Análise de Desempenho de *Frameworks* de *Deep Learning*[*]

Rafael G. Trindade[1], **João V. F. Lima**[1]

[1] Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

{rtrindade,jvlima}@inf.ufsm.br

Resumo. *Este trabalho avalia o desempenho de dois frameworks de Deep Learning (Caffe e TensorFlow) em um ambiente de execução heterogêneo. A avaliação se dá ao mensurar os tempos gastos por imagem, através da variação de hiperparâmetros de duas redes profundas conhecidas. O estudo conclui que o framework TensorFlow apresenta uma menor necessidade de memória, e tempos até 49% menores em GPU e até 70% menores em CPU em relação ao Caffe.*

1. Introdução

Deep Learning (DL) apresenta-se como um subconjunto de métodos de Aprendizado de Máquina que particularmente utiliza Redes Neurais Artificais (RNA) profundas em sua composição. Para que uma RNA aprenda a realizar uma tarefa, é necessário que ela seja treinada para tal. Para auxiliar a tarefa de modelagem e treinamento dessas redes, alguns *frameworks* foram criados com o intuito de aproveitar o poder computacional provido por arquiteturas modernas. O uso desses *frameworks* permitem que o projetista não precise se preocupar com otimização computacional em nível de programação, fornecendo um nível de abstração que o permite focar especificamente na modelagem da rede em si.

O uso de um acelerador como uma GPU mostra-se útil para a realização de computações massivas, e para treinamentos de redes profundas isso não é diferente. A aplicação de filtros convolucionais, comuns nesse tipo de rede, pode ser resumida computacionalmente a multiplicações de matrizes, e, como tal, pode ser realizada de forma eficiente em arquiteturas paralelas. Dadas as diferentes configurações dessas arquiteturas, é cabível mensurar quão rápido os *frameworks* conseguem deixar o treinamento dessas redes. Este trabalho tem como objetivo avaliar o desempenho, através do tempo consumido por imagem durante tarefas de treinamento, de dois *frameworks* de DL em um ambiente de execução que possui a disposição uma GPU de última geração. Ele está organizado da seguinte maneira: a Seção 2 exibe trabalhos relacionados ao assunto; a Seção 3 introduz os dois *frameworks* avaliados neste trabalho; a seção 4 aborda a metodologia utilizada nessa avaliação; a seção 5 exibe o resultado das execuções dos treinamentos, e finalmente, a seção 6 apresenta considerações finais obtidas através deste trabalho.

2. Trabalhos Relacionados

Apesar da crescente popularidade da área, poucos estudos foram conduzidos com o intuito de investigar o desempenho de *frameworks* de DL. Shams et al. [Shams et al. 2017] avaliam o desempenho de três *frameworks* de DL – Caffe, TensorFlow e Apache SINGA, em diferentes configurações de *hardware*, em ambientes com múltiplos nodos computacionais e diferentes CPUs e GPUs, além do uso da tecnologia NVLink e do processador

[*]Trabalho desenvolvido recebendo fomento do Edital Nº 015/2017 FIPE/UFSM.

Intel Xeon Phi. O trabalho fizera uso de uma versão mais antiga do TensorFlow (versão 0.12). Como resultado, Shams et al. [Shams et al. 2017] concluem que para as arquiteturas testadas a biblioteca Caffe apresenta menores tempos de computação e melhor escalabilidade que seus concorrentes.

3. Frameworks

Este trabalho aborda dois *frameworks* de DL bem difundidos pelas comunidades de pesquisa:

- **Caffe**: Desenvolvida pela *Berkeley Vision and Learning Center*, é uma biblioteca implementada em C++ que provê um *framework* simples e customizável para DL. Provê ligações para outras linguagens, como Python e MATLAB, simplificando o treinamento e inferência de redes neurais convolucionais de propósito geral e outros modelos profundos em arquiteturas comuns [Jia et al. 2014].
- **TensorFlow**: Desenvolvida pela Google, TensorFlow permite que computações definidas por grafos de fluxo possam ser executadas com poucas modificações em uma ampla variedade de sistemas heterogêneos. O sistema é flexível e pode ser usado para expressar uma grande variedade de algoritmos, como de treinamento e inferência para modelos de DL [Abadi et al. 2016]. Também é implementado em C++ e possui Python como ligação oficialmente suportada.

