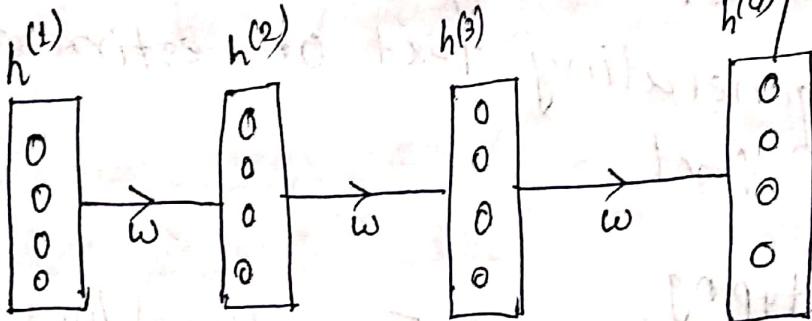


## Lecture 7: Vanishing Gradients, Fancy RNNs

### Vanishing gradient intuition:



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \underbrace{\frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}}}_{\text{what happens if these are small?}} \times \frac{\partial f^{(4)}}{\partial h^{(4)}}$$

what happens if these  
are small? Vanishing gradient  
problem

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

$$h^{(t)} = \delta(W_h h^{(t-1)} + W_x x^{(t)} + b_t)$$

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag}(\delta(W_h h^{(t-1)} + W_x x^{(t)} + b_t)) W_h$$

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \underset{j \neq i}{\sum} \frac{\partial h^{(i)}}{\partial h^{(j)}}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \underset{j \neq i}{\sum} w_n^{(i,j)} \underset{j \neq i}{\sum} \text{diag} \left( \delta' \left( w_n h^{(i-1)} + w_x x^{(i)} + b_i \right) \right)$$

If  $w_n$  is small, then this term get & vanishingly small as  $i$  and  $j$  get further apart.

Vanishing gradient proof sketch: (cont) Considering matrix L2 norm

$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\| \left\| w_n \right\| \underset{j \neq i}{\sum} \left\| \text{diag} \left( \delta' \left( w_n h^{(i-1)} + w_x x^{(i)} + b_i \right) \right) \right\|$$

→ Pascanu et al showed that if the largest eigenvalue of  $w_n$  is less than 1, then the gradient

$\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\|$  will shrink exponentially.

→ Here the bound is 1 because we have sigmoid non-linearity

→ Vanishing gradient is problematic because we will lose the long-term effect for this

Why is vanishing gradient a problem?

### Another explanation

Gradient can be viewed as a measure of the effects of the past on the future.

- ⇒ If the gradient becomes vanishingly small over longer distances (step  $t$  to step  $t+n$ ), then we can't tell whether:
- i) There is no dependency between step  $t$  and step  $t+n$  in the data.
  - ii) We have wrong parameters to capture the true dependency between  $t$  and  $t+n$ .

Due to vanishing gradient RNN-LMs are better at learning from sequential recency than syntactical recency. so they make

Why is exploding gradient a problem?

→ If the gradient becomes too big, then the SGD update step becomes too big

$$\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla J(\theta)$$

Gradient clipping: if the norm of the gradient

is greater than some threshold, scale it down before applying SGD update

$$\hat{g} \leftarrow \frac{\partial E}{\partial \theta}$$

if  $\|\hat{g}\| >$  threshold then

$$\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$$

endif

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

How to fix vanishing gradient problem?

LSTM

## Long Short Term Memory (LSTM)

Hochreiter and Schmidhuber 1997

- proposed as a solution to vanishing gradient problems
- On step  $t$ , there is a hidden state  $h^{(t)}$  and a cell state  $c^{(t)}$ 
  - Both are vectors length  $n$
  - The cell stores long-term information
  - LSTM can erase, write and read information from the cell

The selection of which information is erased and written/read is controlled by three corresponding gates:

- The gates are also vectors length  $n$
- On each timestep, each elements of the gate can be open (1), closed (0) or somewhere between
- The gates are dynamic: their value is computed based on the current context

Forget gate: controls what is kept vs forgotten.

from previous cell state

$$f^{(t)} = \sigma(w_f h^{(t-1)} + U_f x^{(t)} + b_f)$$

$$i^{(t)} = \sigma(w_i h^{(t-1)} + U_i x^{(t)} + b_i)$$

$$o^{(t)} = \sigma(w_o h^{(t-1)} + U_o x^{(t)} + b_o)$$

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

Output Gate: controls what parts of cells are output to hidden state

④ New cell content: the new content to be written to the cell

$$\tilde{c}^{(t)} = \tanh(w_c h^{(t-1)} + U_c x^{(t)} + b_c)$$

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

Cell state

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

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

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

→ LSTM architecture makes it easier for the RNN to preserve information over many timesteps

e.g. if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely

By contrast, it's harder for vanilla RNN to

learn a recurrent weight matrix  $W_R$  that preserves info in hidden states

→ However LSTM doesn't provide the guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.

→ In 2013-2015, LSTMs started achieving SOTA

.. → successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning.

→ LSTM became the dominant approach

Now other approaches like Transformers have become more dominant.

