Generating Turkish Text with Recurrent Neural Networks

Emre Kağan Akkaya
Graduate School of Science and Engineering
Department of Computer Engineering
Hacettepe University
emrekaganakkaya@gmail.com

***Abstract*—Recurrent Neural Networks (RNN) are powerful models that have attracted great interest in many natural processing tasks. These tasks include - among others - machine translation, text generation, text classification, and even handwriting recognition. In this paper, we provide a pre-trained language model; a Gated Recurrent Unit (GRU) LM, which is a special kind of RNN specifically designed to deal with long term dependencies in sentences, trained on custom-built Turkish datasets in order to generate sentences. In addition, Python implementation of the model is provided for further development.**

Keywords—recurrent neural networks; gated recurrent unit; text generation; turkish text generation

I. Introduction

Neural networks have been widely adopted to different research areas since they have proven to be efficient and gained popularity in the last decade. Recurrent neural networks (RNN), a sub branch of the neural networks, have been used for many different task such as generating image description, machine translation, speech recognition and so on.

Main idea of RNNs is to make use of sequential information which uses its hidden state to learn about arbitrarily long sequences and use this state to predict how likely for a word to occur. This can be seen as if they have 'memory' that captures information about what has been calculated so far. As a side effect of being able to predict the next word, a generative model is produced which allows us to generate new text by sampling from the output probabilities. In this research, we have used this ability to generate new Turkish text with the model trained on custom-built Turkish datasets. The work presented here is based on partly [1], [2] and numerous online tutorials.

The outcome of this research is twofold. First, we provide an implementation of a GRU network. This implementation is primarily based on Theano which is a Python library allowing parallelized mathematical computations on both GPU and CPU. Long time-taking processes such as matrix multiplications are handled effectively by this library.

Second, a pre-trained language model with the capability of generating Turkish text is provided. The model parameters are saved automatically during runtime and then can be resumed by a simple command. Moreover, as a by-product, Turkish datasets are also provided. One of them is Wiki dataset collected from Turkish Wikipedia pages and the others are e-books of some authors in the Turkish literature.

In the following sections; we first give a detailed description of an RNN and how a GRU [3] works in Section II. Then in sections III and IV, we define the implemented model and show the experimental results respectively. In Section V, related works are examined briefly. Finally, in Section VI we conclude the paper.

II. Recurrent neural networks

A. Definition

In a traditional neural network we assume that all inputs and outputs are independent of each other. However, in order to predict the next word in a sentence, it is better to know the words before it, that being said, RNNs perform the same prediction task for every element of an input sequence, with the output being depended on the previous computations. An unrolled/unfolded RNN typically looks like the one in Fig. 1 where x_t, s_t, o_t are the input (one-hot vector), hidden state and output at time step t respectively.

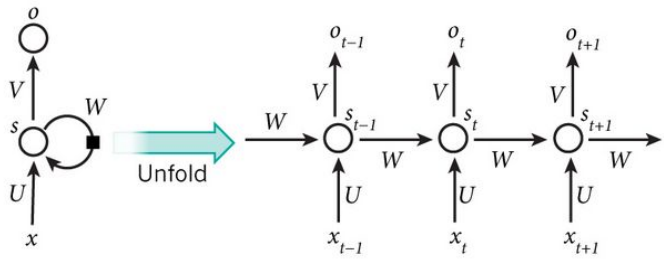

Figure 1. Recurrent Neural Network

For clarity purposes, in order to better understand the network, we briefly explain each input and output; s_t is the *memory* of the network which is calculated based on the previous hidden state and the input at the current step. The equation of this calculation is:

$$s_t = f(Ux_t + Ws_{t-1}) \quad (1)$$

o_t is the output which gives us a vector of probabilities across the vocabulary and the vector can later be used to predict the next word in a sentence

$$o_t = softmax(Vs_t) \quad (2)$$

Unlike a traditional deep neural network which uses different parameters at each layer, an RNN shares the same parameters (*U, V, W* in Fig.1) across all steps. This reflects the fact that we are performing the same task at each step (that is why *recurrent*), just with different inputs. This also reduces the total number of parameters we need to learn. In Fig.1 the network has outputs at each time step, but depending on the task this may not be necessary, e.g. predicting the sentiment of a sentence, only the final output is important. Nevertheless the main feature of an RNN is the hidden state which captures some information about a sequence. Though it is limited to looking only a few steps back, due to exploding/vanishing gradient problem, first described in [5] (German) and surveyed in another work [6].

Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps which is called Backpropagation Through Time (BPTT).

It is worth reminding that our goal is to calculate the gradients of the error with respect to the parameters *U, V,* and *W* and then learn good parameters using Stochastic Gradient Descent. To calculate the gradients, a chain rule of differentiation is used, in other words, the gradients are need to be back-propagated from the current step all the way to the first step. In practice, the backpropagation is truncated to a few steps. To be able to learn long-range dependencies, interactions between words that are several steps apart, going back to all steps has a high cost and lengthens the training process. But this is not the main problem here, during the calculations, the gradient values are shrinking exponentially, eventually vanishing completely after a few time steps which means gradient contributions from further steps become zero and the states at those steps doesn't contribute to the learning process. This defined problem is called vanishing gradient and practically, it makes vanilla RNNs impossible to be used in NLP tasks.

Another issue about the gradient calculations is that, depending on the activation functions and parameters, an exploding instead of vanishing gradient problem can occur. But clipping the gradients at a pre-defined threshold is an effective solution to this issue. That's why vanishing gradient is more problematic since it is not obvious when to occur or how to handle it.

In order to combat vanishing gradient, Long Short-Term Memory (LSTM) [3] and Gated Recurrent Unit (GRU) [4] architectures are proposed and widely used in recent years.

B. Gated Recurrent Unit

A gated recurrent unit has two gates, a reset gate *r* which determines how to combine the new input with the previous memory and an update gate *z* which defines how much of the previous memory to keep around.

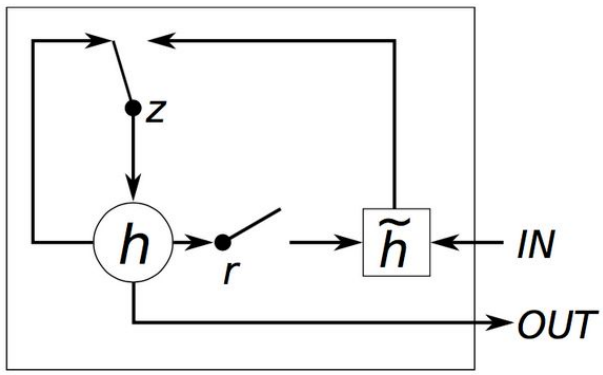

Figure 2. GRU mechanism

If the reset gate is set to 1's and update to 0's, we actually get vanilla RNN model. This gating mechanism is used to learn long-term dependencies and related equations are shown below:

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r) \quad (3)$$
$$h = \tanh(x_t U^h + (s_{t-1} \text{ o } r) W^r)$$
$$s_t = (1\text{-}z) \text{ o } h + z \text{ o } s_{t-1}$$

Both GRU and LSTM are widely used and their tradeoffs haven't been fully explored yet but GRU is newer and according to some comparison GRUs have fewer parameter values (*U* and *V* are smaller) and thus it may train a bit faster. On the other hand, it is reported that if you have enough data, the greater excessive power of LSTMs may lead to better results.

III. Language Model

C. Preprocessing

In order to train the model we need lots of text to learn from. Approximately 800 MB of Wikipedia articles was first downloaded, then stripped of Wiki markup tags and metadata. In addition, e-books of different authors, such as Sabahattin Ali, Uğur Mumcu, and Reşat Nuri Güntekin are collected and any HTML tags or metadata are also removed from them.

To be able to make predictions on a per-word basis, all Wikipedia articles and e-book texts must be tokenized into sentences and sentences into words. NLTK [7], a Python natural language processing toolkit, is used to do tokenization. After tokenization, there are 2.195.043 sentences and 1.041.577 unique tokens.

Then, infrequent words also removed, otherwise having a huge vocabulary will make the model slow to train. Also datasets don't contain a lot of these words which makes it impossible to learn how to use them correctly. Therefore a vocabulary size is defined, varying from 6.000 to 10.000 words in different experiment settings. All words not included in the vocabulary are replaced with UNK token. To be able to learn which words tend to start and end a sentence, a special sentence start token is prepended to each sentence, likewise an end token is appended to each sentence, too.

