Google Research Blog

The latest news from Research at Google

MobileNets: Open-Source Models for Efficient On-Device Vision

Wednesday, June 14, 2017

Posted by Andrew G. Howard, Senior Software Engineer and Menglong Zhu, Software Engineer

(Cross-posted on the Google Open Source Blog)

Deep learning has fueled tremendous progress in the field of computer vision in recent years, with neural networks repeatedly pushing the frontier of visual recognition technology. While many of those technologies such as object, landmark, logo and text recognition are provided for internet-connected devices through the Cloud Vision API, we believe that the ever-increasing computational power of mobile devices can enable the delivery of these technologies into the hands of our users, anytime, anywhere, regardless of internet connection. However, visual recognition for on device and embedded applications poses many challenges — models must run quickly with high accuracy in a resource-constrained environment making use of limited computation, power and space.

Today we are pleased to announce the release of MobileNets, a family of *mobile-first* computer vision models for TensorFlow, designed to effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application. MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how other popular large scale models, such as Inception, are used.

Example use cases include detection, fine-grain classification, attributes and geo-localization.

This release contains the model definition for MobileNets in TensorFlow using TF-Slim, as well as 16 pre-trained ImageNet classification checkpoints for use in mobile projects of all sizes. The models can be run efficiently on mobile devices with TensorFlow Mobile.

Model Checkpoint	Million MACs	Million Parameters	Top-1 Accuracy	Top-5 Accuracy
MobileNet_v1_1.0_224	569	4.24	70.7	89.5
MobileNet_v1_1.0_192	418	4.24	69.3	88.9
MobileNet_v1_1.0_160	291	4.24	67.2	87.5
MobileNet_v1_1.0_128	186	4.24	64.1	85.3
MobileNet_v1_0.75_224	317	2.59	68.4	88.2
MobileNet_v1_0.75_192	233	2.59	67.4	87.3
MobileNet_v1_0.75_160	162	2.59	65.2	86.1
MobileNet_v1_0.75_128	104	2.59	61.8	83.6
MobileNet_v1_0.50_224	150	1.34	64.0	85.4
MobileNet_v1_0.50_192	110	1.34	62.1	84.0
MobileNet_v1_0.50_160	77	1.34	59.9	82.5
MobileNet_v1_0.50_128	49	1.34	56.2	79.6
MobileNet_v1_0.25_224	41	0.47	50.6	75.0
MobileNet_v1_0.25_192	34	0.47	49.0	73.6
MobileNet_v1_0.25_160	21	0.47	46.0	70.7
MobileNet_v1_0.25_128	14	0.47	41.3	66.2

Choose the right MobileNet model to fit your latency and size budget. The size of the network in memory

and on disk is proportional to the number of parameters. The latency and power usage of the network scales with the number of Multiply-Accumulates (MACs) which measures the number of fused Multiplication and Addition operations. Top-1 and Top-5 accuracies are measured on the ILSVRC dataset.

We are excited to share MobileNets with the open-source community. Information for getting started can be found at the TensorFlow-Slim Image Classification Library. To learn how to run models on-device please go to TensorFlow Mobile. You can read more about the technical details of MobileNets in our paper, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

Acknowledgements

MobileNets were made possible with the hard work of many engineers and researchers throughout Google. Specifically we would like to thank:

Core Contributors: *Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam*

Special thanks to: *Benoit Jacob, Skirmantas Kligys, George Papandreou, Liang-Chieh Chen, Derek Chow, Sergio Guadarrama, Jonathan Huang, Andre Hentz, Pete Warden*

Labels: Computer Vision , Machine Perception , Neural Networks , open source , TensorFlow

Introducing the TensorFlow Research Cloud

Wednesday, May 17, 2017

Posted by Zak Stone, Product Manager for TensorFlow

Researchers require enormous computational resources to train the machine learning (ML) models that have delivered recent breakthroughs in medical imaging, neural machine translation, game playing, and many other domains. We believe that significantly larger amounts of computation will make it possible for researchers to invent new types of ML models that will be even more accurate and useful.

To accelerate the pace of open machine-learning research, we are introducing the TensorFlow Research Cloud (TFRC), a cluster of 1,000 Cloud TPUs that will be made available free of charge to support a broad range of computationally-intensive research projects that might not be possible otherwise.

The TensorFlow Research Cloud offers researchers the following benefits:

- Access to Google's all-new Cloud TPUs that accelerate both training *and* inference
- Up to 180 teraflops of floating-point performance per Cloud TPU
- 64 GB of ultra-high-bandwidth memory per Cloud TPU
- Familiar TensorFlow programming interfaces

You can sign up here to request to be notified when the TensorFlow Research Cloud application process opens, and you can optionally share more information about your computational needs. We plan to evaluate applications on a rolling basis in search of the most creative and ambitious proposals.

The TensorFlow Research Cloud program is not limited to academia — we recognize that people with a wide range of affiliations, roles, and expertise are making major machine learning research contributions, and we especially encourage those with non-traditional backgrounds to apply. Access will be granted to selected individuals for limited amounts of compute time, and researchers are welcome to apply multiple times with multiple projects.

Since the main goal of the TensorFlow Research Cloud is to benefit the open machine learning research community as a whole, successful applicants will be expected to do the following:

- Share their TFRC-supported research with the world through peer-reviewed publications, open-source code, blog posts, or other open media
- Share concrete, constructive feedback with Google to help us improve the TFRC program and the underlying Cloud TPU platform over time
- Imagine a future in which ML acceleration is abundant and develop new kinds of machine learning models in anticipation of that future

For businesses interested in using Cloud TPUs for proprietary research and development, we will offer a parallel Cloud TPU Alpha program. You can sign up here to learn more about this program. We recommend participating in the Cloud TPU Alpha program if you are interested in any of the following:

- Accelerating training of proprietary ML models; models that take weeks to train on other hardware can be trained in days or even hours on Cloud TPUs
- Accelerating batch processing of industrial-scale datasets: images,

videos, audio, unstructured text, structured data, etc.
- Processing live requests in production using larger and more complex ML models than ever before

We hope the TensorFlow Research Cloud will allow as many researchers as possible to explore the frontier of machine learning research and extend it with new discoveries! We encourage you to sign up today to be among the first to know as more information becomes available.

Labels: Deep Learning , distributed systems , Machine Learning , Research , TensorFlow , TPU

Updating Google Maps with Deep Learning and Street View

Wednesday, May 03, 2017

Posted by Julian Ibarz, Staff Software Engineer, Google Brain Team and Sujoy Banerjee, Product Manager, Ground Truth Team

Every day, Google Maps provides useful directions, real-time traffic information and information on businesses to millions of people. In order to provide the best experience for our users, this information has to constantly mirror an ever-changing world. While Street View cars collect millions of images daily, it is impossible to manually analyze more than 80 billion high resolution images collected to date in order to find new, or updated, information for Google Maps. One of the goals of the Google's Ground Truth team is to enable the automatic extraction of information from our geo-located imagery to improve Google Maps.

In "Attention-based Extraction of Structured Information from Street View Imagery", we describe our approach to accurately read street names out of very challenging Street View images in many countries, automatically, using a deep neural network. Our algorithm achieves 84.2% accuracy on the challenging French Street Name Signs (FSNS) dataset, significantly outperforming the previous state-of-the-art systems. Importantly, our system is easily extensible to extract other types of information out of Street View images as well, and now helps us automatically extract business names from store fronts. We are excited to announce that this model is now publicly available!

Avenue des Sapins

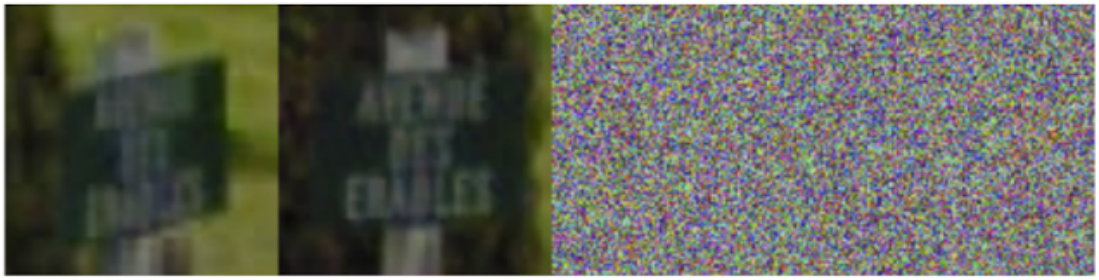
Example of street name from the FSNS dataset correctly transcribed by our system. Up to four views of the same sign are provided.

Text recognition in a natural environment is a challenging computer vision and machine learning problem. While traditional Optical Character Recognition (OCR) systems mainly focus on extracting text from scanned documents, text acquired from natural scenes is more challenging due to visual artifacts, such as distortion, occlusions, directional blur, cluttered background or different viewpoints. Our efforts to solve this research challenge first began in 2008, when we used neural networks to blur faces and license plates in Street View images to protect the privacy of our users. From this initial research, we realized that with enough labeled data, we could additionally use machine learning not only to protect the privacy of our users, but also to automatically improve Google Maps with relevant up-to-date information.

In 2014, Google's Ground Truth team published a state-of-the-art method for reading street numbers on the Street View House Numbers (SVHN) dataset, implemented by then summer intern (now Googler) Ian Goodfellow. This work was not only of academic interest but was critical in making Google Maps more accurate. Today, over one-third of addresses globally have had their location improved thanks to this system. In some countries, such as Brazil, this algorithm has improved more than 90% of the addresses in Google Maps today, greatly improving the usability of our maps.

