Neural Machine Translation

Álvaro Peris*, Miguel Domingo, Francisco Casacuberta

Pattern Recognition and Human Language Technology Research Center,
Universitat Politècnica de València,
Camino de Vera s/n, 46022 Valencia, SPAIN

Abstract

Overview of a Neural Machine Translation system.

1. Neural machine translation

In this section, we briefly review the basics of RNN and GRU units. Next, we set up the NMT methodology and describe the architectural choices made upon this framework for building the neural model used in our experiments.

1.1. Recurrent neural networks

RNNs are a class of artificial neural networks, where connections between units form a directed cycle. This creates an internal state that allows the network to model discrete-temporal sequences.

More formally, given an input sequence of X-dimensional vectors $\mathbf{x}_1^J = \mathbf{x}_1, \ldots, \mathbf{x}_J$, with $\mathbf{x} \in \mathbb{R}^X$, an RNN produces an output sequence $\mathbf{y}_1^J = \mathbf{y}_1, \ldots, \mathbf{y}_J$, with $\mathbf{y} \in \mathbb{R}^Y$, computed as:

$$\mathbf{h}_j = \zeta_h(\mathbf{x}_j, \mathbf{h}_{j-1}) \tag{1}$$

$$\mathbf{y}_j = \zeta_o(\mathbf{h}_j) \tag{2}$$

where $\mathbf{h}_j \in \mathbb{R}^H$ is the state of the network at the time-step j. ζ_h is a non-linear hidden state function, such as the logistic sigmoid function or gated units (see Section 1.2). ζ_o is an output function (e.g. the *softmax* function). Different choices on ζ_h and ζ_o lead to different RNN architectures proposed in the literature, for instance Jordan or Elman networks (Elman, 1990; Jordan, 1986). The left side of Fig. 1 shows an Elman RNN unfolded in time.

The optimal parameters of RNNs are usually estimated from training samples by means of stochastic gradient descent (SGD) (Robbins and Monro, 1951), aiming to minimize a cost function under some optimality criterion. Typically, this criterion is the cross-entropy between the output of the network and the training data probability distribution (Hinton, 1989). A popular algorithm for implementing SGD is backpropagation through time (BPTT) (Werbos, 1990). BPTT relies on the idea of unfolding the RNN over time: an unfolded RNN can be seen as a feedforward network with as many layers as unfolding steps taken. The BPTT algorithm applies standard backpropagation to this unfolded version of the RNN.

Regular RNNs have two main drawbacks: first, they suffer from the so-called vanishing gradient problem (Bengio et al., 1994) (or its exploding version), which makes them unable to model long-term relationships. This problem is tackled by using special types of recurrent units (see Section 1.2). The second problem

*Corresponding author: Phone: (34) 96 387 70 69

Email addresses: lvapeab@prhlt.upv.es (Álvaro Peris), midobal@prhlt.upv.es (Miguel Domingo), fcn@prhlt.upv.es (Francisco Casacuberta)

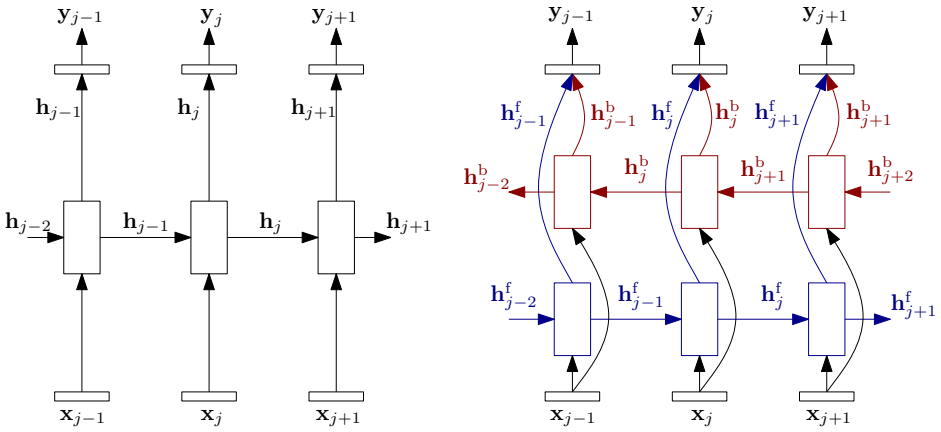

Figure 1: Elman (left) and bidirectional (right) architectures of an RNN, unfolded in time. For the sake of clarity, in the bidirectional network, forward components are blue-coloured and backward components are red-coloured.

is that the input sequence is only scanned in one direction, normally from the left to the right. Hence, information flowing from the right to the left is lost. In order to capture both left and right contexts, Schuster and Paliwal (1997) proposed the so-called bidirectional RNNs.