Since the input to RNN are vector, not strings, we have to create a mapping between words and indices. Specifically, each word is represented as a one-hot vector of vocabulary size. This transformation is handled in computation phase.

D. Training

After the preprocessing, the parameters *U, V*, and *W* must be initialized. Proper initialization depends on the activation function (tanh) and the recommended approach is to initialize the weight randomly in the interval from $[-1/\sqrt{n}, 1/\sqrt{n}]$ where *n* is the number of incoming connections from the previous layer.

Then at each time step, the forward propagation defined by the equations explained in Section II is computed where o_t is a vector of probabilities representing the words in the vocabulary. By using the highest probable word, we can predict the next word and eventually generate sentences.

To train the network, a way to measure the errors it makes is needed. This measure is called loss function which the goal is to find parameters U, V, and W that minimizes the loss function for the given training data. Cross-entropy loss function is used in the experiments which defines the loss with respect to the predictions *o* and the true labels *y* as below:

$$L(y, o) = (-1/N) * \Sigma_{n \in N} y_n \log(o_n) \quad N: \text{training examples} \tag{4}$$

Defined loss function basically sums over the training examples (words) and add to the loss based on how off the predictions are. The further away *y* (the correct words) and *o* (the predictions), the greater the loss value will be. For random predictions, assume we have C=10.000 words in the vocabulary, each word should be predicted with the probability of 1/C, which results in a loss value of L= (-1/N) N*log1/C = logC = 9.21. This value will be the baseline during evaluation.

Main goal is to find the parameters *U, V,* and *W* that minimize the total loss on the training data. The proper way to update the parameters is Stochastic Gradient Descent (SGD) which iterates over all the training examples and during each iteration the parameters are nudged into a direction that reduces the error. SGD takes also a parameter, named *learning rate*, which defines how big of a step it needs to make in each iteration. Since the parameters U,V, and W are shared by all time steps in the network, the gradient of SGD at each output depends not only on the calculations of the current time step, but also the previous time steps. A modified version of backpropagation, called Backpropagation Through Time (BPTT), is used to calculate these gradients. In addition to BPTT, gradient checking is used to verify the correctness of the implementation and compared with the gradient calculated using BPTT. It is also crucial to remind that gradient checking is very expensive to compute and should only be used on a model with a small vocabulary.

To summarize the details explained above, a SGD step is implemented that calculates the gradients with the help of BPTT and performs the updates for one batch. This step is iterated through the training set and as a result the learning rate is adjusted according to the loss value. It must be noted that, in order to speed calculations up, Theano is used to optimize the code where possible.

During this training process, we generate sample sentences every 25.000-30.000 examples so that we can track how meaningful the generated sentences are getting. This process and the sentences are explained briefly in the next section.

IV. Experiments

The described model has been trained on two different datasets. One of them is Wikipedia dataset and the other one is books of author, Sabahattin Ali. The other two datasets (Uğur Mumcu and Reşat Nuri Güntekin) are scheduled to be trained on a later date, after the experiment with Wikipedia dataset finishes successfully. Information about the datasets can be examined in Table 1.

TABLE I. Dataset Definitions

Dataset name	Size	Number of sentences	Number of unique tokens
Wikipedia	800 MB	2.195.043	1.041.577
Sabahattin Ali	6,8 MB	4.244	8.800
Uğur Mumcu	13,1 MB	-	-

Reşat Nuri Güntekin	20,8 MB	-	-

E. Experiment Settings

For the sake of clarity, the simulation environment and the related parameters are described in Table 2.

TABLE II. ENVIRONMENT AND PARAMETERS

Environment or parameter	Value
Operating system	Debian GNU/Linux 6 (64bit)
CPU	8 GHz
Memory	16 GB RAM memory
Vocabulary size	6.000, 8.000, 10.000
Number of epochs	20, 40
Hidden dimension size	100, 120
Number of GRU layers	2
SGD learning rate	0.001
Generate sentences for every # of examples	30.000

Multiple values for an experiment parameter indicate that the same simulation is run multiple times with different parameter values.

F. Results

Preliminary results with the model trained on Sabahattin Ali yields poor output (almost randomly), indicating that size of the training dataset is crucial to learning and it needs even more to learn long-term dependencies. Some of the output is listed in Table III.

