Word Embeddings, CRF, CTC

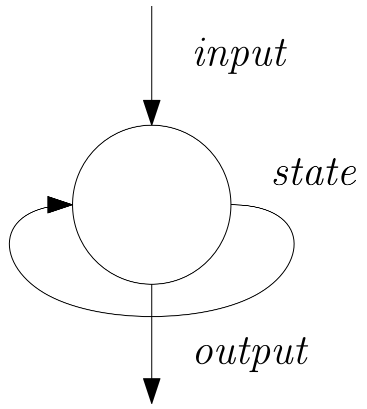
Milan Straka

April 20, 2020

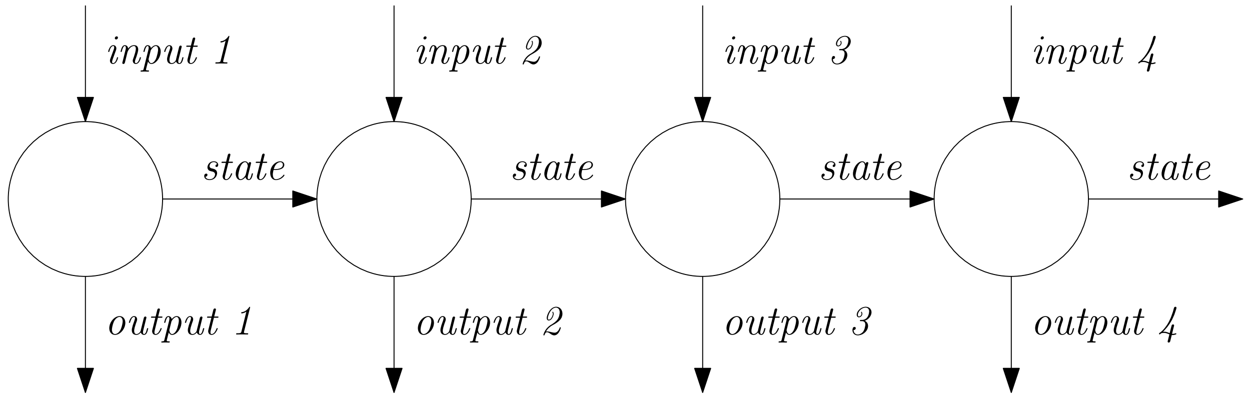
EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

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

Single RNN cell



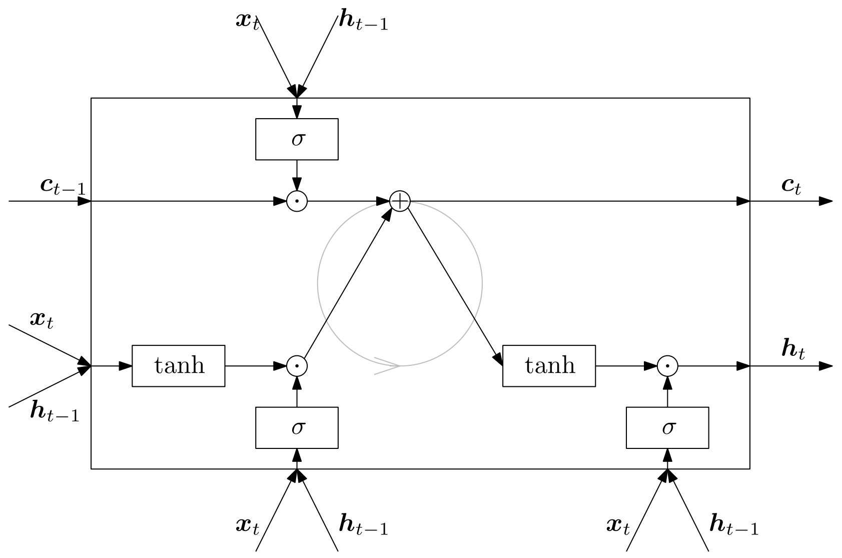
Unrolled RNN cells



Long Short-Term Memory

Later in Gers, Schmidhuber & Cummins (1999) a possibility to *forget* information from memory cell $\boldsymbol{c}_t$ was added.

$$\begin{aligned}
\boldsymbol{i}_t &\leftarrow \sigma(\boldsymbol{W}^i\boldsymbol{x}_t + \boldsymbol{V}^i\boldsymbol{h}_{t-1} + \boldsymbol{b}^i)\\
\boldsymbol{f}_t &\leftarrow \sigma(\boldsymbol{W}^f\boldsymbol{x}_t + \boldsymbol{V}^f\boldsymbol{h}_{t-1} + \boldsymbol{b}^f)\\
\boldsymbol{o}_t &\leftarrow \sigma(\boldsymbol{W}^o\boldsymbol{x}_t + \boldsymbol{V}^o\boldsymbol{h}_{t-1} + \boldsymbol{b}^o)\\
\boldsymbol{c}_t &\leftarrow \boldsymbol{f}_t \cdot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tanh(\boldsymbol{W}^y\boldsymbol{x}_t + \boldsymbol{V}^y\boldsymbol{h}_{t-1} + \boldsymbol{b}^y)\\
\boldsymbol{h}_t &\leftarrow \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t)
\end{aligned}$$



Gated Recurrent Unit

Gated recurrent unit (GRU) was proposed by Cho et al. (2014) as a simplification of LSTM. The main differences are

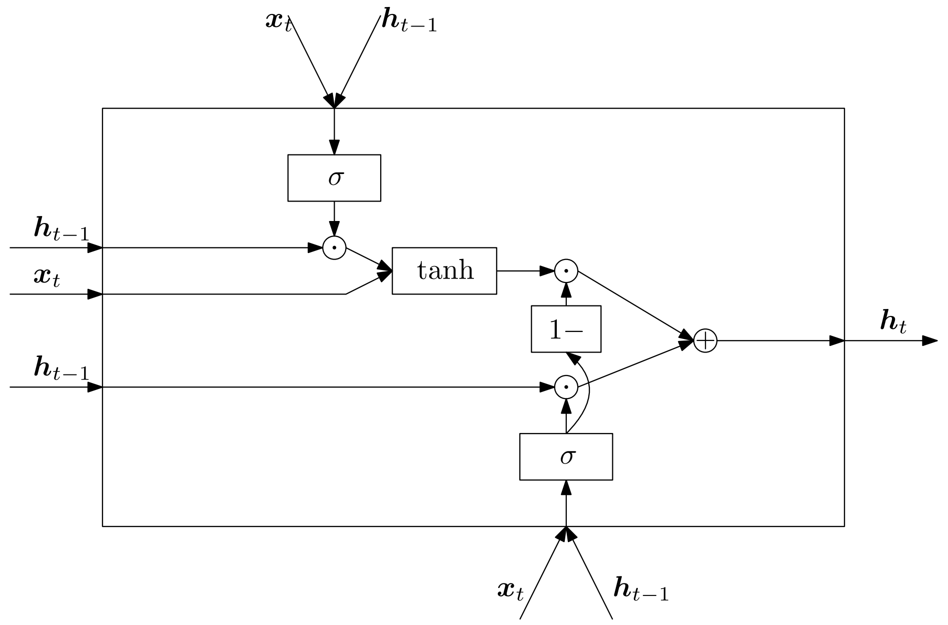
- no memory cell
- forgetting and updating tied together

$$\boldsymbol{r}_t \leftarrow \sigma(\boldsymbol{W}^r \boldsymbol{x}_t + \boldsymbol{V}^r \boldsymbol{h}_{t-1} + \boldsymbol{b}^r)$$

$$\boldsymbol{u}_t \leftarrow \sigma(\boldsymbol{W}^u \boldsymbol{x}_t + \boldsymbol{V}^u \boldsymbol{h}_{t-1} + \boldsymbol{b}^u)$$

$$\hat{\boldsymbol{h}}_t \leftarrow \tanh(\boldsymbol{W}^h \boldsymbol{x}_t + \boldsymbol{V}^h (\boldsymbol{r}_t \cdot \boldsymbol{h}_{t-1}) + \boldsymbol{b}^h)$$

