Introduction to Neural Machine Translation with GPUs (part 1)

Share:
(https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/)

Posted on **May 27, 2015 (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/)** by **Kyunghyun Cho (https://devblogs.nvidia.com/parallelforall/author/kcho/)** | **6 Comments (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-with-gpus/#disqus_thread)**
Tagged **Deep Learning (https://devblogs.nvidia.com/parallelforall/tag/deep-learning/)**, **Machine Learning (https://devblogs.nvidia.com/parallelforall/tag/machine-learning/)**, **Machine Translation (https://devblogs.nvidia.com/parallelforall/tag/machine-translation/)**, **Natural Language Understanding (https://devblogs.nvidia.com/parallelforall/tag/natural-language-understanding/)**, **Neural Networks (https://devblogs.nvidia.com/parallelforall/tag/neural-networks/)**, **Theano (https://devblogs.nvidia.com/parallelforall/tag/theano/)**

Note: This is the first part of a detailed three-part series on machine translation with neural networks by Kyunghyun Cho. You may enjoy part 2 (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/) and part 3 (http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/).

Neural machine translation is a recently proposed framework for machine translation based purely on neural networks. This post is the first of a series in which I will explain a simple encoder-decoder model for building a neural machine translation system [Cho et al., 2014 (http://arxiv.org/abs/1406.1078); Sutskever et al., 2014 (http://arxiv.org/abs/1409.3215); Kalchbrenner and Blunsom, 2013 (https://scholar.google.com/citations?view_op=view_citation&hl=en&user=LFyg0tAAAAAJ&citation_for_view=LFyg0tAAAAAJ:d1gkVwhDpl0C)]. In a later post I will describe how an attention mechanism can be incorporated into the simple encoder-decoder model [Bahdanau et al., 2015], leading to the state-of-the-art machine translation model for a number of language pairs including En-Fr, En-De, En-Tr and En-Zh [Gulcehre et al., 2015 (http://arxiv.org/abs/1503.03535); Jean et al., 2015 (http://arxiv.org/abs/1412.2007)]. Furthermore, I will introduce recent work which has applied this framework of neural machine translation to image and video description generation [Xu et al., 2015 (http://arxiv.org/abs/1502.03044); Li et al., 2015 (http://arxiv.org/abs/1502.08029)].

Statistical Machine Translation

First, let's start with a brief overview of machine translation. In fact, the name, machine translation, says everything. We want a machine to translate text in one language, which we will call the source sentence, to corresponding text in another language, which we call the target sentence. (Although ideally the machine should be able to translate a whole document from one language to another, let us concentrate in this blog post on sentence-level machine translation.)

There are multiple ways to build such a machine that can translate languages. For instance, we can ask a bilingual speaker to give us a set of rules transforming a source sentence into a correct translation. This is not a great solution, as you can imagine, because we don't even know the set of rules underlying a single language, not to mention the rules underlying a pair of languages. It is simply hopeless to write an exhaustive set of rules for translating a source sentence into a correct translation. Hence, in this blog post, we focus on a statistical approach where those rules, either implicitly or explicitly, are automatically extracted from a large corpus of text.

This statistical approach to machine translation is called statistical machine translation. The goal is the same (build a machine that translates a sentence from one language to another), but we let the machine learn from data how to translate rather than design a set of rules for the machine (See Fig. 1 for a graphical illustration.) Learning is based on statistical methods, which should sound familiar to anyone who has taken a basic course on machine learning. In fact, statistical machine translation is nothing but a particular application of machine learning, where the task is to find a function that maps from a source sentence to a corresponding target.