## Gated Recurrent Units (GRU)

→ Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM

→ On each time step  $t$  we have input  $x^{(t)}$  and hidden state  $h^{(t)}$  (no cell state)

$$\rightarrow u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u) \rightarrow \text{Update gate: controls what parts of hidden state are updated vs preserved}$$

$$\rightarrow r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r) \rightarrow \text{Reset gate: controls what parts of previous hidden state are used to compute new content}$$

$$\begin{cases} \tilde{h}^{(t)} = \tanh(W_h(r^{(t)}) \circ h^{(t-1)} + U_h x^{(t)} + b_h) \\ h^{(t)} = \tanh(1 - u^{(t)}) \circ h^{(t-1)} + U^{(t)} \circ \tilde{h}^{(t)} \end{cases} \rightarrow \begin{array}{l} \text{Hidden state:} \\ (\text{update gate}) \\ \text{simultaneously} \\ \text{controls what is} \\ \text{kept from previous} \\ \text{hidden state, and} \end{array}$$

→ New hidden state content: reset gate selects useful parts of previous hidden state, uses that and current input to compute new hidden state what is updated to new hidden state content

Like LSTM, GRU makes it easier to retain info long term (by setting update to 0)

### LSTM vs GRU

- GRU is quicker to compute and has fewer parameters.
- There is no conclusive evidence that one performs better than other.
- LSTM is a good default choice in practice.

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

\* Vanishing/exploding gradient is not just an RNN problem.

• Lots of new deep feedforward/convolutional architectures that add more direct connections.

For example:

Residual connections (skip connections)

- Dense connections aka DenseNet

### Highway connections:

- Similar to ~~not~~ residual connections but the identity connections vs the transformation layer is controlled by a dynamic gate.

Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix. [Bengio, 1994]

### Bidirectional RNNs

- forward RNN → provides info. of left context
- backward RNN → provides info. of right context
- are only applicable if you have access to the entire sequence

↳ Not applicable to Language modeling

BERT (Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on ~~is~~ bidirectionally.

## Multi-layer RNNs (stacked RNNs)

- This allows the network to compute more complex representations
- The lower RNNs should compute lower-level features and higher RNNs should compute higher-level features

### In practice

- High performing RNNs are often multi-layer
- In a 2017 paper Brattet al. find that for NMT, 2 to 4 layers are best for the encoder RNN and 4 layers are best for decoder RNN
- Transformer-based network are frequently deeped like 12 to 24 layers. Transformers have a lot of skipping-like connections

## Lecture 8: Translation, Seq2Seq, Attention

### Section 1: Pre-Neural Machine Translation

Machine Translation is the task of translating a sentence  $x$  from one language (the source language) to a sentence  $y$  in another language (the target language)

Research began in the early 1950s

Russian  $\rightarrow$  English (motivated by the Cold War)

- Those systems were mostly rule-based, using bilingual dictionary to map Russian word to their English counterparts.

### 1990s-2010s: Statistical Machine Translation

Core idea: Learn a probabilistic model from data

$\rightarrow$  We want to find best English sentence  $y$ , given French sentence  $x$

$$\operatorname{argmax}_y P(y|x)$$

$\rightarrow$  Using Bayes rule to break this down to two components:  $\operatorname{argmax}_y P(x|y) P(y)$

$$\text{argmax}_y P(x|y) \cdot P(y)$$

Translation Model

Language model

Models how words and phrases should be translated (fidelity). Learnt from parallel data.

Models how to write good English (fluency) Learnt from monolingual data

(How to learn translation model  $P(x|y)$ )?

→ First, we need large amount of parallel data

Break that down further,

$$P(x; a|y)$$

& where  $a$  is the alignment, i.e. word level correspondence

correspondence between French and English sentence

What is alignment?

Alignment is the correspondence between particular words in the translated sentence pair.

Alignment can be complex

one to one

many to one

one to many (fertile)

many to many

## Learning alignment for SMT

We learn  $P(x, a|y)$  as a combination of many factors, including:

→ Probability of particular words aligning

→ Probability of particular words having particular fertility

• Alignments  $a$  are latent variables: They aren't explicitly specified in the data.

→ Require the use of special learning algos

like Expectation Maximization for learning

the params of distributions with latent variables

## Decoding for SMT

→ We could enumerate every possible  $y$  and calculate the probability  $\Rightarrow$  Too expensive

Answer: Impose strong independence assumptions  
in model use dynamic programming for globally  
optimization solutions (Viterbi Algorithm)

