

Plain recurrent models

Martin Tutek

2020.

Introduction

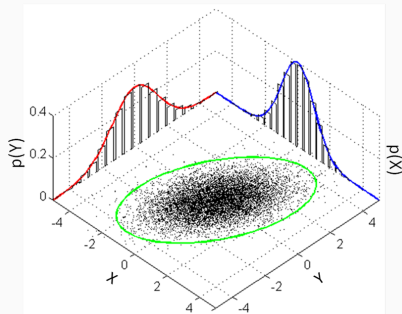
Previously on Deep learning ...

... we have considered to model the data with:

- fixed dimensionality
- explicit interdependence



images from CIFAR-10

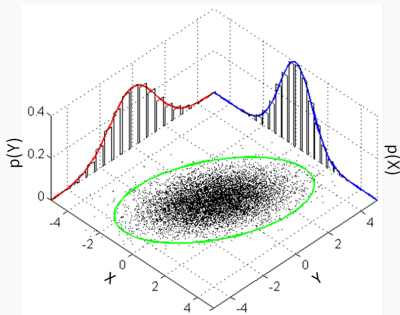


multivariate normal distribution

Previously on Deep learning ...

... we have considered to model the data with:

- explicit interdependence
- fixed dimensionality



multivariate normal distribution



images from CIFAR-10

Fixed input dimensionality:

- All samples from a multivariate normal distribution (cf. Lab #1) have the same dimensionality: $x_i \in \mathbb{R}^d, \forall i$
- Each image from CIFAR-10 has the same dimensionality 32x32x3 (WxHxC)
- **How to assert:** cropping, padding, pooling, scaling (interpolation)

Previously on Deep learning ...

Fixed input dimensionality:

- All samples from a multivariate normal distribution (cf. Lab #1) have the same dimensionality: $x_i \in \mathbb{R}^d, \forall i$
- Each image from CIFAR-10 has the same dimensionality 32x32x3 (WxHxC)
- **How to assert:** cropping, padding, pooling, scaling (interpolation)

Explicit inter-dependence:

- In multi-variate normal distribution inter-dependence is defined through the distribution parameter Σ
- Neighbouring image pixels have similar RGB values
- **How to assert:** use convolutional filters (and positional encodings in transformers)

Natural language processing

Translate the following sentence into English:

Duboko učenje je super. → *Deep learning is great.*

Natural language processing

Translate the following sentence into English:

Duboko učenje je super. → *Deep learning is great.*

The problem requires the following actions:

- segment the sentence (sequence of words/letters) into meaningful components
- "understand" that expression "duboko učenje" does not refer to learning at great depths (eg. at the bottom of the Atlantic); instead, it refers to a branch of machine learning
- understand the meaning in Croatian without requiring a specification of the source language
- generate the English translation with the same meaning.

Textual data is strange:

- **No** fixed dimensions (sentence lengths can vary)
- **No** explicit inter-dependence patterns

Natural language processing

Translate the following sentence into Vietnamese:

最寄りのインターネットカフェはどこですか

→ *Chờ internet ở đâu*

→ *Where is the nearest internet shop*

Natural language processing

Translate the following sentence into Vietnamese:

最寄りのインターネットカフェはどこですか

→ *Chờ internet ở đâu*

→ *Where is the nearest internet shop*

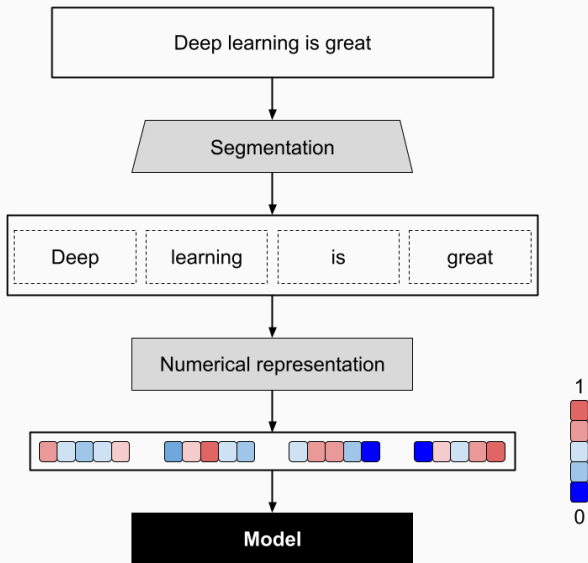
For computers, the language looks about the same as Japanese (but only for us who do not speak Japanese).

The meaning of words and similarity of language units are unknown.

Before processing the text with machine learning algorithms, we can **at least** initialize them with vectorized word representations.

Data representation

Natural language processing



This lecture

Text segmentation:

- we can segment text in **words**
- alternatives: letters, subwords¹

Numerical representation:

- associate each word with a dense vector
- alternatives: one-hot representations (*bag-of-words*), multiprototype representations

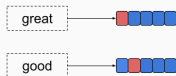
This lecture

Text segmentation:

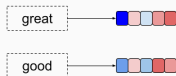
- we can segment text in **words**
- alternatives: letters, subwords¹

Numerical representation:

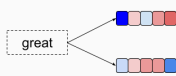
- associate each word with a dense vector
- alternatives: one-hot representations (*bag-of-words*), multiprototype representations



(a)



(b)

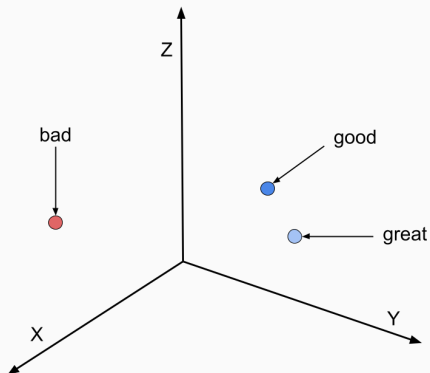


(c)

One-hot (a), dense (b), and multiprototype representations (c).