4. Metodologia

Para a execução dos treinamentos que passariam pela avaliação proposta pelo trabalho, foram escolhidos dois modelos de redes profundas, bem difundidas no campo de pesquisa de DL, para a etapa de avaliação dos *frameworks*: AlexNet e GoogLeNet. Ambas as redes possuem implementações de seus modelos criados pelas equipes de desenvolvimento de ambos os *frameworks*[1][2], permitindo que possam ser treinadas e avaliadas por quem deseja estudar suas arquiteturas. Para cada conjunto de configuração (rede, *framework*, modo e tamanho de lote) fora realizada um treinamento, tendo os valores utilizados para avaliação (segundos por imagem) sido calculados como média de cada treinamento. Os seguintes hiperparâmetros das redes tiveram valores especificamente adotados para os treinamentos:

- **Tamanho de lote**: Os treinamentos foram executados com variações no tamanho dos lotes de imagens: variando, em potências de 2, de 1 a 512;
- **Épocas**: Uma época corresponde a um intervalo de iterações em que todas as imagens do conjunto de entrada tenham passado uma vez pela rede. Este trabalho adotou o uso de 10 épocas de treinamento;
- **Unidade de Processamento (Modo)**: Hiperparâmetro responsável por determinar em qual unidade de processamento a rede será treinada (CPU ou GPU), e em qual memória os dados da rede serão armazenados (RAM ou dedicada/GPU).

5. Resultados Experimentais

Os treinamentos realizados em ambos os *frameworks* foram conduzidos no seguinte ambiente de execução, denominado **lsc5**:

[1]https://github.com/BVLC/caffe/tree/master/models

[2]https://github.com/tensorflow/models/tree/master/research/slim/nets

- CPU Intel Xeon E5620, com 8 núcleos operando a 2.4 GHz;
- GPU NVIDIA GeForce GTX Titan X, com 3072 núcleos CUDA e 12 GB de memória GDDR5 dedicada;
- 11 GB de memória RAM DDR3 e 14 GB de memória *swap*;
- Caffe versão 1.0.0 e TensorFlow versão 1.4.0.

A Figura 1 exibe os resultados das execuções dos treinamentos no ambiente supracitado. Com ele, se é possível visualizar a superioridade em poder computacional da GPU com relação a CPU. Algumas constatações podem ser realizadas:

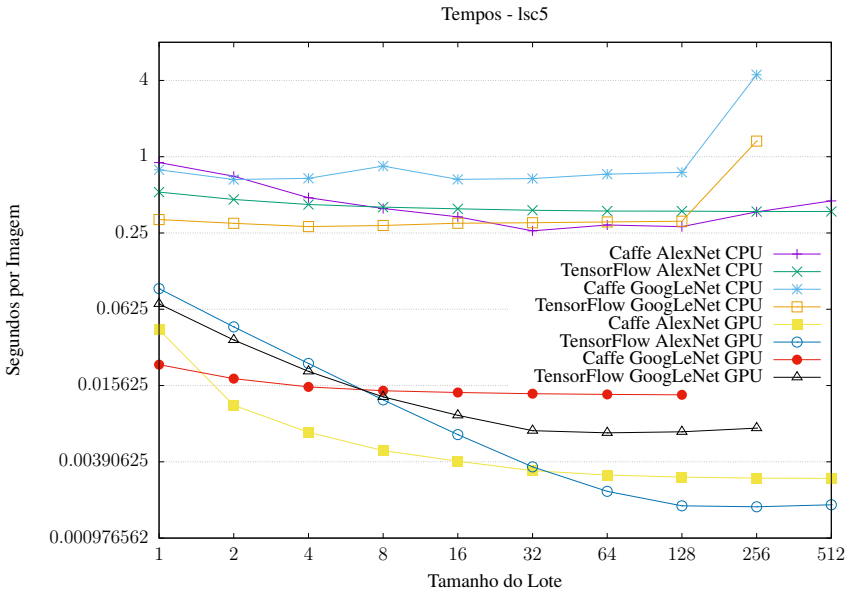

Figura 1. Estatísticas de tempo de execução no ambiente lsc5, com tempos por imagem para todas as combinações de redes, *frameworks* e unidades de processamento.

