Feature Learning

NTV 2026 / HC3b

Today's lecturer: Sandro Pezzelle

s.pezzelle@uva.nl

Where we are at (in the course)

1. **Intro to NLP & Statistics for NLP:** Recognising and estimating statistical laws from linguistic data (with *bonus in-class quiz #1*!)

Where we are at (in the course)

1. **Intro to NLP & Statistics for NLP:** Recognising and estimating statistical laws from linguistic data (with *bonus in-class quiz #1*!)
2. **Text classification and regression:** with generative (NBC) and discriminative (GLM) models

Where we are at (in the course)

1. **Intro to NLP & Statistics for NLP:** Recognising and estimating statistical laws from linguistic data (with *bonus in-class quiz #1*!)
2. **Text classification and regression:** with generative (NBC) and discriminative (GLM) models
3. **Feature learning:** we go beyond the GLM (where features are provided/manually engineered), and we do automated feature learning with Neural Networks (FFNNs and RNNs)

Where we are at (in the course)

1. **Intro to NLP & Statistics for NLP:** Recognising and estimating statistical laws from linguistic data (with *bonus in-class quiz #1*!)
2. **Text classification and regression:** with generative (NBC) and discriminative (GLM) models
3. **Feature learning:** we go beyond the GLM (where features are provided/manually engineered), and we do automated feature learning with Neural Networks (FFNNs and RNNs) We continue today! Also today: *bonus in-class quiz #2*
4. **Midterm exam:** next Tuesday!

How to prepare for the midterm?

- Canvas > Exercises and solutions
- Practice midterm (with solutions)
- Ungraded exercises E1, E2, and E3 (with solutions)
- In-class quizzes (with solutions)
- Any other exercises, e.g., including those seen and solved in class
- Any reading materials, refreshers, and notebooks
- T1, T2, and T3: There won't be programming exercises at the exam, of course, but solving these problems helps reasoning about similar problems!

Feature learning with Neural Networks

Last class: Recap

- **Introduction to (automated) feature learning** or representation learning—going beyond the manually engineered features provided to GLMs

Last class: Recap

- **Introduction to (automated) feature learning** or representation learning—going beyond the manually engineered features provided to GLMs
- **Concept of 'embedding':** translating words, phrases, or text into numerical, dense vectors in a continuous, multi-dimensional space—going beyond sparse one-hot encodings

Last class: Recap

- **Introduction to (automated) feature learning** or representation learning—going beyond the manually engineered features provided to GLMs
- **Concept of 'embedding':** translating words, phrases, or text into numerical, dense vectors in a continuous, multi-dimensional space—going beyond sparse one-hot encodings
- **`embed()` function** to retrieve our separate vectors from a pre-computed matrix (one for each token); not yet a text encoder