Bidirectional RNNs have two independent recurrent layers: the so-called forward layer, which processes the input sequence in forward time direction (from 1 to J); and the backward layer, which processes the input sequence reversed in time (from J to 1). Since hidden layers have no interaction between them, bidirectional RNNs can be trained using similar algorithms as those used for unidirectional RNNs. Following prior notation, a bidirectional RNN is defined as:

$$\mathbf{h}^{\mathrm{f}}_j = \zeta_h(\mathbf{x}_j, \mathbf{h}^{\mathrm{f}}_{j-1}) \tag{3}$$

$$\mathbf{h}^{\mathrm{b}}_j = \zeta_h(\mathbf{x}_j, \mathbf{h}^{\mathrm{b}}_{j+1}) \tag{4}$$

$$\mathbf{y}_j = \zeta_o(\mathbf{h}^{\mathrm{f}}_j, \mathbf{h}^{\mathrm{b}}_j) \tag{5}$$

where $\mathbf{h}^{\mathrm{f}}_j$ is the forward layer and $\mathbf{h}^{\mathrm{b}}_j$ is the backward layer. The output depends on a combination of both forward and backward layers. The right side of Fig. 1 shows the bidirectional version of the Elman network unfolded in time.

1.2. Gated recurrent units

Due to its nature, recurrent networks propagate the error gradient back in time many steps. Depending on the recurrent weight matrix and the activation function, gradients may rapidly tend to be zero. Hence, the influence of the current time-step on a far, previous one becomes almost zero and long relationships in the sequence are lost. This is known as the vanishing gradient problem. For tackling it, gated units were developed and used as hidden state function (ζ_h in previous equations). Analogously to the vanishing gradient, if gradients are large, the exploding version of the problem appears, producing numerical instability and learning issues. Fortunately, by clipping the gradients to a maximum predefined value during training, this problem is solved (Pascanu et al., 2012).

Gated units are able to cope with long-term relationships by deciding, at each time-step, the amount of information that flows from the previous steps to the current one and how much information from the input is taken into account in the current time-step.

In this work we used GRUs as a hidden state function of recurrent networks, following Bahdanau et al. (2015) and Cho et al. (2014). An illustration of such function can be found in Fig. 2. This type of cells can be seen as a simplified version of LSTM units. In a GRU cell, the hidden state $\mathbf{h}_j$ depends on the previous

hidden state $\mathbf{h}_{j-1}$ and an updated state $\tilde{\mathbf{h}}_j \in \mathbb{R}^H$, both modulated by an update gate $\mathbf{z}_j \in \mathbb{R}^H$:

$$\mathbf{h}_j = (1 - \mathbf{z}_j) \odot \mathbf{h}_{j-1} + \mathbf{z}_j \odot \tilde{\mathbf{h}}_j \tag{6}$$

where $\odot$ denotes the element-wise multiplication.

The updated state $\tilde{\mathbf{h}}_j$ depends on the current input $\mathbf{x}_j$ and the previous state $\mathbf{h}_{j-1}$, which is regulated by a reset gate $\mathbf{r}_j \in \mathbb{R}^H$:

$$\tilde{\mathbf{h}}_j = \tanh(\mathbf{W}\mathbf{x}_j + \mathbf{U}[\mathbf{r}_j \odot \mathbf{h}_{j-1}]) \tag{7}$$

where $\mathbf{W}$, $\mathbf{U} \in \mathbb{R}^{H \times H}$ are the weight matrices of the input and the previous hidden state, respectively. For the sake of readability, bias terms are omitted from the equations.

