English To French Neural Machine Translation using Transformer

Moh.Shohanur Rahman
department of CSE
BRAC University
Dhaka, Bangladesh
moh.shohanur.rahman@g.bracu.ac.bd

Saib Ahmed
department of CSE
BRAC University
Dhaka, Bangladesh
saib.ahmed@g.bracu.ac.bd

Shah Sufian Noor Mahady
department of CSE
BRAC University
Dhaka, Bangladesh
shahsufian.noor.mahady@g.bracu.ac.bd

Md Humaion Kabir Mehedi
department of CSE
BRAC University
Dhaka, Bangladesh
humaion.kabir.mehedi@g.bracu.ac.bd

Annajit Alim Rasel
department of CSE
BRAC University
Dhaka, Bangladesh
annajiat@gmail.com

***Abstract*—Neural Machine Translation is a many-to-many problem, like a short sentence in a particular language may become long in another language or vice versa. Sequence-to-sequence models are deep learning models that have excelled in tasks like machine translation, text summarization, and picture captions. A model that outputs another sequence of items from a sequence of input items (words, letters, features of an image, etc.) is known as a sequence-to-sequence model. The Transformer is a model that uses attention to boost the speed with which these models can be trained. In our project, we have used this transformer which is a Seq2Seq model to create an English-to-French language translator. Our project can help to create a more robust and effective translator. It can be used for other languages as well. This model can be used on creating a translator chatbot as well.**

***Index Terms*—Seq2Seq, Transformers, BERT, BLEU-metric, BERT-score, Autoregressive model.**

I. INTRODUCTION

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013), Sutskever et al. (2014) and cho et al., (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn et al., 2003) which consists of many sub-components that are tuned separately , neural machine translation attempts to build and train a single , larghe network that reads asentence and outputs a corrects translation.

Most of the proposed neural machine translation models belong to a family of encoder– decoders (Sutskever et al., 2014; Cho et al., 2014a), with an encoder and a decoder for each lan- guage, or involve a language-specific encoder applied to each sentence whose outputs are then com- pared (Hermann and Blunsom, 2014). An encoder neural network reads and encodes a source sen- tence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence. A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho et al. (2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases. In order to address this issue, we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words. The most important distinguishing feature of this approach from the basic encoder–decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it en- codes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector. We show this allows a model to cope better with long sentences. In this paper, we show that the proposed approach of jointly learning to align and translate achieves significantly improved translation performance over the basic encoder–decoder approach. The im- provement is more apparent with longer sentences, but can be observed with sentences of any length. On the task of English-to-French translation, the proposed approach achieves, with a single model, a translation performance comparable, or close, to the conventional phrase-based system. Furthermore, qualitative analysis reveals that the proposed model finds a linguistically plausible (soft-)alignment between a source sentence and the corresponding target sentence.

II. LITERATURE REVIEW

Results from related research that analyzed various language translation using a variety of approaches and strategies are presented. Machine learning is a development of human-made brainpower that discovers connections between hubs without preparing them in monasteries. Researchers have created and used a variety of prediction models utilizing different data mining techniques, machine learning algorithms, or even a mix of these approaches. In recent years seq-to-seq recurrent neural networks with long term memory cells have proven successful at NLP. In fact results they yielded have been so good that gap between human translations and machine translations has narrowed. In [1] they have introduced a new convolution architecture named SliceNet based on the use of deptwise separable convolutions. They shows this architecture achieves results beating only ByteNet but also previous state of art.

In [2] They presented the Transformer, first sequence transduction based entirely on attention replacing most used encoder-decoder architecture. Transformer can be trained significantly faster than convolution layers or recurrent layers. Transformer to problems involving input and output modalities other than text and investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images and audio.

In [3] they demonstrate that a single deep learning odel can jointly learn from number of large scale tasks from multiple domains. The key to success comes from designing a multimodal architecture in which as many parameters as possible are shared and from using computational blocks from different domains together. We believe that this treads a path towards interesting future work on more general deep learning architectures, especially since our model shows transfer learning from tasks with a large amount of available data to ones where the data is limited.

In [4] In the article they tried to experiment in neural machine translation using recent tensor2Tensor framework and Transformer sequence to sequence model. They examine some important parameters that affect final translation quality m memory usage and stability, they address scaling to multiple GPU and provide training regarding batch size and learning rate. They represent a wide range of basic experiments with the Transformer model for English to Czech neural machine translation. They observed that Transformer model larger batch sizes lead not only to faster training but mor importantly better translation quality. The best performing model we obtained on 8 GPUs trained for 8 days has outperformed the WMT17 winner in several automatic metrics. Matching the features of the data to be learnt with those current methods in a step in selecting a machine learning algorithm. The following sections discusses different machine learning algorithm have been used in this research.

