Desenvolvimento de algoritmo CMA-ES para otimização de hiperparâmetros de pipelines

Henrique Nicolas e Pablo Goulart Silva

***Abstract*— Neste trabalho será apresentado a aplicação do algoritmo CMA-ES para otimização de pipelines de aprendizagem de máquina. O CMA-ES otimiza funções não-lineares de difícil solução, como funções não-convexas e as do tipo caixa-preta, e tem se mostrado uma solução eficaz em problemas na área de AutoML. Os resultados preliminares obtidos nesse trabalham demonstram, sob os aspectos de desempenho e qualidade da solução, que é possível adaptar o CMA-ES para que ele coexista com parâmetros categóricos e maximize o *f-score* de pipelines de AutoML para vários *datasets* de *benchmarking* da literatura.**

I. Introdução

O problema clássico de aprendizagem supervisionada é construir um classificador que, baseado em dados adquiridos de experiências passadas, prediga corretamente as classes associadas à conjuntos de dados inéditos [1]. Algoritmos de aprendizagem supervisionados se tornaram, portanto, ferramentas habituais em laboratórios de pesquisa de dados pois permitem compreender comportamentos, padrões de utilização, sazonalidade de eventos, estatísticas de dados amostrais, dentre outras possibilidades, presentes em bases de dados de maneira escalável, de modo que, na última década, se tornou largamente utilizada em diversos novos mercados. A demanda por ferramentas de aprendizagem, entretanto, exigem que os operadores conheçam profundamente os algoritmos utilizados por essas ferramentas para realizar ajustes em seus hiperparâmetros corretamente orientados ao problema do domínio da aplicação. Contudo, é improvável que os operadores tenham, a princípio, profundo conhecimento nesse ramo, o que tornaria o *time-to-market* excessivo e inviabilizaria a adoção desse tipo de tecnologia por não especialistas.

Ferramentas de aprendizagem têm sido propostas para ajustar automaticamente os melhores hiperparâmetros para atender os diversos cenários em que as técnicas de aprendizagem automática podem ser aplicadas. O ajuste dos pipelines de dados, entretanto, exige algumas contrapartidas - como o poder computacional - como requisito para que os times de diversos mercados utilizem largamente ferramentas de aprendizagem de máquina. A essa disciplina, chamada *Automated Machine Learning* (AutoML), atribui-se a missão de desenvolver algoritmos que ajustam a melhor combinação de hiperparâmetros de configuração do pipeline que executará os *workloads* de aprendizagem de máquina [2].

Neste trabalho, é apresentado o algoritmo CMA-ES - *Covariance Matrix Adaptation Evolution Strategy* - para ajustar hiperparâmetros de pipelines de dados. O CMA-ES tem sido citado [3] [4] [5] como o estado-da-arte para ajustar configurações de pipelines sob condições de não-linearidade da função objetivo e parâmetros reais.

II. CMA-ES

O algoritmo de evolucionário para adaptação da matriz de covariância (CMA-ES) é um algoritmo baseado em Estratégia Evolutiva (ES) utilizado para otimização de funções não-lineares de difícil solução, como funções não-convexas e as do tipo caixa-preta, isso é, funções da qual não há conhecimento prévio sobre suas derivadas - caso em que métodos analíticos de gradiente garantem a convergência. Portanto, o ES tem o objetivo de gerar pontos candidatos à solução cuja avaliação desses pontos pela função caixa-preta seja a menor possível, sendo, portanto, um mapeamento de um conjunto real contendo *D* dimensões, para *D* igual à quantidade de hiperparâmetros reais de uma função que mapeia um único valor $f(x)$ da imagem para cada vetor **x** do domínio do problema [1].

$$\begin{aligned} f : \Re^D &\rightarrow \Re \\ \mathbf{x} &\mapsto f(x) \end{aligned} \tag{1}$$

A. Definição

O CMA-ES é um algoritmo estocástico que amplia o conceito de *mutative strategy parameter control* (MSC), uma técnica que submete parâmetros, chamados *strategy parameters*, à etapa de mutação, e gera novos pontos no espaço de busca com a mutação dos *object parameters* a partir dos *strategy parameters*. O nível de indireção da etapa de mutação assume que parâmetros de estratégia que produziram indivíduos da população que foram selecionados na etapa de elitização são parâmetros aceitáveis em um futuro próximo [4]. O CMA-ES foi estendido com uma especialização do MSC cunhada de *derandomization*, uma etapa que ajuste dos *strategy parameter* que, de acordo com o conceito *completely derandomized auto-adaptative scheme*, ajusta diretamente a matriz de covariância pertencente à distribuição normal - e portanto um *strategy parameter* - utilizada na etapa de mutação do algoritmo para maximizar a probabilidade de produzir pontos de busca que serão selecionados pelo ranking de melhores indivíduos na etapa de seleção [3].