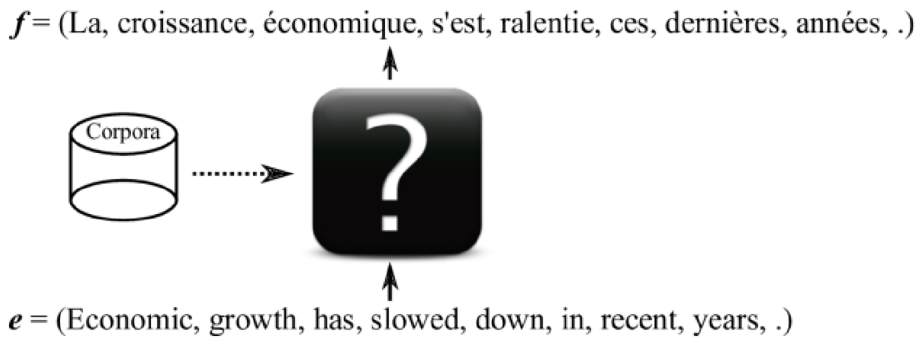

f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Corpora
?
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Figure 1. Statistical Machine Translation

One important characteristics of machine translation is that the target (translation) function is neither one-to-one nor many-to-one as in many other applications of machine learning (such as classification, which is many-to-one), but one-to-many in the sense that one source sentence can be translated into many possible translations. Because of this, we model the translation function not as a deterministic function but as a conditional probability $p(y|x)$ of a target sentence (translation) y given x. The conditional probability may apply an equally high probability to more than one well-separated configurations/sentences, leading to a one-to-many relationship between source and target sentences.

Now let's say you want to build a statistical machine translation system that translates a source sentence in English to a sentence in French. The first and probably most important job is to collect pairs of source sentences and their corresponding translations. I will use x^n and y^n to represent a pair of source and corresponding translation, respectively. The superscript n means that it's the n-th pair in a set of many more pairs (often, we need tens to hundreds of thousands of pairs to train a good translation model.) I'll use $D = (x^1, y^1), \ldots, (x^N, y^N)$ to denote the data set with N pairs.

Where can I get these training pairs? For widely used languages in machine translation, you probably want to check out the Workshop on Statistical Machine Translation (http://www.statmt.org/wmt15/) or the International Workshop on Spoken Language Translation (https://sites.google.com/site/iwsltevaluation2014/home).

With the training data $D = (x^1, y^1), \ldots, (x^N, y^N)$ in hand, we can now score a model by looking at how well the model works on the training data D. The score, which I'll call the log-likelihood of the model, is the average of the log-likelihood of the model on each pair (x^n, y^n). With the probabilistic interpretation of the machine translation model, the log-likelihood of the model on each pair is simply how high a log-probability the model assigns to the pair: $\log p(y^n|x^n, \theta)$, where θ is a set of parameters that defines the model. Then, the overall score of the model on the training data is

$$\mathcal{L}(\theta, D) = \sum_{(x^n, y^n) \in D} \log p(y^n|x^n, \theta).$$

If the log-likelihood $\mathcal{L}$ is low, the model is not giving enough probability mass to the correctly translated pairs, meaning that it's wasting its probability mass on some wrong translations. Thus, we want to find a configuration of the model, or the values of the parameters θ that maximizes this log-likelihood, or score.

In machine learning, this is known as a maximum likelihood estimator. But we're left with a perhaps more important question: how do we model $p(y|x, \theta)$?

Statistical Machine Translation (almost) from Scratch

This question of how to model the conditional distribution $p(y|x)$ has been asked and answered for a long time, starting more than 20 years ago at IBM T.J. Watson Research Center [Brown et al., 1993 (http://www.aclweb.org/anthology/J93-2003) and references therein]. The core of research on statistical machine translation (SMT) since then has been a log-linear model (http://en.wikipedia.org/wiki/Log-linear_model), where we approximate the logarithm of the true $p(y|x)$ with a linear combination of many features:

$$\log p(y|x) \approx \log p(y|x, \theta) = \sum_i \theta_i f_i(x, y) + C(\theta),$$

where C is the normalization constant. In this case, a large part of the research comes down to finding a good set of feature functions f_i, and there is a very well-written textbook that covers all the details about it [Koehn, 2009] (http://www.statmt.org/book/).