III. PRELIMINARY CONCEPTS

In this project we are going to use Bidirectional Encoder Representations From Transformers (BERT) algorithm. We also need concept of Recurrent Neural Network (RNN) and LSTM neural network. A brief details of those will be discussed on point

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. RNN:

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.

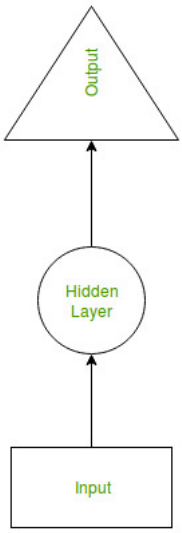

Fig. 1. RNN.

RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

How RNN works The working of an RNN can be understood with the help of the below example:

Example: Suppose there is a deeper network with one input layer, three hidden layers, and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let’s say, for hidden layer 1 the weights and biases are (w1, b1), (w2, b2) for the second hidden layer, and (w3, b3) for the third hidden layer. This means that each of these layers is independent of the other, i.e. they do not memorize the previous outputs.

Now the RNN will do the following:

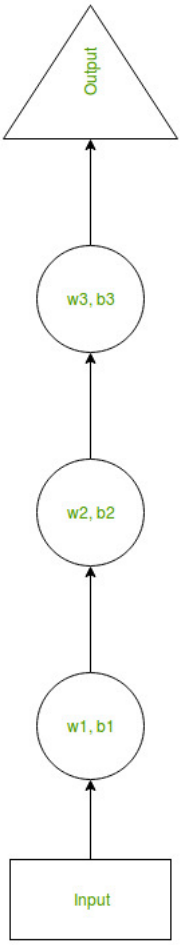

Fig. 2. RNN.

- RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.
- Hence these three layers can be joined together such that the weights and bias of all the hidden layers are the same, in a single recurrent layer.

The formula for calculating current state:

$h_t = f(h_{t-1}, x_t)$

$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$

$y_t = W_{hy}h_t$

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.

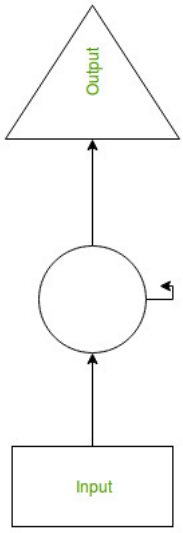

Fig. 3. RNN.

6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

B. LSTM:

It is special kind of recurrent neural network that is capable of learning long term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.

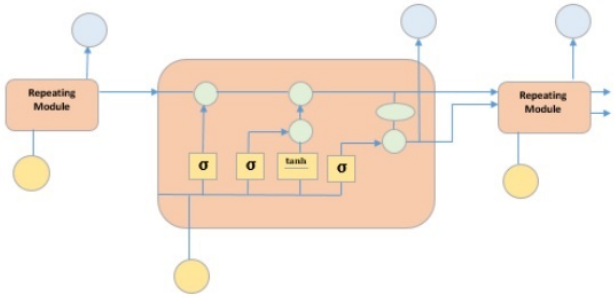

Fig. 4. LSTM.

The picture above depicts four neural network layers in yellow boxes, point wise operators in green circles, input in yellow circles and cell state in blue circles. An LSTM module has a cell state and three gates which provides them with the power to selectively learn, unlearn or retain information from each of the units. The cell state in LSTM helps the information to flow through the units without being altered by allowing only a few linear interactions. Each unit has an input, output and a forget gate which can add or remove the information to the cell state. The forget gate decides which information from the previous cell state should be forgotten

for which it uses a sigmoid function. The input gate controls the information flow to the current cell state using a point-wise multiplication operation of 'sigmoid' and 'tanh' respectively. Finally, the output gate decides which information should be passed on to the next hidden state.

