

PYTHIA: AI 지원 코드 완성 시스템

Alexey Svyatkovskiy Microsoft One Microsoft Way Redmond, WA 98052 alsvyatk@microsoft.com	자오 잉 마이크로소프트 하나의 마이크로소프트 방식 레드몬드, WA 98052	푸 성우 마이크로소프트 하나의 마이크로소프트 방식 레드몬드, WA 98052	닐 순다레산 마이크로소프트 하나의 마이크로소프트 방식 레드몬드, WA 98052
--	---	---	---

추상적인

본 논문에서는 AI 지원 코드 완성을 위한 새로운 중단 간 접근 방식인 Pythia를 제안합니다. 편집 시 소프트웨어 개발자가 사용할 수 있는 메서드 및 API 권장 사항의 순위 목록을 생성합니다. 이 시스템은 현재 Visual Studio Code IDE에서 Intellicode 확장의 일부로 배포됩니다. Pythia는 추상 구문 트리에서 추출한 코드 컨텍스트에서 훈련된 최첨단 대규모 심층 학습 모델을 활용합니다. 100ms 정도에서 가장 일치하는 코드 완성을 예측하는 높은 처리량으로 작동하도록 설계되었습니다.

우리는 시스템의 아키텍처를 설명하고, 빈도 기반 접근 방식과 호출 기반 Markov Chain 언어 모델을 비교하고, 경량 클라이언트 장치에서 Pythia 모델을 제공하는 문제에 대해 논의합니다.

2700개의 Python 오픈 소스 소프트웨어 GitHub 리포지토리에서 얻은 오프라인 평가 결과는 92%의 상위 5개 정확도를 보여 프로젝트 내 및 프로젝트 간 설정 모두에서 클래스에 대해 평균 20% 기준 모델을 능가합니다.

키워드 코드 완성, 신경망, 소프트웨어의 자연성

1. 소개

통합 개발 환경(IDE)[1]을 통한 소프트웨어 개발에서 코드 완성은 가장 널리 사용되는 기능 중 하나입니다. 지능적인 코드 완성 [2]은 입력 오류 및 기타 일반적인 오류를 줄여 개발자 생산성을 효과적으로 향상시켜 개발자를 지원합니다. 이러한 기능에는 입력 시 팝업, 유형이 지정된 클래스 및 개체에 대한 메서드 및 API 호출, 함수의 매개 변수 쿼리, 변수 또는 함수 이름 명확성, 코드 컨텍스트를 사용하여 다음 코드를 안정적으로 예측하는 프로그램 구조 완성이 포함될 수 있습니다.

IDE의 기존 코드 완성 도구는 일반적으로 사용자가 "."를 입력할 때 호출할 수 있는 모든 가능한 특성 또는 메서드를 나열합니다. 또는 "=" 트리거 문자. 그러나 제안된 결과의 순위를 지정하지 않으면 사용자가 알파벳순으로 정렬된 긴 목록을 스크롤해야 하므로 메서드의 전체 이름을 직접 입력하는 것보다 훨씬 느린 경우가 많습니다. 연구에 따르면 개발자는 선택 항목 수를 줄이기 위해 접두사 필터링에 크게 의존하는 경향이 있습니다[3].

이 백서에서는 대규모 오픈 소스 코드 데이터 세트에서 추출한 코드 스니펫으로 훈련된 신경 코드 완성 시스템인 Pythia를 소개합니다. 시스템의 기본 작업은 주어진 코드 스니펫에서 가장 가능성이 높은 메서드를 찾는 것입니다. 즉, 원본 코드 조각 C , 어휘 V 및 가능한 모든 메서드 세트 $M \subset V^*$ 가 주어지면 다음을 결정하려고 합니다.

$$m^* = \operatorname{argmax}(P(m|C)), \forall m \in M. \tag{1}$$

해당 방법을 찾기 위해 응답에 점수를 매긴 다음 가장 가능성이 높은 방법을 결정할 수 있는 모델을 구성합니다.

정적 및 동적으로 유형이 지정된 언어에 대한 많은 지능형 코드 완성 시스템이 문헌 [4, 5, 2, 6, 7, 8]에서 제안되었습니다. BMN(Best Matching Neighbor) 및 n-gram과 같은 통계적 언어 모델, 소스 코드의 순차적 특성을 활용하는 RNN(Recurrent Neural Network) 기반 접근 방식은 이러한 시스템을 만드는 데 특히 효과적이었습니다. 대부분의 과거 접근 방식은 소스 코드의 장거리 순차 특성을 활용하지 않았거나 코드의 AST(추상 구문 트리)에서 제공하는 고유한 기회를 활용하지 않고 자연 언어 메서드를 전송하려고 시도했습니다.

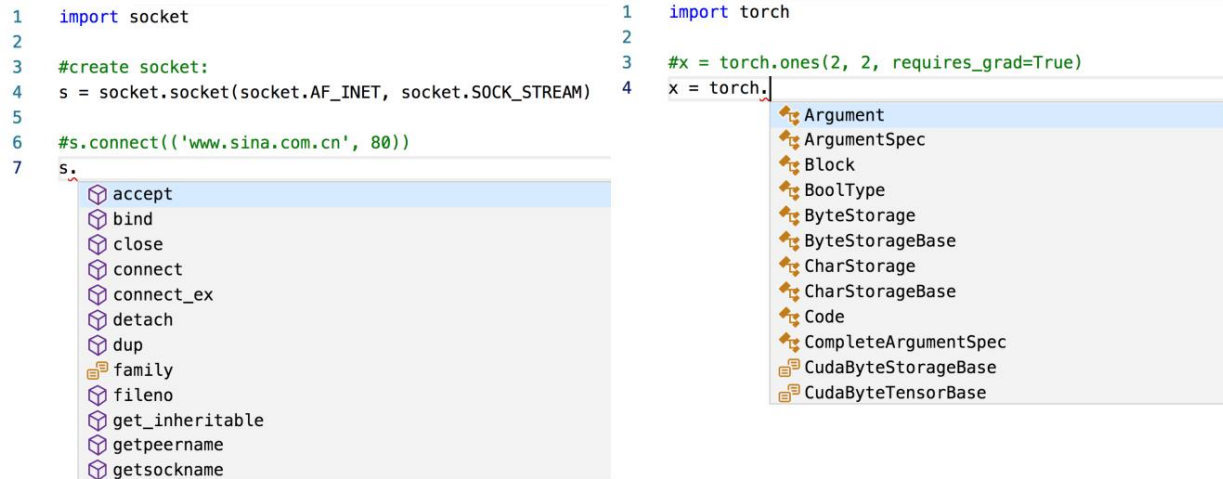


그림 1: 알파벳 순서로 제안을 제공하는 코드 완성 시스템. 왼쪽: "소켓" Python 라이브러리, 오른쪽: "PyTorch".

코드 완성 문제의 본질은 LSTM(Long Short-Term Memory) 네트워크 [9] 를 유망한 후보로 만듭니다. Pythia는 멤버 액세스 표현식 및 모듈 함수 호출을 포함하는 코드 스니펫에 해당하는 부분 AST를 모델 교육을 위한 입력으로 사용하여 멀리 떨어진 노드에서 전달되는 의미 체계를 포착하는 것을 목표로 합니다.

