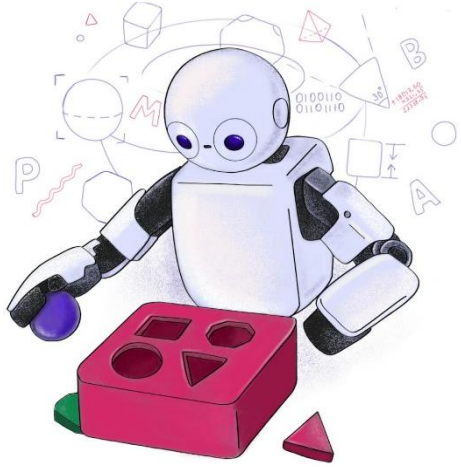
TP558 - Tópicos avançados em Machine Learning:

LLMatic: NAS via LLM e QD Optimization



Inatel

Pedro Guerrato
pguerrato@gmail.com

Introdução

Um dos grandes desafios no deep learning é criar arquiteturas de redes neurais eficazes.

A Busca por Arquiteturas Neurais (Neural Architecture Search - NAS) é o termo genérico usado para esse atividade de automatizar o processo de busca da melhor arquitetura de uma rede neural.

Introdução

Toda ANN (Artificial Neural Network) tem um objetivo especifico, como por exemplo, no caso do artigo é a classificação de imagens utilizando CNN.

A definição do problema pode ser vista como o **estabelecimento de um objetivo** — por exemplo, alcançar a máxima acurácia em uma tarefa de classificação, respeitando restrições de parâmetros e número de ciclos de treinamento — e a formulação da busca pela arquitetura que otimize esse objetivo.

Cada avaliação consiste em **treinar a arquitetura candidata** utilizando métodos como Gradiente Descendente sobre um conjunto de dados de referência, de modo a aferir seu desempenho. Esse processo geralmente demanda a exploração e consequente descarte de milhares de arquiteturas distintas.

Introdução

As duas abordagens mais comuns para criação de NAS são:

Algoritmos de *Reinforcement Learning*: Que geralmente é dado pelo treinamento de uma outra rede ”controladora” que sempre que consegue oferecer uma arquitetura melhor, é premiada; e quando o inverso, é penalizada.

Algoritmos de *Evolutionary Computation*: Onde ocorre diretamente no espaço das arquiteturas neurais, mantendo-se uma população avaliada por métricas de desempenho. Essa abordagem se aproxima da neuroevolução, existente desde os anos 1980, mas adaptada para NAS.

Introdução

As duas abordagens mais comuns para criação de NAS são:

Algoritmos de *Reinforcement Learning*: Que geralmente é dado pelo treinamento de uma outra rede ”controladora” que sempre que consegue oferecer uma arquitetura melhor, é premiada; e quando o inverso, é penalizada.

Algoritmos de *Evolutionary Computation*: Onde ocorre diretamente no espaço das arquiteturas neurais, mantendo-se uma população avaliada por métricas de desempenho. Essa abordagem se aproxima da neuroevolução, existente desde os anos 1980, mas adaptada para NAS.

Fundamentação teórica

O processo de procurar pelas melhores arquiteturas por meio de *Reinforcement Learning* ou *Evolutionary Computing,* por mais que contribuam para uma solução, ainda são lentas e imprecisas.

Por que não explorar por conhecimentos já obtidos para melhores arquiteturas de ANN?

Fundamentação teórica

O artigo “*LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization*” de Nasir et al. propõe combinar LLM com algoritmo evolucionário para gerar arquiteturas com performances no estado da arte na atividade de NAS.

Fundamentação teórica

Os LLMs são reconhecidos por sua capacidade de gerar código e pelo conhecimento do estado da arte por ser treinado por diversas fontes. O artigo propõe usar essa habilidade para introduzir variações significativas no código que define as redes neurais.

No entanto, um **LLM por si só não consegue encontrar uma arquitetura ideal**, pois não pode testar as arquiteturas e aprender com os resultados. Por isso, o artigo sugere combinar o conhecimento de domínio dos LLMs geradores de código com um mecanismo de busca robusto.

Fundamentação teórica

Sendo assim, é possível adotar a estratégia de utilizar algoritmos de *Quality Diversity* (QD) *Optimization* que permite encontrar um conjunto de soluções.

