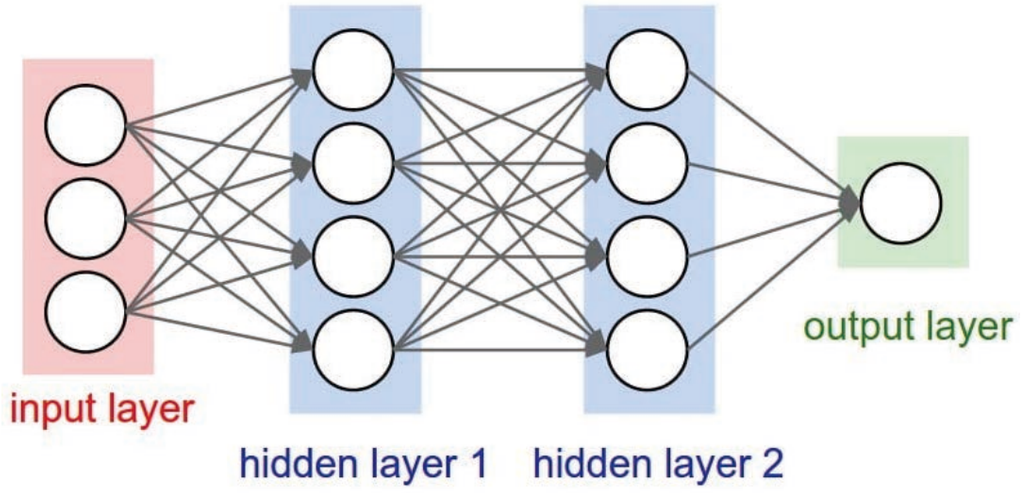
- Assignment 1 submission deadline today
- Extension for ICML and KDD deadlines
- Start thinking about your course project

Things Covered So Far

NTU NLP

- Lecture 2: Linear Models (Linear & Logistic Regression)
  - Training with gradient descent and SGD
- Lecture 3: Extended linear models to MLP/FNN
  - Training with SGD (+ Backdrop)

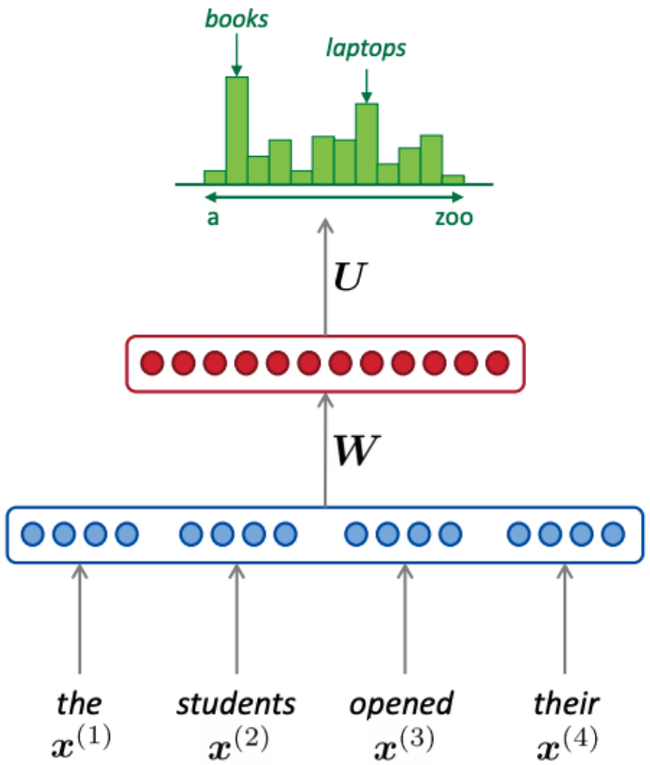


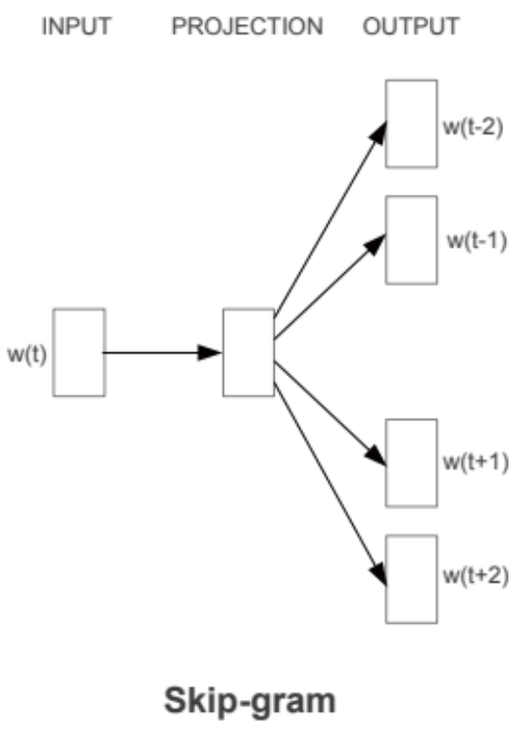


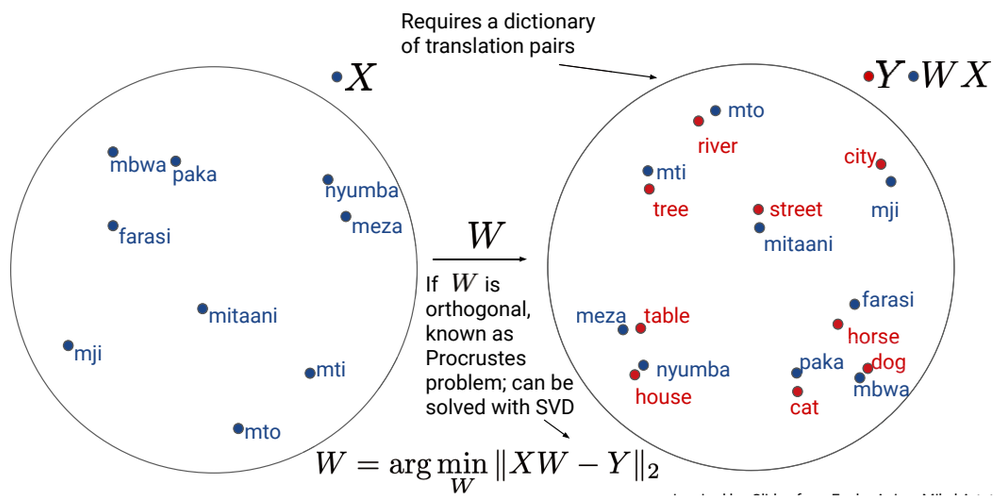
Things Covered So Far

NTU NLP

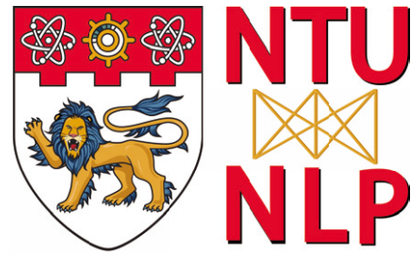
- Lecture 4 & 5: Used simple FNN as
  A. Simple Language Models
  B. Word2Vec Models







Where we are



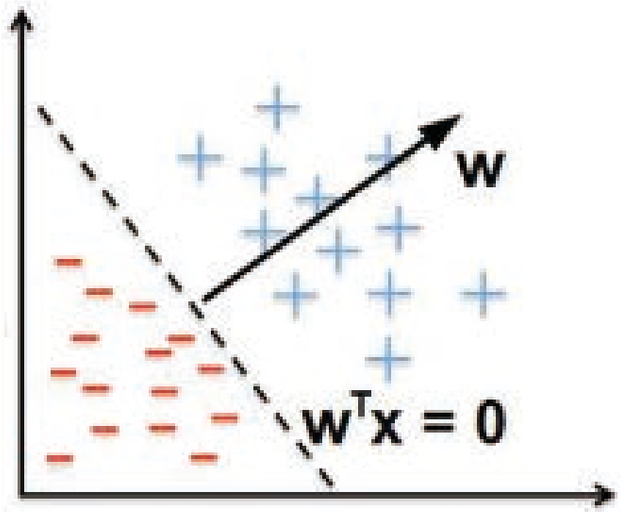
Models/Algorithms

- Linear models
- Feed-forward Neural Nets (FNN)

NLP tasks/applications

- **Word** meaning
- **Language modelling**

Linear Classifiers



- For linear models (logistic regression), we have a linear decision boundary (hyperplane).
- **Traditional approach:** assume x_i are fixed.

  From lecture 2:

Instead of **maximizing** the log-likelihood, we can equivalently **minimize** the negative log likelihood or NLL.

The negative log-likelihood for logistic regression is given by

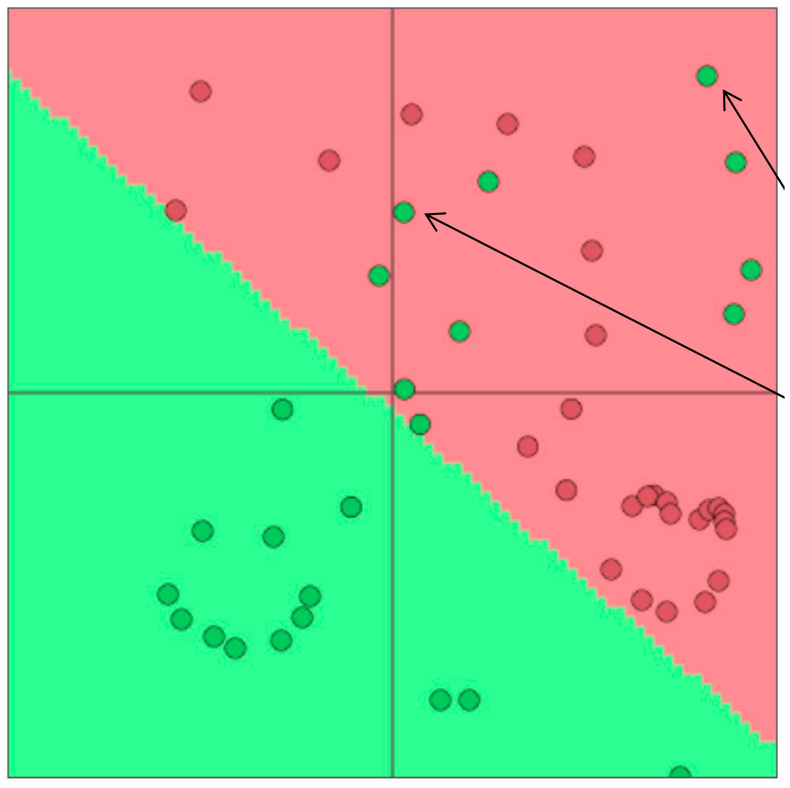
$$NLL(\mathbf{w}) = -\sum_{i=1}^{N} \log[\mu_i^{\mathbb{1}(y_i=1)} \times (1-\mu_i^{\mathbb{1}(y_i=0)})]$$
$$= -\sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$$

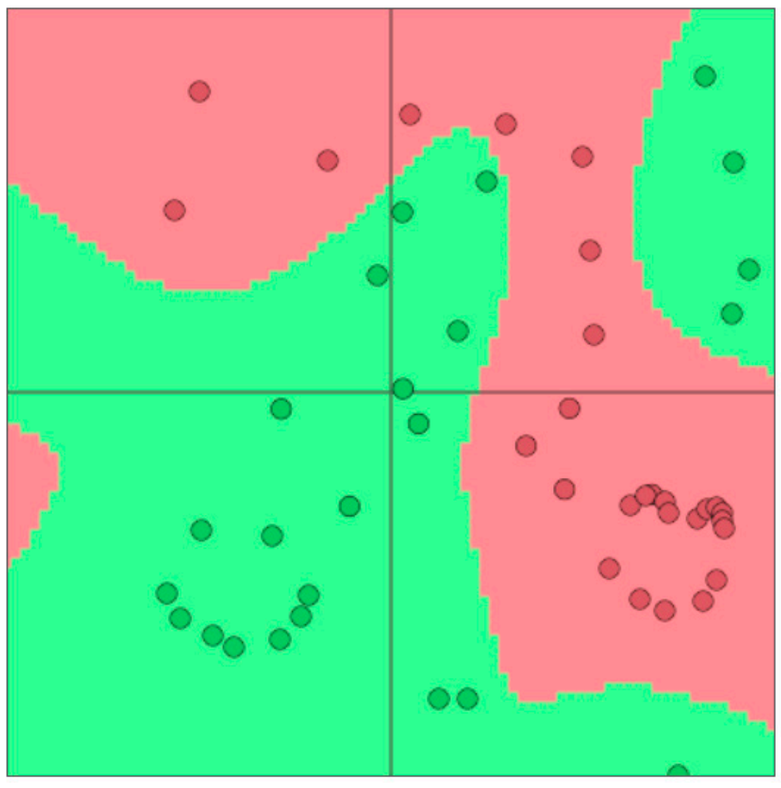
This is also called the Cross Entropy Error Function.

$$H(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

Neural Classifiers

- Linear models or Softmax (≈ logistic regression) alone not very powerful
- Real world classification problems are often too complex involving non-linear complex decision boundary
- Neural networks can learn more complex transformation functions and non-linear decision boundaries.





Source: CS224n

Neural vs. Linear Classifiers

NTU NLP

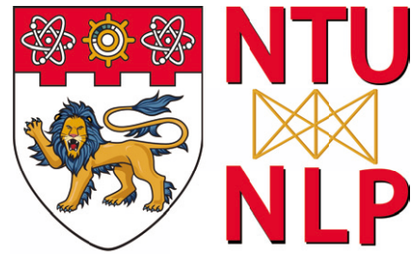
- We learn **both** W (weights, parameters) **and** representations (x & f(x))

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

Very large number of parameters!