The next logical step was to extend these techniques to street names. To solve this problem, we created and released French Street Name Signs (FSNS), a large training dataset of more than 1 million street names. The FSNS dataset was a multi-year effort designed to allow anyone to improve their OCR models on a challenging and real use case. FSNS dataset is much larger and more challenging than SVHN in that accurate recognition of street signs may require combining information from many different images.

Route de la Ruaz



Avenue des Erables

These are examples of challenging signs that are properly transcribed by our system by selecting or combining understanding across images. The second example is extremely challenging by itself, but the model learned a language model prior that enables it to remove ambiguity and correctly read the street name. Note that in the FSNS dataset, random noise is used in the case where less than four independent views are available of the same physical sign.

With this training set, Google intern Zbigniew Wojna spent the summer of 2016 developing a deep learning model architecture to automatically label new Street View imagery. One of the interesting strengths of our new model is that it can normalize the text to be consistent with our naming conventions, as well as ignore extraneous text, directly from the data itself.

Avenida Presidente Castelo Branco

Example of text normalization learned from data in Brazil. Here it changes “AV.” into “Avenida” and “Pres.” into “Presidente” which is what we desire.

Midway Avenue

In this example, the model is not confused from the fact that there is two street names, properly normalizes “Av” into “Avenue” as well as correctly ignores the number “1600”.

While this model is accurate, it did show a sequence error rate of 15.8%. However, after analyzing failure cases, we found that 48% of them were due to ground truth errors, highlighting the fact that this model is on par with the label quality (a full analysis our error rate can be found in our paper).

This new system, combined with the one extracting street numbers, allows us to create new addresses directly from imagery, where we previously didn’t know the name of the street, or the location of the addresses. Now, whenever a Street View car drives on a newly built road, our system can analyze the tens of thousands of images that would be captured, extract the street names and numbers, and properly create and locate the new addresses, automatically, on Google Maps.

But automatically creating addresses for Google Maps is not enough -- additionally we want to be able to provide navigation to businesses by name. In 2015, we published “Large Scale Business Discovery from Street View Imagery”, which proposed an approach to accurately detect business store-front signs in Street View images. However, once a store front is detected, one still needs to accurately extract its name for it to be useful -- the model must figure out which text is the business name, and which text is not relevant. We call this extracting “structured text” information out of imagery. It is not just text, it is text with semantic meaning attached to it.

Using different training data, the same model architecture that we used to read street names can also be used to accurately extract business names out of business facades. In this particular case, we are able to only extract the business name which enables us to verify if we already know about this business in Google Maps, allowing us to have more accurate and up-to-date business listings.

The system is correctly able to predict the business name 'Zelina Pneus', despite not receiving any data about the true location of the name in the image. Model is not confused by the tire brands that the sign indicates are available at the store.

Applying these large models across our more than 80 billion Street View images requires a lot of computing power. This is why the Ground Truth team was the first user of Google's TPUs, which were publicly announced earlier this year, to drastically reduce the computational cost of the inferences of our pipeline.

People rely on the accuracy of Google Maps in order to assist them. While keeping Google Maps up-to-date with the ever-changing landscape of cities, roads and businesses presents a technical challenge that is far from solved, it is the goal of the Ground Truth team to drive cutting-edge innovation in machine learning to create a better experience for over one billion Google Maps users.

140 条评

Labels: Computer Vision , Deep Learning , Google Brain , Optical Character Recognition , TensorFlow

Introducing tf-seq2seq: An Open Source Sequence-to-Sequence Framework in TensorFlow

Tuesday, April 11, 2017

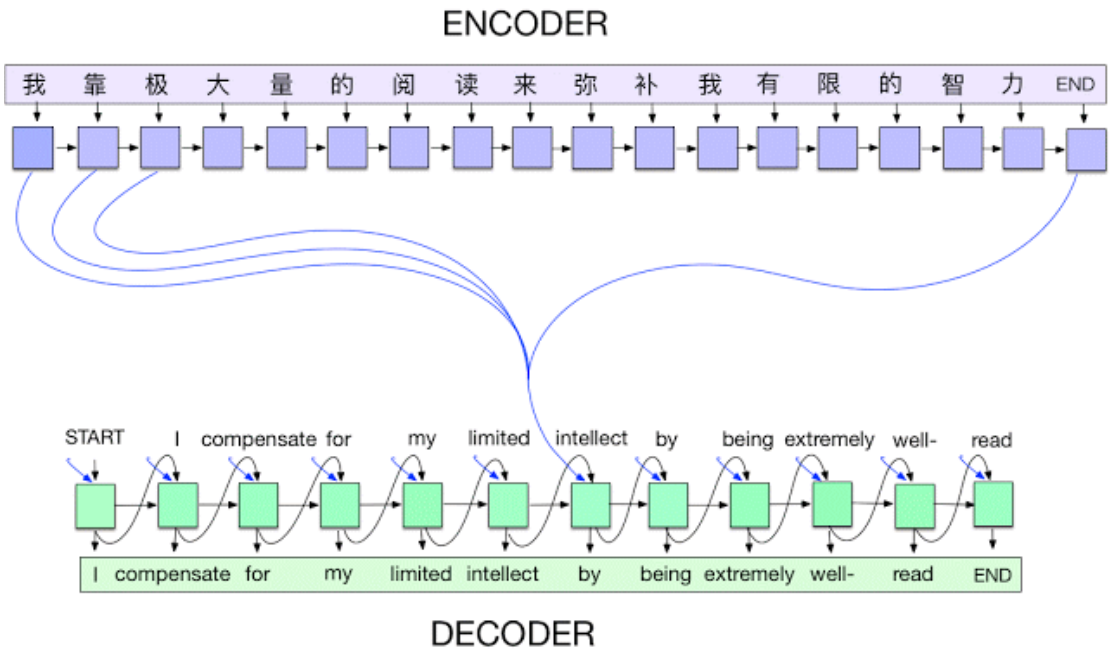
Posted by Anna Goldie and Denny Britz, Research Software Engineer and Google Brain Resident, Google Brain Team

(Crossposted on the Google Open Source Blog)

Last year, we announced Google Neural Machine Translation (GNMT), a sequence-to-sequence ("seq2seq") model which is now used in Google Translate production systems. While GNMT achieved huge improvements in translation quality, its impact was limited by the fact that the framework for training these models was unavailable to external researchers.

Today, we are excited to introduce tf-seq2seq, an open source seq2seq framework in TensorFlow that makes it easy to experiment with seq2seq models and achieve state-of-the-art results. To that end, we made the tf-seq2seq codebase clean and modular, maintaining full test coverage and documenting all of its functionality.

Our framework supports various configurations of the standard seq2seq model, such as depth of the encoder/decoder, attention mechanism, RNN cell type, or beam size. This versatility allowed us to discover optimal hyperparameters and outperform other frameworks, as described in our paper, "Massive Exploration of Neural Machine Translation Architectures."



A seq2seq model translating from Mandarin to English. At each time step, the encoder takes in one Chinese character and its own previous state (black arrow), and produces an output vector (blue arrow). The decoder then generates an English translation word-by-word, at each time step taking in the last word, the previous state, and a weighted combination of all the outputs of the encoder (aka attention [3], depicted in blue) and then producing the next English word. Please note that in our implementation we use wordpieces [4] to handle rare words.

In addition to machine translation, tf-seq2seq can also be applied to any other sequence-to-sequence task (i.e. learning to produce an output sequence given an input sequence), including machine summarization, image captioning, speech recognition, and conversational modeling. We carefully designed our framework to maintain this level of generality and provide tutorials, preprocessed data, and other utilities for machine translation.

We hope that you will use tf-seq2seq to accelerate (or kick off) your own deep learning research. We also welcome your contributions to our GitHub repository, where we have a variety of open issues that we would love to have your help with!

Acknowledgments:

We'd like to thank Eugene Brevdo, Melody Guan, Lukasz Kaiser, Quoc V. Le, Thang Luong, and Chris Olah for all their help. For a deeper dive into how seq2seq models work, please see the resources below.

References:

[1] Massive Exploration of Neural Machine Translation Architectures, *Denny Britz,*

Anna Goldie, Minh-Thang Luong, Quoc Le

[2] Sequence to Sequence Learning with Neural Networks, *Ilya Sutskever, Oriol Vinyals, Quoc V. Le. NIPS, 2014*

[3] Neural Machine Translation by Jointly Learning to Align and Translate, *Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. ICLR, 2015*

[4] Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, *Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean. Technical Report, 2016*

[5] Attention and Augmented Recurrent Neural Networks, *Chris Olah, Shan Carter. Distill, 2016*

[6] Neural Machine Translation and Sequence-to-sequence Models: A Tutorial, *Graham Neubig*

[7] Sequence-to-Sequence Models, *TensorFlow.org*

Labels: Deep Learning , Google Brain , Machine Translation , open source , TensorFlow

An Upgrade to SyntaxNet, New Models and a Parsing Competition

Wednesday, March 15, 2017

Posted by David Weiss and Slav Petrov, Research Scientists

At Google, we continuously improve the language understanding capabilities used in applications ranging from generation of email responses to translation. Last summer, we open-sourced SyntaxNet, a neural-network framework for analyzing and understanding the grammatical structure of sentences. Included in our release was Parsey McParseface, a state-of-the-art model that we had trained for analyzing English, followed quickly by a collection of pre-trained models for 40 additional languages, which we dubbed Parsey's Cousins. While we were excited to share our research and to provide these resources to the broader community, building machine learning systems that work well for languages other than English remains an ongoing challenge. We are excited to announce a few new research resources,

