Machine Learning course,
part 2

Lecture 2: CNN and vanishing gradient

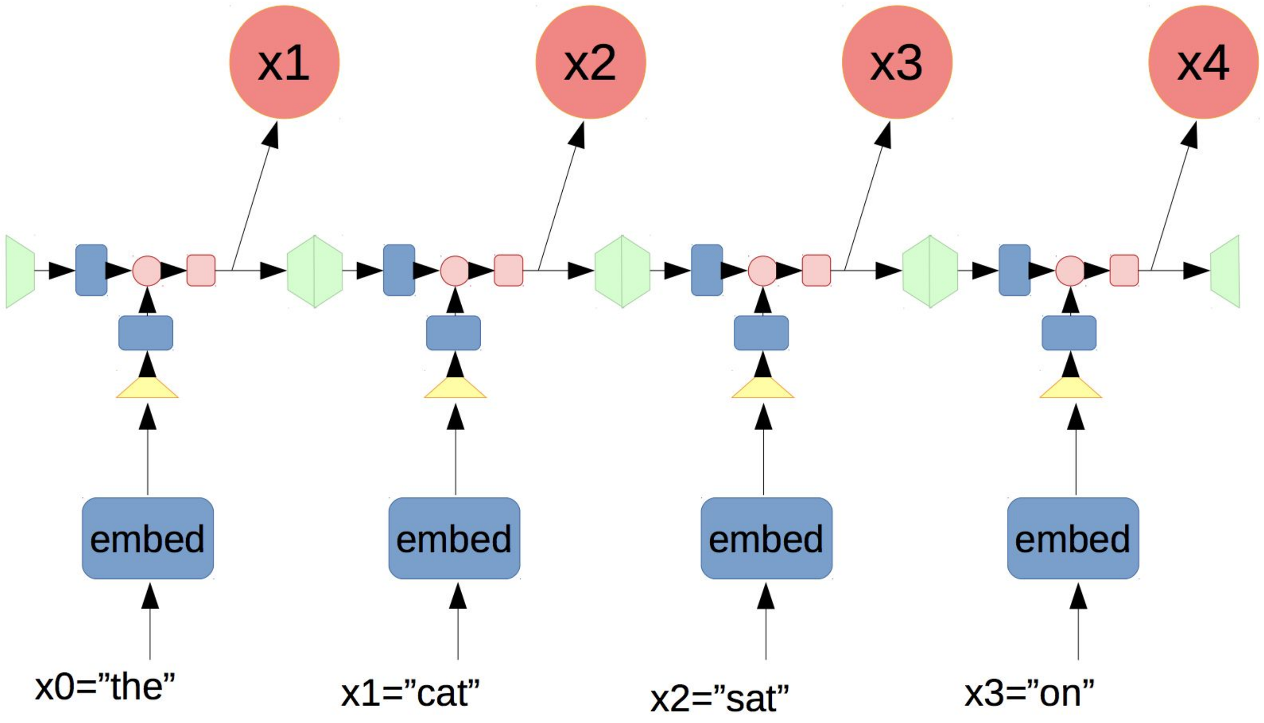
Radoslav Neychev
Ivan Provilkov

MIPT
13.09.2019

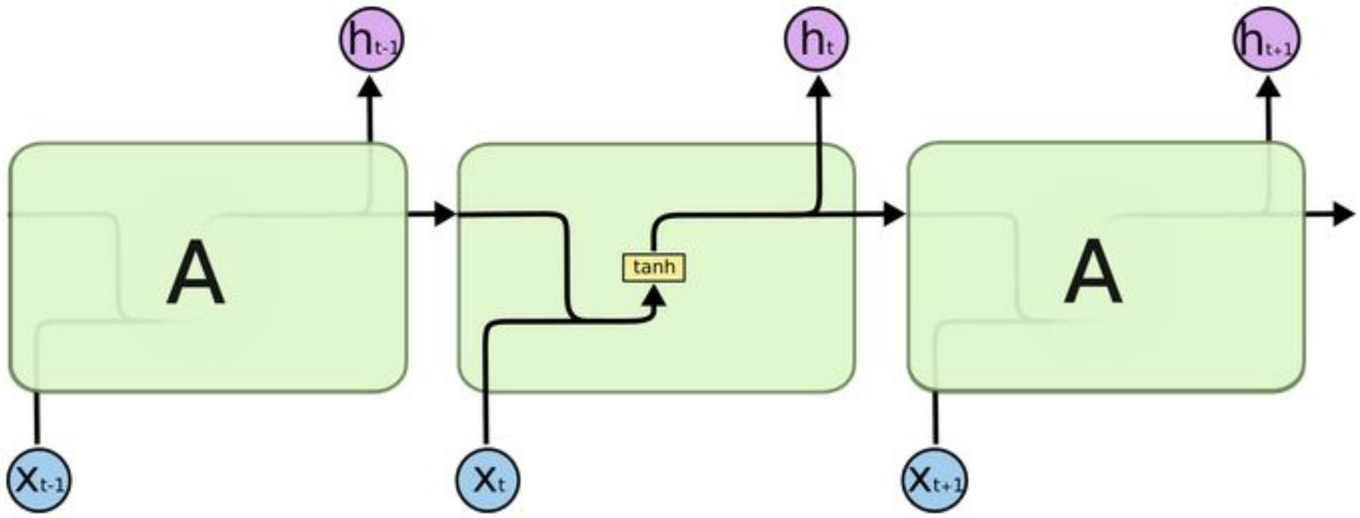
Outline

- Simple RNN recap
- Complex RNN:
  - Vanishing gradient
  - Exploding gradient
  - LSTM/GRU
  - Gradient clipping
  - Skip connections
  - Residual networks as ensembles
- CNNs for text
- Text segmentation

Recap: RNN

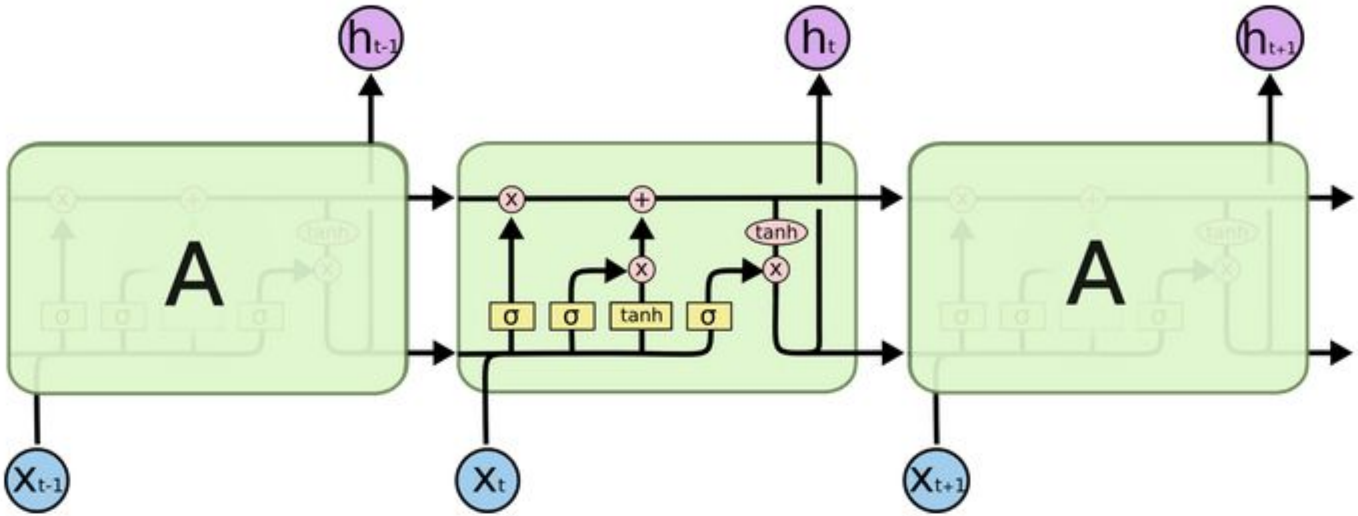


Recap: Vanilla RNN



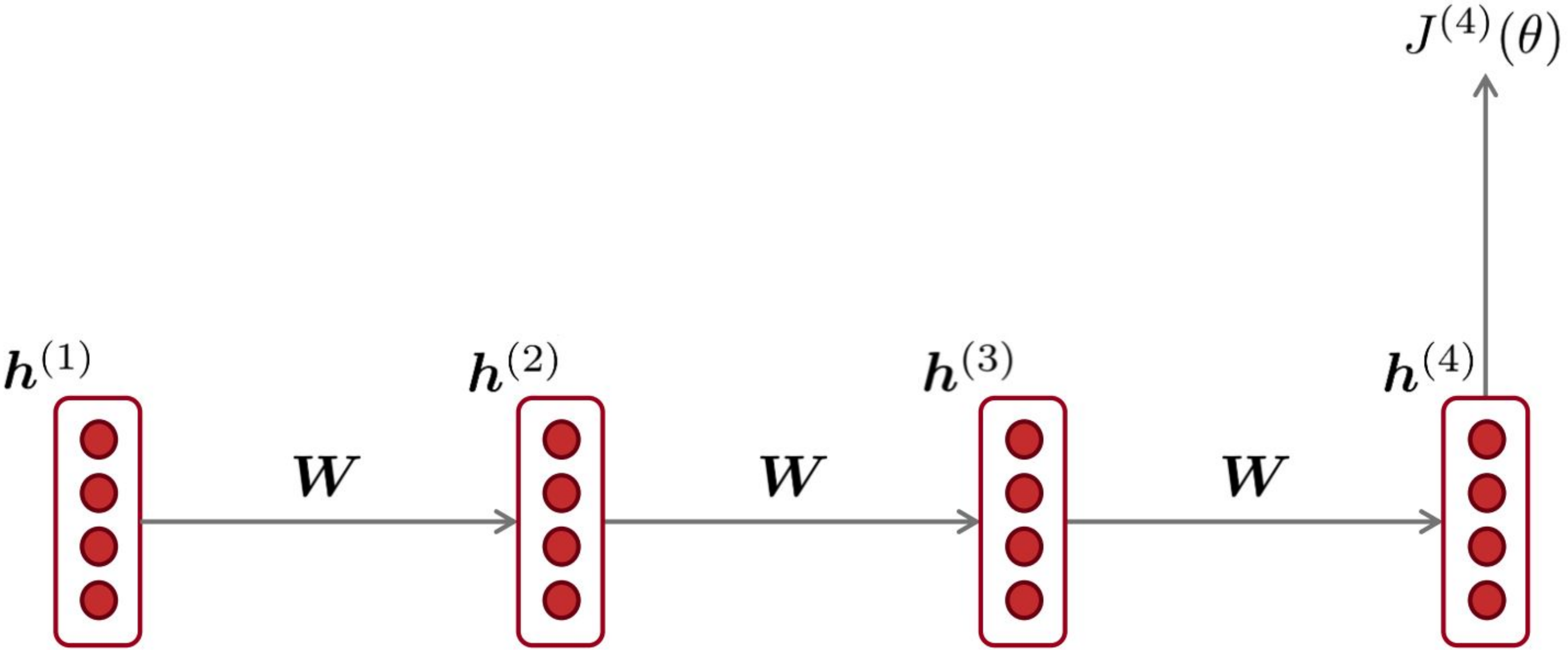
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: LSTM