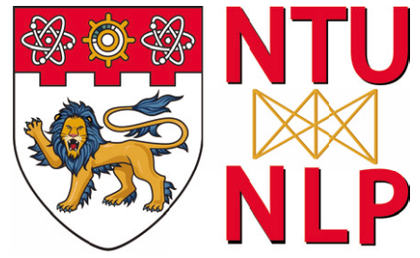
Source: CS224n

Lecture Plan



- Classification tasks in NLP
- Window-based Approach for Language Modeling
- Window-based Approach for NER, POS tagging, and Chunking
- Convolutional Neural Net for NLP
- Max-margin Training
- Scaling Softmax (for Language Modeling)
  - Adaptive input and output.

Classification Problems



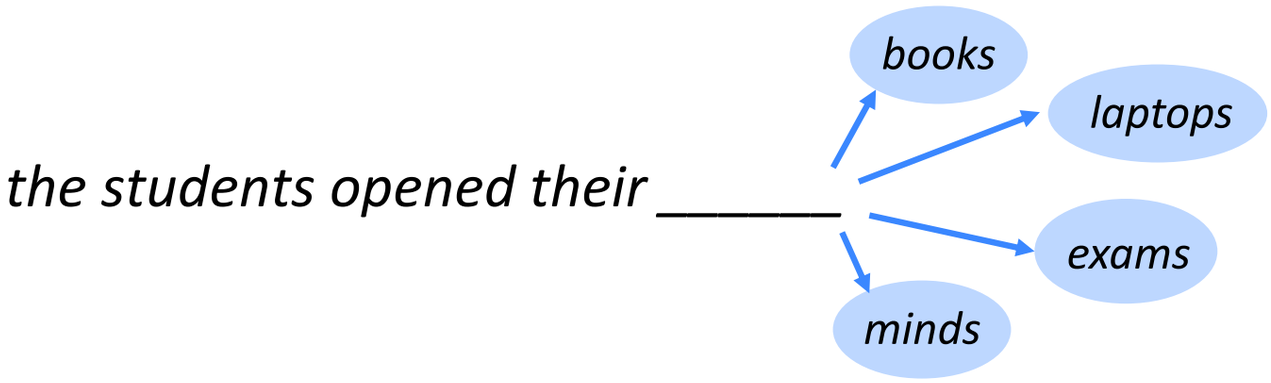
In a supervised setup, we have a training dataset

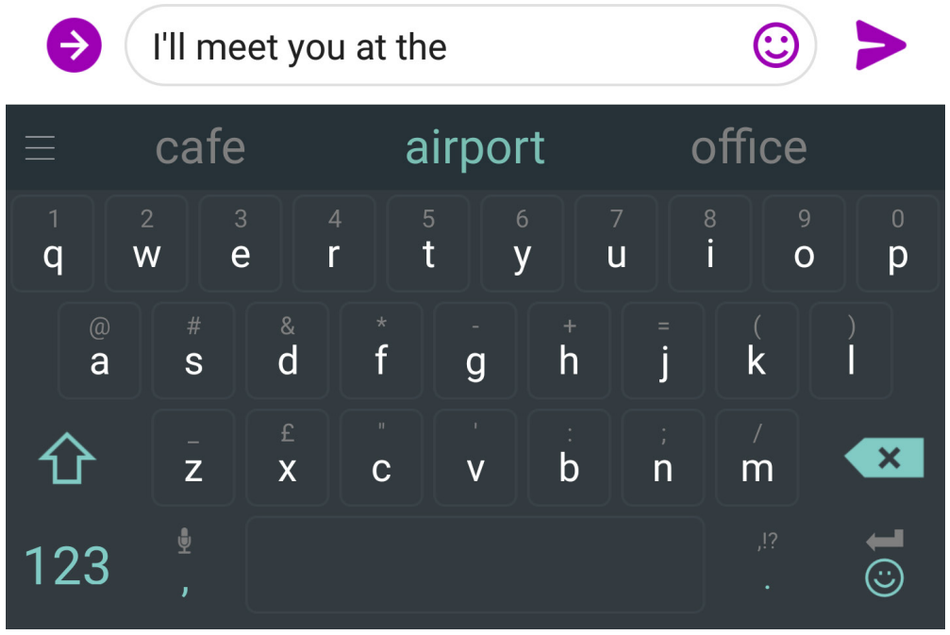
$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

- x_i are inputs, e.g. words, sentences, documents, etc.
- y_i are labels (one of C classes) we try to predict, e.g.,
  - Other words: Language model
  - Classes: POS tag, sentiment, named entities
  - Multi-word sequences: Translation, Summarisation

Language Modelling

A language model takes a list of words (history/context), and attempts to predict the word that follows them





Google

what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

Google Search I'm Feeling Lucky

Source: Stanford 224n

Language Modelling

(i) A language model takes a list of words (history/context), and attempts to predict the word that follows them

More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Source: Stanford 224n

Language Modelling

(ii) Language Models (LM) also assign a probability to a piece of text.

For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

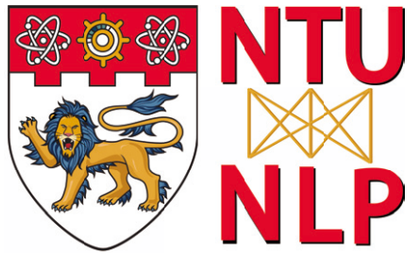
$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Source: Stanford 224n

Language Modelling - Ngram based Approach



Today: we will revisit window-based approach for language modelling

But, before that we will see how language modelling was done in the pre-neural era

Ngram based LM

A n-gram is a sequence of n consecutive words.

- unigrams: "the", "students", "opened", "their"
- bigrams: "the students", "students opened", "opened their"
- trigrams: "the students opened", "students opened their"
- 4-grams: "the students opened their"

Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

Source: Stanford 224n

Ngram based LM

Markov assumption: make a simplifying assumption that a word's probability depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

How can we estimate this probability?

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

Source: Stanford 224n

Ngram based LM

How can we estimate this probability?

$$= \frac{P(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}$$

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(t-n+2)})}$$

Count this in the corpus

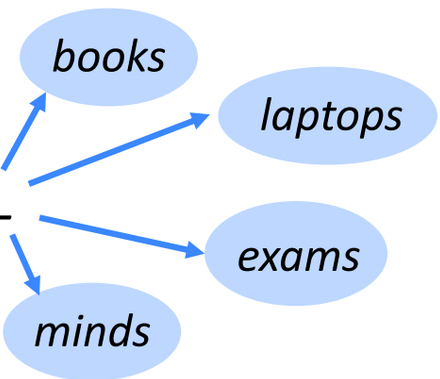
For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Source: Stanford 224n

Problem with the Markov Assumption

~~***As the proctor started the clock,***~~ *the students opened their _____*



For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Source: Stanford 224n

Sparsity Issues with Ngrams

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

(Partial) Solution: Add small δ to the count for every $\boldsymbol{w} \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Markov assumption was introduced in the first place to avoid sparsity

Source: Stanford 224n

Storage Issues with Ngrams

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

Increasing *n* or increasing corpus increases model size!

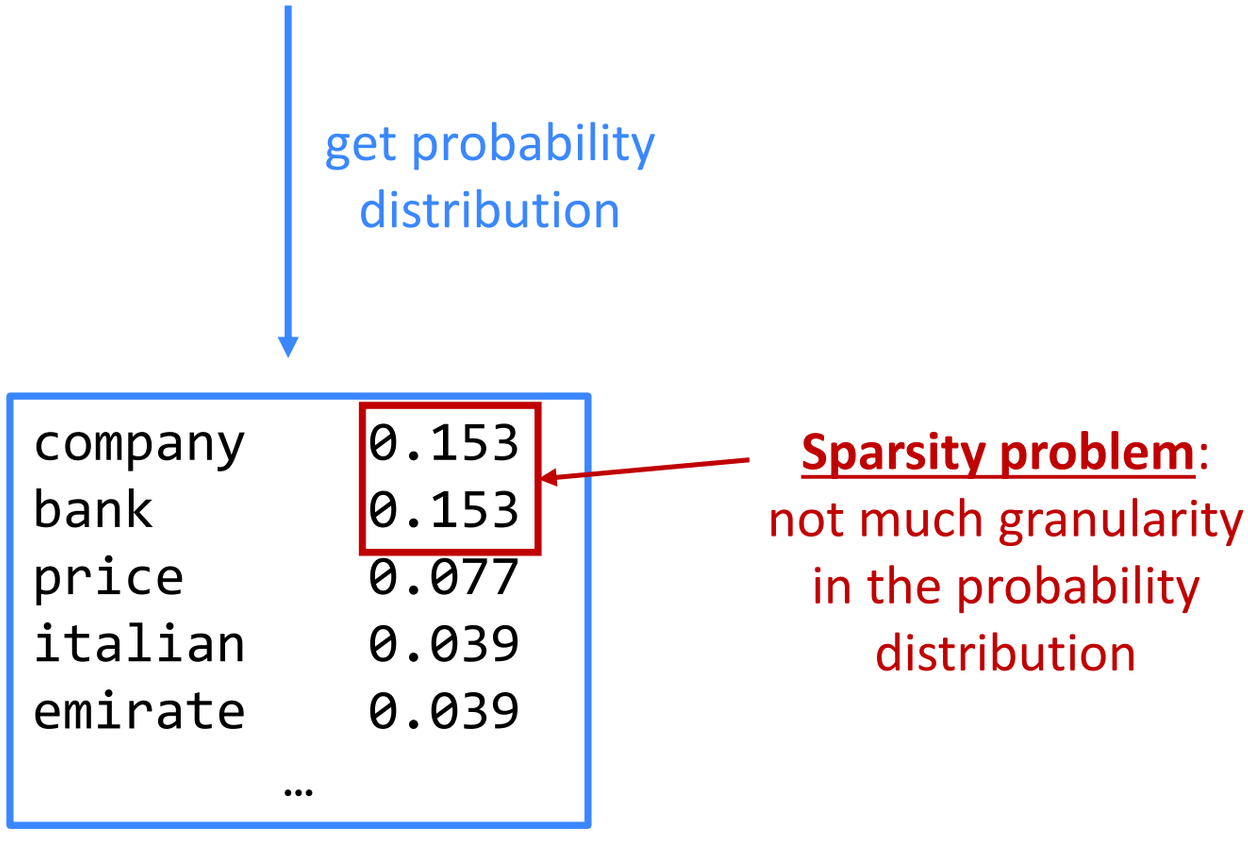
Source: Stanford 224n

Ngram LM in Practice

We can build a simple trigram language model over a
1.7 million word corpus (Reuters) in a few seconds on your laptop

- here is an example code: https://nlpforhackers.io/language-models/

today the _____



Source: Stanford 224n

Ngram LM in Practice

We can also use a language model to generate text.

today the _____

condition on this

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

sample

Source: Stanford 224n

Ngram LM in Practice

NTU NLP

We can also use a language model to generate text.

today the price _____

condition on this

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Source: Stanford 224n

Ngram LM in Practice

We can also use a language model to generate text.

today the price of _____

condition on this

get probability distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Source: Stanford 224n

Ngram LM in Practice

If we keep going.....

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing n worsens sparsity problem, and increases model size...

Source: Stanford 224n

Window-based Language Model (Revisited)

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

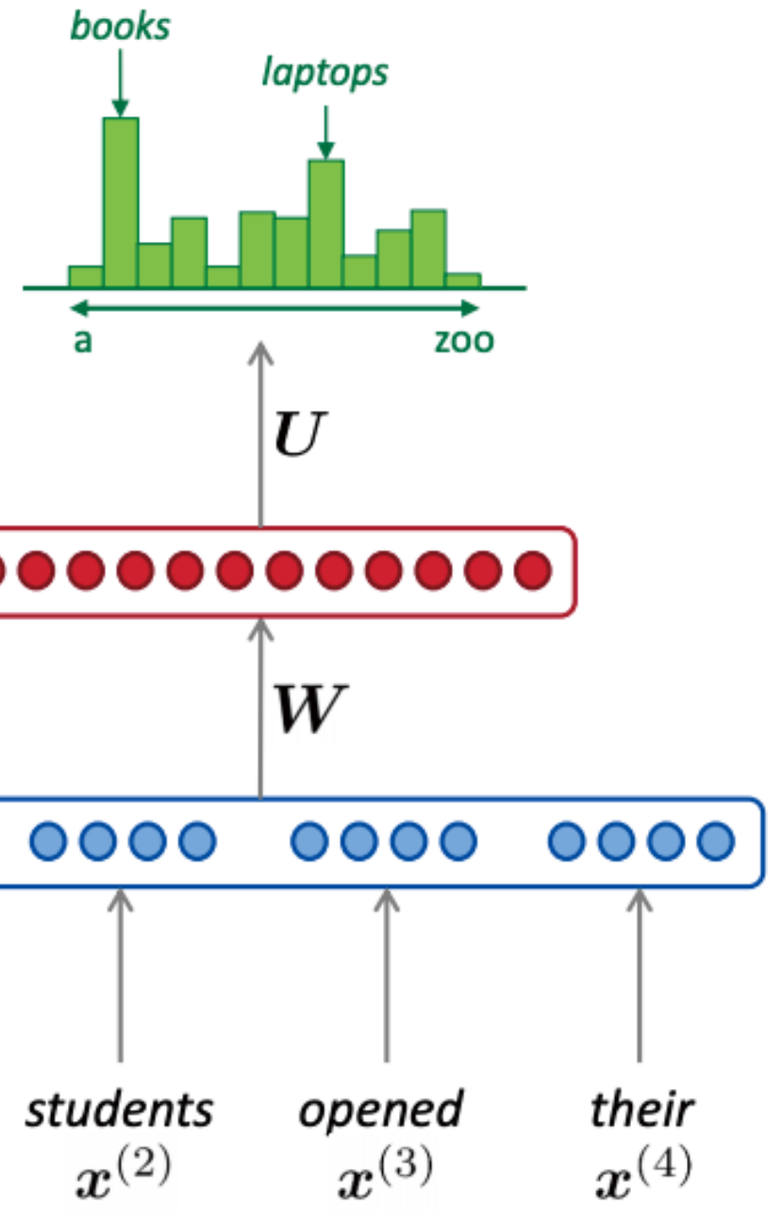
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