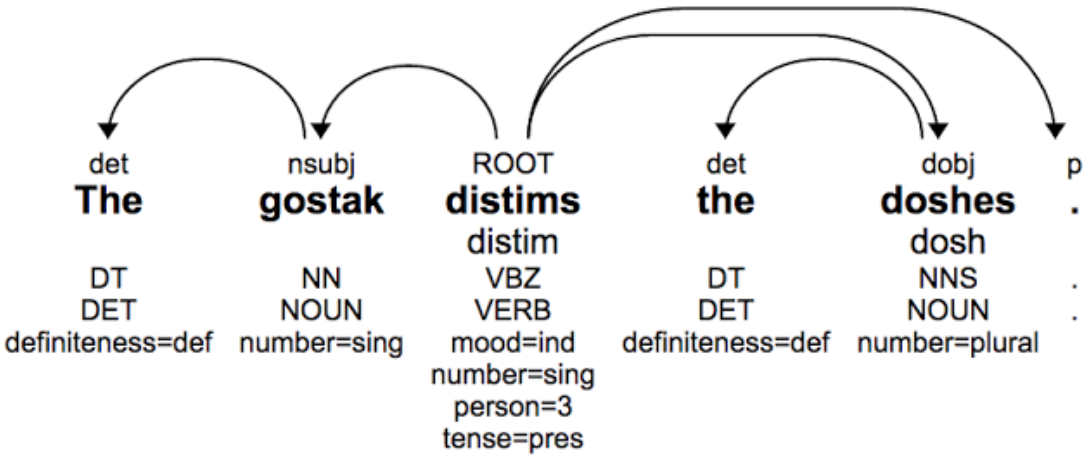
available now, that address this problem.

SyntaxNet Upgrade

We are releasing a major upgrade to SyntaxNet. This upgrade incorporates nearly a year's worth of our research on multilingual language understanding, and is available to anyone interested in building systems for processing and understanding text. At the core of the upgrade is a new technology that enables learning of richly layered representations of input sentences. More specifically, the upgrade extends TensorFlow to allow joint modeling of multiple levels of linguistic structure, and to allow neural-network architectures to be created dynamically during processing of a sentence or document.

Our upgrade makes it, for example, easy to build character-based models that learn to compose individual characters into words (e.g. 'c-a-t' spells 'cat'). By doing so, the models can learn that words can be related to each other because they share common parts (e.g. 'cats' is the plural of 'cat' and shares the same stem; 'wildcat' is a type of 'cat'). Parsey and Parsey's Cousins, on the other hand, operated over sequences of words. As a result, they were forced to memorize words seen during training and relied mostly on the context to determine the grammatical function of previously unseen words.

As an example, consider the following (meaningless but grammatically correct) sentence:



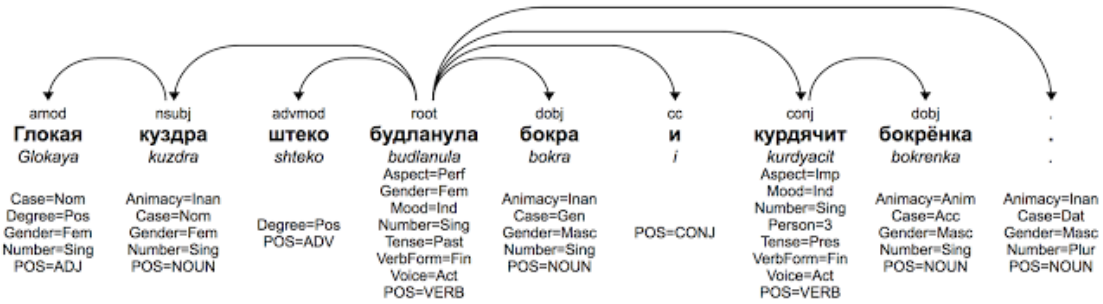
This sentence was originally coined by Andrew Ingraham who explained: "You do not know what this means; nor do I. But if we assume that it is English, we know that the *doshes* are *distimmed* by the *gostak*. We know too that one *distimmer* of *doshes* is a *gostak*." Systematic patterns in morphology and syntax allow us to guess the grammatical function of words even when they are completely novel: we understand that 'doshes' is the plural of the noun 'dosh' (similar to the 'cats'

example above) or that ‘distim’ is the third person singular of the verb distim. Based on this analysis we can then derive the overall structure of this sentence even though we have never seen the words before.

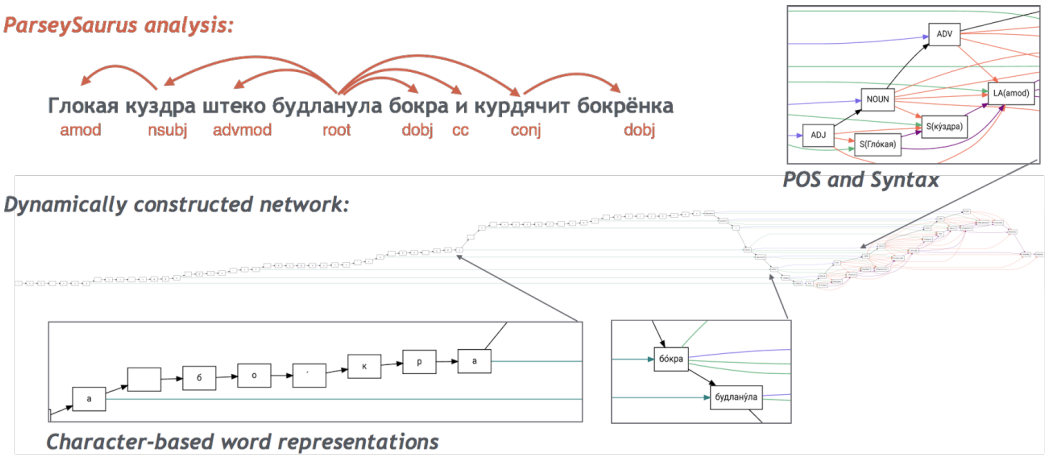
ParseySaurus

To showcase the new capabilities provided by our upgrade to SyntaxNet, we are releasing a set of new pretrained models called ParseySaurus. These models use the character-based input representation mentioned above and are thus much better at predicting the meaning of new words based both on their spelling and how they are used in context. The ParseySaurus models are far more accurate than Parsey’s Cousins (reducing errors by as much as 25%), particularly for morphologically-rich languages like Russian, or agglutinative languages like Turkish and Hungarian. In those languages there can be dozens of forms for each word and many of these forms might never be observed during training - even in a very large corpus.

Consider the following fictitious Russian sentence, where again the stems are meaningless, but the suffixes define an unambiguous interpretation of the sentence structure:



Even though our Russian ParseySaurus model has never seen these words, it can correctly analyze the sentence by inspecting the character sequences which constitute each word. In doing so, the system can determine many properties of the words (notice how many more morphological features there are here than in the English example). To see the sentence as ParseySaurus does, here is a visualization of how the model analyzes this sentence:



Each square represents one node in the neural network graph, and lines show the connections between them. The left-side "tail" of the graph shows the model consuming the input as one long string of characters. These are intermittently passed to the right side, where the rich web of connections shows the model composing words into phrases and producing a syntactic parse. Check out the full-size rendering here.

A Competition

You might be wondering whether character-based modeling are all we need or whether there are other techniques that might be important. SyntaxNet has lots more to offer, like beam search and different training objectives, but there are of course also many other possibilities. To find out what works well in practice we are helping co-organize, together with Charles University and other colleagues, a multilingual parsing competition at this year's Conference on Computational Natural Language Learning (CoNLL) with the goal of building syntactic parsing systems that work well in real-world settings and for 45 different languages.

The competition is made possible by the Universal Dependencies (UD) initiative, whose goal is to develop cross-linguistically consistent treebanks. Because machine learned models can only be as good as the data that they have access to, we have been contributing data to UD since 2013. For the competition, we partnered with UD and DFKI to build a new multilingual evaluation set consisting of 1000 sentences that have been translated into 20+ different languages and annotated by linguists with parse trees. This evaluation set is the first of its kind (in the past, each language had its own independent evaluation set) and will enable more consistent cross-lingual comparisons. Because the sentences have the same meaning and have been annotated according to the same guidelines, we will be able to get closer to answering the question of which languages might be harder to parse.

We hope that the upgraded SyntaxNet framework and our the pre-trained

ParseySaurus models will inspire researchers to participate in the competition. We have additionally created a tutorial showing how to load a Docker image and train models on the Google Cloud Platform, to facilitate participation by smaller teams with limited resources. So, if you have an idea for making your own models with the SyntaxNet framework, sign up to compete! We believe that the configurations that we are releasing are a good place to start, but we look forward to seeing how participants will be able to extend and improve these models or perhaps create better ones!

Thanks to everyone involved who made this competition happen, including our collaborators at UD-Pipe, who provide another baseline implementation to make it easy to enter the competition. Happy parsing from the main developers, Chris Alberti, Daniel Andor, Ivan Bogatyy, Mark Omernick, Zora Tung and Ji Ma!