이 백서의 주요 기여는 다음과 같습니다. (i) 주파수 기반 및 Markov Chain 모델을 포함하여 코드 완성 시스템의 여러 기준을 도입하고 구현합니다 (섹션 2 참조). (ii) 새로운 end-to-end를 제안하고 배포합니다. -소스 코드 스니펫(섹션 4 및 5 참조)에서 훈련된 LSTM 기반 코드 완성 시스템 종료, (iii) 실제 소스 코드에서 추출한 1,580만 메서드 호출 데이터 세트에서 모델을 평가하여 최상의 모델임을 보여줍니다. 92%의 정확도를 달성하여 더 단순한 기준선을 능가합니다(섹션 7 참조), (iv) 고성능 컴퓨팅 클러스터에서 심층 신경망 훈련 및 하이퍼파라미터 조정, 높은 처리량으로 경량 클라이언트 장치에 모델 배포에 대한 실질적인 문제에 대해 논의하고 문서화합니다 (참조 섹션 8).

2 기본 코드 완성 시스템

IDE의 가장 기본적인 코드 완성 시스템은 가능한 모든 특성 또는 메서드를 알파벳순으로 나열합니다. 그림 1은 그러한 시스템이 제공할 수 있는 예제 완성을 보여줍니다. 알 수 있듯이 이 접근 방식은 멤버 수가 상대적으로 적은 클래스(예: "소켓" Python 라이브러리)에 대해 정확한 결과를 산출할 수 있으며, 그렇지 않으면 개발자가 긴 목록을 스크롤해야 합니다(예: "PyTorch"의 경우).

2.1 주파수 모델

알파벳 순으로 순위를 매기는 방식 대신 인기도 순위를 채택하여 추천 관련성을 높일 수 있습니다. 생각할 수 있는 가장 간단한 빈도 기반 모델은 교육 코퍼스의 발생 횟수를 기반으로 각 클래스에 속하는 메서드를 주문하는 것입니다.

보다 강력한 기준선을 찾기 위해 우리는 if-조건에 대한 컨텍스트 정보를 통합하는 개선된 빈도 모델을 구축했습니다 (백서 전체에서 "빈도-if"라고 함). 다음에서는 if 조건 내부에 있는지 여부와 두 개의 해당 인기도 목록을 생성하는지 여부에 따라 모든 호출을 두 그룹으로 세분화합니다.

탭에서 볼 수 있듯이, 1, if 문 외부에서 가장 빈번한 메서드 호출은 변수 또는 자리 표시자 선언, 신경망 설정 또는 훈련 루틴과 관련됩니다. 그러나 if 조건 내에서 일반적인 호출은 파일 입출력 조작 래퍼, 컨텍스트 관리자(예: 즉시 실행 모드) 또는 텐서 속성 지정(희소 또는 조밀)에 대한 것입니다. 결과적으로 "frequency-if" 모델은 전반적인 정확도를 높이는 if 문 내에서 보다 정확한 코드 제안을 제공합니다.

기호	주파수	호출	주파수 905
tf.float32	10508	tf.context.in_graph_mode	640 624
tf.nn	8954	tf.gfile.Exists tf.gfile	448 322
tf.constant	7754	tf.ops.Tensor	
tf.train	6869	tf.context.executing_eagerly	
tf.variable_scope tf.자리 표	6330	tf.sparse_tensor.SparseTensor	
시자	5622		271

표 1: TensorFlow 라이브러리에서 가장 많이 사용되는 메서드 호출 빈도. 왼쪽: if 문 외부, 오른쪽: if 문 내부.

방법	전환 확률 0.22 0.15 0.13
os.rename, shutil.move os.write	0.11 0.10 0.29
shutil.copy	
os.path.isfile	
os.remove 다	
른 모든 방법	

표 2: 3-상태 Markov 체인의 세 번째 상태로 전환 확률: os.path.isfile → os.remove → ?.

2.2 호출 기반 마르코프 체인 모델

Markov Chain 모델은 순차 패턴 발견 [10, 11] 에서 결정 프로세스 모델링[12]에 이르기 까지 확률적 전이 모델링에서 큰 강점을 보여주었습니다. 소스 코드에서 호출 발생을 분석할 때 일부 호출 시퀀스 뒤에는 항상 동일한 메서드가 뒤따른다는 사실을 발견했습니다.

예를 들어 호출 체인 os.path.isfile → os.remove 다음에 오는 파일 조작과 관련된 "os" 및 "shutil" Python 모듈의 모든 함수 호출 을 보면 20% 이상의 "os.rename" 또는 "shutil.move"가 될 시간 입니다. 이 두 모듈에는 파일 조작과 관련된 30개 이상의 메서드가 있지만 .

다음 세 가지 상태의 Markov 프로세스를 고려합니다.

$$os.path.isfile \rightarrow os.remove \rightarrow ?,$$

여기서 세 번째 상태는 "os" 또는 "shutil" 모듈의 함수일 수 있으며 Tab에서 해당 전환 확률을 계산합니다. 2. 이전 호출 발생을 사용하여 다음에 사용될 호출을 예측 하는 것은 코드 완성의 Markov Chain 모델의 핵심입니다. $P(m_i | h_i) = P(m_i | m_{i-n}, \dots, m_{i-1})$,

(2)

여기서 $h_i = m_1, m_2, \dots, m_{i-1}$ 시퀀스는 주어진 클래스에 대해 동일한 문서 범위에서 이전 메서드 호출의 기록을 나타냅니다 . n차 Markov 체인 모델 에서 다음 호출 확 률은 이전 n - 1 호출에만 의존하며 , 이는 훈련 코퍼스 에서 호출 시퀀스 발생 수를 계산하여 추정됩니다 .

GitHub에서 사용할 수 있는 풍부한 고품질 소스 코드 데이터와 코드 스니펫을 추출하고 레이블을 지정할 수 있는 도구를 통해 보다 정확한 예측을 달성하는 고급 기계 학습 모델을 교육하고 배포할 수 있습니다 (섹션 5 참조).

3 데이터 세트

GitHub의 오픈 소스 Python 리포지토리에서 코드 완성 모델을 교육하고 평가하기 위해 데이터 세트를 수집했습니다. 과학 컴퓨팅, 머신 러닝, 데이터 흐름 프로그래밍, 웹 개발을 포함한 다양한 도메인 세트의 라이브러리를 포함하는 총 2700개의 최고 별(포크 제외) Python 프로젝트가 선택되었으며 1,580만 개 이상의 메서드 호출이 있습니다. 그림 2는 데이터 세트에서 메서드 호출 발생이 가장 많은 Python 라이브러리를 보여줍니다.

데이터 세트를 리포지토리 수준에서 70-30의 비율로 개발 및 테스트 세트로 나눕니다. 그런 다음 개발 세트는 무작위로 80-20의 비율로 훈련 세트 와 검증 세트로 나눕니다. 온라인으로 예측을 제공하기 위해 전체 데이터 세트를 사용하여 모델을 재교육합니다.