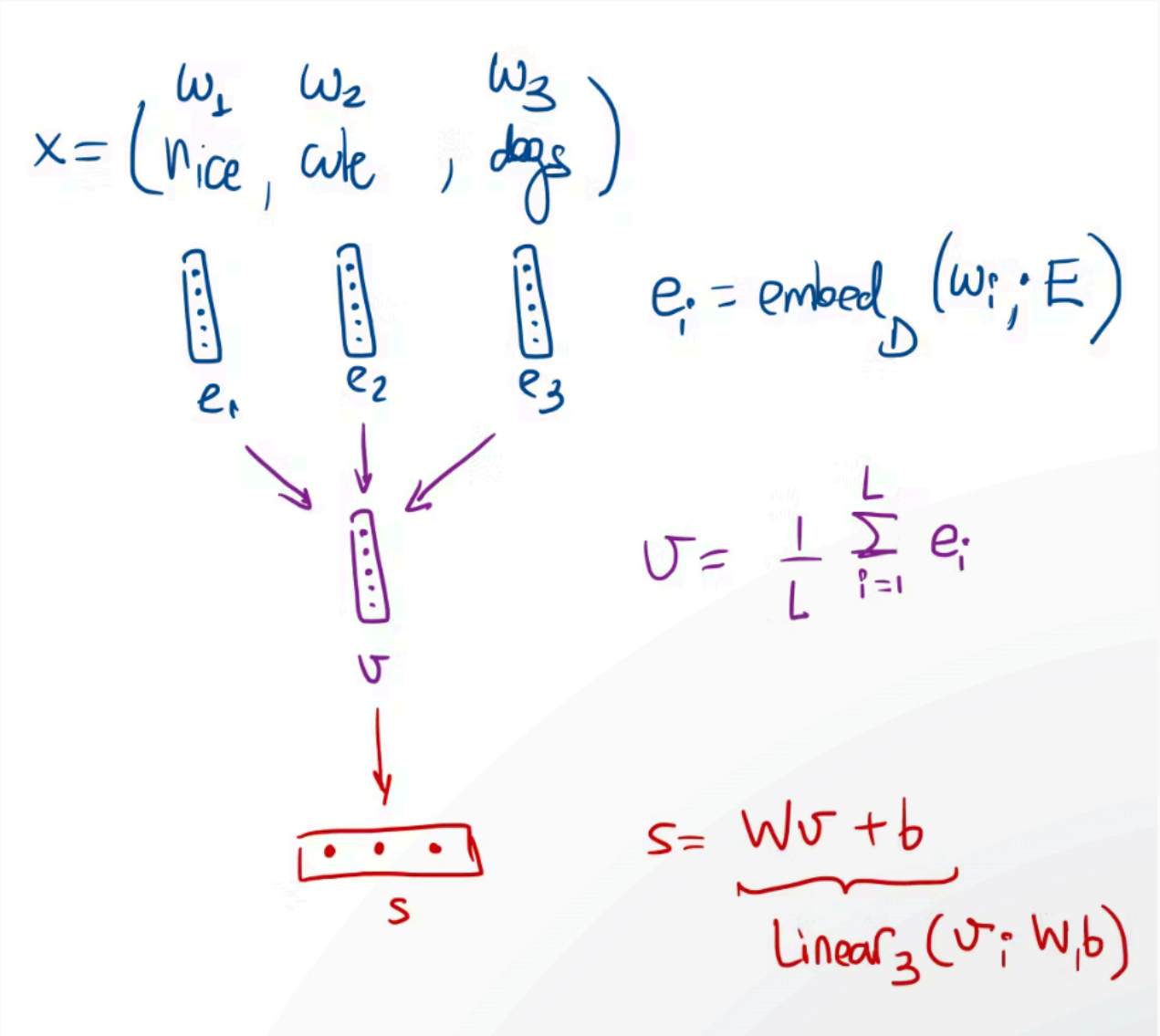
Last class: Recap

- **Introduction to (automated) feature learning** or representation learning—going beyond the manually engineered features provided to GLMs
- **Concept of 'embedding':** translating words, phrases, or text into numerical, dense vectors in a continuous, multi-dimensional space—going beyond sparse one-hot encodings
- **`embed()` function** to retrieve our separate vectors from a pre-computed matrix (one for each token); not yet a text encoder
- **From embeddings to text encoders** via *pooling* (combine embeddings with same dimensionality element-wise, e.g., sum) or *composing operations* (concatenation + some linear transformation)

Last class: Recap

- **Introduction to (automated) feature learning** or representation learning—going beyond the manually engineered features provided to GLMs
- **Concept of 'embedding':** translating words, phrases, or text into numerical, dense vectors in a continuous, multi-dimensional space—going beyond sparse one-hot encodings
- **`embed()` function** to retrieve our separate vectors from a pre-computed matrix (one for each token); not yet a text encoder
- **From embeddings to text encoders** via *pooling* (combine embeddings with same dimensionality element-wise, e.g., sum) or *composing operations* (concatenation + some linear transformation)
- **Linear transformation** to, e.g., a label space. From dimensionality u to v > if we add ***`softmax`*** on top, probabilities

Last class: Recap



Today

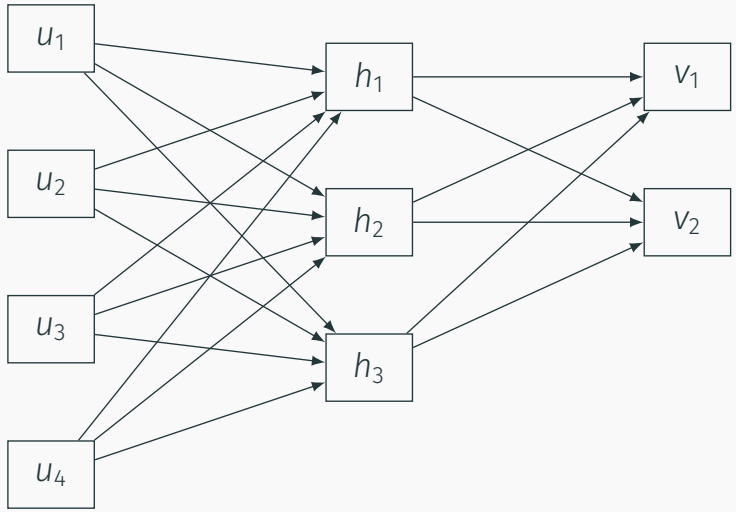
- **Feedforward Neural Network (FFNN):** input number of tokens needs to be fixed; fixed number of inputs, fixed number of outputs

