Last Updated November 23, 2015

Papers 2015 Reading Notes

Ilya Nepomnyashchiy

This is a collection of summaries I wrote up about some (but not all) papers I have read in 2015. If you have any questions, feel free to contact me using the contact info on my website.

Contents

RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response

This paper[2] sets out to create a scheme under which clients can report certain client strings (such as homepage settings, currently running processes, etc.) while preserving anonymity for each client. A very basic example of what such a scheme sets out to do is a survey in which one wishes to ask people something embarassing (e.g. "are you a communist?"). Respondents then flip a coin in private and respond truthfully if the coin comes up heads or always give the incriminating answer if it comes up tails. This provides some plausible deniability to people who truthfully give the incriminating answer, but still allows the survey to obtain reasonable aggregate measures.

This approach is a fine start, but has the problem of losing its advantage if the survey is repeated over time on the same or similar questions where the responses are unlikely to change. This paper develops a scheme that includes an amount of permanent randomness as well as an amount of on-the-spot randomness. As a side note, this scheme is currently used in the Google Chrome browser to collect certain statistics.

The response from the client is hashed into a Bloom Filter of a certain size. Then the client generates a randomized bit string of that same size that it remembers and uses for all future calculations. The Bloom Filter response is garbled based on that randomized string (each bit is either truthful with probability $1 - f$, where f is a parameter into the scheme, or 1 or 0 with probabilities $\frac{1}{2}f$ each). Finally, each time the response is requested, each bit in the response is set to 1 with tunable probability q if the garbled string from the previous step has that bit at 1, and tunable probability p if the previous string had that bit set to 0.

The paper uses a concept called Differential Privacy, as presented in [1], to show rigorously the amount to which each individual client's privacy is preserved. This concept looks at the ratio of probabilities that the scheme returns a certain value with and without one client's response. The proof is some relatively straightforward algebraic manipulation.

Finally, the paper describes a way to retrieve results from the responses, using Lasso Regression to select candidate strings and regular least-squares regression to estimate counts and p-values of the candidate strings. They demonstrate with some examples the effects of setting various parameters and their ability to retrieve client strings. Finally, they caution that RAPPOR is not a panacea and that other privacy-preserving measures must be taken as well.

ImageNet Classification with Deep Convolutional Neural Networks

This paper[6] was referred to me as one of the foundational papers of modern deep learning in computer vision. The ImageNet contest is one in which a very large set of images is classified into 1000 different classes, based on the object in the image. This paper describes a CNN that achieved the state of the art performance at the time and on which many modern CNN techniques are based.

A neural network can be described by a graph of layers of nodes ("neurons") which are used as input into the next layer of nodes, where the connections each have a weight. The value at a node is the weighted sum of its inputs passed into some non-linearity (such as $\tanh(x)$ or $(1+e^{-x})^{-1}$. The simplest type of neural network is one in which every node in layer i is connected to every node in layer $i+1$. A convolutional layer (which composes parts of a convolutional neural network) only connects nodes that are close to each other in the previous layer. For example, nodes in layer $i+1$ may be only connected to 5x5 windows in layer i where the windows move with a stride of 5 nodes/pixels. Furthermore, the weights for each location are the same (and these weights over a window is known as the "convolutional kernel"), and we learn several such kernels over the image. There are other details, which were discussed in this paper (and I will note below). Training a neural net is done through steps known as backpropagation and feedforward.

In practice, the limit on a size of a neural network is the amount of memory available for storing all of the many parameters (e.g. two densely connected layers with n nodes will have to store at least the n^2 weights between the layers, among some other parameters). Usually the networks are trained on GPUs, which can take advantage of the very parallel structure of the network and its training procedure. In 2012, when this paper was written, the top of the line GPUs available were the nVidia GTX 580s, which had 3GB of RAM per card.

Krizhevsky's network had a number of novel and unique features that improved the performance of his CNN. First of all, the nonlinearity at each node (as described above) was $f(x) = \max(0, x)$. This non-linerality decreased the amount of training time required using gradient descent in order to reach a particular error rate. They called these units Rectified Linear Units (ReLUs). Furthermore, the network was trained on two GTX 580 GPUs, where half of each layer was located on each GPU, and connections were only made between nodes on different GPUs in every other layer. This allowed for a larger network (which performed better) while reducing the slow communication required between GPUs, and thus speeding up training.

Since ReLUs do not saturate, input normalization is not required. However, the network used a scheme they call "local response normalization" to aid generalization. Basically, each convolutional layer learns multiple convolutions at a particular pixel (this is spatially denoted by the layer being three-dimensional), and the local normalization normalizes the different convolutions' responses at that pixel, based on some hyper-parameters into the model.

One typical technique in CNNs is to divide a layer into non-overlapping rectangles and outputting only the max on that rectangle to the next layer, known as "max-pooling". This network used max-pooling rectangles that did overlap, and the authors found that this improved training performance.

The overall architecture of this network was five convolutional layers followed by three dense layers, with the last layer being fed into a 1000-way softmax, producing a distribution of weights over the 1000 class labels in ImageNet. The network is trained to maximize the average of the log-probability of the correct label under the produced distribution. The local response normalization follows the first and second convolutional layers and max-pooling layers follow those layers as well as the fifth convolution layers. All non-linearities are the ReLU function. The original paper has a diagram of the network that is worth checking out. One thing worth noting is that the network trained 96 convolution kernels of size 11x11x3 at each pixel. I am told that these days, the kernels are typically smaller.