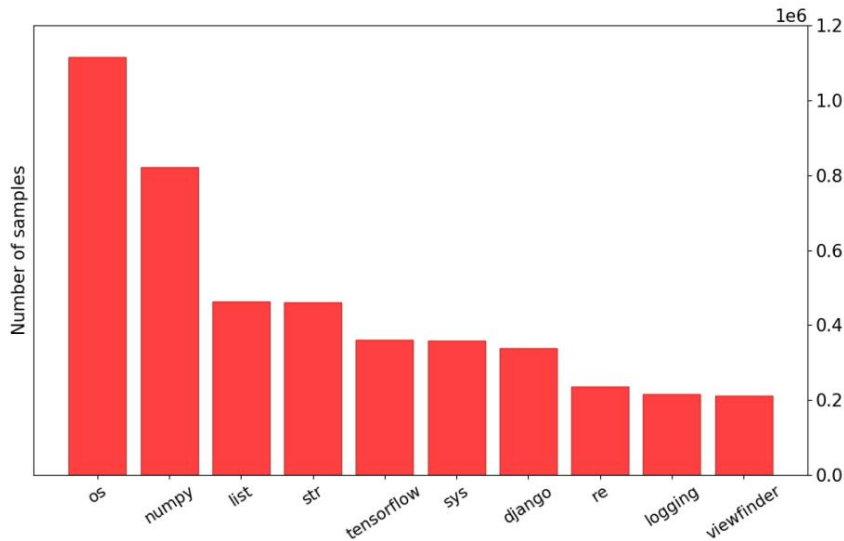


그림 2: 교육 데이터 세트의 상위 10개 라이브러리에 대한 라이브러리당 메서드 호출 수.

4 코드 스니펫 표현

모든 프로그래밍 언어에는 소스 코드를 AST로 구문 분석하는 데 사용할 수 있는 명시적 문맥 자유 문법(CFG)이 있습니다. AST는 뿌리가 있는 n항 트리이며 각 비리프 노드는 구조 정보를 지정하는 CFG의 비터미널에 해당합니다. 각 리프 노드는 구문 토큰 인코딩 프로그램 텍스트에 해당합니다.

Pythia 모델은 학습을 위한 입력으로 멤버 액세스 표현식 및 모듈 함수 호출을 포함하는 코드 스니펫에 해당하는 부분 파일 수준 AST를 사용합니다. LSTM에 공급하기 전에 AST는 순서대로 깊이 우선 순회에 따라 시퀀스로 직렬화됩니다. 트레이닝 시퀀스를 추출할 때 각 메서드 호출 이전에 최대 T개의 룩백 토큰을 유지합니다. 여기서 T는 Tab에서 제공되는 모델의 조정 가능한 매개변수입니다. 삼.

또한 AST가 LSTM에 의해 소비되기 위해서는 노드와 토큰이 숫자 벡터에 매핑되어야 합니다. LSTM의 가중치와 마찬가지로 토큰의 개별 값을 실제 벡터에 매핑하는 조회 테이블은 역전파를 통해 학습할 수 있습니다. Word2Vec 접근법 [13]은 자연어 처리에서 잘 작동하는 것으로 나타났으며 Pythia 시스템에서 사용되어 코드 스니펫을 조밀한 저차원 벡터로 나타냅니다.

각 입력 데이터 파일은 PTVS 구문 분석기 메타데이터를 사용하여 구 1 AST를 추출합니다. 메서드 호출 및 관련 문 분석되며 여기에는 호출 범위, 구문 노드의 직렬화된 시퀀스 및 AST의 토큰, 런타임 유형의 메소드 호출 수신자 토큰이 추출됩니다. 데이터 처리 워크플로우는 그림 3에 요약되어 있습니다.

구문 노드 및 토큰 이름은 1에서 V까지의 정수로 매핑되어 토큰 인덱스를 얻습니다. 어휘 크기를 줄이기 위해 자주 사용되지 않는 토큰이 제거됩니다. 교육 중에 코드 스니펫의 OoV(out-of-vocabulary) 토큰은 교육 코드 스니펫에서 동일한 OoV 토큰의 발생이 동일한 정수로 표시되는 방식으로 어휘 크기 V보다 큰 정수로 매핑됩니다. 각 교육 샘플은 레이블 곱셈을 사용하여 직렬화되는 AST입니다. 코딩의 종주론과 메서드 호출은 양의 레이블을 훈련하여 토큰 인덱스를 코드의 의미론적 관계를 보존하는 저차원 밀집 벡터에 매핑합니다.

4.1 유형 정보 활용

Python은 동적으로 유형이 지정되는 언어입니다. PEP 484는 Python 3.6부터 다양한 공통 유형에 대한 유형 주석을 도입했지만 대부분의 라이브러리는 아직 이를 채택하지 않았습니다. 따라서 Python 인터프리터는 코드가 실행될 때만 변수 유형 검사를 수행합니다. 개체 사용 패턴의 정적 분석을 기반으로 런타임 시 메서드 호출 수신자 토큰 및 로컬 변수의 유형을 추론하고 이 정보를 교육 시퀀스에 포함합니다.

¹<https://github.com/Microsoft/PTVS>

²<https://www.python.org/dev/peps/pep-0484>

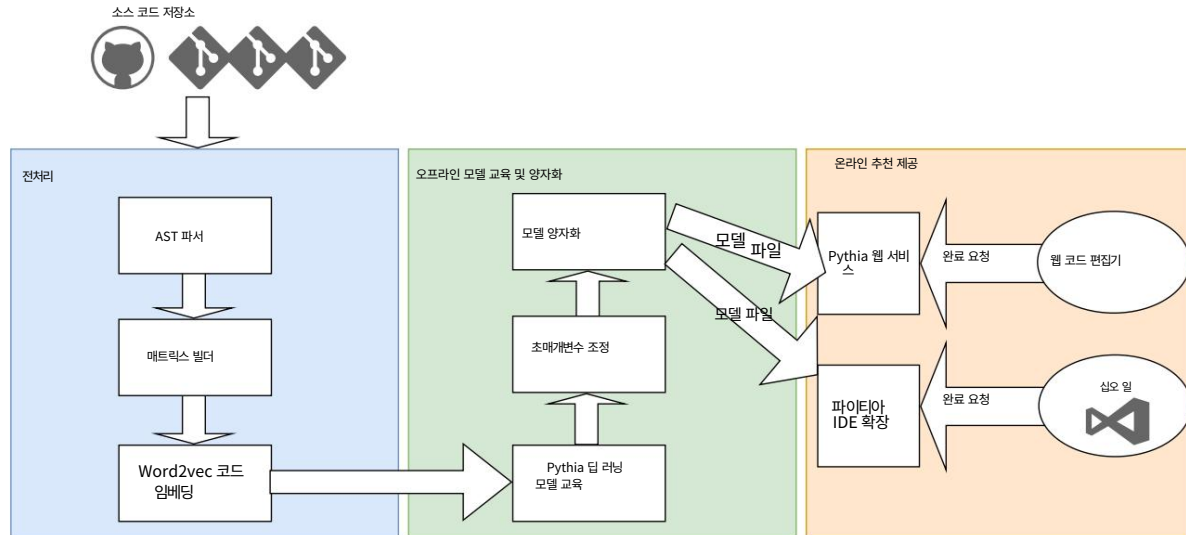


그림 3: Pythia 워크플로: 원시 소스 코드 파일에서 온라인 권장 사항 제공까지.

