Recurrent networks

RNNs
Examples of tasks and sequence types

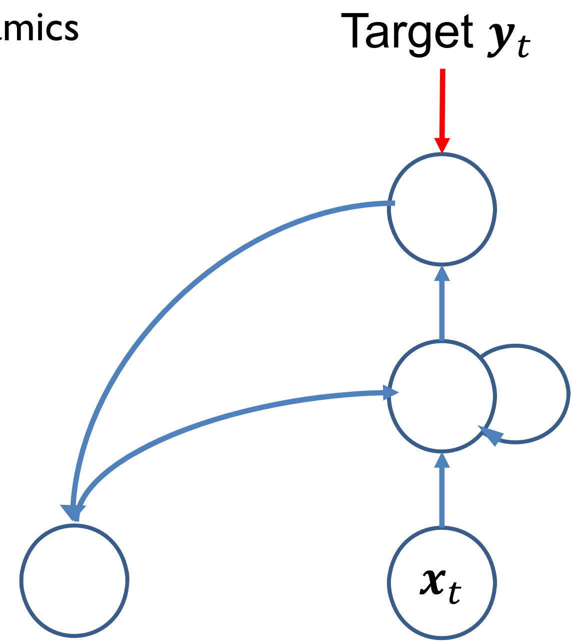
- Sequence classification
  - Input: sequence, output: class
    - Time series classification
    - Sentence classification (topic, polarity, sentiment, etc.)
- Sequence generation
  - Input: initial state (fixed vector), output: sequence
    - Text Generation
    - Music
- Sequence to sequence transduction
  - Input: sequence, output: sequence
    - Natural language processing: Named Entity recognition
    - Speech recognition: speech signal to word sequence
    - Translation

RNNs

- Several formulations of RNN where proposed in the late 80s, early 90s
  - They faced several limitations and were not successful for applications
    - Recurrent NN are difficult to train
    - They have a limited memory capacity
- Mid 2000s successful attempts to implement RNN
  - e.g. A. Graves for speech and handwriting recognition
  - new models where proposed which alleviate some of these limitations
- Today
  - RNNs are used for a variety of applications e.g., speech decoding, translation, language generation, etc
  - They became SOTA for sequence processing tasks around 2015. In 2020 alternative NN ideas (Transformers) have replaced RNNs for most discrete sequence modeling tasks. Initially developped as language models, they are used today in vision and multimodal (e.g. text-image) tasks.
- In this course
  - We briefly survey some of the developments from the 90s
  - We introduce recent developments on RNNs

RNNs

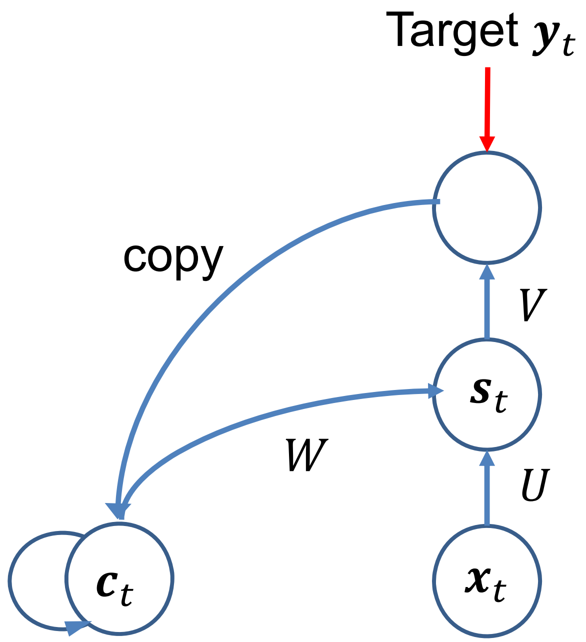
- Imagine a NN with feedback loops, i.e. no more a DAG
  - This transforms the NN into a dynamical/ state-space system
    - Information can circulate according to different dynamics
      - Convergence, stable state?
    - Supervision can occur at different times
    - Inputs: fixed, sequences, etc….

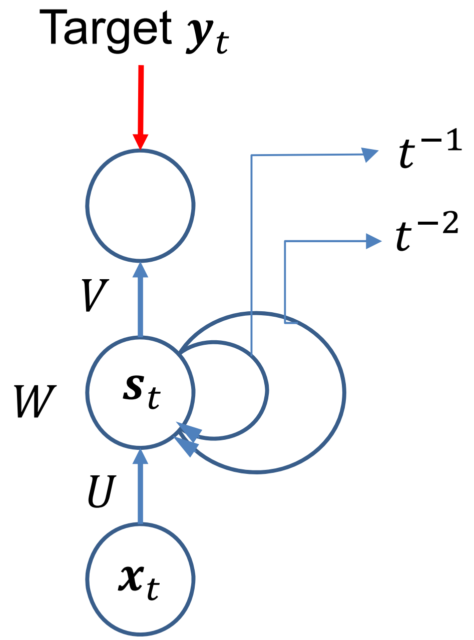


- Two main families
  - Global connections
  - Local connections
- In practice, only a limited class of RNNs is used for applications

RNNs local connections (90s)

- Several local connection architectures proposed in the 90s





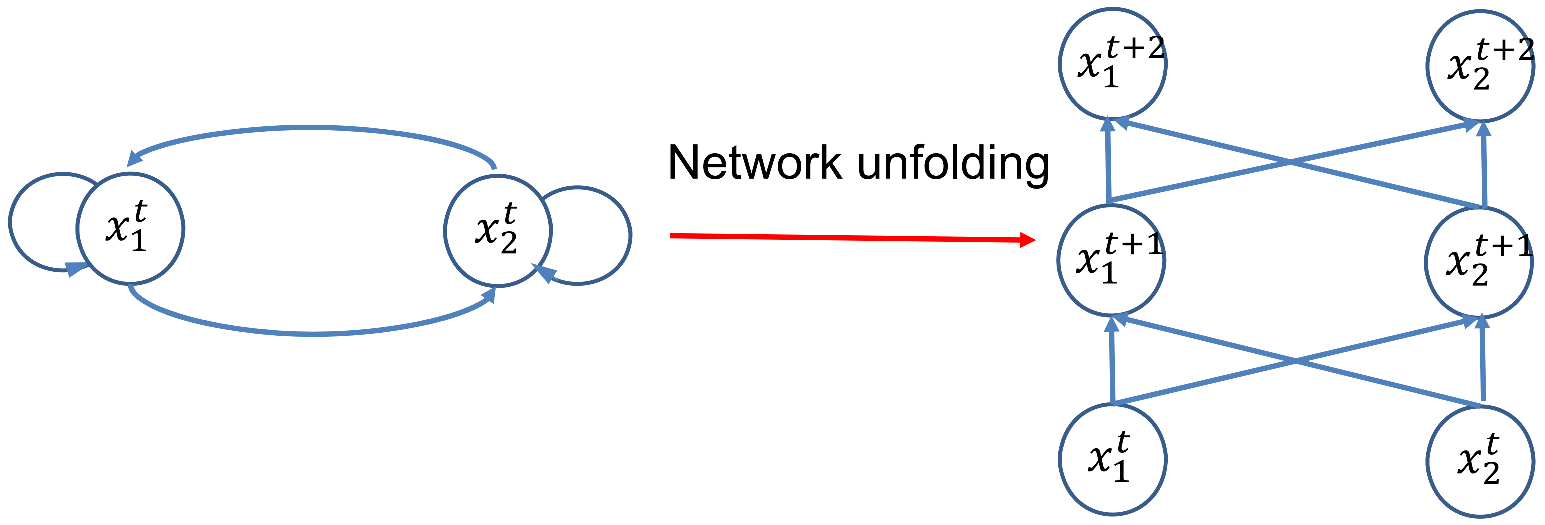
Fixed weights

Only the forward weights are learned: SGD $\boldsymbol{s}_\mathrm{t} = f(W\boldsymbol{c}_t) + U\boldsymbol{x}_t$

All weights learned

$\boldsymbol{s}_\mathrm{t} = f(W\boldsymbol{s}_{t-1}) + U\boldsymbol{x}_t$

RNNs global recurrences (90s)



- Algorithm
  - Back Propagation Through Time (BPTT)
  - For general sequences: $O(n^4) if\ n$ units

Fig. (Pearlmutter, 1995, IEEE Trans. on Neural Networks – nice review paper on RNN form the 90s)

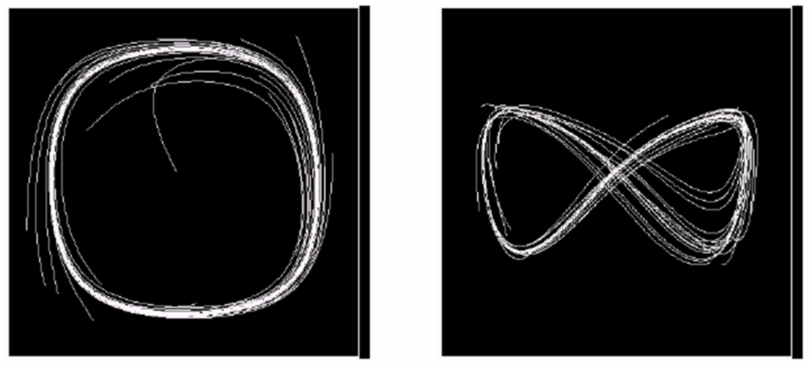
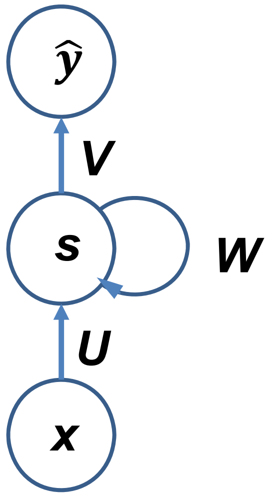


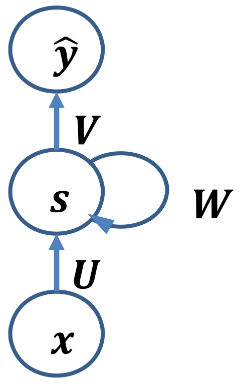
Fig. 9. The output states y_1 and y_2 plotted against each other for a 1000 time unit run, with all the units in the network perturbed by a random amount about every 40 units of time. The perturbations in the circle network (left) were uniform in ± 0.1, and in the figure eight network (right) in ± 0.05.

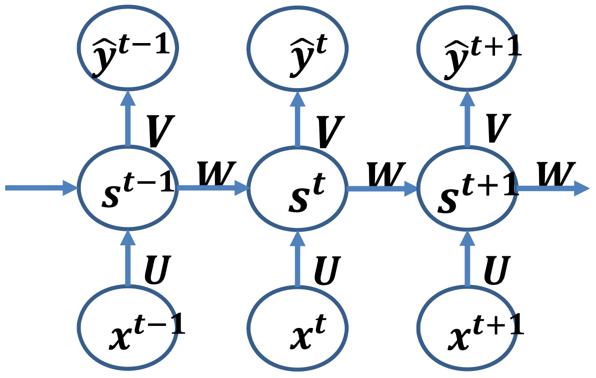
Dynamics of RNN

- We consider different tasks corresponding to different dynamics
  - They are illustrated for a simple RNN with loops on the hidden units
  - This can be extended to more complex architectures
  - However, RNNs used today all make use of local connections similar to this simple RNN
- Basic architecture



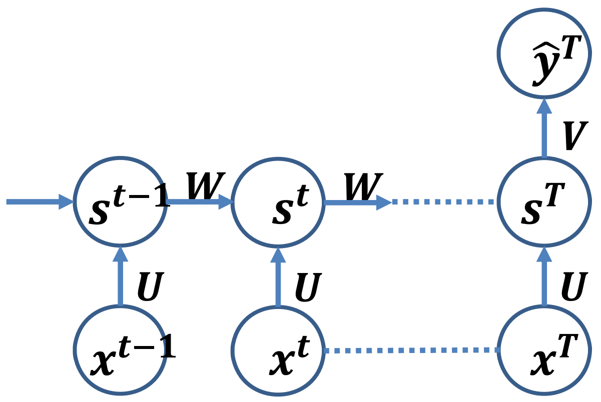
RNNs
Dynamics of RNN – unfolding the RNN



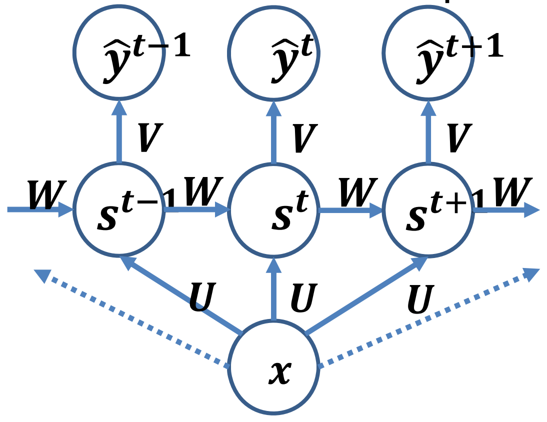


$\hat{y}^{t-1}$ $\hat{y}^{t}$ $\hat{y}^{t+1}$
V V V
s^{t-1} W s^{t} W s^{t+1} W
U U U
x^{t-1} x^{t} x^{t+1}

