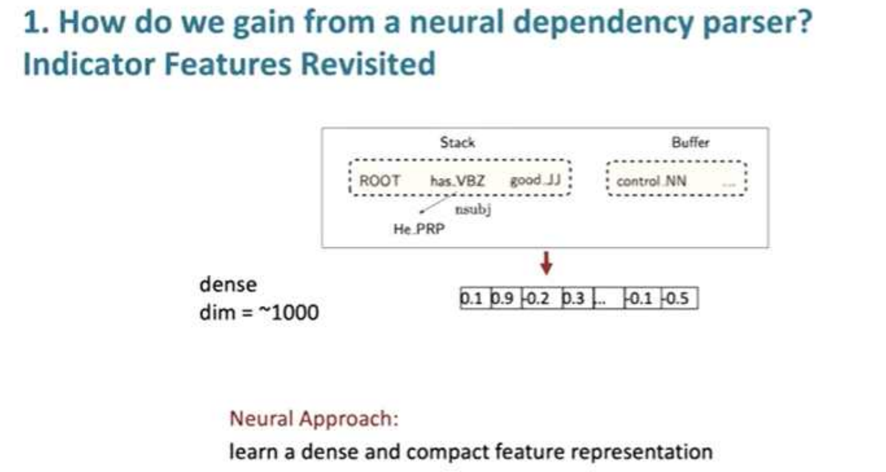
Lec 5 - Recurrent Neural networks(RNNs)

1. Neural dependency parsing

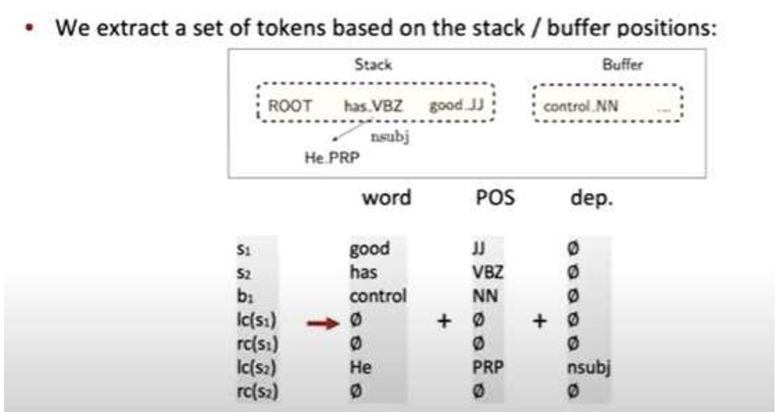
- problems: 1. sparse, 2. incomplete 3. expensive computation

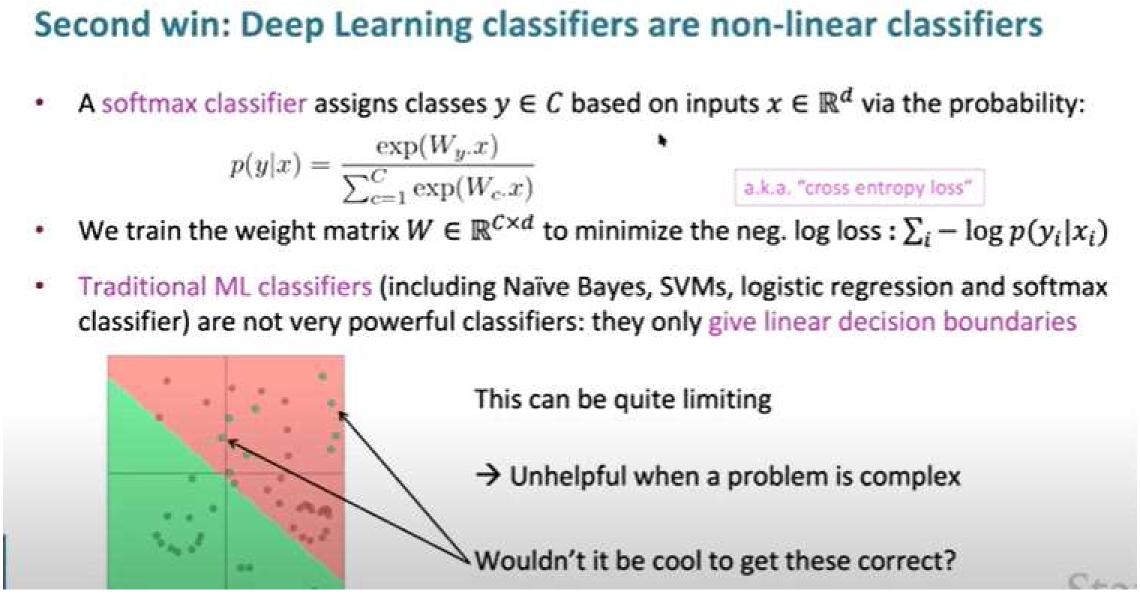


Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

POS and dependency labels are represented as d-dimensional vectors.