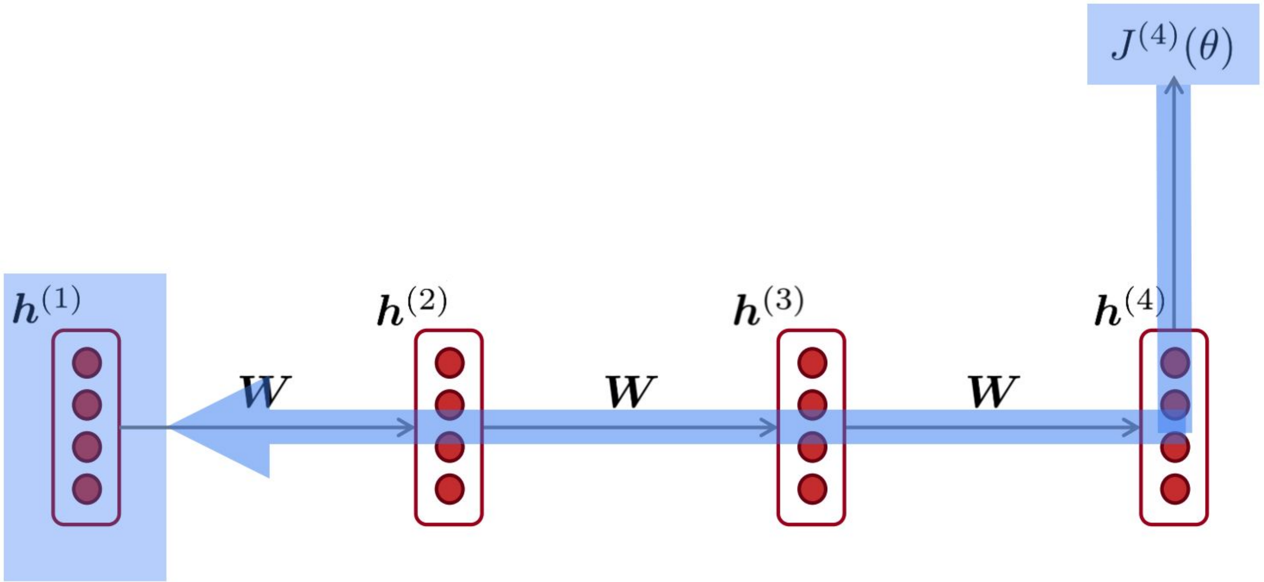
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing gradient problem



Based on: Lecture by Abigail See, [CS224n Lecture 7]

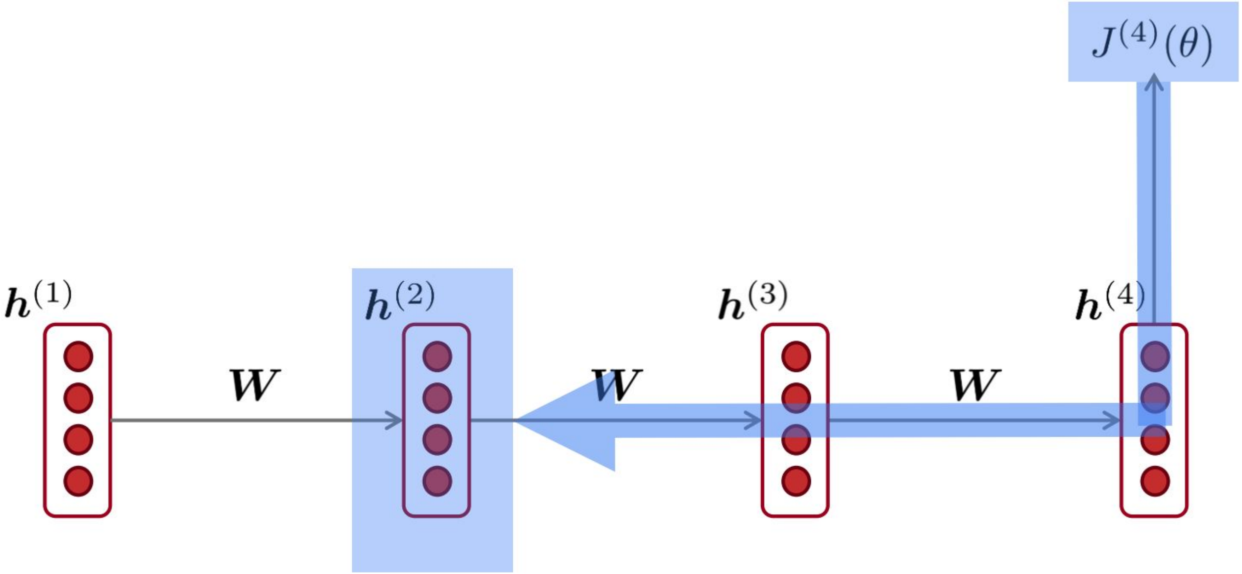
Vanishing gradient problem



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem

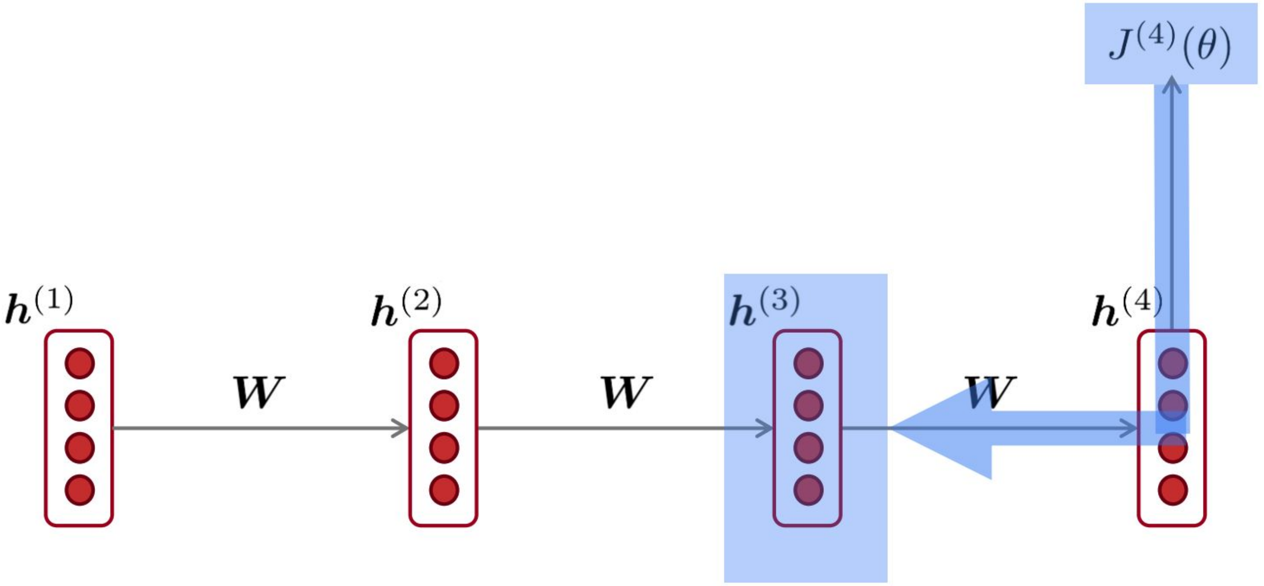


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(2)}}$$

chain rule!

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem

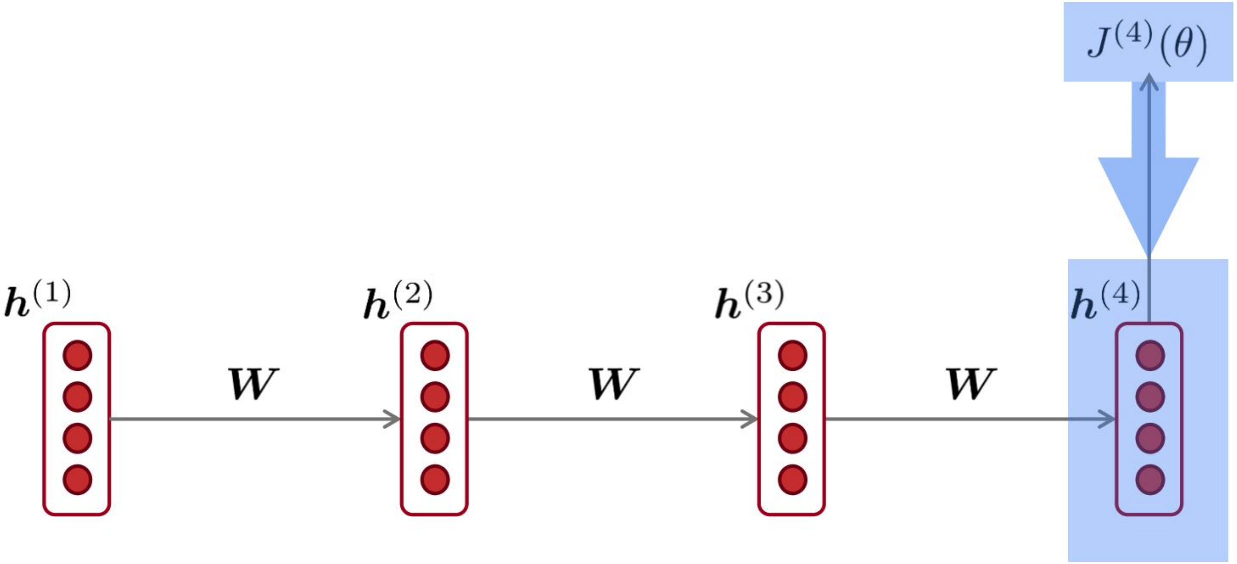


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \quad \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(3)}}$$

chain rule!

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

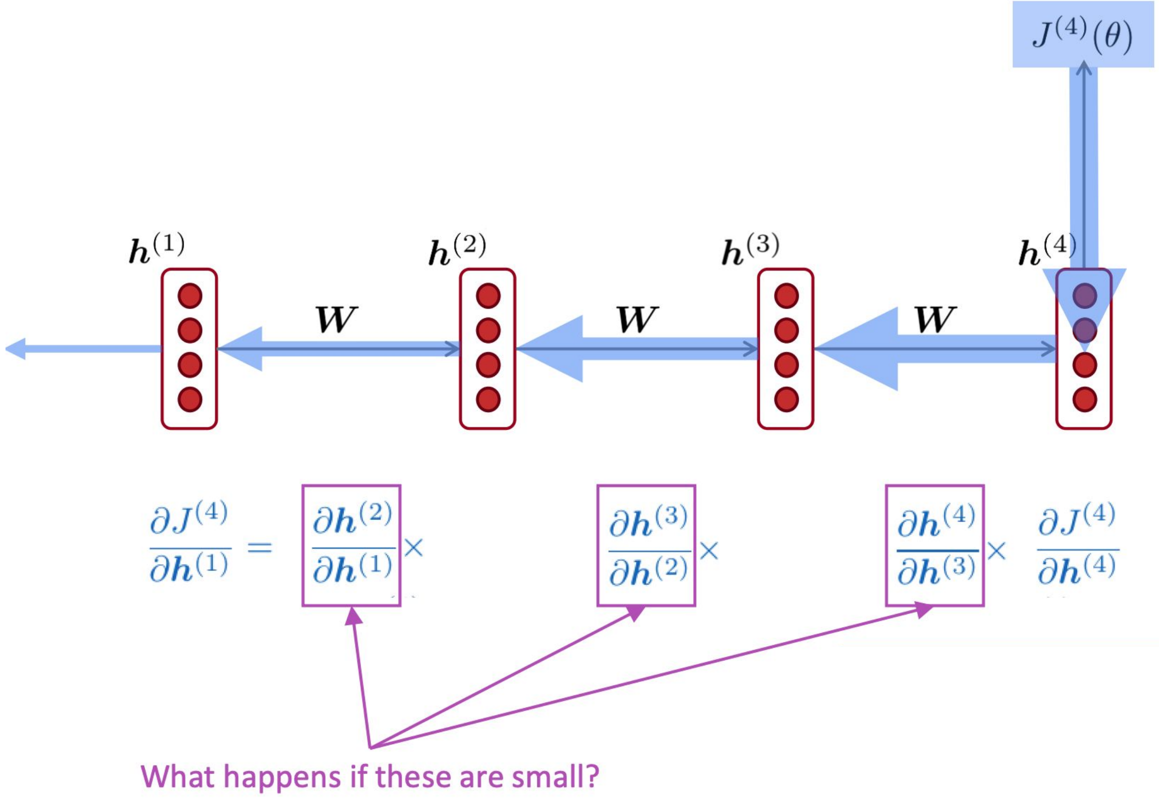
chain rule!