# Viterbi: Decoding with Dynamic Programming

$$P(x, a|y) = \prod_{j=1}^L P(x_j|f_a(j))$$

SMT was a huge field upto 2010s

The best systems were extremely complex

→ systems had many separately-designed  
sub components

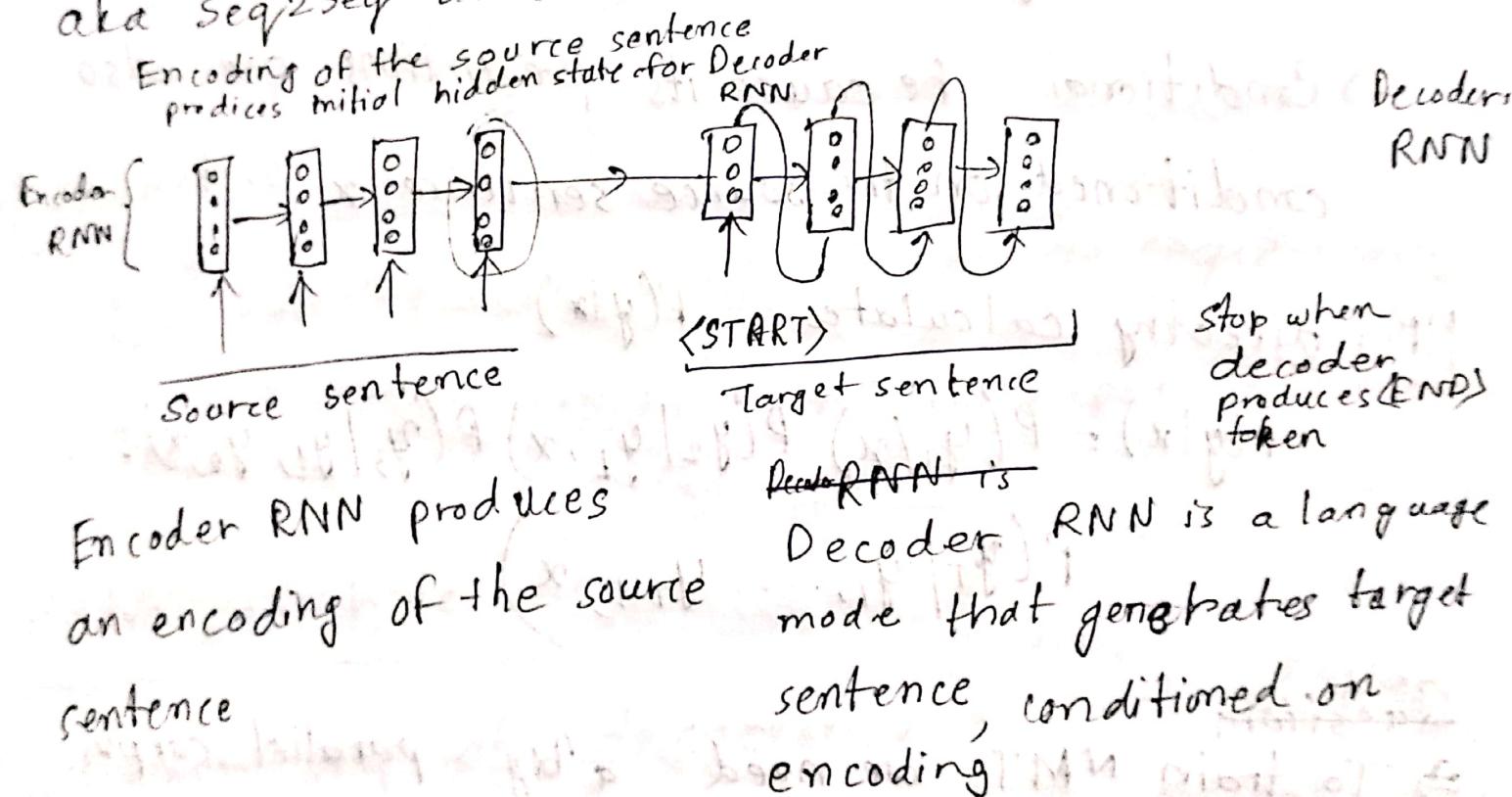
→ lots of feature engineering

→ Requires compiling and maintaining extra  
resources

→ Lots of human effort to maintain

## Section 9: Neural Machine Translation

- Neural Machine Translation (NMT) is a way to do machine translation with a single neural network.
- The neural architecture is called sequence-to-sequence aka seq2seq and it involves two RNNs



- \* This architecture is versatile
  - Summarization (long text  $\rightarrow$  short text)
  - Dialogue (previous utterances  $\rightarrow$  next utterance)
  - Parsing (input text  $\rightarrow$  output parse as sequence)
  - Code generation (natural language  $\rightarrow$  Python code)

The sequence-to-sequence is an example of a Conditional Language Model

→ Language model because the decoder is predicting the next word of the target sentence  $y$ .

→ Conditional because its predictions are also conditioned on the source sentence  $x$ .

NMT directly calculates  $P(y|x)$

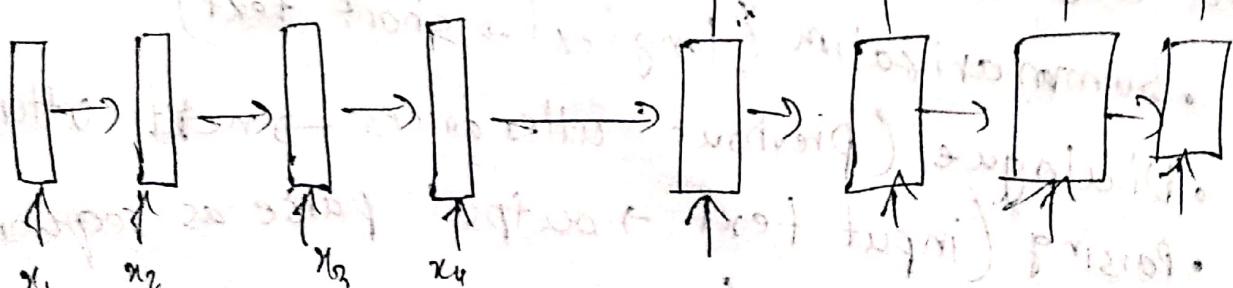
$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots$$

$$P(y_T|y_1, \dots, y_{T-1}, x)$$

Question:

⇒ To train NMT we need a big parallel corpus.

sum of  
Negative log prob  $J = \frac{1}{T} \sum_{t=1}^T J_t$



- Seq2Seq is optimized as a single system. Backprop operates "end-to-end"

There are many ways of decoding

- \* Greedy decoding (take most probable word on each step)

Cons → no way to undo it

Exhaustive search decoding:

We can try computing all possible sequences of length  $T$ .

This  $O(V^T)$  complexity is far too expensive.

Beam search decoding:

On each step of decoder, keep track of the  $k$ -most probable partial transformations (hypotheses).