Many to many, e.g. speech or handwriting decoding, Part of Speech Tagging

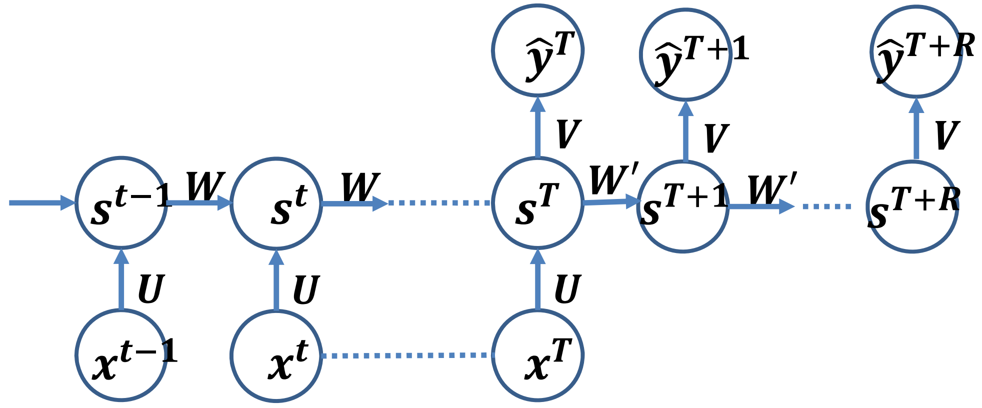


Many to one, e.g. sequence classification



$\hat{y}^{t-1}$ $\hat{y}^{t}$ $\hat{y}^{t+1}$
V V V
W s^{t-1} W s^{t} W s^{t+1} W
U U U
x

One to many, e.g. image annotation

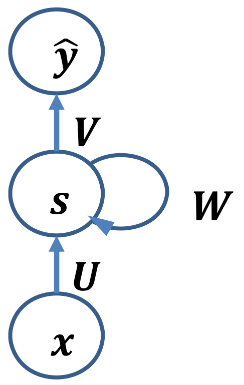


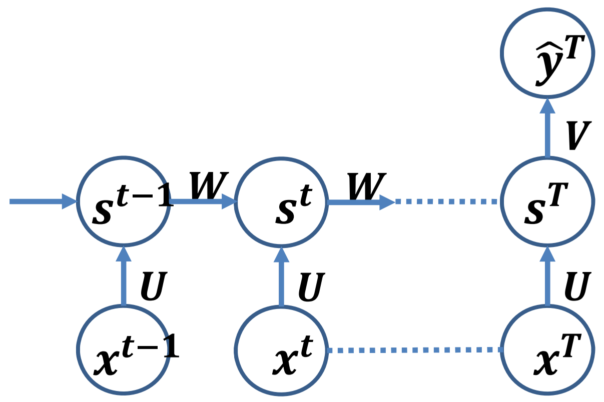
$\hat{y}^{T}$ $\hat{y}^{T+1}$ $\hat{y}^{T+R}$
V V V
s^{t-1} W s^{t} W s^{T} W' s^{T+1} W' s^{T+R}
U U U
x^{t-1} x^{t} x^{T}

Many to many, e.g. translation

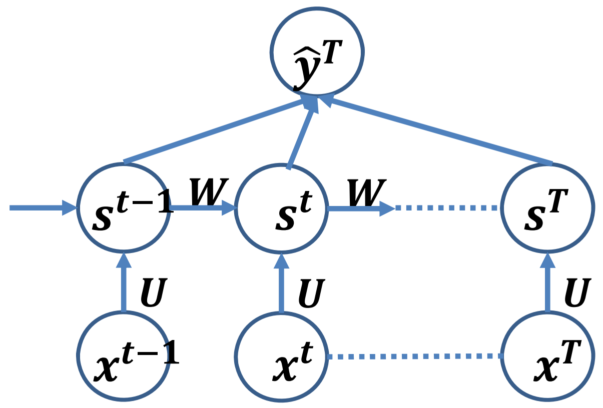
RNNs
Dynamics of RNN – unfolding the RNN

- Different ways to compute sequence encodings





- The final state s^T encodes the sentence



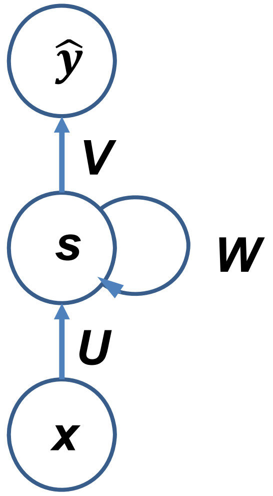
- The whole state sequence encodes the input sequence – usually better: take elementwise max or mean of the hidden states.
- More on that on Attention and Transformers

RNNs
Back Propagation Through Time

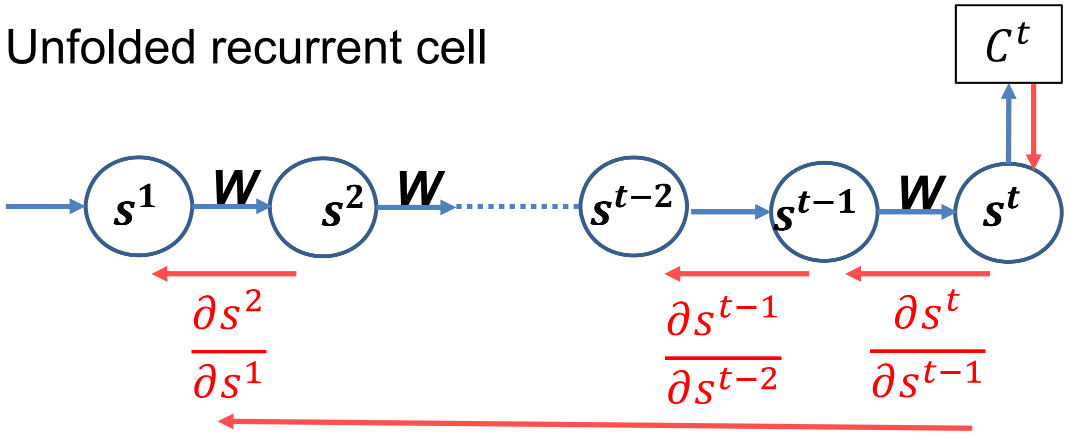
- By unfolding the RNN, one can see that one builds a Deep NN
- Training can be performed via SGD like algorithms
  - This is called Back Propagation Through Time
- Automatic Differentiation is used for training the RNNs
- RNNs suffer from the same problems as the other Deep NNs
  - Gradient exploding
    - Solution: gradient clipping
  - Gradient vanishing
    - In a vanilla RNN, gradient information decreases exponentially with the size of the sequence
  - Plus limited memory
    - Again exponential decay of the memory w.r.t. size of the sequence
- Several attempts to solve these problems
  - We introduce a popular family of recurrent units that became SOTA around 2015:
    - Gated units (GRU, LSTMs)

RNNs
Recurrent units: Long Short Term memory (LSTM – Hochreiter 1997), Gated Recurrent Units (GRU – Cho 2014)

- Vanishing gradient problem
  - Consider a many to many mapping problem such as decoding or building a language model (more on that later)



Unfolded recurrent cell



C^t
s^1
W
s^2
W
s^{t-2}
s^{t-1}
W
s^t
$\frac{\partial s^2}{\partial s^1}$
$\frac{\partial s^{t-1}}{\partial s^{t-2}}$
$\frac{\partial s^t}{\partial s^{t-1}}$

$$s^{t+1} = f(Ws^t + Ux^{t+1})$$

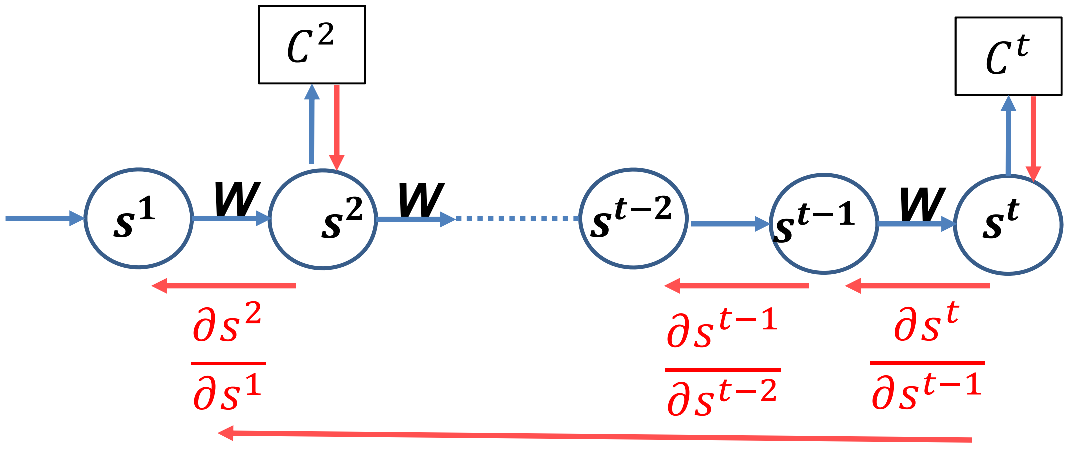
Gradient flow: vanishing gradient

$$\frac{\partial C^t}{\partial s^1} = \frac{\partial s^2}{\partial s^1} \text{x} \dots \text{x} \frac{\partial s^t}{\partial s^{t-1}} \frac{\partial C^t}{\partial s^t}$$

If any of these quantities is small, the gradient from C^t gets smaller and smaller

RNNs
Recurrent units: Long Short Term memory (LSTM – Hochreiter 1997), Gated Recurrent Units (GRU – Cho 2014)

- Vanishing gradient problem



C^2
C^t
s^1
s^2
s^{t-2}
s^{t-1}
s^t
W
W
W
$\frac{\partial s^2}{\partial s^1}$
$\frac{\partial s^{t-1}}{\partial s^{t-2}}$
$\frac{\partial s^t}{\partial s^{t-1}}$

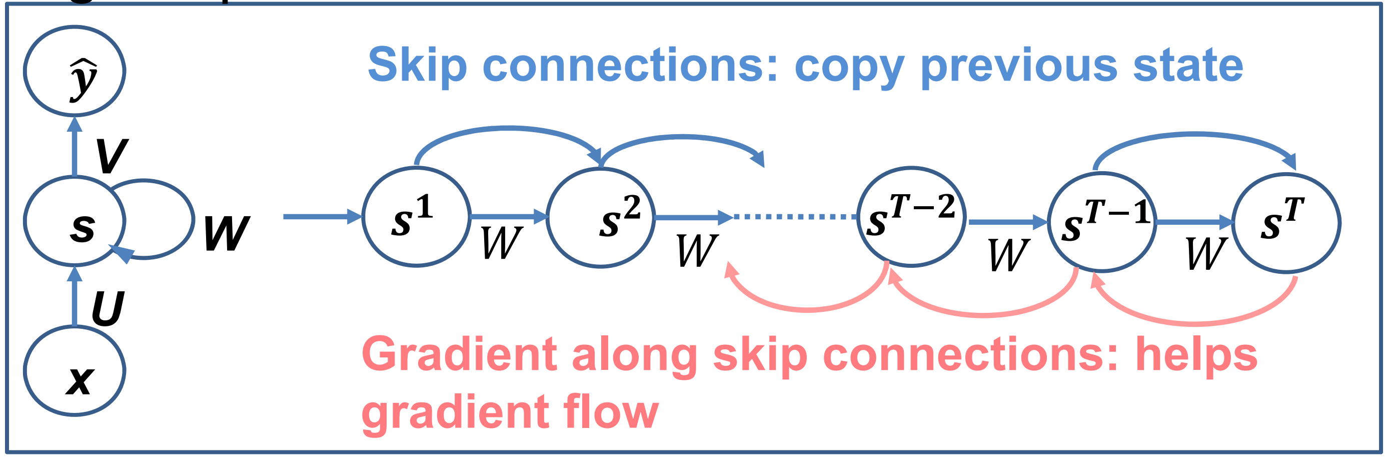
- In this example, the gradient from C^2 is much stronger than the gradient from C^t
- This means that « long » term dependencies are difficult to capture with RNNs

$$\frac{\partial C^t}{\partial s^1} = \frac{\partial s^2}{\partial s^1} \text{x} \dots \text{x} \frac{\partial s^t}{\partial s^{t-1}} \frac{\partial C^t}{\partial s^t}$$

$$\frac{\partial C^2}{\partial s^1} = \frac{\partial s^2}{\partial s^1} \frac{\partial C^2}{\partial s^2}$$