QD é um campo de pesquisa dentro de **algoritmos evolucionários e otimização** que busca **não apenas encontrar a melhor solução possível**, mas sim **descobrir um conjunto diverso de soluções de alta qualidade**.

Em contraste com a otimização tradicional, que tende a convergir para uma única solução ótima, os algoritmos QD exploram o espaço de busca de forma a **equilibrar qualidade (performance)** e **diversidade (variedade de soluções com características diferentes)**.

Fundamentação teórica

Por exemplo:

Em um problema de projeto de robôs que precisam andar, um algoritmo clássico de otimização tentaria encontrar **o robô mais rápido possível**.

Um algoritmo de QD, por outro lado, vai tentar encontrar **vários tipos de robôs eficientes** (um que corre com quatro pernas, outro que rola, outro que salta, etc.), oferecendo **um portfólio de soluções boas e diversas.**

Arquitetura e funcionamento

O sistema é composto por dois arquivos evolutivos principais que trabalham em cooperação:

Arquitetura e funcionamento

1. **Arquivo de Redes (Network Archive)**
   - Armazena arquiteturas de redes neurais candidatas.
   - Cada rede é descrita por dois **descritores comportamentais**:
     - **Relação largura/profundidade** (width-to-depth ratio).
     - **FLOPS** (operações de ponto flutuante por segundo, mais correlacionado ao tempo real de treino do que apenas número de parâmetros).
   - O **fitness** das redes é a **acurácia de teste** após treinamento.
2. **Arquivo de Prompts (Prompt Archive)**
   - Armazena combinações de **prompts + temperatura do LLM** usados para gerar código de redes.
   - A temperatura controla a aleatoriedade da saída do LLM: baixa → determinística, alta → mais criativa.
   - O fitness é baseado na **curiosidade**: se um prompt gera redes que entram no arquivo de redes e melhoram desempenho, ele ganha pontuação.

Arquitetura e funcionamento

O processo segue uma dinâmica evolutiva guiada pelo LLM e pelo QD:

1. Inicialização

- Começa com uma rede simples (1 camada convolucional + 1 camada densa).
- Gera redes iniciais a partir de prompts aleatórios utilizando CodeGen-6.1B.

2. Evolução

- Em cada geração, decide-se aplicar **mutação (70%)** ou **crossover (30%)**.
- **Mutação**: seleciona um prompt e uma rede, gera uma nova variante de arquitetura.
- **Crossover**: combina duas redes semelhantes usando instruções de um prompt fixo.
- A temperatura do LLM é ajustada dinamicamente:
  - ↑ se as novas redes superam as anteriores (exploração).
  - ↓ se as redes pioram (exploração mais controlada).

Arquitetura e funcionamento

O processo segue uma dinâmica evolutiva guiada pelo LLM e pelo QD:

3. **Treino e Avaliação**
   - Cada rede é treinada por algumas épocas no dataset alvo (inicialmente CIFAR-10 e, posteriormente extendido para NAS-Bench-201).
   - O desempenho (acurácia) determina se ela entra no **Network Archive**.
   - O prompt responsável é avaliado e, se for útil, adicionado ao **Prompt Archive**.
4. **Quality Diversity (QD)**
   - O algoritmo usado é o **CVT-MAP-Elites**, que organiza os indivíduos em nichos (células) no espaço de descritores.
   - Isso garante não só alta performance, mas também **diversidade** de arquiteturas (redes pequenas, médias e grandes, por exemplo).

Treinamento e otimização

No caso do NAS, toda vez que um candidato é considerado promissor, o treinamento.

Cada rede gerada pelo LLMatic passa por um **treinamento supervisionado padrão** (quando válida), mas com recursos controlados para eficiência:

Treinamento e otimização

1. **Configuração da rede inicial**
   - Uma convolução (3 canais de entrada, kernel 1x1, 1 saída).
   - Uma camada densa com 1024 neurônios conectada a 10 saídas (classes).
   - Ativação: **ReLU** em todas as camadas.
2. **Treinamento de cada candidato**
   - **Número de épocas**: 50.
   - **Otimizador**: SGD (Stochastic Gradient Descent).
     - **Taxa de aprendizado (lr)** = 0.001
     - **Momentum** = 0.9
   - **Loss function**: Cross-Entropy Loss.