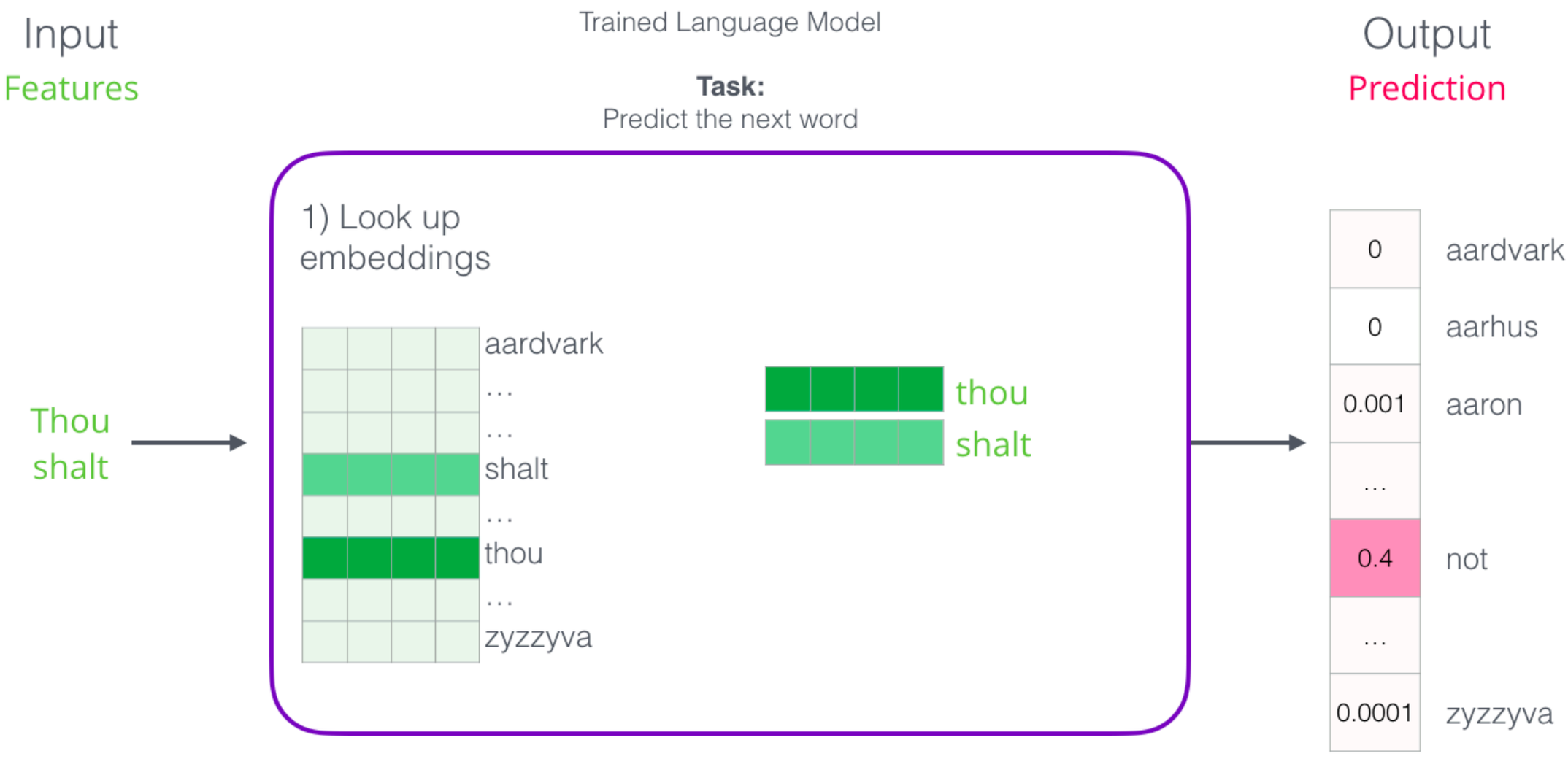
words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



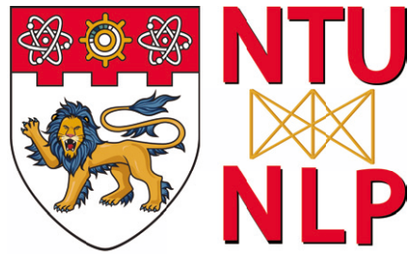
Source: Bengio, 2003

Window-based Language Model (Revisited)



Source:http://jalammar.github.io/

Named Entity Recognition



Named Entity Recognition (NER): Identify and classify entities such as persons, locations and organisations.

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:

LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

Named Entity Recognition

NTU NLP

NER in other Language and Domains

Spanish

B-PER	I-PER	O	O	O	B-ORG	I-ORG	I-ORG
Jaime	Matos	,	de	los	Rebeldes	de	Moca

BIO encoding

Fin.

B-ORG	I-ORG	I-ORG	O	O	B-LOC	O	B-ORG
Silicon	Valley	bank	,	a	California	chartered	bank

..

NER has many applications

- Knowledge graph
- Search
- Translation

Why NER is Hard?

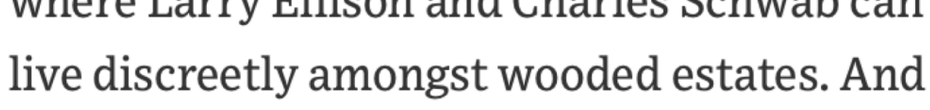
First National Bank Donates 2 Vans To Future School Of Fort Smith

POSTED 3:43 PM, JANUARY 11, 2019, BY 5NEWS WEB STAFF

- Hard to know if something is a name; Is there a school called "Future School" or is it a future school
- Hard to find name boundaries; Is the first entity "First National Bank" or "National Bank"
- Hard to know class of unknown/novel name; what class is "Zig Ziglar"? (A person.)

To find out more about Zig Ziglar and read features by other Creators Syndicate writers and

- Entity class is ambiguous and depends on context; Charles Schwab is a Person or an Org?

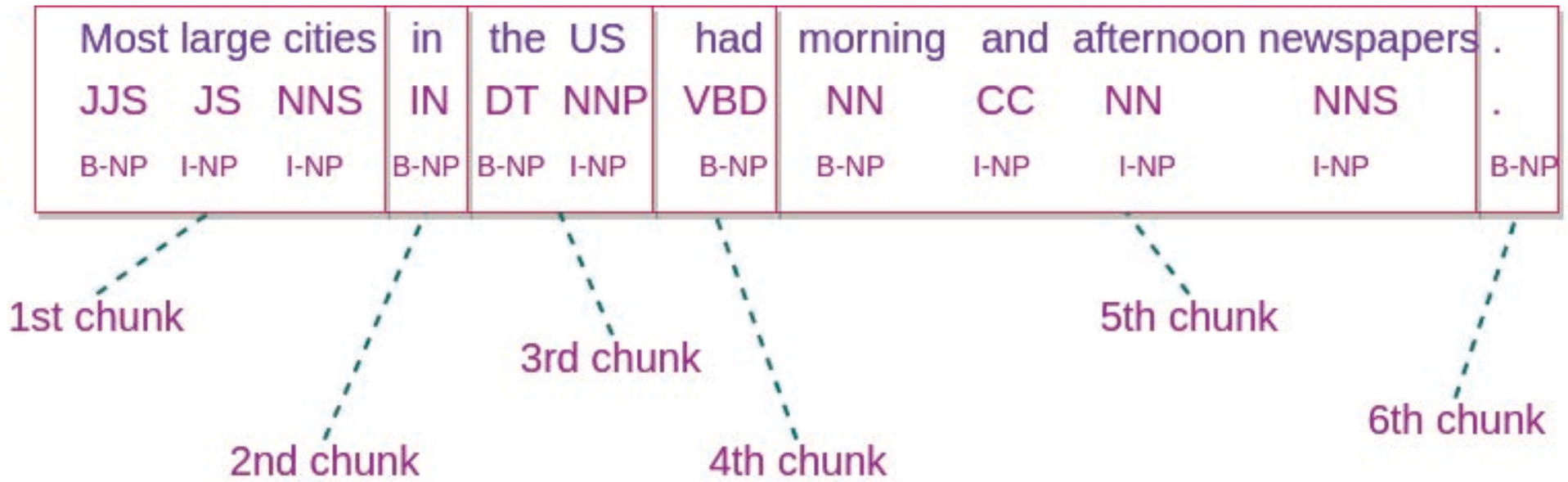


where Larry Ellison and Charles Schwab can live discreetly amongst wooded estates. And

Source: CS224n

Other Similar NLP Tasks

Chunking



Source: Internet

Other Similar NLP Tasks

Part-of-Speech (POS) Tagging

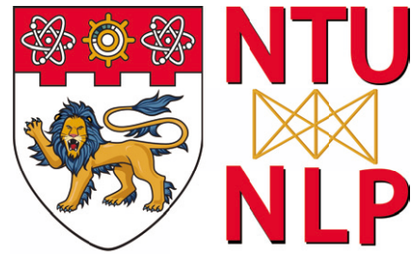
The_DT	first_JJ	time_NN	he_PRP	was_VBD	shot_VBN	in_IN	the_DT
hand_NN	as_IN	he_PRP	chased_VBD	the_DT	robbers_NNS	outside_RB	..

first	time	shot	in	hand	as	chased	outside
JJ	NN	NN	IN	NN	IN	JJ	IN
RB	VB	VBD	RB	VB	RB	VBD	JJ
		VBN	RP			VBN	NN
							RB

Similar to NER, both POS tagging and Chunking have many applications in other NLP tasks and downstream applications

Source: Marquez (2000)

Window-based Approach



- In general, classification of one token depends in its context.
- Example: resolving ambiguous named entities:
  - Paris → Paris, France vs. Paris Hilton vs. Paris, Texas
  - Hathaway → Berkshire Hathaway vs. Anne Hathaway
  - Washington → Washington DC vs. Denzel Washington
  - [redacted]
  - [redacted] and to classify the average vector. [redacted]

Window-based Approach

- Train a classifier to classify a center word by taking c[redacted] o[redacted].
- We can put hidden layers to make the model deep.
- Example: Classify "Paris" in with window length of 2:

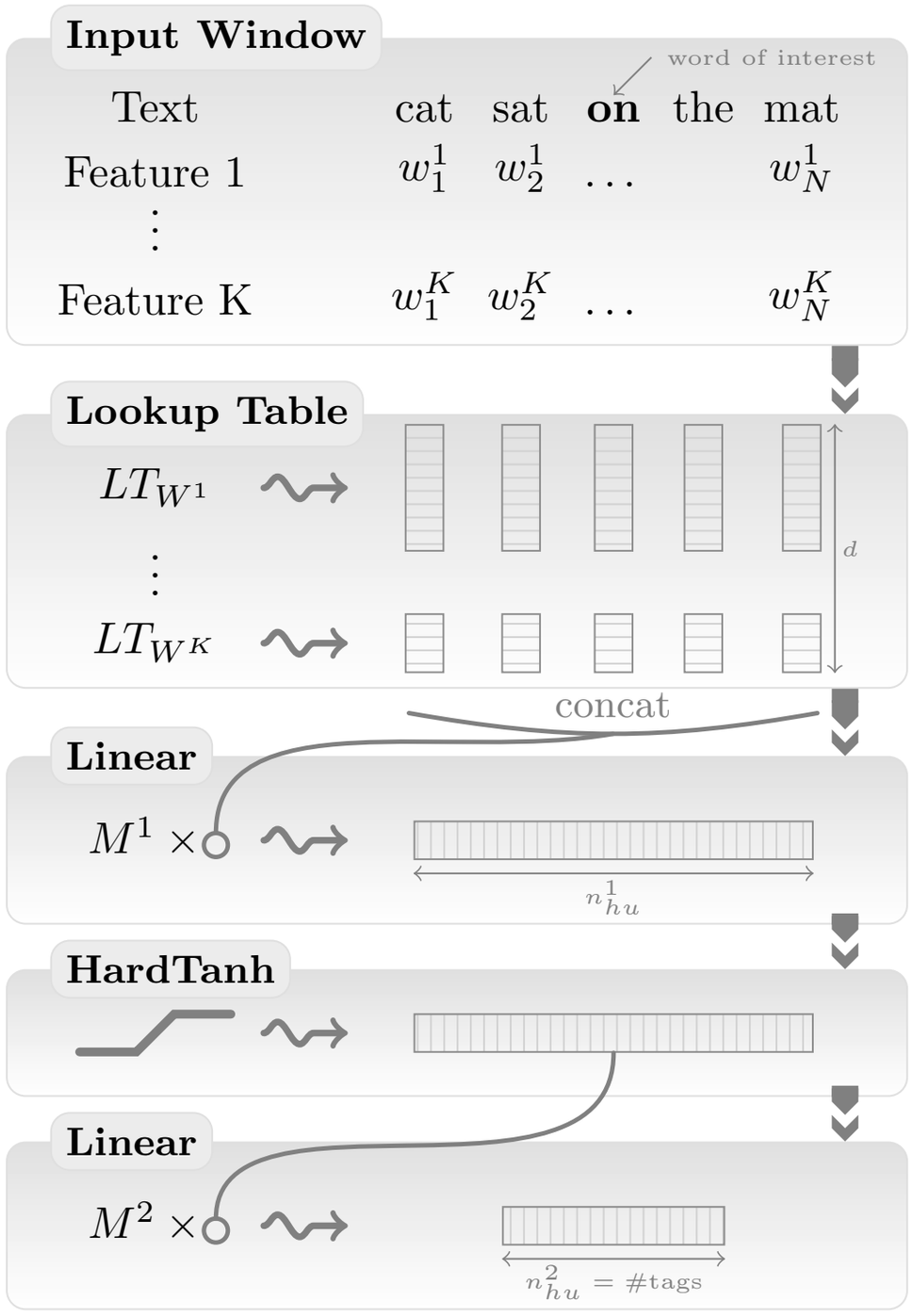
… museums in Paris are amazing … .

$x_{window} = x \in R^{5d}$

$X_{window} = [\ x_{museums}\ \ x_{in}\ \ x_{Paris}\ \ x_{are}\ \ x_{amazing}\ ]^T$

- Similar to word2vec, we go over all positions in a sentence. But this time, it is supervised.