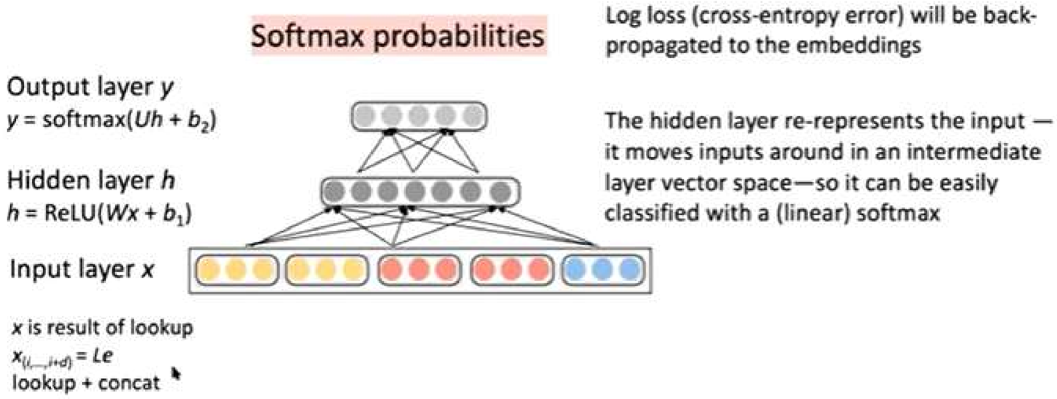
extracting tokens&vector representations from configuration



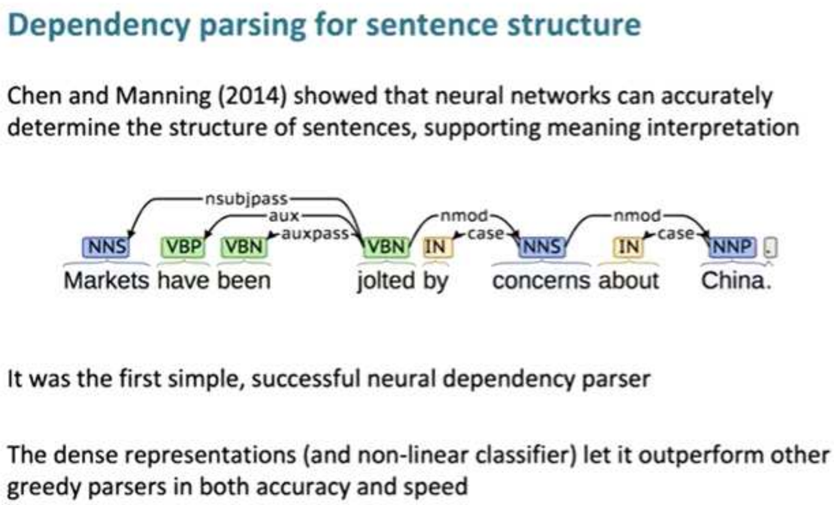


Neural networks are more powerful: nonlinear decision boundaries(complex function)