Labels: Natural Language Processing , Natural Language Understanding , open source , TensorFlow

Preprocessing for Machine Learning with tf.Transform

Wednesday, February 22, 2017

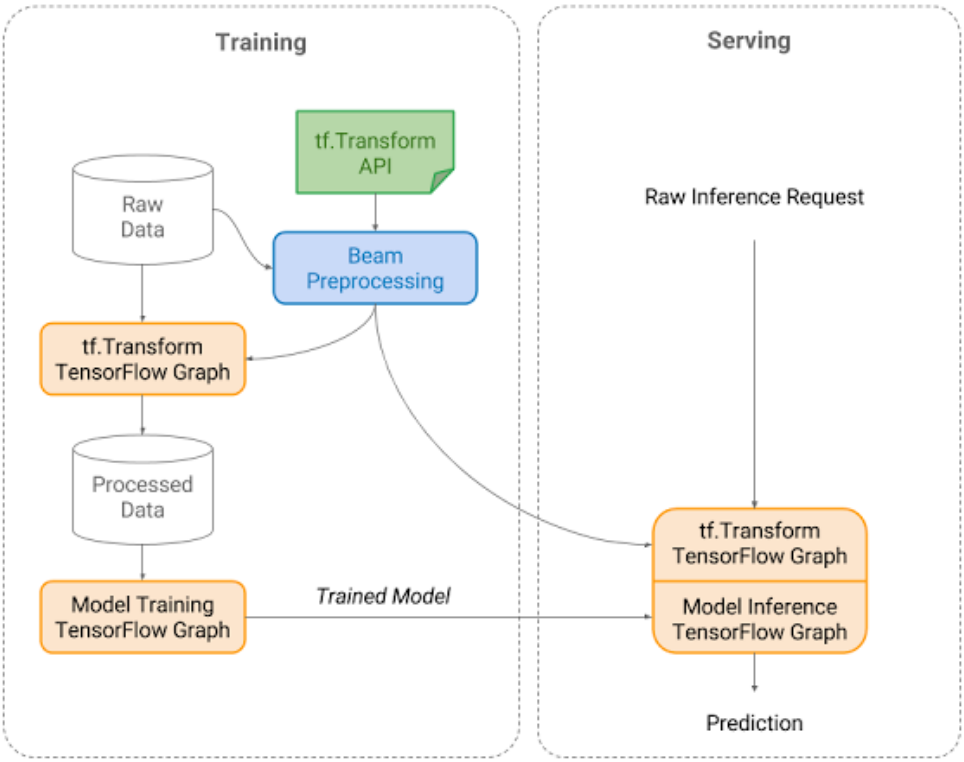
Posted by Kester Tong, David Soergel, and Gus Katsiapis, Software Engineers

When applying machine learning to real world datasets, a lot of effort is required to preprocess data into a format suitable for standard machine learning models, such as neural networks. This preprocessing takes a variety of forms, from converting between formats, to tokenizing and stemming text and forming vocabularies, to performing a variety of numerical operations such as normalization.

Today we are announcing tf.Transform, a library for TensorFlow that allows users to define preprocessing pipelines and run these using large scale data processing frameworks, while also exporting the pipeline in a way that can be run as part of a TensorFlow graph. Users define a pipeline by composing modular Python functions, which tf.Transform then executes with Apache Beam, a framework for large-scale, efficient, distributed data processing. Apache Beam pipelines can be run on Google Cloud Dataflow with planned support for running with other frameworks. The TensorFlow graph exported by tf.Transform enables the preprocessing steps to be replicated when the trained model is used to make predictions, such as when serving the model with Tensorflow Serving.

A common problem encountered when running machine learning models in production is "training-serving skew", where the data seen at serving time differs in some way from the data used to train the model, leading to reduced prediction quality. tf.Transform ensures that no skew can arise during preprocessing, by guaranteeing that the serving-time transformations are exactly the same as those performed at training time, in contrast to when training-time and serving-time preprocessing are implemented separately in two different environments (e.g., Apache Beam and TensorFlow, respectively).

In addition to facilitating preprocessing, tf.Transform allows users to compute summary statistics for their datasets. Understanding the data is very important in every machine learning project, as subtle errors can arise from making wrong assumptions about what the underlying data look like. By making the computation of summary statistics easy and efficient, tf.Transform allows users to check their assumptions about both raw and preprocessed data.



tf.Transform allows users to define a preprocessing pipeline. Users can materialize the preprocessed data for use in TensorFlow training, and also export a tf.Transform graph that encodes the transformations as a TensorFlow graph. This transformation graph can then be incorporated into the model graph used for inference.

We're excited to be releasing this latest addition to the TensorFlow ecosystem, and

we hope users will find it useful for preprocessing and understanding their data.

Acknowledgements

We wish to thank the following members of the tf.Transform team for their contributions to this project: Clemens Mewald, Robert Bradshaw, Rajiv Bharadwaja, Elmer Garduno, Afshin Rostamizadeh, Neoklis Polyzotis, Abhi Rao, Joe Toth, Neda Mirian, Dinesh Kulkarni, Robbie Haertel, Cyril Bortolato and Slaven Bilac. We also wish to thank the TensorFlow, TensorFlow Serving and Cloud Dataflow teams for their support.

66 条



Labels: Machine Learning , TensorFlow

Announcing TensorFlow 1.0

Wednesday, February 15, 2017

Posted by Amy McDonald Sandjideh, Technical Program Manager, TensorFlow

In just its first year, TensorFlow has helped researchers, engineers, artists, students, and many others make progress with everything from language translation to early detection of skin cancer and preventing blindness in diabetics. We're excited to see people using TensorFlow in over 6000 open-source repositories online.

Today, as part of the first annual TensorFlow Developer Summit, hosted in Mountain View and livestreamed around the world, we're announcing **TensorFlow 1.0**:

It's faster: TensorFlow 1.0 is incredibly fast! XLA lays the groundwork for even more performance improvements in the future, and tensorflow.org now includes tips & tricks for tuning your models to achieve maximum speed. We'll soon publish updated implementations of several popular models to show how to take full advantage of TensorFlow 1.0 - including a 7.3x speedup on 8 GPUs for Inception v3 and 58x speedup for distributed Inception v3 training on 64 GPUs!

It's more flexible: TensorFlow 1.0 introduces a high-level API for TensorFlow, with tf.layers, tf.metrics, and tf.losses modules. We've also announced the inclusion of a new tf.keras module that provides full compatibility with Keras, another popular high-level neural networks library.

It's more production-ready than ever: TensorFlow 1.0 promises Python API stability (details here), making it easier to pick up new features without worrying about breaking your existing code.

Other highlights from TensorFlow 1.0:

- Python APIs have been changed to resemble NumPy more closely. For this and other backwards-incompatible changes made to support API stability going forward, please use our handy migration guide and conversion script.
- Experimental APIs for Java and Go
- Higher-level API modules tf.layers, tf.metrics, and tf.losses - brought over from tf.contrib.learn after incorporating skflow and TF Slim
- Experimental release of XLA, a domain-specific compiler for TensorFlow graphs, that targets CPUs and GPUs. XLA is rapidly evolving - expect to see more progress in upcoming releases.
- Introduction of the TensorFlow Debugger (tfdbg), a command-line interface and API for debugging live TensorFlow programs.
- New Android demos for object detection and localization, and camera-based image stylization.
- Installation improvements: Python 3 docker images have been added, and TensorFlow's pip packages are now PyPI compliant. This means TensorFlow can now be installed with a simple invocation of `pip install tensorflow`.

We're thrilled to see the pace of development in the TensorFlow community around the world. To hear more about TensorFlow 1.0 and how it's being used, you can watch the TensorFlow Developer Summit talks on YouTube, covering recent updates from higher-level APIs to TensorFlow on mobile to our new XLA compiler, as well as the exciting ways that TensorFlow is being used:

Click here for a link to the livestream and video playlist (individual talks will be posted online later in the day).

The TensorFlow ecosystem continues to grow with new techniques like Fold for dynamic batching and tools like the Embedding Projector along with updates to our existing tools like TensorFlow Serving. We're incredibly grateful to the community of contributors, educators, and researchers who have made advances in deep learning available to everyone. We look forward to working with you on forums like GitHub issues, Stack Overflow, @TensorFlow, the discuss@tensorflow.org group and at future events.

43 条

Labels: Google Brain , Machine Learning , Neural Networks , open source , TensorFlow

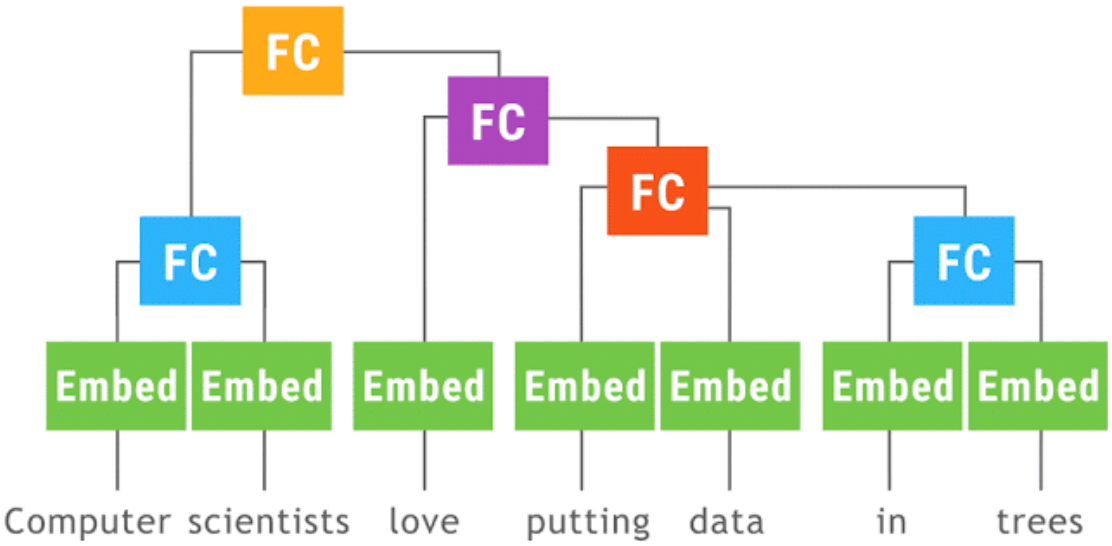
Announcing TensorFlow Fold: Deep Learning With Dynamic Computation Graphs

Tuesday, February 07, 2017

Posted by Moshe Looks, Marcello Herreshoff and DeLesley Hutchins, Software Engineers

In much of machine learning, data used for training and inference undergoes a preprocessing step, where multiple inputs (such as images) are scaled to the same dimensions and stacked into batches. This lets high-performance deep learning libraries like TensorFlow run the same computation graph across all the inputs in the batch in parallel. Batching exploits the SIMD capabilities of modern GPUs and multi-core CPUs to speed up execution. However, there are many problem domains where the size and structure of the input data varies, such as parse trees in natural language understanding, abstract syntax trees in source code, DOM trees for web pages and more. In these cases, the different inputs have different computation graphs that don't naturally batch together, resulting in poor processor, memory, and cache utilization.

