Análise de Hiperparâmetros em Aplicações de Aprendizado Profundo por meio de Dados de Proveniência *

Débora B. Pina[1] **, Liliane Neves**[1]**, Aline Paes** [2]**, Daniel de Oliveira** [2]**, Marta Mattoso**[1]

[1]COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

[2]Instituto de Computação - Universidade Federal Fluminense (IC/UFF)

{dbpina, lneves, marta}@cos.ufrj.br, {alinepaes, danielcmo}@ic.uff.br

Resumo. *O treinamento das Redes Neurais Convolucionais (CNN) requer o ajuste de hiperparâmetros. As soluções existentes para auxiliar a escolha das melhores combinações de hiperparâmetros definem uma representação própria para modelar os relacionamentos de derivação dos dados. Essa representação proprietária dificulta a análise de dados e a interoperabilidade. Este artigo propõe a CNNProv, que adota o padrão W3C PROV para representar relacionamentos de derivação de dados para facilitar a análise das combinações de hiperparâmetros, contribuindo assim para a fase de treinamento das CNNs. A CNNProv captura dados de proveniência e permite a análise de valores de hiperparâmetros durante a execução. Os experimentos mostram a adequação do W3C PROV para a análise de hiperparâmetros e contribui para a qualidade e confiabilidade dos resultados da CNN, com overhead desprezível de até, no máximo, 4%.*

Abstract. *Convolutional Neural Networks (CNN) training requires adjusting hyperparameters. Current solutions to help choosing the best hyperparameter configuration define their own representation to model the data derivation relationships. This proprietary representation makes data analysis and interoperability difficult. This paper proposes CNNProv, which adopts the W3C PROV standard to represent data derivation relationships to facilitate the analysis of hyperparameter configurations, thus contributing to the CNNs training phase. CNNProv captures provenance data and allows hyperparameter analysis at runtime. The experiments show the suitability of the W3C PROV for hyperparameter analysis, while contributing to the quality and reliability of CNN results, with negligible overhead of up to 4%.*

1. Introdução

Na última década, a comunidade científica presenciou um aumento explosivo no número de aplicações de Aprendizado de Máquina (AM) que se baseiam em técnicas de Aprendizado Profundo (AP - *Deep Learning*) [Goodfellow et al. 2016]. Esse aumento se deve principalmente aos avanços em *hardware*, em especial às unidades de processamento gráfico (GPUs), que se mostraram adequadas para resolver de forma eficiente as operações matriciais necessárias ao AP. Uma das principais abordagens utilizadas em AP são as Redes Neurais Convolucionais (CNN) [Lecun et al. 1998]. Uma CNN é uma classe de Rede Neural do tipo *feed-forward* contendo uma ou mais unidades de convolução que utilizam filtros para produzir um mapa de atributos. Os neurônios em uma mesma camada de uma CNN podem ser agrupados em mapas e compartilham pesos, reduzindo a quantidade de parâmetros a serem treinados. Embora as CNNs representem um grande

*Agradecemos ao CNPq, CAPES e FAPERJ por financiarem parcialmente a pesquisa.

avanço na área de AM, para alcançar uma configuração de parâmetros que produza bons resultados, é necessário selecionar valores de certas variáveis, chamadas de hiperparâmetros [Goodfellow et al. 2016], o que pode demandar tempo e custo computacional. Exemplos de hiperparâmetros são a quantidade de iterações de treinamento (*épocas*) e a quantidade de exemplos apresentados em conjunto para o treinamento (*mini-batch*). O desempenho da CNN treinada depende diretamente dos valores de hiperparâmetros definidos.