Window-based Approach



Nonlinearity

Source: Collobert et al (2011)

Window-based Approach

- We can use the same Softmax as the final classification layer:

$$\mathcal{P}(y = c|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^\mathsf{T}\mathbf{x})}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\mathsf{T}\mathbf{x})}$$

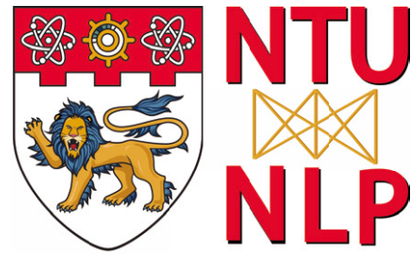
W is final layer weights

- Use the same Cross Entropy loss:

$$\ell(\mathbf{W}) = \sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

$$= \sum_{i=1}^{N}\left[\left(\sum_{c=1}^{C} y_{ic}\mathbf{w}_c^\mathsf{T}\mathbf{x}_i\right) - \log\left(\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\mathsf{T}\mathbf{x}_i)\right)\right]$$

Where we are



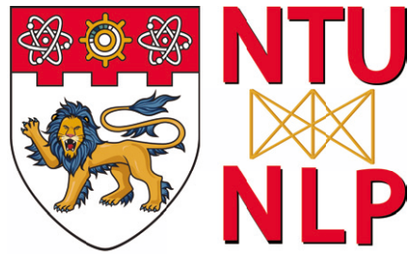
Models/Algorithms

- Linear models
- Feed-forward Neural Nets (FNN)
- Window-based methods
- Convolutional Nets

NLP tasks/applications

- Word meaning
- Language modelling
- Sequence tagging

Convolutional Neural Nets



Convolutional neural network (CNN, or ConvNet) is a type of feed-forward network where the individual neurons are tiled in such a way that they respond to overlapping regions (context windows) in the input.

In other words, CNN is a type of feed-forward neural network with

- Local connectivity (with the input layer)
- Share parameters across spatial/temporal positions

Advantage over FNN

- Requires less parameters (sharing)

Advantage over Recurrent Neural Net (next lecture)

- Parallelizeable

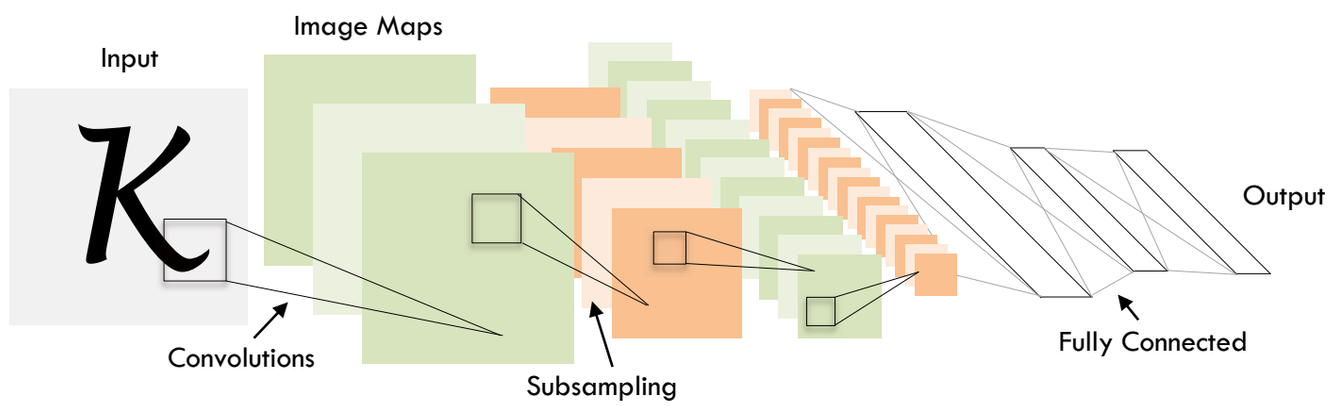
CNNs in Computer Vision

NTU NLP

- CNNs were responsible for major breakthroughs in Image Classification (e.g., face recognition) and are the core of most Computer Vision systems today.

1998

LeCun et al.



# of transistors

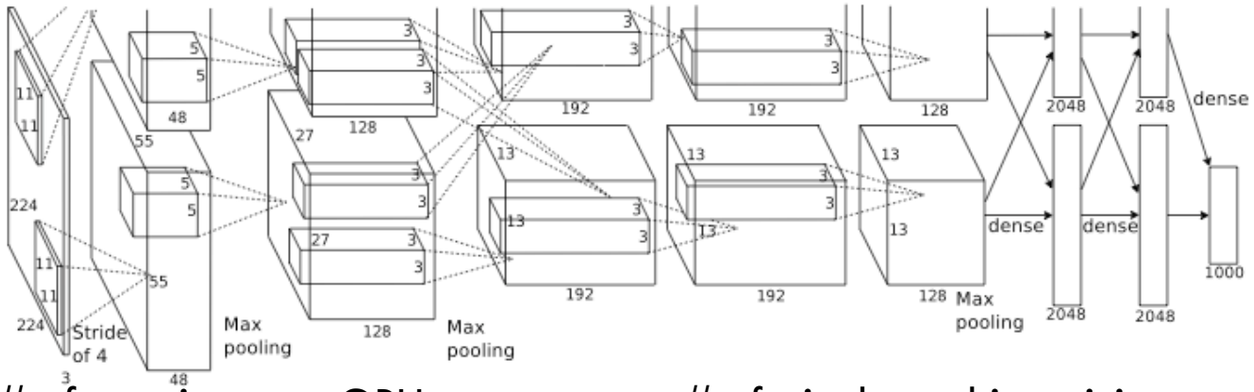
10^6

# of pixels used in training

10^7 NIST

2012

Krizhevsky et al.



# of transistors

10^9

GPUs

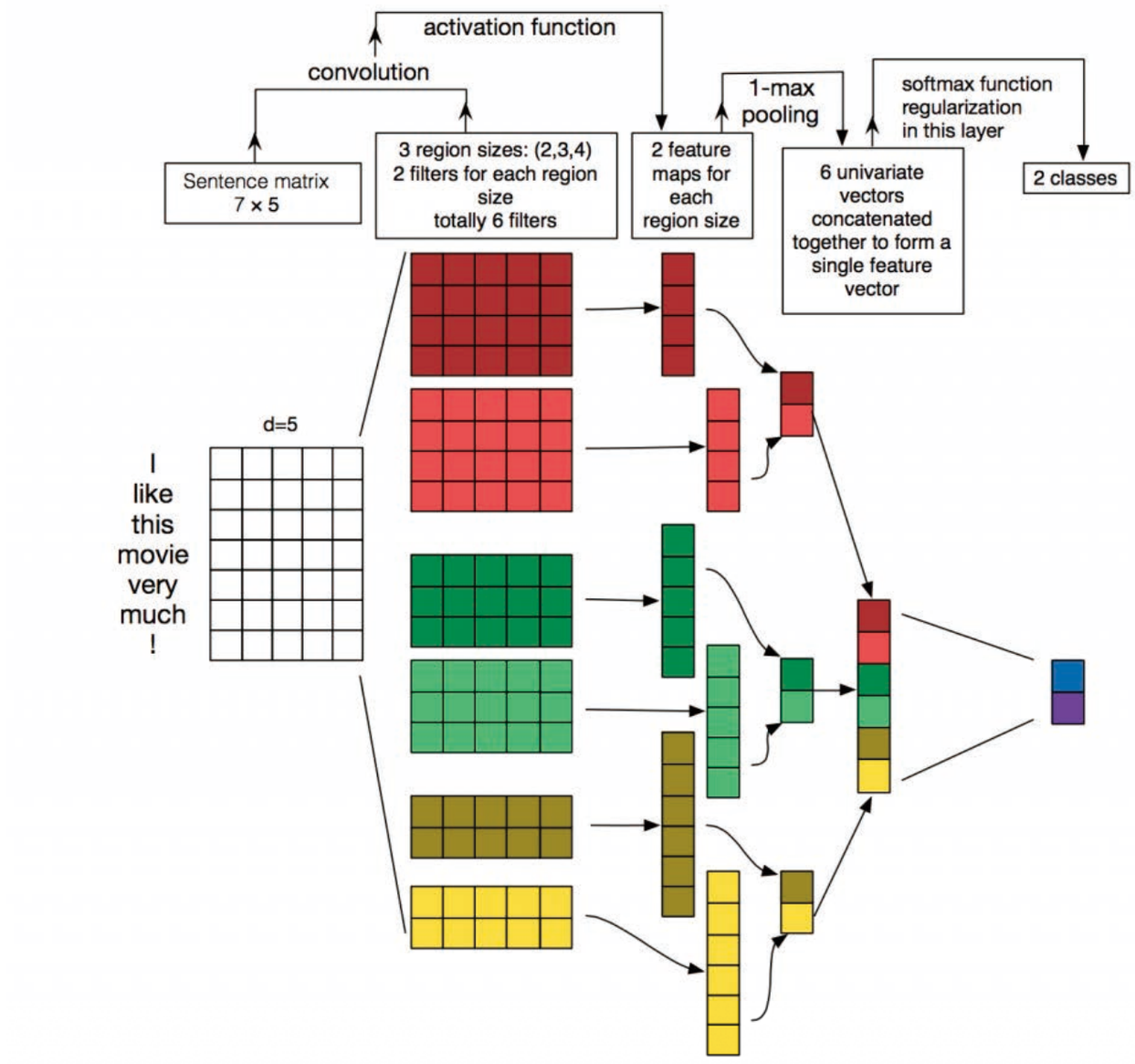
# of pixels used in training

10^{14} IMAGENET

Source: cs231n

CNNs in NLP

- CNNs have in the past years shown break-through results in some NLP tasks, especially classifying phrases and sentences.

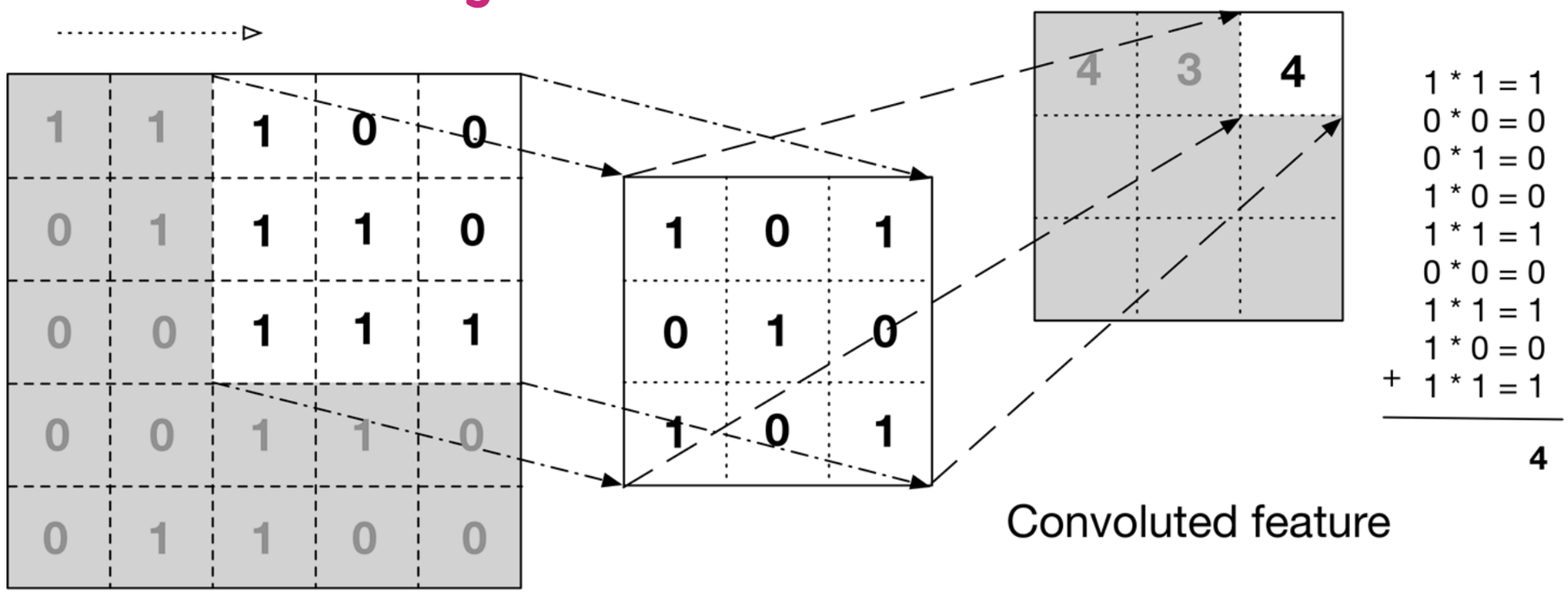


Source: Zhang, Y., & Wallace, B. (2015)

Convolution in Image

A CNN typically involves two operations: **convolution** and **pooling**.

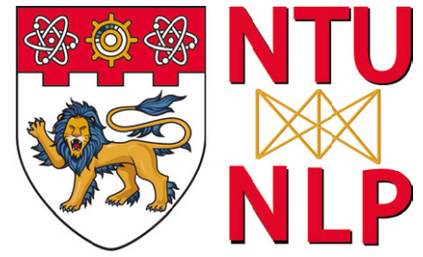
- **(2D) Convolution in image**



- The kernel is moved s pixels to the right/down, where s is the **stride length**
- The output of this process is a matrix with all it's entries filled, called the **convoluted feature** or **input feature map**
- An input image can be convolved with multiple convolution kernels at once, creating one output (feature map) for each kernel.