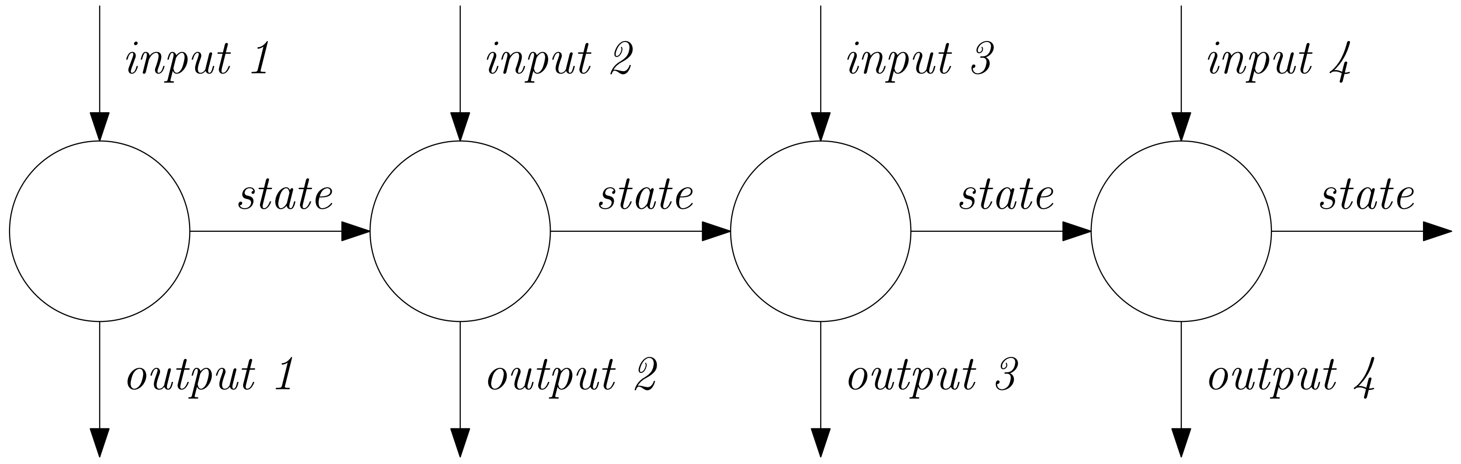
$$\boldsymbol{h}_t \leftarrow \boldsymbol{u}_t \cdot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \cdot \hat{\boldsymbol{h}}_t$$



Basic RNN Applications

Sequence Element Representation

Create output for individual elements, for example for classification of the individual elements.

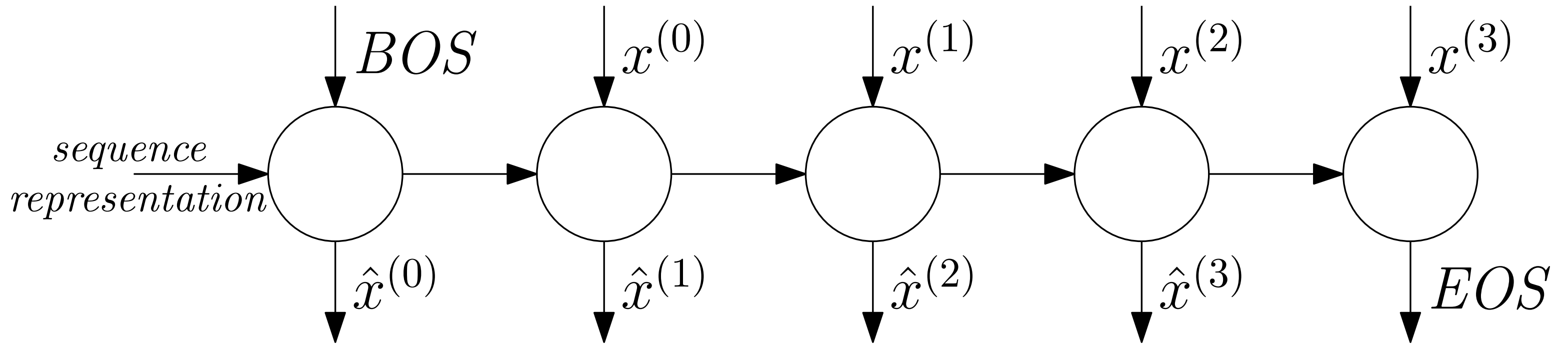


Sequence Representation

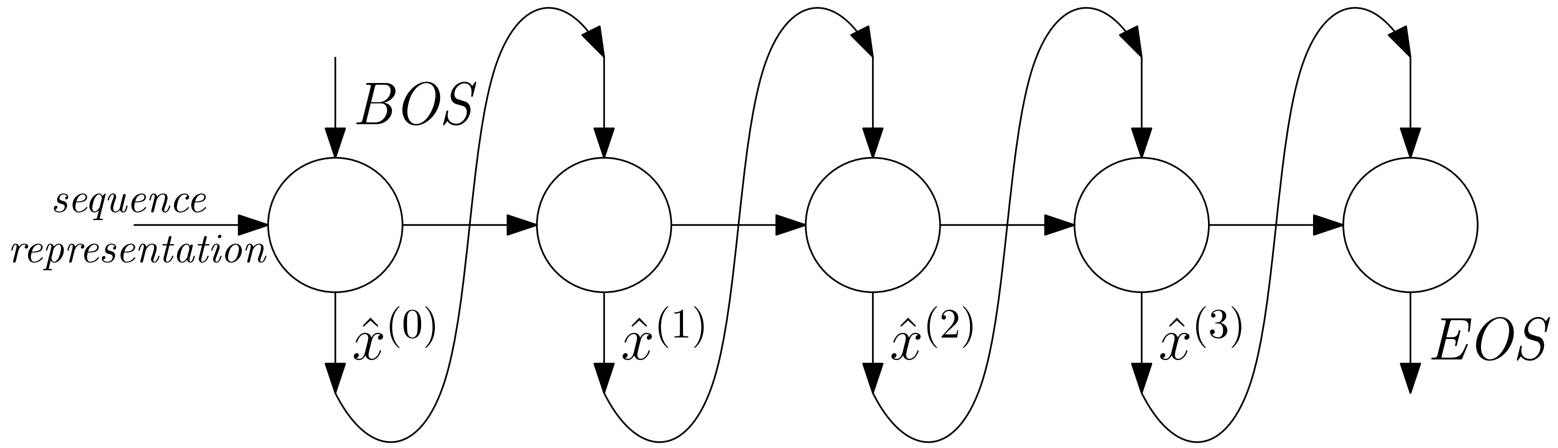
Generate a single output for the whole sequence (either the last output of the last state).

Sequence Prediction

During training, predict next sequence element.



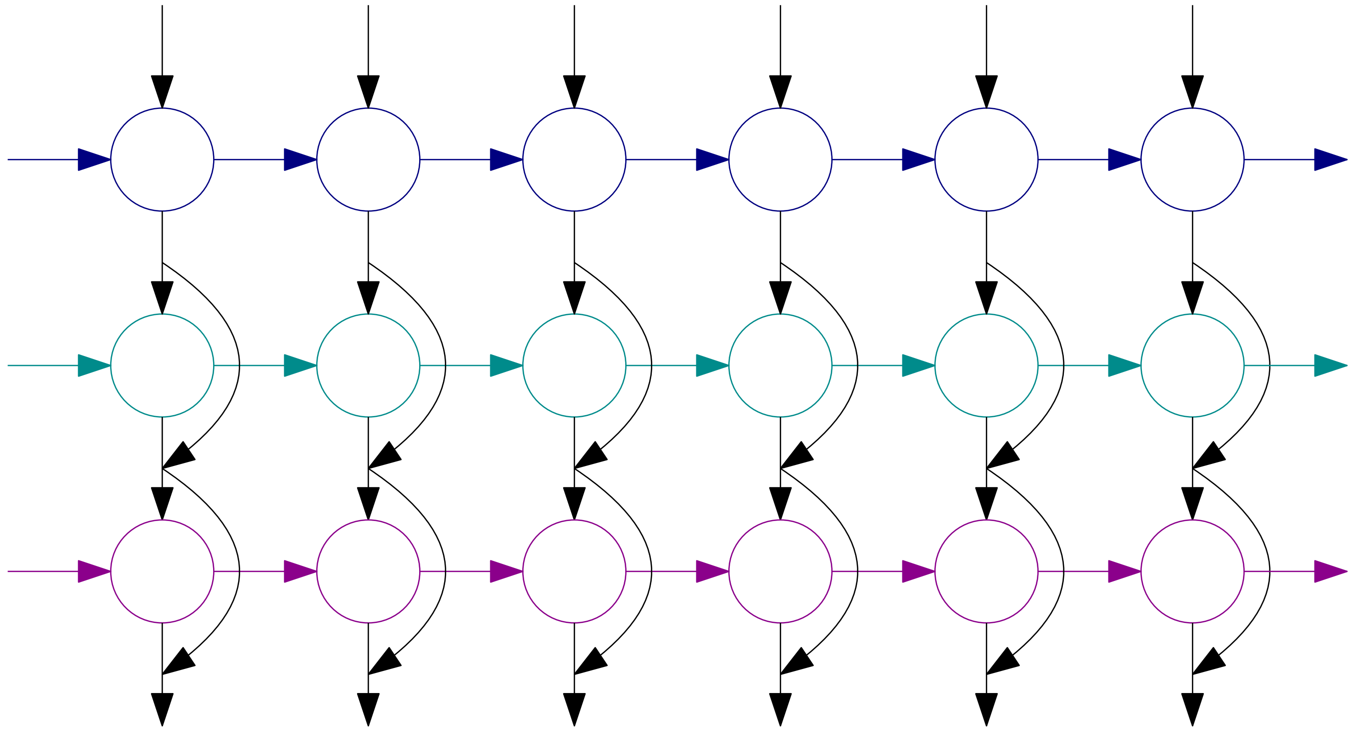
During inference, use predicted elements as further inputs.