De forma a definir os hiperparâmetros, o especialista em AM deve ser capaz de explorar diversas configurações para escolher uma combinação que resulte em um bom desempenho da CNN. Entretanto, esse processo de exploração requer que o treinamento da CNN seja realizado para cada uma das combinações, e, além disso, que as combinações de parâmetros utilizadas sejam registradas. Esse processo exploratório não é simples, uma vez que o espaço de busca da melhor solução é grande e a avaliação de cada configuração de hiperparâmetros pode ser computacionalmente intensiva. Abordagens existentes para AM, como o AutoML [He et al. 2018] e o GridSearch [Bergstra and Bengio 2012], já experimentam diferentes combinações de hiperparâmetros. Porém, tais métodos utilizam uma representação própria do histórico de derivação dos dados, ou seja, quais valores de hiperparâmetros levaram a quais resultados. Essas representações próprias dos dados de derivação podem dificultar a análise e, principalmente, a interoperabilidade. Assim, os resultados de experimentos em ferramentas diferentes vão requerer análises e implementações adicionais para serem comparados, já que não há uma maneira uniforme para determinar se dois modelos foram treinados com os mesmos dados de entrada.

Uma alternativa para esse problema é utilizar recomendações como o W3C PROV [Moreau and Groth 2013], que contém informações básicas para apoiar a interoperabilidade de dados de proveniência em diversos domínios. As consultas para avaliação de hiperparâmetros consumidos durante o treinamento de uma CNN se assemelham às típicas consultas de proveniência encontradas em diversos sistemas [Freire et al. 2008]. Ao definir o caminho de derivação, os dados de proveniência podem desempenhar um papel fundamental na análise e escolha dos hiperparâmetros para o treinamento. Existem diversas abordagens de captura de dados de proveniência [Stamatogiannakis et al. 2016]. Abordagens de captura automática de dados possuem granularidade muito fina, gerando um *overhead* significativo na execução de scripts, principalmente os de larga escala. A abordagem de captura de dados de proveniência por instrumentação permite a pré-seleção dos dados relevantes para análise, tendo menos impacto no tempo de execução. A DfAnalyzer [Silva et al. 2018] foi utilizada em diversos scripts de aplicações científicas com sucesso na análise de dados durante a execução do script, facilitando inclusive adaptações na configuração de parâmetros e por isso foi escolhida para esse artigo. Ademais, a captura de dados da DfAnalyzer é executada de forma assíncrona, ao longo do treinamento da CNN, e não interfere no desempenho do treinamento. Desta forma, ela é capaz de registrar informações complementares quanto ao desempenho do treinamento, além de associá-las a configurações de hiperparâmetros. A DfAnalyzer não é dependente de linguagens de programação e pode ser invocada por meio de uma biblioteca, da mesma forma que os especialistas já invocam outros componentes no código (p. ex., bibliotecas de visualização). Assim, a DfAnalyzer pode ser conectada às CNNs independente da ferramenta escolhida. Além disso, a DfAnalyzer adota o W3C PROV para representação de proveniência de dados, o que facilita a análise de dados e sua reprodutibilidade. A geração de dados de proveniência possui diversas vantagens [Freire et al. 2008], em particular a confiabilidade da CNN treinada.

Este artigo apresenta a abordagem CNNProv, que acopla a DfAnalyzer com CNNs para capturar e analisar valores de hiperparâmetros durante as fases de treinamento e execução

do modelo de forma automática Para avaliar a abordagem utilizamos a CNN AlexNet [Krizhevsky et al. 2012] como estudo de caso e o TensorFlow [Abadi et al. 2016] para o treinamento da CNN. Os experimentos realizados mostram a adequação da representação genérica da W3C PROV para a análise de hiperparâmetros, ao mesmo tempo em que contribui para a qualidade e confiabilidade dos resultados com *overhead* desprezível. Este artigo está organizado como a seguir. A Seção 2 discute os trabalhos relacionados a Seção 3 apresenta a abordagem CNNProv, a Seção 4 os experimentos, e a Seção 5 conclui.