C. BERT

BERT (Bidirectional Encoder Representations from Transformers) is a Natural Language Processing Model proposed by researchers at Google Research in 2018. When it was proposed it achieve state-of-the-art accuracy on many NLP and NLU tasks such as:

- General Language Understanding Evaluation
- Stanford Q/A dataset SQuAD v1.1 and v2.0
- Situation With Adversarial Generations

Soon after few days of release the published open-sourced the code with two versions of pre-trained model BERTBASE and BERTLARGE which are trained on a massive dataset. BERT also use many previous NLP algorithms and architectures such that semi-supervised training, OpenAI transformers, ELMo Embeddings, ULMFit, Transformers. BERT Model Architecture: BERT is released in two sizes BERTBASE and BERTLARGE. The BASE model is used to measure the performance of the architecture comparable to another architecture and the LARGE model produces state-of-the-art results that were reported in the research paper. Semi-supervised Learning: One of the main reasons for the good performance of BERT on different NLP tasks was the use of Semi-Supervised Learning. This means the model is trained for a specific task that enables it to understand the patterns of the language. After training the model (BERT) has language processing capabilities that can be used to empower other models that we build and train using supervised learning.

BERT is basically an Encoder stack of transformer architecture. A transformer architecture is an encoder-decoder network that uses self-attention on the encoder side and attention on the decoder side. BERTBASE has 12 layers in the Encoder stack while BERTLARGE has 24 layers in the Encoder stack. These are more than the Transformer architecture described in the original paper (6 encoder layers). BERT architectures (BASE and LARGE) also have larger feedforward-networks (768 and 1024 hidden units respectively), and more attention heads (12 and 16 respectively) than the Transformer architecture suggested in the original paper. It contains 512 hidden units and 8 attention heads. BERTBASE contains 110M parameters while BERTLARGE has 340M parameters.

This model takes CLS token as input first, then it is followed by a sequence of words as input. Here CLS is a classification token. It then passes the input to the above layers. Each layer applies self-attention, passes the result through a feedforward network after then it hands off to the next encoder. The model outputs a vector of hidden size (768 for BERT BASE). If we want to output a classifier from this model we can take the output corresponding to CLS token

Now, this trained vector can be used to perform a number of tasks such as classification, translation, etc. For Example, the

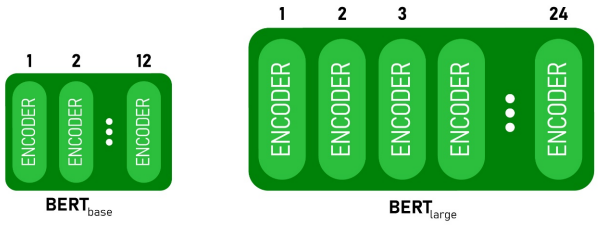

Fig. 5. BERT architecture.

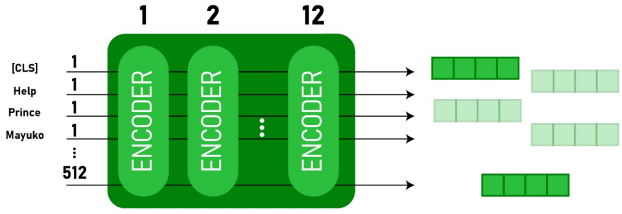

Fig. 6. BERT output as Embeddings.

paper achieves great results just by using a single layer NN on the BERT model in the classification task. ELMo Word Embeddings: This article is good for recapping Word Embedding. It also discusses Word2Vec and its implementation. Basically, word Embeddings for a word is the projection of a word to a vector of numerical values based on its meaning. There are many popular words Embedding such as Word2vec, GloVe, etc. ELMo was different from these embeddings because it gives embedding to a word based on its context i.e contextualized word-embeddings.To generate embedding of a word, ELMo looks at the entire sentence instead of a fixed embedding for a word. Elmo uses a bidirectional LSTM trained for the specific task to be able to create those embeddings. This model is trained on a massive dataset in the language of our dataset, and then we can use it as a component in other architectures that are required to perform specific language tasks.

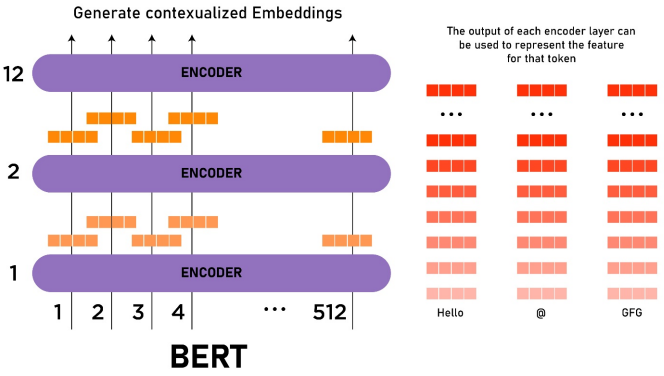

Fig. 7. Elmo Contextualize Embeddings Architecture.