RNNs - Gated Units
Long Short Term memory (LSTM – Hochreiter 1997)
Gated Recurrent Units (GRU – Cho 2014)

- Introducing « skip connections » - similar to ResNet


Skip connections: copy previous state
$\hat{y}$
V
s
W
U
x
s^1
W
s^2
W
s^{T-2}
W
s^{T-1}
W
s^T
Gradient along skip connections: helps gradient flow

Past value

New candidate value:

$$\boldsymbol{s}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{s}^{t-1} + \boldsymbol{z}^t \odot \boldsymbol{s}'^t$$

Gating mechanism

$$\boldsymbol{s}'^t = \tanh(U\boldsymbol{x}^t + W s^{t-1})$$
$$\boldsymbol{z}^t = \sigma(U_z\boldsymbol{x}^t + W_z\boldsymbol{s}^{t-1})$$

$\odot$ is the Hadamard product

U_z and W_z learned by SGD

RNNs
Gated Recurrent Units (GRU – Cho 2014)
Skip connections

- The output s_j^t of cell j is a weighted sum of the cell output at time $t-1$, s_j^{t-1} and a new value of the cell s'^t_j
  - $\boldsymbol{s}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{s}^{t-1} + \boldsymbol{z}^t \odot \boldsymbol{s}'^t$
  - z is a gating function
    - If $z = 0$, s_j^t is a simple copy of s_j^{t-1}
    - If $z = 1$ it takes the new value s'^t_j
    - w.r.t the classical recurrent unit formulation, this new form allows us to remember the value of the hidden cell at a given time in the past and reduces the vanishing gradient phenomenon

RNNs
Gated Recurrent Units (GRU – Cho 2014)

- Skip connection with Forget Gate + Reset Gate

Past value

New candidate value:

$$\boldsymbol{s}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{s}^{t-1} + \boldsymbol{z}^t \odot \boldsymbol{s}'^t$$

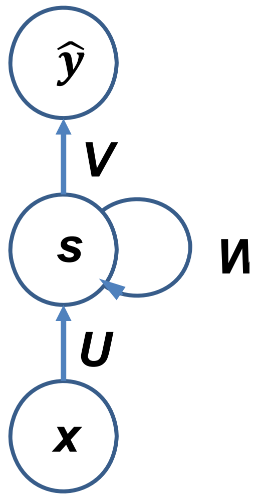
Gating mechanism

$\boldsymbol{s}'^t = \tanh(U\boldsymbol{x}^t + W(r^t \odot s^{t-1}))$

Forget gate $\boldsymbol{z}^t = \sigma(U_z\boldsymbol{x}^t + W_z\boldsymbol{s}^{t-1})$

Reset Gate $\boldsymbol{r}^t = \sigma(U_r\boldsymbol{x}^t + W_r\boldsymbol{s}^{t-1})$

$\odot$ is the Hadamard product



RNNs
Gated Recurrent Units (GRU – Cho 2014) - followed

- The gating function is a function of the current input at time t and the past value of the hidden cell $\boldsymbol{s}^{t-1}$
  - $\boldsymbol{z}^t = \sigma(U_z\boldsymbol{x}^t + W_z\boldsymbol{s}^{t-1})$
- The new value $\boldsymbol{s}'^t$ is a classical recurrent unit where the values at time $t-1$ are gated by a reset unit $\boldsymbol{r}_t$
  - $\boldsymbol{s}'^t = \tanh(U\boldsymbol{x}^t + W(r^t \odot s^{t-1}))$
- The reset unit $\boldsymbol{r}^t$allows us to forget the previous hidden state and to start again a new modeling of the sequence
  - This is similar to a new state in a Hidden Markov Model (but it is soft)
  - $\boldsymbol{r}^t = \sigma(U_r\boldsymbol{x}^t + W_r\boldsymbol{s}^{t-1})$

RNNs
Gated Recurrent Units (GRU – Cho 2014)

- There are two main novelties in this GRU
  - The z gating function which implements skip connections and acts for reducing the vanishing gradient effect
  - The r gating function which acts for forgeting the previous state and starting again a new subsequence modeling with no memory

- Each unit adapts its specific parameters, i.e. each may adapt its own time scale and memory size

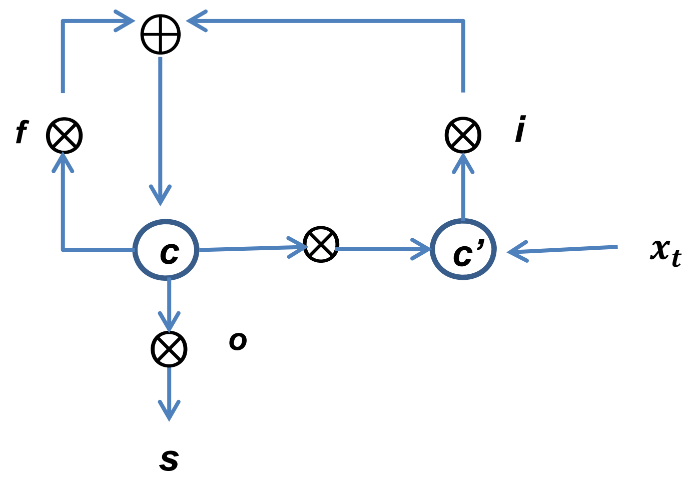
- Training
  - is performed using an adaptation of backpropagation for recurrent nets
  - All the functions – unit states and gating functions are learned from the data using some form of SGD

Long short term memory - LSTM

- This was initially proposed in 1997 (Hochreiter et al.) and revised later.
- State of the art on several sequence prediction problems
  - Speech, handwriting recognition, translation
  - Used in conjontions with other models e.g. HMMs or in standalone recurrent neural networks
  - The presentation here is based on (Graves 2012)

Long short term memory

- In the LSTM, there are 3 gating functions
  - i: input gating
  - o: output gating
  - f: forget gating

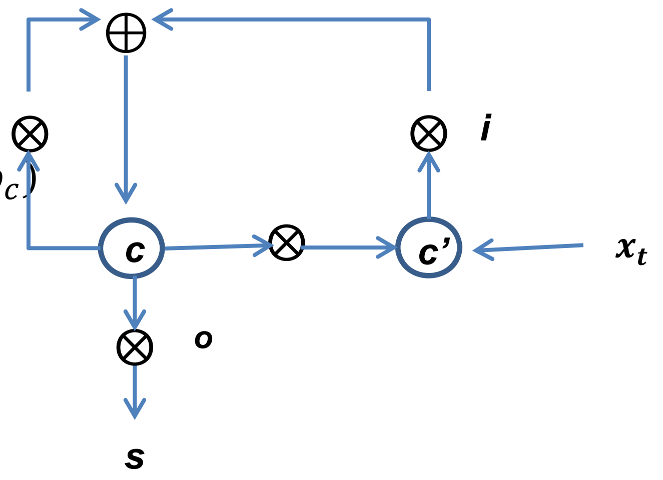


- Difference with the gated recurrent cell
  - Similarities
    - Both use an additive form for computing the hidden cell state (c) here.
      - This additive component reduces the vanishing gradient effect and allows us to keep in memory past state values.
    - Both use a reset (called here forget (f)) gate
      - The reset permits to start from a new « state » a subsequence prediction
  - Differences
    - No output gating in the GRU
    - Reset does not play exactly the same role

Long short term memory

- For the forward pass, the different activations are computed as follows and the this order

  - $i^t = \sigma(W_{xi}x^t + W_{hi}s^{t-1} + W_{ci}c^{t-1} + b_i)$
  - $f^t = \sigma\left(W_{xf}x^t + W_{hf}s^{t-1} + W_{cf}c^{t-1} + b_f\right)$
  - $c^t = f_t \odot c^{t-1} + i_t \odot \tanh(W_{xc}x^t + W_{hc}s^{t-1} + b_c)$
  - $o^t = \sigma(W_{xo}x^t + W_{ho}s^{t-1} + W_{co}c^{t-1} + b_o)$
  - $s^t = o^t \tanh(c^t)$



  - c_t^i is a memory of cell *i* at time *t*, $c_t$is computed as for the GRU as a sum of c_{t-1} and of the new memory content $c_t' = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$
  - o is an output gate
  - σ is a logistic function
  - W_{ci}, W_{cf}, W_{co} are diagonal matrices

Bidirectional and multilayer RNNs

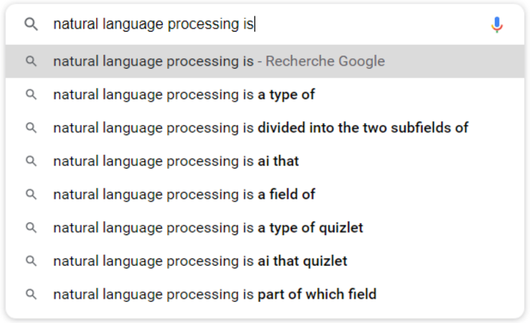
RNNs Future

- RNNs variants (GRU, LSTM) became the dominant approach around 2015, for several tasks including speech recognition, translation, text generation etc
- These last years (2019-2020) they have become superseded by other approaches for many of these tasks
  - Transformers are now more frequently used for a large variety of tasks dealing with discrete sequences, in NLP for example
  - Note: after the Transformer » revolution » in NLP, their use in other domains s.a. vision, is increasing.

Language models

- Objective:
  - Probability models of sequences $(x^1, x^2, \dots, x^t)$
  - Items may be words, characters, character ngrams, word pieces, etc
  - Formally: given a sequence of items, what is the probability of the next item?
    - $p(x^t | x^{t-1}, \dots, x^1)$
- Example
  - « S'il vous plaît… dessine-moi …» what next ?
  - « $x^1 x^2 x^3 \dots \dots \dots \dots \dots . \dots . . x^{t-1} \dots$ » what is x^t ?
- Language models in everyday use
  - Sentence completion
    - Search engine queries
    - Smartphone messages, etc
  - Speech recognition, handwriting recognition, etc

Google


natural language processing is
natural language processing is - Recherche Google
natural language processing is a type of
natural language processing is divided into the two subfields of
natural language processing is ai that
natural language processing is a field of
natural language processing is a type of quizlet
natural language processing is ai that quizlet
natural language processing is part of which field

Language models

- Language models can be used to compute the probability of a piece of text
- Let $(x^1, x^2, \dots, x^{\mathrm{T}})$ be a sequence of text, its probability according to a language model is:
  - $p(x^1, x^2, \dots, x^{\mathrm{T}}) = \prod_{t=1}^{T} p(x^t | x^{t-1}, \dots, x^1)$
    - With $p(x^t | x^{t-1}, \dots, x^1)$ computed by the language model

Language models
How to learn a language model - n-grams

- A simple solution: n-grams
  - n-grams are sequences of n consecutive words (or characters, or any items)
  - Language model is based on n-gram statistics
  - Markov assumption
    - x^t only depends on the $n-1$ preceding words
      - $p(x^t|x^{t-1},\dots,x^1)=p(x^t|x^{t-1},\dots,x^{t-n+1})$
    - Use Bayes formula $p(x^t|x^{t-1},\dots,x^{t-n+1}) = \dfrac{p(x^t,x^{t-1},\dots,x^{t-n+1})}{p(x^{t-1},\dots,x^{t-n+1})}$

n-gram probability

n-1-gram probability

  - Given large text collections, it is possible to compute estimates of the posterior probabilities
    - An estimate could be $\hat{p}(x^t|x^{t-1},\dots,x^{t-n+1}) = \dfrac{count(x^t,x^{t-1},\dots,x^{t-n+1})}{count(x^{t-1},\dots,x^{t-n+1})}$
    - Where $count(x^t,x^{t-1},\dots,x^{t-n+1})$ is the number of occurrences of the sequence in the corpus