Python 프로그래밍 언어에서 사용자는 가져온 모듈의 별칭을 자유롭게 지정할 수 있습니다. 예를 들어, 일반적으로 사용되는 "import numpy as np"는 "import numpy as an_unusual_alias"일 수 있습니다. "np" 및 "numpy" 토큰에 대한 임베딩 벡터는 임베딩 공간 근처에 있습니다. 그러나 "an_unusual_alias"와 같이 자주 사용되지 않는 별칭은 어휘에 없는 토큰일 수 있으며 임베딩에 표시되지 않습니다. 훈련 중에 유형 정보를 포함한다는 것은 "numpy" 토큰에 해당하는 임베딩 벡터가 위에서 설명한 모든 경우에 사용된다는 것을 의미합니다.

변수 이름은 개발자가 선택한 명사이며 코드 구현마다 다를 수 있습니다. 원래 형식으로 유지하면 어휘 크기가 늘어나 코드 완성 시스템이 변수 이름의 철자에 의존하게 됩니다. 이를 방지하기 위해 변수 이름은 `var :< variabletype >` 규칙에 따라 정규화됩니다.

5 신경 코드 완성 모델

순환 신경망은 임의의 길이의 순차적 데이터를 처리할 수 있는 신경망 제품군입니다.

특히 성공적인 형태의 RNN인 장단기 기억 네트워크 [9]는 대규모 데이터 세트에서 훈련되어 이미지 캡션 [14]에서 감정 분석 및 기계 번역 [15, 16]에 이르기까지 다양한 도메인에서 놀라운 성능 결과를 얻을 수 있습니다. LSTM 장치는 순환 구성 요소 내에서 활성화 기능을 사용하지 않으며, 물류 기능을 사용하여 구현된 게이트를 사용하여 정보 흐름을 제어합니다. 따라서 저장된 값은 시간이 지남에 따라 반복적으로 확장되거나 압축되지 않으며 기술기가 훈련될 때 폭발하거나 사라지는 경향이 없습니다.

메서드 완료 작업은 코드 스니펫 C의 AST의 터미널 노드에 해당하는 구문 토큰 시퀀스 $ct, t = 0 \dots T$ 에 대한 조건부 토큰 m^* 을 예측하는 것입니다. 시퀀스 토큰 "." 이 문제는 LSTM을 사용한 시퀀스 학습에 적합합니다.

$$x_t = Lct, h_t = \quad (3)$$

$$f(x_t, h_{t-1}), P(m|C) = \quad (4)$$

$$y_t = \text{softmax}(W h_t + b), m = \text{argmax}(P(m|C)). \quad (5)$$

$$C)). \quad (6)$$

여기서 행렬 $L \in \mathbb{R}^{d \times |V|}$ 는 단어 임베딩 행렬이고, dx 는 단어 임베딩 차원이고 $|V|$ 어휘의 크기입니다. Pythia의 경우 함수 $f(\cdot, \cdot)$ 는 현재 입력과 이전 hidden state를 취하여 다음 시간 단계에서 hidden state를 생성하는 stacked LSTM을 나타냅니다. $W \in \mathbb{R}^{|V| \times d_h}$ 및 $b \in \mathbb{R}^{|V|}$ 출력 프로젝션 매트릭스와 바이어스입니다. dh 는 LSTM의 숨겨진 상태의 크기입니다.

[17]에서 영감을 받아 그림 4와 같이 출력 분류 행렬로 입력 단어 임베딩 행렬을 재사용하고 있습니다. 이를 통해 대규모 완전 연결 레이어를 제거하고 훈련 가능한 매개변수의 수와 디스크의 모델 크기를 크게 줄일 수 있습니다. 보다 구체적으로, 임의의 균일 분포에 따라 초기화된 투영 행렬 $A = (a)_{ij} \in \mathbb{R}^{d_h \times d_x}$ 를 소개합니다. LSTM으로 인코딩된 스니펫과 마지막 시간에 숨겨진 상태가 주어집니다.

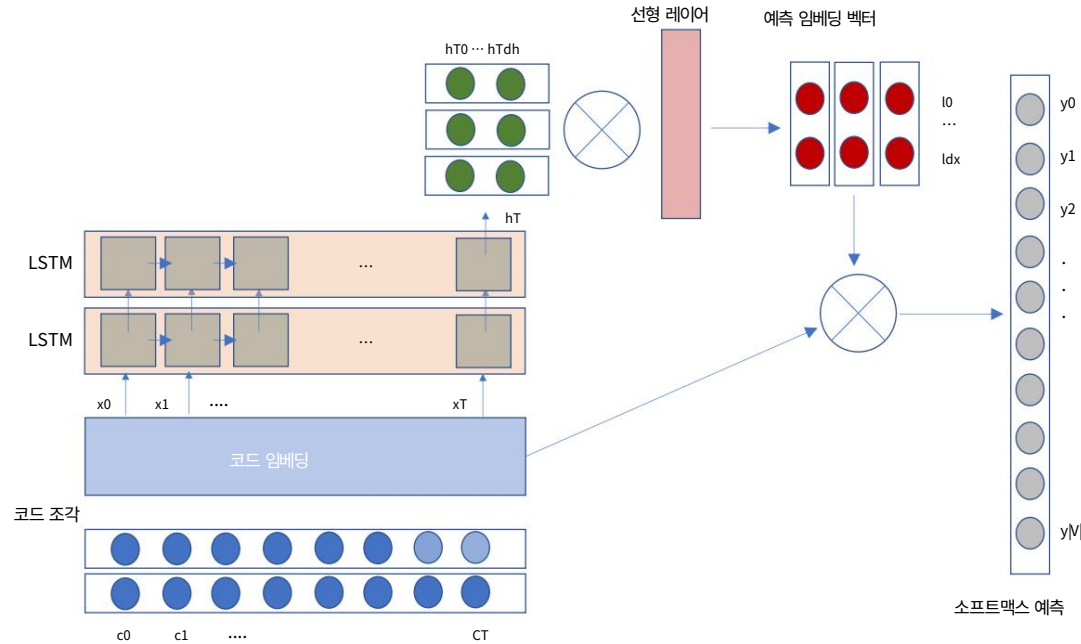


그림 4: Pythia 코드 완성 시스템에 배포된 신경망의 아키텍처.

단계 $h_T \in R_{dh}$, 둘을 함께 곱하여 예측 임베딩 벡터 $L_{pred} = (l_{pred})_j \in R_{dx}$ 를 다음과 같이 얻습니다.

$$l_{pred} = \begin{matrix} \text{hT} \\ \text{아이아이} \end{matrix} \quad (7)$$

결과적으로 신경망의 비정규화 예측은 다음과 같이 얻습니다.

$$y_k = \begin{matrix} l_{kj} \\ \text{아이} \end{matrix} \begin{matrix} \text{예측} \\ \text{아이} \end{matrix} + b_k \quad (8)$$

여기서 $b_k, k = 0 \dots |M| - 1$ 은 역전파 전에 0으로 초기화된 편향 벡터입니다.