In this approach of statistical machine translation, often the only thing left to machine learning is to find a nice set of coefficients θ_i that balance among different features, or to filter/re-rank a set of potential translations decoded from the log-linear model [Schwenk, 2007 (http://www.google.com/url?q=http%3A%2F%2Fwww.sciencedirect.com%2Fscience%2Farticle%2Fpii%2FS0885230806000325&sa=D&sntz=1&usg=AFQjCNGsoe0L0ioiBqvl9tmmFRVPNoz
More specifically, neural networks have been used both as a part of the feature functions or to re-rank so-called n-best lists of possible translations, as in the middle and right panels of Fig. 2.

(file:///Users/mharris/Downloads/NMT/NMT.html#) (file:///Users/mharris/Downloads/NMT/NMT.html#)

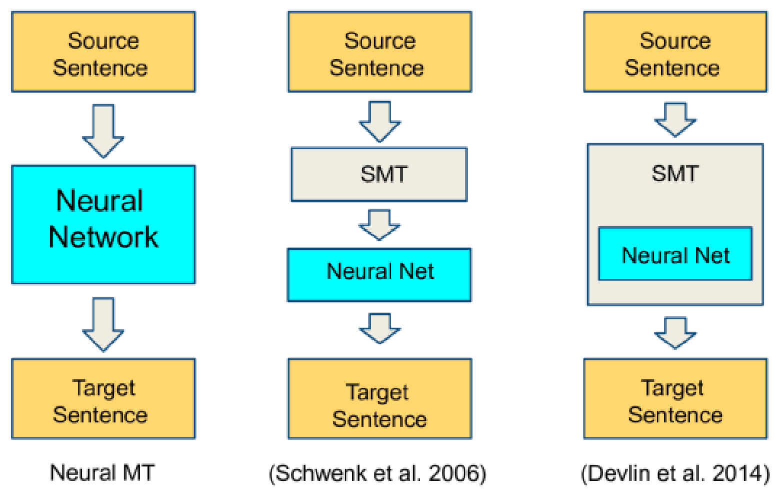


Figure 2. Graphical illustration of Neural MT, SMT+Reranking-by-NN and SMT-NN. From [Bahadanau et al., 2015] slides at ICLR 2015.

In this blog post, on the other hand, I focus on a recently proposed approach, called neural machine translation, where machine learning, and more specifically a neural network, has more or even full control, as in the left panel of Fig. 2.

Neural Machine Translation

As is usual with general deep learning (https://developer.nvidia.com/deep-learning), neural machine translation (NMT) does not rely on pre-designed feature functions. (By pre-designed feature functions, I mean those that are not learned.) Rather, the goal of NMT is to design a fully trainable model of which every component is tuned based on training corpora to maximize its translation performance.

A fully trainable NMT model $\mathcal{M}$ starts from as raw a representation of a source sentence as possible and finishes by generating as raw a representation of a target sentence as possible. Here, let's consider a sequence of words as the most raw representation of a sentence. (This is not true for most natural languages, but without loss of generality, I will consider a word the smallest unit.) Each word in a sequence is represented by its integer index in a vocabulary. For instance, in the vocabulary of English sorted according to frequency, there will be the first word, represented as an integer 1. Let me use $X = (x_1, x_2, \ldots, x_T)$ to denote a source sentence, and $Y = (y_1, y_2, \ldots, y_{T'})$ a target sentence.

Given a source sequence $X = (x_1, x_2, \ldots, x_T)$ of word indices, the NMT model $\mathcal{M}$ computes the conditional probability of $Y = (y_1, y_2, \ldots, y_{T'})$. Next I'll discuss how we can build a neural network to approximate this conditional probability $p(Y|X)$.

Recurrent Neural Networks

One important property of machine translation, or any task based on natural languages, is that we deal with variable-length input $X = (x_1, x_2, \ldots, x_T)$ and output $Y = (y_1, y_2, \ldots, y_{T'})$. In other words, T and T' are not fixed.

To deal with these types of variable-length input and output, we need to use a recurrent neural network (RNN). Widely used feed-forward neural networks, such as convolutional neural networks, do not maintain internal state other than the network's own parameters. Whenever a single sample is fed into a feed-forward neural network, the network's internal state, or the activations of the hidden units, is computed from scratch and is not influenced by the state computed from the previous sample. On the other hand, an RNN maintains its internal state while reading a sequence of inputs, which in our case will be a sequence of words, thereby being able to process an input of any length.