Treinamento e otimização

3. **Avaliação (Fitness)**
   - O desempenho é medido pela **acurácia no conjunto de teste**.
   - Esse valor é usado como **fitness** no **Network Archive**.
4. **Atualização do Prompt Archive**
   - Se o prompt gerou uma rede que entrou no Network Archive ou superou a geração anterior, seu **fitness aumenta**.
   - Caso contrário, o fitness cai, e a curiosidade associada ao prompt também diminui.

Vantagens

1. **Eficiência em número de avaliações**

   Encontra arquiteturas competitivas em **~2.000 avaliações**, enquanto métodos tradicionais (ex.: EfficientNet-B0) precisaram de **~8.000**.

2. **Exploração guiada por conhecimento prévio**

   Ao usar LLMs treinados em código, o sistema aproveita o conhecimento prévio de arquiteturas de redes neurais já embutido nesses modelos.

3. **Diversidade de soluções (Quality Diversity)**
   - Em vez de encontrar apenas uma rede ótima, o LLMatic gera um conjunto de arquiteturas diversas e de qualidade (redes pequenas, médias e grandes).
   - Isso ajuda em cenários práticos, como escolher arquiteturas para dispositivos com diferentes restrições de hardware (RAM, FLOPS, etc.).

Vantagens

4. **Combinação inovadora de dois arquivos cooperativos**
   - Network Archive → avalia arquiteturas em termos de acurácia + descritores (largura/profundidade, FLOPS).
   - Prompt Archive → avalia a eficácia dos prompts e ajusta automaticamente a temperatura do LLM.
   - Essa cooperação gera um ciclo virtuoso: bons prompts → boas redes → melhora dos arquivos → melhores próximos prompts.
5. **Resultados competitivos com SOTA (state-of-the-art)**
   - Em CIFAR-10 e NAS-Bench-201, o LLMatic chegou muito próximo dos melhores resultados conhecidos, apesar de usar menos recursos.
   - Superou outros métodos baseados em LLM (como GENIUS, com GPT-4).

Desvantagens

1. Existem LLMs maiores que possibilitariam melhorar a **qualidade e a criatividade das arquiteturas geradas;**
2. CodeGen-6.1B é pago. Seria interessante o uso de um LLM gratuito;
3. Alto custo computacional.

Exemplo(s) de aplicação

1. Implantação em dispositivos com diferentes restrições de hardware (ex. TinyML);
2. Cenários de trade-off entre desempenho e custo;
3. Uso em cenários em novos domínios precisam ser entrados rapidamente;
4. Cenários com risco de overfitting em soluções únicas;
5. Aplicações educacionais e de pesquisa em AutoML;

Comparação com outros algoritmos

1. Reinforcement Learning (RL-NAS)

- **Como funciona**: um controlador (rede neural) gera arquiteturas, recebe como recompensa a acurácia das redes treinadas.
- **Limitação**: precisa de muitas avaliações (muito caro).
- **LLMatic vs RL-NAS**:
  - LLMatic usa **LLM + evolução/QD**, guiando a busca com conhecimento de código.
  - Encontra boas arquiteturas em **menos avaliações** e com mais **diversidade de soluções**.

Comparação com outros algoritmos

2. Evolução pura (Neuroevolution / Evolutionary NAS)

- **Como funciona**: populações de redes evoluem por mutação e crossover.
- **Limitação**: busca é **“cega”** — as mutações não são guiadas por conhecimento prévio.
- **LLMatic vs Evolução pura**:
  - O LLM gera arquiteturas mais **plausíveis e treináveis**, pois tem **conhecimento embutido** de boas práticas em código.
  - Menos arquiteturas inválidas → maior eficiência.

Perguntas?

Referências

- LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization
  https://arxiv.org/pdf/2306.01102

- Neural Architecture Search with Reinforcement Learning
  https://arxiv.org/pdf/1611.01578

- Evolution through Large Models
  https://arxiv.org/pdf/2206.08896

Obrigado!

Quiz!

https://forms.gle/FRc6wmk1q6ioX7F48