The reset and update gates are computed as:

$$\mathbf{r}_j = \sigma(\mathbf{W}_r \mathbf{x}_j + \mathbf{U}_r \mathbf{h}_{j-1}) \tag{8}$$

$$\mathbf{z}_j = \sigma(\mathbf{W}_z \mathbf{x}_j + \mathbf{U}_z \mathbf{h}_{j-1}) \tag{9}$$

where $\mathbf{W}_r$, $\mathbf{U}_r \in \mathbb{R}^{H \times H}$ are the weight matrices of the reset gate; $\mathbf{W}_z$, $\mathbf{U}_z \in \mathbb{R}^{H \times H}$ are the weight matrices of the update gate and σ is the element-wise logistic sigmoid function.

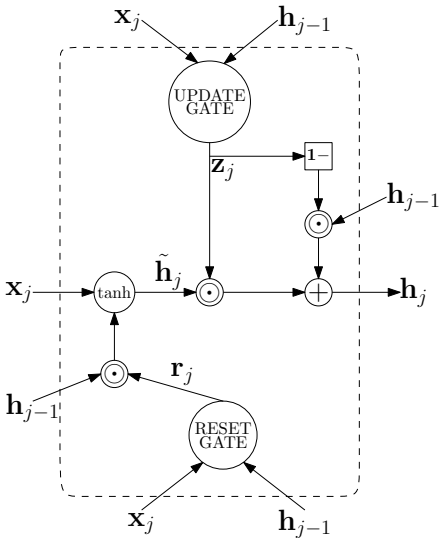

Figure 2: Gated recurrent unit. The output of the unit is modulated by the reset and the update gates. The first one decides how much information from previous states is taken into account, while the latter one modulates the amount of information from the current time-step that is considered.

1.3. Neural machine translation

From **??**, NMT systems aim to directly model the conditional translation probability:

$$\Pr(y_1^I | x_1^J) = \prod_{i=1}^{I} \Pr(y_i \mid y_1^{i-1}, x_1^J) \tag{10}$$

Most NMT systems rely on an RNN encoder-decoder framework: the encoder RNN reads a source sentence and computes a distributed representation of it. Given this representation, the decoder generates, word by word, the translated sentence. In this work, we followed the architectural choices made by Bahdanau et al. (2015), which are summarized below. In view of clarity, in this section, encoder components are denoted by index j, while decoder components are referred by index i.

The input of the system is a sequence of words $x_1^J = x_1, \ldots, x_J$, each of them belonging to the source vocabulary V_x and following a one-hot codification scheme. This can be seen as having, for each word x_j, a vocabulary-sized vector $\bar{\mathbf{x}}_j \in \mathbb{N}^{|V_x|}$ with all elements set to zero, except for that one located at the position of x_j in V_x, which is set to one. Each word is linearly projected to a fixed-size real-valued vector:

$$\mathbf{x}_j = \mathbf{E}_s \, \bar{\mathbf{x}}_j \tag{11}$$

where $\mathbf{x}_j \in \mathbb{R}^R$ is the embedding of word x_j, $\mathbf{E}_s \in \mathbb{R}^{R \times |V_x|}$ is the source language projection matrix and R is the embedding size.

The sequence of word embeddings is the input of a bidirectional RNN that computes a sequence of annotations $\mathbf{h}_1^J = \mathbf{h}_1, \ldots, \mathbf{h}_J$. The encoder combines the forward and backward layers by concatenating them (ζ_o in Eq. (5)) and uses GRU cells as ζ_h in Eq. (3) and Eq. (4). An annotation for the j-th input word is computed as:

$$\mathbf{h}_j = [\mathbf{h}_j^{\mathrm{f}}; \mathbf{h}_j^{\mathrm{b}}] \in \mathbb{R}^{2H} \tag{12}$$

where $[\cdot;\cdot]$ denotes the vector concatenation operation and $\mathbf{h}_j^{\mathrm{f}}$, $\mathbf{h}_j^{\mathrm{b}}$ are the encoder forward and backward layers respectively, at the time-step j. For the sake of simplicity in the notation, we assume that both layers have the same size H.