Let me explain this in more detail. The main idea behind RNNs is to compress a sequence of input symbols into a fixed-dimensional vector by using recursion. Assume at step t that we have a vector h_{t-1} which is the history of all the preceding symbols. The RNN will compute the new vector, or its internal state, h_t which compresses all the preceding symbols $(x_1, x_2, \ldots, x_{t-1})$ as well as the new symbol x_t by

$$h_t = \phi_\theta(x_t, h_{t-1}),$$

where ϕ_θ is a function parametrized by θ which takes as input the new symbol x_t and the history h_{t-1} up to the $(t-1)$-th symbol. Initially, we can safely assume that h_0 is an all-zero vector.

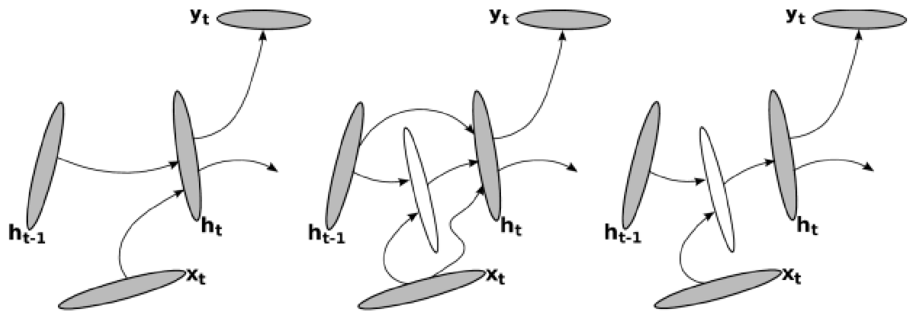


Figure 3. Graphical illustration of different types of recurrent neural networks. From [Pascanu et al., 2014 (http://arxiv.org/abs/1312.6026)]

(file:///Users/mharris/Downloads/NMT/NMT.html#) (file:///Users/mharris/Downloads/NMT/NMT.html#)

The recurrent activation function ϕ is often implemented as, for instance, a simple affine transformation followed by an element-wise nonlinear function:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b).$$

In this formulation, the parameters include the input weight matrix W, the recurrent weight matrix U and the bias vector b. I must say that this is not the only possibility, and there is a very large room for designing a novel recurrent activation function. See Fig. 3 for some examples from [Pascanu et al., 2014 (http://www.google.com/url?q=http%3A%2F%2Farxiv.org%2Fabs%2F1312.6026&sa=D&sntz=1&usg=AFQjCNGd8qr8YSn13g29o9hoQh9TgnhTGg)].

This simple type of RNN can be implemented very easily using (for instance) Theano (http://www.google.com/url?q=http%3A%2F%2Fdeeplearning.net%2Fsoftware%2Ftheano%2F&sa=D&sntz=1&usg=AFQjCNFJoBcoA4NcY2Q-YzLHUW1Ho-Wexw), which allows your RNN to be run on either the CPU or the GPU transparently. See Recurrent Neural Networks with Word Embeddings (http://deeplearning.net/tutorial/rnnslu.html#rnnslu); note that the whole RNN code is written in less than 10 lines (https://github.com/mesnilgr/is13/blob/master/rnn/elman.py#L38-L45)!

Recently, it has been observed that it is better, or easier, to train a recurrent neural network with more sophisticated activation functions such as long short-term memory units [Hochreiter and Schmidhuber, 1997 (http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735?journalCode=neco&)] and gated recurrent units [Cho et al., 2014 (http://arxiv.org/abs/1406.1078)].

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$u_t = \sigma(W_u x_t + r_t \odot (U_u h_{t-1}) + b_u)$$
$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tanh(W x_t + r_t \odot (U_u h_{t-1}) + b)$$

As was the case with the simple recurrent activation function, the parameters here include the input weight matrices W, W_r and W_u, the recurrent weight matrices U, U_r and U_u and the bias vectors b, b_r and b_u.