•  $k$  is the beam size (5-10)

- A hypothesis  $y_1 \dots y_t$  has a score which is its log probability

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x)$$

$$= \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

- Scores are all negative and higher score is better
- We search for high-scoring hypothesis, ~~the~~ tracking top k on each step
  - Beam search is not guaranteed to find optimal solution. But much more efficient than exhaustive search

### Stopping criterion

- In greedy decoding, usually we decode until the model produces a `<END>` token
- In beam search decoding, different hypothesis may produce `<END>` tokens in different time step.
  - \* When a hypothesis produces `<END>`, that hypothesis is complete

\* Place it aside and continue exploring other hypothesis via beam search

→ Usually we continue beam search until:

- we reach timestep  $T$

- We have at least  $n$  completed hypotheses

[ $T$  and  $n$  are predefined cutoffs]

Finishing up:

We have our list of completed hypotheses

Each hypothesis  $y_1 \dots y_T$  on our list has a score

$$\text{score}(y_1, \dots, y_T) = \log \text{PLM}(y_1, \dots, y_T | x)$$

$$= \sum_{i=1}^T \log \text{PLM}(y_i | y_1, \dots, y_{i-1}, x)$$

Problem  
Cons → longer hypotheses have lower scores

Fix: Normalize by length, use this to select top one instead

$$\frac{1}{T} \sum_{i=1}^T \log \text{PLM}(y_i | y_1, \dots, y_{i-1}, x)$$

## Advantages of NMT

Compared to SMT

→ Better performance

- Better & More fluent
- Better use of context
- Better use of phrase similarities

→ A single neural network to be optimized end-to-end

- No subcomponents to be individually optimized

→ Requires much less human engineering effort

- No feature engineering
- Same method for all language pairs

## Disadvantages

- less interpretable
- difficult to control
  - can't easily specify rules or guidelines for translation
  - safety concerns

## Evaluation

### BLEU (Bilingual Evaluation Understudy)

- compares the machine-generated translation to one or several human-written translation and compute a similarity score based on n-gram precision [usually for 1, 2, 3, 4-grams]
- plus a penalty for too-short system translation

\* BLEU is useful but imperfect

. There are many valid ways to translate a sentence

• So a good translation can get a poor BLEU score because it has low n-gram overlap with the human translation

NMT went from a fringe research activity in 2014 to the leading standard in 2016

- 2014: first seq2seq paper published
- 2016: Google Translate switches from SMT to NMT

→ Machine Translation is not yet solved.

### Difficulties

- Out-of-vocabulary words
- Domain mismatch between train and test data
- Maintaining context over longer text
- Low resource language pairs
- Using common sense is still hard
- Idioms are difficult to translate
- picks up biases in training data
- uninterpretable systems do strange things.