Today we are releasing TensorFlow Fold to address these challenges. TensorFlow Fold makes it easy to implement deep-learning models that operate over data of varying size and structure. Furthermore, TensorFlow Fold brings the benefits of batching to such models, resulting in a speedup of more than 10x on CPU, and more than 100x on GPU, over alternative implementations. This is made possible by *dynamic batching*, introduced in our paper *Deep Learning with Dynamic Computation Graphs*.



This animation shows a recursive neural network run with dynamic batching. Operations with the same color are batched together, which lets TensorFlow run them faster. The Embed operation converts words to vector representations. The fully connected (FC) operation combines word vectors to form vector representations of phrases. The output of the network is a vector representation of an entire sentence. Although only a single parse tree of a sentence is shown, the same network can run, and batch together operations, over multiple parse trees of arbitrary shapes and sizes.

The TensorFlow Fold library will initially build a separate computation graph from each input.

Because the individual inputs may have different sizes and structures, the computation graphs may as well. Dynamic batching then automatically combines these graphs to take advantage of opportunities for batching, both within and across inputs, and inserts additional instructions to move data between the batched operations (see our paper for technical details).

To learn more, head over to our github site. We hope that TensorFlow Fold will be useful for researchers and practitioners implementing neural networks with dynamic computation graphs in TensorFlow.

Acknowledgements

This work was done under the supervision of Peter Norvig.

Labels: Machine Learning , open source , TensorFlow

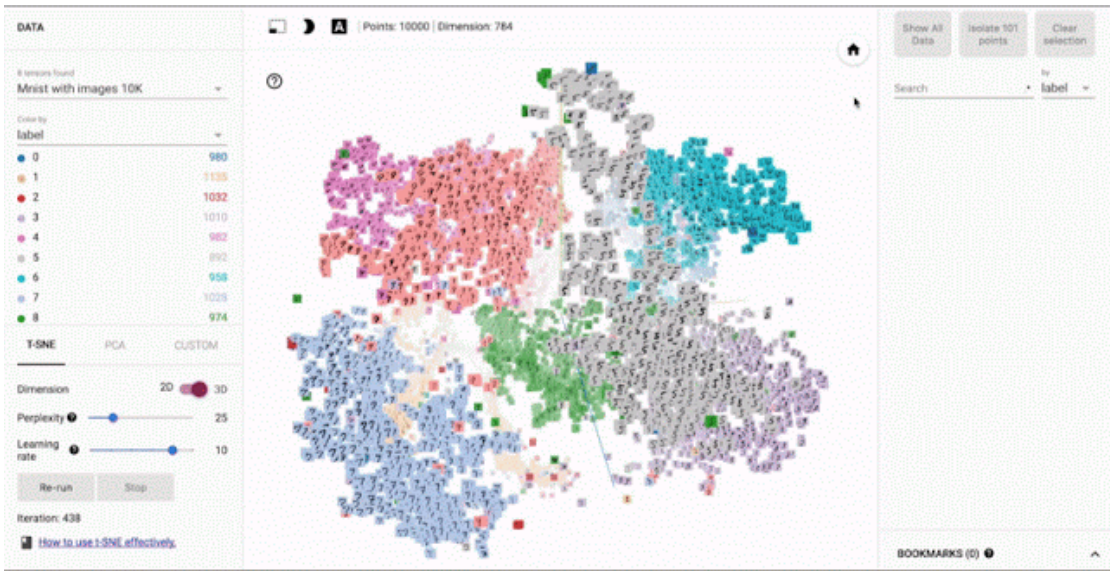
Open sourcing the Embedding Projector: a tool for visualizing high dimensional data

Wednesday, December 07, 2016

Posted by Daniel Smilkov and the Big Picture group

Recent advances in Machine Learning (ML) have shown impressive results, with applications ranging from image recognition, language translation, medical diagnosis and more. With the widespread adoption of ML systems, it is increasingly important for research scientists to be able to explore how the data is being interpreted by the models. However, one of the main challenges in exploring this data is that it often has hundreds or even thousands of dimensions, requiring special tools to investigate the space.

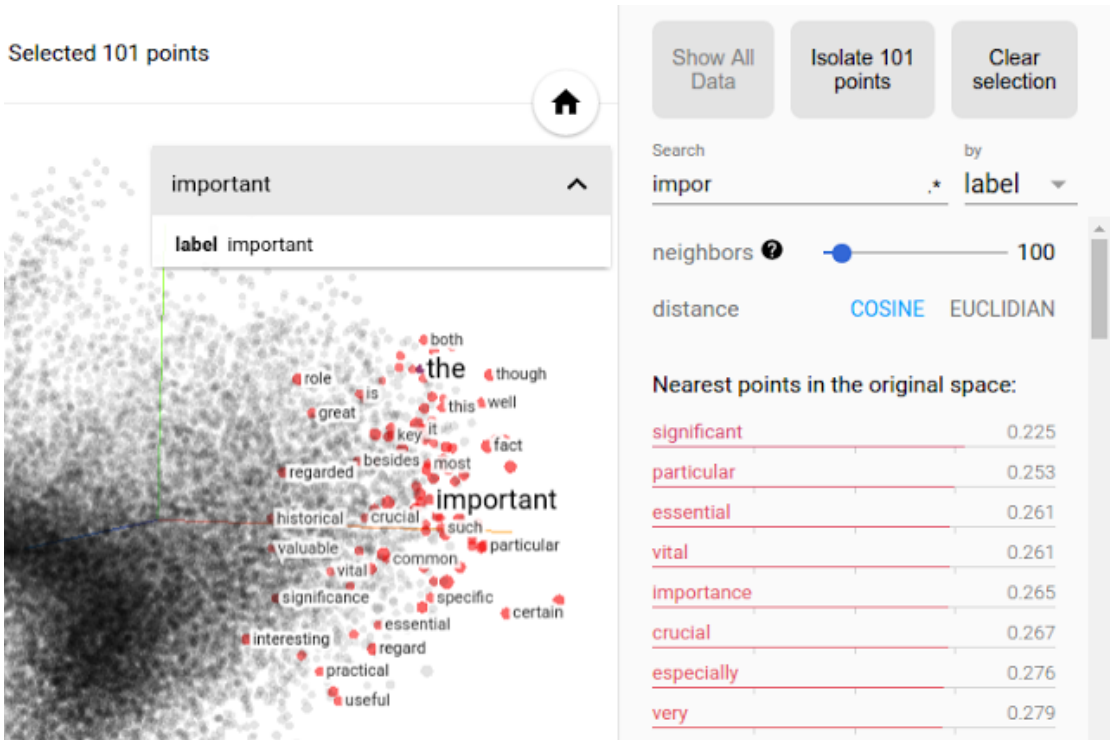
To enable a more intuitive exploration process, we are open-sourcing the Embedding Projector, a web application for interactive visualization and analysis of high-dimensional data recently shown as an A.I. Experiment, as part of TensorFlow. We are also releasing a standalone version at projector.tensorflow.org, where users can visualize their high-dimensional data without the need to install and run TensorFlow.