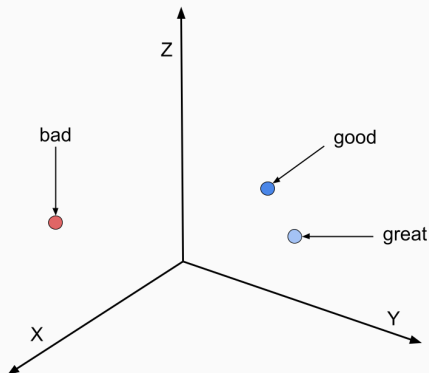
Dense word representations

Distance in representation space should correspond to (semantic and syntactic) similarity between words



Dense word representations

Distance in representation space should correspond to (semantic and syntactic) similarity between words



- what is the distance between one-hot word representations?
- how could we compute the distance for multiprototype representations?

Dense word representations

In practice, the dimensionality of the word representation space is considerably larger than in the figure.

We must rely on intuitions from lower-dimensional spaces although they may be deceptive.

- If you are not used to thinking about hyper-planes in high-dimensional spaces, now is the time to learn.
- To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say "fourteen" to yourself very loudly. **Everyone does it.**
 - But remember that going from 13-D to 14-D creates as much extra complexity as going from 2-D to 3-D.

Lectures, "Neural Networks for Machine Learning", Geoffrey Hinton

Word representations

We may initialize the word representations by pre-training on some other task

- Many of these tasks predict words from local contexts on large corpora of real text (Wikipedia, Common crawl²)
- Models: word2vec³ (CBOW, Skip-gram), GloVe⁴, FastText⁵
- The training optimizes the embedding matrix that contains a vector representation for each word

In this course, we will not consider separate learning of word representations

- instead we shall either use random initializations or rely on pre-trained embeddings

²<https://commoncrawl.org/>

³<https://en.wikipedia.org/wiki/Word2vec>

⁴<https://nlp.stanford.edu/projects/glove/>

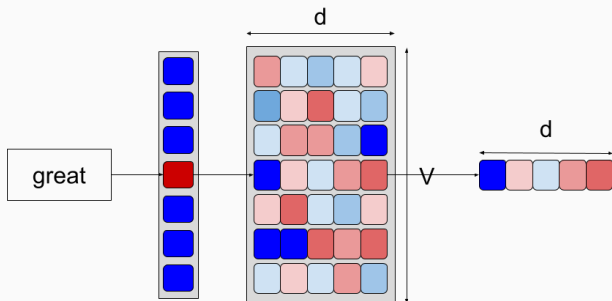
⁵<https://fasttext.cc/>

Embedding matrices in practice

We recover word embeddings by multiplying the embedding matrix $E \in \mathbb{R}^{d \times V}$ with the one-hot vector of the word:

$$\mathbf{e}_i = E \mathbf{x}_i \quad \mathbf{x}_i = \underbrace{[0, \dots, 0, \overset{i}{1}, 0, \dots, 0]^T}_V$$

Further slides will denote dense embeddings \mathbf{e}_i as \mathbf{x}_i !



Word representations: hyper-parameters

Vocabulary size: V

- number of unique training words
- it depends on available memory and word relevance⁶
- in practice we often take top V most common words from the training corpus
- the remaining words can be i) filtered out or ii) replaced with the token $\langle UNK \rangle$

Word embedding dimension: d

- we often use $d = 300$ as a sensible default⁷
- in practice the choice will also depend on available memory and pre-trained embeddings.

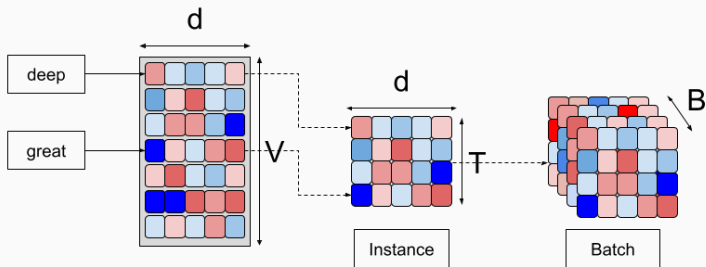
⁶rare words and words with low information content are better omitted in order to avoid performance hit

⁷other choices may lead to under-fitting and over-fitting

Representing text

We form text representations (sentences, paragraphs, etc) by concatenating word embeddings

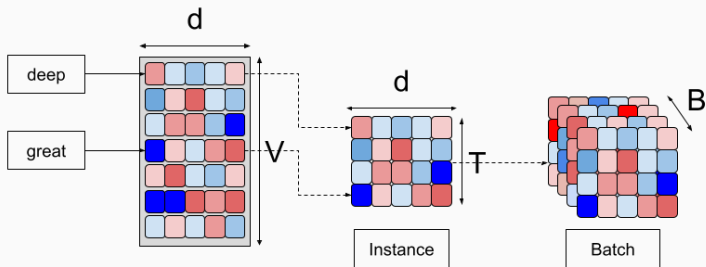
We denote the text length as the number of words T (*temporal dimension*)



Representing text

We form text representations (sentences, paragraphs, etc) by concatenating word embeddings

We denote the text length as the number of words T (*temporal dimension*)



This approach has to address several problems. Can you point them out?