In order to reduce overfitting, the authors tried several techniques. First of all, they artificially enlargred the training set by using class-preserving transformations. These included shifting the image by several pixels, reflecting them horizontally, and adding multiples of principal components of the image (found using PCA) proportional to those components' eigenvalues. This allows for a much bigger network than would otherwise be possible without overfitting. Another method that was used is known as drop-out (to be described in more detail in another paper I will read soon). This method involves randomly "cancelling" connections in the network while training, so that different inputs can use very different paths through the network. This helps prevent specific adaptations of the network to the data set.

The grcadient descent was trained using a momentum term. All weights were initialized to a gaussian distribution with mean 0 and a small standard deviation. The overall training for this network took five to six days on the two GPUs. The overall network exhibited very good results.

Improving Neural Networks by Preventing Co-adaptation of Feature Detectors

This paper[3] discusses a method to reduce overfitting in feed-forward neural networks. One method that had been previously used to reduce overfitting was to train many separate neural networks and average their predictions together. Dropout, the method discussed in this paper, acts in effectively the same way while saving the training time and computational resources from having to train so many networks.

The way dropout works is that on every presentation of each training case, a hidden unit has a probability of .5 (or a tunable parameter) of being omitted from the network. That is, that neuron produces an activation signal of 0. At test time, the "mean network" is used, in which all outgoing weights on the hidden units are halved (as none are dropped out in testing). This allows the network to avoid co-adaptations of features that may appear together in the training set.

The network was trained with the standard stochastic gradient descent method, but with a modified penalty term. Instead of a penalty on the size of weights, there is an upper bound on the L2 norm of the incoming weight vector for each hidden unit. If a weight update causes the vector to go above this upper bound, it is renormalized below the upper bound.

This method was tested on several standard datasets and shown to achieve better results. It is also used in the AlexNet paper (the one immediately before this). There is also a discussion comparing Dropout to Bayesian model averaging, "bagging" and "naive bayes". The discussion finds that dropout is easier to implement than the first solution, is similar to the second (and allows a similar method to be used on Neural Networks over, say, Decision Trees), and the paper calls the third an extreme case of Dropout. The paper concludes with a discussion of the similiarity of Dropout to evolution.

Visualizing and Understanding Recurrent Networks

This paper[5] explores the performance of neural networks with Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN). These networks have been shown to have very impressive results on several data sets, including generating documents with structure (such as Wikipedia articles, LaTeX documents, and code).[4] In this paper, the authors find that some of the memory cells in the LSTM network are directly interpretable as a property of the document (such as the position within a line or being within a quote or bracket).

If we think of the state vectors of a regular neural networks for an input x as a set h^l, where l is the depth within the network, we can think of the vectors in an RNN as h^l_t, where the vectors depend on the input vectors x_t. In the end, the vector h_t^l depends on h_{t-1}^l and h_t^{l-1} through some recurrence that depends on the type of RNN.

In a vanilla RNN, this recurrence is of the form $h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$. The LSTM network and alternative GRU (Gated Recurrent Unit) are alternatives designed to make such networks easier to train. This is accomplished through a memory vector c_t^l and a gating mechanism based on a sigmoid function applied to the product of the weight matrix and the vector above.

For the experiments in this paper, character-level language modeling was used. Assuming a vocabulary of K charecters, input characters were encoded as 1-of$K$ vectors and fed into the network to obtain a vector that was interpreted as holding the log probability of the next character in the sequence. The two data sets used were *War and Peace* (containing 3,258,246 characters) and the *Linux Kernel* (containing 6,206,996 characters).

The experiment compared LSTM/GRU/RNN networks with $1, 2, 3$ layers and 4 different sizes of parameters. In general, the strongest result was that GRU and LSTM networks strongly outperformed the RNN, which was somewhat supported by a t-SNE embedding of the networks. The more interesting result of the experiment is visualizations that show interpretable uses of the memory cells by the LSTM. For example, one cell was used by the LSTM to count the line length in the *War and Peace* dataset. Other cells turned on "inside quotes, the parenthesis after if statements, inside strings or comments, or with increasing strength as the indentation of a block of code increases." Although the optimization method truncated backpropagation, preventing direct correlations of dependencies longer than 100 characters, the network still managed to keep track of states that persisted for longer than 100 characters.

A comparison to a regular neural network that takes as input the previous n characters (the "n-NN") and an $(n+1)$-gram language model ("n-gram") shows that the LSTM effectively utilizes information beyond 20 characters. A comparison is made between the errors of the 20-gram model and LSTMs. One specific example given is the closing curly brace in the Linux Kernel dataset, as it requires some of the most long-term reasoning. While the LSTM only slightly outperforms the 20-gram model in cases when the close brace is 0-20 characters away from the open brace, it significantly outperforms the 20-gram model in the medium size bins (the ones in the 20-100 range), with its performance delta decaying as the distance increases.

In the final experiment, oracles were applied to the network in order to see different classes of errors that the LSTM makes. The oracles remove errors given by the LSTM if a given condition is true (for example, if any of a 1-gram to 9-gram classifiers got the character right (the "n-gram" oracle), or if the error is on a word that appears fewer than 5 times in the dataset (the "rare words oracle")). This is a pretty interesting method of investigating errors, and one interesting result is that 18% of the errors were eliminated with the "n-gram" oracles.

References

[1] Dwork, C. Differential privacy. In *Encyclopedia of Cryptography and Security*. Springer, 2011, pp. 338–340.

[2] Erlingsson, Ú., Korolova, A., and Pihur, V. RAPPOR: randomized aggregatable privacy-preserving ordinal response. *CoRR abs/1407.6981* (2014).

[3] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[4] Karpathy, A. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015.

[5] Karpathy, A., Johnson, J., and Li, F. Visualizing and understanding recurrent networks. *CoRR abs/1506.02078* (2015).

[6] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.