Approaches to MT: SMT, PBMT, NMT

Ondřej Bojar

March 5, 2020

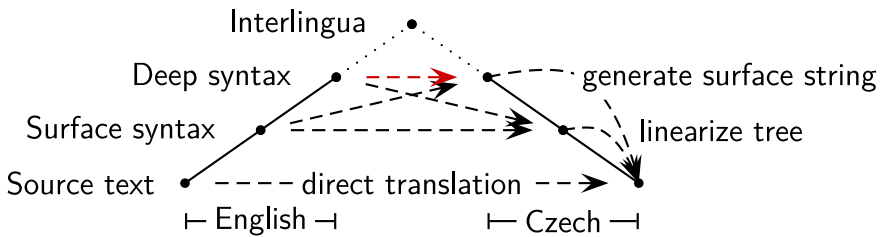
EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Outline

- Approaches to MT.
- What makes MT statistical.
  - Probability of a sentence, Bayes' law.
  - Log-linear model.
- Phrase-Based MT.
  - Features used.
  - Training Pipeline.
  - Unjustified independence assumptions.
- Neural MT.
  - Deep learning summary.
  - Representing text
  - Encoder-decoder architecture.

Approaches to Machine Translation



- The deeper analysis, the easier the **transfer** should be.
- A hypothetical interlingua captures pure meaning.
- Statistical systems learn "automatically" from data.
- Rule-based systems implemented by linguists-programmers.

Until NMT, it was best to combine the approaches.

Zap through History

- Rule-Based MT.
  Linguists/language experts write rules.
  Controlled Language: Authors restricted to produce MT-translatable text.
- Example-Based MT.
  Given translation memories, find examples similar to input.
- Statistical MT:
  1. Word-Based.
  2. Phrase-Based.
  3. Syntax-Based.
  4. Neural.

Quotes

Warren Weaver (1949):

> *I have a text in front of me which is written in Russian but I am going to pretend that it is really written in English and that is has been coded in some strange symbols. All I need to do is strip off the code in order to retrieve the information contained in the text.*

Noam Chomsky (1969):

> *...the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.*

Frederick Jelinek (80's; IBM; later JHU and sometimes ÚFAL)

> *Every time I fire a linguist, the accuracy goes up.*

Hermann Ney (RWTH Aachen University):

> *MT* = *Linguistic* **M***odelling* + *Statistical Decision* **T***heory*

The Statistical Approach

(Statistical = Information-theoretic.)

- Specify a probabilistic model.
  = How is the probability mass distributed among possible outputs given observed inputs.
- Specify the training criterion and procedure.
  = How to learn free parameters from training data.

Notice:

- Linguistics helpful when designing the models:
  - How to divide input into smaller units.
  - Which bits of observations are more informative.

Ultimate Goal of Traditional SMT

Find **minimum translation units** (MTUs) $\sim$ graph partitions:

- such that they are frequent across many sentence pairs.
- without imposing (too hard) constraints on reordering.
- (ideally in an unsupervised fashion, no reliance on linguistics).

Available data: Word co-occurrence statistics:

- In large monolingual data (usually up to 10^9 words).
- In smaller parallel data (up to 10^7 words per language).
- Optional automatic rich linguistic annotation.

Statistical MT

Given a source (foreign) language sentence $f_1^J = f_1 \dots f_j \dots f_J$,
Produce a target language (English) sentence $e_1^I = e_1 \dots e_j \dots e_I$.
Among all possible target language sentences, choose the sentence with the highest probability:

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\operatorname{argmax}}\, p(e_1^I | f_1^J) \tag{1}$$

We stick to the e_1^I, f_1^J notation despite translating from English to Czech.

Brute-Force MT

Translate only sentences listed in a "translation memory" (TM):

Good morning.	=	Dobré ráno.
How are you?	=	Jak se máš?
How are you?	=	Jak se máte?

$$p(e_1^I|f_1^J) = \begin{cases} 1 & \text{if } e_1^I = f_1^J \text{ seen in the TM} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Any problems with the definition?

Brute-Force MT

Translate only sentences listed in a "translation memory" (TM):

Good morning.	=	Dobré ráno.
How are you?	=	Jak se máš?
How are you?	=	Jak se máte?

$$p(e_1^I|f_1^J) = \begin{cases} 1 & \text{if } e_1^I = f_1^J \text{ seen in the TM} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

- Not a probability. There may be f_1^J, s.t. $\sum_{e_1^I} p(e_1^I|f_1^J) > 1$.