Exploring Embeddings

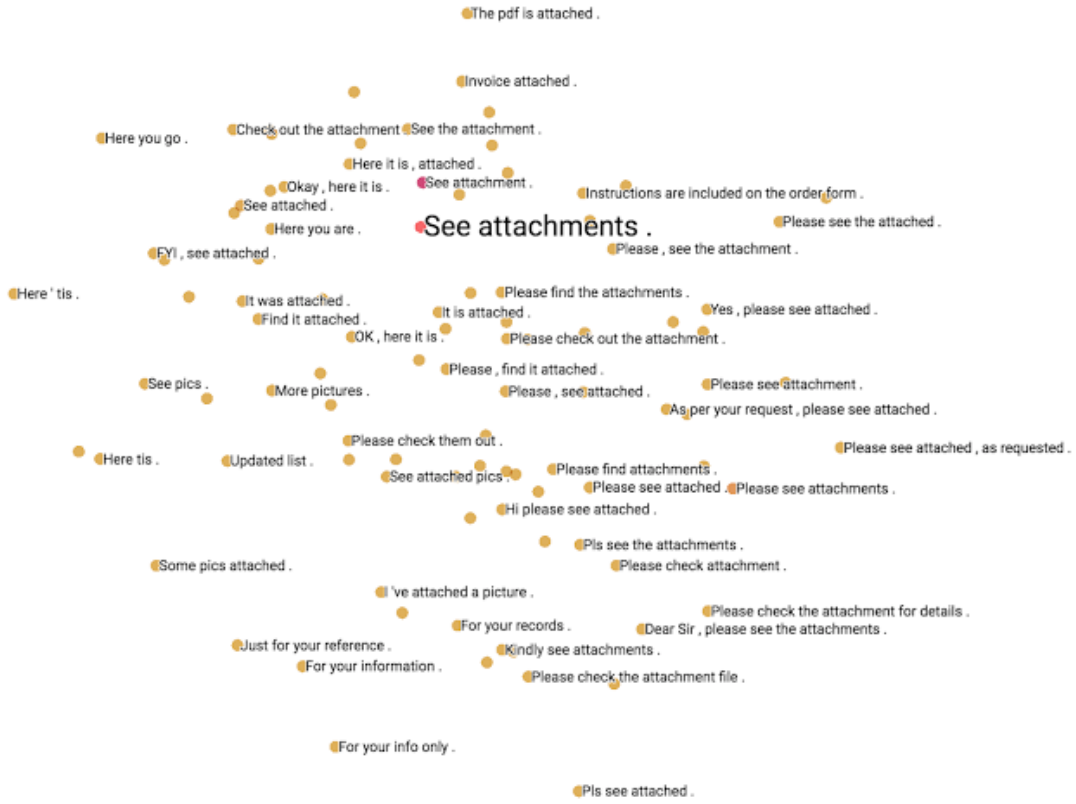
The data needed to train machine learning systems comes in a form that computers don't immediately understand. To translate the things we understand naturally (e.g. words, sounds, or videos) to a form that the algorithms can process, we use *embeddings*, a mathematical vector representation that captures different facets (dimensions) of the data. For example, in this language embedding, similar words are mapped to points that are close to each other.

With the Embedding Projector, you can navigate through views of data in either a 2D or a 3D mode, zooming, rotating, and panning using natural click-and-drag gestures. Below is a figure showing the nearest points to the embedding for the word “important” after training a TensorFlow model using the word2vec tutorial. Clicking on any point (which represents the learned embedding for a given word) in this visualization, brings up a list of nearest points and distances, which shows which words the algorithm has learned to be semantically related. This type of interaction represents an important way in which one can explore how an algorithm is performing.



Methods of Dimensionality Reduction

The Embedding Projector offers three commonly used methods of data dimensionality reduction, which allow easier visualization of complex data: PCA, t-SNE and custom linear projections. PCA is often effective at exploring the internal structure of the embeddings, revealing the most influential dimensions in the data. t-SNE, on the other hand, is useful for exploring local neighborhoods and finding clusters, allowing developers to make sure that an embedding preserves the meaning in the data (e.g. in the MNIST dataset, seeing that the same digits are clustered together). Finally, custom linear projections can help discover meaningful "directions" in data sets - such as the distinction between a formal and casual tone in a language generation model - which would allow the design of more adaptable ML systems.



A custom linear projection of the 100 nearest points of "See attachments." onto the "yes" - "yeah" vector ("yes" is right, "yeah" is left) of a corpus of 35k frequently used phrases in emails

The Embedding Projector website includes a few datasets to play with. We've also made it easy for users to publish and share their embeddings with others (just click on the "Publish" button on the left pane). It is our hope that the Embedding Projector will be a useful tool to help the research community explore and refine their ML applications, as well as enable anyone to better understand how ML algorithms interpret data. If you'd like to get the full details on the Embedding Projector, you can read the paper here. Have fun exploring the world of embeddings!

Labels: Google Brain , Machine Learning , open source , TensorFlow , Visualization

Zero-Shot Translation with Google's Multilingual

Neural Machine Translation System

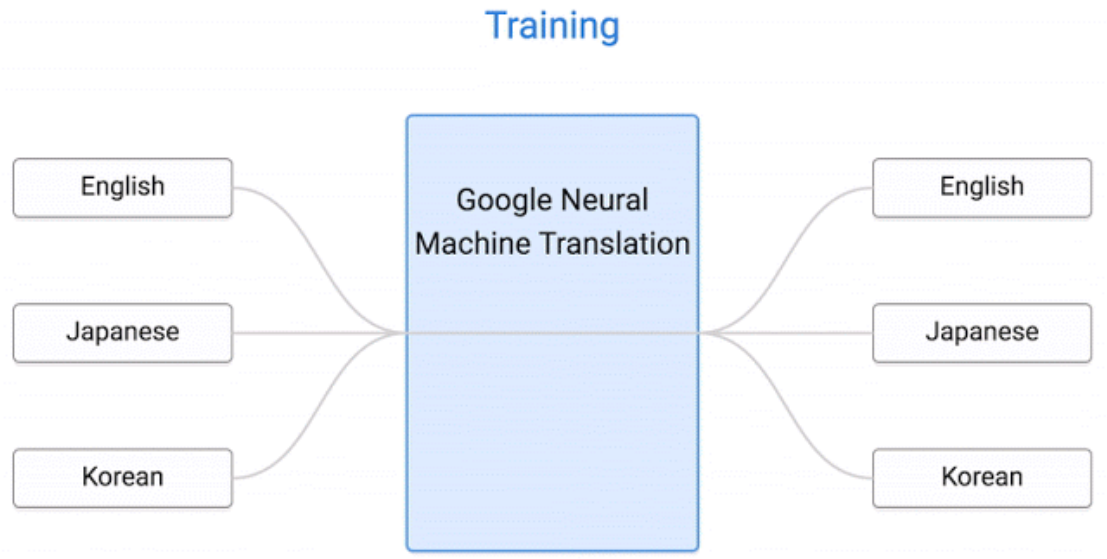
Tuesday, November 22, 2016

Posted by Mike Schuster (Google Brain Team), Melvin Johnson (Google Translate) and Nikhil Thorat (Google Brain Team)

In the last 10 years, Google Translate has grown from supporting just a few languages to 103, translating over 140 billion words every day. To make this possible, we needed to build and maintain many different systems in order to translate between any two languages, incurring significant computational cost. With neural networks reforming many fields, we were convinced we could raise the translation quality further, but doing so would mean rethinking the technology behind Google Translate.

In September, we announced that Google Translate is switching to a new system called Google Neural Machine Translation (GNMT), an end-to-end learning framework that learns from millions of examples, and provided significant improvements in translation quality. However, while switching to GNMT improved the quality for the languages we tested it on, scaling up to all the 103 supported languages presented a significant challenge.

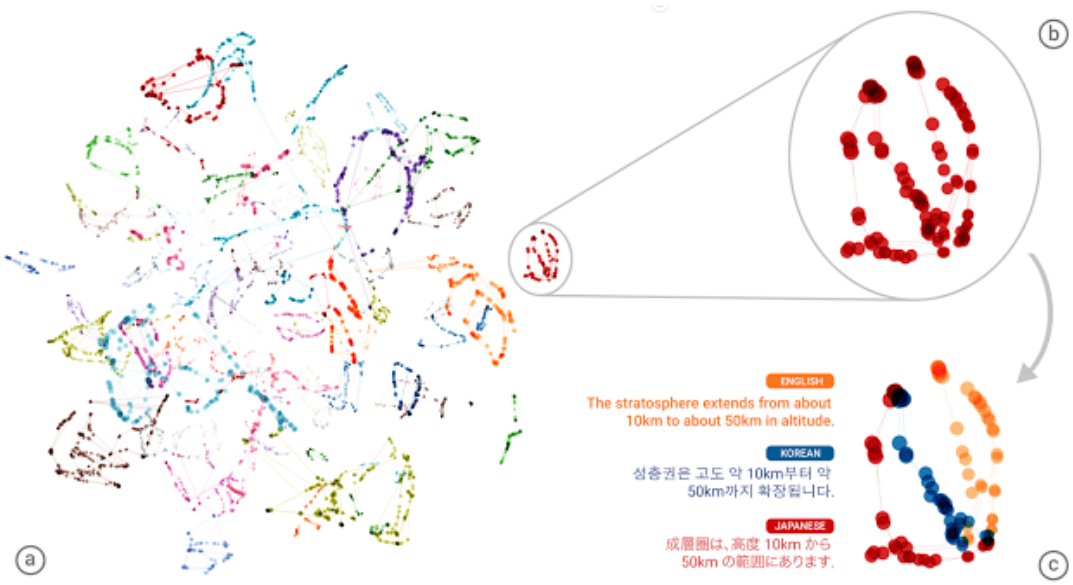
In "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation", we address this challenge by extending our previous GNMT system, allowing for a single system to translate between multiple languages. Our proposed architecture requires no change in the base GNMT system, but instead uses an additional "token" at the beginning of the input sentence to specify the required target language to translate to. In addition to improving translation quality, our method also enables "Zero-Shot Translation" — translation between language pairs never seen explicitly by the system.



Here's how it works. Let's say we train a multilingual system with Japanese⇆English and Korean⇆English examples, shown by the solid blue lines in the animation. Our multilingual system, with the same size as a single GNMT system, shares its parameters to translate between these four different language pairs. This sharing enables the system to transfer the "translation knowledge" from one language pair to the others. This transfer learning and the need to translate between multiple languages forces the system to better use its modeling power.