Para conferir um efeito de memória na preservação das chances de seleção de *strategy parameters* que produziram bons candidatos, o CMA-ES utiliza o *Cumulative Evolution Path* para maximizar a taxa de progresso em direção à uma solução ótima (ou aproximada), em oposição à preferência por *strategy parameters* utilizados em uma única etapa de

mutação. A tabela **??** mostra os parâmetros do CMA-ES com suas respectivas descrições.

Símbolo	Descrição
$C^{(g)}$	Matriz de covariância da geração *g*
λ	Tamanho da população
μ	Número de pontos do espaço de busca selecionados da população
c_c	Taxa de aprendizagem para acumulação do *rank-one update*
c_1	Taxa de aprendizagem do *rank-one update*
c_μ	Taxa de aprendizagem para o *rank-*μ *update*
c_σ	Taxa de aprendizagem da acumulação de σ
d_σ	Parâmetro de amortecimento para atualização do σ
μ_{eff}	Termo *variance effective selection mass*
$m^{(g)}$	Valor médio da distribuição do espaço de busca na geração *g*
w	Pesos para recombinação dos indivíduos da população
$x_k^{(g)}$	Indivíduo *k* da geração *g*
$x_{i:\lambda}^{(g)}$	Indivíduos $x_i^{(g)}, ..., x_\lambda^{(g)}$ da geração *g*

Tabela I: Tabela com parâmetros apresentados nas equações de adaptação do CMA-ES

B. Amostragem

A etapa de geração de pontos gera λ novos indivíduos para a população pela amostragem de uma distribuição normal multivariada, de covariância C e média m dada pelo valor médio dos candidatos da geração atual, e utiliza um parâmetro único, chamado de tamanho de passo global σ, como o desvio padrão da distribuição. Os hiperparâmetros C, m e σ são adaptados iterativamente para gerar a próxima população de candidatos à solução.

C. Adaptação

O processo adaptativo do CMA-ES pode ser decomposto em duas etapas: adaptação dos *strategy parameters* e dos *object parameters*.

Os *strategy parameters*, são responsáveis por controlar a velocidade de adaptação - etapa de mutação - do algoritmo. Em contraste, os *object parameters* são parâmetros utilizados para definir os pontos no espaço de busca do algoritmo [6].

Os *object parameters* são utilizados na adaptação dos valores da população, dos descendentes gerados pelo método, do tamanho do passo de mutação dos vetores candidatos (população) e da matriz utilizada como base ortonormal do algoritmo. Dentre os *strategy parameters* está, por exemplo, a matriz de covariância *C* a ser adaptada pelo algoritmo evolucionário, que contém as correlações (e os coeficientes de variação) da distribuição de probabilidade utilizada para mutação dos vetores que representam a população da estratégia evolutiva. O mecanismo *derandomized* de adaptação dos *strategy parameters* é assim definido devido ao fato de seus hiperparâmetros não serem diretamente adaptados na etapa de mutação. Isso se dá pela redução do conjunto de hiperparâmetros sujeitos a modificação estocástica, o que na prática, significa utilizar um conjunto de parâmetros constantes com poucos atributos adaptáveis e sobre condições que impeçam adaptações que possam prejudicar a convergência do algoritmo.