  $\Rightarrow$ Have to normalize, use $\frac{\text{count}(e_1^I, f_1^J)}{\text{count}(f_1^J)}$ instead of 1.
- Not "smooth", no generalization:

Good morning.	$\Rightarrow$	Dobré ráno.
Good evening.	$\Rightarrow$	$\emptyset$

Bayes' Law

Bayes' law for conditional probabilities: $p(a|b) = \frac{p(b|a)p(a)}{p(b)}$

So in our case:

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\text{argmax}}\, p(e_1^I | f_1^J)$$

Apply Bayes' law

$$= \underset{I, e_1^I}{\text{argmax}}\, \frac{p(f_1^J | e_1^I) p(e_1^I)}{p(f_1^J)}$$

$p(f_1^J)$ constant

$$= \underset{I, e_1^I}{\text{argmax}}\, p(f_1^J | e_1^I) p(e_1^I)$$

$\Rightarrow$ irrelevant in maximization

Also called "Noisy Channel" model.

Motivation for Noisy Channel

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\text{argmax}}\, p(f_1^J | e_1^I) p(e_1^I) \tag{3}$$

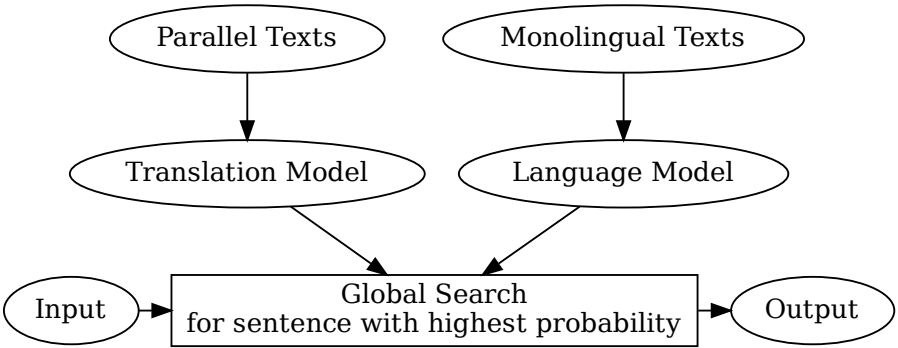
Bayes' law divided the model into components:

$p(f_1^J | e_1^I)$ Translation model ("reversed", $e_1^I \rightarrow f_1^J$)
...is it a likely translation?

$p(e_1^I)$ Language model (LM)
...is the output a likely sentence of the target language?

- The components can be trained on different sources.
  There are far more monolingual data ⇒ language model can be more reliable.

Without Equations



Summary of Language Models

- $p(e_1^I)$ should report how "good" sentence e_1^I is.
- We surely want $p(\text{The the the.}) < p(\text{Hello.})$
- How about $p(\text{The cat was black.}) < p(\text{Hello.})$?

  ...We don't really care in MT. We hope to compare synonymic sentences.

LM is usually a 3-gram language model:

$$p(\curvearrowright\ \curvearrowright \text{ The cat was black . } \curvearrowleft\ \curvearrowleft) = \begin{matrix} p(\text{The}|\curvearrowright\ \curvearrowright) & p(\text{cat}|\curvearrowright \text{ The}) & p(\text{was}|\text{The cat}) \\ p(\text{black}|\text{cat was}) & p(.|\text{was black}) & p(\curvearrowleft|\text{black .}) \\ p(\curvearrowleft|.\ \curvearrowleft) & & \end{matrix}$$

Formally, with $n = 3$:

$$p_{\text{LM}}(e_1^I) = \prod_{i=1}^{I} p(e_i|e_{i-n+1}^{i-1}) \tag{4}$$

Estimating and Smoothing LM

$$p(w_1) = \frac{\text{count}(w_1)}{\text{total words observed}} \quad \text{Unigram probabilities.}$$
$$p(w_2|w_1) = \frac{\text{count}(w_1 w_2)}{\text{count}(w_1)} \quad \text{Bigram probabilities.}$$
$$p(w_3|w_2, w_1) = \frac{\text{count}(w_1 w_2 w_3)}{\text{count}(w_1 w_2)} \quad \text{Trigram probabilities.}$$

Unseen ngrams ($p(\textit{ngram}) = 0$) are a big problem, invalidate whole sentence: $p_{\text{LM}}(e_1^I) = \cdots \cdot 0 \cdot \cdots = 0$

$\Rightarrow$ Back-off with shorter ngrams:

$$p_{\text{LM}}(e_1^I) = \prod_{i=1}^{I} \Big(\begin{array}{l} 0.8 \cdot p(e_i|e_{i-1}, e_{i-2}) + \\ 0.15 \cdot p(e_i|e_{i-1}) + \\ 0.049 \cdot p(e_i) + \\ 0.001 \end{array} \Big) \neq 0 \tag{5}$$

From Bayes to Log-Linear Model

Och (2002) discusses some problems of Equation 3:

- Models estimated unreliably $\Rightarrow$ maybe LM more important:

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\operatorname{argmax}}\, p(f_1^J|e_1^I)(p(e_1^I))^2 \tag{6}$$

- In practice, "direct" translation model equally good:

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\operatorname{argmax}}\, p(e_1^I|f_1^J)p(e_1^I) \tag{7}$$

- Complicated to *correctly* introduce other dependencies.
  $\Rightarrow$ Use log-linear model instead.