Representing text

Problem: temporal dimension **varies:**

- sentences of a learning batch will have different lengths
- this can be addressed by zero-padding and truncation

Goal 1: model processes each word of the sentence in the same manner

Representing text

Problem: temporal dimension **varies:**

- sentences of a learning batch will have different lengths
- this can be addressed by zero-padding and truncation

Goal 1: model processes each word of the sentence in the same manner

- this limits the model dimensionality and reduces overfitting

Goal 2: model is sensitive to the word order

Representing text

Problem: temporal dimension **varies:**

- sentences of a learning batch will have different lengths
- this can be addressed by zero-padding and truncation

Goal 1: model processes each word of the sentence in the same manner

- this limits the model dimensionality and reduces overfitting

Goal 2: model is sensitive to the word order

- otherwise, we would not be able to distinguish sentences with same words:
 - *Dog eats cat* i *Cat eats dog*

We achieve both goals with **recurrent models**

- Let us first look into some alternatives for addressing the variable temporal dimension.

Representing text: alternatives

Temporal aggregation by pooling

- express the text representation by pooling word embeddings
(*eng. mean/average pooling*)

$$r_{\mu}(\text{text}) = \frac{1}{T} \sum_t^T x^{(t)} \quad (1)$$

- such aggregation loses positional information but it does not require any **parameters**
- variants of this approach: sum pooling, **weighted pooling**

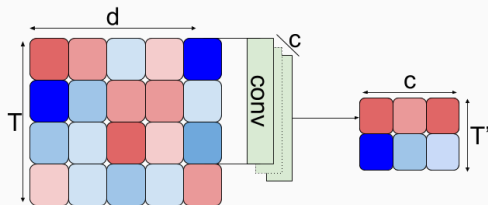
$$r_w(\text{text}) = \sum_t^T w_t x^{(t)} \quad w_t = f(x^{(t)}, \text{text}) \quad (2)$$

- weighted pooling (2) multiplies word embeddings with a factor that depends on the text and the word (eg. TF-IDF)

Representing text: alternatives

1D convolution

- convolutions retain information about the word ordering



- this still requires some form of pooling in order to preclude variable output
- simple convolutional architecture misses the *global* context
 - this can be alleviated by increasing the depth...
 - ... or by harnessing dilated convolutions or spatial pooling

Representing text: summary

Natural language processing (NLP) confronts unexpected problems

- segment the text into words, **subwords** or letters
- detect and correct typographic errors
- many ways to put the same thoughts into writing

Usual assumptions:

- input text segmented into words
- word embeddings can be pre-trained or randomly initialized
- each word assigned to exactly *one* dense embedding

Approaches to address variable input:

- 1D convolutions and pooling layers
- recurrent modelling

Plain recurrent models

Recurrent models: motivation

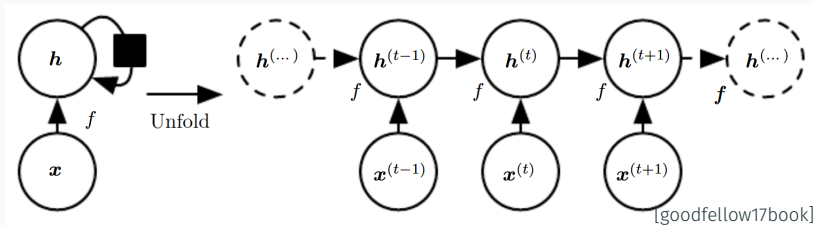
We are looking for a suitable model for sequential data:

- accommodate arbitrary sequence lengths
- number of parameters does not depend on the length
- model is aware of data ordering

Inspired by **dynamic systems**:

- **hidden state** $h^{(t)}$ uniformly updated across all inputs $x^{(t)}$:

$$h^{(t)} = f(x^{(t)}, h^{(t-1)})$$



Recurrent models: formulation

Recurrent model f is invariant to time (location within the sequence):

$$h^{(t)} = f(x^{(t)}, h^{(t-1)})$$

A simple recurrent model involves one non-linearity for each input:

$$h^{(t)} = g(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

- W_{hh} , W_{xh} , and b_h - parameters of the recurrent affine transform
- $a^{(t)}$ - recurrent linear score
- g - non-linearity (sigmoid, tanh,...)

Recurrent models: intuition

The state $h^{(t)}$ represents the **observed** portion of the sequence

- We hope that it encodes semantics if the model is well-learned

The matrix W_{xh} projects the input into the representation space

- We hope that this projection removes useless information from the input

The matrix W_{hh} models the evolution of the state:

- it models the passage of time and hopefully removes the excess information from the state

$$h^{(t)} = g(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

The dimensionalities of the state and the input **may differ**:

$$h \in \mathbb{R}^h \quad x \in \mathbb{R}^d$$

Recurrent models: izlazni sloj

The state $h^{(t)}$ represents all observed information so far

- Besides semantics, the state must also remember complementary information
 - eg. ambiguous words or references to people or locations
- some of these pieces of information may be **irrelevant** for prediction!

The *output layer* projects the state into the output space:

$$o^{(t)} = W_{hy}h^{(t)} + b_o \quad (3)$$

- The vector $o^{(t)}$ contains logits with predictions at time t
- this steps *filters* unimportant information
- **[!!]** Pytorch RNN cell does not contain the output projection -- you have to add it explicitly.