D. Seleção e Recombinação

A nova média *m* é o resultado da média ponderada dos μ pontos melhor avaliados, que são selecionados a partir da população de tamanho λ. Portanto:

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)} \aleph(\mathbf{0}, \mathbf{C}^{(g)}) \tag{2}$$

O algoritmo apresentado nesse trabalho replica a estratégia adotada por [4], que utiliza pesos diferentes para os μ pontos selecionados, normalizados, dada pela função:

$$w_i^{'} = \ln \frac{\lambda + 1}{2} - \ln i \quad \forall i \in [1, \lambda] \tag{3}$$

A partir da medida dos pesos, a medida chamada seleção da massa de variância efetiva, ou μ_{eff}, é calculada com o objetivo de servir como parâmetro de cálculo de hiperparâmetros do CMA-ES:

$$\mu_{eff} = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \tag{4}$$

Portanto, a técnica CMA-ES constrói matrizes de correlação C adaptadas de maneira a preservar a informação de iterações anteriores, com a ressalva de lhes atribuir peso controlado pelo parâmetro c_μ, ou taxa de aprendizagem da matriz de covariância. c_μ é um valor, no intervalo – intervalo – , que balanceia a contribuição de C(old) e C(new).

$$c_\mu = \min \left(1 - c_1, \alpha_{cov} \frac{\mu_{eff} - 2 + 1/\mu_{eff}}{(n+2)^2 + \alpha_{cov}\mu_{eff}/2} \right) \tag{5}$$

A adaptação de *C*, conforme equação 6, é uma estratégia obtida pela combinação das técnicas *rank-*μ*-update* e *rank-one update*. A primeira técnica busca acelerar a convergência do método em troca de sacrificar a qualidade da estimativa de C. Portanto, a utilização de informação passada tem o objetivo de corrigir o estimador de C por um fator de memória. Se o valor de c_μ for baixo, o algoritmo apresentará aprendizado lento, enquanto um alto valor ocasionará a falha do método, devido à degeneração da matriz C. A segunda técnica explora o conceito de caminho de evolução, que é calculado como a soma das diferenças entre as médias obtidas em iterações consecutivas ponderada pelo tamanho do passo sigma da iteração. Dessa forma, a fórmula de adaptação global da matriz *C* pode ser definida como:

$$C^{(g+1)} = (1 - c_1 - c_\mu \sum w_j) C^{(g)} + c_1 p_c^{(g+1)} p_c^{(g+1)^T} + c_\mu \sum_{i=1}^{\lambda} w_i y_{i:\lambda}^{(g+1)} (y_{i:\lambda}^{(g+1)})^T \tag{6}$$

Por último, o controle o tamanho do passo σ segue a estratégia de caminho evolutivo utilizada na adaptação de C, onde a média m de iterações consecutivas é utilizada em conjunto com outros parâmetros de normalização. Em primeiro lugar, o tamanho esperado do caminho evolutivo p_σ é calculado, e este valor é utilizado na atualização do σ através das fórmulas:

$$\mathbf{p}_\sigma \leftarrow (1-c_\sigma)\mathbf{p}_\sigma + \sqrt{c_\sigma(2-c_\sigma)\mu_{eff}}\mathbf{C}^{-\frac{1}{2}}\langle \mathbf{y}\rangle_w \quad (7)$$

$$\sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\|\aleph(O,\mathbf{I})\|}\right)\right) \quad (8)$$

Uma premissa da elaboração da proposta CMA-ES original, apresentada por [4], se baseia na contribuição de apenas alguns poucos parâmetros na obtenção de melhores candidatos à solução. Entretanto, o tempo de adaptação é considerado o fato limitante do desempenho do algoritmo para dimensões do problema que excedem um limiar $D \geq D_{limiar}$, usualmente de tamanho 10 [7].

III. Implementação

O *AutoML Framework* foi construído para ser um arcabouço para a experimentação de algoritmos de aprendizagem em funções objetivo caixa-preta. Ele visa otimizar a configuração dos hiperparâmetros de uma coleção de algoritmos de aprendizagem de máquina, além de escolher, dentre as técnicas e parâmetros possíveis, a configuração mais pertinente ao problema sob avaliação com grau de precisão personalizado. O *AutoML Framework* abarca as etapas de definição de gramática que descreve quais as técnicas possível de serem aplicadas no pipeline de processamento, avaliação de pipelines em um servidor que executa uma possível combinação da gramática de pipelines, e um cliente que permite construir de algoritmos, acoplá-los aos métodos existentes no framework - como outros algoritmos de aprendizagem ou técnicas de pré-processamento - e submetê-los para avaliação. O objetivo do *AutoML Framework* é unificar, em uma única plataforma, a exploração de técnicas diferentes de otimização de hiperparâmetros sem pôr em risco a credibilidade da avaliação pela inserção de vieses que naturalmente poderiam ser inseridos em experimentações executadas de maneira não uniformizada.