Today

- **Feedforward Neural Network (FFNN):** input number of tokens needs to be fixed; fixed number of inputs, fixed number of outputs
- **Recurrent Neural Network (RNN):** can handle variable number of inputs; hidden states conditioned on the previous ones

Feed-Forward Neural Network

Used to compose a **fixed (!) number of inputs**. Combines two or more linear transformations with non-linear activation functions (*tanh*, *sigmoid*) in between



The input features *interact* forming the intermediate (*hidden*) features, which interact forming the final (*output*) features

Formally, every continuous function of $\mathbf{u}$ can be represented by a wide enough FFNN

Example: with input $\mathbf{u} \in \mathbb{R}^I$,

$$\mathbf{h} = \tanh(\mathrm{linear}_H(\mathbf{u}; \theta_{\mathrm{hid}}))$$

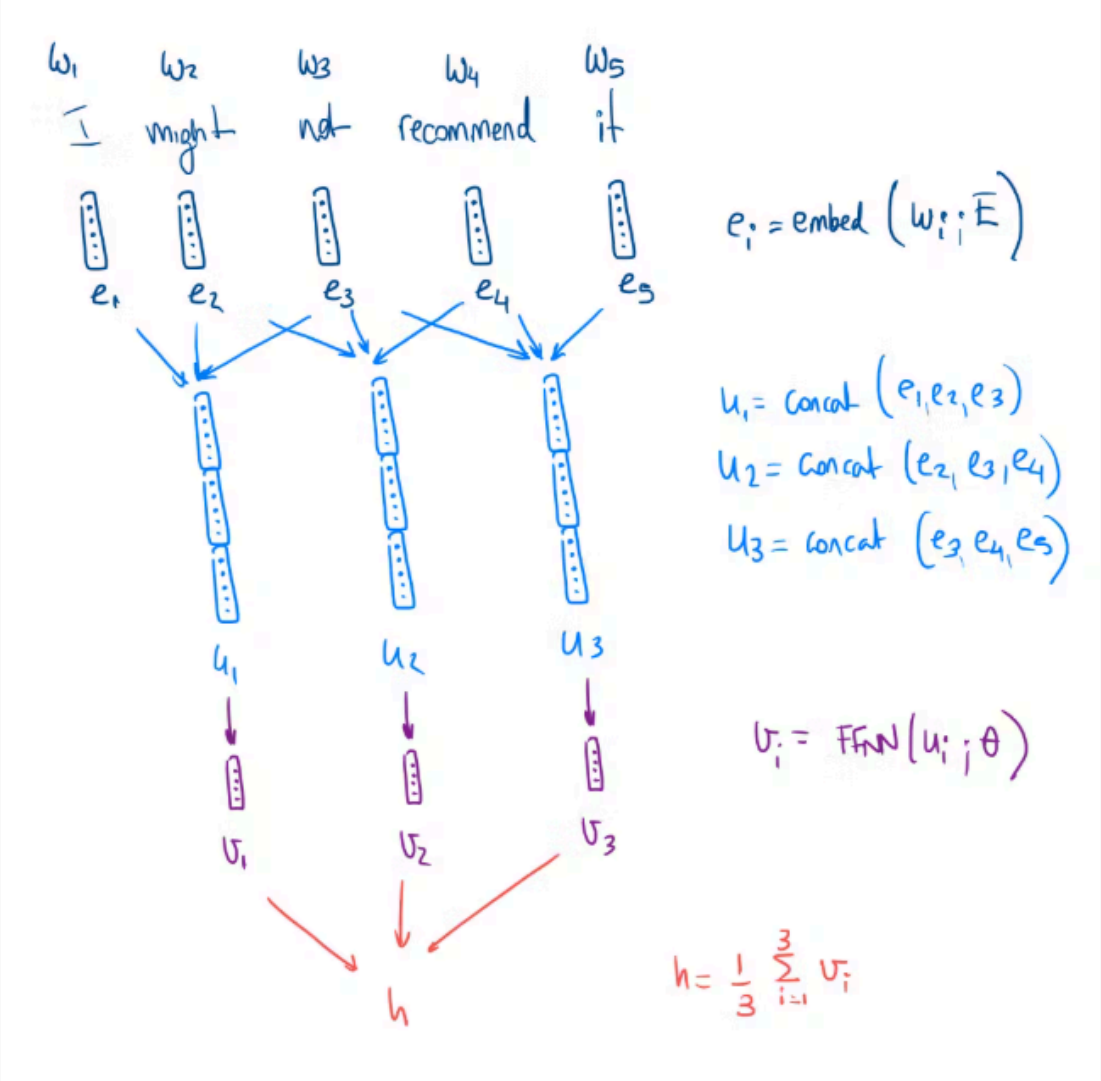
$$\mathbf{v} = \mathrm{linear}_O(\mathbf{h}; \theta_{\mathrm{out}})$$