2. Trabalhos Relacionados

A captura de dados de proveniência em experimentos de AM já foi proposta anteriormente na literatura. [Schelter et al. 2017] propõem uma abordagem para rastrear automaticamente os parâmetros de experimentos de AM, incluindo sua extração, armazenamento e gerência. De forma similar a DfAnalyzer [Silva et al. 2018], a abordagem proposta por [Schelter et al. 2017] captura os dados de forma automática com a premissa de que o usuário tenha definido quais dados carregar *a priori*. Além disso, a abordagem proposta atua de forma síncrona ao treinamento do modelo, o que pode introduzir um *overhead* não desprezível. Ademais, tal abordagem é uma representação própria da derivação dos dados, não seguindo recomendações padrão como o W3C PROV, o que complica a interoperabilidade.

DL-Steer [Souza et al. 2018] é uma biblioteca voltada para AP que coleta valores de entrada dos hiperparâmetros do modelo e relaciona-os aos resultados dos modelo treinados, permitindo que usuários alterem os valores dos hiperparâmetros durante o treinamento. Porém, a DL-Steer é limitada apenas para *scripts* Python com chamadas de funções para o TensorFlow, além de não considerar os dados de domínio da aplicação durante o processo de extração. Já o OpenML [Vanschoren et al. 2014] oferece serviços de compartilhamento de conjuntos de dados, implementações e resultados, proporcionando a reprodutibilidade de experimentos de AP. No entanto, o OpenML não representa o fluxo de transformações e não oferece apoio a consultas sobre os metadados.

3. Abordagem Proposta: CNNProv

A CNNProv visa a capturar e analisar dados de proveniência de CNNs. Esta é uma abordagem agnóstica à linguagem de programação ou bibliotecas como TensorFlow e Theano. A CNNProv pode ser descrita como uma extensão à DfAnalyzer com o objetivo de simplificar a adaptação do código da CNN para que o usuário defina quais hiperparâmetros deseja monitorar e consultar. Assim, a estratégia escolhida assume que os programas são caixas cinzas, ou seja, parte do seu código fonte é passível de ser adaptado, enquanto que a outra parte pode invocar um código fonte privado (caixa preta), para, p. ex., realizar invocações a outros programas ou bibliotecas. A abordagem é composta de cinco componentes, de acordo com a Figura 1: (i) Extrator de Proveniência (EP); (ii) Extrator de Dados Brutos (EDB); (iii) Visualizador do Dataflow (DVis); (iv) Interface de Consulta (IC); e (v) Banco de Dados estendido para modelos de CNN (MonetDB). Os dois primeiros componentes são invocados ao adaptarmos o código da CNN, enquanto os outros três têm interfaces independentes para o usuário consultar e analisar dados em tempo de execução.

A primeira etapa do uso da CNNProv é a etapa de adaptação do código que modela o treinamento. Na CNNProv, o código da CNN já vem previamente adaptado, porém os usuários podem considerar a inclusão de novos hiperparâmetros. A adaptação é dividida em duas subetapas. Na primeira, o usuário define as transformações de dados a serem rastreadas e quais hiperparâmetros serão considerados. Na segunda subetapa, o usuário indica no código onde obter os valores dos hiperparâmetros, para serem capturados em tempo de execução.

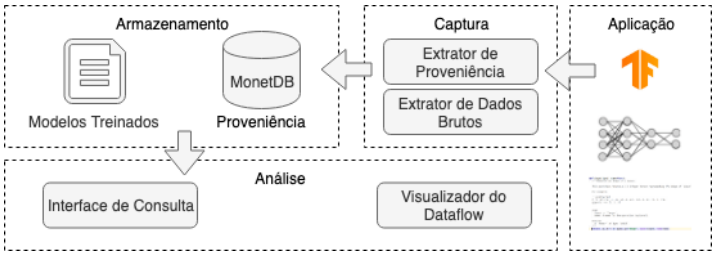


Figura 1. Arquitetura da CNNProv

Uma vez adaptado o código da CNN e seu treinamento tenha sido iniciado, o EP e o EDB são invocados assincronamente. Para cada chamada da CNNProv no código, dados das transformações, dos hiperparâmetros e dos modelos treinados são armazenados no banco de dados de proveniência usando o sistema colunar MonetDB e seguindo a representação Prov-Df [Silva et al. 2018]. Esses dados incluem o histórico de derivação dos dados, erros que tenham ocorrido, e a associação entre os valores de hiperparâmetros e os arquivos de modelo treinados. Finalmente, a interface de consulta e o visualizador do *dataflow* permitem ao usuário analisar os dados de proveniência coletados seja por meio da submissão de uma consulta ao CNNProv ou por meio de visualização.