- simple feed-forward neural network multi-class classifier

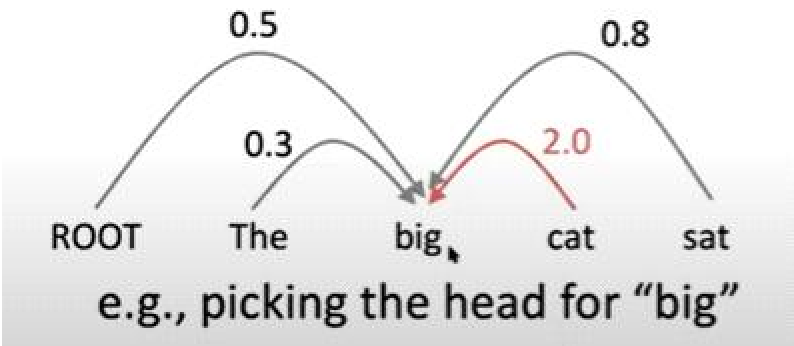


-ReLU

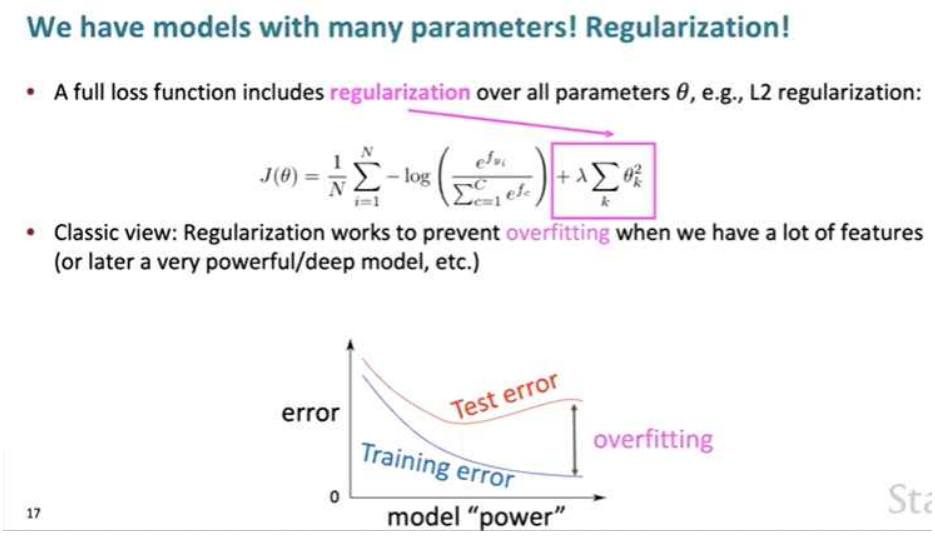


- Dependency parsing for sentence structure

-Graph-based dependency parsers (eg. MST algorithm)



2. A bit more about neural networks



- vectorization:

- E.g., looping over word vectors versus concatenating them all into one large matrix and then multiplying the softmax weights with that matrix:

```python
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

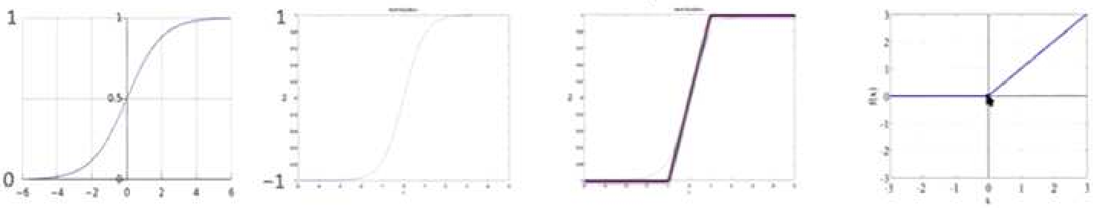
- 1000 loops, best of 3: **639 µs** per loop
  10000 loops, best of 3: **53.8 µs** per loop ← Now using a single a C x N matrix
- Matrices are awesome!!! Always try to use vectors and matrices rather than for loops!
- The speed gain goes from 1 to 2 orders of magnitude with GPUs!

Sta

- Non-linearities, old and new

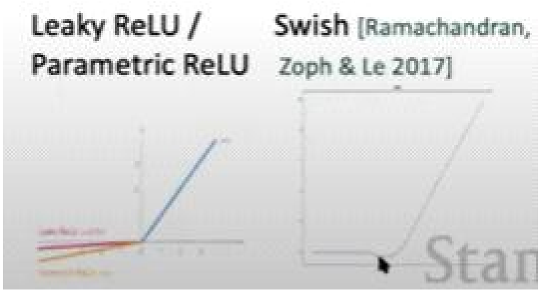
Non-linearities, old and new

logistic ("sigmoid")	tanh	hard tanh	ReLU (Rectified Linear Unit)
$f(z) = \frac{1}{1+\exp(-z)}$.	$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$,	$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$	$\text{rect}(z) = \max(z,0)$



tanh is just a rescaled and shifted sigmoid (2 × as steep, [−1,1]):

$$\tanh(z) = 2\text{logistic}(2z) - 1$$



-parameter intialization

- You normally must initialize weights to small random values (i.e., not zero matrices!)
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights** ~ Uniform($-r$, r), with r chosen so numbers get neither too big or too small [later the need for this is removed with use of layer normalization]
- Xavier initialization has variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size):

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

-Optimizers: 대부분 SGD 잘 작동(hand tuning learning rate 힘듦)

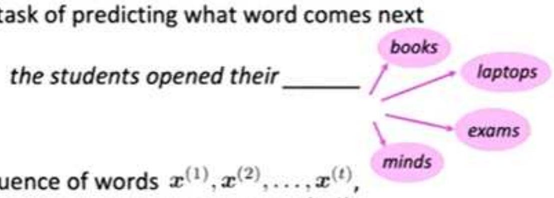
-Learning Rates

Learning Rates

- You can just use a constant learning rate. Start around $lr = 0.001$?
  - It must be order of magnitude right – try powers of 10
    - Too big: model may diverge or not converge
    - Too small: your model may not have trained by the assignment deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every k epochs
    - An epoch = a pass through the data (shuffled or sampled – not in same order each time)
  - By a formula: $lr = lr_0 e^{-kt}$, for epoch t
  - There are fancier methods like cyclic learning rates (q.v.)
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so you may want to start with a higher learning rate

3. Language modeling+RNNs

- Language Modeling is the task of predicting what word comes next



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

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

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

What is language modeling? - is the task of predicting what word comes next/as a system that assigns probabilties to a piece of text

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

$$P(x^{(1)}, \dots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \cdots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)})$$
$$= \prod_{t=1}^{T} \underbrace{P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)})}_{\text{This is what our LM provides}}$$

- eg. 메시지 자동완성

- n-gram Language Models

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding n-1 words

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

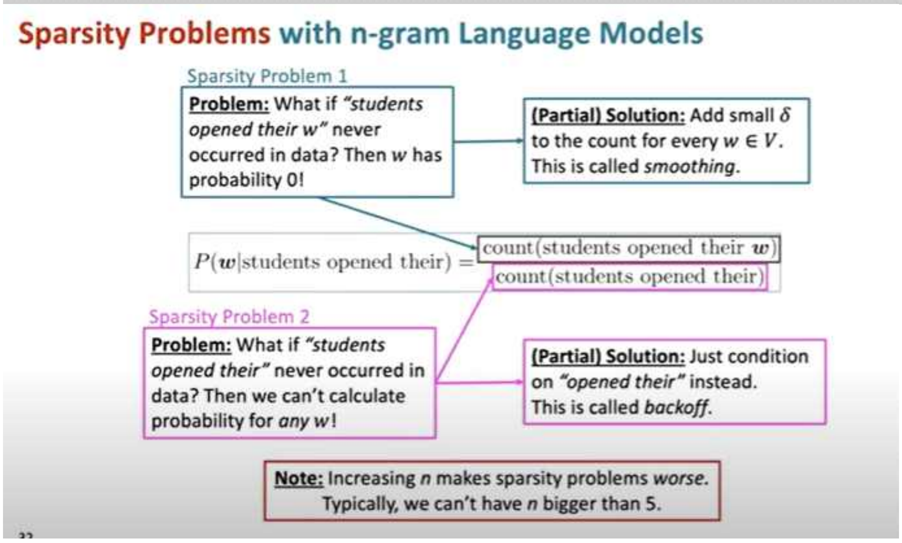
$$= \frac{\boxed{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}}{\boxed{P(x^{(t)}, \dots, x^{(t-n+2)})}} \qquad \text{(definition of conditional prob)}$$

prob of a n-gram

prob of a (n-1)-gram

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

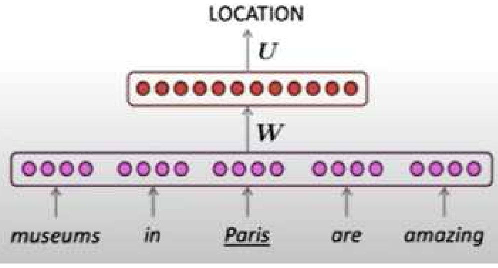
$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \qquad \text{(statistical approximation)}$$



How to build a neural Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition in Lecture 3:

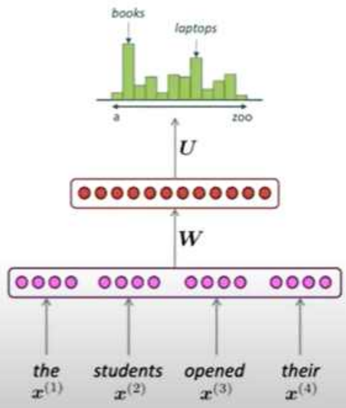


A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:
- No sparsity problem
- Don't need to store all observed *n*-grams

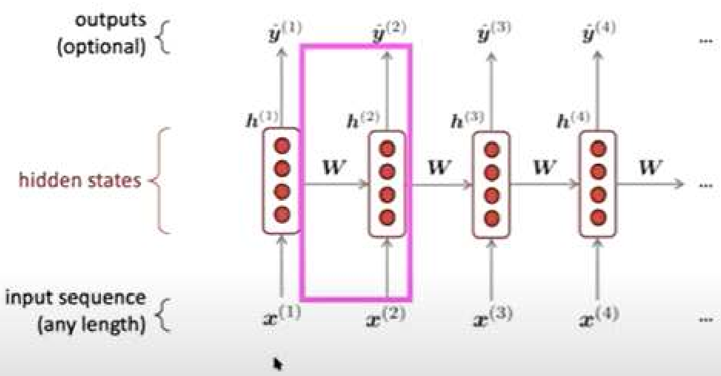
Remaining **problems**:
- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.



Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*



<A simple RNN Language Model>