Trainable parameters: $\theta = (\theta_{\mathrm{hid}}, \theta_{\mathrm{out}})$

(If you want to know more) Universal Approximation Theorem

- **Universal Approximation Theorem (UAT)**: Mathematical proof that a feedforward network with a non-linear activation function can express a transformation from an input space $\mathbb{R}^n$ to an output space $\mathbb{R}^m$
- It establishes that such networks can approximate any continuous function f on a compact subset $K \subset \mathbb{R}^n$ to arbitrary precision (we restrict ourselves to a bounded, closed region of the input space)
- The proof requires the activation function $\sigma(\cdot)$ to be **non-linear**
- **Without non-linearity:** If σ is linear, the composition of layers reduces to $W_L W_{L-1} \cdots W_1 x = Wx$, which is just a single matrix multiplication. This can only represent linear mappings
- **With non-linearity:** Activation functions such as *`ReLU`*, *`sigmoid`*, or *`tanh`* allow the network to create the "bent" and localized shapes necessary to approximate arbitrary curved functions

Example: Trigrams



Each $\mathbf{v}_i$ represents a trigram in the input text. Compare this to a trigram feature function for the NBC. Do you see clear advantages?

Advantages

Neural Networks *do not store* the observations, e.g., the trigrams, in a table, but they **predict** the representations of these observations, e.g., the trigrams

Advantages

Neural Networks *do not store* the observations, e.g., the trigrams, in a table, but they **predict** the representations of these observations, e.g., the trigrams

With trigram-based NBCs, the size of the feature function would grow **exponentially**; with NNs, it grows **linearly**

Advantages

Neural Networks *do not store* the observations, e.g., the trigrams, in a table, but they **predict** the representations of these observations, e.g., the trigrams

With trigram-based NBCs, the size of the feature function would grow **exponentially**; with NNs, it grows **linearly**

Keep in mind: Neural Networks achieve a lot of feature combinations at a very reduced cost!

Recurrent Neural Network

Feed-forward type building block that we use repeatedly in a sort of ‘for loop’, **recursively**

Recurrent Neural Network

Feed-forward type building block that we use repeatedly in a sort of ‘for loop’, **recursively**

It allows us to compose a variable number of inputs (solving the limitation of FFNNs)

Recurrent Neural Network

Feed-forward type building block that we use repeatedly in a sort of ‘for loop’, **recursively**

It allows us to compose a variable number of inputs (solving the limitation of FFNNs)

By a very simple strategy: composing a fixed of inputs at a time

Recurrent Neural Network

Feed-forward type building block that we use repeatedly in a sort of ‘for loop’, **recursively**

It allows us to compose a variable number of inputs (solving the limitation of FFNNs)

By a very simple strategy: composing a fixed of inputs at a time

RNN step or elementary composition function of the RNN: combining a fixed number of inputs, 2 in most applications

Recurrent Neural Network

Feed-forward type building block that we use repeatedly in a sort of ‘for loop’, **recursively**

It allows us to compose a variable number of inputs (solving the limitation of FFNNs)