A. CMA-ES no AutoML Framework

De acordo com a definição do apresentada na documentação do framework [8], a classe BaseOptimizer foi extendida para conter a implementação do método CMA-ES. A classe BaseOptimizer exige a sobrecarga do método *optimize*, responsável por, efetivamente, avaliar os pipelines de dados. Para comportar as etapas presentes no algoritmo CMA-ES, algumas classes foram desenvolvida e inseridas no diretório *library*, do código desenvolvido neste trabalho. As funcionalidades de avaliação (*Evaluation*), evolução (*Evolution*), geração de novos indivíduos (*Generation*), modelo (*model*) e algoritmo (*algorithm*).

Em *Evaluation*, foram implementadas as classes Felli, Cigar e SKLearnEvaluation, responsáveis por avaliar a função objetivo. Particularmente, a classe SKLearnEvaluation é dedicada à avaliação do método *evaluation_pipeline* do *SKLearn*, executando métodos de pré-processamento da gramática e métodos de penalização para gramáticas inválidas, enquanto *Felli* e *Cigar* foram utilizadas para a experimentação inicial da técnica em testes de sanidade do algoritmo, e serão desconsideradas no decorrer deste relatório. A função de penalização foi desenvolvida de maneira heurística, e é dada pela fórmula 2, onde *base_penalization* é um valor de penalização padrão, que é somado ao produto de *get_penalization_term*, um fator de correção proporcional à quantidade de algarismos inteiros presentes no valor *get_norm_value*. *get_norm_value* representa, em absoluto, a soma dos valores abaixo de zero, que é o limite inferior de qualquer parâmetro do tipo numérico da gramática. Para corrigir os valores de tipo inteiro presentes na gramática e que, eventualmente ajustados pelo CMA-ES apresentam valores reais, utilizou-se o método *parse_integers_in_evaluation* que, efetivamente, substitui as entradas de tipo inteiro da gramática otimizada pelos valores obtidos pelo CMA-ES convertidos em inteiro.

```
class SKLearnEvaluation:
  def run(self, x):
    candidate = load_pipeline(x)
    candidate = parse_integers_in_evaluation(candidate)
    results = evaluate_pipeline(candidate)

    if invalid(results):
      return penalized_value(candidate)
    return results['f1_weighted']['mean']
```

Fig. 1: Pseudo-código do método de avaliação *evaluation_pipeline*

```
class SKLearnEvaluation:
  def penalized_value(self, x):
    return base_penalization +
            get_penalization_term_from(x) *
            get_norm_value_from(s)
```

Fig. 2: Pseudo-código do método de penalização para entradas inválidas em *evaluation_pipeline*

A métrica *f1_weighted* é utilizada pelo método *evaluation_pipeline*, que é a soma ponderada da métrica de precisão e de revocação e, portanto, representa uma métrica de maximização. Portanto, o método de avaliação *run*, presente na classe SKLearnEvaluation, retorna o inverso do valor de *f1_weighted* retornado pelo método *evaluation_pipeline*, já que o método CMA-ES é orientado a problemas de minimização.

Em *library/evolution*, foi implementado a classe Recombination, que executa o produto interno entre o vetor de candidatos a solução e os pesos definidos pela variável *weight*, utilizado pelo método CMA-ES para ponderar a contribuição dos candidatos mais adaptados na etapa de geração de novos indivíduos.

O diretório *library/model* contém a implementação da classe Cmaes. O método *next* executa as etapas de mutação, recombinação, seleção de candidatos da população e a avaliação dos valores da função objetivo *evaluate_pipeline*. Portanto, a classe *Cmaes* recebe, via injeção de dependências, os métodos que o algoritmo utilizará em cada etapa presente no método *next*.