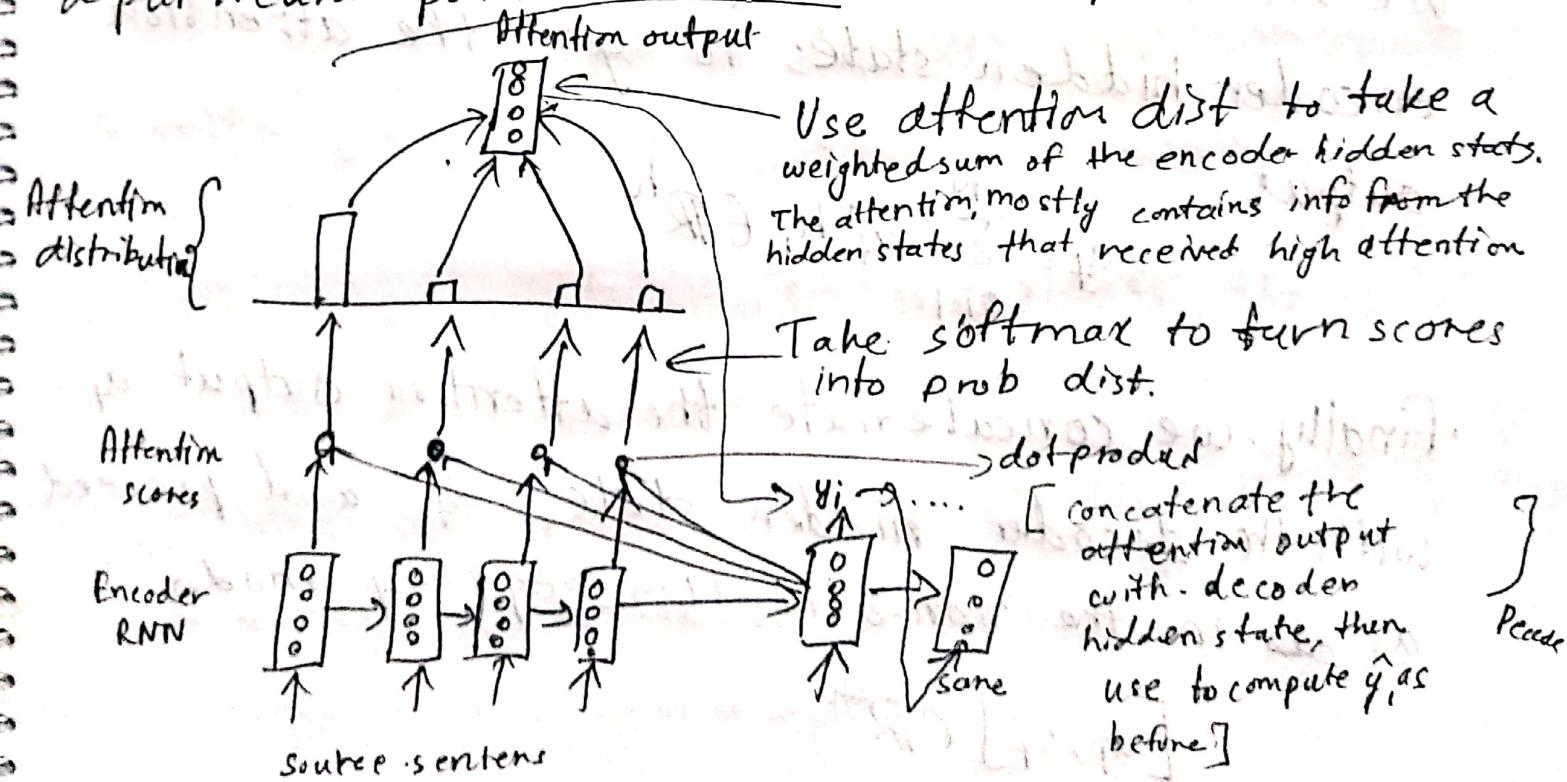
### Section 3: Attention

Sequence-to-sequence: the bottleneck

Encoding of the source sentence needs to capture all infos about the source sentence info bottleneck

Attention provides solution to the bottleneck problem

→ Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence



## In equations

- We have encoder hidden states  $h_1, \dots, h_N \in \mathbb{R}^h$
- On timestep  $t$ , we have decoder hidden state  $s_t$
- We get the attention scores  $e^t$  for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution  $a^t$  for this step (this is a prob dist and sum to 1)

$$a^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use  $a^t$  to take weighted sum of the encoder hidden states to get the attention output

$$a_t = \sum_{i=1}^N a_i^t h_i \in \mathbb{R}^h$$

- Finally, we concatenate the attention output  $a_t$  with the decoder hidden state  $s_t$  and proceed as ~~do~~ in the non-attention seq2seq model

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

Attention "is great"

. Attention significantly improves NMT performance

-> it's very useful to allow to focus, on certain parts of the source

. Attention solves bottleneck problem

-> allows decoder look directly at source;  
bypass bottleneck

. Attention helps with vanishing gradient problem

-> provides shortcut to faraway states

. Attention provides some interpretability

-> By inspecting attention distribution, we can see what the decoder was focusing on.

-> We get (soft) alignment for free

-> This is cool because we never explicitly trained an alignment system

-> The network just learned alignment by itself

Attention is general deep learning technique

We can use attention in many architectures and many tasks.

### Attentions

Given a set of vector values and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query.

→ query attend to the values

→ For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)

### Intuition:

. The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.

- Attention is a way to obtain a fixed size representation of an arbitrary set of representations (the values), dependent on some other representations (the query)

There are several attention variants.

- We have some values  $h_1, \dots, h_N \in \mathbb{R}^{d_L}$  and a  $q \in \mathbb{R}^{d_Q}$
- Attention always involves
  - 1) Computing the attention scores  $e \in \mathbb{R}^N$
  - 2) Taking softmax to get attention distribution  $\alpha = \text{softmax}(e) \in \mathbb{R}^N$
  - 3) Using attention distribution to take weighted sum of values
$$\alpha = \sum_{i=1}^N \alpha_i h_i \in \mathbb{R}^{d_L}$$

thus obtaining the attention output a (sometimes called context vector)

## Attention variants

### • Basic dot product

$$e_i = s^T h_i / \lVert R \rVert \quad (\text{from adj})$$

### • Multiplicative attention

$$e_i = s^T W h_i / \lVert R \rVert$$

where  $W \in \mathbb{R}^{d_2 \times d_2}$  is a weight matrix.

### • Additive attention

$$e_i = v^T \tanh(W_1 h_i + W_2 s) / \lVert R \rVert$$

→ where  $W_1 \in \mathbb{R}^{d_3 \times d_2}$ ,  $W_2 \in \mathbb{R}^{d_3 \times d_2}$  are weight matrices  
and  $v \in \mathbb{R}^{d_3}$  is a weight vector.

→  $d_3$  (the attention dimensionality) is a hyperparameter.

→ We then softmax to get probability weights.

(other methods follow similarly.)

## Lecture 9: Practical Tips for Project

Two basic starting points, for all of science:

[Nails] Start with a (domain) problem of interest and try to find better ways to address it than are currently known/use.

[Hammer] Start with a technical approach of interest, and work out good ways to extend and improve it or new ways to apply it.

Herb Simon: "If you see a research area where many people are working, go somewhere else"

Wayne Gretzky: "I skate to where the puck is going, not where it has been"

## Autoregressive language model (AR)

Kind of model that uses the context

word to predict the next word. But

here the context word is constrained to two directions either forward or backward

## Autoencoder language model

Aims to reconstruct the original data from corrupted input. Context of both sides

### Disadvantages

- It uses [MASK] in the pretraining, but this

### In XL-Net

Language model consists of two phases, the pre-train phase and fine-tune phase. XL-Net focus on pre-train phase. In the pretrain phase,

it proposed a new objective called Permutation Language Modeling

Intuitively, what happens with RNNs?

1) Measure the influence of the past on the future

$$\frac{\partial \log p(x_{1:n} | x_{1:n})}{\partial h_t} =$$

5) The large output vocabulary problem in NMT (on all N Log)

• Word generation problem

• Vocab used are usually modest: 50K

Possible approach for output

• Hierarchical softmax  $\rightarrow$  tree structure vocab

• Noise constrained estimation  $\rightarrow$  binary classification

$\rightarrow$  Train on a subset of the vocab at a time.

test on a smart on the set of possible

$\rightarrow$  Use attention to work out what you are translating

$\rightarrow$  we can do something simple like dictionary lookup

BLEU formula (counts n-grams up to length 4)

$$\exp(0.3 \times \log p_1 + 0.25 \log p_2 + 0.125 \log p_3 + 0.125 \log p_4 - \max(\text{words in reference}/\text{words in machine} - 1.0))$$

P = precision

BLEU is a weighted geometric mean, with a brevity penalty factor added.