6 모델 교육

신경망은 반복적으로 훈련되어 최소로 수렴되기 전에 전체 데이터 세트에 대해 여러 번 통과합니다.

BPTT(Backpropagation through time)는 LSTM을 stateful하게 훈련하기 위해 적용하는 기울기 기반 신경망 훈련 알고리즘입니다.

심층 신경망 훈련은 고성능 컴퓨팅 클러스터의 참여가 필요한 계산 집약적인 문제입니다. Pythia는 Adam 옵티마이저와 함께 데이터 병렬 분산 교육 알고리즘을 사용하여 각 작업자의 전체 신경 모델 사본을 유지하고 교육 데이터 세트의 다양한 미니 배치를 병렬 록스텝으로 처리합니다.

Pythia 시스템의 오프라인 교육 모듈은 분산 교육을 위해 TensorFlow와 CUDA 인식 MPI를 통합하는 Python 라이브러리로 구현됩니다. 소프트웨어 스택은 CUDA 9, CuDNN 7의 GPU 가속 딥 러닝 프리미티브 및 TensorFlow 1.10을 사용합니다. 모델 교육 및 하이퍼파라미터 최적화를 위해 Kubernetes 리소스 관리자와 함께 주문형 GPU 클러스터를 제공하는 Azure 클라우드 기반 Batch를 사용하며 ONNX와 PyTorch를 사용하여 모델을 배포합니다.

4.

6.1 일괄 처리 및 학습 속도 일정

Python 파일 수준 추상 구분 트리의 노드 수는 $O(10^2)$ 에서 $O(10^4)$ 까지입니다. 소스 코드에서 장거리 종속성을 활용하기 위해 100 - 1000 범위의 시퀀스 길이를 고려합니다.

³ <https://docs.microsoft.com/en-us/azure/batch-ai/overview>

⁴ <https://github.com/onnx/onnx>

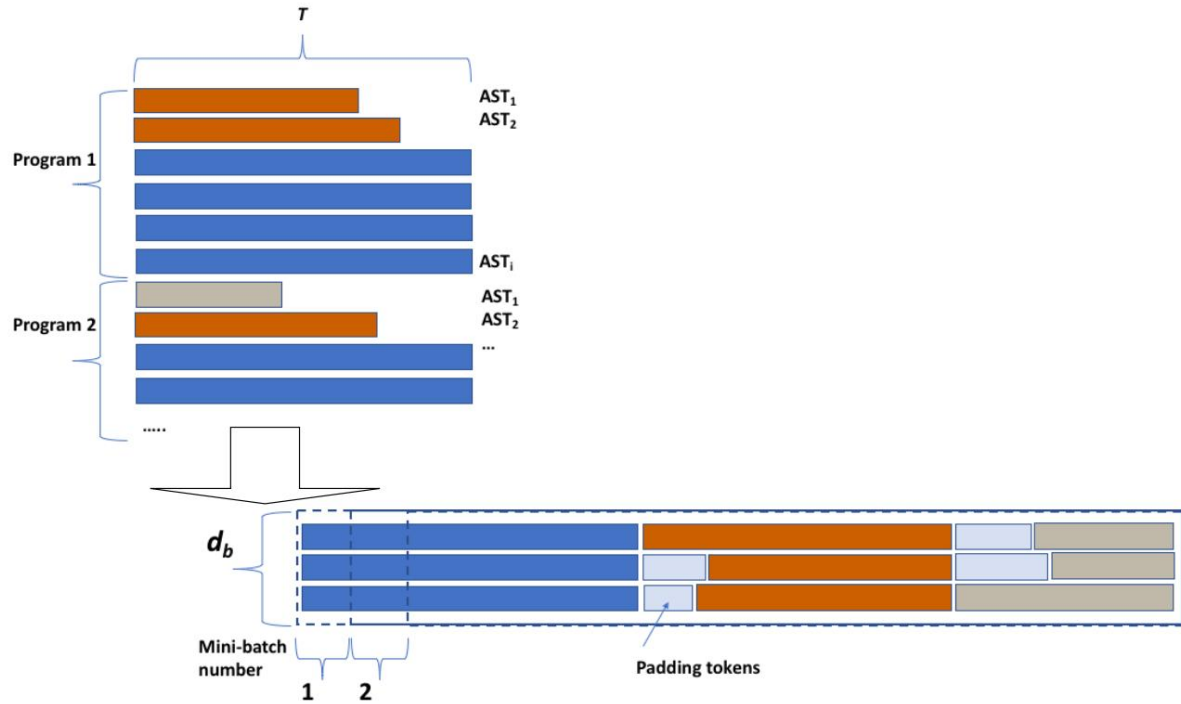


그림 5: 훈련 버퍼의 스냅샷.

긴 시퀀스에 대한 소실 문제를 해결하기 위해 우리는 시간을 통해 잘린 역전파를 통해 모델 매개변수에 대한 손실 기울기 계산을 근사화합니다 [18].

최신 GPU의 병렬성을 완전히 활용하려면 효율적인 일괄 처리가 중요합니다. 일반적으로 훈련 시퀀스는 그림 5와 같이 최대 길이가 T 인 가변 길이를 갖습니다. 훈련 샘플은 미리 정렬 되어 길이에 따라 세 개의 버킷으로 나뉩니다. 각 버킷 내에서 특수한 패딩 토큰을 사용하여 시퀀스를 최대 길이까지 패딩한 다음 마스킹을 통해 손실 계산에서 제외합니다. d_b 개별 AST에 속하는 시퀀스를 유지하기 위한 트레이닝 버퍼가 할당됩니다. 모든 학습 단계에서 버퍼의 첫 번째 TRNN 타임스텝이 다음 배치로 공급됩니다. 그런 다음 버퍼는 TRNN에 의해 이동됩니다. 시퀀스가 버퍼에서 완료될 때마다 새로운 시퀀스 집합이 로드되고 LSTM 내부 상태가 교육을 위해 재설정됩니다.

그래디언트 최적화 동안 가중치 업데이트의 크기를 제어하는 학습 속도는 지수 감쇠에 따라 각 에포크가 완료될 때 낮아집니다. 분산 체제에서 학습 속도는 처음 몇 epoch 동안 확장되어 신뢰할 수 있는 모델 수렴을 용이하게 합니다. [19]를 밀접하게 따라 처음 4세대("웜업" 기간) 동안 작업자 Nworker의 수에 비례하여 학습 속도를 선형으로 확장합니다.

$$\lambda_0(Nworker) = \lambda_0 \cdot \gamma^i \cdot \frac{Nworker}{\alpha} \quad (9)$$

여기서 λ_0 는 기본 학습률, γ 는 학습률 감쇠 상수, i 는 에포크 번호, 매개변수 α 는 워밍업 기간이 끝날 때 학습률을 제어하는 스케일링 비율, $\alpha = 4$ 는 하이퍼파라미터 검색을 통해 가장 잘 작동합니다.

그림 6은 분산 훈련으로 인한 훈련 시간 감소와 학습률 스케줄의 효과를 보여준다.