Source: Gibson & Patterson

Convolution in NLP



Convolution in NLP

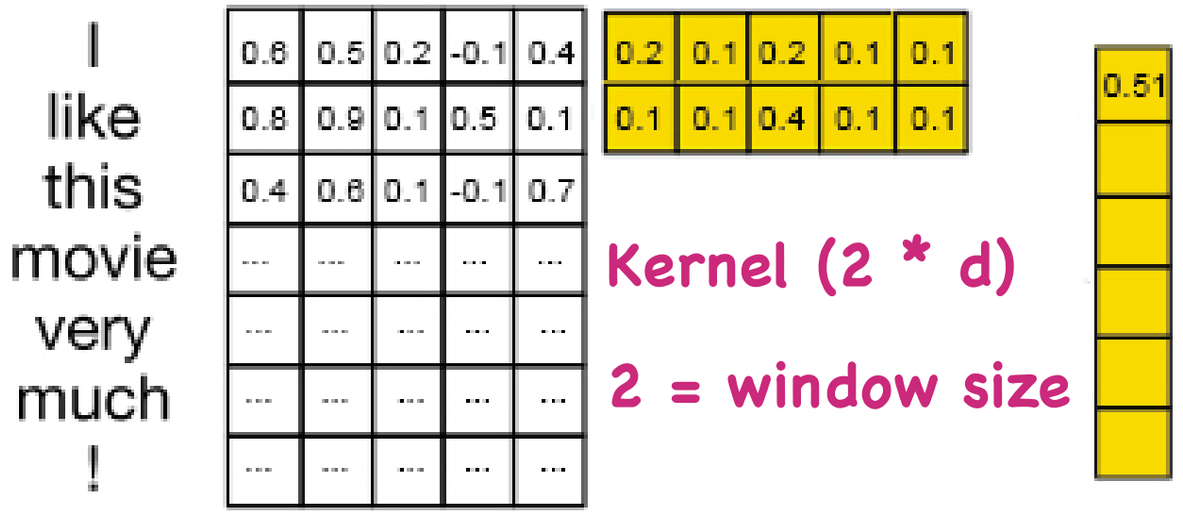
- Generally 1D (temporal)
- For each window, it is same as our window-based FFN method
- One Conv. operation

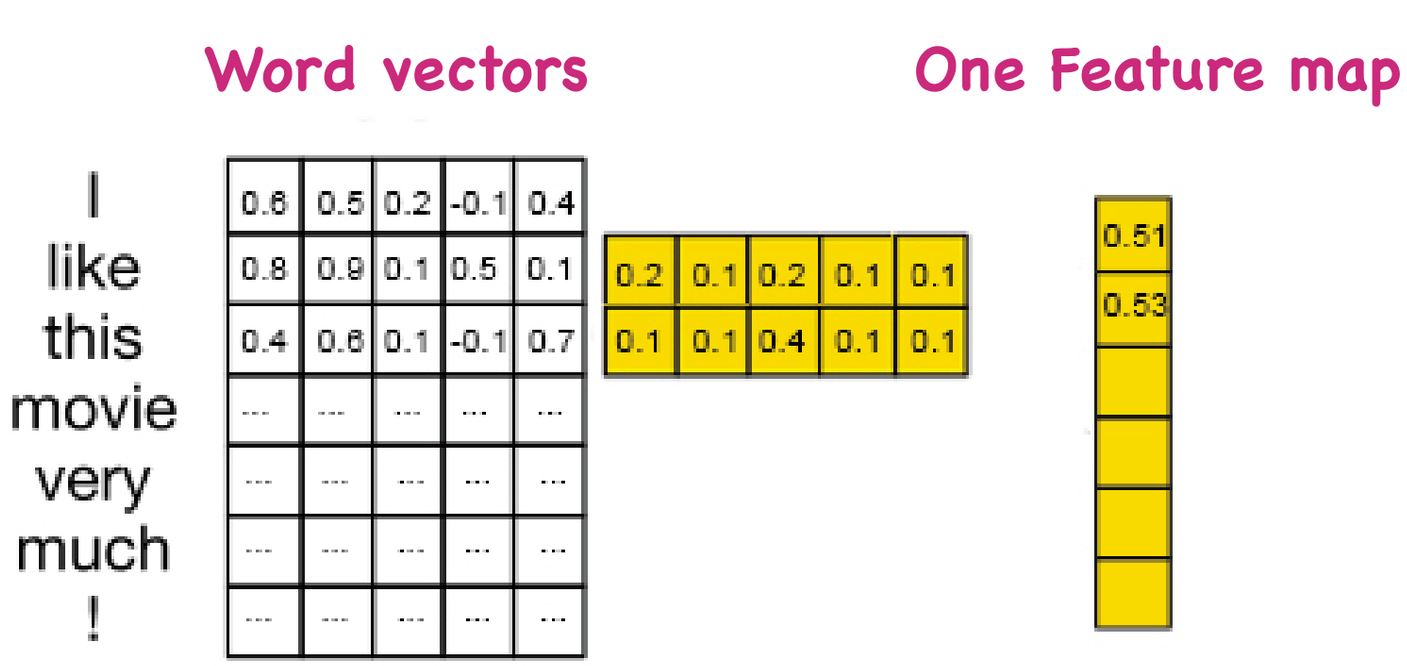
$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

$$filter\ \mathbf{w} \in \mathbb{R}^{hk}$$

- One feature map

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$$

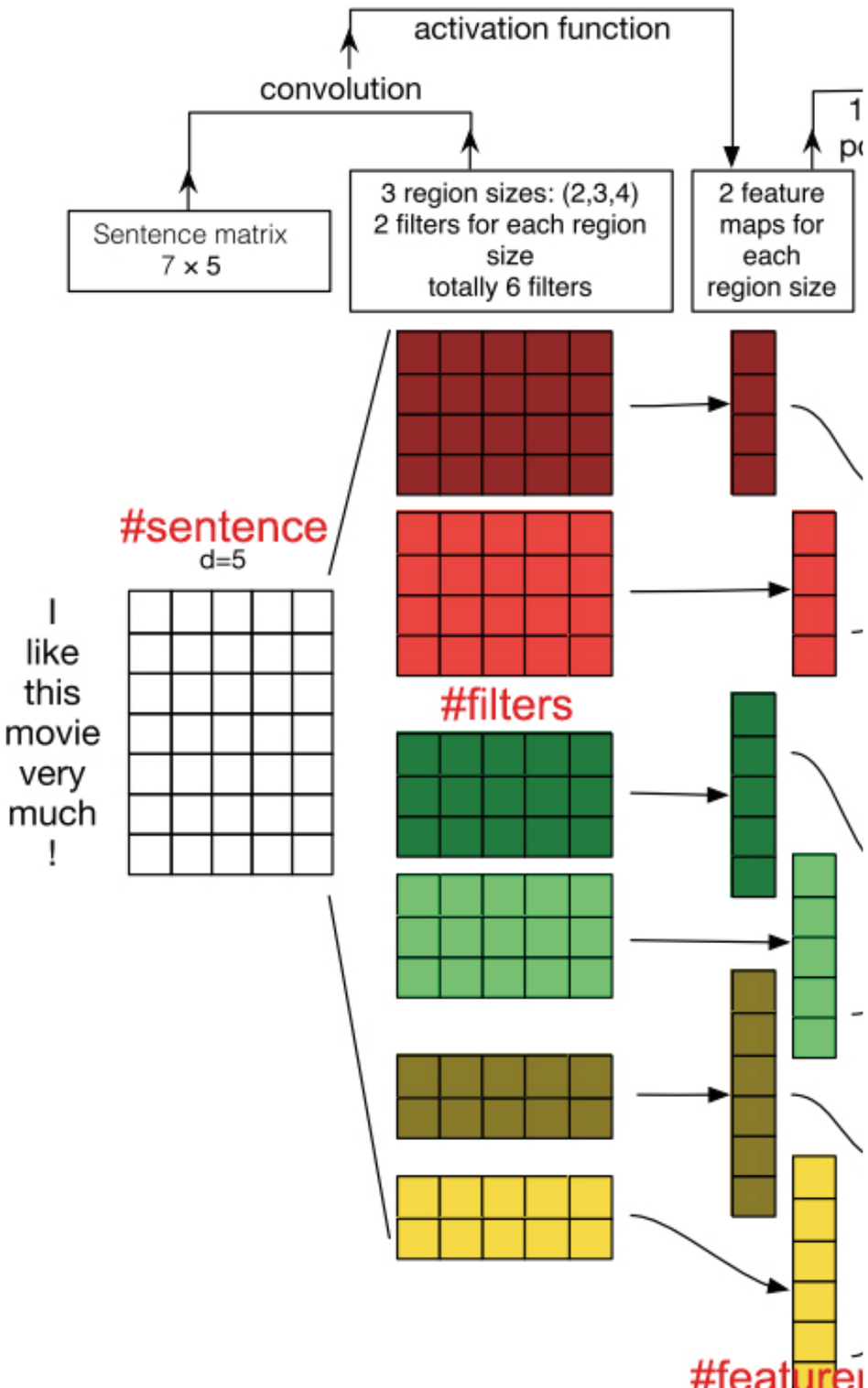




Convolution in NLP

Convolution with multiple filters

Multiple filters (with varying window sizes) to obtain multiple features



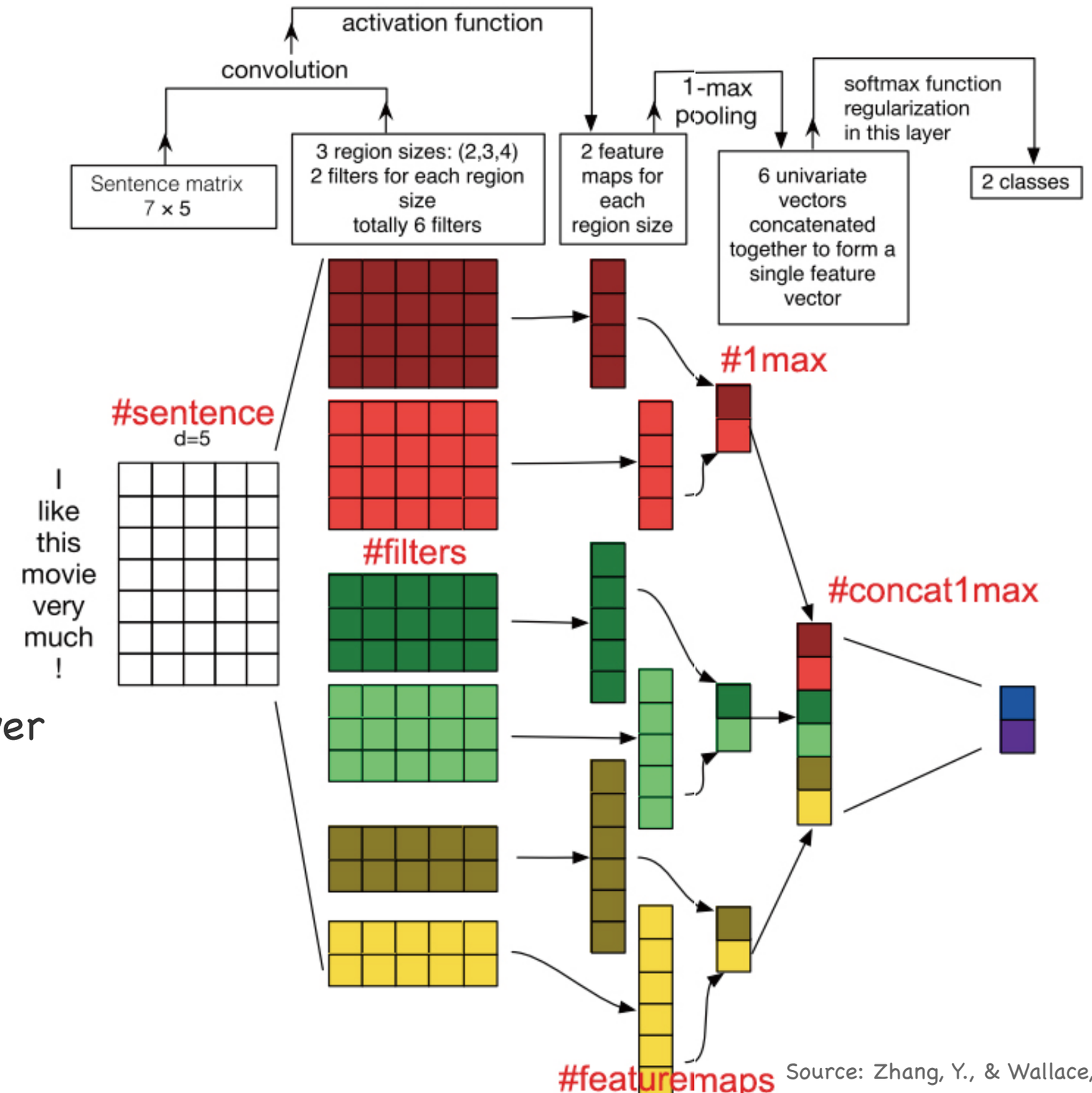
Source: Zhang, Y., & Wallace, B. (2015)

Two Basic Operations in CNNs

Pooling

- Mean, Max

- Performed over convoluted features for
  - Dimension reduction
  - Selecting important features
  - No parameter is added
- Often followed by a Fully connected layer to ensure fixed length input to final layer