- Initial results showed that BLEU predicts human judgement well.

Automatic evaluation of MT  
People started optimizing their systems to maximize BLEU score

- BLEU scores improved rapidly
- The correlation between BLEU and human judgements of quality went way down.
- MT BLEU scores now approach those of human translations but their true quality remains far below human translations

MT

Coming up with automatic evaluations

has become its own research field.

There are many proposals : TER, METEOR,

Maxsim, SEPIA, RTE-MT

TERPA is a representative good one that handles some word choices

\* MT research requires some automatic metric to allow a rapid development and evaluation cycle.

### Steps of Doing Project

1) Define Task

2) Define Dataset

- Search for academic datasets

- Define your own data (harder need new baselines)

- Build baseline system

### 3) Dataset hygiene

- Right at the beginning separate of dev-test and test splits

### 4) Define metric

for  
Summarization [ROUGE - Recall Oriented  
Understudy for Generating  
Evaluation]

### 5) Establish baseline

- Implement simplest model & well-known metrics
- Compute metrics on train and dev
- Analyze errors
- If metrics are amazing and no error → Done! Problem was too easy.

### 6) Implement existing neural network

- Compute metric on train and dev
- Analyze outputs and errors
- Minimum bar for this class

F) Always be close to your data! (except for  
the final test set) in building your

- Visualize the dataset
- Collect summary stats about the data
- Look at errors
- Analyze how different params affect performance

g) Try out different models and model

variants

Aim to iterate quickly via having a  
good experimental setup

- fixed window neural net
- Recurrent neural network
- Recursive neural network
- Convolutional neural network
- Attention-based model

## Pots of data

- Many publicly available datasets are released with a train/dev/test structure, We're all on the honor system to do test runs only when development is done.
- Splits like this presuppose a fairly large dataset. If there is no dev set or you want a separate tune set, then you create one by splitting training data though you have to weight its size/usefulness against the reduction in train-set size.
- Having a fixed dev/test set ensures that all systems are assessed against the same gold standard.
- Generally good, but it is problematic where the test set turns out to have unusual properties that distort progress on the task.

## Training models and sets of data

- When training, model overfits to what

you are training ~~on~~ of new data.

→ The model correctly describes what happened to occur in particular data

we trained on, but the patterns are not general enough to be likely to

apply to new data.

- The way to monitor and avoid overfitting is using independent validation and test set.

Only at the end, you evaluated and present

final numbers on a test set ideally

only once, don't do it to early to no good

validation by the process.

## Getting your NN to train

- Positive attitude ~~labor, patient, ready~~  
• NN's want to learn  
→ If the net isn't learning, we're doing something to prevent it from learning successfully
- Realize the grim reality
  - There are lots of things that can cause neural nets to not learn ~~well~~ at all or to not learn very well
    - Debugging might take more time than just implementing
    - Hard to work out but experience, experimental care and rules of thumb help.
    - Learning rate and initialization

## Experimental Strategy:

- Work incrementally
- Start with a very simple model and get it to work
- Add bells and whistles one-by-one and get the model working with each of them (or abandon them)
- Initially run on a tiny amount of data
  - bags can be noticed easily
  - 8 examples is good
  - Make sure you can get 100% on this
- Otherwise your model is definitely either not powerful enough or it is broken.

- Run your model on a large dataset
  - It should still close to 100% on the training data after optimization.
- Otherwise, you probably want to consider a more powerful model
  - Overfitting to train data is not something to be scared of when doing deep learning
  - These models are usually good at generalizing because of the way distributed representations share statistical strength regardless of overfitting to train data.
- We still need good generalization performance
  - Regularize your model until it doesn't overfit on dev data
    - L2 regularization
    - Generous dropout.

## Some General Text Preprocessing Steps (Not all should be done in all cases)

- 1) Remove HTML Tags
- 2) Remove extra whitespace
- 3) Convert accented characters to ASCII  
special characters
- 4) Expand contradictions
- 5) Remove special characters
- 6) Lowercase all text
- 7) Convert numbers words to numeric form
- 8) Remove numbers
- 9) Remove stop words
- 10) Lemmatization

## Lecture 10: Question answering

Project writeup

Abstract/Introduction

Prior related work

Model Model

Data  $\oplus$

Experiments

Results

Analysis & Conclusion

Interesting thing

Google's Knowledge Graph

## Question Answering:

- 1) Finding documents that (might) contain an answer [traditional info retrieval techniques are used]
- 2) finding an answer in a paragraph or a document [Reading comprehension problem]

## A Brief History of Reading Comprehension

- i) Schank, Abelson, Lehner, et. al. 1977 - "Yale AI project"
- ii) Revised by Lynette Hershman in 1999
- iii) Revised again by Chris Burges in 2013 with MC Test [Machine Comprehension]
- iv) Floodgates opened in 2015/16 with the production of large datasets which permit supervised neural systems to be built
  - Hermann Rajpukar et.al. (EMNLP 2016) SQuAD
  - Hermann et. al. (NIPS 2016) Deepmind EN/DM dataset

## A Brief History of Open-domain Question Answering

- Simon's et. al. (1964) did first exploration on answering from an expository text based on matching & dependency parses
- Maron (1993) aimed to answer questions over an online encyclopedia using IR and shallow linguistic processing

The NIST TREC QA track began in 1999 by first rigorously investigating answering fact questions over a large collection of documents.