Multilayer RNNs

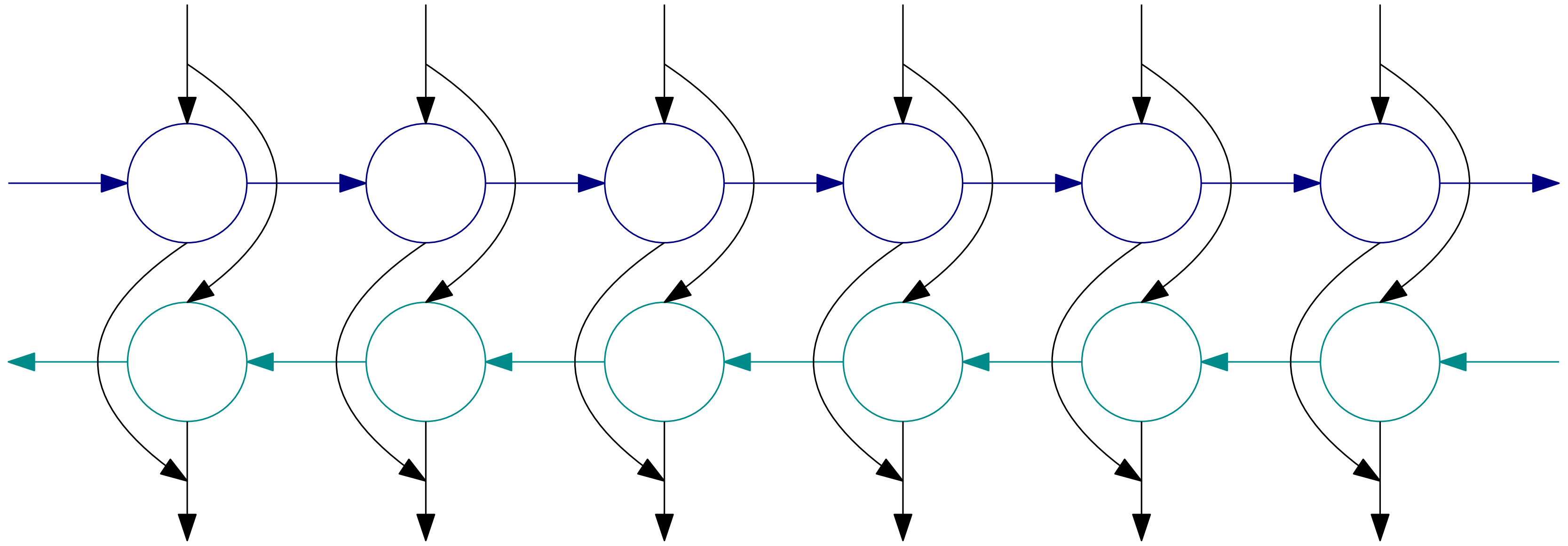
We might stack several layers of recurrent neural networks. Usually using two or three layers gives better results than just one.

In case of multiple layers, residual connections usually improve results. Because dimensionality has to be the same, they are usually applied from the second layer.



Bidirectional RNN

To consider both the left and right contexts, a *bidirectional* RNN can be used, which consists of parallel application of a *forward* RNN and a *backward* RNN.



The outputs of both directions can be either *added* or *concatenated*. Even if adding them does not seem very intuitive, it does not increase dimensionality and therefore allows residual connections to be used in case of multilayer bidirectional RNN.

Word Embeddings

We might represent *words* using one-hot encoding, considering all words to be independent of each other.

However, words are not independent – some are more similar than others.

Ideally, we would like some kind of similarity in the space of the word representations.

Distributed Representation

The idea behind distributed representation is that objects can be represented using a set of common underlying factors.

We therefore represent words as fixed-size *embeddings* into $\mathbb{R}^d$ space, with the vector elements playing role of the common underlying factors.

These embeddings are initialized randomly and trained together with the rest of the network.

Word Embeddings

The word embedding layer is in fact just a fully connected layer on top of one-hot encoding. However, it is not implemented in that way.

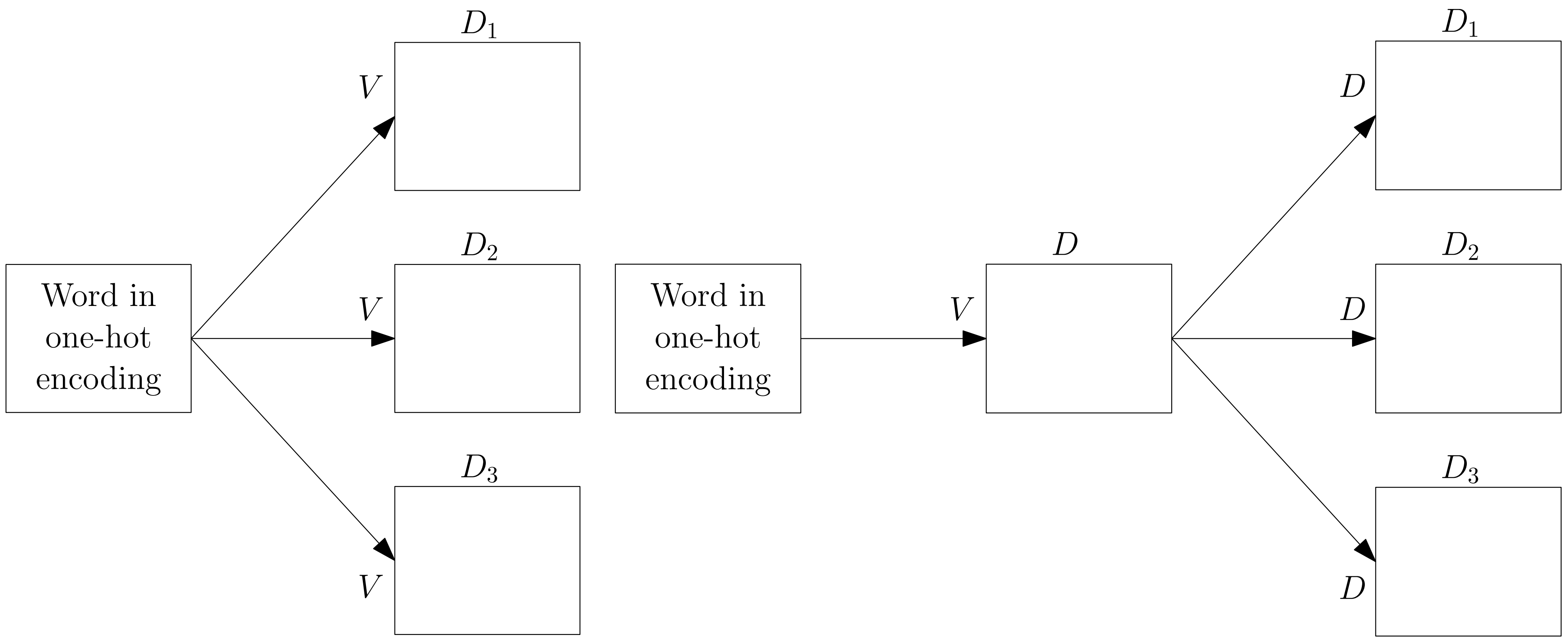
Instead, a so-called *embedding* layer is used, which is much more efficient. When a matrix is multiplied by an one-hot encoded vector (all but one zeros and exactly one 1), the row corresponding to that 1 is selected, so the embedding layer can be implemented only as a simple lookup.