Log-Linear Model (1)

- $p(e_1^I|f_1^J)$ is modelled as a weighted combination of models, called "feature functions": $h_1(\cdot,\cdot) \ldots h_M(\cdot,\cdot)$

$$p(e_1^I|f_1^J) = \frac{\exp(\sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J))}{\sum_{e'^{I'}_1} \exp(\sum_{m=1}^{M} \lambda_m h_m(e'^{I'}_1, f_1^J))} \quad (8)$$

- Each feature function $h_m(e, f)$ relates source f to target e.
  E.g. the feature for n-gram language model:

$$h_{\mathrm{LM}}(f_1^J, e_1^I) = \log \prod_{i=1}^{I} p(e_i|e_{i-n+1}^{i-1}) \quad (9)$$

- Model weights λ_1^M specify the relative importance of features.

Log-Linear Model (2)

As before, the constant denominator not needed in maximization:

$$\begin{aligned} \hat{e}_1^{\hat{I}} &= \text{argmax}_{I,e_1^I} \frac{\exp(\sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J))}{\sum_{e'^{I'}_1} \exp(\sum_{m=1}^{M} \lambda_m h_m(e'^{I'}_1, f_1^J))} \\ &= \text{argmax}_{I,e_1^I} \exp(\sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J)) \end{aligned} \quad (10)$$

Relation to Noisy Channel

With equal weights and only two features:

- $h_{\mathsf{TM}}(e_1^I, f_1^J) = \log p(f_1^J|e_1^I)$ for the translation model,
- $h_{\mathsf{LM}}(e_1^I, f_1^J) = \log p(e_1^I)$ for the language model,

log-linear model reduces to Noisy Channel:

$$\begin{aligned}
\hat{e}_1^{\hat{I}} &= \operatorname{argmax}_{I,e_1^I} \exp\left(\sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J)\right) \\
&= \operatorname{argmax}_{I,e_1^I} \exp(h_{\mathsf{TM}}(e_1^I, f_1^J) + h_{\mathsf{LM}}(e_1^I, f_1^J)) \\
&= \operatorname{argmax}_{I,e_1^I} \exp(\log p(f_1^J|e_1^I) + \log p(e_1^I)) \\
&= \operatorname{argmax}_{I,e_1^I} p(f_1^J|e_1^I) p(e_1^I)
\end{aligned} \tag{11}$$

Common Features of PBMT

- Phrase translation probability:
  $h_{\text{Phr}}(f_1^J, e_1^I, s_1^K) = \log \prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k)$ where $p(\tilde{f}_k|\tilde{e}_k) = \frac{\text{count}(\tilde{f}, \tilde{e})}{\text{count}(\tilde{e})}$
  $\Rightarrow$ Are all used units $\tilde{f} \leftrightarrow \tilde{e}$ likely translations?
- Word count/penalty: $h_{\text{wp}}(e_1^I, \cdot, \cdot) = I$
  $\Rightarrow$ Do we prefer longer or shorter output?
- Phrase count/penalty: $h_{\text{pp}}(\cdot, \cdot, s_1^K) = K$
  $\Rightarrow$ Do we prefer translation in more or fewer less-dependent bits?
- Reordering model: different basic strategies (Lopez, 2009)
  $\Rightarrow$ Which source spans can provide continuation at a moment?
- n-gram LM: $h_{\text{LM}}(\cdot, e_1^I, \cdot) = \log \prod_{i=1}^{I} p(e_i|e_{i-n+1}^{i-1})$
  $\Rightarrow$ Is output n-gram-wise coherent?

Features: Constructing or Scoring?

Are features used to construct hypotheses or just score them?

- Phrase translation probabilities: $\Rightarrow$ for construction, see below.
- Counts/penalties: $\Rightarrow$ for scoring only.
- Language models: $\Rightarrow$ for scoring only.
  But it *could* be used for construction: predict next word and confirm from translation.

Traditional MT "Pipeline"

"Training the Translation Model"

1. Find relevant parallel texts.
2. Align at the level of sentences.
3. Align at the level of words.
4. Extract translation units, with scores (co-oc. stats.).

(Language Model similar, "simple" words co-oc. stats, no alignment.)

"Tuning" ("MERT") = Actual training in the ML sense

5. Identify TM/LM/other model component weights.

Translation:

6. Decompose input into known units.
7. Search for best combinations of units.

1: Align Training Sentences

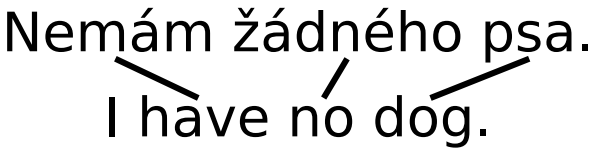
Nemám žádného psa.

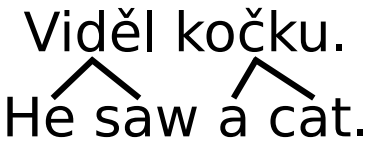
I have no dog.