- IBM's Jeopardy! System (DeepQA, 2011) brought attention to a version of the problem; it used an ensemble of many methods.
- DrQA (Chen et.al. 2016) uses IR followed by neural reading comprehension to bring learning to open-domain QA

Turn-of-the Millennium full NLP QA: [Factoid QA]

Complex systems but they did work fairly well on "factoid" questions. [2003]

Factoid QA → answer is an entity

Blindsight

## Evaluation

### Exact Match

23% (avg)

F1

Knowledge base for factoid questions

### SQuAD 2.0

in the paragraph

- All questions have an answer
- Systems (implicitly) rank candidates and choose the best one
- In SQuAD 2.0, 1/3 of the training questions have no answers and about 1/2 of the dev/test ques have no answers

Simplest system approach to SQuAD 2.0

probabilistic approach to begin with a span of answers

a question

### NLI (Natural Language Inference)

basic NLU errors

Good systems are great, but still

### SQuAD Limitations

- only span-based answers (no, yes/no, counting, implicit, why)

- Questions are constructed looking at the passages

→ Not genuine info needs

→ Generally greater lexical and syntactical

matching between questions and answers

spans than you get in real life

- Barely any multi-fact / sentence inference

beyond coreference.

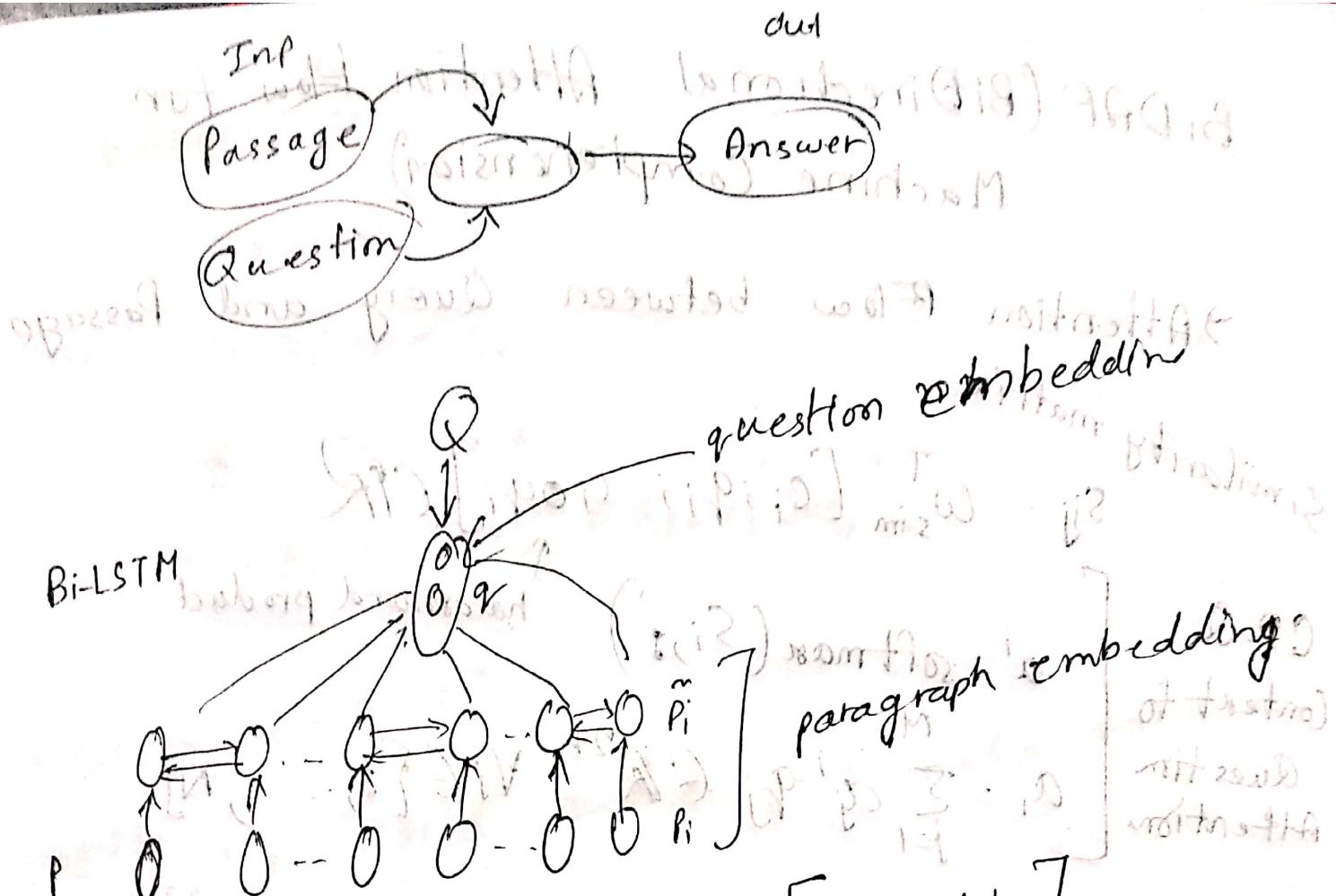
→ Nevertheless, it is well-targeted, well-structured and clean dataset

most used and competed on QA datasets

useful starting point for building systems in industry

### Stanford Attentive Reader [DrQA]

- Demonstrated a minimal, highly successful architecture for reading comprehension and question answering



$$(\text{start}) \quad \alpha_i = \text{softmax}(q^T w_s p_i) \quad \left. \begin{array}{l} \\ \text{Attention} \end{array} \right\}$$

$$(\text{end}) \quad d_i^r = \text{soft man}(q_i^T c_{\text{eff}}' P_i)$$

Stanford attentive reader  $\exp(w_{qj})$

$$q_j = \sum_{j=1}^n b_j q_j \quad \omega_j; \quad b_j \in \overline{\exp(\omega, q_{ij})}$$