In TensorFlow, the embedding layer is available as

```
tf.keras.layers.Embedding(input_dim, output_dim)
```

Word Embeddings

Even if the embedding layer is just a fully connected layer on top of one-hot encoding, it is important that this layer is *shared* across the whole network.



Recurrent Character-level WEs

In order to handle words not seen during training, we could find a way to generate a representation from the word **characters**.

A possible way to compose the representation from individual characters is to use RNNs – we embed *characters* to get character representation, and then use a RNN to produce the representation of a whole *sequence of characters*.

Usually, both forward and backward directions are used, and the resulting representations are concatenated/added.

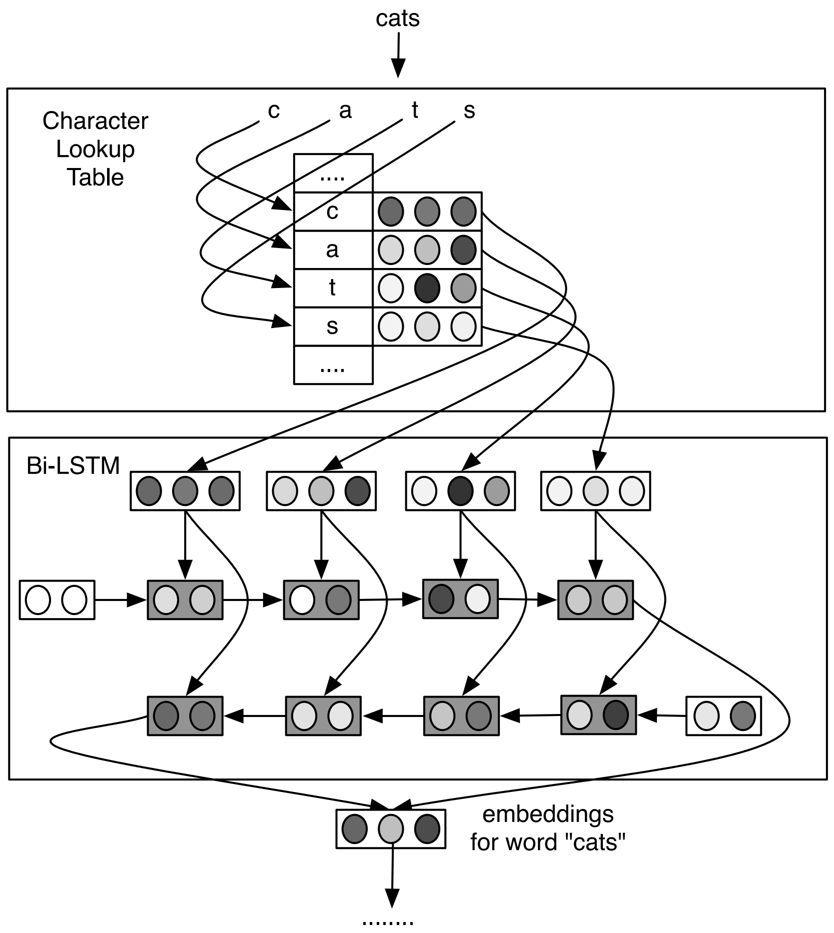


Figure 1 of paper "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation",

Convolutional Character-level WEs

Alternatively, 1D convolutions might be used.

Assume we use a 1D convolution with kernel size 3. It produces a representation for every input word trigram, but we need a representation of the whole word. To that end, we use *global max-pooling* – using it has an interpretable meaning, where the kernel is a *pattern* and the activation after the maximum is a level of a highest match of the pattern anywhere in the word.

Kernels of varying sizes are usually used (because it makes sense to have patterns for unigrams, bigrams, trigrams, …) – for example, 25 filters for every kernel size $(1, 2, 3, 4, 5)$ might be used.

Lastly, authors employed a highway layer after the convolutions, improving the results (compared to not using any layer or using a fully connected one).

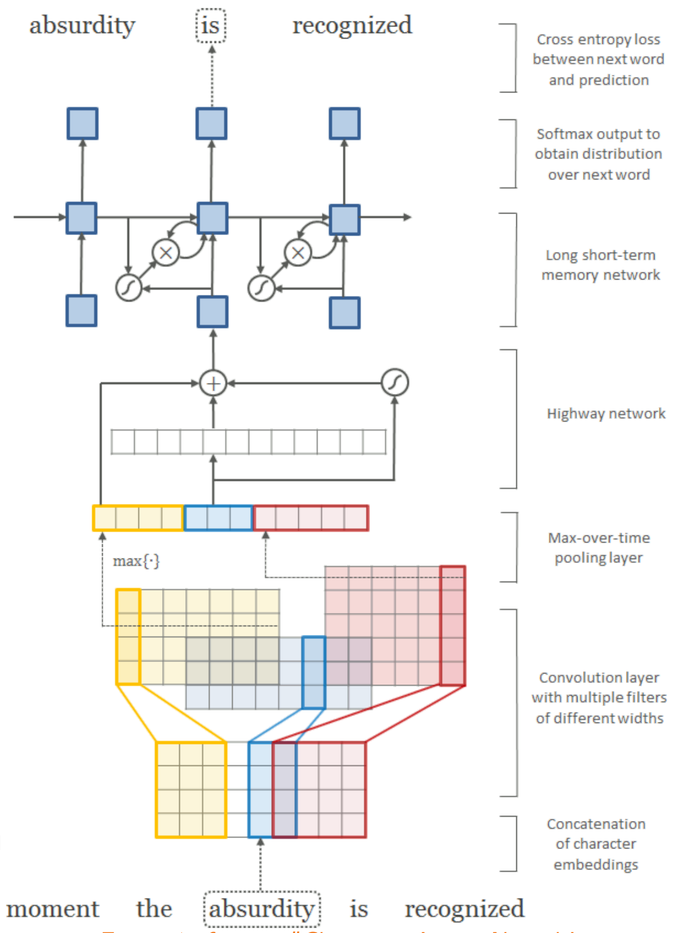

absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

Figure 1 of paper "Character-Aware Neural Language Models", https://arxiv.org/abs/1508.06615.

Character-level WE Implementation

Training

- Generate unique words per batch.
- Process the unique words in the batch.
- Copy the resulting embeddings suitably in the batch.

Inference

- We can cache character-level word embeddings during inference.