TABLE III. OUTPUT OF THE MODEL TRAINED WITH SABAHATTIN ALI DATASET

Generated sentence
yazıları da resimde birdenbire
cümle öykülerden sanayi olmayacağını hep ?
pansiyonda filiz'in içine bir atmaya olurdu .
sözünü vermek birbirimize sakın ?
söyledi … bir sıkıldım mı ?
tedbiri ben bu o fakat zaman !
çok bu içime tanesini bakıyordu .
fakat sonra bana .
hayret etti :
daha bana sabahleyin öyküleri
zavallının üstüne dedim .
evine bu kadar değirmenci sordum .

After the preprocessing steps are applied as described in Section III, Wikipedia dataset is used as another simulation and the results are more satisfactory which means it is able to learn dependencies between words that are several steps apart and it can learn syntax and generate somewhat meaningful data. Some of the generated sentences are shown in Table IV.

TABLE IV. OUTPUT OF THE MODEL TRAINED WITH WIKIPEDIA DATASET

Generated sentence
içme suyu şebekesi
uzun 2010 yılında
ptt şubesi ve ptt acentesi yoktur .
köye ulaşımı sağlayan yol asfalt olup
köyün ekonomisi tarım ve hayvancılığa dayalıdır .
mahallenin adının nereden geldiği ve geçmişi hakkında bilgi yoktur .
köyde ilköğretim okulu vardır .
15 mart 2010 tarihinde yapılan bir nüfus sayımına göre şehrin nüfusunun ve

In order to further increase the accuracy of model and generate more meaningful sentences, we believe that Wikipedia dataset should be cleaned thoroughly (some HTML/CSS attributes and metadata, which is very hard to replace using regex or any other method, still remains in the dataset). Another optimization might be to increase both the vocabulary size and the hidden dimension size, but in order to compensate slowness of the training process, Wikipedia dataset may also be reduced to a reasonable size or another dataset might be used instead.

V. RELATED WORK

In Sutskever's research [1] an alternative recurrent neural network, multiplicative RNN (MRNN), is proposed to generate text with the help of a character-level model. The proposed model uses Hessian-Free optimizer (HF) to be successfully applied to character-level language modeling tasks. In the paper, it is stated that using characters that even 'predict' some words not in the training data, and can outweigh the extra work of having to learn words.

Another research about generating text with RNN is [9] which uses LSTM for both text generation and handwriting synthesis. It has also introduced "a convolutional mechanism to allow an RNN to condition its predictions on an auxiliary annotation sequence, and use this approach to synthesise diverse and realistic samples of online handwriting" [9].

Also, there are other works on recurrent neural networks which focus on capturing image descriptions, speech recognition, machine translation etc.

VI. Conclusions

We implemented a recurrent neural network with gated recurrent units which takes sentences in a custom-built dataset as inputs and outputs auto-generated Turkish sentences. Our experiments indicates that the model can be used efficiently to learn long-term dependencies in Turkish sentences, although it needs a considerable amount of time to train.

In terms of the aim we defined for the term project; a Python implementation for RNN is provided, along with custom-built Turkish datasets, Wiki and e-books. Finally a pre-trained language model is presented which can be used to generate sentences or to experiment with.

In future work, we hope to increase the accuracy and the number of the meaningful generated sentences by adding an embedding layer such as word2vec or GloVe. Instead of using one-hot vectors to represent input words, the vectors learned using these methods carry semantic meaning which means similar words have similar vectors. We also have another aim which is to provide implementation of the other researched model, LSTM, as well.

References

[1] Sutskever, Ilya, James Martens, and Geoffrey E. Hinton, "Generating text with recurrent neural networks," Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011.

[2] Mikolov, Tomas, et al. "Recurrent neural network based language model," Interspeech. Vol. 2, 2010.

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory," Neural computation 9.8 (1997): 1735-1780.

[4] Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv preprint arXiv: 1406,1078 (2014).

[5] Hochreiter, Sepp. "Untersuchungen zu dynamischen neuronalen Netzen," Diploma Thesis, TU Munich, 1991.

[6] Hochreiter, Sepp, et al. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," (2001).

[7] NLTK, Natural Language Processing with Python, http://www.nltk.org

[8] Theano, a Python library to define, optimize, and evaluate mathematical expressions, http://deeplearning.net/software/theano

[9] Graves, Alex. "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308,0850 (2013).