By a very simple strategy: composing a fixed of inputs at a time

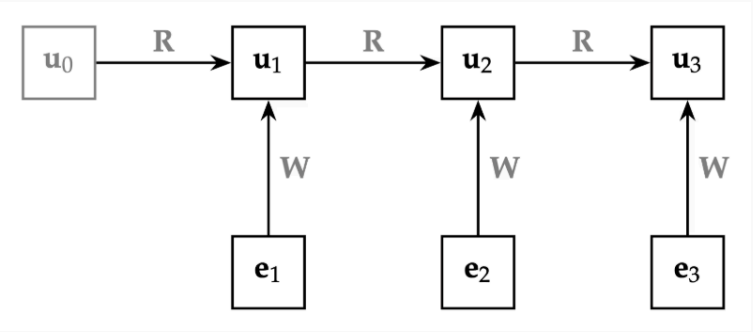
RNN step or elementary composition function of the RNN: combining a fixed number of inputs, 2 in most applications

Learned composition of differentiable functions; it’s parametric; it learns parameters (based on an objective)

Recurrent Neural Network

Used to compose a **variable number of inputs**

Made of a simple transformation in a 'for loop'. At each step i, it composes two vectors: the RNN *state* $\mathbf{u}_{i-1}$ (representing the past) and an input $\mathbf{e}_i$. The final state represents the **entire** input sequence.



Example: inputs $\langle \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \rangle$.

$$\mathbf{u}_0 = \mathbf{0}, \qquad \mathbf{u}_1 = \tanh(\mathbf{R}\mathbf{u}_0 + \mathbf{W}\mathbf{e}_1),$$
$$\mathbf{u}_2 = \tanh(\mathbf{R}\mathbf{u}_1 + \mathbf{W}\mathbf{e}_2), \quad \mathbf{u}_3 = \tanh(\mathbf{R}\mathbf{u}_2 + \mathbf{W}\mathbf{e}_3)$$

Trainable parameters:
$\theta = (\mathbf{R} \in \mathbb{R}^{H \times H}, \mathbf{W} \in \mathbb{R}^{H \times D})$

RNNs are a ‘template’ model

In practice, this simple RNN has some limitations (problem with long-term dependencies/forgetting the past; exploding & vanishing gradient, etc.)

In practice, you’d be using more complex and powerful RNNs, such as LSTMs, biLSTMs, and GRU—you’ll see them in the *laptopcollege*

These have more complex, more capable cell states

Mentimeter

Outline

- → Applications
- → Limitations
- → Improvements

Text Retrieval

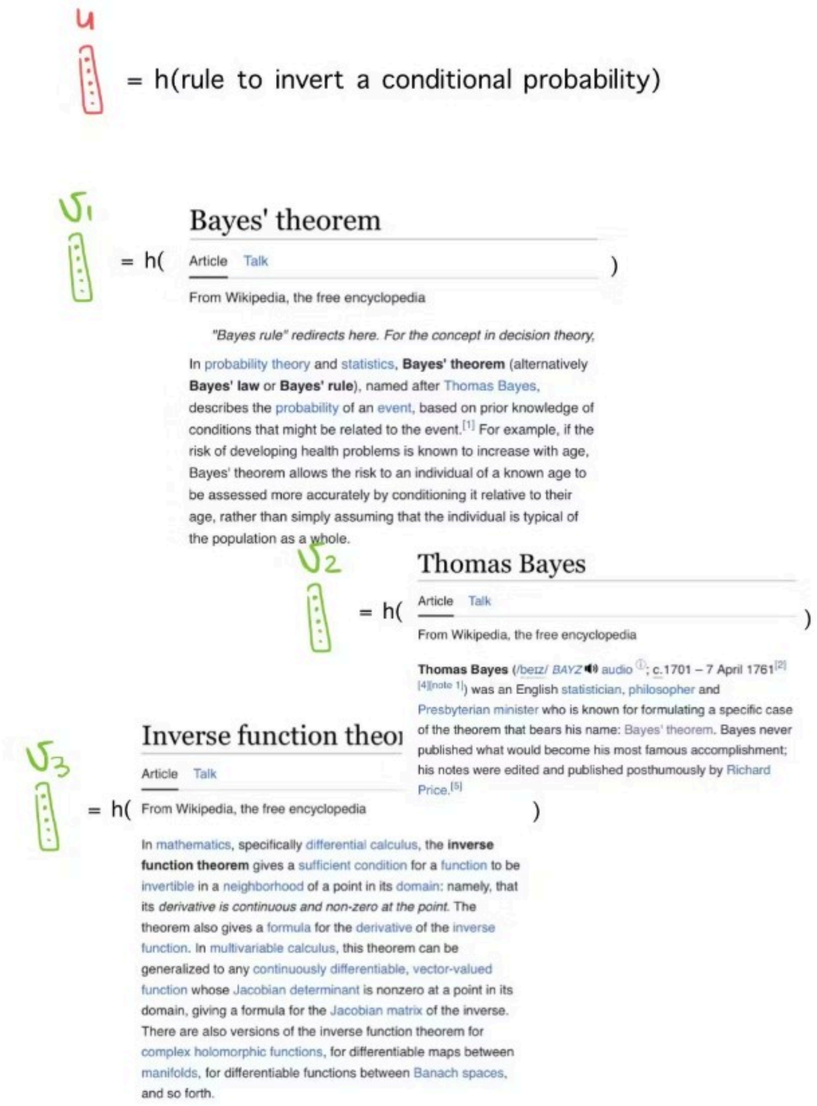
A text encoder $\mathbf{h}(\cdot; \theta)$ can be used to *retrieve* documents that are relevant to a query.