NLP Processing with CLEs

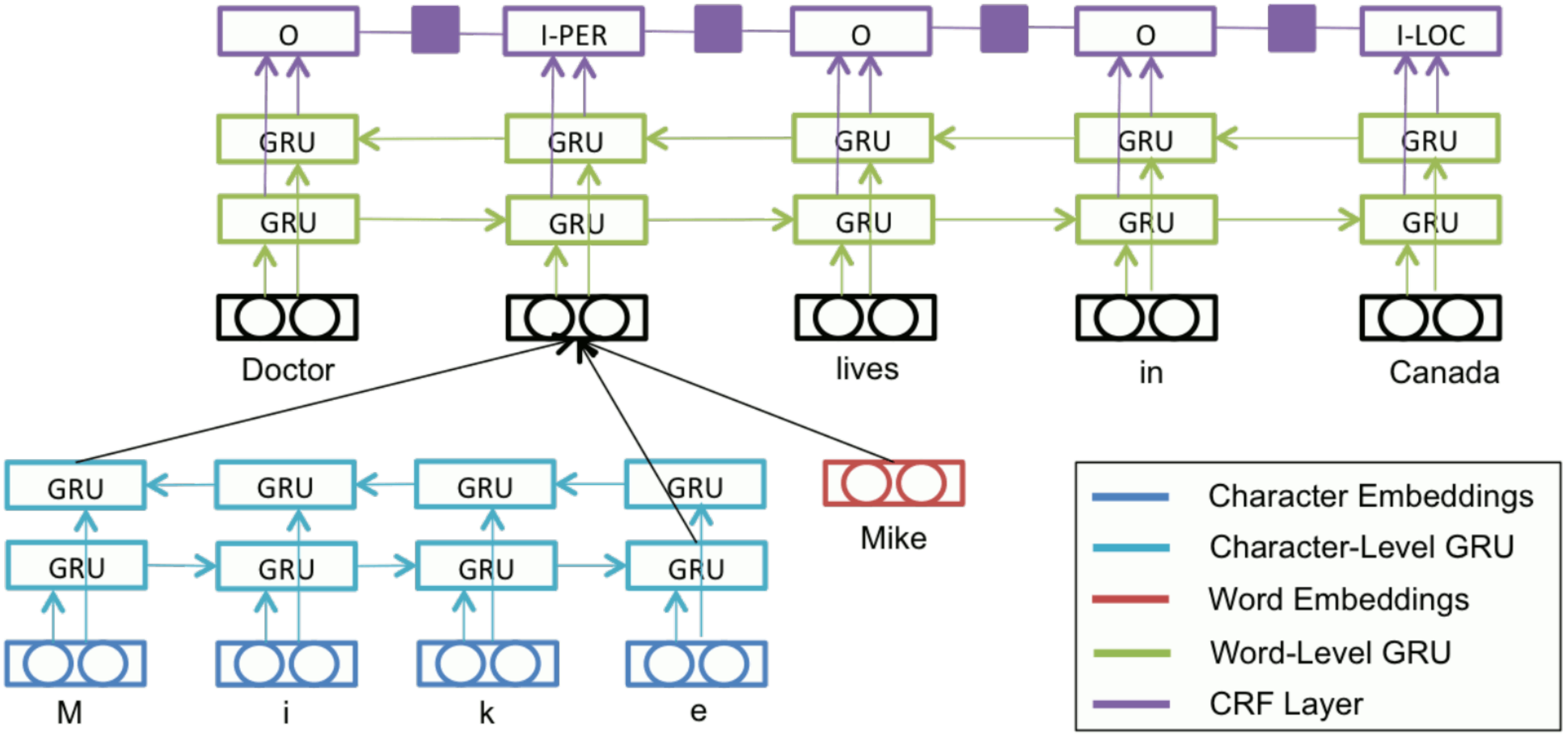
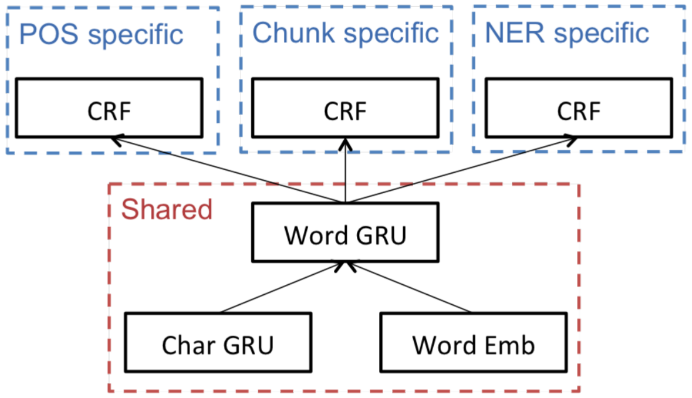
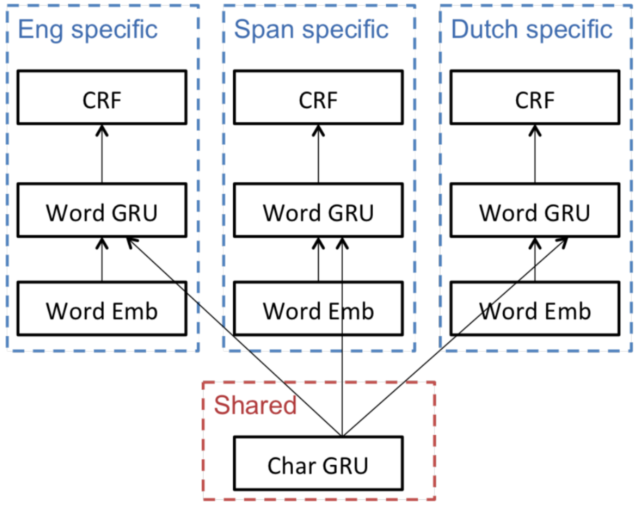


Figure 1 of paper "Multi-Task Cross-Lingual Sequence Tagging from Scratch", https://arxiv.org/abs/1603.06270.

NLP Multitask Learning



(a) Multi-Task Joint Training



(b) Cross-Lingual Joint Training

Figure 2 of paper "Multi-Task Cross-Lingual Sequence Tagging from Scratch", https://arxiv.org/abs/1603.06270.

NLP Multitask Learning

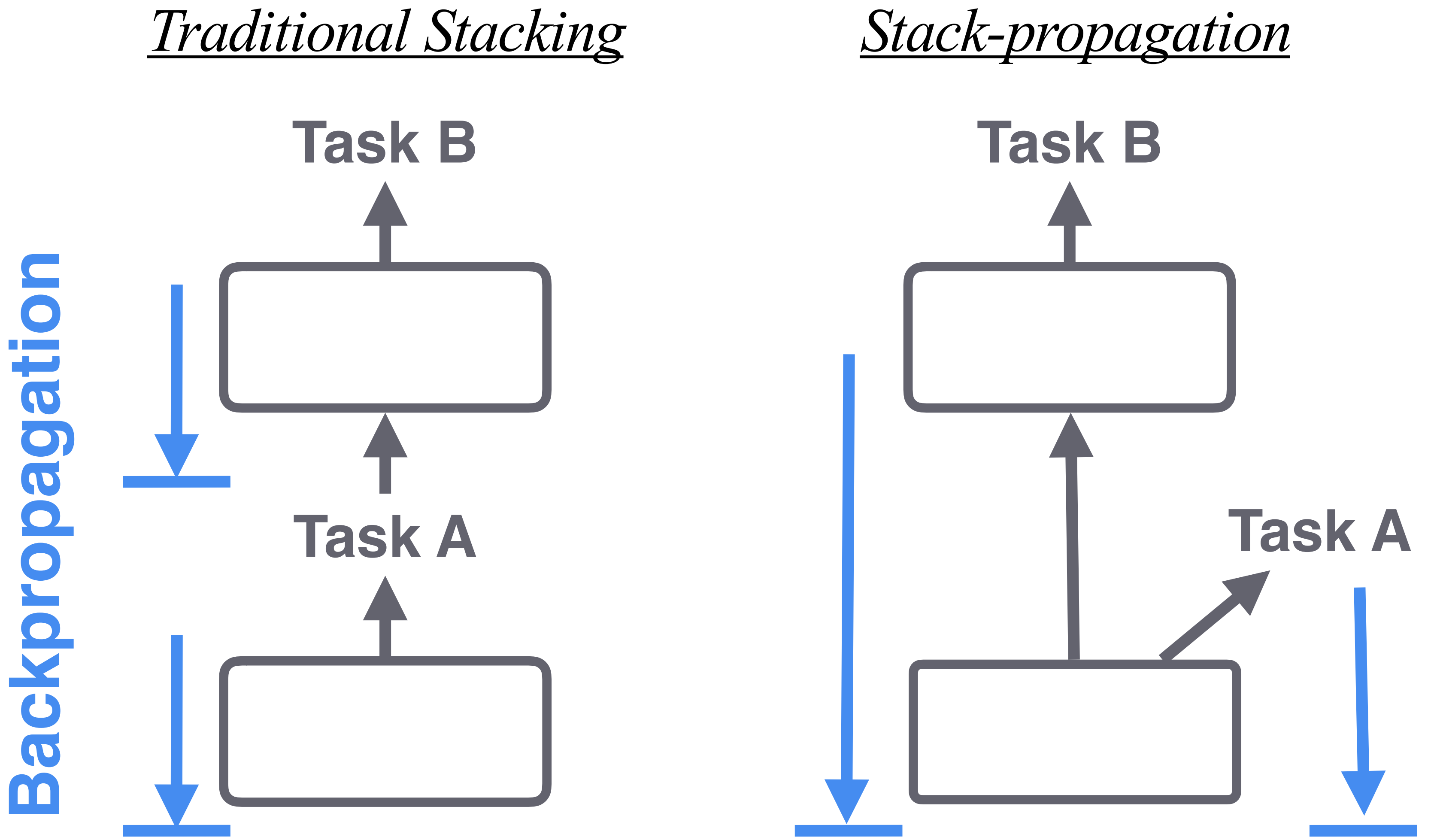


Figure 1 of paper "Stack-propagation: Improved Representation Learning for Syntax", https://arxiv.org/abs/1603.06598.