ELMo gained its language understanding from being trained to predict the next word in a sequence of words – a task called Language Modeling. This is convenient because we have vast amounts of text data that such a model can learn from without labels can be trained. ULM-Fit: Transfer Learning In NLP: ULM-Fit introduces a new language model and process to effectively fine-tuned that language model for the specific task. This enables NLP architecture to perform transfer learning on a pre-trained model similar to that is performed in many Computer vision tasks. Open AI Transformer: Pre-training: The above Transformer architecture pre-trained only encoder architecture. This type of pre-training is good for a certain task like machine-translation, etc. but for the task like sentence classification, next word prediction this approach will not work. In this architecture, we only trained decoder. This approach of training decoders will work best for the next-word-prediction task because it masks future tokens (words) that are similar to this task. The model has 12 stacks of the decoder layers. Since there is no encoder, these decoder layers only have self-attention layers. We can train this model for language modelling (next word prediction) task by providing it with a large amount of unlabeled dataset such as a collection of books, etc.

IV. DATASET

The following English-French parallel corpora are included in WMT '14: UN (421M), Europarl (61M), news commentary (5.5M), and two crawling corpora (90M and 272.5M words, respectively). Together, these corpora total 850M words. The combined corpus is down to 348M words in size. Only parallel corpora are used. To create a development (validation) set and assess the models on the test, use news-test-2013 and 2012. The test set (news-test-2014) from WMT '14, which comprises of 3003 phrases not found in the training data, is used to evaluate the models. A shortlist of the 30,000 most frequent terms in each language is used to train the models after a standard tokenization. Every term that is not on the shortlist is assigned to a unique token ([UNK]). There was no additional special data preprocessing, such as lowercasing or stemming.

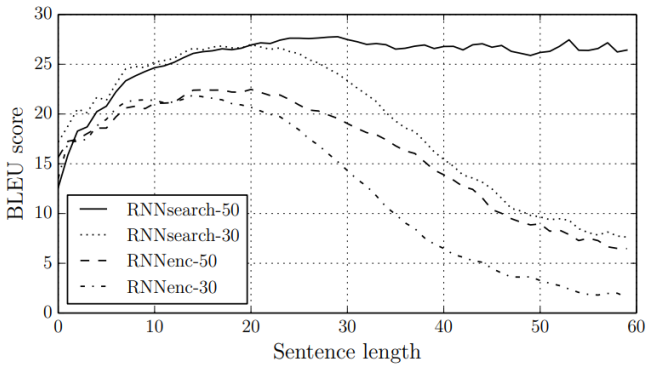

Fig. 8. Dataset.

V. METHODOLOGY

At first, we load up our English and French language pair dataset. There we have over 20,000 samples. Unfortunately, this much data will take a lot of time and computation power. That's why we took only half of the dataset. After that, we split the dataset into train and test.We load up the tokenizer object and use checkpoint = 'Helsinki-NLP/opus-mt-en-fr'.In the next step, we print out an example sentence pair which we will subsequently tokenize later. Then we tokenized the input sentence and check the tokenized result. After that, we tokenized the target sentence which requires to use of the 'as_target_tokenizer' context manager.

Then we drew a histogram of the sequence length of the inputs. We do this because we want to specify the maximum sequence length for truncation. From the visualization, we can see most of the samples in the train set fall in the shorter end. That's why we set the maximum input length 256. Next, we draw a similar histogram for the targets instead of inputs and set the maximum target length as 256. Next, we write a tokenizer function. At first, we set both the input and target length 256. Then we define our tokenizer function named tokenizer_fn, which takes a batch of data as input. Inside the function, we begin by splitting up the samples into input and target lists. In the next step, we tokenized the input. Next, we tokenized the targets.

The Hugging Face API expects the targets to be in a field called 'labels'. So that is where we placed the token IDs from the targets. The attention mask is not needed, although the tokenizer does produce them. Finally, we return the Tokenized Inputs object, which contains both the input fields and the targets or labels field. In the next step, we mapped our tokenizer function to the dataset dictionary, as well as removed the original columns which we no longer needed.

Next, we loaded up our pre-trained model and for that, we selected AutoModelForSeq2SeqLM class. Then we created data collator. As with token classification, we needed to explicitly define a data collator and in this case, it also accepts a model as input. This model is used to generate the token decoder input IDs. The data collator expects as input a list of samples where each sample is a dictionary.

After that, we tested our data collator on the inputs. We also printed out the keys of the result to see if we get anything new. We simply use the data collator to pad the sequences and convert them into torch tensors.