Para praticamente todos as combinações de rede e unidade de processamento o TensorFlow apresentou tempos menores em maior parte dos tamanhos de lote. Algumas exceções podem ser observadas, como os tempos relativamente semelhantes para a rede AlexNet rodando em CPU. Nas execuções da rede GoogLeNet em CPU, para ambas os *frameworks*, é possível visualizar um aumento no tempo por imagem com um lote de tamanho 256, pois, houve a necessidade dos *frameworks* recorrerem à memória *swap* para armazenar os dados internos da rede, quadruplicando o tempo médio por imagem.

Em suma, TensorFlow necessita de menos recursos de memória para armazenar os dados necessários para o treinamento. Esse fato fica parcialmente visível no gráfico, onde o *framework* consegue realizar o treinamento com um lote de tamanho 256, enquanto Caffe chegou ao limite com um lote de tamanho 128. Entretanto, isso nada se refere à quantia de alocação de memória realizada pelo *framework*: TensorFlow aloca por padrão o máximo de memória possível em GPU, mesmo sem necessitar de todo o espaço para o armazenamento dos dados intermediários das redes, e independentemente se a execução se dá em CPU ou GPU.

A Tabela 1, que exibe o número de chamadas e o tempo total gasto pelas instruções de multiplicações de matrizes em GPU, sugere que o tempo por imagem reduzido utili-

zado pelo TensorFlow se deve a um menor uso de multiplicações de matrizes (SGEMM), e à computação de algumas em lote (*Batched SGEMM*), habilidade fornecida pela biblioteca de algebra linear para GPUs NVIDIA, cuBLAS, e que confere a habilidade de realizar múltiplas multiplicações de matrizes pequenas de uma vez só, enquanto que o Caffe somente faz uso de instruções SGEMM simples.

Tabela 1. Tempos de execução e chamadas dos métodos SGEMM utilizados.

Método SGEMM	Caffe		TensorFlow			
	Comum		Comum		Batched	
	Tempo (s)	Chamadas	Tempo (s)	Chamadas	Tempo (s)	Chamadas
128x64_nn	6,90060	75264	0,016219	21	0,14928	26
128x64_nt	4,02506	71680			0,089606	21
128x64_tn	5,02266	71701				
128x128_tn	1,22427	10282	0,18607	60		
128x128_nn	1,00165	10812	0,16892	42	0,31289	45
128x128_nt	0,20359	60	0,16529	60	0,30616	42
64x64_nt					0,39549	64
64x64_nn					0,11230	21

6. Considerações Finais

Este trabalhou avaliou o desempenho de dois *frameworks* de DL em uma arquitetura híbrida dispondo de uma GPU moderna. Dois modelos de rede foram selecionados para execução dos treinamentos, e variações de hiperparâmetros foram realizados de treinamento para treinamento. Ao final das execuções, é possível avaliar uma clara vantagem em tempo computacional do *framework* TensorFlow, tanto em CPU quanto em GPU – atingindo tempos até duas vezes menores em GPU –, além de necessitar menos memória que o *framework* Caffe para o armazenamento dos dados da rede. Trabalhos futuros podem abordar diferentes arquiteturas e tamanhos maiores de lote, além de modelos de redes diferentes.

Agradecimentos

Este trabalho foi financiado pelo Fundo de Incentivo à Pesquisa – FIPE / UFSM. Agradecimentos especiais ao programa NVIDIA Hardware Grant Program que cedeu a GPU utilizada nos testes realizados nesse trabalho.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., and Corrado, G. S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093.

Shams, S., Platania, R., Lee, K., and Park, S. J. (2017). Evaluation of deep learning frameworks over different hpc architectures. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1389–1396.