Viděl kočku.

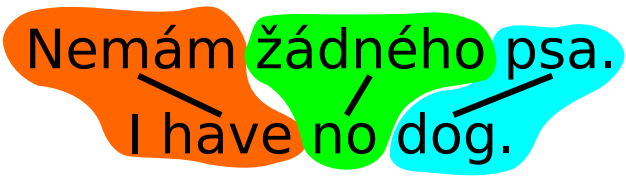
He saw a cat.

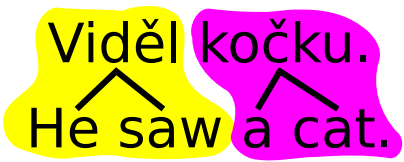
2: Align Words



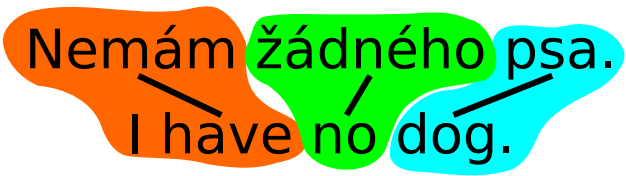


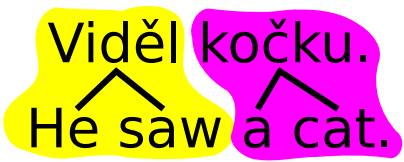
3: Extract Phrase Pairs (MTUs)





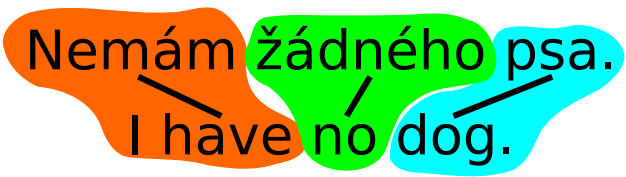
4: New Input

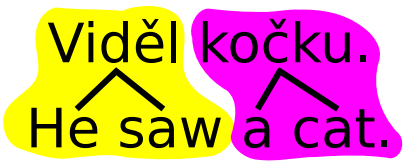




New input: Nemám kočku.

4: New Input





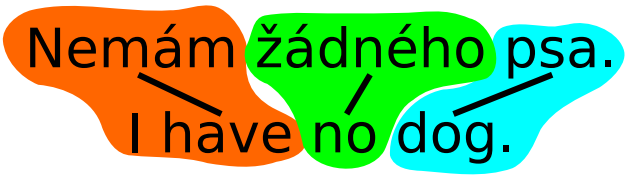
... I don't have cat.

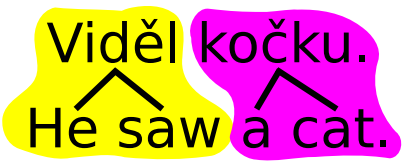
New input: Nemám kočku.

5: Pick Probable Phrase Pairs (TM)

6: So That n-Grams Probable (LM)

Meaning Got Reversed!





... I don't have cat.

New input:

What Went Wrong?

$$\hat{e}_1^{\hat{I}} = \underset{I, e_1^I}{\text{argmax}}\, p(f_1^J|e_1^I)p(e_1^I) = \underset{I, e_1^I}{\text{argmax}} \prod_{(\hat{f},\hat{e}) \in \text{phrase pairs of } f_1^J, e_1^I} p(\hat{f}|\hat{e})p(e_1^I) \quad (12)$$

- Too strong phrase-independence assumption.
  - Phrases do depend on each other.
    Here "nemám" and "žádného" jointly express one negation.
  - Word alignments ignored that dependence.
    But adding it would increase data sparseness.
- Language model is a separate unit.
  - $p(e_1^I)$ models the target sentence independently of f_1^J.

Redefining $p(e_1^I|f_1^J)$

What if we modelled $p(e_1^I|f_1^J)$ directly, word by word:

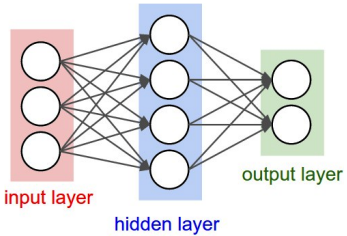
$$\begin{aligned} p(e_1^I|f_1^J) &= p(e_1, e_2, \dots e_I|f_1^J) \\ &= p(e_1|f_1^J) \cdot p(e_2|e_1, f_1^J) \cdot p(e_3|e_2, e_1, f_1^J) \dots \\ &= \prod_{i=1}^{I} p(e_i|e_1, \dots e_{i-1}, f_1^J) \end{aligned} \tag{13}$$

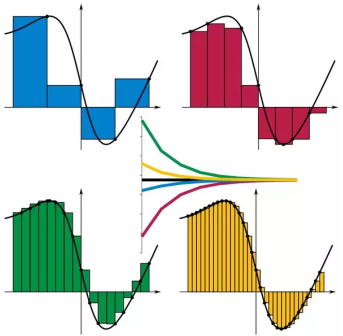
...this is "just a cleverer language model:" $p(e_1^I) = \prod_{i=1}^{I} p(e_i|e_1, \dots e_{i-1})$

Main Benefit: All dependencies available.