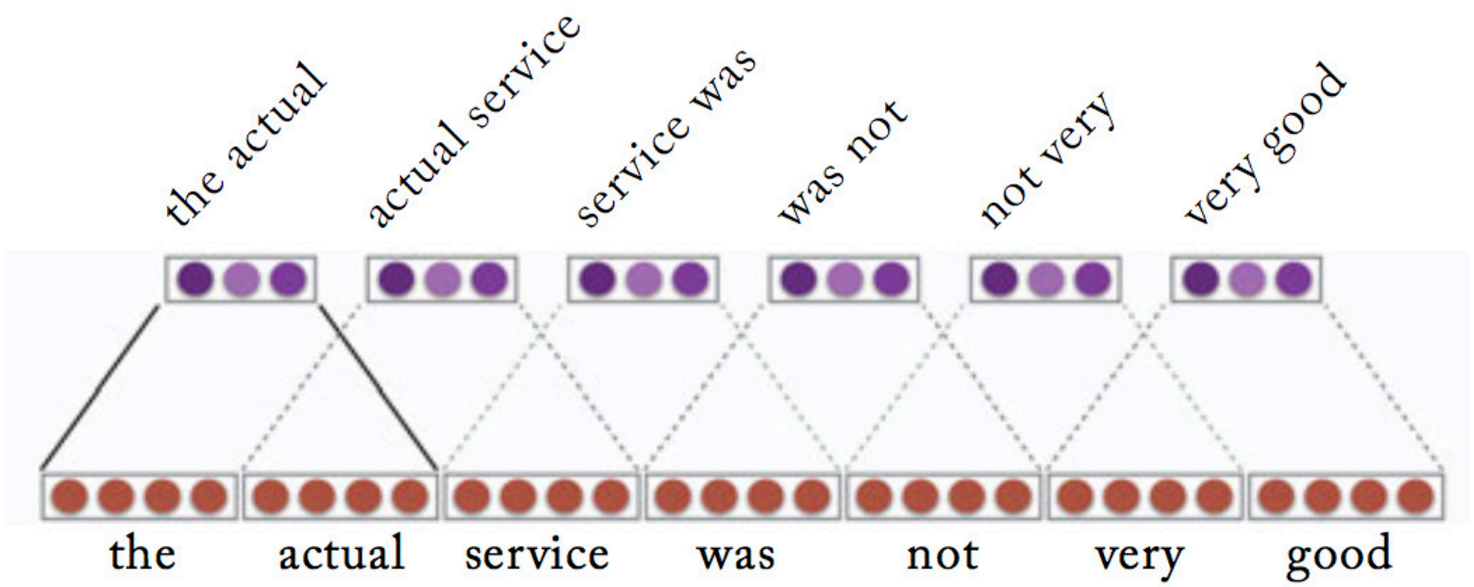
Source: Zhang, Y., & Wallace, B. (2015)

CNN for Tagging Tasks

NTU NLP

CNN applied to phrases in a sentence

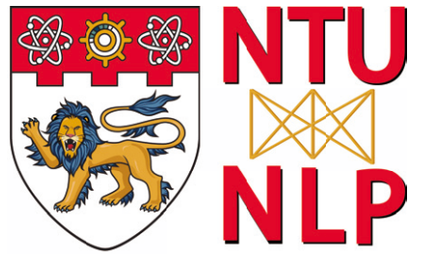
- Apply to POS tagging, Chunking, NER
- We can add more hidden (CNN, FNN) layers
- Final layer is a softmax layer over the possible tags
- Extract meaningful sub-structures (n-grams) that are useful for the overall prediction task
- Also can be applied to character level to capture morphological features



Convolution with window size=2 and dimensional output (number of filters/output channel) = 3.

Source: Yoav Goldberg. (2013)

CNN for Tagging Tasks



- Apply to POS tagging, Chunking, NER
- We can add more hidden (CNN, FNN) layers
- Final layer is a softmax layer over the possible tags
- Extract meaningful sub-structures (n-grams) that are useful for the overall prediction task
- Also can be applied to character level to capture morphological features

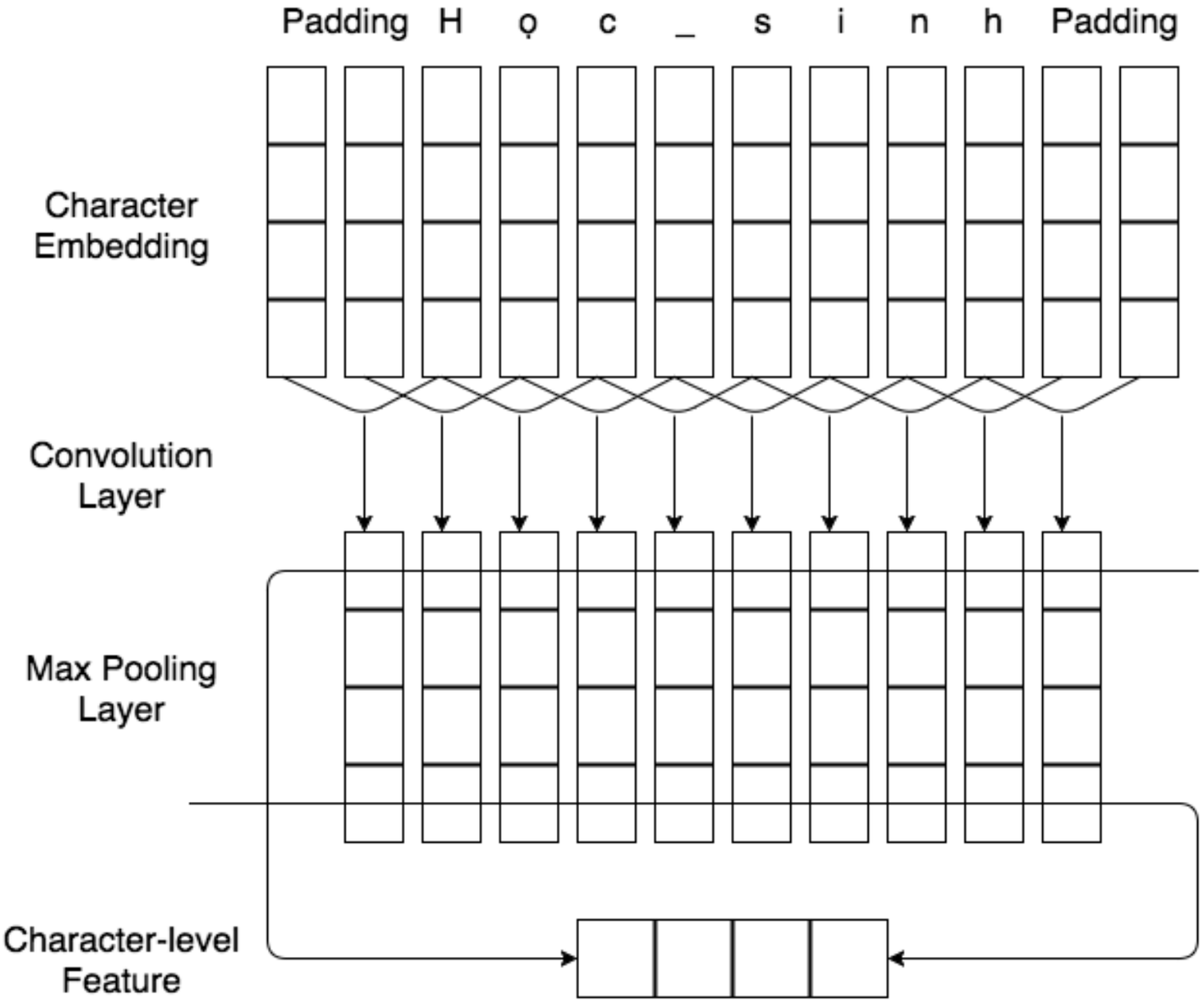
Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)	**SRL** (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99
NN+WLL+LM1	97.05	91.91	85.68	58.18
NN+SLL+LM1	97.10	93.65	87.58	73.84
NN+WLL+LM2	97.14	92.04	86.96	58.34
NN+SLL+LM2	97.20	93.63	88.67	74.15

Source: Collobert et al (2011)

CNN for Encoding

Character-level CNN

- CNNs can be applied to characters of a word
- Model morphology/subword
- Handle unknown (out of vocabulary) words
- Commonly used in Neural MT



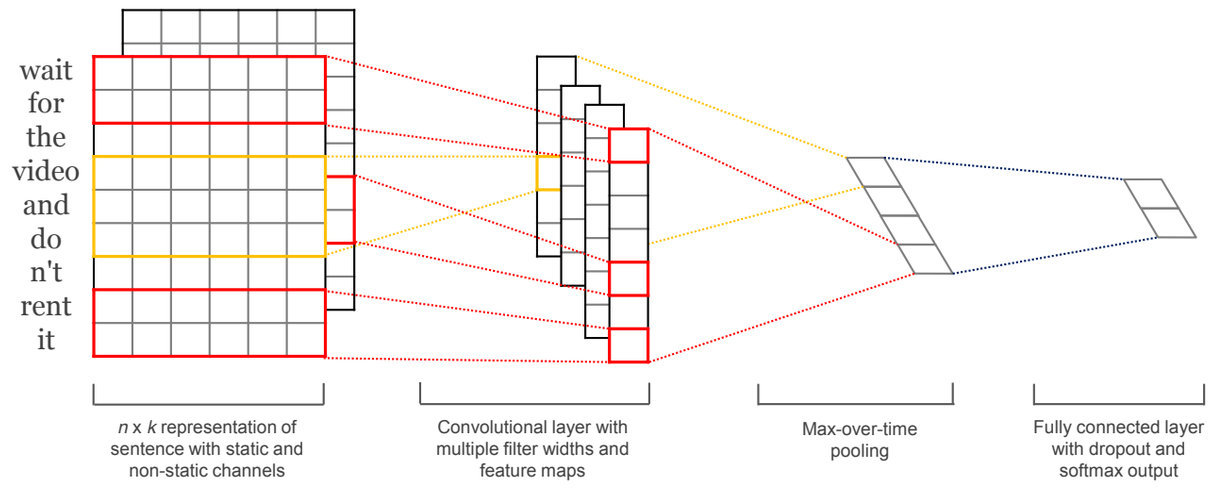
Source: Yoav Goldberg. (2013)

CNN for Tagging Tasks

NTU NLP

CNN for sentence modelling

- Apply to sentence classification tasks (e.g., sentiment)
- Final layer is a softmax layer over the possible classes
- Convolution extracts meaningful local (n-gram) features that are useful for the overall sentence level prediction task



Source: kim. (2014)

Other CNN Types

Dilated Convolution in Image

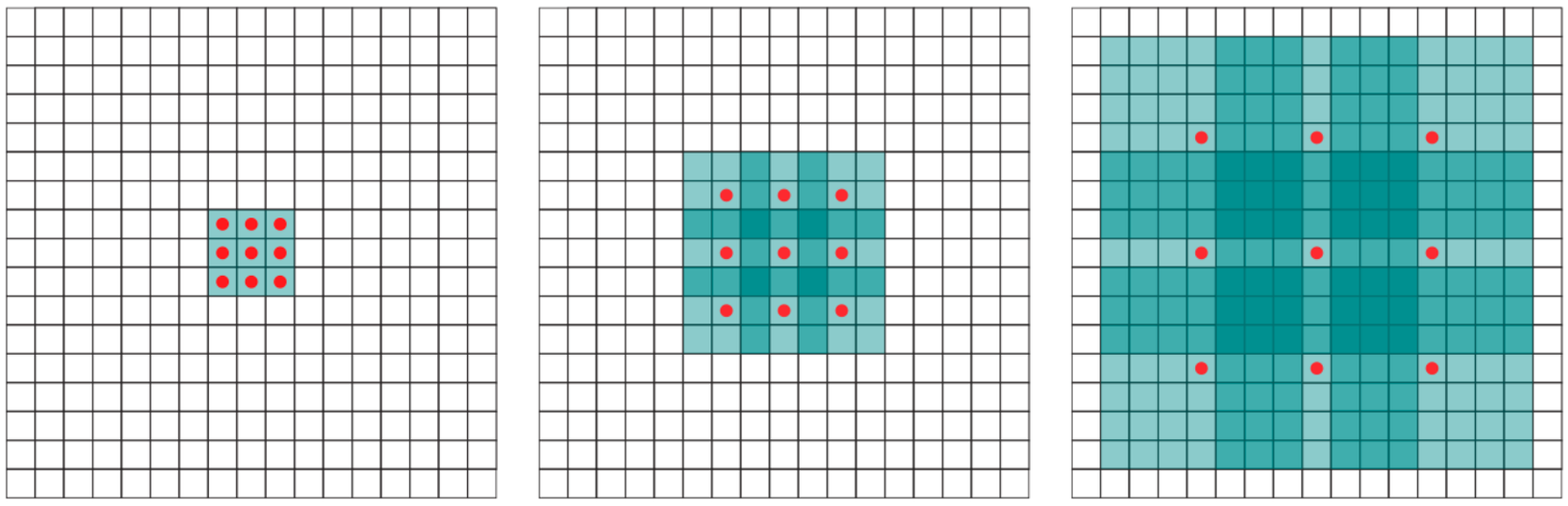


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Other CNN Types

Dilated Convolution in NLP

Regular convolution,

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t \pm k}$$

Dilated Convolution,

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t \pm k\delta}$$

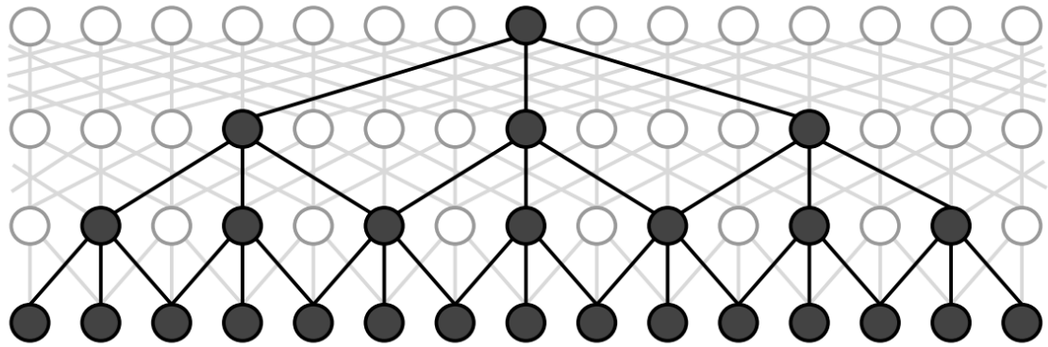
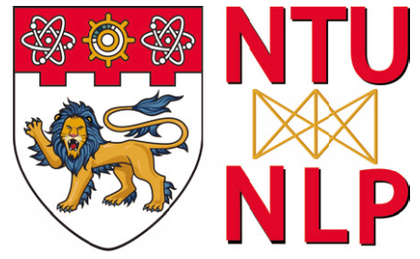


Figure: A dilated CNN block with maximum dilation width 4 and filter width 3.