Por fim, o diretório *library/algorithm* contém a implementação da classe Run, que implementa o laço

Tabela II: Resultados de F1 Weighted para cada Base de Dados

Pipelines / Datasets	IRIS	MAGIC	ML-PROVE	WILT	WINEQUALITY-WHITE
1)	0,98663325	0,34235671	0,26205578	0,91984267	0,32395198
2)	0,95979849	0,77819269	0,40603809	0,91984267	0,28853921
3)	0,93299714	0,73923581	0,22195078	0,91984267	0,48342169
4)	0,95938825	0,55463417	0,19210759	0,79336383	0,23540006
5)	0,91841182	0,77810186	0,31171814	0,91984267	0,28864608
6)	0,95293064	0,83348789	0,40374297	0,98484899	0,44705200
7)	0,92566241	0,69507880	0,31326425	0,92674614	0,29926149
8)	0,94646515	0,73442649	0,32113266	0,98140835	0,48342169
9)	0,10000000	0,78129377	0,30855605	0,95199846	0,40805780
10)	0,96664996	0,60585892	0,32659913	0,93191578	0,32077365
Média	0,86489371	0,68426671	0,30671654	0,92496522	0,35785257
Mediana	0,94646515	0,73442649	0,31171814	0,92496522	0,32395198
Desvio Padrão	0,23449810	0,12816786	0,05936559	0,04562351	0,07832161
Tempo de execução (h)	01:41:14	03:08:27	03:04:14	01:28:19	03:06:43

de controle do método que avalia o modelo *Cmaes* pela chamada iterativa do método *next* e garante a terminação do algoritmo, seja pelo ajuste do *fitness* de acordo com a tolerância definida, seja pela terminação do número de iterações máximo.

IV. Metodologia

O método CMA-ES atende, originalmente, ao problema de otimização de valores reais, sendo, portanto, não completamente aderente à gramática do *AutoML Framework* que possui uma combinação de valores categóricos utilizados no processo de avaliação de pipelines. Portanto, para avaliar a solução implementada, o método principal (*main*) itera sobre gramáticas candidatas gerado pelo método *get_random_candidates*, da classe CANDIDATE_SOLUTION. O método *get_random_candidates* é utilizado pelo algoritmo *random_search* para obtenção de combinações válidas de pipeline a partir da gramática e foi, portanto, utilizado pelo algoritmo implementado neste trabalho para gerar combinações de gramática que terão seus valores numéricos otimizados pelo *Cmaes*. Um exemplo de pipeline gerado a partir da gramática é: 'PCA 2 TRUE AUTO 0.01 13 SVC 0.03125 LINEAR 0.03125 800.0 17 0.09 4000' que terá seus valores numéricos ajustados iterativamente pelo CMA-ES. A cada avaliação, a string é reconstruída com os valores candidatos daquela iteração do CMA-ES, mantendo-se a ordem original dos valores numéricos na string, para a avaliação pelo método *evaluate_pipeline*.

Avaliamos ao algoritmo CMA-ES com as bases de dados IRIS, MAGIC, ML-PROVE, WILT e WINEQUALITY-WHITE, que são bases de dados utilizdas em *benchmarking* de algoritmos de aprendizagem de máquina. Foram geradas 10 combinações de pipelines, com 100 iterações cada, que foram submetidos ao CMA-ES com o objetivo de obter variabilidade dos parâmetros categóricos artificialmente, isto é, não provocada diretamente pela adaptação do CMA-ES devido à sua limitação de não trabalhar com valores categóricos.

V. Resultados

A Tabela II apresenta os resultados de F1 Weighted, para cada um dos 10 Pipelines avaliados em cada um das 5 bases de dados. Para cada pipeline, 100 iterações foram executadas a fim de obter o valor final apresentado na tabela. Os pipelines são:

1) PCA 2 True auto 0.01 13 SVC 0.03125 linear 0.03125 800.0 17 0.09 4000
2) logR l1 0.0 2048.0 False 350 True
3) PCA 9 True arpack 0.08 4 gaussianNB
4) kBest 4 SVC 2048.0 linear 2.0 30.0 2 0.05 200
5) kBest 1 logR l1 0.01 2048.0 True 200 True
6) PCA 5 False auto 0.06 10 MLP invscaling 0.3 0.4 200 relu
7) PAC 0.05 512.0 500 False
8) PCA 10 True randomized 0.08 16 gaussianNB
9) PCA 7 False auto 0.08 5 SGD l2 0.1 300 modified_huber True
10) PAC 0.02 32.0 4000 True

É possível observar que a base de dados que apresentou os melhores resultados foi o WILT, possuindo a maior média e mediana, além do menor desvio padrão. Foi também aquela que apresentou o menor tempo de execução entre as 5 avaliadas. O pior caso ocorreu para o ML-PROVE, que apresentou os menores valores de média e mediana. Foi o terceiro maior tempo de execução, porém com uma diferença de apenas 4 minutos para o MAGIC, o qual gastou mais tempo.