Recurrent models: formulation

A simple recurrent model is defined by the following equations:

1. Update of the hidden state

$$h^{(t)} = \tanh(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

2. Projection into output space

$$o^{(t)} = W_{hy}h^{(t)} + b_o$$

In practice, we often activate the state with tanh activation:

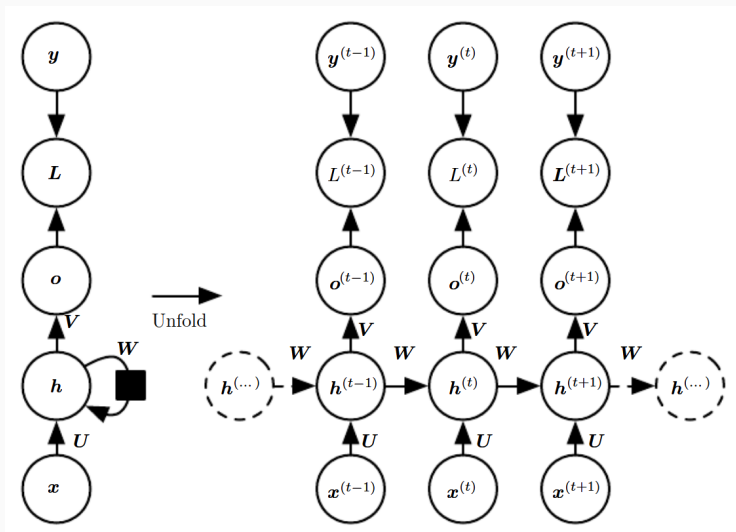
- other options are the sigmoid and the hinge.

Parameter dimensions: $W_{hh} \in \mathbb{R}^{h \times h}$, $W_{xh} \in \mathbb{R}^{h \times d}$, $W_{hy} \in \mathbb{R}^{y \times h}$

- d and y denote the input and the output dimensionality

[!!] Textbook notation: $W := W_{hh}$, $U := W_{xh}$, $V := W_{hy}$, $b := b_h$, $c := b_o$

Recurrent models: visualization



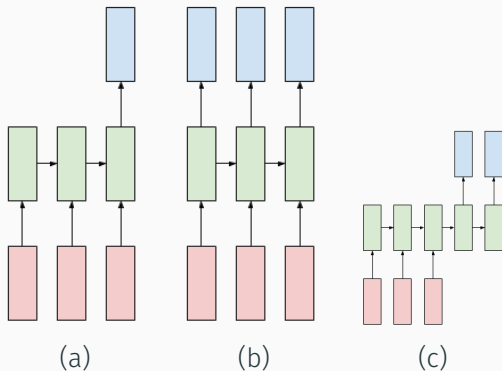
Unfolded recurrent model with inputs $x^{(t)}$, loss $L^{(t)}$ and outputs $o^{(t)}$.

Adapting the model to the task

A recurrent model can generate output at each time step -- but do we need that?

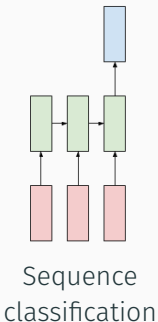
- output configuration of our models depend on particular task!

Try to think of problems which require only one output **(a)** output in each input **(b)** variable number of outputs, but more than one **(c)**



NLP Tasks

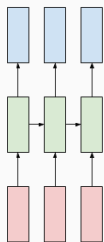
Sequence classification: basic NLP problem.



- the task is to determine one target variable for the entire input sequence
- different classification tasks: sentiment analysis, document categorization, determining a music genre, ...

The output layer receives only **the last** latent state.

Sequence "labeling: requires prediction for each input element.



Sequence "labeling"

- the task is to determine a sequence of targets where each output corresponds to exactly one input
- different segmentation tasks: part-of-speech tagging, named entity recognition, extractive text summarization, segmentation of text in video, ...

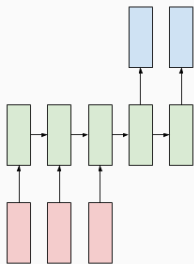
Predictions do not have to be synchronized with the inputs (there can exist a small temporal offset), but this is often the case in practice.

We use quotes since the word *labeling* suggests human annotation

- a better term: dense prediction

NLP Tasks

Sequence-to-sequence (*seq2seq*) tasks involve more general forms of generating sequences from sequences.



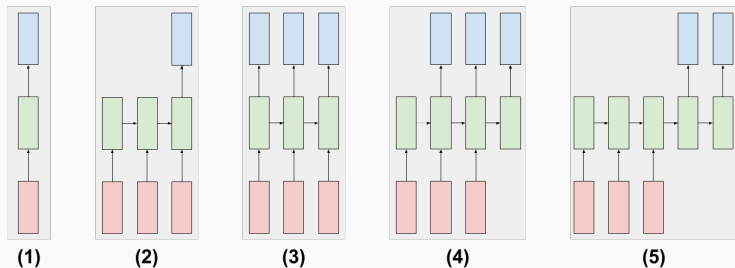
Sequence-to-sequence

- the task is to determine the target sequence of unknown length given input
- machine translation, abstractive text summarization, ...

Consider potential applications of this task:

- how to start the translation?
- should we always generate the most probable word?
- how to decide when to conclude the translation?

NLP Tasks: overview



Directions for problem classification:

- whenever the number of outputs equals the number of inputs, the problem corresponds to dense prediction regardless of whether the offset exists **(4)** or not **(3)**
- sequence-to-sequence problems **(5)** are often decomposed into text summarization (where we do not perform any prediction) and sequence generation (which generates the entire output sequence)

Analysis: Sequence classification

Sentiment analysis

Classification problem: to determine the sentiment of the input text

- binary formulation: positive/negative
- categoric formulation: star rating [1,10]
 - this could also be formulated as regression
- example datasets: IMDB⁸, YELP⁹
- a positive example from IMDB:

... was the story that blew me away. hurray for Takahisa

Our analysis assumes binary fomrulation and neglects (considerable) pre-processing issues.