Although these units look much more complicated than the simple RNN, the implementation with Theano, or any other deep learning framework, such as Torch, is just as simple. For instance, see LSTM Networks for Sentiment Analysis (http://deeplearning.net/tutorial/lstm.html#lstm) (example code (https://github.com/kyunghyuncho/DeepLearningTutorials/blob/master/code/lstm.py#L155-L201)).

I have explained a recurrent neural network (RNN) as a history compressor, but it can also be used to probabilistically model a sequence. Here, by probabilistically modeling a sequence I mean a machine learning model that computes the probability $p(X)$ of any given sequence $X = (x_1, x_2, \ldots, x_T)$. How can we formulate $p(X)$ such that it can be written as a recurrence?

Let's start by rewriting $p(X) = p(x_1, x_2, \ldots, x_T)$ into

$$p(x_1, x_2, \ldots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_T|x_1, \ldots, x_{T-1}),$$

which comes from the definition of conditional probability, $p(X|Y) = \frac{P(X,Y)}{P(Y)}$. From this, we can make a recursive formula such that

$$p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_T|x_1, \ldots, x_{T-1}) = \prod_{t=1}^{T} p(x_t|x_{<t})$$

Now, we let an RNN model $p(x_t|x_{<t})$ at each time t by

$$p(x_t|x_{<t}) = g_\theta(h_{t-1})$$
$$h_{t-1} = \phi_\theta(x_{t-1}, h_{t-2}).$$

g_θ outputs a probability distribution conditioned on the whole history up to the $(t-1)$-th symbol via h_{t-1}. In other words, at each time step, the RNN tries to predict the next symbol given the history of the input symbols.

There are a lot of interesting properties and characteristics of a recurrent neural network that I would love to spend hours talking about, but I will have to stop here for this blog post, since after all, what I have described so far are all the things you need to start building a neural machine translation system. For those who are more interested in recurrent neural networks, I suggest you to read the following papers. Obviously, this list is definitely not exhaustive. Or, you can also check out my slides on how to use recurrent neural networks for language modeling (https://drive.google.com/file/d/0B16RwCMQqrtdNEhwbHN2bXJzdXM/view).

- Graves, Alex. “Generating sequences with recurrent neural networks.” arXiv preprint arXiv:1308.0850 (2013).
- Pascanu, Razvan et al. “How to construct deep recurrent neural networks.” arXiv preprint arXiv:1312.6026 (2013).
- Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent. “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription.” arXiv preprint arXiv:1206.6392 (2012).
- Mikolov, Tomas et al. “Recurrent neural network based language model.” INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010 1 Jan. 2010: 1045-1048.
- Hochreiter, Sepp, and Jürgen Schmidhuber. “Long short-term memory.” Neural computation 9.8 (1997): 1735-1780.
- Cho, Kyunghyun et al. “Learning phrase representations using rnn encoder-decoder for statistical machine translation.” arXiv preprint arXiv:1406.1078 (2014).
- Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult.” Neural Networks, IEEE Transactions on 5.2 (1994): 157-166.

The Story So Far, and What’s Next

In this post, I introduced machine translation, and described how statistical machine translation approaches the problem of machine translation. In the framework of statistical machine translation, I have discussed how neural networks can be used to improve the overall translation performance.

The goal of this blog series is to introduce a novel paradigm for neural machine translation; this post laid the groundwork, concentrating on two key capabilities of recurrent neural networks: sequence summarization and probabilistic modeling of sequences.

Based on these two properties, in the next post, I will describe the actual neural machine translation system based on recurrent neural networks. I’ll also show you why GPUs are so important for Neural Machine Translation! Stay tuned.