$r = l(w-1)+1$
where , r number of node covered at layer l with filter width w.
total number of node at layer l is, $2^{l+1}-1$ (**exponential growth**).

Source: Strubell et al. (2018)

Other CNN Types



Dilated Convolution in NLP

Regular convolution,

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t \pm k}$$

Dilated Convolution,

$$c_t = W_c \bigoplus_{k=0}^{r} x_{t \pm k\delta}$$

Model	Speed
Bi-LSTM-CRF	1×
Bi-LSTM	9.92×
ID-CNN-CRF	1.28×
5-layer CNN	12.38×
ID-CNN	14.10×

Model	F1
Ratinov and Roth (2009)	86.82
Collobert et al. (2011)	86.96
Lample et al. (2016)	90.33
Bi-LSTM	89.34 ± 0.28
4-layer CNN	89.97 ± 0.20
5-layer CNN	90.23 ± 0.16
ID-CNN	90.32 ± 0.26
Collobert et al. (2011)	88.67
Passos et al. (2014)	90.05
Lample et al. (2016)	90.20
Bi-LSTM-CRF (re-impl)	90.43 ± 0.12
ID-CNN-CRF	**90.54 ± 0.18**

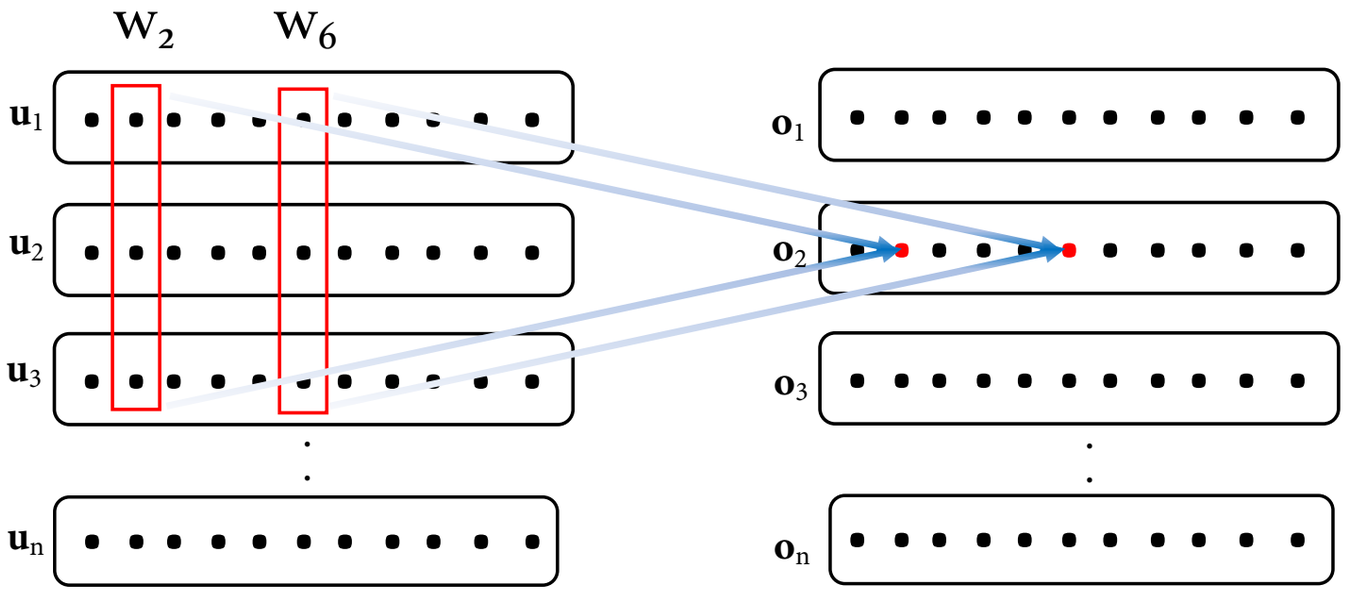
NER Results

Source: Strubell et al. (2018)

Other CNN Types

Depth-wise Convolution in NLP

The convolutions are done over the input dimensions.



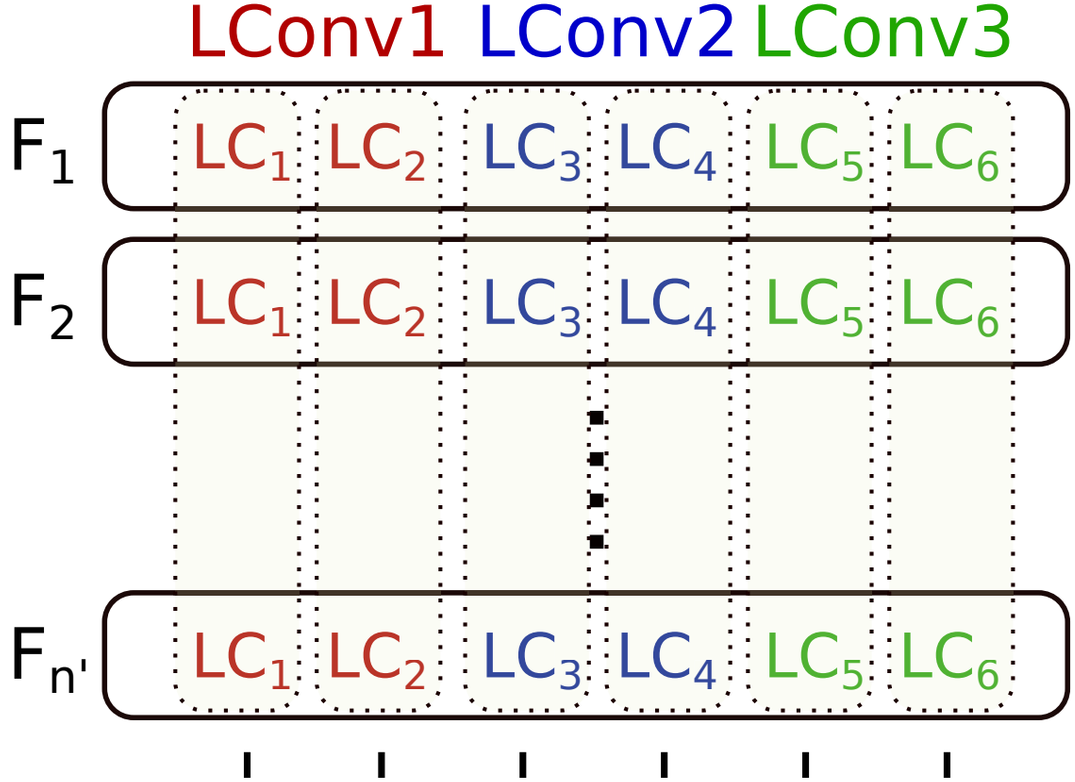
Depth-wise convolution for kernel size k = 3.

Source: Moon et al (2019)

Other CNN Types

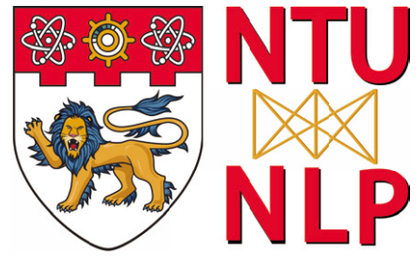
Depth-wise Lightweight Convolution

- The convolutions are done over the input dimensions.
- Consecutive dimensions share parameters



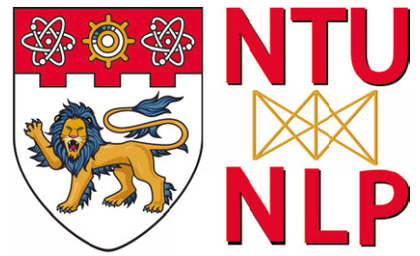
Source: Moon et al (2019)

Lecture Plan



- Classification tasks in NLP
- Window-based Approach for Language Modeling
- Window-based Approach for NER, POS tagging, and Chunking
- Convolutional Neural Net for NLP
- Max-margin Training
- Scaling Softmax (for Language Modeling)
  - Adaptive input and output.

Pairwise (Contrastive) Training



Training losses covered so far

- Mean-Squared Error (MSE) for regression

$$\sum_{i=1}^{N}(y_i - \mathbf{w}^\mathsf{T}\mathbf{x}_i)^2$$

- Cross entropy

binary CE $$-\sum_{i=1}^{N}[y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)]$$

For multi-class $$-\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

Source: Moon et al (2019)

Pairwise (Contrastive) Training

- Cross entropy

Binary CE $$-\sum_{i=1}^{N}[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$$

Sigmoid $$sigm(\mathbf{w}^\mathsf{T}\mathbf{x})$$

For multi-class $$-\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \mu_{ic}$$

Softmax $$\mathcal{P}(y=c|\mathbf{x},\mathbf{W}) = \frac{\exp(\mathbf{w}_c^\mathsf{T}\mathbf{x})}{\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\mathsf{T}\mathbf{x})}$$

Sigmoid

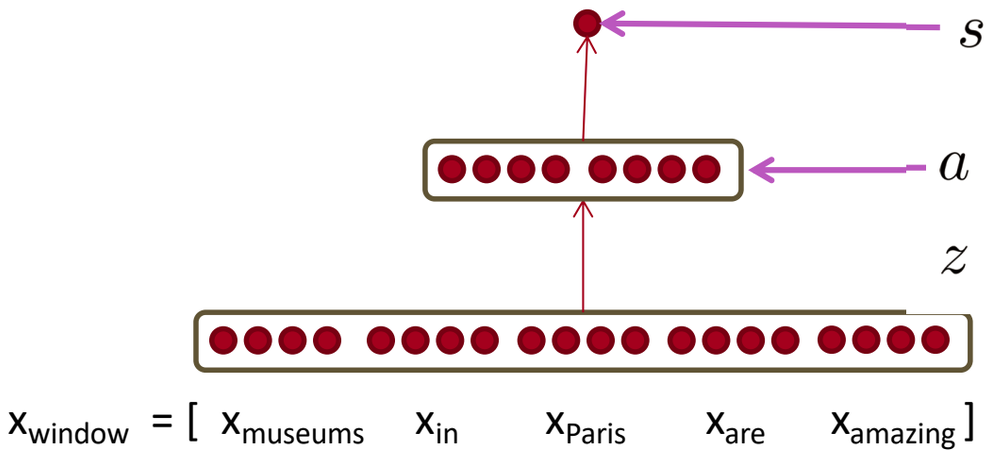
Negative sampling $$-\log\sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k}\mathbb{E}_{j\sim P(w)}\left[\log\sigma\left(-u_j^T v_c\right)\right]$$

- Negative sampling is a type of Noise Contrastive Estimation (NCE)
- Pairwise ranking loss is also similar in spirit

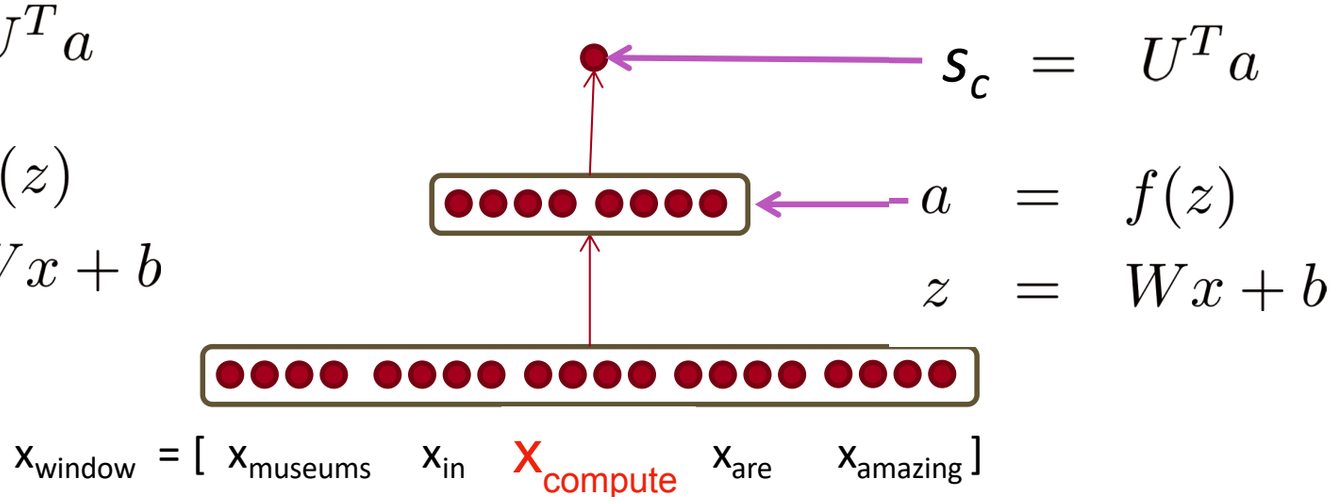
Source: Moon et al (2019)

Pairwise (Contrastive) Training

- Pairwise ranking loss (Hinge loss, Margin loss, Contrastive loss)



Positive example (middle word Paris is original)



Negative example (middle word Paris is a random word)

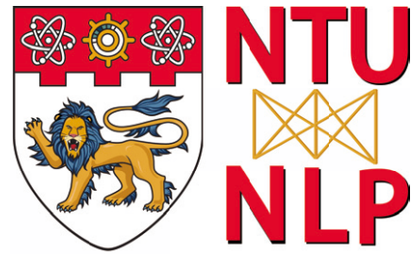
Minimize

$$J = \max(0, 1 - s + s_c)$$

xxx |← 1 →| ooo

See Collobert et al (2011)

Scaling Softmax: Hierarchical Softmax



If h is the last hidden layer of size d and V is the output vocabulary of k items, Softmax layer requires d x k (e.g., 1000 x 200000) computations for every example during training and testing.