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem

Vanishing gradient problem:

When the derivatives are small, the gradient signal gets smaller and smaller as it backpropagates further

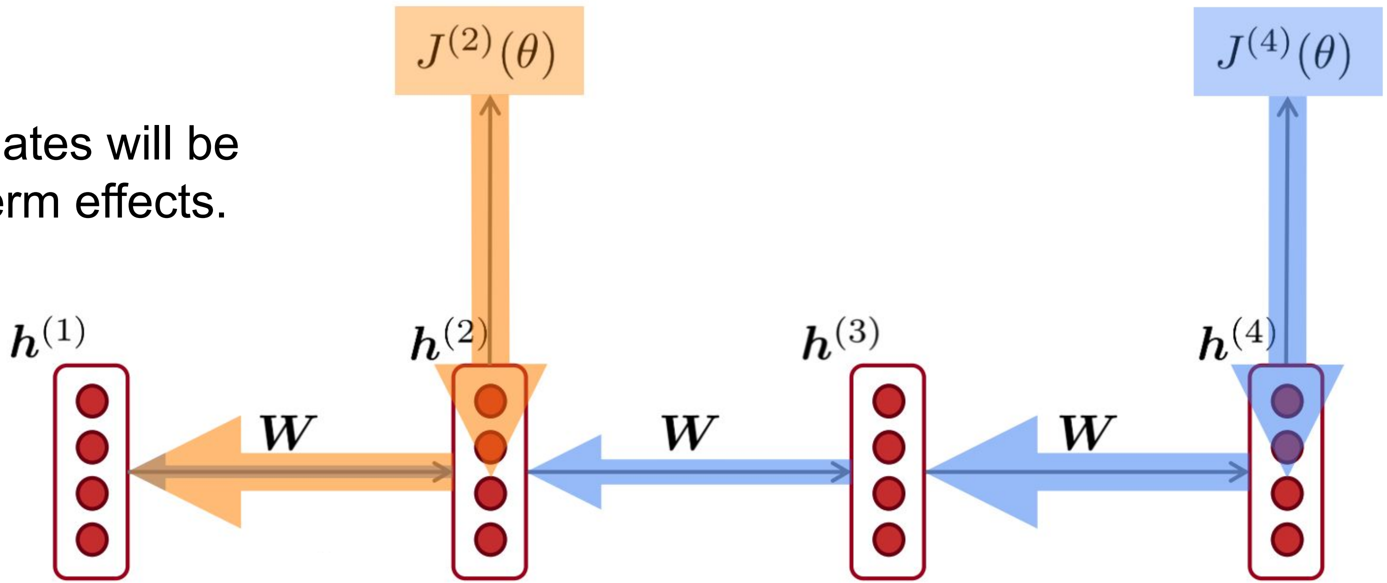


More info: “On the difficulty of training recurrent neural networks”, Pascanu et al, 2013
http://proceedings.mlr.press/v28/pascanu13.pdf

Vanishing gradient problem

Gradient signal from far away is lost because it's much smaller than from close-by.

So model weights updates will be based only on short-term effects.



Based on: Lecture by Abigail See, CS224n Lecture 7

Exploding gradient problem

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Exploding gradient solution

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

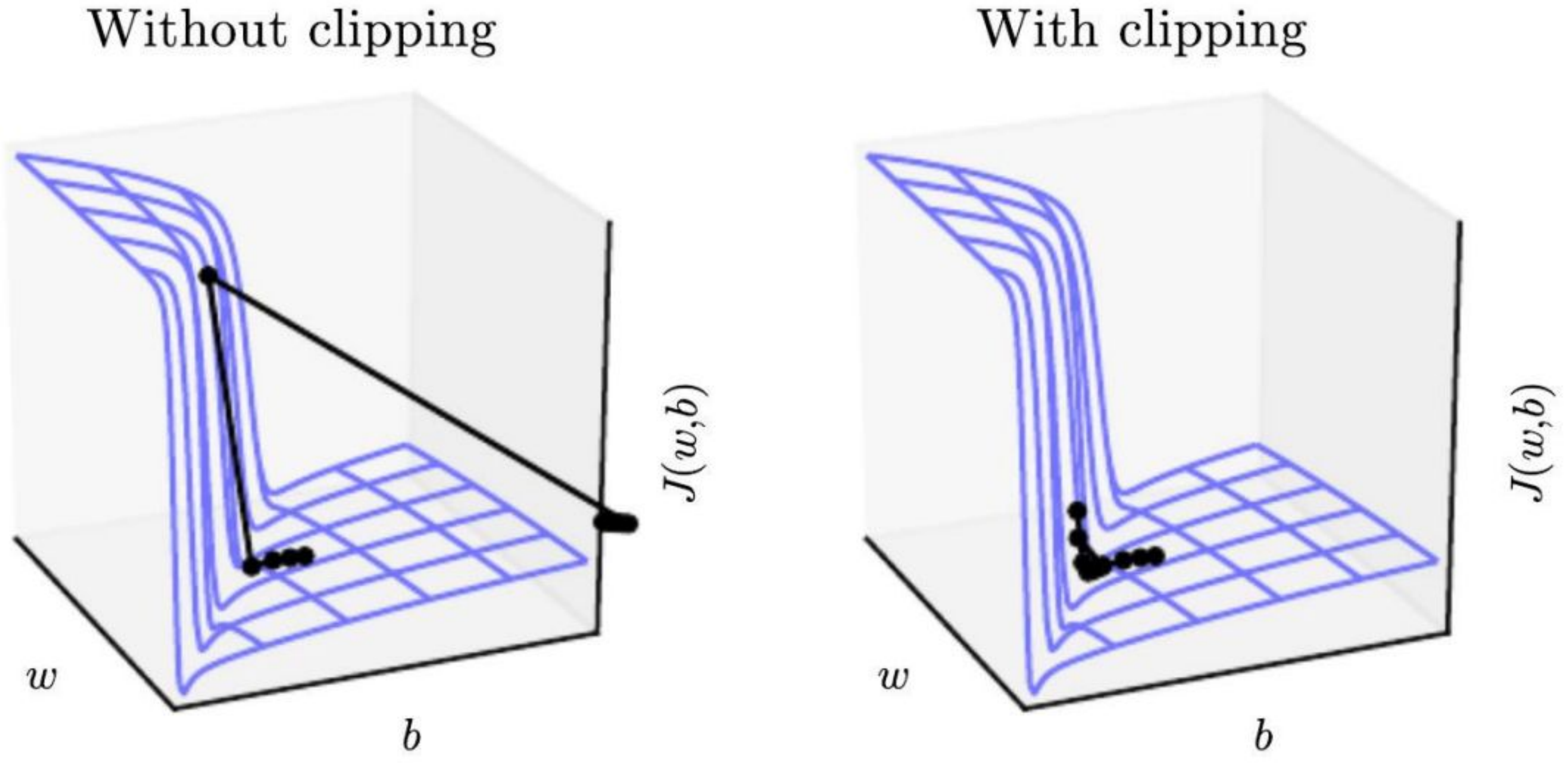
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Intuition: take a step in the same direction, but a smaller step

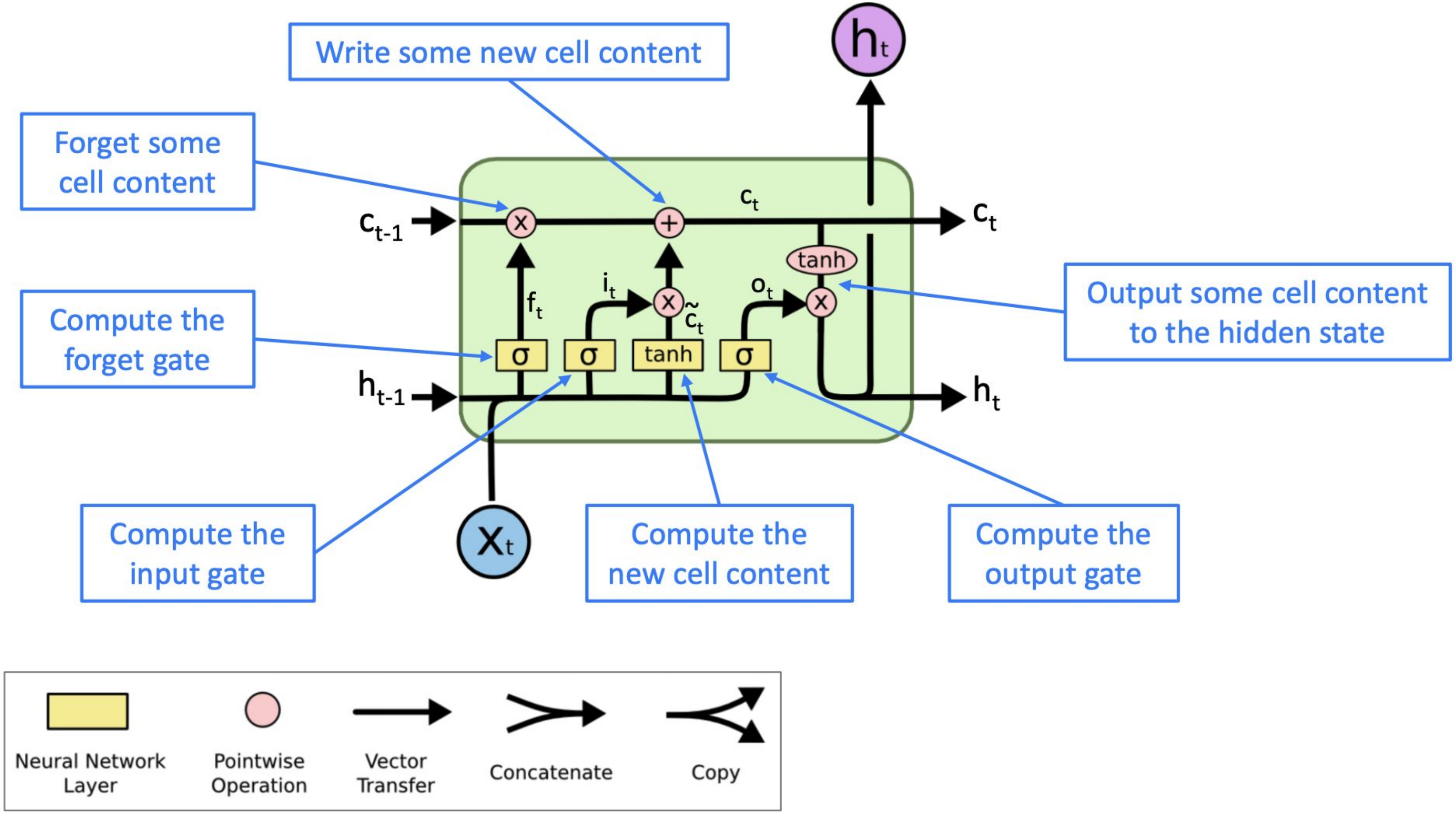
Based on: Lecture by Abigail See, [CS224n Lecture 7]

Exploding gradient solution



Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient: LSTM



Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient: LSTM

Sigmoid function: all gate values are between 0 and 1

Forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

Input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

Output gate: controls what parts of cell are output to hidden state

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

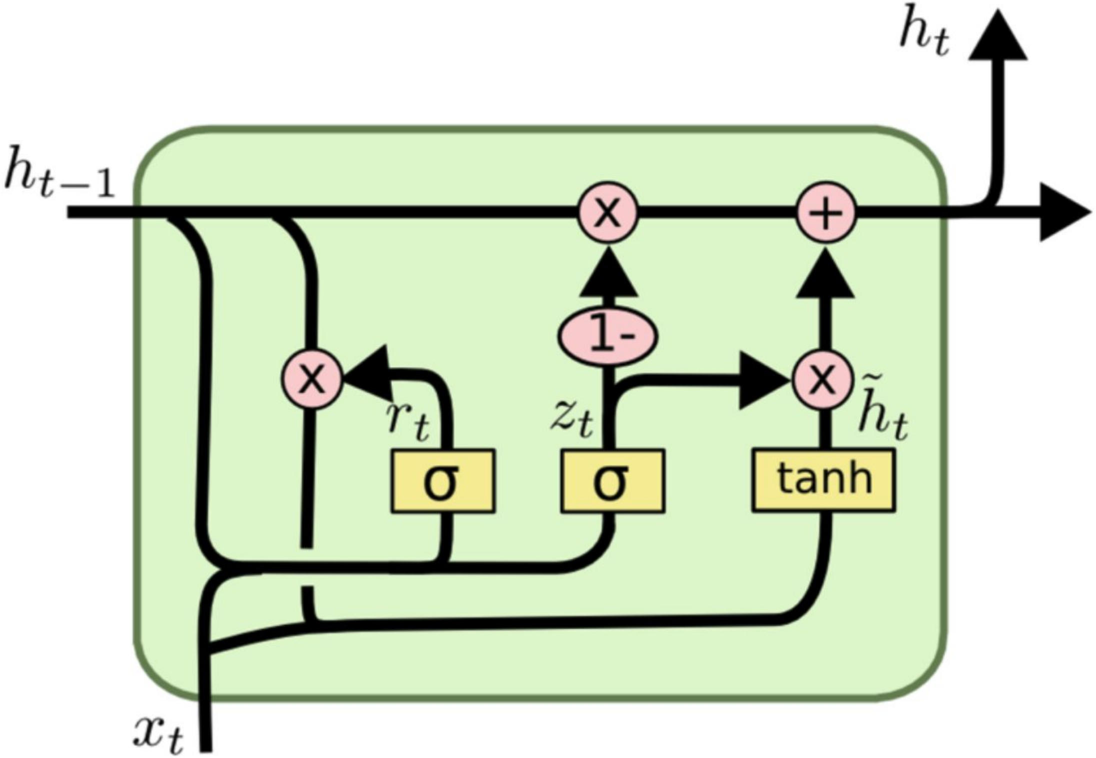
Hidden state: read ("output") some content from the cell