6.2 하이퍼파라미터

전반적으로 모델 아키텍처, 교육 절차 및 데이터 정규화는 예측 성능을 최대화하기 위해 조정해야 하는 많은 수의 하이퍼 매개변수를 생성합니다. 이러한 하이퍼파라미터에는 학습 속도 및 LSTM 레이어 수, 임베딩 공간 차원과 같은 수치 값뿐만 아니라 정확한 모델 아키텍처 또는 정규화 알고리즘과 같은 추상 범주 변수도 포함됩니다. 이러한 매개변수는 탭에 요약되어 있습니다. 삼.

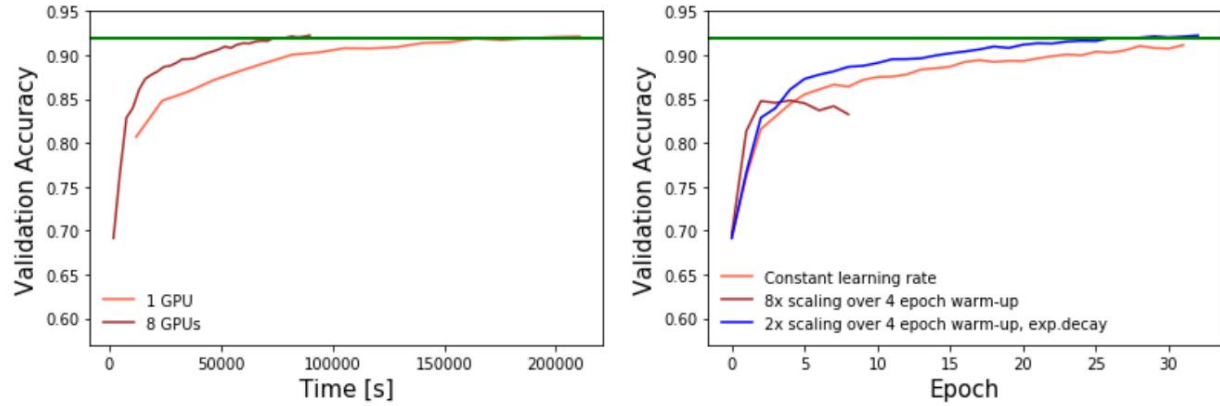


그림 6: 다양한 교육 체제에 대한 검증 세트의 정확도. 왼쪽: 8개의 작업자 GPU를 사용한 직렬 및 분산 교육; 오른쪽: 다양한 학습률 일정을 위한 8개의 GPU를 사용한 분산 교육. 가로 녹색 선은 92%의 대상 모델 정확도를 나타냅니다.

하이퍼파라미터	설명	최고의 가치
λ_0	기본 학습률	0.002
γ	Epoch당 학습률 감쇠	0.97 2
엔	순환 신경망 레이어 수	100
디 _	레이어당 LSTM의 은닉 유닛 수	1000
티	로컬 토큰 수, 역전파가 실행되는 시간 단계	
TRNN	역전파가 실행되는 시간 단계 수	100
RNN 유형	RNN db의 유	LSTM
형	배치 크기	256
손실 함수	손실 함수 dx의 유형	범주 교차 엔트로피 150
	포함된 벡터 차원	Adam 0.8 10
옵티마이저	확률적 최적화 체계	10 500
탈락	탈락 유지 확률	
L2 정규화	모든 레이어의 가중치 정규화	
클립 기준	가중치의 최대 표준	
토큰 빈도 임계값 어휘에 포함하기 위한	말뭉치의 구문 토큰의 최소 빈도	

표 3: 설명 및 성능이 우수한 값과 함께 최적화할 하이퍼파라미터.

이 작업 전반에 걸쳐 하이퍼파라미터 튜닝을 통해 "최상의" 모델이 결정됩니다. 이는 각 방법의 각 하이퍼파라미터 공간에서 임의의 검색을 통해 수행됩니다. 즉, 트레인 세트에서 임의의 하이퍼파라미터로 여러 모델을 트레이닝하고 검증 세트에서 가장 높은 성능을 가진 모델을 선택합니다.

6.3 신경망 아키텍처 조정

신경망의 아키텍처는 하이퍼파라미터입니다. LSTM 외에도 GRU(gated recurrent unit), 시간 데이터에 대한 어텐션 메커니즘이 있는 LSTM[20], LSTM을 따르는 분류 계층의 변형을 고려합니다.

온라인 서비스에 가장 적합한 모델을 선택하는 것은 정확도와 모델 크기 간의 균형입니다. 탭. 그림 4는 디스크의 다양한 신경 모델 아키텍처 및 관련 크기에 대한 검증 수준 정확도를 보여줍니다. 표시된 바와 같이 예측 임베딩을 위해 완전히 연결된 대규모 분류 계층을 제거하면 훈련 가능한 매개변수의 수와 디스크의 모델 크기가 25% 감소합니다. 최고의 상위 5개 정확도는 어텐션 메커니즘을 사용할 때 달성되지만 이 모델은 8% 더 크고 추론 속도가 느립니다. 결과적으로 우리는 배포를 위해 예측 임베딩이 있는 LSTM을 선택했습니다.

모델 아키텍처	Top-5 정확도	모델 크기(양자화 크기), MB
LSTM+완전 연결	0.91	202(51) 152(38)
GRU+예측 임베딩	0.91	152(38) 164(41)
LSTM+예측 임베딩	0.92	
LSTM+주의	0.93	

표 4: 다양한 신경 모델 아키텍처 및 관련 모델 크기에 대한 검증 세트의 정확도.

방법	Top-1 정확도	Top-5 정확도	MRR
알파벳순			0.36 0.372
빈도	0.38		0.47
빈도 if	0.40		0.64
마르코프 체인	0.58		0.67
피티아	0.71		0.83

표 5: Pythia 신경 모델 및 다양한 기준선에 대한 테스트 세트의 정확도 및 평균 역수 순위.

7 평가

7.1 평가 지표

Top-k 정확도와 MRR(Mean Reciprocal Rank) [21] 은 권장 사항의 품질을 측정하는 데 사용됩니다.

처럼:

$$\text{누적}(k) = \frac{\text{Ntop-k}}{\text{큐}}, \tag{10}$$

$$\text{MRR} = \frac{1}{\text{큐}} \cdot \sum_{i=1}^{\text{큐}} \frac{1}{\text{rank}_i}, \tag{11}$$

여기서 Ntop-k는 상위 k 제안의 관련 권장 사항 수를 나타내고 Q는 총 테스트 데이터 샘플 수를 나타내며 rank_i 는 권장 사항의 예측 순위입니다.

상위 1위의 정확도는 상위 권장 사항이 올바른 빈도를 알려주는 반면 상위 5위 정확도 는 상위 5개 권장 사항 목록에 사용자가 찾고 있는 제안이 얼마나 자주 포함되는지에 대한 아이디어를 제공합니다 . MRR 은 결과의 전체 순위를 캡처하여 모델 예측 이 상위 제안 목록에서 얼마나 벗어났는지에 대한 정보를 제공합니다. MRR 값이 1에 가까울수록 성능이 더 좋은 모델에 해당하는 전반적으로 더 작은 추천 순위를 나타냅니다.