Language models
n-grams

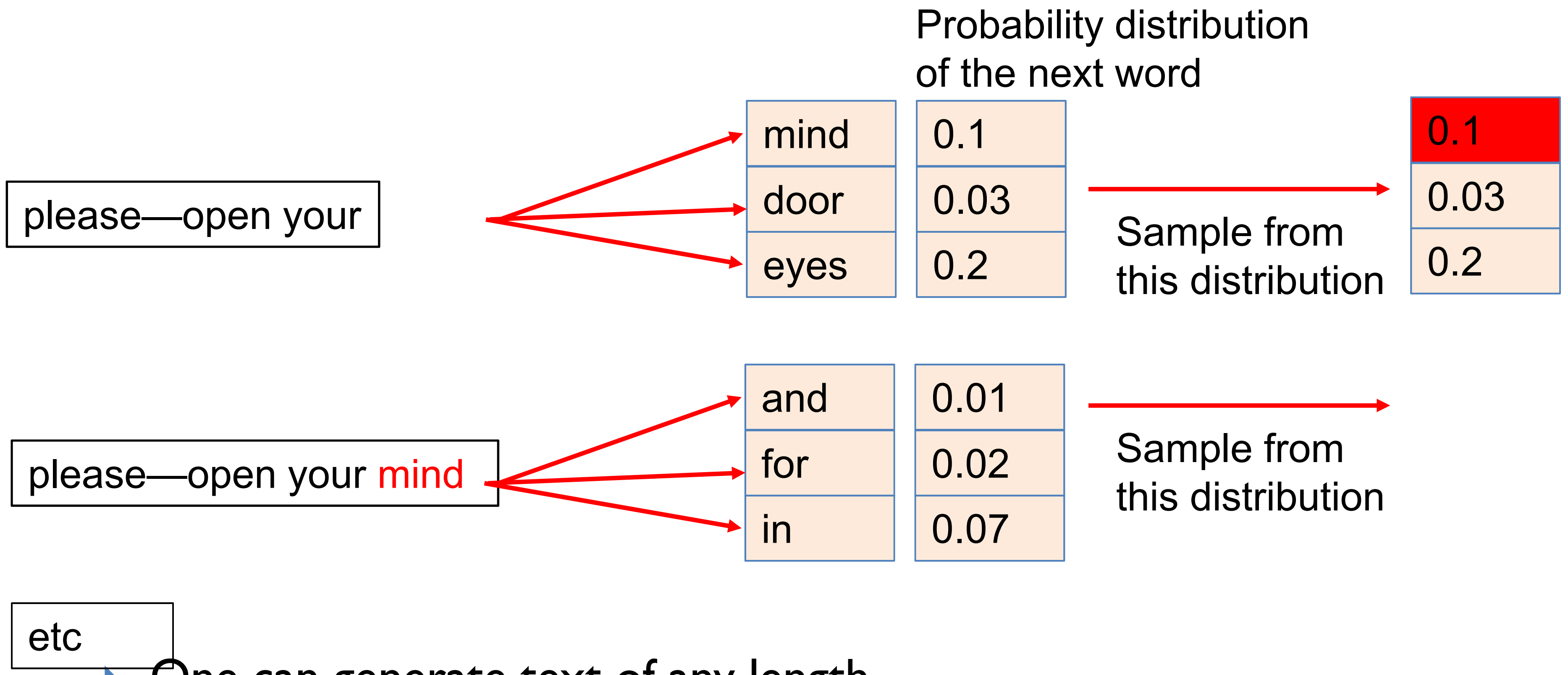
- Sparsity problem
  - In order to get good estimates, this requires large text quantities
  - The larger n is, the larger the training corpus should be
  - For a dictionnary of 10 k words, there could be
    - $10^{4\text{x}2}$ bigrams
    - $10^{4\text{x}3}$ trigrams, etc
    - Note: the number of n-grams in a language is smaller than $10^{4\text{x}n}$ but still extremely large and grows exponentially with n
    - The model size increases exponentially with n
  - n-gram counting is limited to relatively short sequences
    - Only large companies like Google could afford computing/ storing estimates for $n > 10$

Language models
n-grams

- Additional problems
  - Consider the sentence « Please open your mind » and a 4-gram model
    - What if « mind » never occured in the corpus?
      - The probability of the sequence becomes 0, which is not realistic
      - Solution: every 4-gram is set to a minimum probability value of ϵ
      - This is a smoothing operation – there exists different smoothing estimates
    - What if « Please open your » never occured in the corpus?
      - The 4-gram probability cannot be computed
      - Smooth using backoff estimates
      - e.g. $p(please\ open\ your\ mind) = p(open\ your\ mind)$
  - More generally, n-gram models are often smoothed with n-1 gram, n-2 grams etc
    - $p(x^t|x^{t-1}, \dots, x^{\mathrm{t-n+1}}) \backsimeq \sum_{i=1}^{n-1} \alpha_i p\left(x^t\middle|x^{t-1}, \dots, x^{\mathrm{t-n+}i}\right)$

Language models
n-grams – text generation

- Any language model can be used for text generation



Probability distribution of the next word
please—open your
mind
door
eyes
0.1
0.03
0.2
Sample from this distribution
0.1
0.03
0.2
please—open your mind
and
for
in
0.01
0.02
0.07
Sample from this distribution
etc

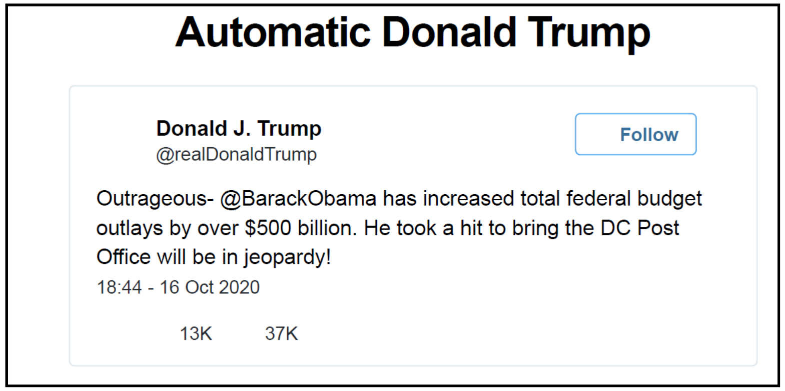
- One can generate text of any length

Language models
n-grams – text generation

- Example from https://projects.haykranen.nl/markov/demo/
  - 4 gram trained on the Wikipedia article on Calvin and Hobbes
  - Generated text

Rosalyn is a standary children used each otherwise as he stereotypically comic stand for an impulsive real-life Watterson's stuffed tiger, much as "grounded in reality rathmore spacious circle: because associety The club has said they have the archive shifting into low art some of the strip was one larger than Calvin articipate indulges in his hands attribute red-and-black pants, magenta socks and Susie Derkins specifically characters like school where were printerestrainstory

- Example from https://filiph.github.io/markov/


Automatic Donald Trump

Donald J. Trump
@realDonaldTrump

Follow

Outrageous- @BarackObama has increased total federal budget outlays by over $500 billion. He took a hit to bring the DC Post Office will be in jeopardy!

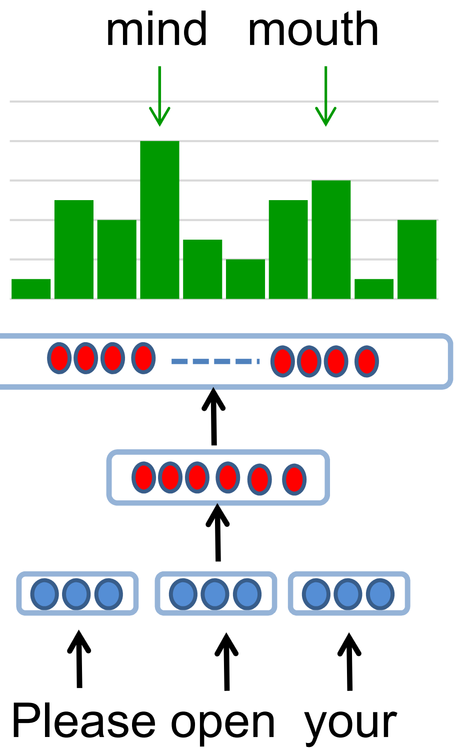
18:44 - 16 Oct 2020

13K 37K

Language models
Neural networks

- Fixed input size NN

- The NN could be typically a convolutional NN with all the input word representations sharing the same weights
- It could also be made fully convolutional
- Less sensitive than n-grams to sparsity



- Posterior estimate of the next word
- Classification layer, softmax among all vocabulary words
- Hidden layer(s)
- Word representation, e.g. w2Vec
- Input sentence, one hot encoding

RNNs
Language models

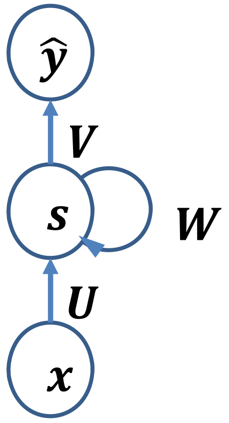
- RNNs offer an alternative approach to non recurrent NNs
- Objective:
  - Probability models of sequences $(x^1, x^2, \ldots, x^t)$
  - Estimate with RNNs:
    - $p(x^t | x^{t-1}, \ldots, x^1)$

prediction

$\hat{\boldsymbol{y}}^t = g(V\boldsymbol{s}^t)$

memory

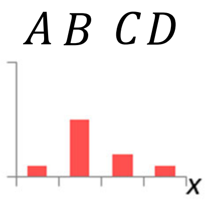
$s^t = f(Ws^{t-1} + Ux^t)$



- g is typically a softmax
- f could be a sigmoid, Relu, …
- x will usually be a word/ item representation learned from large corpora

Recurrent neural networks Language models

- Training
  - Use a corpus of text, e.g. a sequence of words $(x^1, x^2, \dots, x^T)$
  - Feed the sequence into the RNN, one word at a time
  - Compute the output distribution $\hat{\boldsymbol{y}}^t$ for each time step
    - $\hat{\boldsymbol{y}}^t$ is a distribution on the word dictionary
      - This is the estimated posterior probability distribution given past subsequence
      - If the dictionary is $V = \{A, B, C, D\}$:



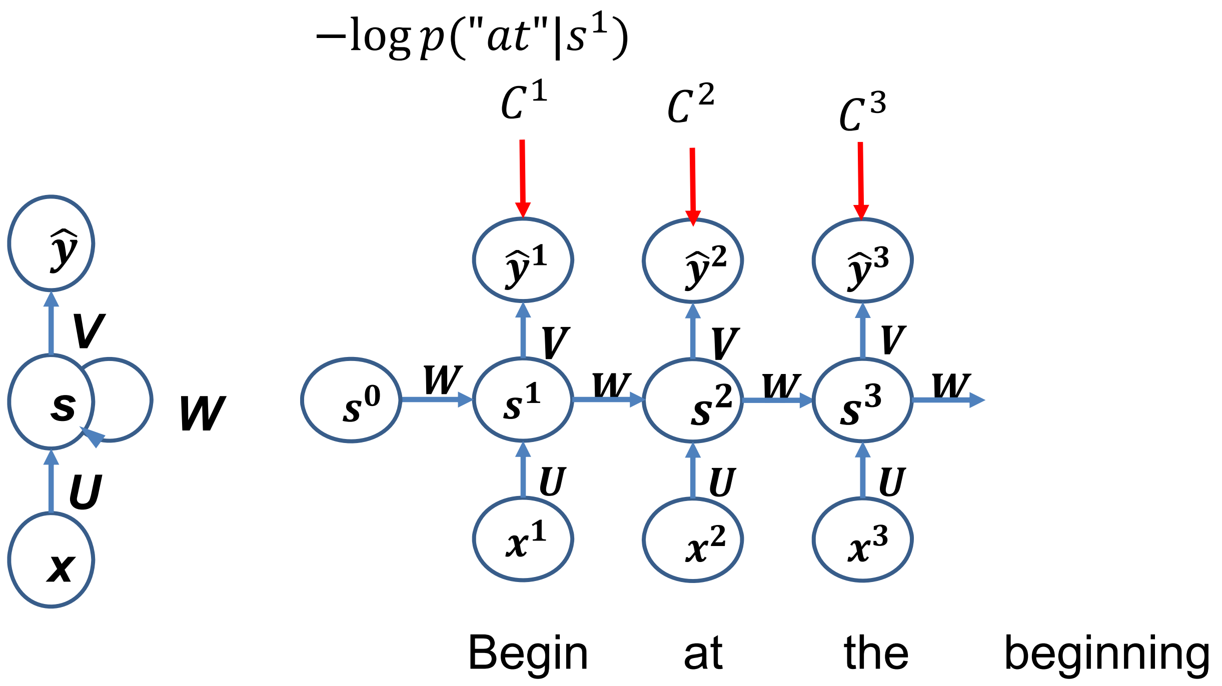
$$\hat{\boldsymbol{y}}^t = \boldsymbol{P}(\boldsymbol{x}^{t+1} | \boldsymbol{s}^t)$$