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length *n*

Gates are applied using element-wise product

Vanishing gradient: GRU



Vanishing gradient: GRU

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

Vanishing gradient: LSTM vs GRU

- LSTM and GRU are both great
  - GRU is quicker to compute and has fewer parameters than LSTM
  - There is no conclusive evidence that one consistently performs better than the other
  - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)

Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution**: direct (or skip-) connections (just like in ResNet)

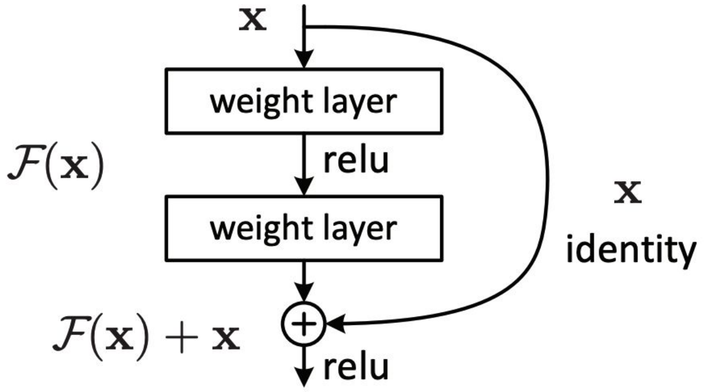


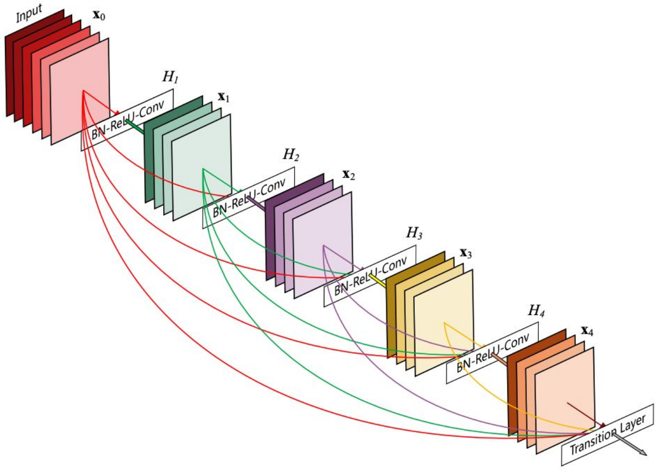
Figure 2. Residual learning: a building block.

Source: "Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution**: dense connections (just like in DenseNet)



Source: ”Densely Connected Convolutional Networks", Huang et al, 2017 https://arxiv.org/pdf/1608.06993.pdf

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

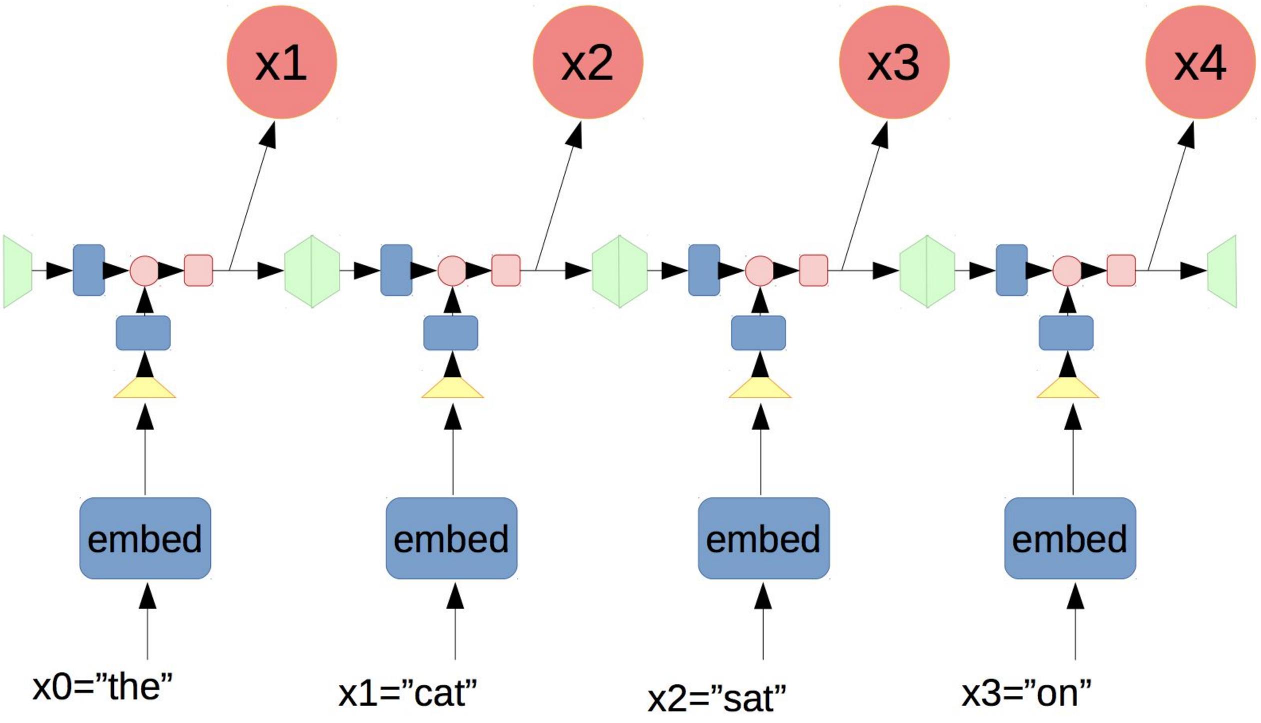
- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution(but not actually for that problem)**: dense connections (just like in DenseNet)

Conclusion:

Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]. Gradients magnitude drops exponentially with connection length.

Source: ”Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

Recap: RNN



From RNN to CNN

- RNN: Get compositional vectors for grammatical phrases only

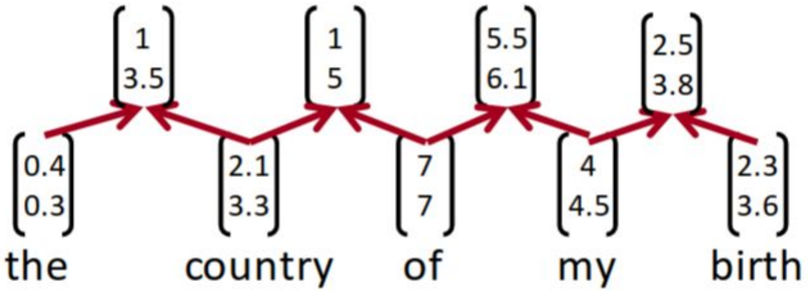
- CNN: What if we compute vectors for every possible phrase?
  - Example: *"the country of my birth"* computes vectors for:
    - *the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth*

- Regardless of whether it is grammatical
- Wouldn't need parser
- Not very linguistically or cognitively plausible

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

From RNN to CNN

- Imagine using only bigrams



- Same operation as in RNN, but for every pair

$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

- Can be interpreted as convolution over the word vectors

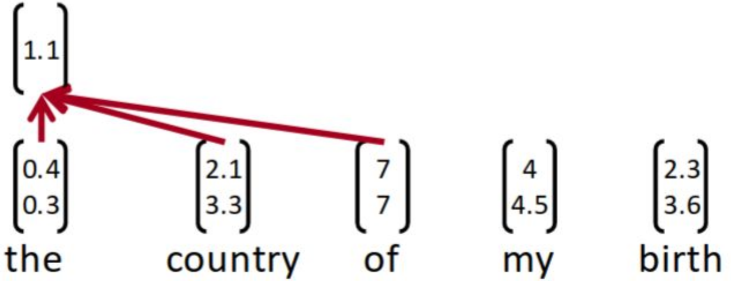
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

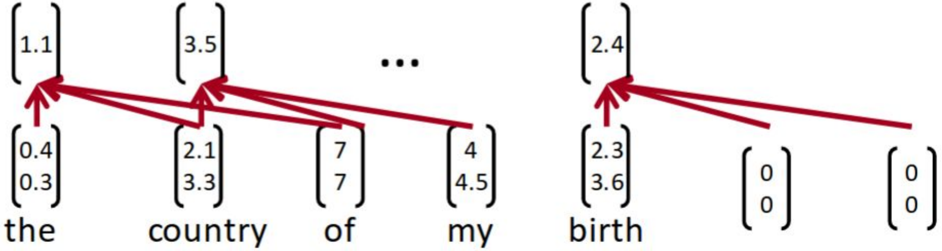
From RNN to CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

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

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





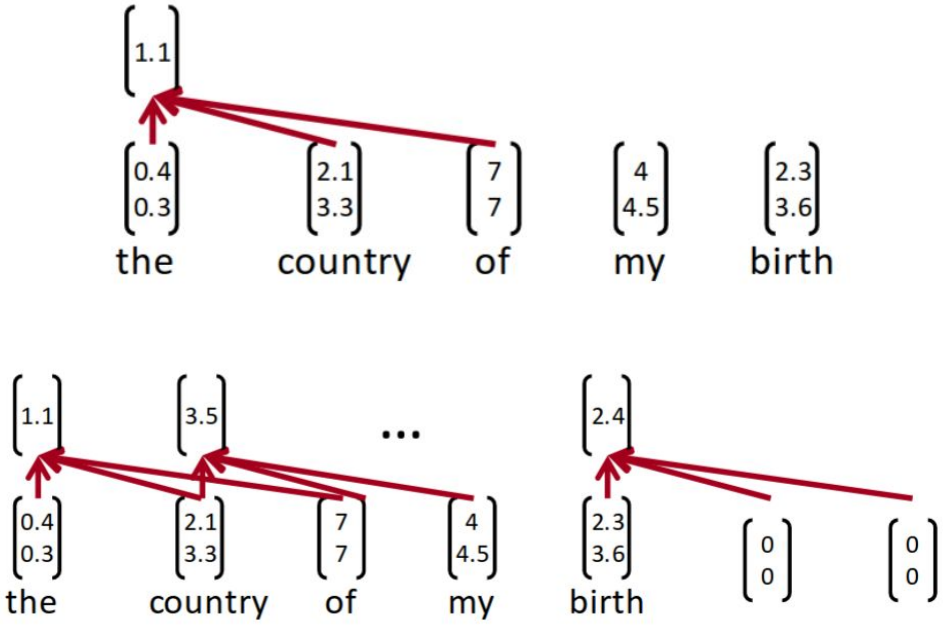
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

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

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



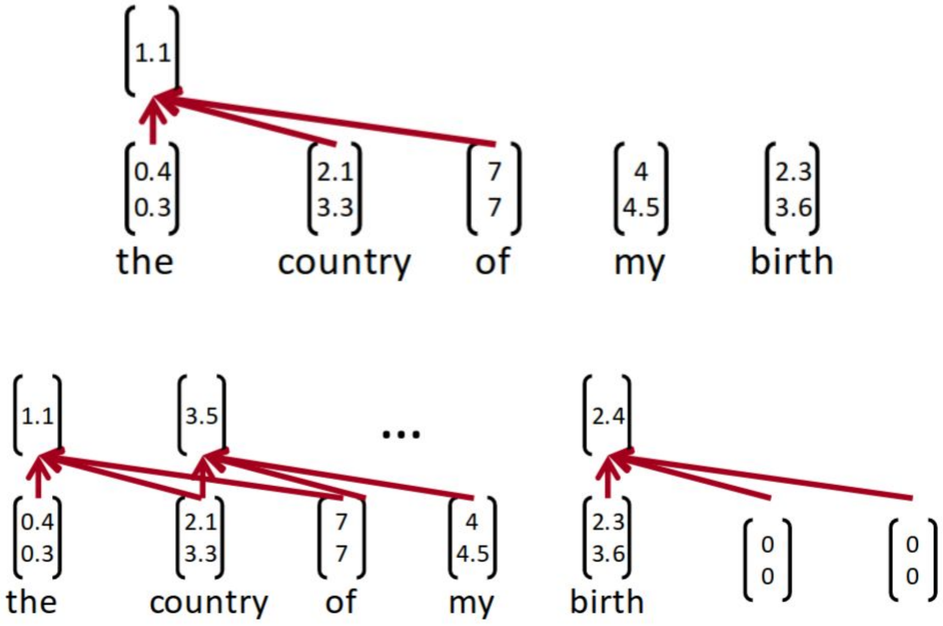
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

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

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



What's next?