We represent the query q and candidate documents $d_1, \ldots, d_n$ in the same D-dimensional space, then use a notion of vector similarity to rank documents for relevance.

Query encoding: $\mathbf{u} = \mathbf{h}(q; \theta)$
Document encoding: $\mathbf{v}_n = \mathbf{h}(d_n)$
Similarity: $\cos(\mathbf{u}, \mathbf{v}_n) = \frac{\mathbf{u}^\top \mathbf{v}_n}{||\mathbf{u}||_2 ||\mathbf{v}_n||_2}$

Efficient ranking via *maximum inner product search* (MIPS).

See Chapter 14 for more.



Mentimeter

Sentiment Classification

We condition on an input document x and learn to predict probabilities over K sentiment levels. Our text encoder $\mathbf{h}(x)$ typically employs a BiLSTM.

Figure from this paper, which investigates various types of text encoders.

Mentimeter

Aspect-Based Sentiment Analysis

We condition on an input document x and learn to predict probabilities over K sentiment levels for each of J aspects.

A single text encoder $\mathbf{h}(x)$, typically based on a BiLSTM, is shared across J output layers (one for each aspect).

Example (assuming Categorical output):

$$\mathbf{u} = \mathbf{h}(x; \theta_{\text{enc}})$$
$$\mathbf{s}_1 = \text{linear}_K(\mathbf{u}; \theta_{\text{feel}})$$
$$\mathbf{s}_2 = \text{linear}_K(\mathbf{u}; \theta_{\text{look}})$$
$$\mathbf{s}_3 = \text{linear}_K(\mathbf{u}; \theta_{\text{smell}})$$
$$\mathbf{s}_4 = \text{linear}_K(\mathbf{u}; \theta_{\text{taste}})$$
$$\mathbf{s}_5 = \text{linear}_K(\mathbf{u}; \theta_{\text{overall}})$$

Sometimes this is modelled as regression.

Figure from this paper.

‘Partridge in a Pear Tree’, brewed by ‘The Bruery’

Dark brown with a light tan head, minimal lace and low retention. Excellent aroma of dark fruit, plum, raisin and red grape with light vanilla, oak, caramel and toffee. Medium thick body with low carbonation. Flavor has strong brown sugar and molasses from the start over bready yeast and a dark fruit and plum finish. Minimal alcohol presence. Actually, this is a nice quad.

Feel: 4.5 Look: 4 Smell: 4.5 Taste: 4 Overall: 4

Mentimeter

Textual Entailment

We condition on two pieces of text, a premise x and a hypothesis y and predict the probabilities of logical relations (contradiction, neutral, or entailment).

We typically encode x and y with the same BiLSTM and combine their encodings (the BiLSTM final state for each of the two input sequences) as inputs to an output layer for a 3-way classifier:

$$\mathbf{u} = \text{encode}(x; \theta_{\text{in}})$$
$$\mathbf{v} = \text{encode}(y; \theta_{\text{in}})$$
$$\mathbf{s} = \text{FFNN}_3([\mathbf{u}, \mathbf{v}, \mathbf{u} - \mathbf{v}, \mathbf{u} \odot \mathbf{v}]; \theta_{\text{out}})$$

Figure from this paper.

A man inspects the uniform of a figure in some East Asian country.	**contradiction** C C C C C	The man is sleeping
An older and younger man smiling.	**neutral** N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	**contradiction** C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	**entailment** E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	**neutral** N N E C N	A happy woman in a fairy costume holds an umbrella.