4. Avaliação Experimental

Utilizamos a CNN AlexNet [Krizhevsky et al. 2012] como estudo de caso da CNNProv. A AlexNet é uma CNN que tem como objetivo o reconhecimento de imagens. Ela é composta de cinco camadas de convolução e três camadas completamente conectadas (Figura 3(a)). A redução na quantidade de parâmetros e no tempo de treinamento é alcançada ao conectar os filtros da 2^a, 4^a e 5^a camadas convolucionais apenas aos mapas de filtro das camadas anteriores que estão na mesma GPU – os filtros da terceira camada são conectados a todos os mapas de filtros da 2^a camada. Para reduzir o problema da dissipação do gradiente, após cada camada convolucional aplica-se a função de ativação *Relu*. Para reduzir *overfitting*, antes da 1^a e 2^a camadas completamente conectadas é incluída uma operação de *dropout* [Krizhevsky et al. 2012].

Durante a fase de adaptação do código da AlexNet para capturar dados de proveniência, foram definidas três transformações: *training*, *adaptation* e *testing*. A transformação *training* consome o nome do *dataset* de imagens de entrada (OxfordFlower17) e a proporção utilizada de dados para treino e teste, e produz um conjunto de dados que define os hiperparâmetros gerados durante a etapa de treinamento, p. ex., o valor da época, da acurácia, da função de custo (*loss function*), o tempo decorrido e a data e hora do fim da execução da época. A adaptação que é realizada pela AlexNet gera uma nova taxa de aprendizado (α') ao fim da época utilizando a função *Scheduler Step Decay* que diminui significativamente o valor da taxa de aprendizado (α) a cada n épocas em um fator m. Dessa forma, a transformação *adaptation* recebe como dados de entrada o conjunto produzido pela transformação anterior: *training* e um conjunto de dados com informações como o fator m, o valor de n e a taxa de aprendizado inicial. O conjunto de dados produzido por essa transformação contém a nova taxa de aprendizado, o valor da época e a data e hora em que a adaptação ocorreu, além de uma identificação para a adaptação, conforme apresentado na Figura 2, onde $t2$ representa a tarefa de uma transformação que terá seus dados extraídos e armazenados e *oAdaptation* representa o conjunto de dados de saída da transformação *adaptation*. Por último, a transformação *testing* provê uma avaliação do modelo de acordo com o conjunto de dados de treinamento e que tem como saída os valores de acurácia e da função de custo da CNN treinada.

Tabela 1. Tempo de treinamento

	Taxa de aprendizado		
# épocas	0,0005	0,001	0,002
20	1.367,51	1.345,42	1.365,18
50	3.387,08	3.354,28	3.359,43
100	6.745,14	6.769,15	6.757,80

Tabela 2. *Overhead* da CNNProv

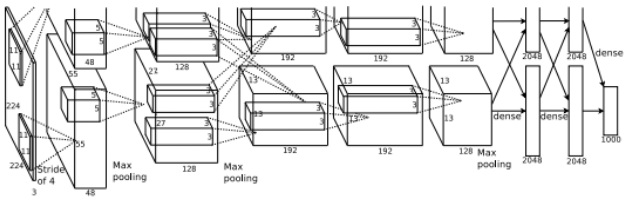
	Taxa de aprendizado		
# épocas	0,0005	0,001	0,002
20	3,57%	1,03%	3,23%
50	2,96%	2,20%	2,69%
100	3,16%	3,74%	3,33%

```
def on_epoch_begin(self, epoch, logs=None):
    old_lr = float(K.get_value(self.model.optimizer.lr))
    new_lr = self.schedule(epoch, lr)
    if (old_lr != new_lr):
        self.adaptation_id += 1
        K.set_value(self.model.optimizer.lr, new_lr)
        t2_output = DataSet("oAdaptation", [Element([new_lr,
        datetime.now().strftime('%Y-%m-%d %H:%M:%S'), epoch,
        str(self.adaptation_id)])])
        t2.add_dataset(t2_output)
        t2.save()
```

Figura 2. Trecho Adaptado do código do Tensorflow que implementa a AlexNet.