      - Loss function
        - Classically the cross entropy between distribution $\hat{\boldsymbol{y}}^t$ and the target distribution $\boldsymbol{y}^t$
        - Loss at time t in the sequence: $C^t = C(\hat{\boldsymbol{y}}^t, \boldsymbol{y}^t) = -\sum_{i=1}^{|V|} y_i^t log\, \hat{y}_\mathrm{i}^t = -log \hat{y}_{\mathrm{x_{t+1}}}^t$
          - With $\hat{y}_{\mathrm{x_{t+1}}}^t$ denoting the predicted output for the target class y_i^t (i.e. next word to predict)
        - Loss over a sequence of length T corpus $C = \sum_{t=1}^{T} C^t$
        - In practice, one uses a mini batch of sentences sampled from the corpus and use a stochastic gradient algorithm

Recurrent neural networks Language models

- Training

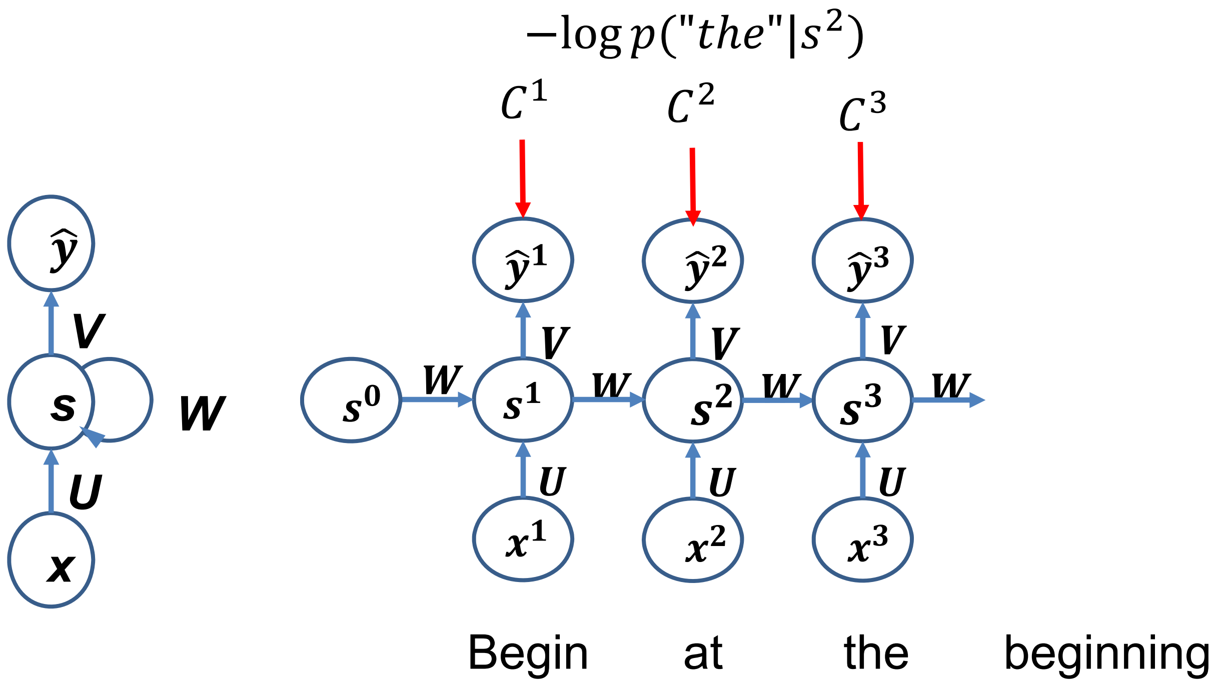
$$\hat{\boldsymbol{y}}^t = \boldsymbol{P}(\boldsymbol{x}^{t+1}|\boldsymbol{s}^t)$$



Recurrent neural networks Language models

- Training

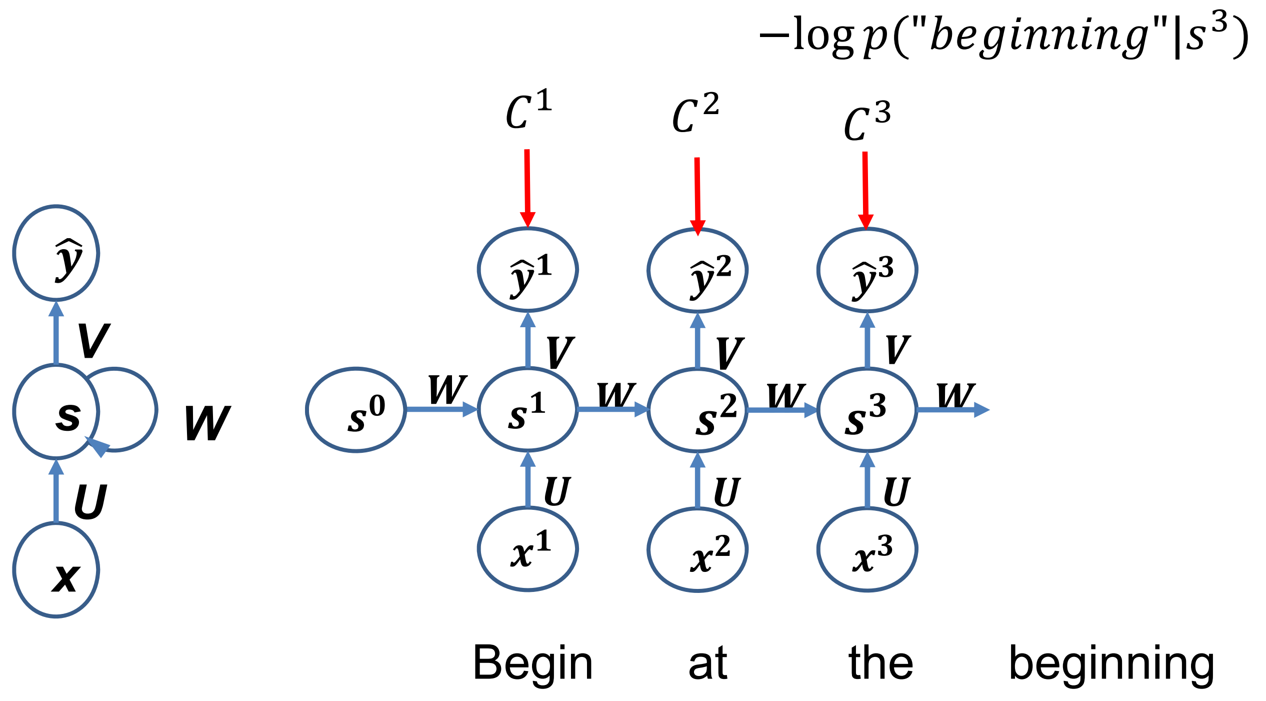
$$\hat{y}^t = P(x^{t+1}|s^t)$$



Recurrent neural networks Language models

- Training

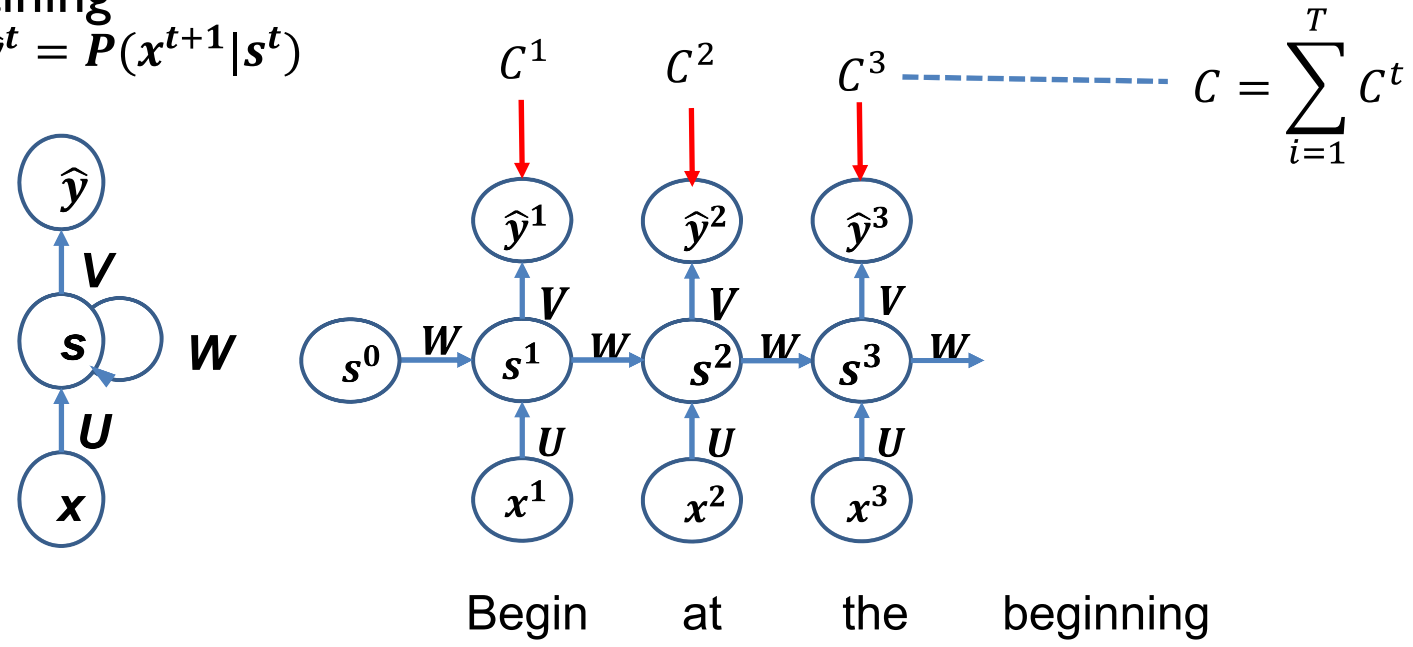
$\hat{y}^t = P(x^{t+1}|s^t)$



Recurrent neural networks Language models

- Training

  $\hat{\boldsymbol{y}}^t = \boldsymbol{P}(\boldsymbol{x}^{t+1}|\boldsymbol{s}^t)$



- Note
  - Weights are shared: only one U, one V, one W for the whole NN

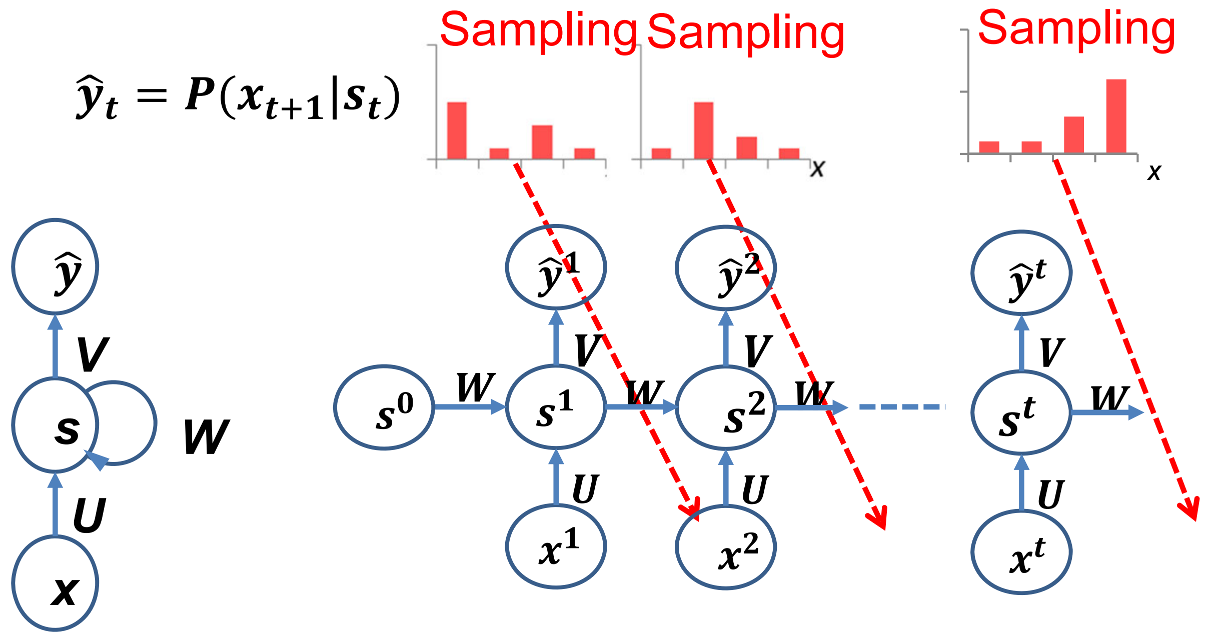
Recurrent neural networks Language models