7.2 평가 결과

탭. 그림 5는 Pythia 신경 모델과 기본 알파벳 순서, 빈도 모델 및 호출 기반 Markov Chain 모델을 포함한 다양한 간단한 기준선의 성능 비교를 보여줍니다 . 보시다시피, 우리 모델은 특히 상위 1개 제안의 정확도에서 모든 기준선을 훨씬 능가합니다. 탭. 6은 가장 인기 있는 10개의 완성 클래스에 대한 Pythia 시스템 의 성능을 요약합니다 . 그림 7 은 테스트 세트의 Markov Chain 기준선과 비교하여 Pythia 신경 모델 에 의해 달성된 성능 향상을 보여줍니다 . 히스토그램에서 볼 수 있듯이 Pythia는 거의 6000개의 완료 클래스에 대해 50% 이상의 정확도 향상을 제공합니다. Markov Chain 모델은 유형 추론에 의존하므로 Pythia 신경 모델보다 적용 범위가窄습니다. Markov Chain 모델에서 다루지 않는 클래스는 overflow bin에 포함됩니다.

8 모델 전개

Pythia 코드 완성 시스템은 현재 Intellicode 확장의 일부로 배포됩니다. 경량 클라이언트 장치에서 모델을 제공할 수 있 Visual Studio Code IDE의 5 . 는 온라인 시스템을 설계하고 구현하는 데 있어 주요 과제는 예측 대기 시간과 메모리 공간입니다. 신경망 양자화를 8비트 부호 없는 정수 형식으로 적용하여 디스크 크기의 메모리 공간과 모델 크기를 줄입니다.

⁵<https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscointellicode>

등급명	Top-5 정확도, MC Top-5 정확도, Pythia	0.950 0.697 0.989 0.988	$\frac{\Delta Acc}{Acc}$	12567%	Nrecommend Acc 10.0
os	0.863	0.957 0.959 0.533			
numpy	0.575	0.914 0.980 0.754	21.2	113227	
목록	0.978		1.1	79406	
str	0.974		1.4	77507	
os.path	0.895		6.8	75270	
시스템	0.821		16.8	51231	
wx	0.272		95.9	47821	
로깅 시간	0.846		8.01	40871	
텐서플로	0.951		3.1	27874	
	0.511		47.6	16620	

표 6: Pythia 신경 모델 및 Markov Chain 기준에 대한 테스트 세트의 성능 요약. 상대 비율 정확도 변화는 다음과 같이 정의됩니다.

$$\frac{\Delta Acc}{Acc} = 100 \cdot (Acc_{Pythia} - Acc_{MC}) / Acc_{MC} [\%]$$

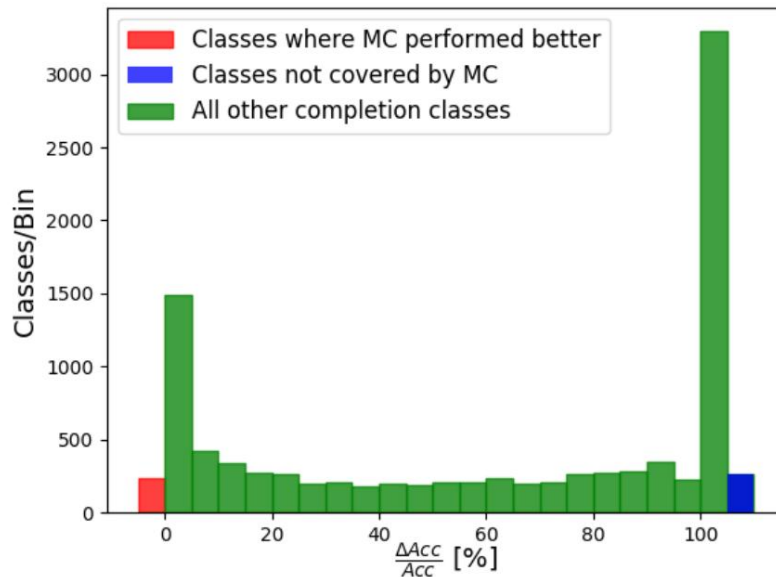


그림 7: 호출 기반 Markov Chain 기준선과 비교하여 Pythia 신경 = $100 \cdot (Acc_{Pythia} - Acc_{MC}) / Acc_{MC} [\%]$ 모델에 대해 주어진 상대 백분율 정확도 차이가 있는 완성 클래스의 수.

8.1 신경망 양자화

신경망 양자화는 가중치를 저장하기 위해 비트 수를 줄이는 과정입니다. Pythia 신경 모델 훈련 중에 사용되는 숫자 형식은 IEEE 754 32비트 부동 소수점입니다. 양자화는 계층별로 수행되어 부동 소수점 형식, 제로 이동 및 스케일링에서 가중치 및 활성화의 최소값과 최대값을 추출합니다. 계층에 대한 가중치 행렬 $W = w_{ij}$ 가 주어지면 양자화된 가중치 행렬 W_q 는 다음과 같이 구합니다.

$$\frac{\text{최대}(W) - \text{최소}(W)}{\beta} \cdot \beta, \quad q_{w_{ij}} = \frac{w_{ij} - \min(W)}{\beta} \cdot \beta. \quad (12)$$

Pythia의 경우 8비트 정수로의 사후 훈련 양자화 결과 모델 크기가 152MB에서 38MB로 4분의 1로 줄어들어 상위 5개 정확도가 92%에서 허용 가능한 89%로 감소합니다.

Pythia 신경 모델이 제공하는 온라인 권장 사항의 예는 그림 8에 나와 있습니다. 표시된 것처럼 Pythia는 가장 가능성이 높은 메서드 호출에 최적화 프로그램이 포함되어야 한다고 정확하게 제안하며 상위 5개 권장 사항 중 2개는 Adam 및 SGD 최적화 프로그램입니다. 다른 제안에는 변수를 저장하고 복원하는 작업을 추가하는 방법인 "Saver"가 포함됩니다.

```

1  import tensorflow as tf
2  # Model parameters
3  W = tf.Variable([.3], tf.float32)
4  b = tf.Variable([-0.3], tf.float32)
5  # Set up linear model
6  x = tf.placeholder(tf.float32)
7  linear_model = W * x + b
8  # target labels
9  y = tf.placeholder(tf.float32)
10 # loss calculation
11 loss = tf.reduce_sum(tf.square(linear_model - y))
12 # optimizer
13 optimizer = tf.train.

```

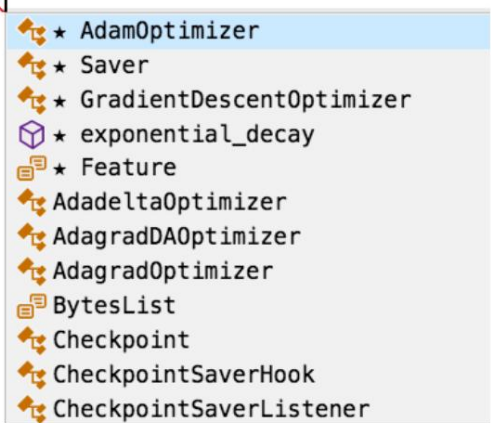


그림 8: Pythia에서 제공하는 예제 코드 완성.