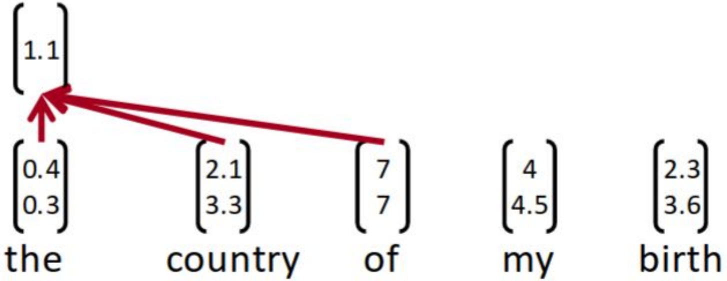
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

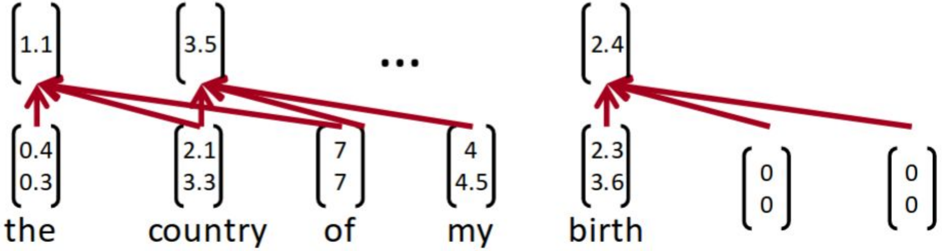
One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

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

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





What's next?

We need more features!

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

One layer CNN

- Feature representation is based on some applied filter:

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

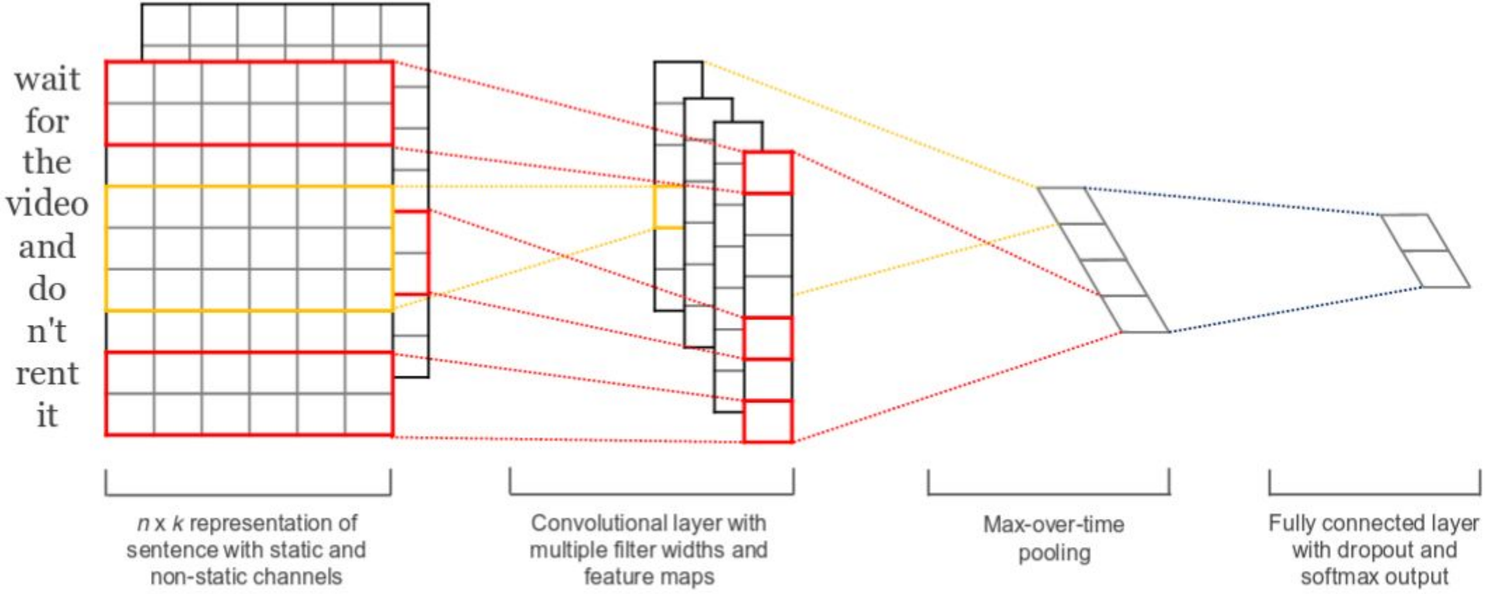
- Let's use pooling over the time axis: $\hat{c} = \max\{\mathbf{c}\}$

- Now the length of **c** is irrelevant!

- So we can use filters based on unigrams, bigrams, tri-grams, 4-grams, etc.

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

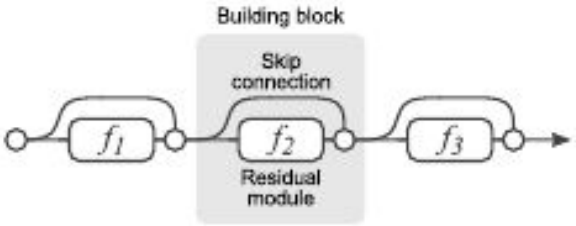
Another example from Kim (2014) paper



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

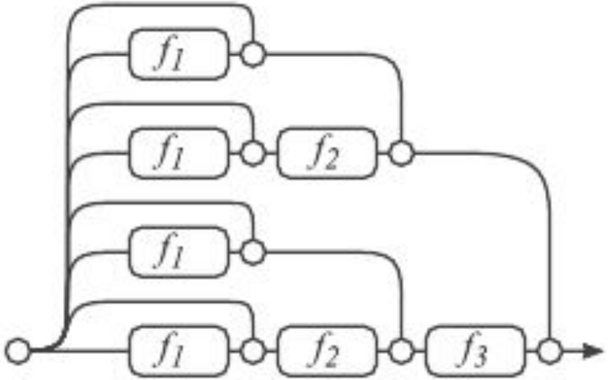
Another view on ResNets and vanishing gradient

“**Residual Networks Behave Like Ensembles of Relatively Shallow Networks**”



(a) Conventional 3-block residual network

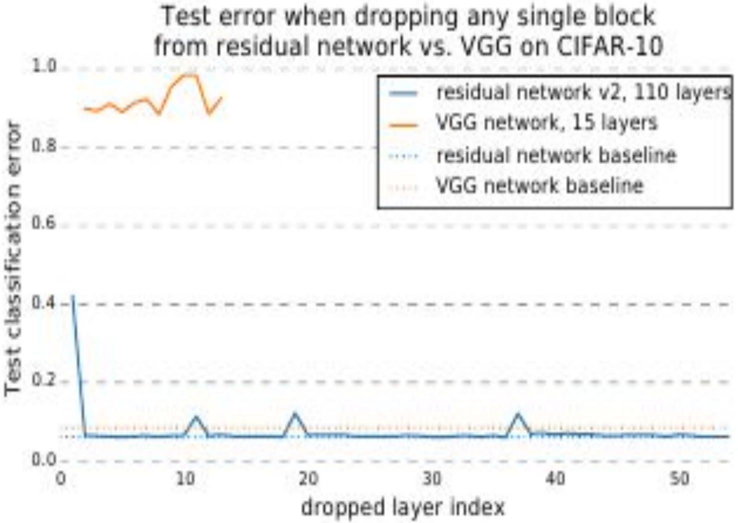
=

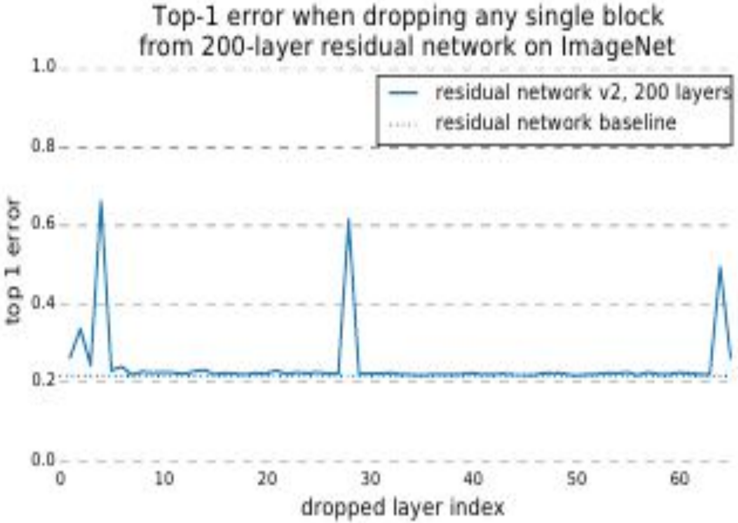


(b) Unraveled view of (a)

Source: https://arxiv.org/pdf/1605.06431.pdf

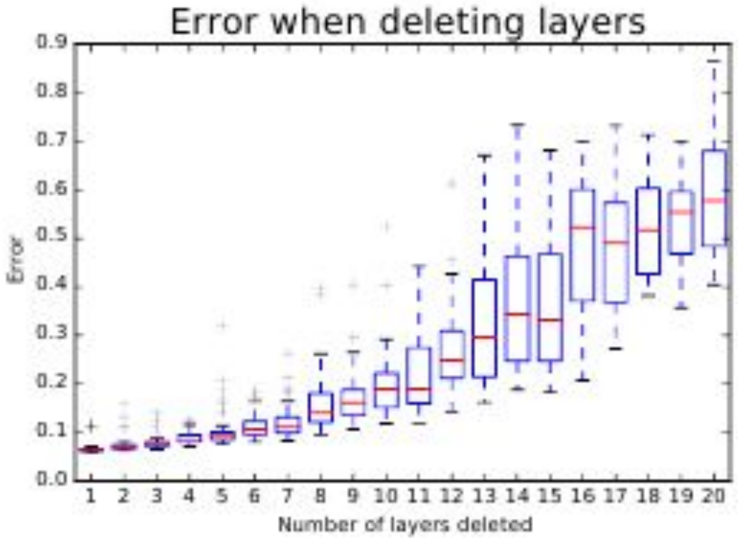
“Residual Networks Behave Like Ensembles of Relatively Shallow Networks”



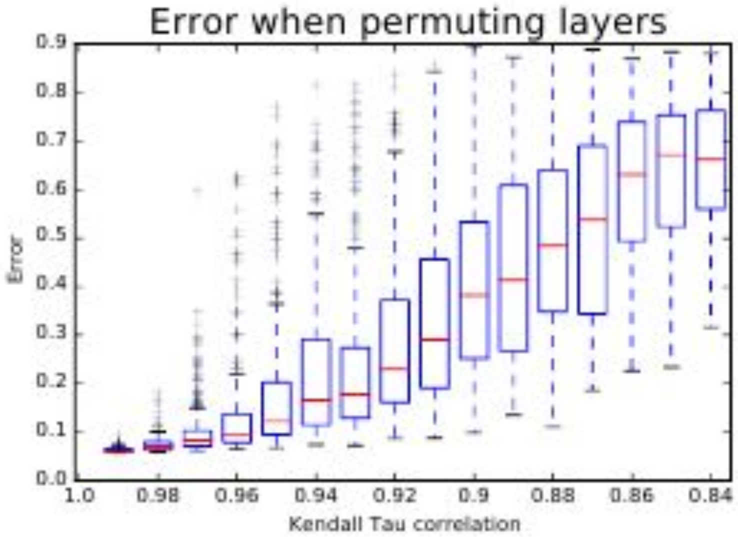


https://arxiv.org/pdf/1605.06431.pdf

“Residual Networks Behave Like Ensembles of Relatively Shallow Networks”