- Training algorithm: Back Propagation Through Time - BPTT
  - Consider a sequence of words $(x^1, x^2, \dots, x^T)$ sampled from the training set
  - Loss function for a sequence : $C = \sum_{t=1}^{T} C^t$
    - SGD: compute the loss for the sequence (actually a batch of sequences), compute the gradient and upfate the parameters
    - Recall, weights are shared: only one U, one V, one W
  - Example: update of the shared W weights
    - Gradient of the loss for the whole sequence: compute the derivatives w.r.t. each C^t and sums them:
      - $\frac{\partial C}{\partial W} = \sum_{t=1..T} \frac{\partial C^t}{\partial W}$
    - Gradient of the loss for the loss at time t, C^t:
      - $\frac{\partial C^t}{\partial W} = \sum_{i=1}^{t} \left(\frac{\partial C^t}{\partial W}\right)_{(i)}$ where $\left(\frac{\partial C^t}{\partial W}\right)_{(i)}$ is the gradient of the loss w.r.t. weight at position $i \leq t$
        - Backpropagate over time steps $i = 1 \dots t$, summing the gradient: BPTT
  - This training regime is called teacher forcing
    - Successive sequential inputs correspond to the true sequence
    - Different during inference (see next slide)

RNNs
Language models

- Inference
  - Suppose the RNN has been trained
  - Inference processes by sampling from the predicted distribution

$$\hat{y}_t = P(x_{t+1}|s_t)$$



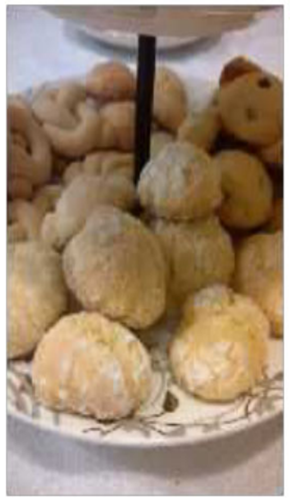
RNNs
Language models – Word representation

- Words, characters, n-grams, word pieces are all discrete data
- How to represent them
  - The usual way is to embed the words, etc in a continuous space of high dimension e.g. R^{200}, i.e. each word will be a vector in R^{200}
  - This could be done
    - Off line using some embeding technique (e.g. Word2Vec, see later)
      - Advantage, this can be done by using very large text collections
      - These representations could then be used for downstream tasks (e.g. classification)
    - On line while training the language model
      - In this case, the xs are initialized at random values in R^n and are learned by backpropagating the error, together with the other parameters
      - We usually loose the benefit of training on large corpora

Language models – examples

- Language models can be used to learn text representations, Generate text, Translation, Dialogue, etc

Inverse Cooking: Recipe Generation from Food Images, Salvador et al CVPR 2019



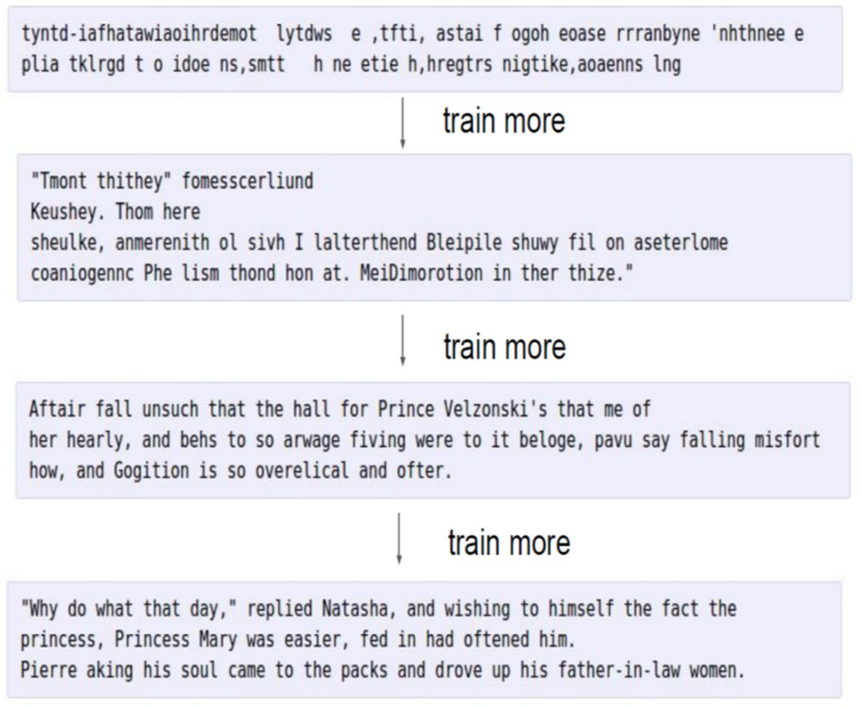
Title: Biscuits

Ingredients:
Flour, butter, sugar, egg, milk, salt.

Instructions:
- Preheat oven to 450 degrees.
- Cream butter and sugar.
- Add egg and milk.
- Sift flour and salt together.
- Add to creamed mixture.
- Roll out on floured board to 1/4 inch thickness.
- Cut with biscuit cutter.
- Place on ungreased cookie sheet.
- Bake for 10 minutes.

Figure 1: **Example of a generated recipe**, composed of a title, ingredients and cooking instructions.