This inspired us to ask the following question: Can we translate between a language pair which the system has never seen before? An example of this would be translations between Korean and Japanese where Korean⇆Japanese examples were not shown to the system. Impressively, the answer is yes — it can generate reasonable Korean⇆Japanese translations, even though it has never been taught to do so. We call this "zero-shot" translation, shown by the yellow dotted lines in the animation. To the best of our knowledge, this is the first time this type of transfer learning has worked in Machine Translation.

The success of the zero-shot translation raises another important question: Is the system learning a common representation in which sentences with the same meaning are represented in similar ways regardless of language — i.e. an "interlingua"? Using a 3-dimensional representation of internal network data, we were able to take a peek into the system as it translates a set of sentences between all possible pairs of the Japanese, Korean, and English languages.



Part (a) from the figure above shows an overall geometry of these translations. The points in this view are colored by the meaning; a sentence translated from English to Korean with the same meaning as a sentence translated from Japanese to English share the same color. From this view we can see distinct groupings of points, each with their own color. Part (b) zooms in to one of the groups, and part (c) colors by the source language. Within a single group, we see a sentence with the same meaning but from three different languages. This means the network must be encoding something about the semantics of the sentence rather than simply memorizing phrase-to-phrase translations. We interpret this as a sign of existence of an interlingua in the network.

We show many more results and analyses in our paper, and hope that its findings are not only interesting for machine learning or machine translation researchers but also to linguists and others who are interested in how multiple languages can be processed by machines using a single system.

Finally, the described Multilingual Google Neural Machine Translation system is running in production today for all Google Translate users. Multilingual systems are currently used to serve 10 of the recently launched 16 language pairs, resulting in improved quality and a simplified production architecture.

236 条评

Labels: Google Brain , Google Translate , Machine Learning , Machine Translation , TensorFlow

Celebrating TensorFlow's First Year

Wednesday, November 09, 2016

Posted by Zak Stone, Product Manager for TensorFlow, on behalf of the TensorFlow team

(Cross-posted on the Google Open Source Blog & Google Developers Blog)

It has been an eventful year since the Google Brain Team open-sourced TensorFlow to accelerate machine learning research and make technology work better for everyone. There has been an amazing amount of activity around the project: more than 480 people have contributed directly to TensorFlow, including Googlers, external researchers, independent programmers, students, and senior developers at other large companies. TensorFlow is now the most popular machine learning project on GitHub.

With more than 10,000 commits in just twelve months, we've made numerous performance improvements, added support for distributed training, brought TensorFlow to iOS and Raspberry Pi, and integrated TensorFlow with widely-used big data infrastructure. We've also made TensorFlow accessible from Go, Rust and Haskell, released state-of-the-art image classification models, and answered thousands of questions on GitHub, StackOverflow and the TensorFlow mailing list along the way.

At Google, TensorFlow supports everything from large-scale product features to exploratory research. We recently launched major improvements to Google Translate using TensorFlow (and Tensor Processing Units, which are special hardware accelerators for TensorFlow). Project Magenta is working on new reinforcement learning-based models that can produce melodies, and a visiting PhD student recently worked with the Google Brain team to build a TensorFlow

model that can automatically interpolate between artistic styles. DeepMind has also decided to use TensorFlow to power all of their research – for example, they recently produced fascinating generative models of speech and music based on raw audio.

We're especially excited to see how people all over the world are using TensorFlow. For example:

- Australian marine biologists are using TensorFlow to find sea cows in tens of thousands of hi-res photos to better understand their populations, which are under threat of extinction.
- An enterprising Japanese cucumber farmer trained a model with TensorFlow to sort cucumbers by size, shape, and other characteristics.
- Radiologists have adapted TensorFlow to identify signs of Parkinson's disease in medical scans.
- Data scientists in the Bay Area have rigged up TensorFlow and the Raspberry Pi to keep track of the Caltrain.

We're committed to making sure TensorFlow scales all the way from research to production and from the tiniest Raspberry Pi all the way up to server farms filled with GPUs or TPUs. But TensorFlow is more than a single open-source project – we're doing our best to foster an open-source ecosystem of related software and machine learning models around it:

- The TensorFlow Serving project simplifies the process of serving TensorFlow models in production.
- TensorFlow "Wide and Deep" models combine the strengths of traditional linear models and modern deep neural networks.
- For those who are interested in working with TensorFlow in the cloud, Google Cloud Platform recently launched Cloud Machine Learning, which offers TensorFlow as a managed service.

Furthermore, TensorFlow's repository of models continues to grow with contributions from the community, with more than 3000 TensorFlow-related repositories listed on GitHub alone! To participate in the TensorFlow community, you can follow our new Twitter account (@tensorflow), find us on GitHub, ask and answer questions on StackOverflow, and join the community discussion list.

Thanks very much to all of you who have already adopted TensorFlow in your cutting-edge products, your ambitious research, your fast-growing startups, and

your school projects; special thanks to everyone who has contributed directly to the codebase. In collaboration with the global machine learning community, we look forward to making TensorFlow even better in the years to come!

28 条

Labels: Google Brain , Machine Learning , open source , TensorFlow

Supercharging Style Transfer

Wednesday, October 26, 2016

Posted by Vincent Dumoulin*, Jonathon Shlens and Manjunath Kudlur, Google Brain Team

Pastiche. A French word, it designates a work of art that imitates the style of another one (not to be confused with its more humorous Greek cousin, *parody*). Although it has been used for a long time in visual art, music and literature, pastiche has been getting mass attention lately with online forums dedicated to images that have been modified to be in the style of famous paintings. Using a technique known as *style transfer*, these images are generated by phone or web apps that allow a user to render their favorite picture in the style of a well known work of art.

Although users have already produced gorgeous pastiches using the current technology, we feel that it could be made even more engaging. Right now, each painting is its own island, so to speak: the user provides a content image, selects an artistic style and gets a pastiche back. But what if one could combine many different styles, exploring unique mixtures of well known artists to create an entirely unique pastiche?

Learning a representation for artistic style

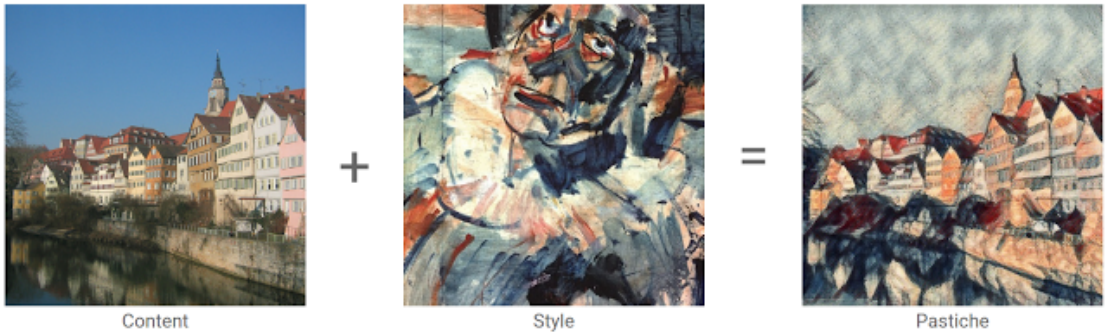
In our recent paper titled "*A Learned Representation for Artistic Style*", we introduce a simple method to allow a single deep convolutional style transfer network to learn multiple styles at the same time. The network, having learned multiple styles, is able to do *style interpolation*, where the pastiche varies smoothly from one style to another. Our method enables style interpolation in real-time as well, allowing this to be applied not only to static images, but also videos.

Credit: awesome dog role played by Google Brain team office dog Picabo.

In the video above, multiple styles are combined in real-time and the resulting style is applied *using a single style transfer network*. The user is provided with a set of 13 different painting styles and adjusts their relative strengths in the final style via sliders. In this demonstration, the user is an active participant in producing the pastiche.

A Quick History of Style Transfer

While transferring the style of one image to another has existed for nearly 15 years [1] [2], leveraging neural networks to accomplish it is both very recent and very fascinating. In "*A Neural Algorithm of Artistic Style*" [3], researchers Gatys, Ecker & Bethge introduced a method that uses deep convolutional neural network (CNN) classifiers. The pastiche image is found via optimization: the algorithm looks for an image which elicits the same kind of activations in the CNN's lower layers - which capture the overall rough aesthetic of the style input (broad brushstrokes, cubist patterns, etc.) - yet produces activations in the higher layers - which capture the things that make the subject recognizable - that are close to those produced by the content image. From some starting point (e.g. random noise, or the content image itself), the pastiche image is progressively refined until these requirements are met.



Content image: The Tübingen Neckarfront by Andreas Praefcke, Style painting: "Head of a Clown", by Georges Rouault.

The pastiches produced via this algorithm look *spectacular*:

Figure adapted from L. Gatys et al. "A Neural Algorithm of Artistic Style" (2015).

This work is considered a breakthrough in the field of deep learning research because it provided the first proof of concept for neural network-based style transfer. Unfortunately this method for stylizing an individual image is computationally demanding. For instance, in the first demos available on the web,

one would upload a photo to a server, and then still have plenty of time to go grab a cup of coffee before a result was available.

This process was sped up significantly by subsequent research [4, 5] that recognized that this optimization problem may be recast as an image transformation problem, where one wishes to apply a single, fixed painting style to an arbitrary content image (e.g. a photograph). The problem can then be solved by teaching a feed-forward, deep convolutional neural network to alter a corpus of content images to match the style of a painting. The goal of the trained network is two-fold: maintain the content of the original image while matching the visual style of the painting.