NLP Multitask Learning

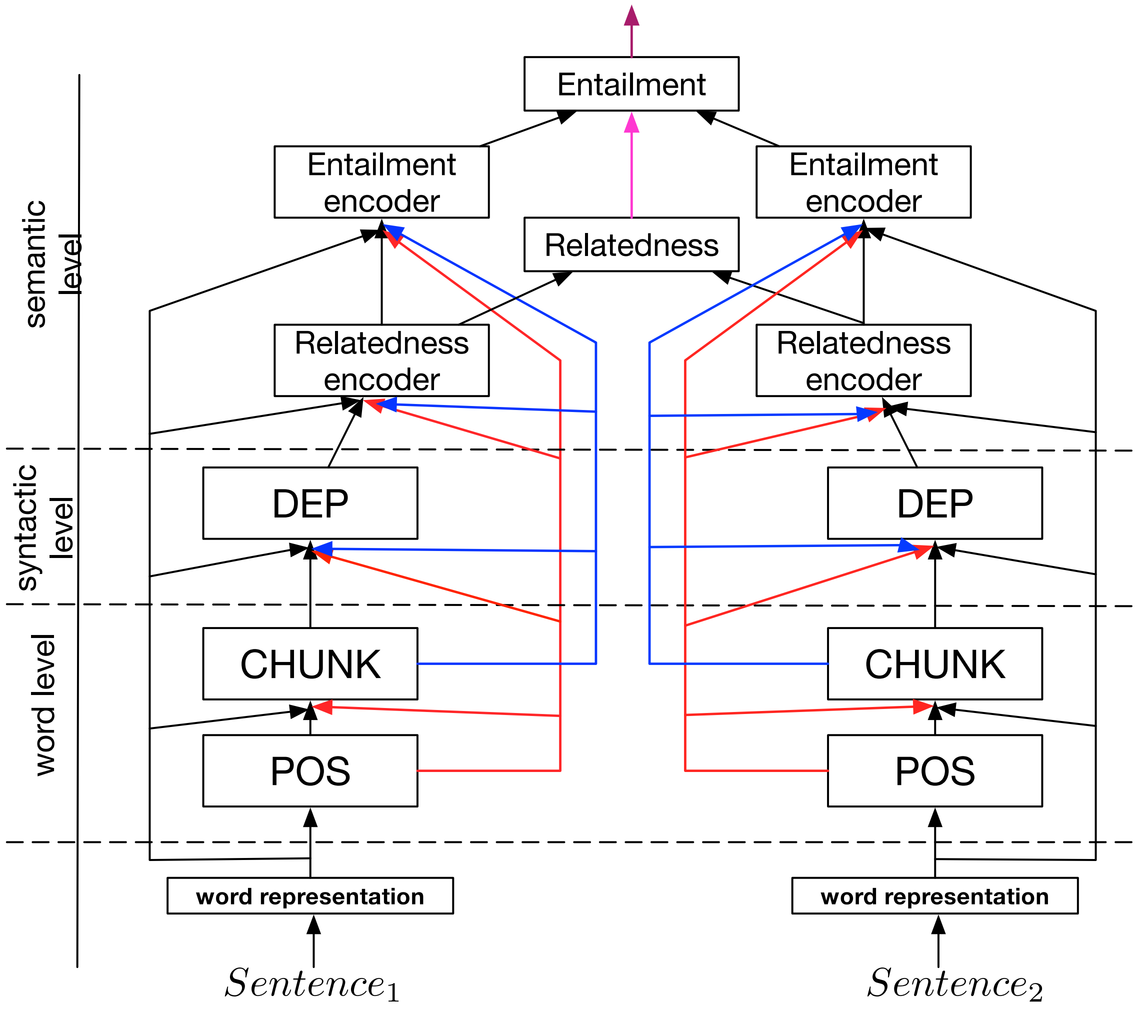


Figure 1 of paper "A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks", https://arxiv.org/abs/1611.01587.

Structured Prediction

Structured Prediction

Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Predicting each sequence element independently models the distribution $P(y_i | \boldsymbol{X})$.

$$\begin{tikzcd}
\boldsymbol{x}_1 \arrow[dd] & \boldsymbol{x}_2 \arrow[dd] & \boldsymbol{x}_3 \arrow[dd] & \cdots & \boldsymbol{x}_N \arrow[dd] \\
 & & & \cdots & \\
y_1 & y_2 & y_3 & \cdots & y_N
\end{tikzcd}$$

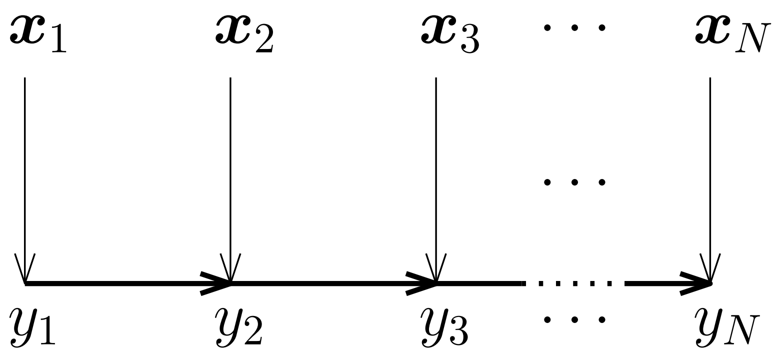
However, there may be dependencies among the y_i themselves, which is difficult to capture by independent element classification.

Maximum Entropy Markov Models

We might model the dependencies by assuming that the output sequence is a Markov chain, and model it as

$$P(y_i | \boldsymbol{X}, y_{i-1}).$$

Each label would be predicted by a softmax from the hidden state and *the previous label*.



The decoding can be then performed by a dynamic programming algorithm.

Maximum Entropy Markov Models

However, MEMMs suffer from a so-called *label bias* problem. Because the probability is factorized, each $P(y_i | \boldsymbol{X}, y_{i-1})$ is a distribution and **must sum to one**.

Imagine there was a label error during prediction. In the next step, the model might "realize" that the previous label has very low probability of being followed by any label – however, it cannot express this by setting the probability of all following labels low, it has to "conserve the mass".

Conditional Random Fields

Let $G = (V, E)$ be a graph such that $\boldsymbol y$ is indexed by vertices of G. Then $(\boldsymbol X, \boldsymbol y)$ is a **conditional random field**, if the random variables $\boldsymbol y$ conditioned on $\boldsymbol X$ obey the Markov property with respect to the graph, i.e.,

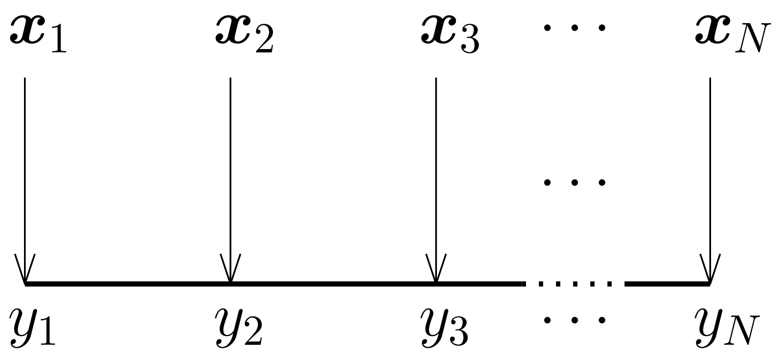
$$P(y_i|\boldsymbol X, y_j, i \neq j) = P(y_i|\boldsymbol X, y_j\ \forall j : (i, j) \in E).$$

By a *fundamental theorem of random fields*, the density of a conditional random field can be factorized over the cliques of the graph G:

$$P(\boldsymbol y|\boldsymbol X) = \prod_{\text{clique } C \text{ of } G} P(\boldsymbol y_C|\boldsymbol X).$$

Linear-Chain Conditional Random Fields (CRF)

Usually often assume that dependencies of $\boldsymbol{y}$, conditioned on $\boldsymbol{X}$, form a chain.