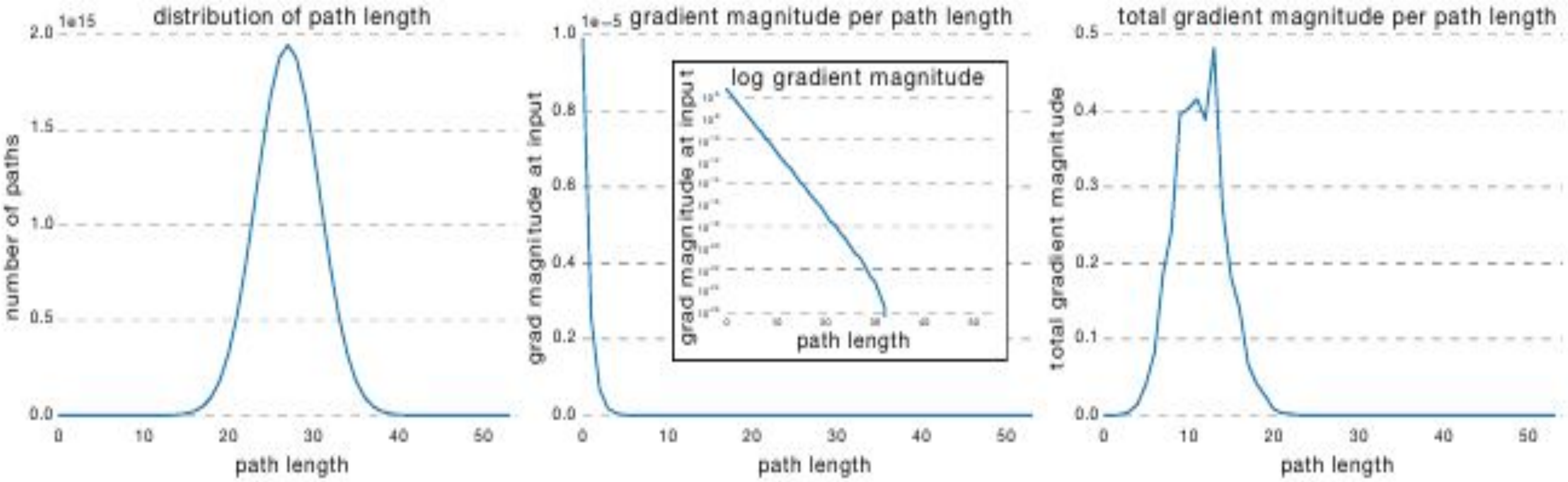
(a)



(b)

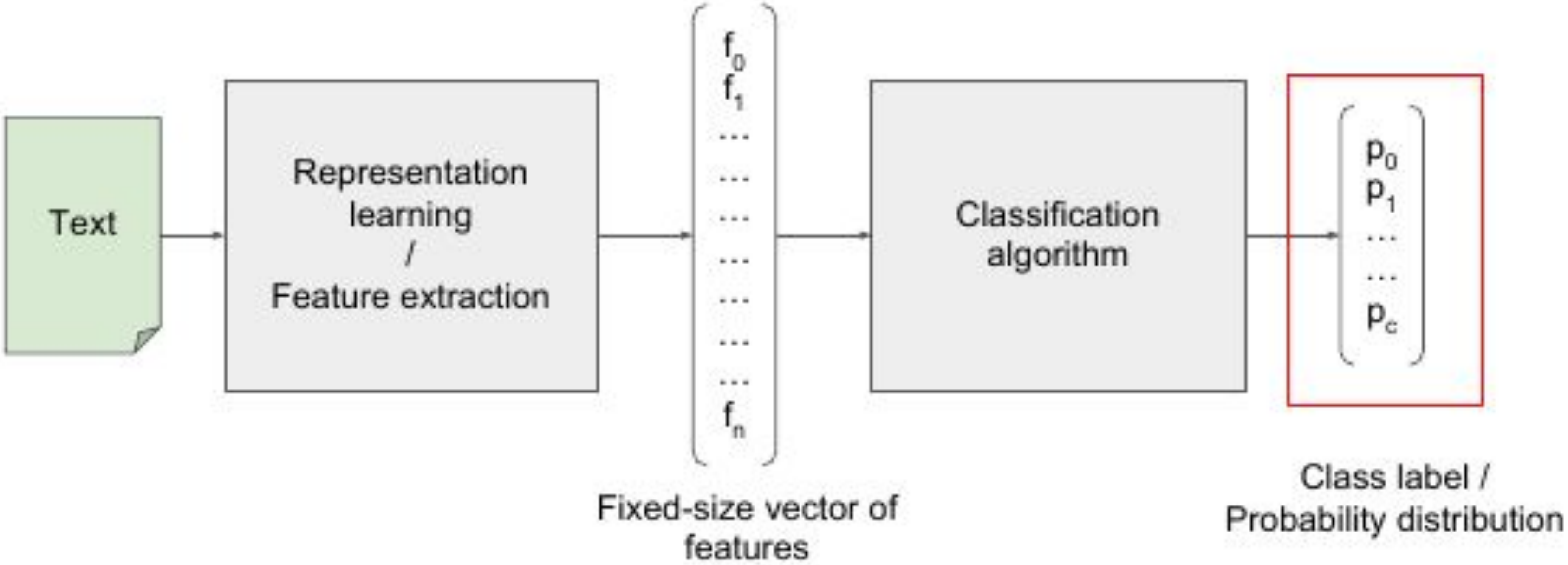
https://arxiv.org/pdf/1605.06431.pdf

“Residual Networks Behave Like Ensembles of Relatively Shallow Networks”