But what technical device can learn this?

NNs: Universal Approximators

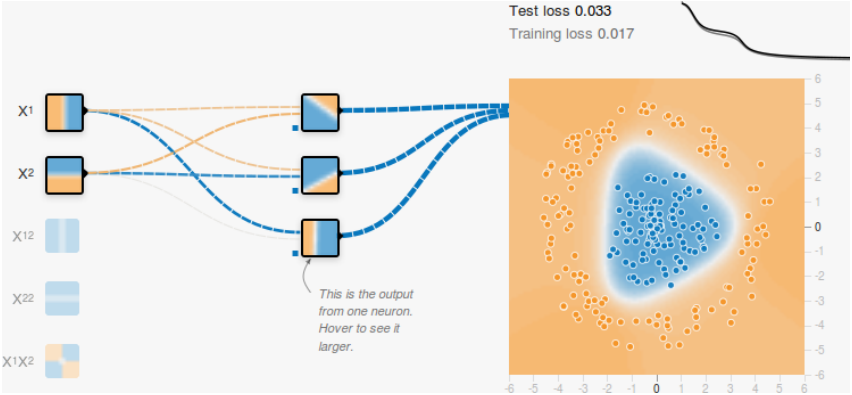




- A neural network with a single hidden layer (possibly huge) can approximate any continuous function to any precision.
- (Nothing claimed about learnability.)

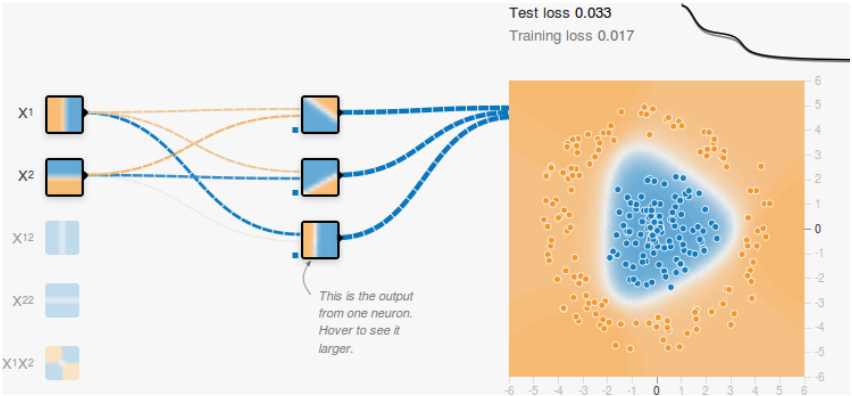
https://www.quora.com/How-can-a-deep-neural-network-with-ReLU-activations-in-its-hidden-layers-approximate-any-function

Play with playground.tensorflow.org



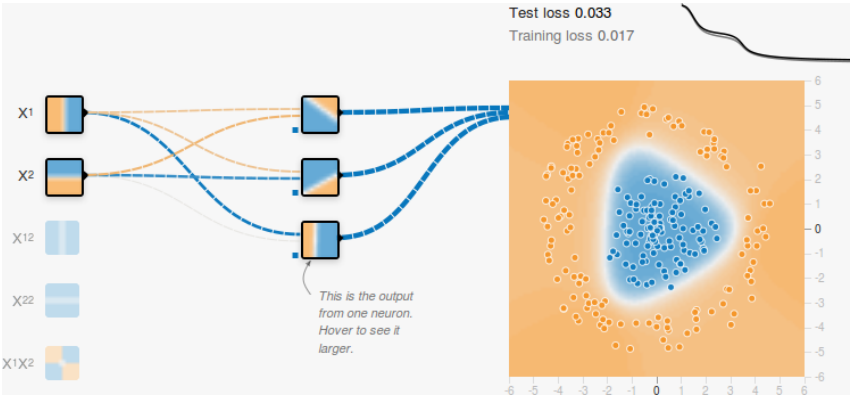
$$-0.43x_1 - 0.89x_2 + 2.0 > 0$$
and $$-0.67x_1 + 0.89x_2 + 2.1 > 0$$
and $$1.4x_1 - 0.067x_2 + 2.3 > 0$$

A DL "Program" Is Just a Computation...