⁸<https://ai.stanford.edu/~amaas/data/sentiment/>

⁹<https://www.yelp.com/dataset/>

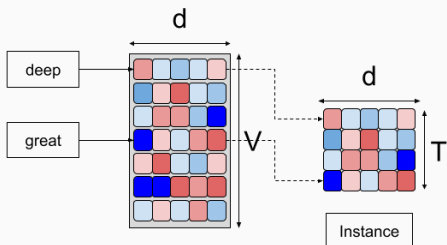
Sentiment analysis: preliminaries

Target variable:

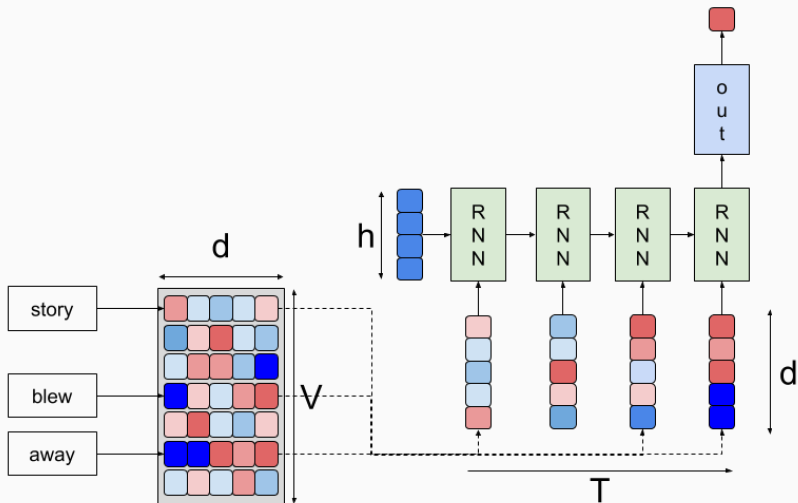
- Sentiment variables: {positive, negative}
- We represent them as indices: {0, 1}

Input variables:

- we must select the size of the input vocabulary
- we map words onto vectors of the embedding matrix



Sentiment analysis: top-level sketch



Sentiment analysis: pseudocode

Algoritam 1: Plain RNN for sequence classification

Single output RNN ($X, W_{hh}, W_{xh}, W_{hy}, b_h, b_o$)

inputs: A sequence of vectors X , parameters $W_{...}, b_{...}$

output: A logit \hat{y}

$h_t \leftarrow \text{zero_init};$

foreach x **in** X **do**

$a_t \leftarrow W_{hh}h_t + W_{xh}x + b_h;$

$h_t \leftarrow \tanh(a_t);$

end

$o \leftarrow W_{hy}h_t + b_y;$

return $o;$

We can adapt this algorithm towards dense prediction by generating elements of the output vector within the loop.

Analysis: dense prediction

Part-of-speech tagging

Dense prediction along the sequence: predict a POS class for each word

- multi-class formulation: noun, adjective, verb, ...

Datasets: Penn Treebank (PTB) (behind paywall), Universal Dependencies¹⁰ (UD)

- PTB taxonomy: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Training example, U-POS labels and POS labels:

```
I    do  n't  think  it  matters.
PRON AUX PART VERB  PRON VERB  .
PRP  VBP RP   VB    PRP  VBZ   .
```

¹⁰<https://universaldependencies.org/>

Part-of-speech tagging: preliminaries

Target variables:

- part-of-speech class: $\{ADP, PROPN, \dots\}$
- we leverage class indices: $\{0, \dots, Y\}$

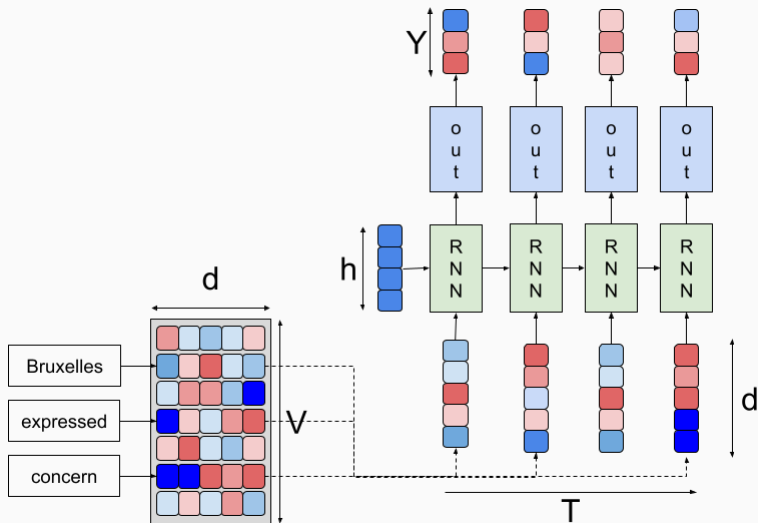
Input variables:

- we must choose the size of the vocabulary (!!)
- we map words onto vectors of the embedding matrix

Very similar to sequence classification:

- important difference: target dimensionality
- we have to use dense loss instead of sentence-wide loss