Cho et al. (2014) and Sutskever et al. (2014) used a simple RNN as encoder, using the last hidden state for computing a representation of the complete input sequence. However, this approach performed poorly with long sentences: this representation vector was unable of properly store the amount of information that long sentences required. While the latter authors overcame this issue reversing the source sentences, Bahdanau et al. (2015) incorporated an attention mechanism to the model, which was able to cope with long sentences. This attention mechanism allows the decoder to selectively focus on parts of the input sequence, therefore complex relationships in long sentences are not lost. Fig. 3 shows the architecture of an attention-based NMT system.

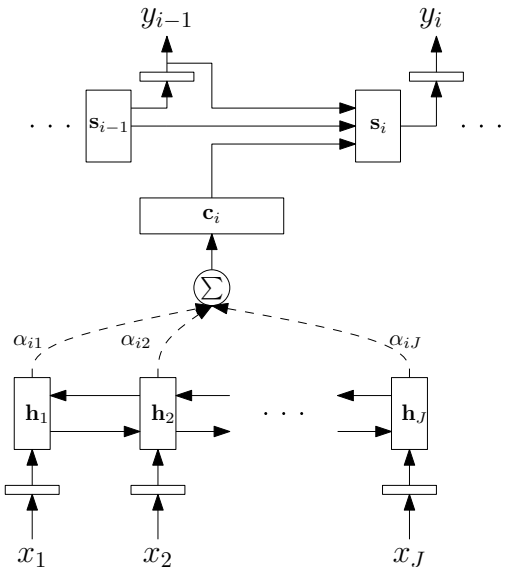

Figure 3: Architecture of the neural machine translation system, equipped with an attentional model, similarly depicted as Bahdanau et al. (2015). The bidirectional RNN encoder processes the source sentence x_1^J and generates the sequence of annotations $\mathbf{h}_1^J$. The alignment model assigns a weight α_{ij} to each annotation, according to the previous state of the decoder $\mathbf{s}_{i-1}$ and the annotation. Annotations are then combined in order to obtain the context vector $\mathbf{c}_i$. The decoder generates the next word y_i according to the context vector, its previous hidden state and the previously generated word y_{i-1}.

At each time-step i, the attention mechanism computes a different context vector $\mathbf{c}_i \in \mathbb{R}^{2H}$ as the weighted sum of the sequence of annotations $\mathbf{h}_1^J$:

$$\mathbf{c}_i = \sum_{j=1}^{J} \alpha_{ij} \mathbf{h}_j \tag{13}$$

where $\alpha_{ij} \in \mathbb{R}$ is the weight assigned to each annotation $\mathbf{h}_j$. This weight is computed by means of the softmax function:

$$\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k=1}^{J} \exp(a_{ik})} \tag{14}$$

where $a_{ij} \in \mathbb{R}$ is a score provided by a soft alignment model, which measures how well the inputs from the source position j and the outputs around the target position i match. This alignment model is implemented by a perceptron with N units:

$$a_{ij} = \mathbf{v}_a^\top \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a \mathbf{h}_j) \tag{15}$$

where $\mathbf{s}_{i-1} \in \mathbb{R}^H$ is the hidden state from the decoder, $\tanh(\cdot)$ is applied element-wise and $\mathbf{v}_a \in \mathbb{R}^N$, $\mathbf{W}_a \in \mathbb{R}^{N \times H}$ and $\mathbf{U}_a \in \mathbb{R}^{N \times 2H}$ are the weight matrices.

The decoder is an RNN with GRU units, that generates the translated sentence $y_1^I = y_1, \ldots, y_I$. Each word y_i depends on the previously generated word y_{i-1}, the current hidden state of the decoder $\mathbf{s}_i$, and the context vector $\mathbf{c}_i$. Assuming a model for $\Pr(y_i \mid y_1^{i-1}, x_1^J)$ in Eq. (10) of parameters θ (the different matrices in the encoder and the decoder), the probability of a word at the times-step i is defined as:

$$p(y_i \mid y_1^{i-1}, x_1^J; \theta) \;=\; \bar{\mathbf{y}}_i^\top \boldsymbol{\varphi}(\mathbf{V}\; \boldsymbol{\eta}(y_{i-1}, \mathbf{s}_i, \mathbf{c}_i)) \tag{16}$$