Com o código adaptado, a AlexNet foi treinada variando-se o número de épocas e a taxa de aprendizado. Cada combinação de hiperparâmetros foi executada cinco vezes e a média de tempo de execução foi considerada. A Tabela 1 apresenta o tempo de treinamento (em segundos) para cada combinação de parâmetros, enquanto que a Tabela 2 apresenta o *overhead* de tempo introduzido pela abordagem CNNProv para captura de proveniência. O objetivo desta medição foi avaliar o impacto das chamadas à CNNProv durante o treinamento da AlexNet com dados reais. Podemos observar que o *overhead* da solução proposta corresponde a um aumento de menos de 4% no pior caso sobre o tempo total de treinamento da CNN. Esse *overhead* pode ser ser considerado desprezível, principalmente em treinamentos mais demorados, considerando que o usuário terá o benefício das consultas e visualizações aos dados de proveniência capturados.

Foram levantadas e submetidas à CNNProv uma série de consultas desejadas para análise de dados de treinamento, como por exemplo: "Qual a perda, o tempo decorrido de treinamento para cada época?". A Figura 3(b) apresenta o resultado do processamento dessa consulta. A relação descrita na Figura 3(b) apresenta o tempo decorrido (*elapsed time*) em segundos, o número da época (*epoch*) e o valor da perda (*loss*). Mesmo nesse exemplo simples, podemos observar que com as adaptações no código da AlexNet foi plenamente viável consultar quanto tempo levou o treinamento em cada época e qual o valor de *loss function* associado para o treinamento.



elapsed_time	epoch	loss
106.55	20	0.50
110.92	19	0.53
113.86	18	0.66
100.06	17	0.71
109.21	15	0.73

Figura 3. (a) Arquitetura da AlexNet (b) Exemplo de consulta que mostra o tempo de corrido em cada época por ordem descrescente de perda

5. Conclusões

Este artigo apoia a fase de treinamento das CNNs ao registrar informações relevantes à análise de combinações de hiperparâmetros e posteriores reconfigurações. Foi apresentada a CNNProv, que adota uma arquitetura distribuída, com *overhead* desprezível e análise de dados via grafos de proveniência. Experimentos evidenciam a adequação do uso de proveniência nas atividades de análise e monitoramento, contribuindo para um padrão de consultas que pode consultar de forma integrada os dados de proveniência eventualmente associados aos dados usados no treinamento.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the ECCV*, pages 784–800.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Adv. in neural inf. proc. sys.*, pages 1097–1105.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Moreau, L. and Groth, P. T. (2013). *Provenance: An Introduction to PROV*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

Schelter, S., Böse, J.-H., Kirschnick, J., Klein, T., and Seufert, S. (2017). Automatically tracking metadata and provenance of machine learning experiments. In *MLS workshop @ NIPS*.

Silva, V., de Oliveira, D., Mattoso, M., and Valduriez, P. (2018). Dfanalyzer: Runtime dataflow analysis of scientific applications using provenance. *PVLDB*, 11(12):2082–2085.

Souza, R., Neves, L., Azeredo, L., Luiz, R., Tady, E., Cavalin, P. R., and Mattoso, M. (2018). Towards a human-in-the-loop library for tracking hyperparameter tuning in deep learning development. In *LADaS@VLDB*.

Stamatogiannakis, M., Kazmi, H., Sharif, H., Vermeulen, R., Gehani, A., Bos, H., and Groth, P. (2016). Trade-offs in automatic provenance capture. In *IPAW*, pages 29–41. Springer.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). Openml: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60.