O pipeline número 6 apresentou os melhores resultados para MAGIC e WILT. Para o IRIS o melhor foi o pipeline 1, para o ML-PROVE o número 2 e para o WINEQUALITY-WHITE o pipeline de número 8. O pipeline número 4 apresentou os piores valores para WILT e WINEQUALITY-WHITE. Analisando a média dos resultados de pipeline em todas as bases de dados, o número 6 teve o melhor resultado com o valor de 0,72441250, enquanto a menor média ocorreu para o número 9 com aproximadamente 0,50998122.

Na base de dados IRIS para o pipeline 9 foi encontrado o valor final de 0,1. Analisando esse caso especificamente, até as 25 primeiras iterações foi encontrado o valor 0,1, sendo que na trigésima iteração o valor calculado foi de 0.93251594. Houve uma oscilação entre valores altos e baixos, até o término da execução na centésima iteração. É

provável que o algoritmo tenha encontrado um ótimo durante a maximização, tenha continuado tentando achar melhores valores expandindo a área de busca sem sucesso. A partir da iteração 75 o valor permaneceu em 0,1.

Com exceção do caso anteriormente citado, os valores baixos encontrados pelo algoritmo apresentaram variações a cada iteração, com uma variância pequena. Observando essas execuções, houve momentos em que o algoritmo encontrou melhores valores e depois regrediu, apresentando um padrão de inconstância e não convergente. Em alguns casos de valores altos obtidos, verificamos que o algoritmo encontrou logo nas primeiras iterações um bom valor e continuou alternando o resultado em torno desse valor inicial.

Considerando o desempenho geral do CMA-ES para todas essas execuções, a média de todos os valores encontrados foi de 0,62773895 e a mediana 0,71475265. O desvio padrão considerando a população inteira de 50 valores foi 0,29069391. O tempo total de execução foi de aproximadamente 12,5 horas.

VI. Conclusão

Neste trabalho, avaliou-se a implementação do método CMA-ES apresentado por [5] no contexto de otimização de hiperparâmetros de pipelines de avaliação de algoritmos de aprendizagem de máquina. O CMA-ES, conforme defendido pela literatura, possui capacidade de otimizar funções de avaliação caixa-preta com um baixo custo de tempo, sendo portanto, ideal em pipelines que possuem alto número de camadas e parâmetros.

Além disso, outras características como a invariância à base do CMA-ES, que é especialmente valiosa por permitir que o algoritmo mantenha a qualidade da busca mesmo que uma transformação de base tenha sido realizada sobre o espaço de busca do problema, tornam o CMA-ES um algoritmo popular para otimização de hiperparâmetros em modelos de aprendizagem de máquina, um problema inerentemente não-linear, caixa-preta, com diferentes translações do sistema de coordenadas devido a grande variabilidade de algoritmos como redes neurais, *Naive Bayes*, *Deep Learning*, *SVM*, etc. A avaliação do CMA-ES, neste trabalho, apresentou uma adaptação do *AutoML Framework* para lidar com a existência de valores categóricos.

O algoritmo implementado realizou transformações nos pileines candidatos para extrair seus valores numéricos e construir novos candidatos a partir da otimização destes. Os experimentos demostraram que o CMA-ES obteve bons resultados para três das cinco bases de dados, avaliando a média e a mediana dos pipelines utilizados. A média geral para todas as bases de dados e pipelines avaliados foi de aproximadamente 0,63 e a mediana 0,71. Pode-se dizer que o método obteve um resultado bom, considerando todos os contextos analisados.

References

[1] Thomas G Dietterich. Machine learning for sequential data: A review. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 15–30. Springer, 2002.

[2] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

[3] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[4] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[5] Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

[6] Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *ICGA*, pages 57–64, 1995.

[7] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[8] DCC UFMG. Automl framework.