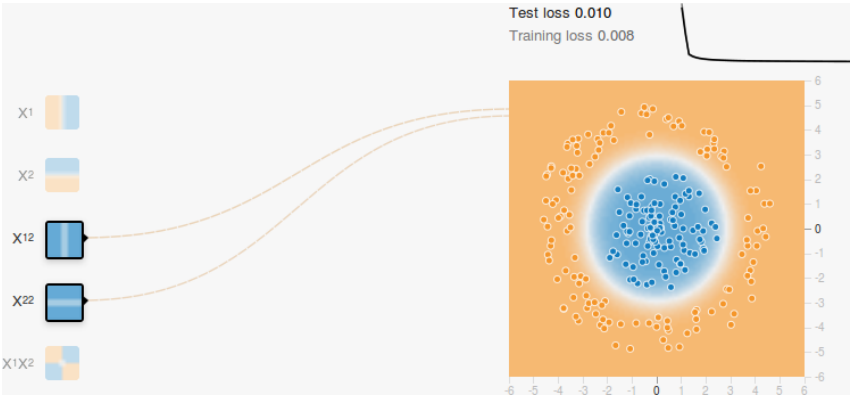
In fact: $1\tanh(-0.43x_1 - 0.89x_2 + 2.0)$
$+1\tanh(-0.67x_1 + 0.89x_2 + 2.1)$
$+1\tanh(1.4x_1 - 0.067x_2 + 2.3) - \pi/2 > 0$

... with Parameters Guessed Automatically



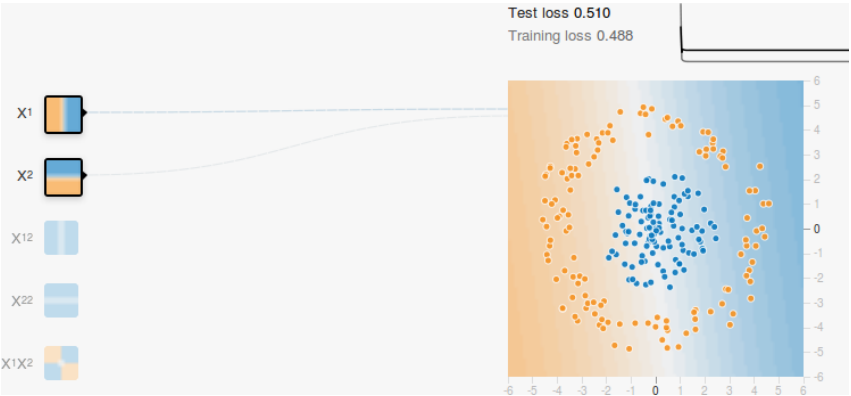
In fact: $1\tanh(-0.43x_1 - 0.89x_2 + 2.0)$
$+1\tanh(-0.67x_1 + 0.89x_2 + 2.1)$
$+1\tanh(1.4x_1 - 0.067x_2 + 2.3) - \pi/2 > 0$

Perfect Features

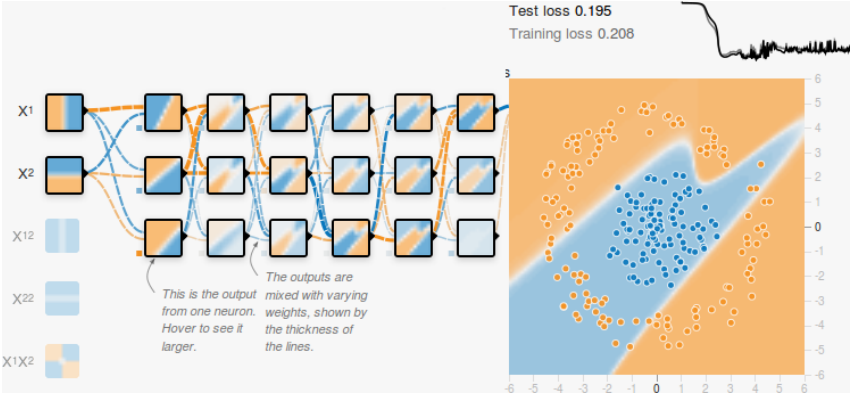


$$1x_1^2 + 1x_2^2 - 1 < 0$$

Bad Features & Low Depth



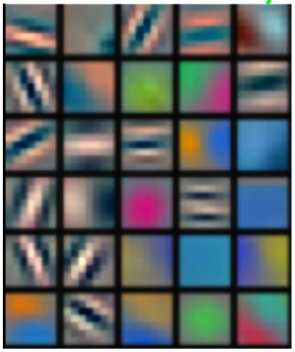
Too Complex NN Fails to Learn

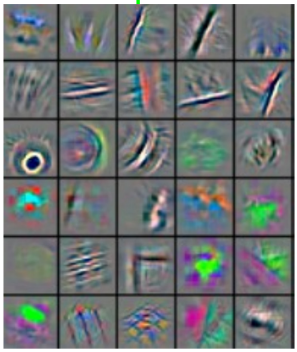


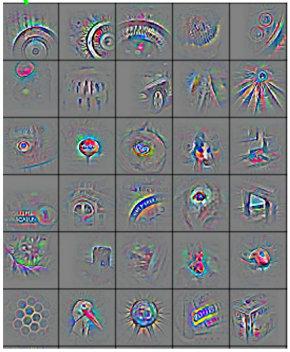
Deep NNs for Image Classification

- It's deep if it has more than one stage of non-linear feature transformation

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier







Processing Text with NNs

- Map each word to a vector of 0s and 1s ("1-hot repr."):