Then we installed two libraries 'sarcrebleu' and 'bertscore' for implementing the metrics bleu-score and bert-score. Then we used load_metric function to load up our two metrics. Bleu-score is not good for detecting similar words, or the word with the same meaning. Bert-score is useful here. Then we defined our own compute_metrics function using these two libraries. We defined our function, which takes as input a tuple of predictions and labels. The API is inconsistent here and we actually get token IDs and not just logits. In the next step, we decode the predictions back in two sentences. This also makes use of the tokenizer, since it has all the

information needed to convert strings into token IDs and vice versa. For some reason the tokenizer does not recognize the -100 values, so we manually replaced all the -100 in the labels with the patch token ID. Then we decoded the labels into sentences as well. Then we get rid of extraneous whitespace and also put the targets into their lists as required by the metrics. Next, we called both the 'bleu_metric.compute' and 'bert_metric.compute', and finally we return a dictionary containing both the 'bleu-score' and 'bert-score'. The 'bert-score' is returned per sample, so we took the mean.

Next, we trained and evaluated our transformer model. For this, we started by creating an object of type 'Seq2SeqTrainingArguments'. We kept the "predict_with_generate = True" so that during evaluation the model will be evaluated without using the true targets as input into the decoder. Instead, the decoder will generate the translation by itself in an autoregressive manner. In other words, it will generate one token at a time, and each token it generates will be fed back into the input for the next time step. We also kept "evaluation_strategy = no" and did it manually to reduce the computation time. For the same reason, we also set "fp16 = True" which will cause the model to use floating point numbers with 16 bits instead of 32 bits.

After that, we instantiated a 'Seq2SeqTrainer' object. Then we evaluated our model before we begin the training process. Then we trained our model and evaluated it again to compare the evaluation that we have done before.

VI. RESULT

	Before Training	After Training
eval_loss	1.7008	0.9545
eval_bleu	39.4026	51.2316
eval_bert_score	0.8593	0.8927

TABLE I
EVALUATION OF THE MODEL

The evaluation loss, bleu-score & bert-score are given in the table. After that we have given an english sentence to our model and it has translated the sentence into french quite well. we can see details in Fig. 9

```
[ ]  translator("I hope you are doing well.")

     [{'translation_text': "J'espère que vous allez bien."}]
```

Fig. 9. code1.

VII. CONCLUSION

Neural machine translation is a form of end-to-end learning that encompass all types of machine translation where an artificial neural network is used.In this paper, we present an encoder-decoder approach that encodes a whole input sentence into a fixed-length vector from which a translation will be decoded.This framework is much simpler compared to phrase-based models.It uses an encoder to process a source sentence into vectors for a second recurrent neural network.To increase fluency and accuracy the system uses Google Neural Machine Translation.It applies a large data set for training its algorithms and its end-to-end design allows the system to learn over time and create better, more natural translations."Zero-shot translations" is a direct translation processed by Google Neural Machine Translation.For example, the translation from French to Spanish is a zero-shot translation because it is a direct translation.Google Translation first translate the initial language into English, and then translate that English to the target language.For the proposed model we use two models RNN Encoder–Decoder (RNNencdec) and decoder (RNNsearch).The proposed model (RNNsearch) is much better than the conventional model (RNNencdec) at translating long sentences.

REFERENCES

[1] NIPS-2017-attention-is-all-you-need-Paper.
[2] NEURAL MACHINE TRANSLATION.
[3] Jacob Devlin, Ming-Wei Chang, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 11 Oct 2018 . [Online]. Available: https://arxiv.org/abs/1810.04805?fbclid=IwAR3WVX67aiRP4VgZcOpDEe52ltzDF5YSNn9di8WerAWn4fRZGA. BY JOINTLY LEARNING TO ALIGN AND TRANSLATE
[4] Stéphane Clinchant, "On the use of BERT for Neural Machine Translation", 27 Sept 2019 .
[5] Dzmitry Bahdanau, "Neural Machine Translation by Jointly Learning to Align and Translate", 1 Sep 2014.
[6] Yoshua Bengio, "A Neural Probabilistic Language Model", 02 April 2003.
[7] Dzmitry Bahdanau, "Neural Machine Translation by Jointly Learning to Align and Translate", 1 Sep 2014 .
[8] S. Bengio and Y. Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. IEEE Transactions on Neural Networks, special issue on Data Mining and Knowledge Discovery, 11(3):550–557, 2000a.
[9] S. Bengio and Y. Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. IEEE Transactions on Neural Networks, special issue on Data Mining and Knowledge Discovery, 11(3):550–557, 2000a.
[10] Y. Bengio and J-S. Senécal. Quick training of probabilistic neural nets by importance sampling. In AISTATS, 2003.