Question Answering

We are given a question q and a passage d. The task is to identify a span of text that contains the answer a to the question: we predict for each position i the probability p_i that it *begins* the answer and the probability q_i that it *ends* the answer.

Input: $x = w_{1:L}$, the concatenation of q and d (as in the figure)
.

We embed the tokens $\mathbf{e}_i = \text{embed}_D(w_i; \theta_{\text{in}})$, then encode all positions exploiting contextual information, for example, using a BiLSTM $\mathbf{h}_{1:L} = \text{BiLSTM}(\mathbf{e}_{1:L}; \theta_{\text{enc}})$.

For each position we predict:

$$p_i = \text{sigmoid}(\text{linear}_1(\mathbf{h}_i; \theta_{\text{begin}}))$$
$$q_i = \text{sigmoid}(\text{linear}_1(\mathbf{h}_i; \theta_{\text{end}}))$$

See Chapter 14 for more.

Mentimeter

MachineTranslation Evaluation

Inputs: text x and its translation y; in some settings also a reference translation r.

Output: score capturing translation quality.

- Data: human direct assessment (DA).
- Model: text regression to mean or text regression with Gaussian output distribution.

Modern models use a cross-encoder of x and y (and r, when available).

Figure from [this paper](#).

MT	DA	COMET	UA-COMET
Она сказала, ’Это не собирается работать.	-0.815	*0.586*	0.149 **[-0.92, 1.22]**
Gloss: “*She said, ‘that’s not willing to work*”			
Она сказала: «Это не сработает.	0.768	1.047	1.023 [0.673, 1.374]
Gloss: “*She said, «That will not work*”			

Table 1: Example of uncertainty-aware MT evaluation for a sentence in the WMT20 dataset. Shown are two Russian translations of the same English source “*She said, ‘That’s not going to work.*” with reference “Она сказала: “Не получится.” For the first sentence,

Mentimeter

Outline

- Applications
- **Limitations**
- Improvements

Mentimeter

Data Hunger

Interesting text encoders tend to have many trainable parameters (a large embedding matrix, some FFNNs, some RNNs). When we estimate them via MLE, the model tends to overfit (i.e., the training criterion improves, but our heldout/validation criterion does not).

There are generally two strategies: more data and more regularisation.

Mentimeter

Lack of Parallelism

RNNs process inputs sequentially, the RNN state is a bottleneck through which information has to flow over many steps.

It can be difficult for an RNN to retain relevant information over very long sequences, or allow sufficient interaction amongst features from very far away portions of the input.

Mentimeter

Outline

- → Applications
- → Limitations
- → **Improvements**

Mentimeter

Dropout

A form of regularisation (hence, used during *training*) whereby, at random, we set some input units to 0 before a linear layer. This encourages FFNNs to use their capacity to create some redundancy (multiple ways to encode the same), as opposed to overfit to the observed data.

Word dropout: same idea, but we drop words at random (this is particularly useful for regularising RNNs).

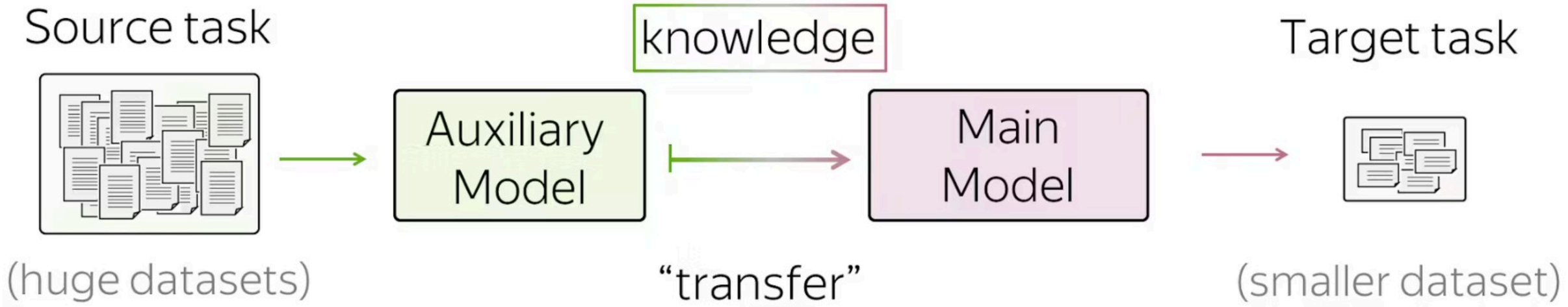
Mentimeter

Transfer Learning

An alternative to obtaining *more labelled data* is to train (parts of) our models using data for other tasks (when those are easier to obtain).