$$\text{cat} \mapsto (0, 0, \ldots, 0, 1, 0, \ldots, 0)$$

- Sentence is then a matrix:

		the	cat	is	on	the	mat
↑	a	0	0	0	0	0	0
	about	0	0	0	0	0	0
	...	...	...	...	...	...	...
	cat	0	**1**	0	0	0	0
	...	...	...	...	...	...	...
	is	0	0	**1**	0	0	0
	...	...	...	...	...	...	...
	the	**1**	0	0	0	**1**	0
	...	...	...	...	...	...	...
↓	zebra	0	0	0	0	0	0

Processing Text with NNs

- Map each word to a vector of 0s and 1s ("1-hot repr."):

$$\text{cat} \mapsto (0, 0, \ldots, 0, 1, 0, \ldots, 0)$$

- Sentence is then a matrix:

		the	cat	is	on	the	mat
↑	a	0	0	0	0	0	0
	about	0	0	0	0	0	0
	...	...	...	...	...	...	...
	cat	0	**1**	0	0	0	0
	...	...	...	...	...	...	...
	is	0	0	**1**	0	0	0
	...	...	...	...	...	...	...
	the	**1**	0	0	0	**1**	0
	...	...	...	...	...	...	...
↓	zebra	0	0	0	0	0	0

Vocabulary size:
1.3M English
2.2M Czech

Processing Text with NNs

- Map each word to a vector of 0s and 1s ("1-hot repr."):

$$\text{cat} \mapsto (0, 0, \ldots, 0, 1, 0, \ldots, 0)$$

- Sentence is then a matrix:

		the	cat	is	on	the	mat
↑	a	0	0	0	0	0	0
	about	0	0	0	0	0	0
	...	...	...	...	...	...	...
	cat	0	**1**	0	0	0	0
Vocabulary size:	...	...	...	...	...	...	...
1.3M English	is	0	0	**1**	0	0	0
2.2M Czech	...	...	...	...	...	...	...
	the	**1**	0	0	0	**1**	0
	...	...	...	...	...	...	...
↓	zebra	0	0	0	0	0	0

Main drawback: No relations, all words equally close/far.

Solution: Word Embeddings

- Idea: Map each word to a dense vector.
- Result: 300–2000 dimensions instead of 1–2M.
  - The dimensions have no clear interpretation.
- The "embedding" is the mapping.
  - Technically, the first layer of NNs for NLP is the matrix that maps 1-hot input to the first layer.
- Embeddings are trained for each particular task.
  - Sentence classification (sentiment analysis, etc.)
  - Neural language modelling.
  - The famous word2vec (Mikolov et al., 2013):
    - CBOW: Predict the word from its four neighbours.
    - Skip-gram: Predict likely neighbours given the word.
  - End-to-end neural MT.

Further Compression: Sub-Words

- SMT struggled with productive morphology (>1M wordforms).

  nejneobhodpodařovávatelnějšími, Donaudampfschifffahrtsgesellschaftskapitän

- NMT can handle only 30–80k dictionaries.

⇒ Resort to sub-word units.

Orig	český politik svezl migranty
Syllables	čes ký ␣ po li tik ␣ sve zl ␣ mig ran ty
Morphemes	česk ý ␣ politik ␣ s vez l ␣ migrant y
Char Pairs	če sk ý ␣ po li ti k ␣ sv ez l ␣ mi gr an ty
Chars	č e s k ý ␣ p o l i t i k ␣ s v e z l ␣ m i g r a n t y
BPE 30k	český politik s@@ vez@@ l mi@@ granty

BPE (Byte-Pair Encoding) uses n most common substrings (incl. frequent words).

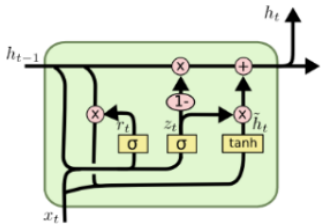
Variable-Length Inputs

Variable-length input can be handled by recurrent NNs:

- Reading one input symbol at a time.
  - The same (trained) transformation used every time.
- Unroll in time (up to a fixed length limit).

Tricks needed to train (to avoid "vanishing gradients"):

- LSTM, Long Short-Term Memory Cells (Hochreiter and Schmidhuber, 1997).
- GRU, Gated Recurrent Unit Cells (Chung et al., 2014).



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

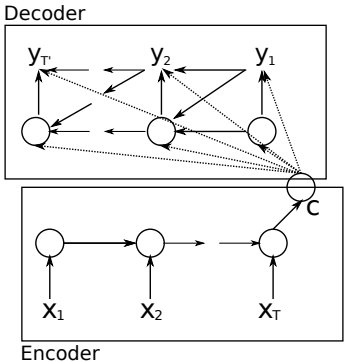
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

NNs as Translation Model in SMT

Cho et al. (2014) proposed:

- encoder-decoder architecture and
- GRU unit (name given later by Chung et al. (2014))
- to score variable-length phrase pairs in PBMT.