where $\boldsymbol{\varphi}(\cdot) \in \mathbb{R}^{|V_y|}$ is a softmax function that produces a vector of probabilities, $|V_y|$ is the size of the target vocabulary, $\bar{\mathbf{y}}_i \in \mathbb{N}^{|V_y|}$ is the one-hot representation of the word y_i, $\mathbf{V} \in \mathbb{R}^{|V_y| \times L}$ is the weight matrix and $\boldsymbol{\eta}$ is the output of an Elman RNN with GRU units and an L-sized maxout (Goodfellow et al., 2013) output layer.

1.4. Training and decoding

To estimate the model parameters θ, the training objective is to maximize the log-likelihood over a bilingual parallel corpus $\mathcal{S} = \{(x^{(s)}, y^{(s)})\}_{s=1}^S$, consisting of S sentence pairs, with respect to θ:

$$\hat{\theta} = \arg\max_\theta \sum_{s=1}^{S} \sum_{i=1}^{I_s} \log(p(y_i^{(s)} \mid y_1^{i-1\,(s)}, x^{(s)}; \theta)) \tag{17}$$

where I_s is the length of the s-th target sentence. For the sake of clarity, we drop the dependencies on θ in the rest of equations of this manuscript.

Once the model parameters are estimated, at the test phase, the goal of the NMT system is to obtain, given a source sentence x_1^J, the target language sentence $\hat{y}_1^{\hat{I}}$ with maximum posterior probability:

$$\hat{y}_1^{\hat{I}} = \arg\max_{I, y_1^I} \; p(y_1^I \mid x_1^J) \tag{18}$$

An optimal solution to this expression would require to search over the space of all possible target sentence, which is unaffordable. For generating translations, suboptimal decoding strategies such as beam-search were used by Bahdanau et al. (2015), Cho et al. (2014) and Sutskever et al. (2014). In this method, instead of considering the full target language space, only the $\mathcal{B}$ most promising hypotheses are considered.

At each time-step of the generation process, each one of the $\mathcal{B}$ partial hypotheses is expanded with every word in the target vocabulary. Then, this large set of partial hypotheses is pruned, remaining only the $\mathcal{B}$ hypotheses with highest probability.

As the end-of-sequence symbol is appended to a hypothesis, the beam width $\mathcal{B}$ is decreased in one and this hypothesis is marked as complete. The process continues until the beam width reaches zero. The final hypothesis produced by the system is the one with maximum probability, among the set of complete hypotheses generated during the process.

References

Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In: Proceedings of the International Conference on Learning Representations. `arXiv:1409.0473 [cs.CL]`.

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. Inst. of Electr. and Electr. Eng. Trans. on Neural Netw. 5 (2), 157–166.

Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y., 2014. On the properties of neural machine translation: Encoder-decoder approaches. In: Proceedings of the Workshop on Syntax, Semantic and Structure in Statistical Translation. pp. 103–111.

Elman, J. L., 1990. Finding structure in time. Cogn. Sci. 14 (2), 179–211.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y., 2013. Maxout networks. In: Proceedings of the International Conference on Machine Learning. pp. 1319–1327.

Hinton, G. E., 1989. Connectionist learning procedures. Artif. intell. 40 (1), 185–234.

Jordan, M. I., 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In: Proceedings of the Eighth Annual Conference of the Cognitive Science Society. pp. 531–546.

Pascanu, R., Mikolov, T., Bengio, Y., 2012. On the difficulty of training recurrent neural networks, `arXiv:1211.5063 [cs.LG]`.

Robbins, H., Monro, S., 1951. A stochastic approximation method. The Ann. of Math. Statistics, 400–407.

Schuster, M., Paliwal, K. K., 1997. Bidirectional recurrent neural networks. Inst. of Electr. and Electr. Eng. Trans. on Signal Process. 45 (11), 2673–2681.

Sutskever, I., Vinyals, O., Le, Q. V., 2014. Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems. Vol. 27. pp. 3104–3112.

Werbos, P. J., 1990. Backpropagation through time: What it does and how to do it. Vol. 78. pp. 1550–1560.