RELATED POSTS

Introduction to Neural Machine Translation with GPUs (part 3) (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/)

Introduction to Neural Machine Translation with GPUs (Part 2) (https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/)

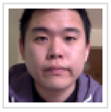
Deep Learning in a Nutshell: Core Concepts (https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/)

Understanding Aesthetics with Deep Learning (https://devblogs.nvidia.com/parallelforall/understanding-aesthetics-deep-learning/)

∥∀

Share:

About Kyunghyun Cho



Kyunghyun Cho is an assistant professor in the Department of Computer Science, Courant Institute of Mathematical Sciences and the Center for Data Science at New York University (NYU) (starting September, 2015). Previously, he was a postdoctoral researcher at the University of Montreal under the supervision of Prof. Yoshua Bengio after obtaining a doctorate degree at Aalto University (Finland) in early 2014. Kyunghyun's main research interests include neural networks, generative models and their applications, especially, to language understanding.
View all posts by Kyunghyun Cho → (https://devblogs.nvidia.com/parallelforall/author/kcho/)

6 Comments   Parallel Forall   1 Login

Recommend 3   Share   Sort by Best

Join the discussion…

Paweł Kaczor • 2 years ago
One equation is missing. Looks like latex error.
2 • Reply • Share ›

Mark Harris Mod → Paweł Kaczor • 2 years ago
Thanks, I've fixed it. Wordpress latex is tricky...
1 • Reply • Share ›

Aanchan mohan • 9 months ago
Had one question, there is a function g_theta specified towards the end of the post to model the conditional probability of p(x|x_(less_than(t))), but it is not defined anywhere. Is g_theta the soft-max function? Also is g_theta used at any point in the training?
• Reply • Share ›

Mark • 2 years ago
Awesome. I can't wait for the next posts!
• Reply • Share ›

korbonits • 2 years ago
This is fantastic. Thanks! Great to include the extra papers as jumping off points.
• Reply • Share ›

Jimmy Ren • 2 years ago
Nice post! Thanks!
• Reply • Share ›

ALSO ON PARALLEL FORALL

Beyond GPU Memory Limits with Unified Memory on Pascal
9 comments • 3 months ago•
Troels Henriksen — That's a shame. Is there any hope of support being added in the future?

Deep Learning in a Nutshell: Reinforcement Learning
3 comments • 6 months ago•
Liza Loop — Thanks for the comments, Carl. I don't think the data we need exists yet in useable form because a) we …

NVIDIA Docker: GPU Server Application Deployment Made Easy
18 comments • 9 months ago•
FelixAbecassis — See this issue: https://github.com/NVIDIA/n...If you spawn a VM and setup GPU …

Mixed-Precision Programming with CUDA 8
1 comment • 5 months ago•
li pengfei — 1. cuDNN 6 will add support for INT8 inference convolutions. Does the INT8 converlution will use …

Subscribe   Add Disqus to your site Add Disqus Add   Privacy

ACCELERATED COMPUTING (HTTPS://DEVELOPER.NVIDIA.COM/ACCELERATED-COMPUTING)

GAMEWORKS (HTTPS://DEVELOPER.NVIDIA.COM/GAMEWORKS)

EMBEDDED COMPUTING (HTTPS://DEVELOPER.NVIDIA.COM/EMBEDDED-COMPUTING)

DESIGNWORKS (HTTPS://DEVELOPER.NVIDIA.COM/DESIGNWORKS)

GET STARTED

About CUDA (https://developer.nvidia.com/about-cuda)

Parallel Computing (https://developer.nvidia.com/accelerated-computing-training)

CUDA Toolkit (https://developer.nvidia.com/cuda-toolkit)

CUDACast (http://www.youtube.com/playlist?list=PL5B692fm6--vScfBaxgY89IRWFzDt0Khm)

LEARN MORE

Training and Courseware (https://developer.nvidia.com/cuda-education-training)

Tools and Ecosystem (https://developer.nvidia.com/tools-ecosystem)

Academic Collaboration (https://developer.nvidia.com/academia)

Documentation (http://docs.nvidia.com/cuda/index.html)

GET INVOLVED

Forums (https://devtalk.nvidia.com/)

Parallel Forall Blog (https://devblogs.nvidia.com/parallelforall/)

Developer Program (https://developer.nvidia.com/cuda-registered-developer-program)

Contact Us (https://developer.nvidia.com/contact)