그리고 체크포인트에서 “exponential_decay” – 학습률 감소 제어 및 “Feature” – TensorFlow에 특정한 데이터 래퍼로 세션을 생성하고 교육 작업을 실행하기 전에 모두 호출될 가능성이 높습니다.

9 결론

우리는 소프트웨어 개발자가 사용할 수 있는 메서드 및 API 권장 사항의 순위 목록을 생성하는 Pythia라는 AI 자원 코드 완성을 위한 새로운 종단 간 접근 방식을 도입했습니다. Visual Studio Code IDE에서 Intellicode 확장 의 일부로 시스템을 배포했습니다. Pythia 는 추상 구문 트리에 서 추출한 장거리 코드 컨텍스트에서 훈련된 장단기 메모리 네트워크를 사용하여 멀리 떨어진 노드가 전달하는 의미를 캡처합니다. 실제 소스 코드에 서 추출한 1,580만 개의 메서드 호출로 구성된 대규모 데이터 세트에 대한 평가 결과, 최고의 모델이 상위 5개 정확도를 92% 달성하여 더 단순한 기준선을 능가하는 것으로 나타났습니다.

우리는 고성능 컴퓨팅 클러스터에서 심층 신경망 훈련 및 하이퍼파라미터 조정, 경량 클라이언트 장치에 모델 배포를 훈련하여 편집 시 가장 일치하는 코드 완성을 예측하는 몇 가지 실제적인 문제를 극복하고 문서화했습니다.

Python 외에도 Intellicode 확장 은 Markov Chain 모델을 기반으로 하는 C#, Java, C++ 및 XAML을 포함한 다양한 프로그래밍 언어에 대해 AI 지원 코드 완성을 제공합니다. 앞으로 Python 이외의 프로그래밍 언어에 대한 응용 프로그램과 보다 정교한 코드 완성 시나리오를 위해 고급 딥 러닝 접근 방식을 탐색할 것입니다.

10 승인

신경망 아키텍처 및 학습 가능한 매개 변수 수 감소에 대한 귀중한 토론에 대해 Microsoft Research의 Miltiadis Allamanis에게 감사드립니다. 또한 모델 배포를 도와준 Microsoft AI 프레임워크 및 ML.NET 팀과 원고를 읽어준 Microsoft Research의 Christian Bird에게도 감사드립니다.

참조

- [1] GC Murphy, M. Kersten 및 L. Findlater. Java 소프트웨어 개발자는 eclipse ide를 어떻게 사용하고 있습니까? IEEE, 23(4):76–83, 2006.
- [2] Sebastian Proksch, Johannes Lerch, Mira Mezini. 베이지안 네트워크를 사용한 지능형 코드 완성. 소프트웨어 엔지니어링 및 방법론(TOSEM)에 대한 ACM 트랜잭션, 25(1):3, 2015.
- [3] M. Maraioiu, L. Church 및 A. Blackwell. 전문가의 코드 완성 사용에 대한 실증적 조사 소프트웨어 개발자. 2015.
- [4] D'Souza Andrea Renika, Di Yang, Cristina V Lopes. 파이썬에서 더 스마트한 API 추천을 위한 집단지성. 소스 코드 분석 및 조작(SCAM), 2016 IEEE 16차 국제 작업 회의, 51–60페이지. IEEE, 2016.
- [5] Veselin Raychev, Martin Vechev, Eran Yahav. 통계 언어 모델을 사용한 코드 완성. ACM에서 SIGPLAN 공지, 49권, 419–428페이지. ACM, 2014.
- [6] 마르셀 브루흐, 마틴 몬테루스, 미라 메지니. 코드 완성 시스템을 개선하기 위해 예제를 통해 학습합니다. 소프트웨어 엔지니어링의 기초에 관한 유럽 소프트웨어 엔지니어링 컨퍼런스 및 ACM SIGSOFT 심포지엄의 7차 합동 회의 절차, 213–222페이지. ACM, 2009.
- [7] A. Alnusair, T. Zhao 및 E. Bodden. 효과적인 API 탐색 및 재사용. IRI, 7~12페이지, 2010.
- [8] T. Gvero, V. Kuncak, I. Kuraj 및 R. Piskac. 유형과 무게를 사용하여 완성하십시오. PLDI'13에서. ACM, 2013.
- [9] Felix A Gers, Jürgen Schmidhuber 및 Fred Cummins. 잊는 법 배우기: lstm을 사용한 지속적인 예측. 1999.
- [10] A. Zimdars, DM Chickering 및 C. Meek. 권장 사항을 만들기 위해 시간 데이터를 사용합니다. 절차에서 인공 지능의 불확실성에 관한 7차 회의, 580-588페이지. UAI, 2001.
- [11] B. Mobasher, H. Dai, T. Luo 및 M. Nakagawa. 예측 웹에서 순차 및 비순차 패턴 사용 사용 마이닝 작업. 데이터 마이닝에 관한 국제 회의에서. IEEE, 2002.
- [12] RI Brafman G. Shani 및 D. Heckerman. mdp 기반 추천 시스템. 인공 지능의 불확실성에 관한 제8차 회의 진행. UAI, 2002.
- [13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, Jeff Dean. 단어 및 구문의 분산 표현 및 해당 구성. CJC Burges에서 L. Bottou, M. Welling, Z. Ghahramani 및 KQ Weinberger, 편집자, 신경 정보 처리 시스템의 발전 26, 페이지 3111–3119. Curran 어소시에이츠, Inc., 2013.
- [14] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, Alan Yuille. 다중 모달 순환 신경망(m-rnn)을 사용한 딥 캡션. ICLR, 2015.
- [15] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, Christopher Potts. 감정 트리뱅크에 대한 의미론적 구성성을 위한 재귀 심층 모델. 자연어 처리의 경험적 방법에 관한 2013년 회의 절차, 1631–1642페이지, 펜실베이니아주 Stroudsburg, 2013년 10월. 전산 언어학 협회.
- [16] Ilya Sutskever, Oriol Vinyals 및 Quoc V Le. 신경망을 사용하여 시퀀스 학습을 시퀀싱합니다. Z. Ghahramani, M. Welling, C. Cortes, ND Lawrence 및 KQ Weinberger, 편집자, Advances in Neural Information Processing Systems 27, 페이지 3104–3112. Curran 어소시에이츠, Inc., 2014.
- [17] Hakan Inan, Khashayar Khosravi, Richard Socher. 단어 벡터와 단어 분류기 묶기: 언어 모델링을 위한 손실 프레임워크. CoRR, abs/1611.01462, 2016.
- [18] 알렉스 그레이브스. 순환 신경망으로 시퀀스 생성. CoRR, abs/1308.0850, 2013.
- [19] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia 및 Kaiming He. 정확하고 큰 미니배치 SGD: 1시간 안에 이미지넷 훈련. CoRR, abs/1706.02677, 2017.

- [20] Colin Raffel과 Daniel PW Ellis. 주의력이 있는 피드포워드 네트워크는 일부 장기 메모리를 해결할 수 있습니다. 문제. CoRR, abs/1512.08756, 2015.
- [21] Dragomir R. Radev, Hong Qi, Harris Wu, Weiguo Fan. 웹 기반 질문 응답 시스템 평가. 언어 자원 및 평가, 2002.