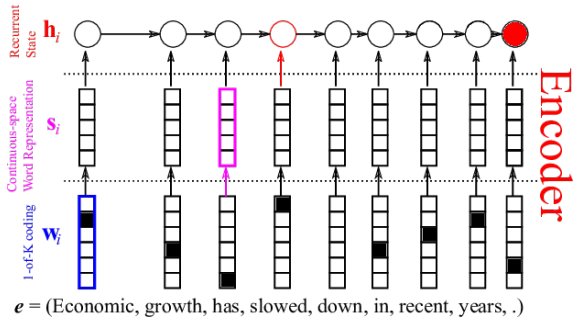


NMT: Sequence to Sequence

Sutskever et al. (2014) use:

- LSTM RNN encoder-decoder
- to consume
  and produce variable-length sentences.

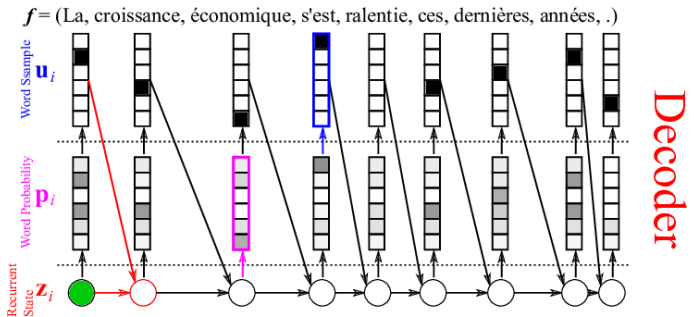
First the Encoder:



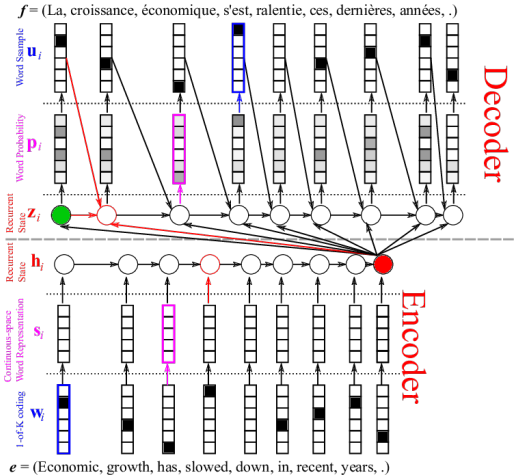
Then the Decoder

Remember: $p(e_1^I|f_1^J) = p(e_1|f_1^J) \cdot p(e_2|e_1, f_1^J) \cdot p(e_3|e_2, e_1, f_1^J) \ldots$

- Again RNN, producing one word at a time.
- The produced word fed back into the network.
  - (Word embeddings in the target language used here.)



Encoder-Decoder Architecture



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

Ultimate Goal of SMT vs. NMT

Goal of "classical" SMT (e.g. PBMT):

Find **minimum translation units** $\sim$ graph partitions:

- such that they are frequent across many sentence pairs.
- without imposing (too hard) constraints on reordering.
- in an unsupervised fashion.

Goal of neural MT:

Avoid minimum translation units. Find NN architecture that

- Reads input in as original form as possible.
- Produces output in as final form as possible.
- Can be optimized end-to-end in practice.

Summary of the Lecture

- Statistical MT chooses the most probable sentence:
  $\hat{e}_1^{\hat{I}} = \text{argmax}_{I,e_1^I}\, p(e_1^I|f_1^J)$
- The probability modelled in Bayes' or Log-Linear decomposition:
  $\hat{e}_1^{\hat{I}} = \text{argmax}_{I,e_1^I}\, p(f_1^J|e_1^I)p(e_1^I)$
  or $\hat{e}_1^{\hat{I}} = \text{argmax}_{I,e_1^I}\, \exp(\sum_{m=1}^{M} \lambda_m h_m(e_1^I, f_1^J))$
- Phrase-Based MT models $p(f_1^J|e_1^I)$ as product of phrase translation probabilities in a segmentation s_1^K: $\prod_{k=1}^{K} p(\tilde{f}_k|\tilde{e}_k)$
- Other (ling.-motivated) decompositions or features possible.
- Probabilities estimated from data (parallel/monolingual).
- Neural MT predict word by word; "just a clever LM".
  - Sub-word units, word embeddings, encoder-decoder.

References

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural Comput., 9(8):1735–1780, November.

Adam Lopez. 2009. Translation as weighted deduction. In Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), pages 532–540, Athens, Greece, March. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781.

Franz Joseph Och. 2002. Statistical Machine Translation: From Single-Word Models to Alignment Templates. Ph.D. thesis, RWTH Aachen University.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In Advances in neural information processing systems, pages 3104–3112.