CS Talks

Neural Image Captioning February 10, 2016

Lecturer: Samy Bengio Scribe: Kiran Vodrahalli

Contents

1 Overview

Samy Bengio from Google is here today; he's been there for 9 years.

I'm going to talk about neural image captioning, we call our team the "Google Brain" team since we're involved in deep learning. You have an image, and you'd like to be able to produce a caption about the image. We call our model NIC: Neural image capationing. How can we align sentences with images?

2 Deep convolutional networks for image classification

Let's start with the basics: Image classification. The goal here is to classiy images into a small number of classes, say 1000. Every year there is a challenge, and the goal is to get the best classification error on a separate test set. For many years, standard computer vision research uses various features, trying to find what features are good to represent an image. Examples: SIFT, HOG, and so on. Once these features are extracted, we turn our head to the machine learning experts which have very well-understood machine learning tools, i.e. SVM. Everything was good and happy until about 2012, when the deep learning wave started to hit everyone. The idea of deep learning is to use both feature extraction and prediction as one task where you have to do everything. Since 2012, no one else has been using the classical way. We have a deep convolutional net (convolutions, pooling, dense layers, etc.). Then you do gradient descent to update the weights. In 2012, there were about 7 or 8 layers. Since then, things have evolved into more layers 22. A team this year at MSR had 152 layers, and they were the winner of the 2016 competition. They become bigger and bigger and like monsters, but it works. Note that these layers actually share a lot of parameters - they just make better use of them. We don't have anything about how many samples you need. The MSR team had very specific structural changes (shortcuts/ highways) which works for a reason. There are a few influential papers from the 90s which show that long term gradients are difficult (vanishing gradient problem) - so it's hard to train the very deep model, unless you have tricks to keep the gradient flowing. (AN: The idea seems to be that if you can keep training, it's good - it's weird that they're using tricks to get things to just keep training, and to train quickly so that training error drops quickly for generalization - there's no explanantion for why this should be though).

2.1 Training

Each layer must be made of a differentiable function (AN: Actually, why do we have to use gradients at all.. maybe look at other ways of updating for nonconvex functions). Usually the last layer is the layer where you take the decision, and there's a loss you try to minimize, and you have a gradient descent approach, where you compute the gradient with respect to all the layers in the model where you use the chain rule and the gradient descent rule. The layers people were using over time have changed. For instance, sigmoid was used a while ago. After you got to 0 parts, the gradient would be zero and people wouldn't train. ReLU is popular today (nonzero part is a long tail, it goes to infinty). If you have enough of them, they will compensate for the ones that have no gradient. GPUs are also very important for training. At Google, we developed asynchronous techniques to train multiple models in parallel to exchange updates and so on in various ways, so they could train faster. How did we reconcile the models? There are various approaches: We have a parameter server, which could live in many machines. Each client or worker would not know about the other workers: It would get part of the data, and everytime it would want to process a batch, and ask for the current model, find the gradient, send gradient back to server, and so on. They would not know that the model would change in the meanwhile. Other works are updating the model - but it works well up to a certain scale, but we now think there are better ways of doing this asynchronously; it's an open problem.

It's not clear which tricks are the best ones, and it's an open problem to do it well, but it does work well.

3 Recurrent neural networks for language modeling

The second building block for doing image captioning task is what to do about words. If you represent them naively, then it's way too large. Also no useful properties - they're completely symmetric with each other. So you want to embed the words. You can put as many words as you want inside the hypercube, and ideally similar words will be nearby each other. There have been zillions of approaches. There are two kinds of approaches which are popular. One of them is an online pass over a text corpus (i.e. Word2Vec), another is to use co-occurrence statistics and move embeddings to estimate them by dot product (actually these are secretly the same thing, see Arora et al). There's a notion of head and tail when we work with documents. There's a few topics which are everywhere, and there's a lot of topics (the tail) which only a few of you are aware of. The sum of the tail-type elements is larger than that of the head (most of the well-known ones). This creates the so-called Zipf distribution. Most words are rare.

One thing often used at many places where you are interested in text is called language modeling. In general, you can factorize by conditioning on all the previous ones. In practice, we truncate the number of previous words we condition on: This is called n-grams. We have a sequence of words and we want to find a representation of them such that you maximize the probability of observing this sequence. You can use a recurrent neural network to do this to build a language model (see Mikolov 2010). We write

$$\prod_t \mathbf{P}\{y_t|h_t\}$$

with $h_t = f(y_{t-1}, h_{t-1})$.

One corpus we have at Google is the Google News benchmark. You feed your model starting from a start symbol, and you output the next h, and one symbol produces the next symbol. The figure represents the exact same thing as this equation. The words are represented as word embeddings in $\mathbb{R}^k$ where k does not depend on the dictionary size.