Language generation, Training on Tolstoy's War and Peace a character language model, Stacked RNNs (LSTMs) (Karpathy 2015- https://karpathy.github.io/2015/05/21/rnneffectiveness/)



```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Learning word vector representations
Word2Vec model (Mikolov et al. 2013a, 2013b)

- Goal
  - Learn word representations
    - Words or language entities belong to a discrete space
    - They could be described using one hot encoding, but this is meaningless
    - How to represent these entities with meaningful representations?
  - Word2Vec model
    - Learn robust vector representation of words that can be used in different Natural Language Processing or Information retrieval tasks
    - Learn word representations in phrase contexts
    - Learn using **very** large text corpora
    - Learn efficient, low complexity transformations
  - Successful and influential work that gave rise to many developments and extensions
  - Still in use, but superseded by Transformer based learned representations

Semantics: words
How to encode words according to their semantic meaning

- **Representing words as discrete symbols**
  - In traditional NLP, we regard words as discrete symbols: Words can be represented by **one-hot vectors** - Each word is a distinct symbol
  - **Example**: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel".
    - motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
    - hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
      - These two vectors are orthogonal.
      - There is **no natural notion of similarity** for one-hot vectors!
  - **Vector dimension** = number of words in vocabulary (e.g., 500,000)
    - Very large dimensional discrete space - Problem for machine learning - sparsity

Semantics: words

- Instead: learn to encode similarity in the vectors themselves
  - GloVe (Pennington et al. 2014)

Nearest words to
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Words in vector space
Representing words by their context

- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
  - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
  - Use the many contexts of w to build up a representation of w

…government debt problems turning into **banking** crises as happened in 2009…
…saying that Europe needs unified **banking** regulation to replace the hodgepodge…
…India has just given its **banking** system a shot in the arm…

context words will represent ***banking***

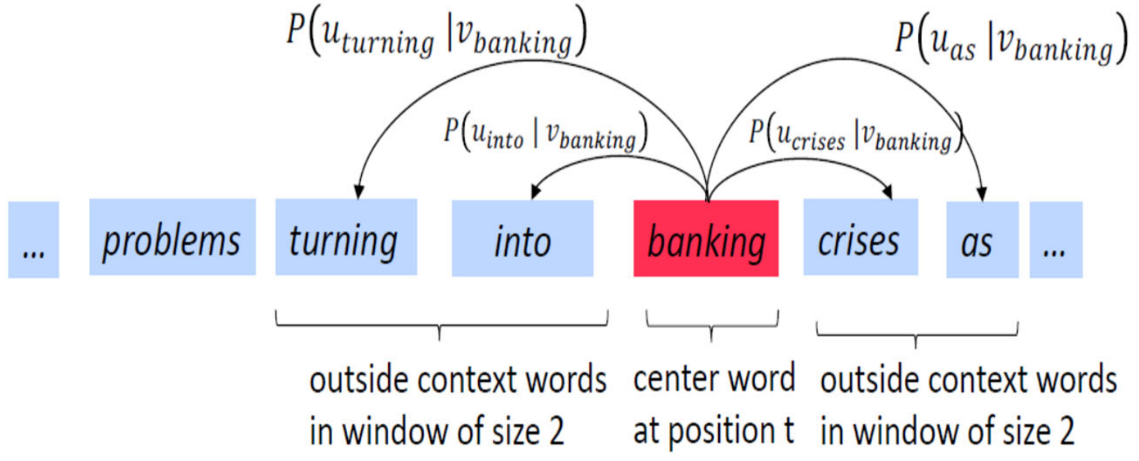
Words in vector space
Representing words by their context

- Word embeddings
  - We represent words by vectors so that words with similar contexts share « close » representations in the vector space

$$\boldsymbol{banking} = \begin{bmatrix} 0.87 \\ 0.45 \\ -0.34 \\ -0.63 \\ 0.23 \\ 0.16 \end{bmatrix}$$

- Key idea
  - These representations are learned from very large corpora for representing a large variety of situations/ contexts
    - No nead for supervision
  - These embeddings will be used for doswnstream tasks, e.g. classification

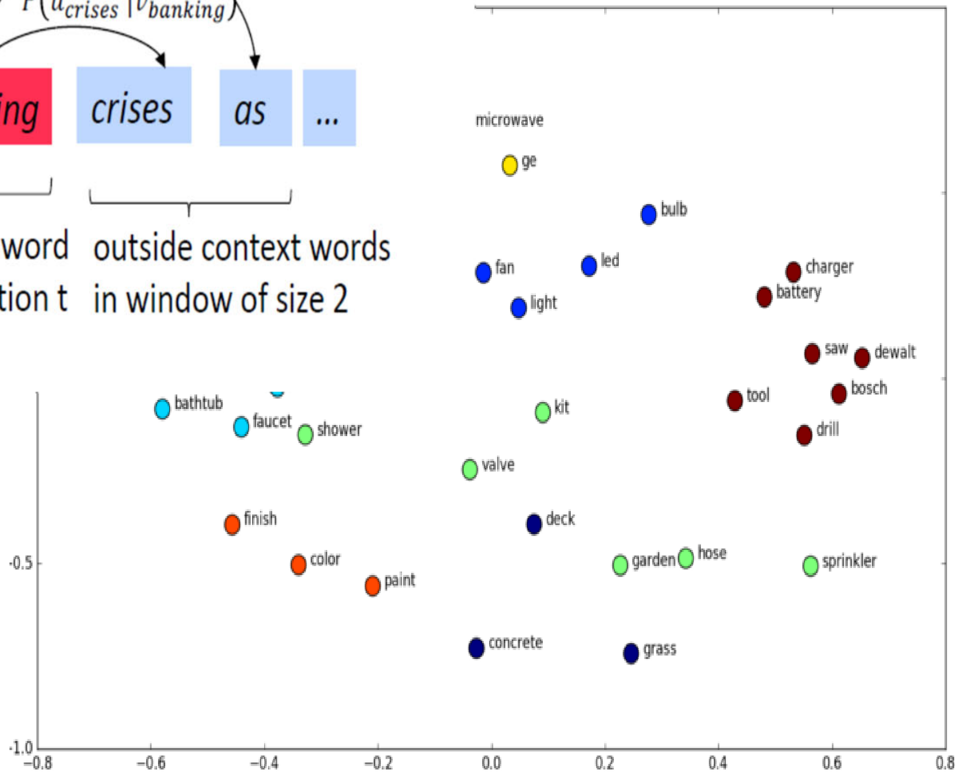
Word embeddings
Word2Vec – Mikolov et al. 2013



$P(u_{turning} | v_{banking})$
$P(u_{into} | v_{banking})$
$P(u_{crises} | v_{banking})$
$P(u_{as} | v_{banking})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

30

$$p(o|c) = \frac{\exp(u_o . v_c)}{\sum_{w \in Vocabulary} \exp(u_w . v_c)}$$



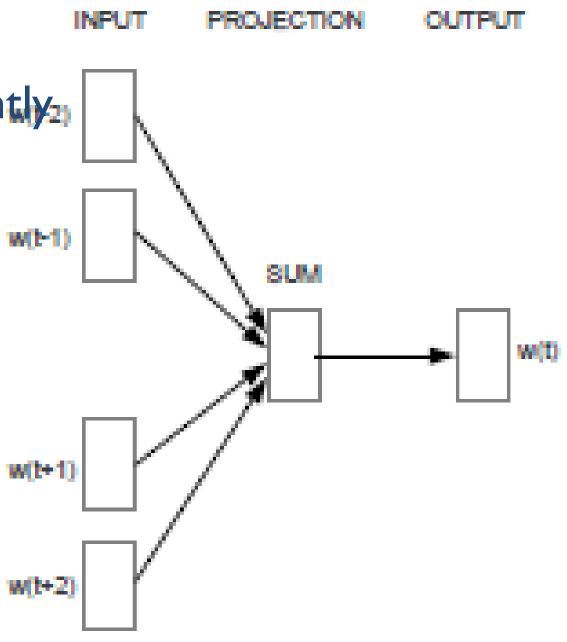
Word embeddings projections on 2D space: words with similar contexts are close in the embedding space

Learning word vector representations
(Mikolov et al. 2013a, 2013b)

- CBOW model
  - Task
    - Predict the midle word of a sequence of words
  - Input and output word representations are learned jointly
    - (random initialization)
  - The projection layer is linear followed by a sigmoid
  - Word weight vectors in the projection layer

are shared (all the weight vectors are the same)
  - The output layer computes a hierarchical softmax
    - See later
    - This allows computing the output in

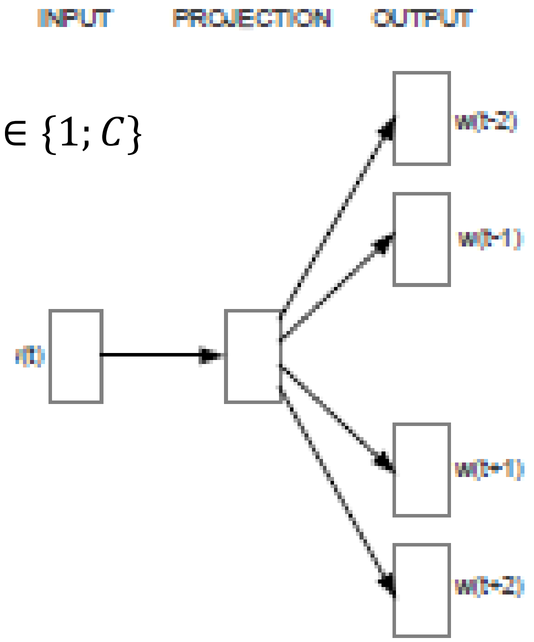
$O(\log_2(dictionary\ size))$ instead of $O(dictionary\ size)$
  - The context is typically 4 words before and 4 after



CBOW

Learning word vector representations (Mikolov et al. 2013a, 2013b)

- Skip Gram model
  - Similar to the CBOW model, except that the context is predicted from the central word instead of the reverse
  - Input and outputs have different representations for the same word
  - The output is computed using a hierarchical softmax classifier
  - Output words are sampled less frequently if they are far from the input word
    - i.e. if the context is $C = 5$ words each side, one selects $R \in \{1; C\}$ and use R words for the output context



Skip-gram

Learning word vector representations (Mikolov et al. 2013a, 2013b)

- Skip gram model
  - Loss average log probability
  - $L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$
    - Where T is the number of words in the whole sequence used for training (roughly number of words in the corpus) and c is the context size
  - $p(w_{out}|w_{in}) = \frac{\exp(\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}})}{\sum_{w=1}^{V}\exp(\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}})}$
    - Where $\boldsymbol{v}_w$ is the learned representation of the w vector (the hidden layer), $\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}}$ is a dot product and V is the vocabulary size
    - Note that computing this softmax function is impractical since it is proportional to the size of the vocabulary
    - In practice, this can be reduced to a complexity proportional to $\log_2 V$ using a binary tree structure for computing the softmax
      - Other alternatives are possible to compute the softmax in a reasonable time
        - In Mikolov 2013: simplified version of negative sampling
        - $l(w_{in}, w_{Out}) = \log\sigma\left(v_{w_{out}}.v_{w_{in}}\right) + \sum_{i=1}^{k}\log\sigma(-v_{w_i}.v_{w_{in}}))$
        - with $\sigma(x) = \frac{1}{1+\exp(-x)}$

Learning word vector representations (Mikolov et al. 2013a, 2013b)

- Properties
  - « analogical reasoning »
  - This model learns analogical relationships between terms in the representation space
    - i.e. term pairs that share similar relations are share a similar geometric transformation in the representation space
    - Example for the relation « capital of »
    - In the vector space
      - Paris – France + Italy = Rome
      - At least approximatively
      - i.e. Rome is the nearest vector to
      - Paris – France + Italy
  - Reasoning via more complex inferences
  - is however difficult:
    - Combination of transformations
    - to infer more complex facts is not effective

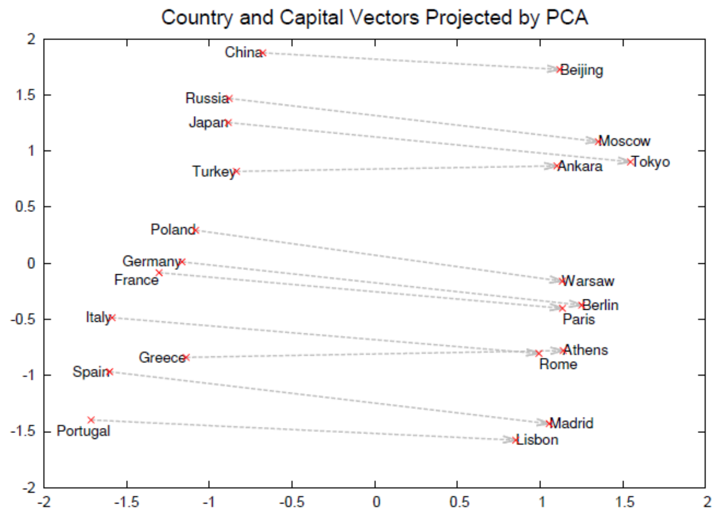


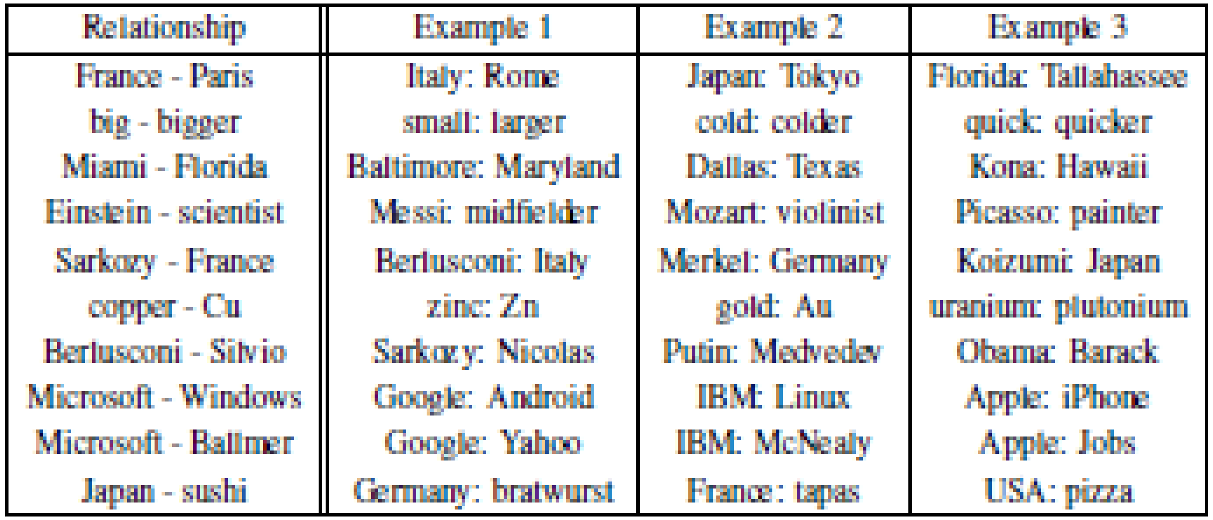
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Figure from Mikolov 2013

Learning word vector representations
(Mikolov et al. 2013a, 2013b)

- Paris – France + Italy = Rome

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).



Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Word2Vec extensions, example of FastText

- After W2V, several similar ideas and extensions have been published
  - Among the more popular are Glove (Pennington 2014) and FastText (Bojanowski 2017)
  - Vector representations learned on large corpora with these methods are made available