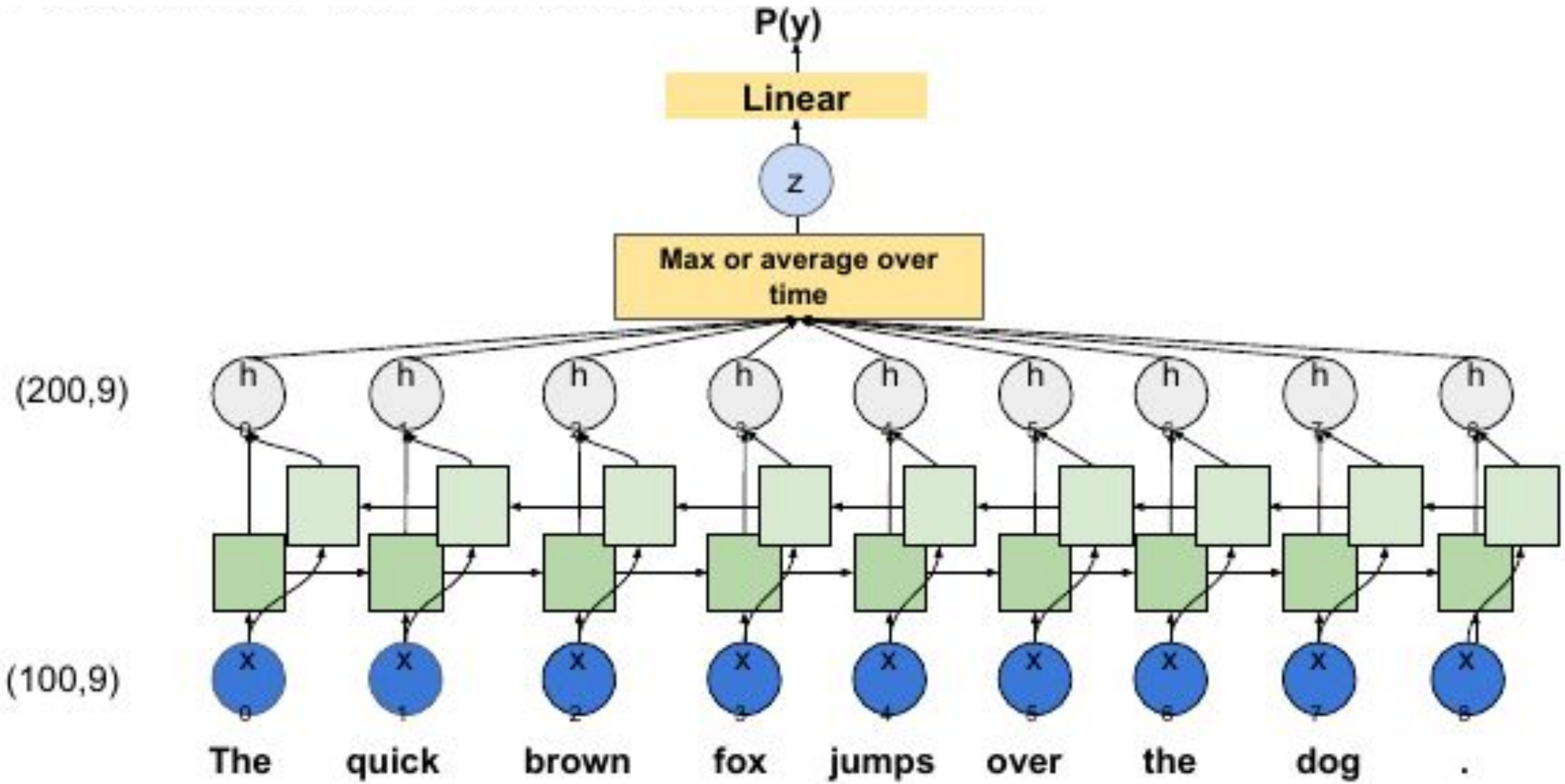
https://arxiv.org/pdf/1605.06431.pdf

Text classification



From YSDA nlp course 2018

Recurrent neural networks for texts

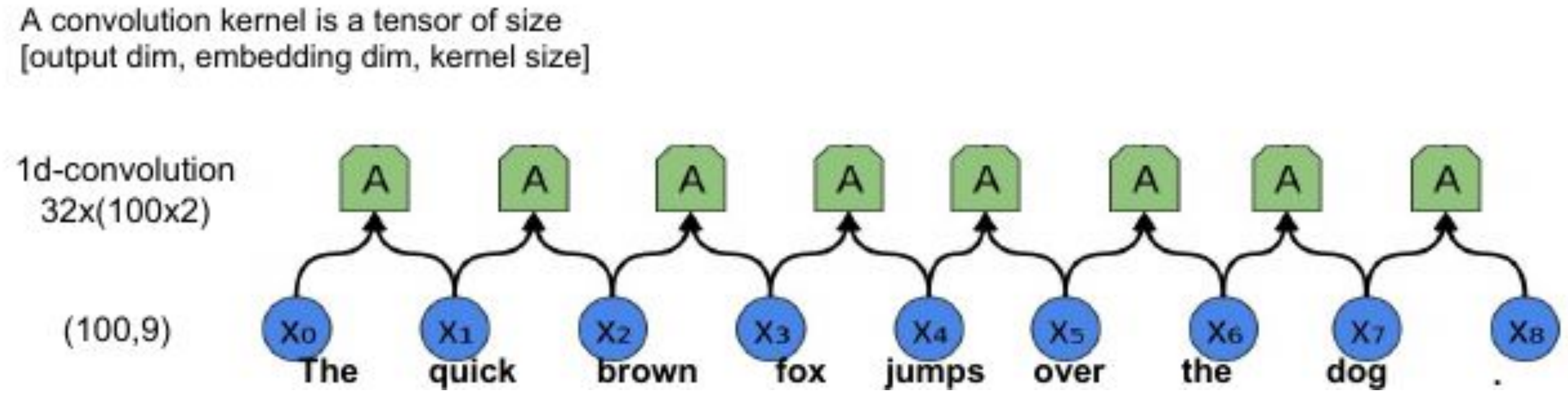


From YSDA nlp course 2018

Convolutional neural networks for texts

From YSDA nlp course 2018

CNN for texts

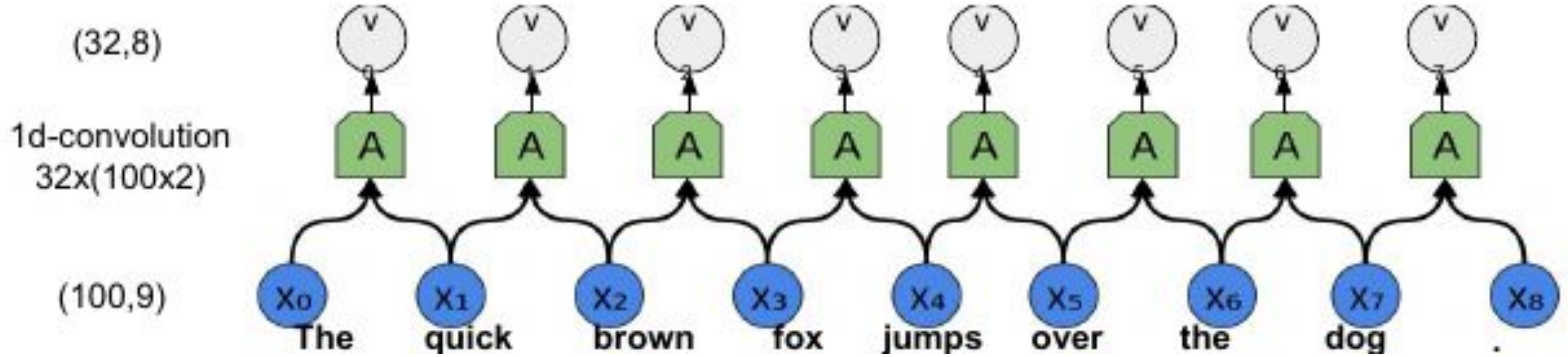


From YSDA nlp course 2018

CNN for texts

$$\mathbf{v}_0 = \mathbf{A}(\mathbf{x}_0, \mathbf{x}_1)$$

$$v_{0i} = A_i(\mathbf{x}_0, \mathbf{x}_1) = \sum_j (K_{0ij} x_{0j} + K_{1ij} x_{1j})$$

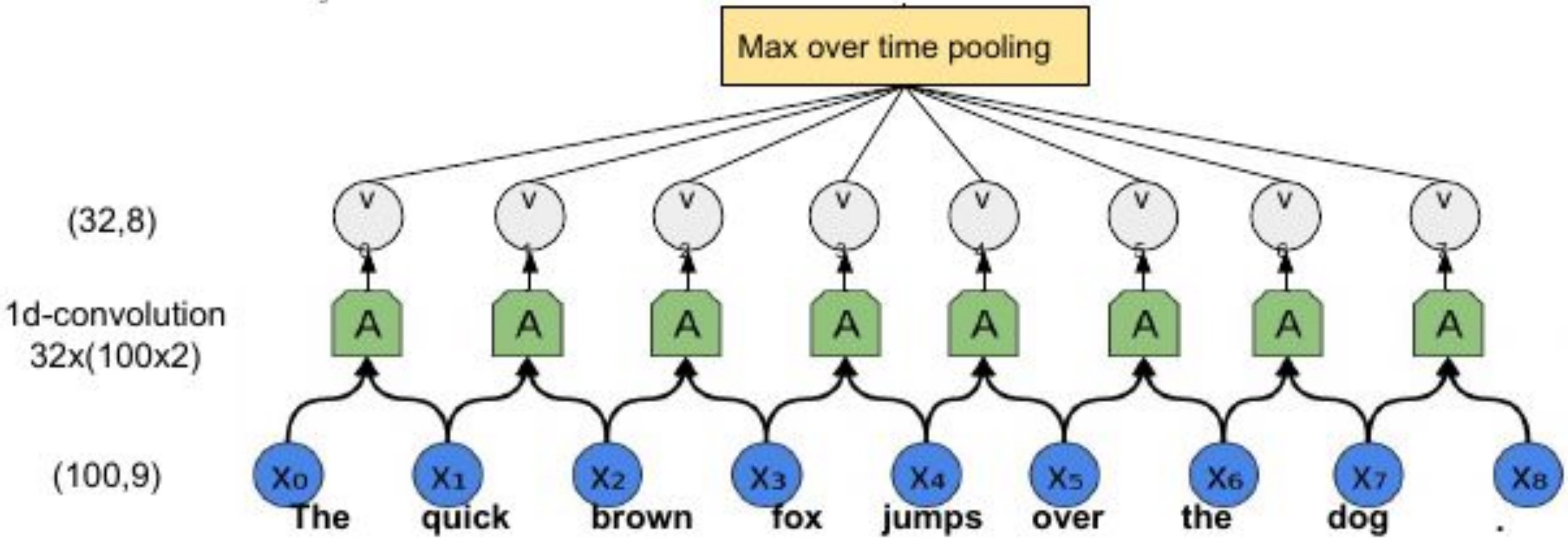


From YSDA nlp course 2018

CNN for texts

$$\mathbf{v}_0 = \mathbf{A}(\mathbf{x}_0, \mathbf{x}_1)$$

$$v_{0i} = A_i(\mathbf{x}_0, \mathbf{x}_1) = \sum_j (K_{0ij} x_{0j} + K_{1ij} x_{1j})$$

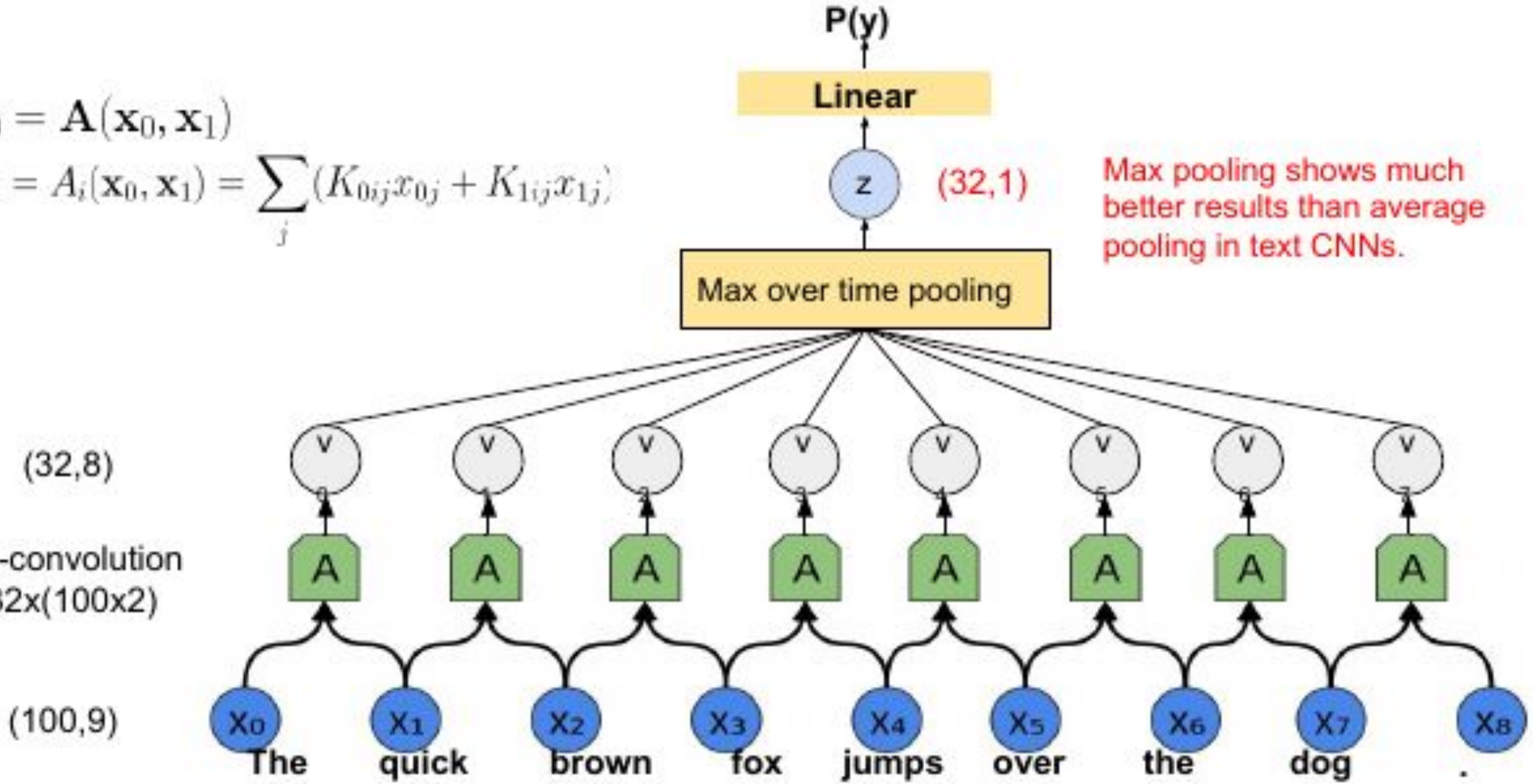


From YSDA nlp course 2018

CNN for texts

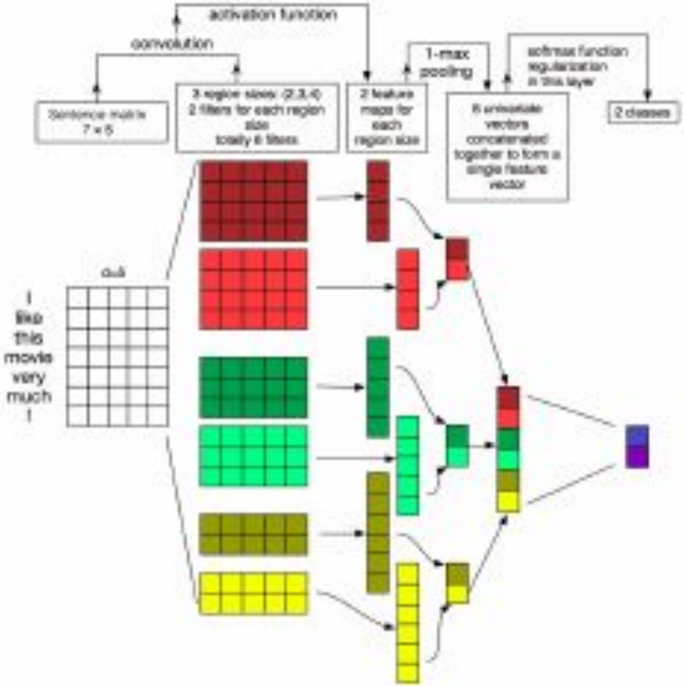
$$\mathbf{v}_0 = \mathbf{A}(\mathbf{x}_0, \mathbf{x}_1)$$

$$v_{0i} = A_i(\mathbf{x}_0, \mathbf{x}_1) = \sum_j (K_{0ij} x_{0j} + K_{1ij} x_{1j})$$



From YSDA nlp course 2018

CNN for texts: Improvements

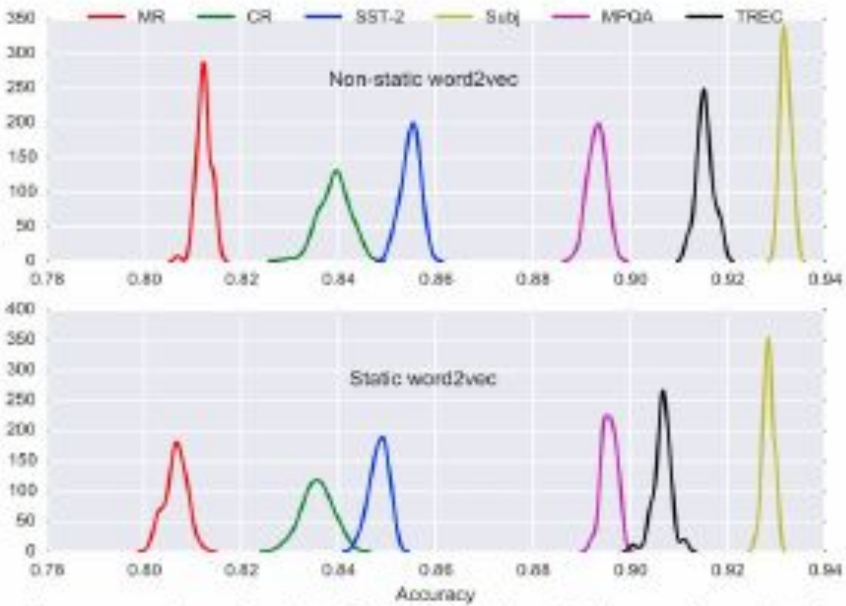


- Use convolutional layers with different kernel size, separate max-pooling over time and concatenation.
- K-max pooling: take not 1 but k highest activations in their original order.
  E.g. (0,1,3,2,0,1,4,1) → (3,2,4)