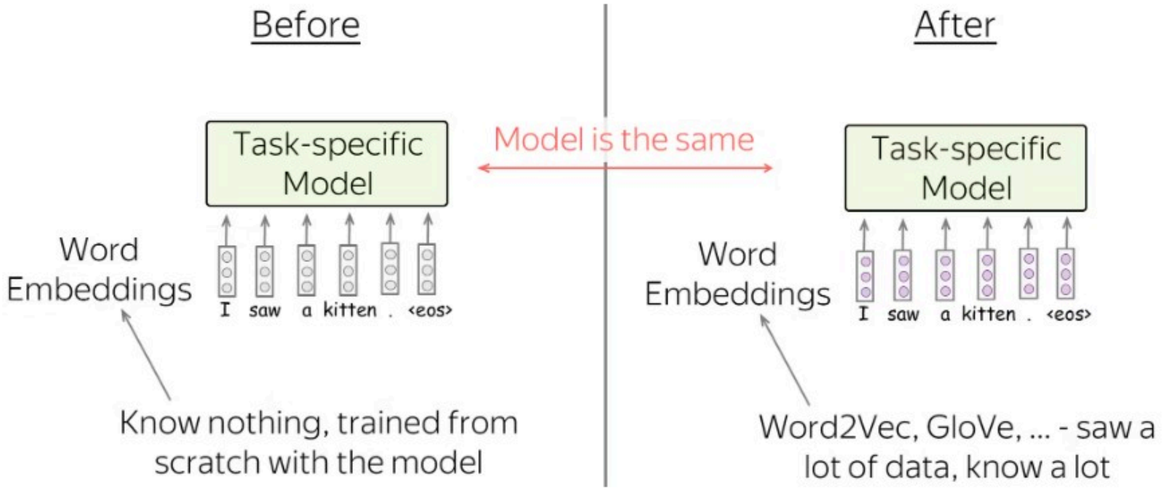
These *auxiliary tasks* are there to help us learn a good text encoder. Think of it this way: they might not capture much about our downstream task (e.g., sentiment classification), but they might capture plenty of general linguistic processing skills.

Mentimeter



[Illustration by Lena Voita]

Mentimeter



Word Embeddings

We learn to represent words in terms of their distributional patterns in text.

We can learn this using lots of text, even the entire web (for a certain language), and then transfer these powerful representations to our downstream tasks.

Mentimeter

(From HC1a)

I've had a wonderful weekend! I always wanted to buy a *berimbau*. On Saturday, I finally went to that fancy music store in Haarlem. The rest of the weekend, I practised some of my favourite tunes on it.

Mentimeter

"You shall know a word by the company it keeps"

John R. Firth

Mentimeter

Distributional semantics

An artificial task for which we can find abundant supervision is that of predicting the probability of two words occurring together (e.g., in the same sentence, or close to each other in a sentence).

I went to the park and saw a dog chasing a little cat .

Mentimeter

word2vec

For each *target* word in a corpus (i.e., any occurrence of a word we care to represent), we parameterise a binary classifier:

$$
\begin{aligned}
&Y|T=t, C=c \sim \\
&\text{Bernoulli}(\text{sigmoid}(\mathbf{u}^\top \mathbf{v})) \\
&\mathbf{u} = \text{embed}_D(t; \mathbf{E}) \\
&\mathbf{v} = \text{embed}_D(c; \mathbf{E})
\end{aligned}
$$

We train this classifier via MLE. **Positive examples** are obtained from observed sentences. Negative examples are sampled randomly from the corpus (a.k.a., 'negative sampling').

We transfer the **embedding matrix**.

I went to the park and saw a dog chasing a little cat .

Mentimeter

LSTM and GRU

Long short-term memory (LSTM) and gated-recurrent unit (GRU) are two modern RNN cells that are better able to work with long sequences.

They essentially define a slightly more complex RNN step function.

In T3 you work with LSTMs.

Mentimeter

Summary

- Feature learning powers *all* moderns NLP applications
- We learn text encoders along with the entire model via MLE
- Regularisation and transfer learning are essential for good designs

Mentimeter

What Next

- Wrap up T3 and E3
- Midterm
- After the midterm: text generation