- Assign each word w to a unique class c(w).
- For a given input, first predict the class c (using a softmax)
- You do a softmax over all the words for just that cluster.
- Factorize the conditional probability of word w_t given final hidden state h_t as

$$p(w_t \mid h_t) = p_1(\mathcal{C}(w_t) \mid h_t) \times p_2(w_t \mid \mathcal{C}(w_t),\ h_t)$$

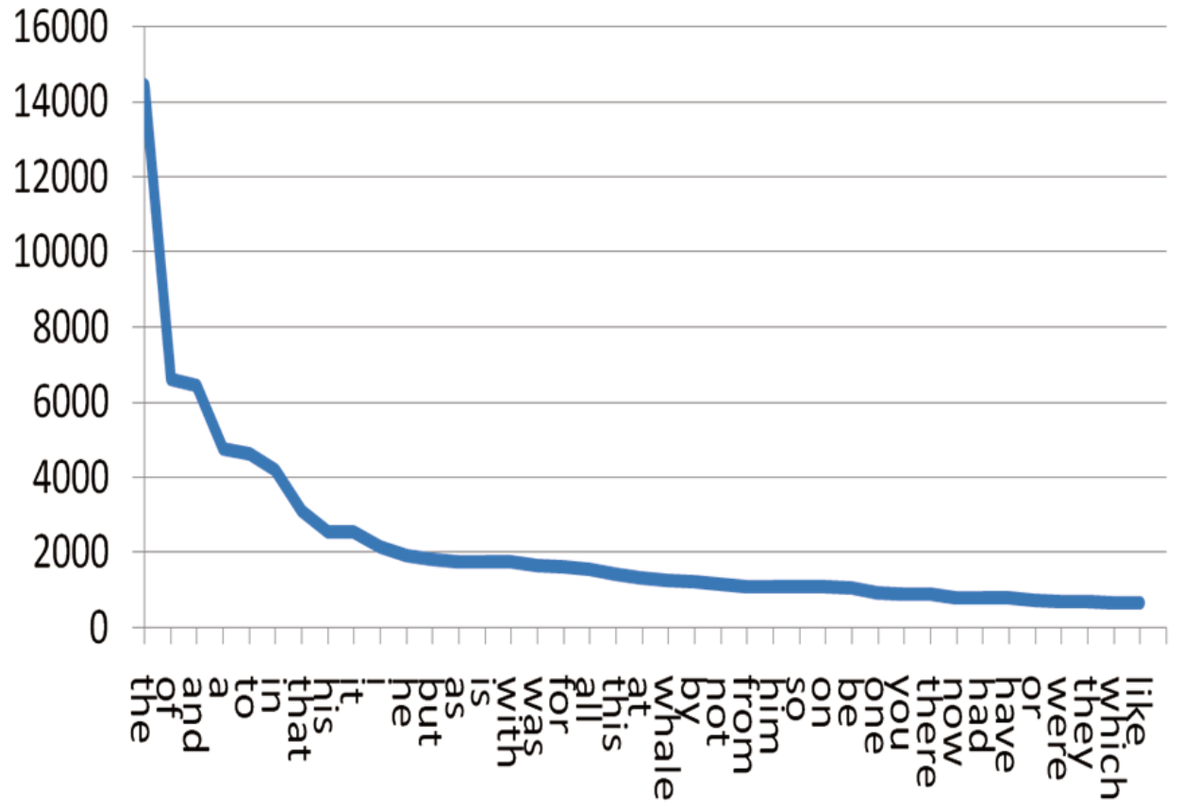
- If each class contains $\sqrt{k}$ words, the computation cost O(d$\sqrt{k}$)

Scaling Softmax: Adaptive Softmax

NTU NLP

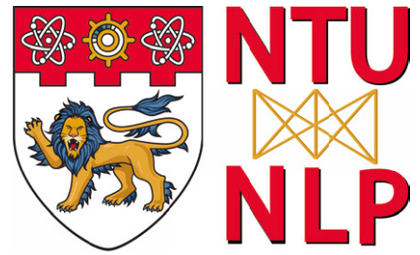
- In natural languages, the distribution of the words follows a Zipf law (Zipf, 1949). Most of the probability mass is covered by a small fraction of the dictionary.

- Partition the dictionary V into two clusters as V_h and V_t, where V_h denotes the head of the distribution consisting of the most frequent words, and V_t is the tail containing a large number of rare words.

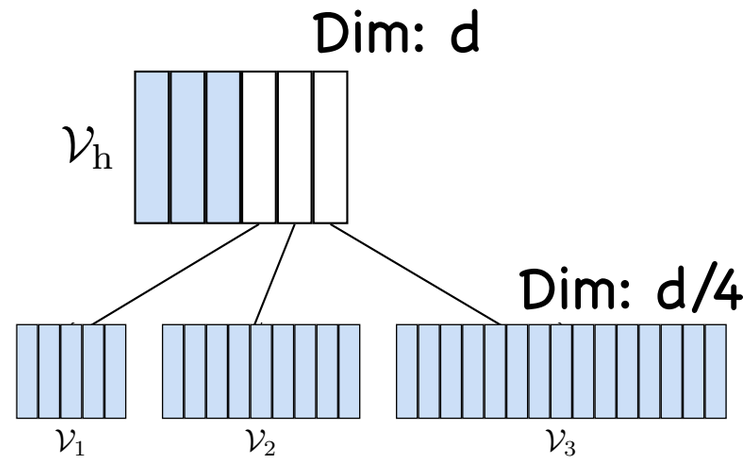


- The partitions have the property: $|\mathcal{V}_{\mathrm{h}}| \ll |\mathcal{V}_{\mathrm{t}}|$ and probabilities $P(\mathcal{V}_{\mathrm{h}}) \gg P(\mathcal{V}_{\mathrm{t}})$

Scaling Softmax: Adaptive Softmax



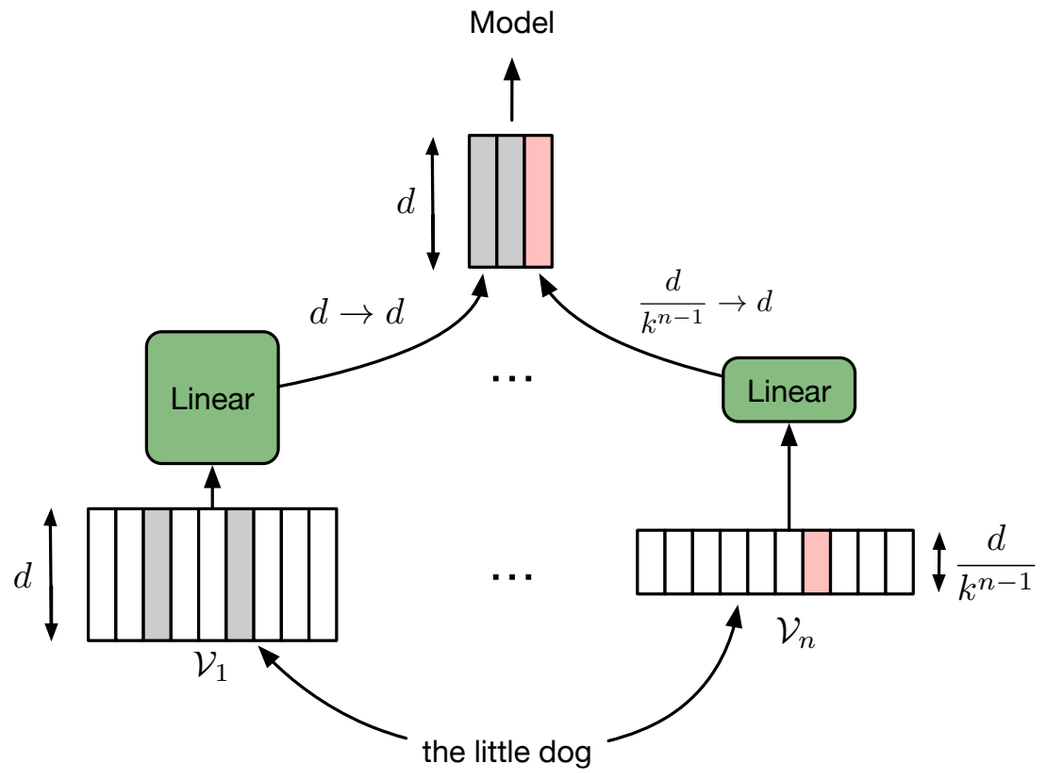
- Put the words in V_h in the root node along with the "other class" ids.
- The tree has two levels.
- If the word w comes from V_h, the probability is simply p(w|h). If it comes from one of the other clusters then the probability is $P(V_i|h) * P(w|V_i, h)$
- Reduce complexity (and parameters) further by having a variable capacity scheme for output word embeddings, assigning more parameters (d for V_h) to frequent words and fewer parameters (d/4 for others) to rare words. This is done by projecting h into h_i of dimensions d/4 (for other classes)



Source: Grave et al, 2017

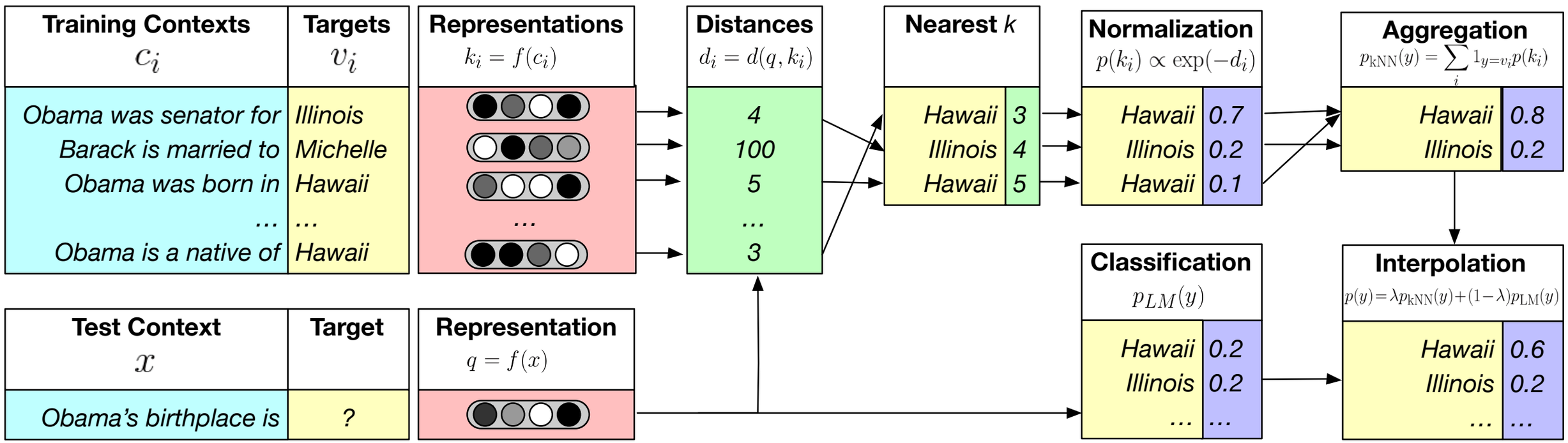
Adaptive Input

- The variable capacity idea can be applied to input word embeddings as well
- Assign more capacity to frequent words and reduce the capacity for less frequent words with the benefit of reducing overfitting to rare words.
- Define a number of clusters that partitions the frequency ordered vocabulary $V = V_1 \cup V_2,\ldots,V_{n-1} \cup V_n$ such that $V_i \cap V_j = \varnothing$ for $\forall i,j$, and I != j, where V_1 contains the most frequent words and V_n the least frequent words.
- Reduce the capacity for each cluster by a factor of k. That is, if words in V_1 have dimension d, then words in V_n have dimension $\frac{d}{k^{n-1}}$



Source: Baevski & Auli, 2019

KNN-LM



- Construct a datastore with an entry for each training token, and an encoding of its context.
- For inference, encode a test context, and retrieve the k most similar training contexts from the datastore, along with the corresponding targets.
- A distribution over targets is computed based on the distance of the corresponding context from the test context. This distribution is then interpolated with the original model's output distribution

Source: Khandelwal et al. 2020

Language Model Results

	Test	Train Time (hours)	Parameters
Dauphin et al. (2017)	31.9	-	428M
Józefowicz et al. (2016)	30.0	-	1,040M
Shazeer et al. (2017)	28.0	-	4,371M†
Char-CNN	25.88	79	366M
Adaptive inputs	25.22	55	331M
Adaptive inputs (large)	23.91	72	465M
Adaptive inputs (very large)	**23.02**	145	1026M
10 LSTMs + SNM10-SKIP (Shazeer et al., 2016)	23.7	-	-

Table 1: Test perplexity on BILLION WORD. Adaptive inputs share parameters with an adaptive softmax. Training times of Char-CNN and Adaptive input models are measured when training with 64 GPUs.
†does not include embedding and softmax layers

	Test	Train Time (hours)	Parameters
Grave et al. (2016)	40.8	-	
Dauphin et al. (2017)	37.2	-	229M
Merity et al. (2018)	33.0	-	151M
Rae et al. (2018)	29.2	-	
Adaptive inputs	**18.7**	67	247M

Table 2: Test perplexity on WIKITEXT-103 (cf. Table 1). Training time is based on 8 GPUs.

Source: Baevski & Auli, 2019

Language Model Results

Model	Perplexity (↓) Dev	Test	# Trainable Params
Baevski & Auli (2019)	17.96	18.65	247M
+Transformer-XL (Dai et al., 2019)	-	18.30	257M
+Phrase Induction (Luo et al., 2019)	-	17.40	257M
Base LM (Baevski & Auli, 2019)	17.96	18.65	247M
+kNN-LM	**16.06**	**16.12**	247M
+Continuous Cache (Grave et al., 2017c)	17.67	18.27	247M
+kNN-LM + Continuous Cache	**15.81**	**15.79**	247M

Table 1: Performance on WIKITEXT-103. The kNN-LM substantially outperforms existing work. Gains are additive with the related but orthogonal continuous cache, allowing us to improve the base model by almost 3 perplexity points with no additional training. We report the median of three random seeds.

Model	Perplexity (↓) Dev	Test	# Trainable Params
Base LM (Baevski & Auli, 2019)	14.75	11.89	247M
+kNN-LM	**14.20**	**10.89**	247M

Table 2: Performance on BOOKS, showing that kNN-LM works well in multiple domains.

Source: Khandelwal et al. 2020