Zhang et al.
https://arxiv.org/abs/1510.03820

From YSDA nlp course 2018,

CNN for texts: Improvements



Accuracy density plots for non-static w2v (upper) and static w2v (lower) [for 10-fold CV over the 100 replications]

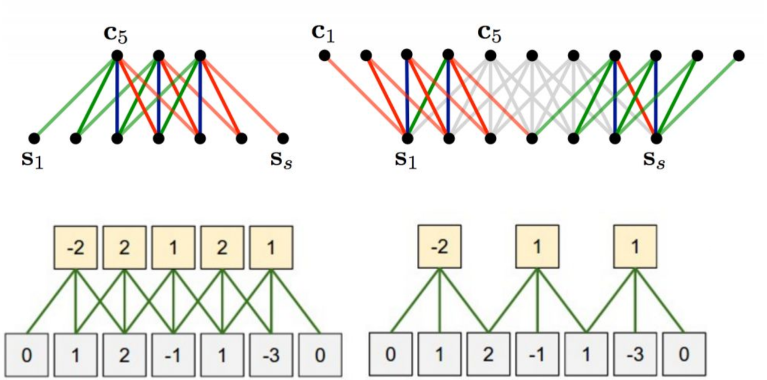
- Use convolutional layers with different kernel size, separate max-pooling over time and concatenation.
- K-max pooling: take not 1 but k highest activations in their original order.
  E.g. (0,1,3,2,0,1,4,1) → (3,2,4)
- Use pre-trained word vectors only for embedding layer initialization, train it jointly with model

Zhang et al.
https://arxiv.org/abs/1510.03820

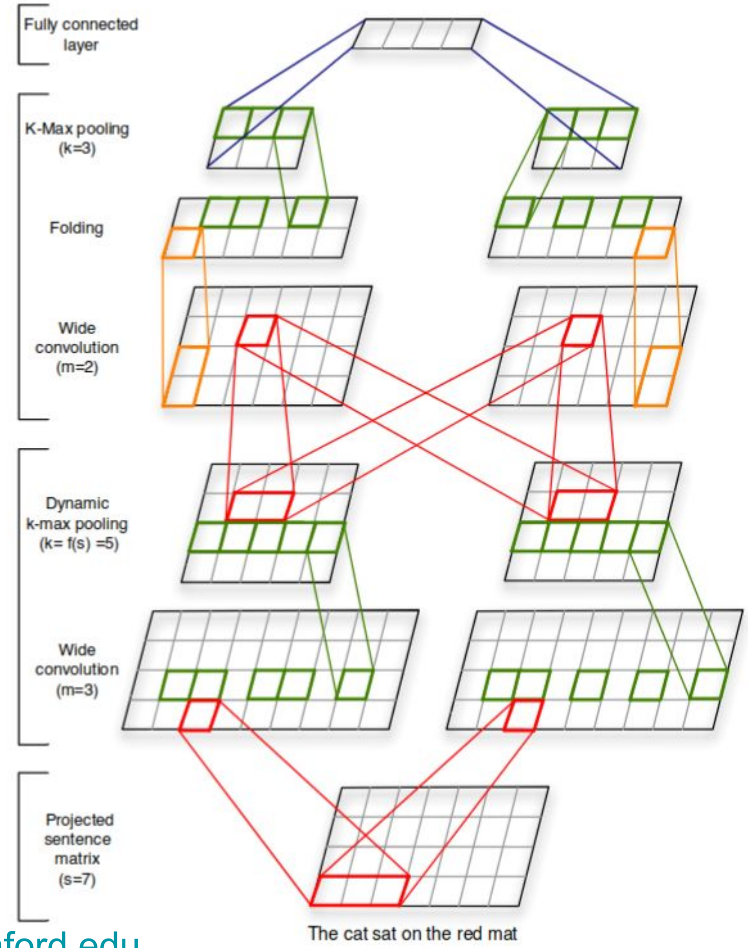
From YSDA nlp course 2018,

More about CNN

- Narrow vs wide convolution (stride and zero-padding)



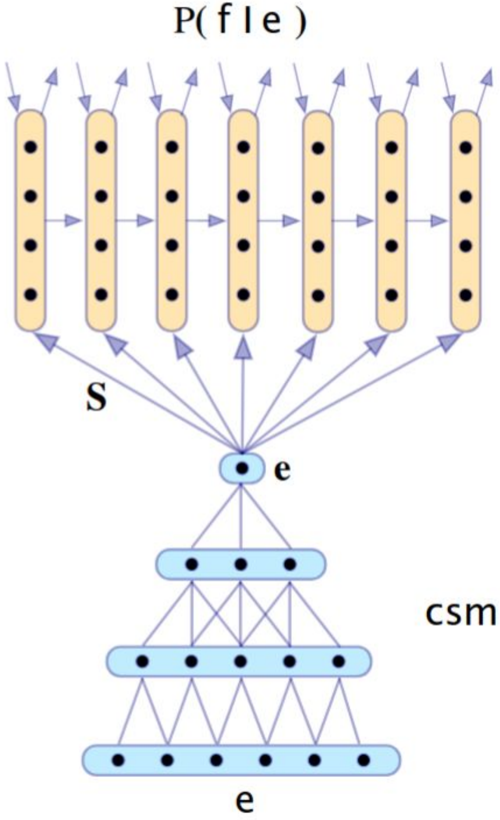
- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

CNN applications

- Neural machine translation: CNN as encoder, RNN as decoder
- Kalchbrenner and Blunsom (2013) “Recurrent Continuous Translation Models”
- One of the first neural machine translation efforts



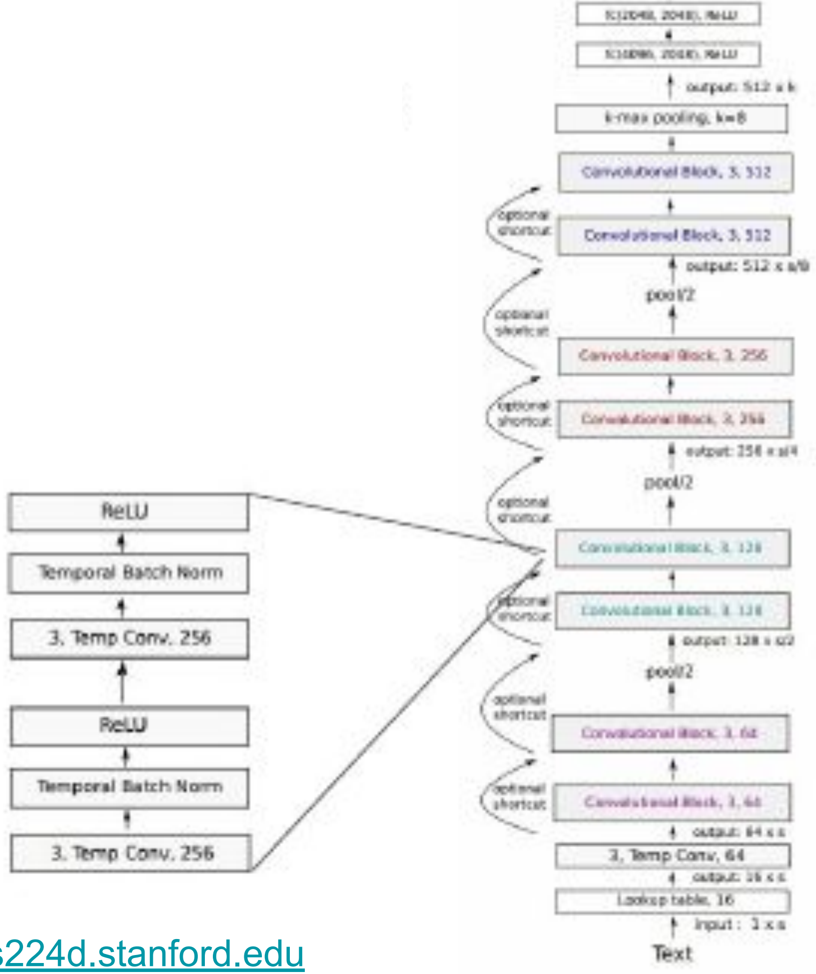
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

Deep Convolutional networks for texts

Q: Can we get some quality points just stacking much more layers?

A: It does make sense in case character-level convolutional architectures.

VDCNN [Conneau et al. 2015] ~ ResNet-like network with 29 conv. layers

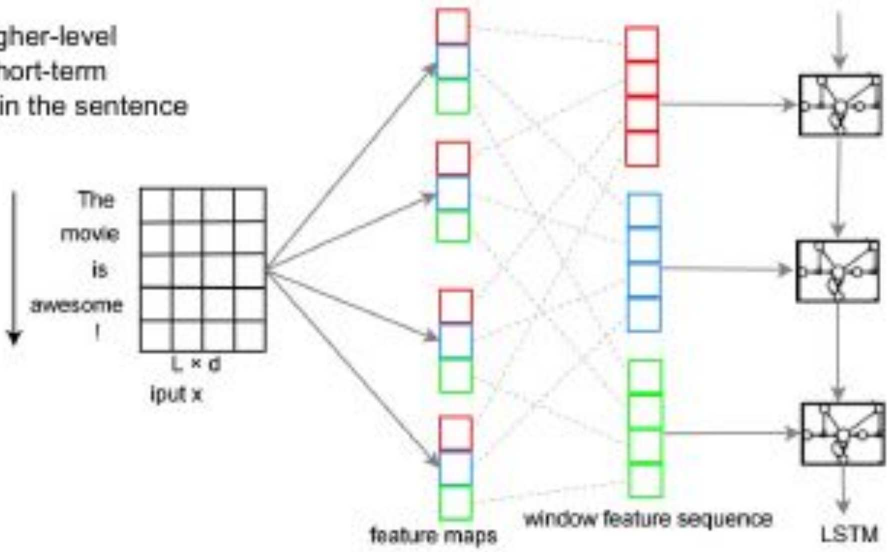


We can combine CNN and RNN together

C-LSTM [Zhou et al. 2015]

[conv.]->[LSTM]

C-LSTM utilizes CNN to extract a sequence of higher-level phrase representations, and are fed into a long short-term memory recurrent neural network (LSTM) to obtain the sentence representation.



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

Text augmentations

Like with images we can increase our training corpora size with augmentations.

Examples:

- Text deformations: mix some texts, change order…
- Reformulations
- Word dropout

Text segmentation

Open vocabulary problem: In NLP language vocabulary is usually very big. To produce a good quality with particular word the algorithm should see a lot of examples with it. This is a big problem for rare words.

There are two extreme approaches for vocabulary modelling:

- Char level: small vocabulary, a lot of examples with each element, slow training, long sequences during encoding and decoding
- Each word is a new item in vocabulary: Big vocabulary, small number of examples for rare words, fast training, short sequences during encoding and decoding

Text segmentation: Balance, BPE

We can balance vocabulary size with length of sequence

Bait Pair Encoding (BPE) (Sennrich et al.): Let’s split rare words into subwords, while leave frequent sequences as a one token.

1) Compute merge table: Starting from characters let’s one by one merge the most frequent symbols into one symbol until reaching desired vocabulary size.
2) During inference let’s greedily (priority=number of step, when this pair was added in (1)) apply merge rules

https://arxiv.org/pdf/1508.07909.pdf

BPE Benefits

- We have not got out-of-vocabulary words, because we start from all characters.
- We can balance vocabulary size with decoding efficiency

Example:

“mother” -> (BPE) mother

“sweetish” -> (BPE) sweet ish

“asft” -> (BPE) as f t

https://arxiv.org/pdf/1508.07909.pdf

Outro and Q & A

- Vanishing gradient is present not only in RNNs
  - Use some kind of memory or skip-connections
- LSTM and GRU are both great
  - GRU is quicker, LSTM catch more complex dependencies
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient
- Clip your gradients
- Combining RNN and CNN worlds? Why not ;)

That’s all. Feel free to ask any questions.