If you change the architecture of recurrent networks, you can train them: for instance, LSTM. It's very popular flavor of recurrent nets. How we measure whether language models are good or not is by measuring perplexity (AN: if perplexity is directly related to cross-entropy loss, then it's basically like we're picking a proxy for "a good model" and optimizing it directly - the goodness of the model is completely dependent on how you define goodness, which seems to be kind of circular). Apparently a small change in perplexity gives a big impact on what you can do with them. They've gotten perplexity down to 30. (AN: What's the intuition for why perplexity mdeans you can do more things..It seems very artifical to me).

3.1 Sequence-to-Sequence Framework

If you want to translate sentences, you can just have an input sentence and an output sentence matching the English sentence until you obtain end of sentence (you just have two recurrent networks, where one feeds

into the other after you feed in the whole sentence). They obtain state-of-the-art on dataset made available. Hopefully you see these models are taking over; they're all gradient based, you can put them together, etc. For training time, you put in the truth - the model is not robust to noise.

4 Image Captioning

Now we have all the building blocks. Instead of translating between languages, we translate from language space to image space. You pass your image through your favorite conv net, then you put this right into the convolutional net. You train model by taking image, passing it through the conv net, and then parsing it with th eproper image associated with it; you want to maximize the probability of the sentence, etc, and just do gradient descent. The MS-COCO dataset released in 2014 started a lot of things. There were about 80000 images, it was produced by Microsoft. Now it's around 300000 images, it's getting better. We had the best image model back then (GoogLeNet), and we had done well at modeling language, let's try to merge the two and get a joint model at the same time. The dictionary was a very restricted language ($10,000$ words). We trained them jointly, and looked at the metrics: BLEU-4, METEOR, CIDER: It's very hard to measure whether your model works or not: It's much more involved task to decide whether the task is right. People use multiple metrics, and all are based on agreement in n-grams. For BLEU, you count how many sequences of words they have in common; the higher the values the better. Humans can find many ways to describe a sentence, so BLEU may not agree - the numbers don't mean much. The fact that we can beat a human says how bad the evaluation is. What's the number in terms of beating human: We ask the human on a scale of 1 to 4; humans rank human sentences, you get 3.7. 2.9 is what we get. The language model is good because the English grammar is somewhat correct. **Never forget that it's not working as much as the numbers show**. The sentences we write are about 25 words or so; LSTMs are only good for up to about 15 time steps (in words).

Remark 4.1. There's a difference between training time and inference time. At inference time, we have to ask the model to pick a symbol to insert into the next round. You need a way to produce the next round. This is far from the truth! And now you're in a place which was very far from training, and you will diverge very quickly. What you could do is change the training procedure, such that it will explore where it's training. So once in a while, while training, you will either show the truth or something from the model itself (train the model by showing the model how it behaves) - you have to be very careful, because you're no longer training the maximum likelihood. But if you do it properly, you can explore more sequences of words, and become more robust to small mistakes. This is called **scheduled sampling**.

5 Conclusion

Image captioning is another application of sequence-to-sequence framework. It is important during training to expose the model to diverse situations it can encounter at inference trime. Sampling from the model provides diversity. Only sampling model during training is too hard. "Curriculum learning" is a reasonable approach to go from completely guided mode towards a mode similar to inference. But we get good performance on a few tasks. To win the competition we trained an ensemble (10 different models). At the end we ensembled their decisions.

6 Questions

Video captioning is much harder: You want to describe multiple parts of the video in together: There are two alignment procedures. There's two things happening. The resources required are huge, training + alignment makes it more involved.

Hyperparameter tuning takes a lot of time. Or you can try all of them if you have enough machines. It's a combination of being dumb and being smart at Google. It's really an open problem. There is no secret sauce.

To win the competition, we fine-tuned the image model (after being trained on image net) - so it's completely end-to-end. So the GoogleNet serves as an initialization for the part. We also tried to do the same thing on the word embeddings, but it neither helped nor harmed.

Can you localize images with attention - that's a method which lets this task generates words look at the part of the image, and it learns to do that.

The training set is now $4X$ bigger $300,000$ images with 5 captions per image. Previously there was a lot of overfitting with respect to generating same descriptions from the training data and we had to do a ton of regularization to beat this problem. However, now that the dataset is bigger, this is less of a problem (also keep in mind we're dealing with 10000 words).

We would like an unsupervised framework obviously. There's another big trend: Deep Dream: generating images from other images by applying nice-looking tricks. There's a recent paper which does caption $\rightarrow$ image, completely generated. Image from the database is so last year - it's a lot harder and it doesn't work well.

Nobody addresses biological framework. Visual question-answer assumes a lot of common knowledge which is in the question, and you need the information to answer the question. If there was enough training set, it's possible it coudl be present in the data: But it's too small now.