[takes all states]  $\mapsto [ ] = id$

More (deep and complicated) embeddings.  
~~(not 2 = 1)~~ (Miner)

# BiDAF (BiDirectional Attention Flow for Machine Comprehension)

→ Attention flow between Query and Passage

Similarity matrix

$$S_{ij} = w_{\text{sim}}^T [c_i; q_i; c_i \otimes q_i] \in \mathbb{R}$$

hadamard product

C2Q  
Context to Question Attention

$$\begin{aligned} a_i &= \text{softmax}(S_{i,:}) \\ a_i &= \sum_{j=1}^N a_j^i q_j \end{aligned}$$

Q2Q

$$m_i = \max_j S_{ij}$$

$$\beta = \text{softmax}(m) \in \mathbb{R}^N$$

$$c' = \sum_{i=1}^N \beta_i c_i \in \mathbb{R}^{2h}$$

For each position, output of BiDAF layer is

$$b_i = [c_{ij}; a_i; c_i \otimes a_i; c_i \otimes c'] \in \mathbb{R}^{8h}$$

FusionNet

$$c_{ij} = s^T \tanh(w_1 c_i + w_2 q_i)$$

$O(mnk)$  -  $w$  is fixed

(MLP Additive)

Bilinear form:

$$S_{ij} = c_i^T w q_j$$

$$S_{ij} = c_i^T U^T V q_j$$

$$S_{ij} = c_i^T W^T D w q_j$$

$$S_{ij} = \text{ReLU}(c_i^T w_i^T) D \text{ReLU}(w q_j)$$

$\mathcal{O}((m+n)k)$

ELMo and BERT

[2018] Contextual word representation

[2018]

The transformer architecture used in BERT is [2017]  
soft as of attention on steroids. More factors  
BERT achieves SOTA in QA

## Lecture 11: ConvNets for NLP

From RNNs to ConvNets

- RNN cannot capture phrases without prefix context
- often capture too much of the last words in final vector

Main CNN idea:  
What if we compute vectors for every possible word subsequences of a certain length?

Convolution  $(f * g)[n] = \sum_{m=1}^M f[n-m]g[m]$   
↳ usually used to ~~classically~~ extract features from images.

1D conv for text  
kernel → context vector → padding → conv  
↓  
maxpool

Note so, in PyTorch

batch-size = 16

word-embed-size = 6

seq-len = 7

input = torch.randn(batch\_size, word-embed-size)

conv1 = conv1D (in=word-embed-size, 3, 3)

hidden1 = conv1(input)

hidden2 = torch.max(hidden1, dim=2) # maxpool

• Other less useful notions:

stride

local max pool, ~~sliding~~ strides

k-max pooling

dilation (skipping values)

• Single Layer CNN for Text Classification

Yoon Kim (2014) CNN for Sentence classification

Multi-channel input idea

• Initialized with pretrained word vector

• Start with two copies

• Backprop into only one set, keep other "static"

• Both channels are summed to  $q$  before max pooling

# Zhang and Wallace A sensitivity Analysis of CNN

## Regularizations

Dropout: Create masking vector  $m$  for all Bernoulli rv with probability  $p$  of being 1  
Reasoning: Prevents co-adaption (overfitting to see specific feature constellations)

Bag of Words

Window Model

CNNs

RNNs

gate units vertically [donor 209] [mitochondria 209]  
Grating skipping that we saw in LSTMs and GRU's  
are in a ~~gate~~ is a general idea and  
used in other place like Resnet [performing]

### Batch normalization:

[Ioffe and Szegedy 2015]

- Often used in CNNs
- Transform the convolution output of a batch by scaling the activations to have zero mean and unit variance
- Familiar to Z-transform of statistic
- Updated per batch so fluctuations don't affect things much
- Makes models much less sensitive
- Use Batch Normalization [initialization] since outputs are automatically rescaled.
- It also tends to make tuning of learning simpler

## $1 \times 1$ convolutions (one convolution)

[Lin, Chen, Yen 2013, Network-in-Network]

- $1 \times 1$  convolutions, o.k.a. Network-in-Networks (NiN)  
connections, are convolutional kernels with  
kernel-size = 1
- A  $1 \times 1$  conv gives you fully connected  
linear layer across channels
- It can be used to map from many channels  
to fewer channels or even a lot of channels.  
 $\rightarrow$  unlike FC layers which add a lot of params
- $1 \times 1$  convs add additional NN layers without  
very few additional parameters

## Learning Character-level Representations for POS

Dos Santos and Zadrozny [2014]

- Convolution over characters to generate word embeddings.
- Fixed window of word embeddings used for POS tagging.

# Very Deep ConvNets for Text Classification

- Conneau, Schwenk, Lecan, Barrault, EACL 2017
- What happens when we build a vision-like deep model for NLP.

VD-CNN

Very unlike ~~DL~~ Deep learning NLP system

→ Each conv block is two convolutional layers,  
each followed by batch norm and a ReLU non-linearity

- Convolutions of size 3
- Pad to ~~reserve~~ preserve dimension

RNNs are slow

- Parallelize badly (GPU doesn't work like CNN)

Idea: take the best and parallelizable parts of  
RNNs and CNNs

Quasi-RNN [ICLR 2017]