Part-of-speech tagging: top-level sketch



Algoritam 2: Pseudokod RNNa for multi-class dense prediction

Dense sequence prediction RNN ($X, W_{hh}, W_{xh}, W_{hy}, b_h, b_o$)

inputs: A sequence of vectors X , parameters $W_{...}, b_{...}$

output: A sequence of logits \hat{y}

$h_t \leftarrow \text{zero_init};$

$o \leftarrow [];$

foreach x *in* X **do**

$a_t \leftarrow W_{hh}h_t + W_{xh}x + b_h;$

$h_t \leftarrow \tanh(a_t);$

$o_t \leftarrow W_{hy}h_t + b_y;$

$o.append(o_t)$

end

return $o;$

Summary

Recurrent model acts as a dynamical system:

- input information is iteratively built into the latent state
- the new latent state is iteratively updated with respect to the old state and current input

The state (most often) does not contain predictions:

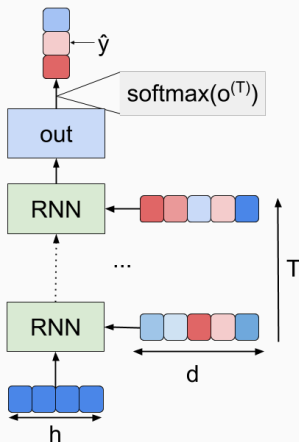
- predictions are formed by projecting the state
- the actual task determines the number of projections (only one, per-input or variable)
- parameters of the output layer are also **shared**

The three main tasks are **sequence classification**, **dense prediction** and **sequence-to-sequence translation**

Training recurrent models

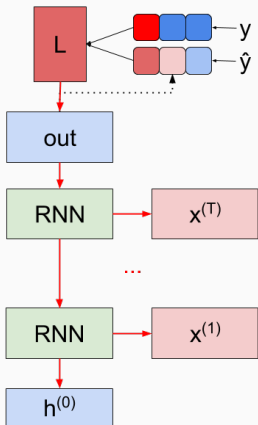
Introduction

Recurrent models can be unfolded into feed-forward models with distinct inputs and shared parameters:

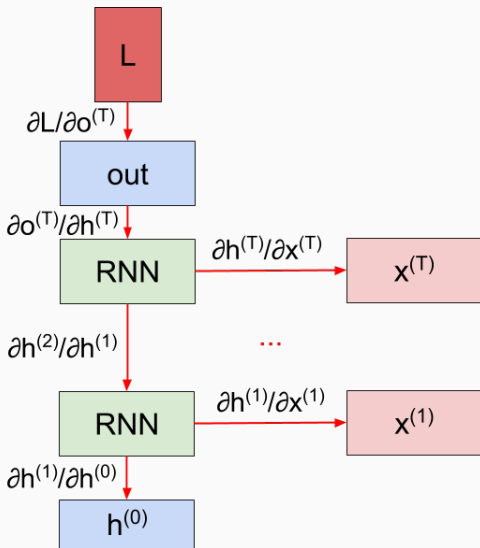


Example: sequence classification

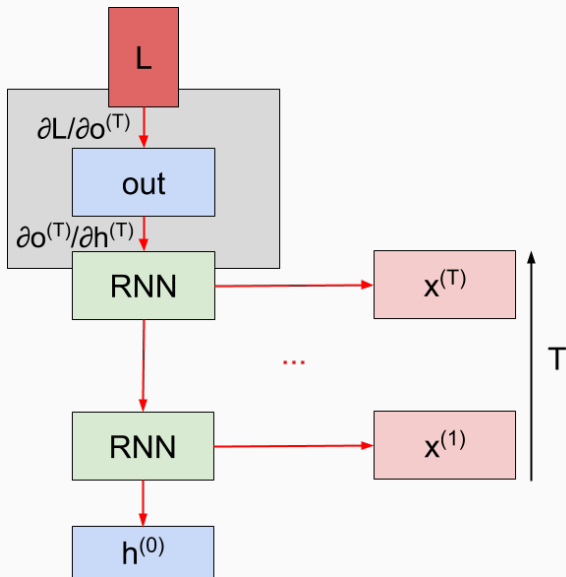
A model for sequence-wide prediction can be trained with standard backprop:



Example: sequence classification



RNN: backprop through the output layer



RNN: backprop through the output layer (2)

Equations of the output projection:

$$o^{(t)} = W_{hy}h^{(t)} + b_y$$
$$\frac{\partial \mathbb{L}}{\partial o^{(t)}} = \underbrace{\text{softmax}(\mathbf{o}^{(t)})}_{\hat{y}} - y$$

Here are the gradients wrt parameters of the output layer W_{hy} and b_y :

$$\frac{\partial \mathbb{L}}{\partial W_{hy}} = \frac{\partial \mathbb{L}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial W_{hy}} = (\hat{y} - y) \cdot h^{(t)}$$
$$\frac{\partial \mathbb{L}}{\partial b_y} = \frac{\partial \mathbb{L}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial b_y} = \hat{y} - y$$

RNN: backprop through the state update

The gradient of the loss wrt the hidden state propagates towards earlier timesteps:

$$h^{(t)} = f(h^{(t-1)}, \dots, h^{(0)})$$

Backprop at time $t = T$ (last timestep)

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}}{\partial o^{(t)}} W_{hy}^T$$

Za $t < T$ (preostale vremenske korake)

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(T-1)}} \cdots \frac{\partial h^{(t+1)}}{\partial h^{(t)}}$$

In general, two kinds of gradients may arrive to a hidden state:

1. gradients from the **corresponding** prediction (top)
2. gradients from the **future** hidden states (bottom)

RNN: backprop through the state update (3)

Equation of the RNN state update (reminder):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

We require the derivative of the hyperbolic tangent:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x)$$