The end result of this was that what once took a few minutes for a single static image, could now be run real time (e.g. applying style transfer to a live video). However, the increase in speed that allowed real-time style transfer came with a cost - a given style transfer network is tied to the style of a *single* painting, losing some flexibility of the original algorithm, which was not tied to any one style. This means that to build a style transfer system capable of modeling 100 paintings, one has to train and store *100 separate style transfer networks*.

Our Contribution: Learning and Combining Multiple Styles

We started from the observation that many artists from the impressionist period employ similar brush stroke techniques and color palettes. Furthermore, painting by say, Monet, are even more visually similar.

Poppy Field (left) and Impression, Sunrise (right) by Claude Monet. Images from Wikipedia

We leveraged this observation in our training of a machine learning system. That is, we trained a single system that is able to capture and generalize across many Monet paintings or even a diverse array of artists across genres. The pastiches produced are qualitatively comparable to those produced in previous work, while

originating from the *same style transfer network*.

Pastiches produced by our single network, trained on 32 varied styles. These pastiches are qualitatively equivalent to those created by single-style networks: Image Credit: (from top to bottom) content photographs by Andreas Praefcke, Rich Niewiroski Jr. and J.-H. Janßen, (from left to right) style paintings by William Glackens, Paul Signac, Georges Rouault, Edvard Munch and Vincent van Gogh.

The technique we developed is simple to implement and is not memory intensive. Furthermore, our network, trained on several artistic styles, permits arbitrary combining multiple painting styles **in real-time**, as shown in the video above. Here are four styles being combined in different proportions on a photograph of Tübingen:

Unlike previous approaches to fast style transfer, we feel that this method of modeling multiple styles at the same time opens the door to exciting new ways for users to interact with style transfer algorithms, not only allowing the freedom to create new styles based on the mixture of several others, but to do it in real-time. Stay tuned for a future post on the Magenta blog, in which we will describe the algorithm in more detail and release the TensorFlow source code to run this model and demo yourself. We also recommend that you check out Nat & Lo's fantastic video explanation on the subject of style transfer.

References

[1] Efros, Alexei A., and William T. Freeman. *Image quilting for texture synthesis and transfer* (2001).

[2] Hertzmann, Aaron, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. *Image analogies* (2001).

[3] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. *A Neural Algorithm of Artistic Style* (2015).

[4] Ulyanov, Dmitry, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images (2016).

[5] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution (2016).

* This work was done during an internship with the Google Brain Team. Vincent is currently a Ph.D. candidate at MILA, Université de Montréal. ↩

Labels: Google Brain , Neural Networks , Style Transfer , TensorFlow

Image Compression with Neural Networks

Thursday, September 29, 2016

Posted by Nick Johnston and David Minnen, Software Engineers

Data compression is used nearly everywhere on the internet - the videos you watch online, the images you share, the music you listen to, even the blog you're reading right now. Compression techniques make sharing the content you want quick and efficient. Without data compression, the time and bandwidth costs for getting the information you need, when you need it, would be exorbitant!

In "*Full Resolution Image Compression with Recurrent Neural Networks*", we expand on our previous research on data compression using neural networks, exploring whether machine learning can provide better results for image compression like it has for image recognition and text summarization. Furthermore, we are releasing our compression model via TensorFlow so you can experiment with compressing your own images with our network.

We introduce an architecture that uses a new variant of the Gated Recurrent Unit (a type of RNN that allows units to save activations and process sequences) called Residual Gated Recurrent Unit (Residual GRU). Our Residual GRU combines existing GRUs with the residual connections introduced in "*Deep Residual Learning for Image Recognition*" to achieve significant image quality gains for a given compression rate. Instead of using a DCT to generate a new bit representation like many compression schemes in use today, we train two sets of neural networks - one to create the codes from the image (encoder) and another to create the image from the codes (decoder).

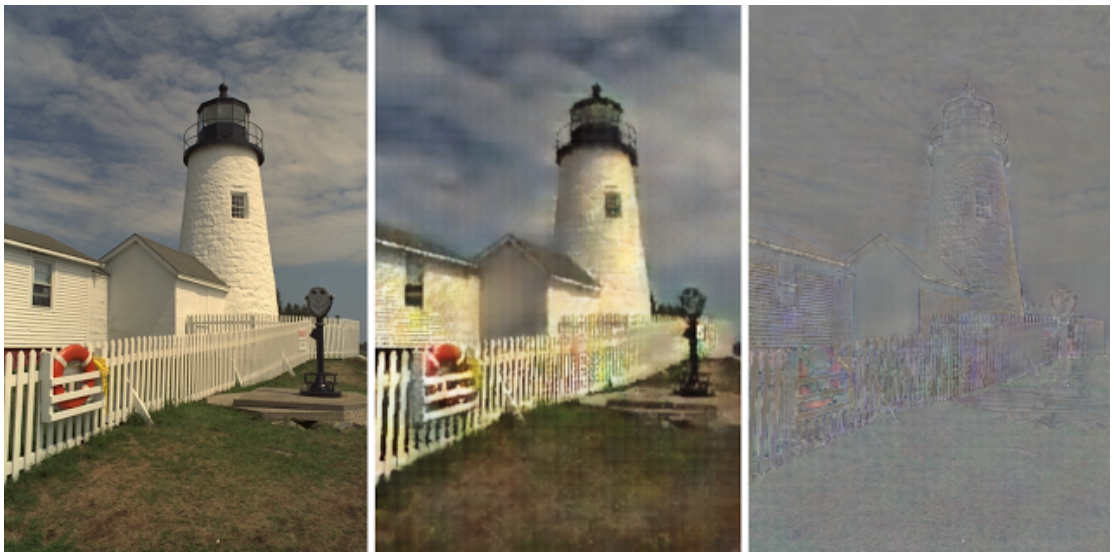
Our system works by iteratively refining a reconstruction of the original image, with both the encoder and decoder using Residual GRU layers so that additional

information can pass from one iteration to the next. Each iteration adds more bits to the encoding, which allows for a higher quality reconstruction. Conceptually, the network operates as follows:

1. The initial residual, R[0], corresponds to the original image I: R[0] = I.
2. Set i=1 for to the first iteration.
3. Iteration[i] takes R[i-1] as input and runs the encoder and binarizer to compress the image into B[i].
4. Iteration[i] runs the decoder on B[i] to generate a reconstructed image P[i].
5. The residual for Iteration[i] is calculated: R[i] = I - P[i].
6. Set i=i+1 and go to Step 3 (up to the desired number of iterations).

The residual image represents how different the current version of the compressed image is from the original. This image is then given as input to the network with the goal of removing the compression errors from the next version of the compressed image. The compressed image is now represented by the concatenation of B[1] through B[N]. For larger values of N, the decoder gets more information on how to reduce the errors and generate a higher quality reconstruction of the original image.

To understand how this works, consider the following example of the first two iterations of the image compression network, shown in the figures below. We start with an image of a lighthouse. On the first pass through the network, the original image is given as an input (R[0] = I). P[1] is the reconstructed image. The difference between the original image and encoded image is the residual, R[1], which represents the error in the compression.



Left: Original image, I = R[0]. **Center:** Reconstructed image, P[1]. **Right:** the residual, R[1], which represents the error introduced by compression.

On the second pass through the network, R[1] is given as the network's input (see figure below). A higher quality image P[2] is then created. So how does the system recreate such a good image (P[2], center panel below) from the residual R[1]? Because the model uses recurrent nodes with memory, the network saves information from each iteration that it can use in the next one. It learned something about the original image in Iteration[1] that is used along with R[1] to generate a better P[2] from B[2]. Lastly, a new residual, R[2] (right), is generated by subtracting P[2] from the original image. This time the residual is smaller since there are fewer differences between the reconstructed image, and what we started with.

The second pass through the network. **Left:** R[1] is given as input. **Center:** A higher quality reconstruction, P[2]. **Right:** A smaller residual R[2] is generated by subtracting P[2] from the original image.

At each further iteration, the network gains more information about the errors introduced by compression (which is captured by the residual image). If it can use that information to predict the residuals even a little bit, the result is a better reconstruction. Our models are able to make use of the extra bits up to a point. We see diminishing returns, and at some point the representational power of the network is exhausted.

To demonstrate file size and quality differences, we can take a photo of Vash, a Japanese Chin, and generate two compressed images, one JPEG and one Residual GRU. Both images target a perceptual similarity of 0.9 MS-SSIM, a perceptual quality metric that reaches 1.0 for identical images. The image generated by our

learned model results in an file 25% smaller than JPEG.

Left: Original image (1419 KB PNG) at ~1.0 MS-SSIM. **Center:** JPEG (33 KB) at ~0.9 MS-SSIM. **Right:** Residual GRU (24 KB) at ~0.9 MS-SSIM. This is 25% smaller for a comparable image quality

Taking a look around his nose and mouth, we see that our method doesn't have the magenta blocks and noise in the middle of the image as seen in JPEG. This is due to the blocking artifacts produced by JPEG, whereas our compression network works on the entire image at once. However, there's a tradeoff -- in our model the details of the whiskers and texture are lost, but the system shows great promise in reducing artifacts.

Left: Original. **Center:** JPEG. **Right:** Residual GRU.

While today's commonly used codecs perform well, our work shows that using neural networks to compress images results in a compression scheme with higher quality and smaller file sizes. To learn more about the details of our research and a comparison of other recurrent architectures, check out our paper. Our future work will focus on even better compression quality and faster models, so stay tuned!

60 条

Labels: Image Processing , Neural Networks , TensorFlow