  - FastText is a simple extension of the skipgram model in W2V, where n-grams are used as text units instead of words in W2V
    - Consider the word « where » and 3-grams. « where » will be represented as:
      - <wh, whe, her, ere, re>, with « < » and « > » corresponding to special « begin » and « end » characters
      - A vector representation z_i is associated to each n-gram i
      - The word representation is simply the sum of the n-gram representations of the word description
  - Remember $p(w_{out}|w_{in}) = \frac{\exp(\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}})}{\sum_{w=1}^{V}\exp(\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}})}$ in W2V
    - $\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}}$ is replaced by $\sum_{z_i \in \text{ngram}(w_{in})} \boldsymbol{v}_{w_{out}}.\, z_i$
    - And similarly for $\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}}$

Language models - Evaluation

- A classical criterion for evaluating language models is **perplexity**
- $PP(model\ p_{LM}) = \left(\frac{1}{p_{LM}(x^1,\ldots,x^T)}\right)^{1/T} = \left(\prod_{t=1}^{T} \frac{1}{p_{LM}(x^{t+1}|x^t,\ldots,x^1)}\right)^{1/T}$
- Where $p_{\mathrm{LM}}()$ is the probability estimate of the language model
- $PP(model\ p_{LM}) = \left(\prod_{t=1}^{T} \frac{1}{\sum_{i=1}^{|V|} y_i^t \hat{y}_i^t}\right)^{1/T} = \left(\prod_{t=1}^{T} \frac{1}{\hat{y}_{x_{t+1}}^t}\right)^{1/T}$
  - With $y_i^t \in \{0,1\}$ the target code at time t for word i and $\hat{y}_i^t$ the corresponding predicted value. $\hat{y}_{x_{t+1}}^t$ is the prediction for input x_{t+1}
- $PP(model\ p_{LM}) = \exp\left(\frac{1}{T}\sum_{t=1}^{T} -log\hat{y}_{x_{t+1}}^t\right) = \exp(C)$
  - i.e. exponential of the cross-entropy loss C
  - Perplexity $PP(model\ p_{LM})$ is estimated on a test set of sentences
  - Lower is better

Language models - Evaluation

- Interpretations
  - Weighted average branching factor of a language: average nb of words following another word
    - e.g. for random digit sequences, perplexity is 10
  - Perplexity estimates on the WSJ corpus (1.5 M words test corpus, dictionnary size = $20\ k$ words) for n-gram models

Unigram	**Bigram**	**Trigram**
962	170	109

Fig. from XX

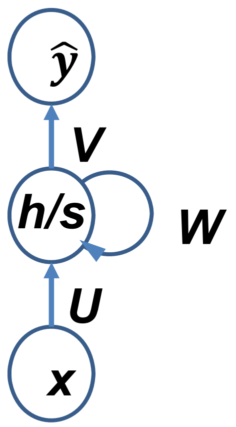
Translation

- NN have been used for a long time in translation systems (as an additional component, e.g. for reranking or as language model)
- Recently translation systems have been proposed that are based on recurrent neural networks with GRU or LSTM units.
  - Initial papers: Sutskever et al. 2014, Cho et al. 2014
- General principle
  - Sentence to sentence translation
  - Use an encoder-decoder architecture
  - Encoding is performed using a RNN on the input sentence (e.g. English)
  - This transforms a variable length sequence into a fixed size vector which encodes the whole sentence
  - Starting with this encoding, another RNN generates the translated sentence (e.g. French)
  - Instead of using a fixed length encoding, current systems use an **attention mechanism**

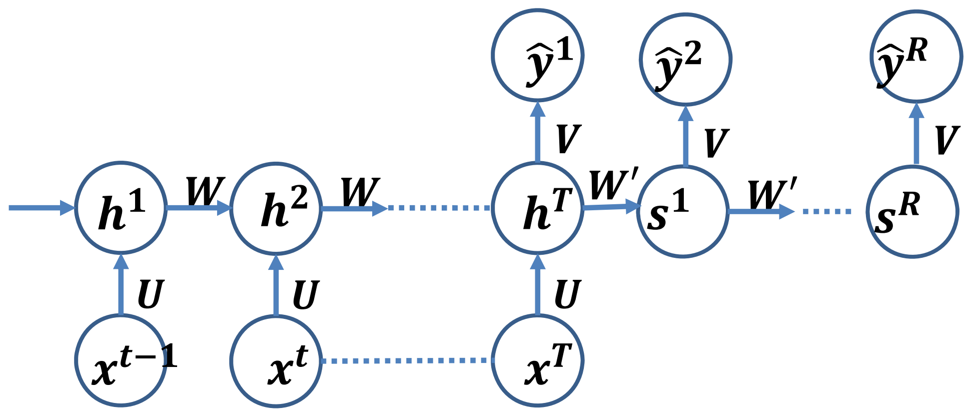
Encoder-Decoder paradigm: example of neural translation – (Cho et al. 2014, Sutskever et al. 2014)

- First attempts for DL Machine Translation with RNNs

Recurrent NN



Unfolded recurrent NN for translation



- Proof of concept, did not match SOTA, several improvements since this first attempt
- Replaced by Attention Models - Transformers

Translation

- Let
  - $x^1, \dots, x^T$ be an input sentence
  - $y^1, \dots y^{T'}$ be an output sentence
  - Note that T and T' are most often different and that the word order in the two sentences is also generally different
- Objective
  - Learn $p(y^1, \dots y^{T'} | x^1, \dots, x^T)$
  - Encoder
    - Reads each symbol of the input sentence sequentially using a RNN
    - After each symbol the state of the RNN is changed according to $\boldsymbol{h}^t = f(\boldsymbol{x}^t, \boldsymbol{h}^{t-1})$
    - After reading the sentence, the final state is $\boldsymbol{h}^T = \boldsymbol{v}$
  - Decoder
    - Generates the output sequence by predicting the next symbol y^t given $\boldsymbol{s}^{t-1}, y^{t-1}$ and the vector $\boldsymbol{v}$
      - ☐ $\boldsymbol{s}^t = f(\boldsymbol{y}^{t-1}, \boldsymbol{s}^{t-1}, \mathbf{v})$
      - ☐ $p(y^t | y^{t-1}, \dots y^1, \boldsymbol{v}) = g(y^{t-1}, \boldsymbol{s}^t, \mathbf{v})$
- Training: cross-entropy loss
  - $\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\boldsymbol{y}_s^n | \boldsymbol{x}_s^n)$, where $\boldsymbol{x}_s^n$ and $\boldsymbol{y}_s^n$ are sentences and p_θ is the translation model, N is the number of sentences

Translation

- Typical architecture
  - RNN with 1000 hidden cells
  - Word embeddings of dimension between 100 and 1000
  - Softmax at the output for computing the word probabilities
  - Of the order of 100 M parameters

Google Neural Machine Translation System

(Wu et al 2016)
https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

- General Architecture

Encoder: 8 stacked LSTM RNN + residual connections

Decoder: 8 stacked LSTM RNN + residual connections + Softmax output layer

Attention mechanism

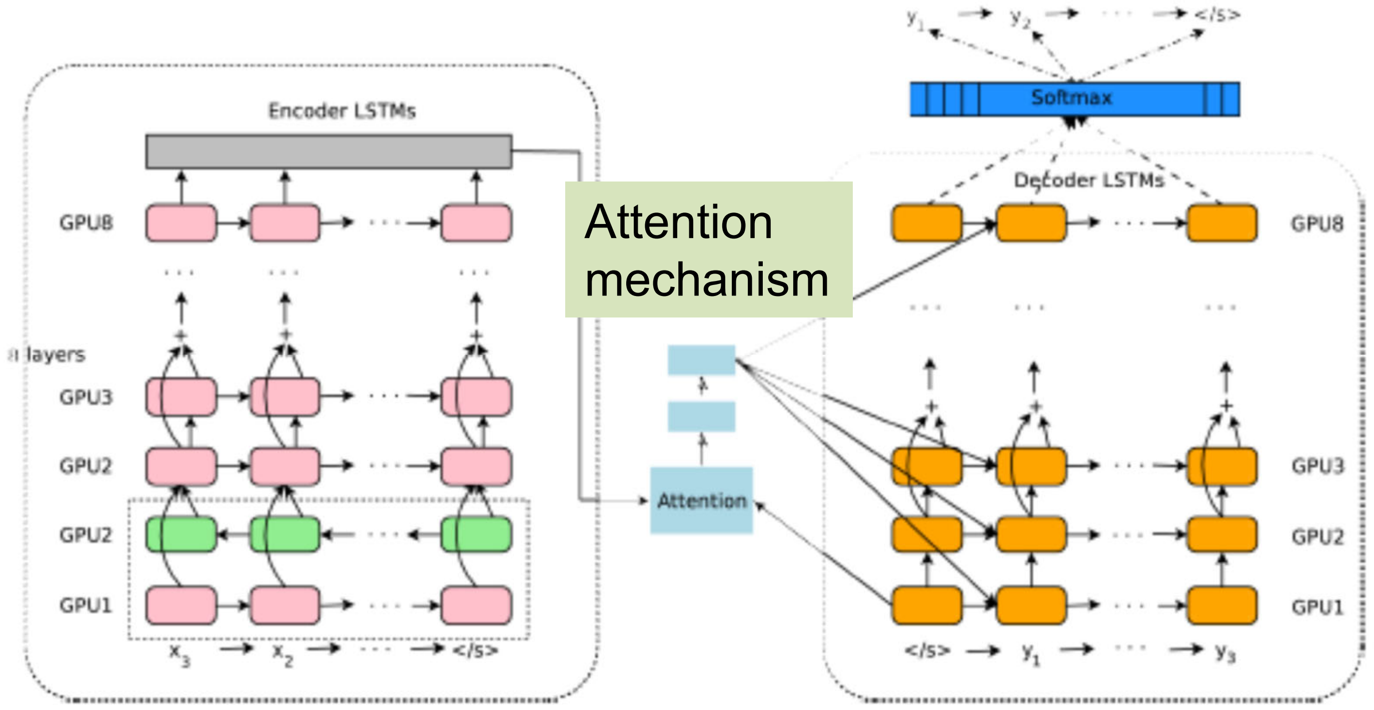


Figure from Wu et al. 2016

Neural image caption generator (Vinyals et al. 2015)

- Objective
  - Learn a textual description of an image
    - i.e. using an image as input, generate a sentence that describes the objects and their relation!
- Model
  - Inspired by a translation approach but the input is an image
    - Use a RNN to generate the textual description, word by word, provided a learned description of an image via a deep CNN

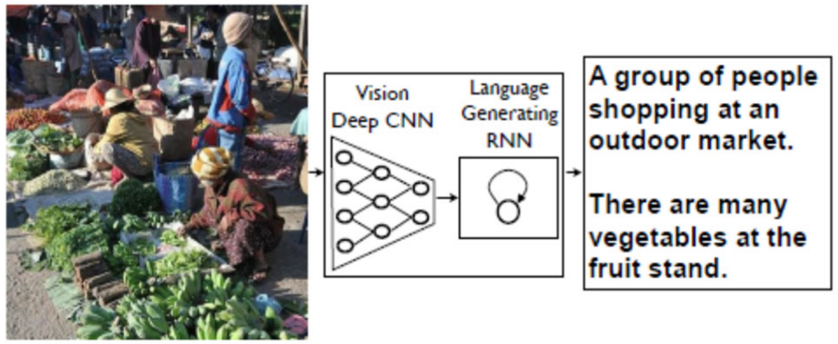

Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Figure 1. NIC, our model, is based end-to-end on a neural network consisting of a vision CNN followed by a language generating RNN. It generates complete sentences in natural language from an input image, as shown on the example above.

Neural image caption generator (Vinyals et al. 2015)

- Loss criterion
  - $\max_{\theta} \sum_{I,S} \log p(S|I;\theta)$
    - Where (I, S) is an associated couple (Image, Sentence)
    - Notations correspond to the figure
  - $\log p(S|I;\theta) = \sum_{t=1}^{N} \log p(S_t|I, S_0, \dots, S_{t-1})$
  - $p(S_t|I, S_0, \dots, S_{t-1})$ is modeled with a RNN with $S_0, \dots, S_{t-1}$ encoded into the hidden state h_t of the RNN
  - Here $\boldsymbol{s}^{t+1} = f(\boldsymbol{s}^t, x_t)$ is modelled using a RNN with LSTM cells
  - For encoding the image, a CNN is used

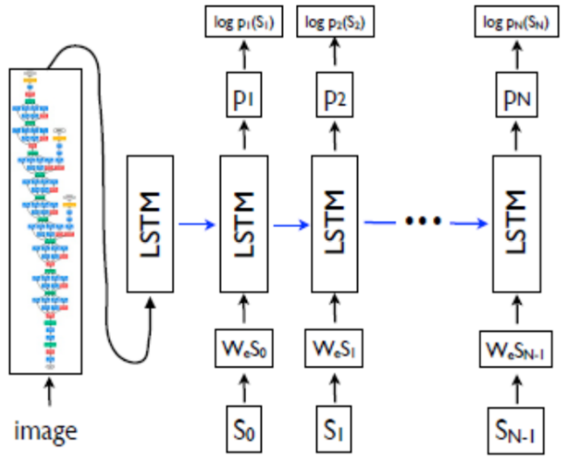


Figure 3. LSTM model combined with a CNN image embedder (as defined in [30]) and word embeddings. The unrolled connec-

Neural image caption generator (Vinyals et al. 2015)

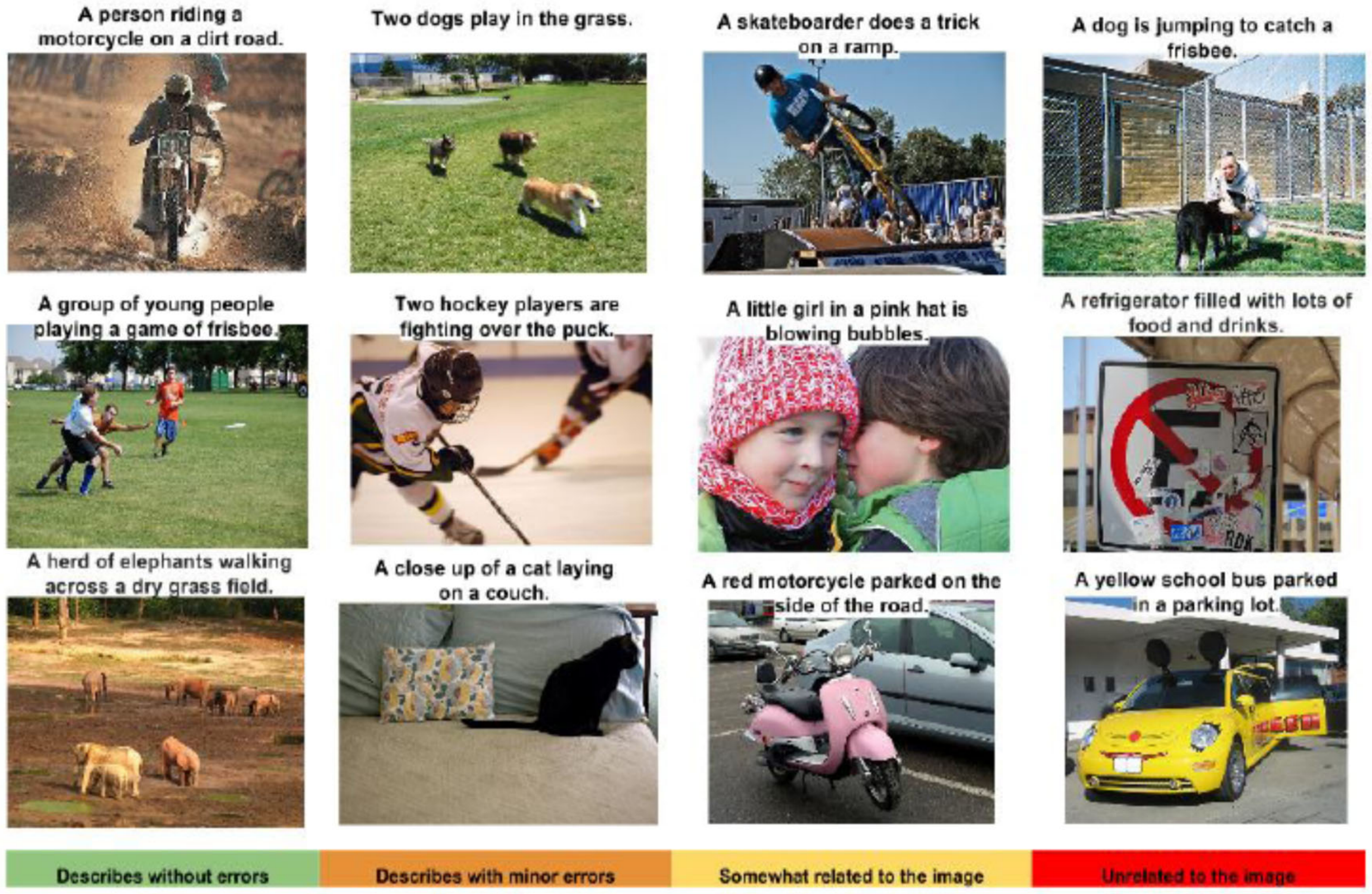

A person riding a motorcycle on a dirt road.
Two dogs play in the grass.
A skateboarder does a trick on a ramp.
A dog is jumping to catch a frisbee.
A group of young people playing a game of frisbee.
Two hockey players are fighting over the puck.
A little girl in a pink hat is blowing bubbles.
A refrigerator filled with lots of food and drinks.
A herd of elephants walking across a dry grass field.
A close up of a cat laying on a couch.
A red motorcycle parked on the side of the road.
A yellow school bus parked in a parking lot.
Describes without errors
Describes with minor errors
Somewhat related to the image
Unrelated to the image

Figure 5. A selection of evaluation results, grouped by human rating.