Hyperbolic tangent is a sigmoid "stretched" over $[-1, 1]$

$$\tanh(x) = 2\sigma(2x) - 1$$

RNN: backprop through the state update (4)

Equation of the RNN state update (reminder):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Gradient wrt the pre-activation $a^{(t)}$:

$$\frac{\partial \mathbb{L}}{\partial a^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial a^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(t)}} \frac{\partial \tanh}{\partial a^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(t)}} (1 - h^{(t)^2})$$

Gradients wrt the RNN cell parameters (W_{hh} , W_{xh} , b_h):

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{hh}} = \left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(h^{(t-1)} \right)^\top$$

$$\frac{\partial \mathbb{L}}{\partial W_{xh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{xh}} = \left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(x^{(t)} \right)^\top$$

$$\frac{\partial \mathbb{L}}{\partial b_h} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial b_h} = \frac{\partial \mathbb{L}}{\partial a^{(t)}}$$

RNN: gradients wrt inputs

Equation of the RNN state update (reminder):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

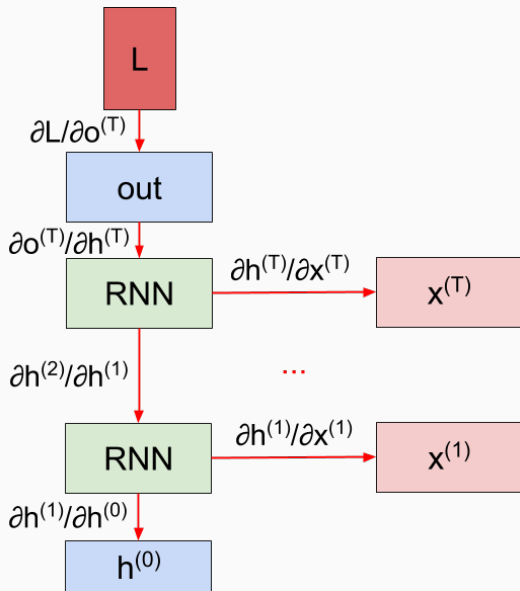
1. recover the gradient wrt the previous hidden state:

$$\frac{\partial \mathbb{L}}{\partial h^{(t-1)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial h^{(t-1)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} W_{hh}$$

2. recover the gradient wrt input (think why!):

$$\frac{\partial \mathbb{L}}{\partial x^{(t)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial x^{(t)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} W_{xh}$$

RNN: backprop through sequence classification, summary



RNN: backprop through sequence classification, summary

Gradients of the loss arrive to the state update parameters through each hidden state:

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{hh}} = \underbrace{\left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(h^{(t-1)} \right)^\top}_{\forall t \in \{1, 2, \dots, T\}}$$

These gradient **accumulate** through time:

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial \mathbb{L}}{\partial W_{hh}} = \sum_{t=1}^T \left(\frac{\partial \mathbb{L}^{(t)}}{\partial a^{(t)}} \right)^\top \left(h^{(t-1)} \right)^\top$$

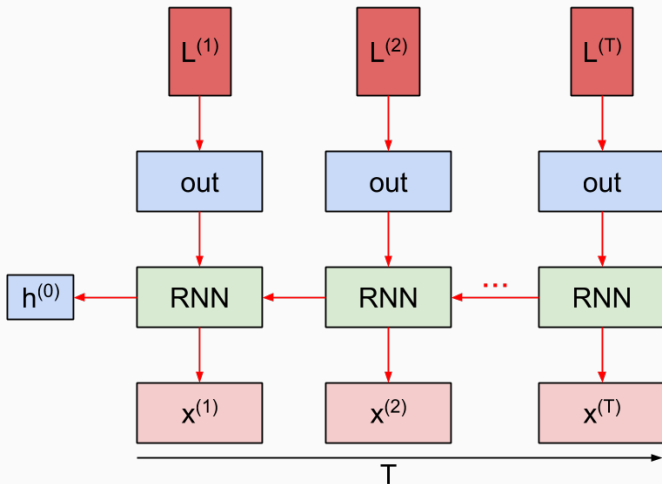
The optimization step uses the accumulated gradient

Motivation for recovering gradients wrt inputs ($x^{(t)}$):

1. learning (fine-tuning) the word embedding matrix
2. the input may correspond to the output of another recurrent

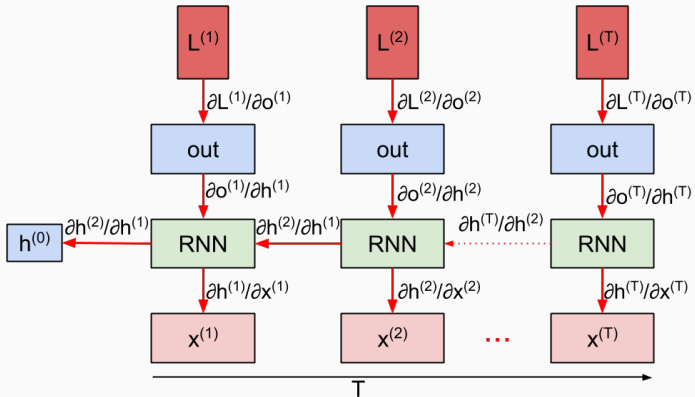
Dense prediction loss

Dense prediction involves the loss in **each** time step

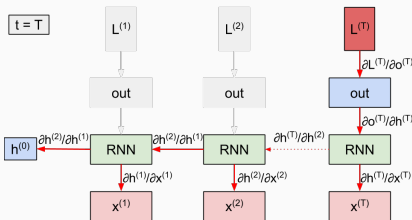
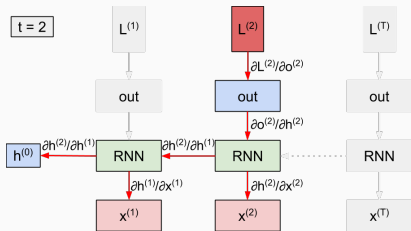
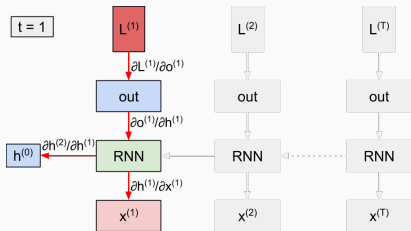