Then, the cliques are *nodes* and *edges*, and we usually factorize the probability as:

$$P(\boldsymbol{y}|\boldsymbol{X}) \propto \exp\Big(\sum_{i=1}^N \log P(y_i|\boldsymbol{x}_i) + \sum_{i=2}^N \log P(y_i, y_{i-1})\Big).$$

Linear-Chain Conditional Random Fields (CRF)

Linear-chain Conditional Random Field, usually abbreviated only to CRF, acts as an output layer. It can be considered an extension of a softmax – instead of a sequence of independent softmaxes, CRF is a sentence-level softmax, with additional weights for neighboring sequence elements.

$$s(\boldsymbol X, \boldsymbol y; \boldsymbol\theta, \boldsymbol A) = \sum_{i=1}^N \big(\boldsymbol A_{y_{i-1},y_i} + f_{\boldsymbol\theta}(y_i|\boldsymbol X)\big)$$

$$p(\boldsymbol y|\boldsymbol X) = \operatorname{softmax}_{\boldsymbol z \in Y^N}\big(s(\boldsymbol X, \boldsymbol z)\big)_{\boldsymbol y}$$

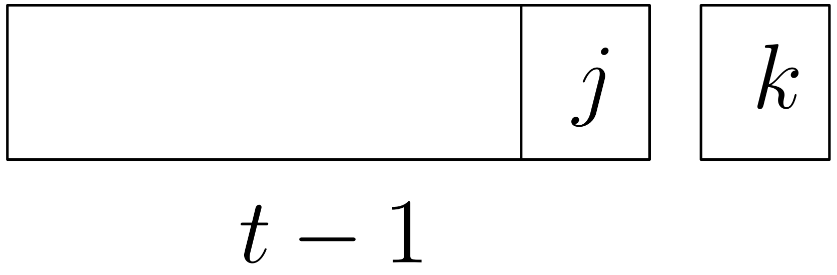
$$\log p(\boldsymbol y|\boldsymbol X) = s(\boldsymbol X, \boldsymbol y) - \operatorname{logadd}_{\boldsymbol z \in Y^N}(s(\boldsymbol X, \boldsymbol z))$$

Linear-Chain Conditional Random Fields (CRF)

Computation

We can compute $p(\boldsymbol{y}|\boldsymbol{X})$ efficiently using dynamic programming. We denote $\alpha_t(k)$ the logarithmic probability of all t-element sequences with the last label y being k.

The core idea is the following:



$$\alpha_t(k) = f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X}) + \operatorname{logadd}_{j\in Y}(\alpha_{t-1}(j) + \boldsymbol{A}_{j,k}).$$

For efficient implementation, we use the fact that

$$\ln(a+b) = \ln a + \ln(1 + e^{\ln b - \ln a}).$$

Conditional Random Fields (CRF)

Inputs: Network computing $f_{\boldsymbol\theta}(y_t = k|\boldsymbol X)$, an unnormalized probability of output sequence element probability being k at time t.
Inputs: Transition matrix $\boldsymbol A \in \mathbb{R}^{Y\times Y}$.
Inputs: Input sequence $\boldsymbol X$ of length N, gold labeling $\boldsymbol y^g \in Y^N$.
Outputs: Value of $\log p(\boldsymbol y^g|\boldsymbol X)$.
Time Complexity: $\mathcal{O}(N \cdot Y^2)$.

```
• For $t = 1, \ldots, N$:
    ◦ For $k = 1, \ldots, Y$ :
        ▪ $\alpha_t(k) \leftarrow f_{\boldsymbol\theta}(y_t = k|\boldsymbol X)$
        ▪ If $t > 1$:
            ▪ For $j = 1, \ldots, Y$:
                ▪ $\alpha_t(k) \leftarrow \operatorname{logadd}(\alpha_t(k), \alpha_{t-1}(j) + \boldsymbol A_{j,k})$
• Return $\sum_{t=1}^N f_{\boldsymbol\theta}(y_t = y_t^g|\boldsymbol X) + \sum_{t=2}^N \boldsymbol A_{y_{t-1}^g, y_t^g} - \operatorname{logadd}_{k=1}^Y(\alpha_N(k))$
```

Conditional Random Fields (CRF)

Decoding

We can perform optimal decoding, by using the same algorithm, only replacing logadd with $\max$ and tracking where the maximum was attained.

Applications

CRF output layers are useful for *span labeling* tasks, like

- named entity recognition
- dialog slot filling

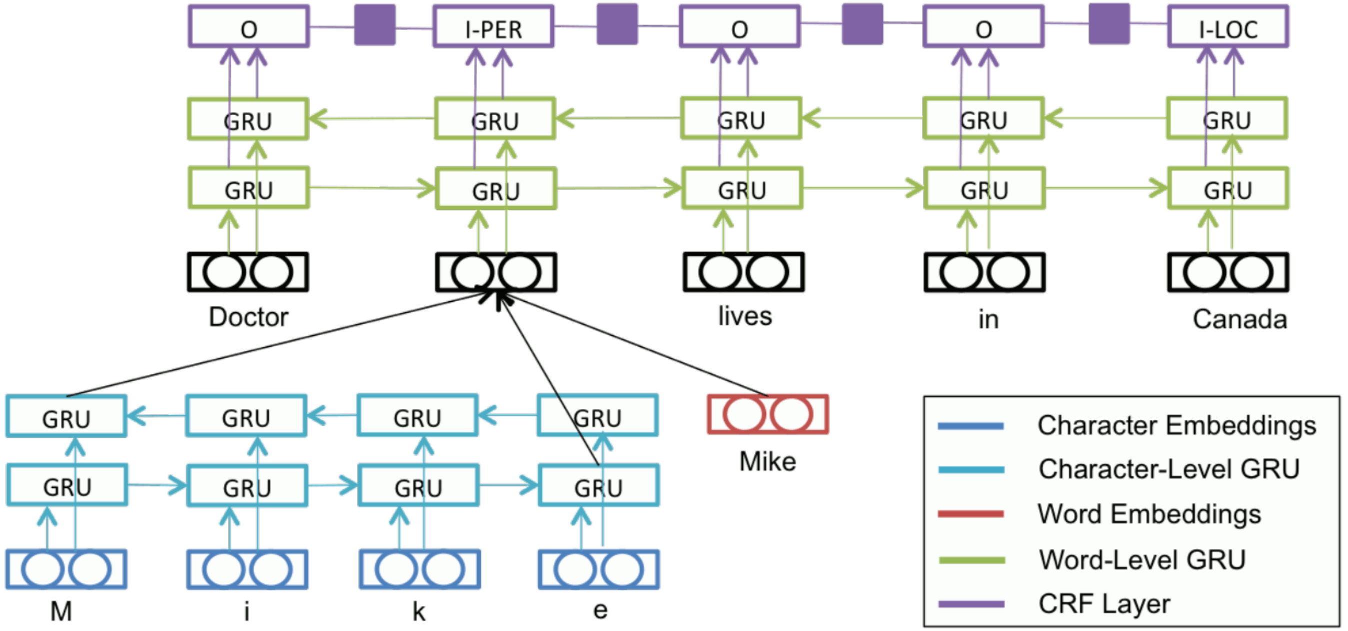


Figure 1 of paper "Multi-Task Cross-Lingual Sequence Tagging from Scratch", https://arxiv.org/abs/1603.06270.

Connectionist Temporal Classification

Let us again consider generating a sequence of $y_1, \ldots, y_M$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, but this time $M \leq N$ and there is no explicit alignment of $\boldsymbol{x}$ and y in the gold data.