Backprop through dense prediction

In this case, backprop has to accumulate the loss terms from all future time steps:



Backprop through dense prediction: intuition

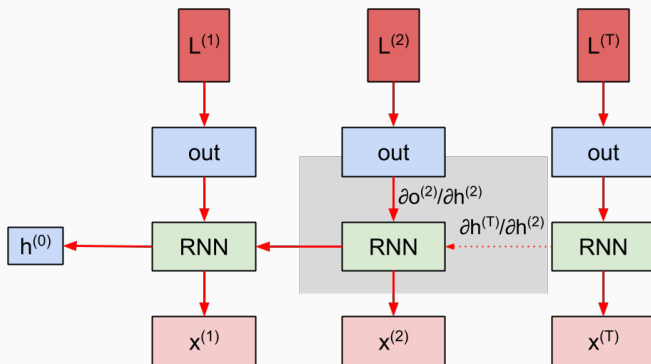


In practice (pytorch), we perform multiple backward passes (one at each time-step) before each model update

Each RNN cell receives gradients from the above (loss at the current time-step) and the right (loss at the future time-steps)

Backprop through dense prediction: gradients

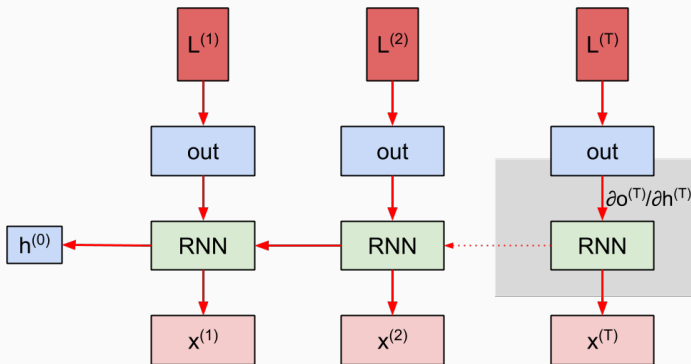
State update is affected both by the current loss (vertical dependency) and by all future losses (horizontal dependency):



Backprop through dense prediction: gradients

State update is affected both by the current loss (vertical dependency) and by all future losses (horizontal dependency):

- ... except at the last time-step where we do not have any contribution from the future



Backprop through dense prediction: gradients

The loss is **factorized** into temporal components $\mathbb{L} = \sum_t \mathbb{L}^{(t)}$

- we require adaptations wrt sequence-wide case **only** for $\frac{\partial \mathbb{L}}{\partial h^{(t)}}$

For $t = T$ (last time-step) we require **no adaptation**:

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial o^{(t)}} W_{hy}$$

For $t < T$ (the remaining time-steps):

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \underbrace{\frac{\partial \mathbb{L}^{(t)}}{\partial h^{(t)}}}_{\text{current loss}} + \underbrace{\frac{\partial \mathbb{L}^{(t^* > t)}}{\partial h^{(t)}}}_{\text{future loss}}$$

$$\frac{\partial \mathbb{L}^{(t^* > t)}}{\partial h^{(t)}} = \sum_{t^* > t}^T \frac{\partial \mathbb{L}^{(t^*)}}{\partial h^{(t)}} = \sum_{t^* > t}^T \frac{\partial \mathbb{L}^{(t^*)}}{\partial h^{(t^*)}} \cdots \frac{\partial h^{(t+1)}}{\partial h^{(t)}}$$

Backprop through dense prediction: summary

Backprop through recurrent models is often referred to as backpropagation through time (BPTT):

- conceptually, BPTT is equivalent to backward propagation
- dense prediction can be interpreted as a jointly optimized multi-task classification problem
- RNN cell can be viewed as a layer of a plain recurrent model

Due to shared parameters and gradient accumulation, training can be **unstable**

- high incidence of vanishing and exploding gradients
- the presented basic formulation is seldom used in practice

Practical advice

Dimensionality of the latent state h :

- as large as possible, other hiperparams may be more important
- model depth may also be more important (next lecture)

Sequence length T :

- plain RNNs have **very** short-term memory ($T \approx 20$ for text)
- we often clip samples in order to enforce some maximal length
- the threshold depends on the data and the task

Sensible defaults: Adam, gradient clipping (even with LSTMs)

RNN alternatives **exist**:

- SVM for simple text classification:
<https://github.com/mesnilgr/nbsvm>
- **attention** may be all you need
- sound: (dilated) convolutions, attention

Questions?

Reading list (**besides the textbook**):

1. Peter's notes: Implementing a NN / RNN from scratch
<http://peterroelants.github.io/>
2. Andrej Karpathy: Unreasonable Effectiveness of Recurrent Neural Networks <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
3. Cristopher Olah: Understanding LSTM's <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
4. CS224d: Deep Learning for Natural Language Processing
<http://cs224d.stanford.edu/syllabus.html>