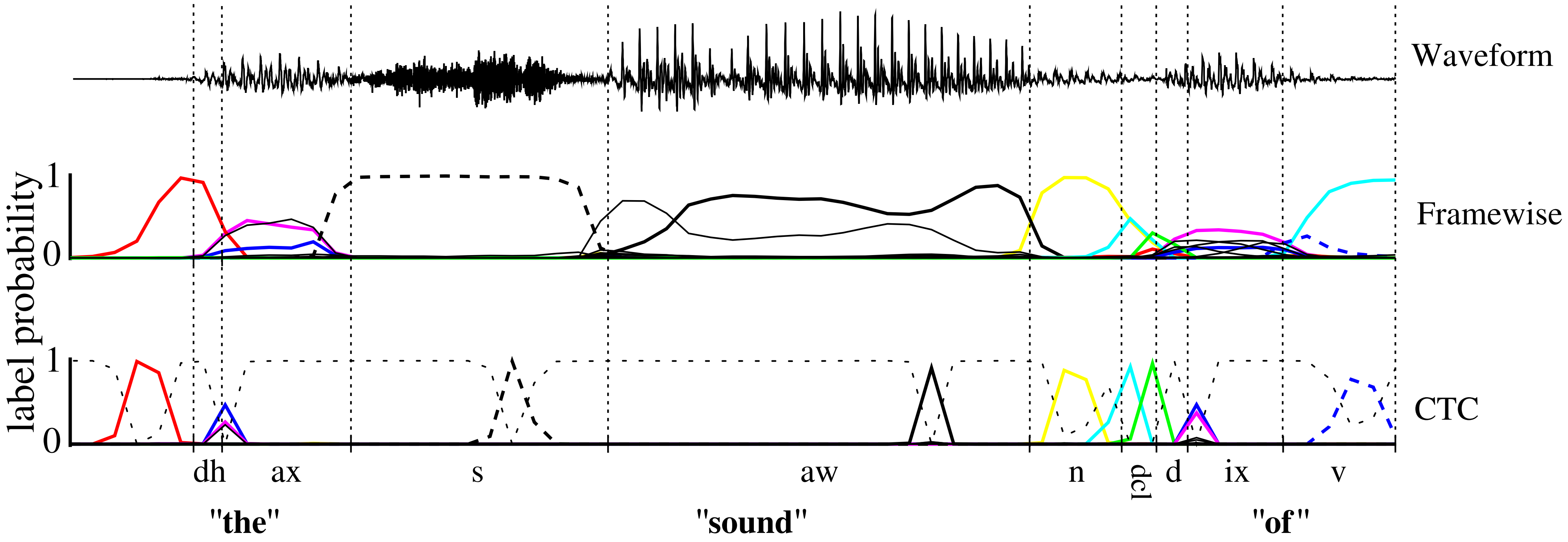


Figure 7.1 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

Connectionist Temporal Classification

We enlarge the set of output labels by a – (*blank*) and perform a classification for every input element to produce an *extended labeling*. We then post-process it by the following rules (denoted $\mathcal{B}$):

1. We remove neighboring symbols.
2. We remove the –.

Because the explicit alignment of inputs and labels is not known, we consider *all possible* alignments.

Denoting the probability of label l at time t as p_l^t, we define

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}:\mathcal{B}(\boldsymbol{\pi}_{1:t})=\boldsymbol{y}_{1:s}} \prod_{t'=1}^{t} p_{\boldsymbol{\pi}_{t'}}^{t'}.$$

CRF and CTC Comparison

In CRF, we normalize the whole sentences, therefore we need to compute unnormalized probabilities for all the (exponentially many) sentences. Decoding can be performed optimally.

In CTC, we normalize per each label. However, because we do not have explicit alignment, we compute probability of a labeling by summing probabilities of (generally exponentially many) extended labelings.

Connectionist Temporal Classification

Computation

When aligning an extended labeling to a regular one, we need to consider whether the extended labeling ends by a *blank* or not. We therefore define

$$\alpha_-^t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol\pi : \mathcal{B}(\boldsymbol\pi_{1:t}) = \boldsymbol y_{1:s}, \pi_t = -} \prod_{t'=1}^{t} p_{\boldsymbol\pi_{t'}}^{t'}$$

$$\alpha_*^t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol\pi : \mathcal{B}(\boldsymbol\pi_{1:t}) = \boldsymbol y_{1:s}, \pi_t \neq -} \prod_{t'=1}^{t} p_{\boldsymbol\pi_{t'}}^{t'}$$

and compute $\alpha^t(s)$ as $\alpha_-^t(s) + \alpha_*^t(s)$.

Connectionist Temporal Classification

Computation – Initialization

We initialize αs as follows:

- $\alpha_-^1(0) \leftarrow p_-^1$
- $\alpha_*^1(1) \leftarrow p_{y_1}^1$

Computation – Induction Step

We then proceed recurrently according to:

- $\alpha_-^t(s) \leftarrow p_-^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s)\right)$
- $\alpha_*^t(s) \leftarrow \begin{cases} p_{y_s}^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s-1) + \alpha_*^{t-1}(s-1)\right), \text{if } y_s \neq y_{s-1} \\ p_{y_s}^t\left(\alpha_*^{t-1}(s) + \alpha_-^{t-1}(s-1)\right), \text{if } y_s = y_{s-1} \end{cases}$

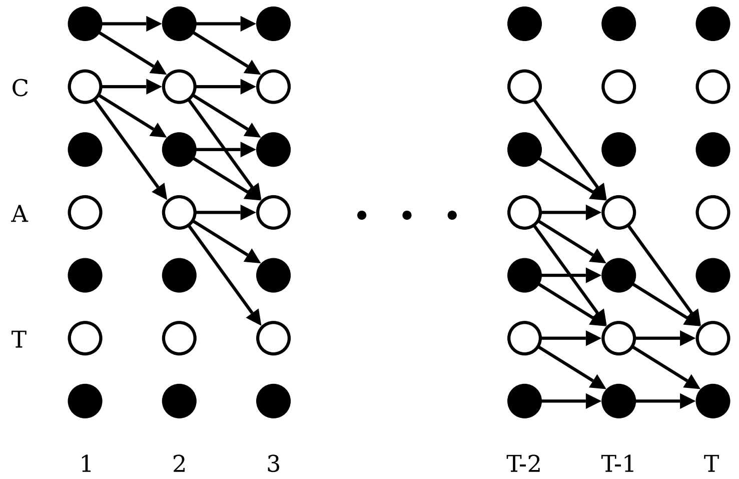
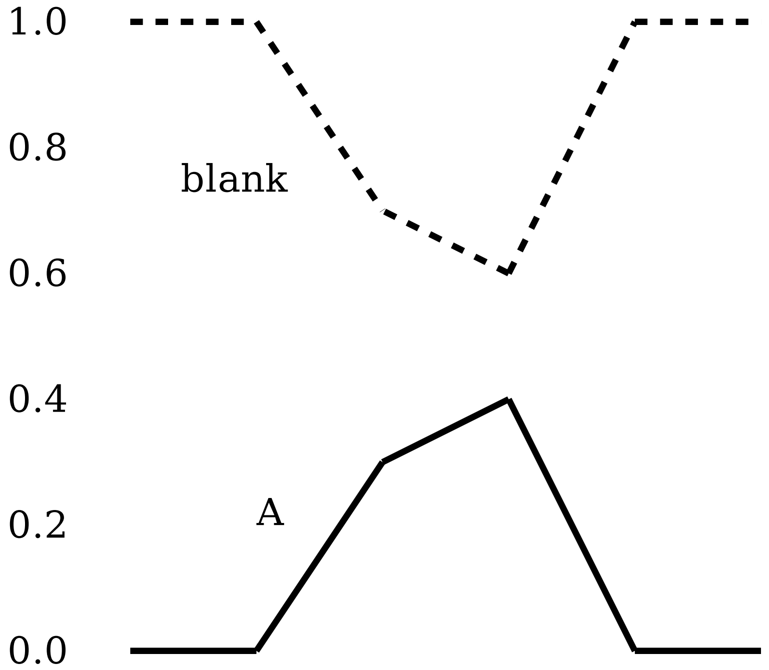


Figure 7.3 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

CTC Decoding

Unlike CRF, we cannot perform the decoding optimally.

The key observation is that while an optimal extended labeling can be extended into an optimal labeling of a larger length, the same does not apply to regular (non-extended) labeling. The problem is that regular labeling corresponds to many extended labelings, which are modified each in a different way during an extension of the regular labeling.



$$\begin{aligned} p(l=\text{blank}) &= p(\text{- -}) \\ &= 0.7*0.6 \\ &= 0.42 \end{aligned}$$

$$\begin{aligned} p(l=A) &= p(AA)+p(A\text{-})+p(\text{-}A) \\ &= 0.3*0.4 + 0.3*0.6 + 0.7*0.4 \\ &= 0.58 \end{aligned}$$

Figure 7.5 of the dissertation "Supervised Sequence Labelling with Recurrent Neural Networks" by Alex Graves.

CTC Decoding

Beam Search

To perform beam search, we keep k best **regular** (non-extended) labelings for each prefix of the extended labelings. For each regular labeling we keep both α_- and a_* and by *best* we mean such regular labelings with maximum $\alpha_- + \alpha_*$.

To compute best regular labelings for longer prefix of extended labelings, for each regular labeling in the beam we consider the following cases:

- adding a *blank* symbol, i.e., updating both α_- and α_*;
- adding a non-blank symbol, i.e., updating α_*.

Finally, we merge the resulting candidates according to their regular labeling and keep only the k best.