Materiały dydaktyczne EITC/AI/TFF TensorFlow Fundamentals

Zawartość

Lekcja: Wprowadzenie do TensorFlow 2

Temat: Podstawy uczenia maszynowego 2

Temat: Podstawy widzenia komputerowego z ML 4

Temat: Wprowadzenie do sieci neuronowych splotowych 6

Temat: Budowanie klasyfikatora obrazów 7

Lekcja: Uczenie się struktur neuronowych z TensorFlow 8

Temat: Omówienie struktury uczenia się opartego na strukturze neuronowej 8

Temat: Szkolenie z grafami naturalnymi 8

Temat: Szkolenie z grafami syntezowanymi 9

Temat: Uczenie się adwersarskie w celu klasyfikacji obrazów 10

Lekcja: Przetwarzanie języka naturalnego z TensorFlow 10

Temat: Tokenizacja 10

Temat: Sekwencjonowanie – przekształcanie zdań w dane 13

Temat: Szkolenie modelu w celu rozpoznawania sentymentów w tekście 15

Temat: ML z rekurencyjnymi sieciami neuronowymi 16

Temat: Pamięć długotrwała i krótkotrwała dla przetwarzania języka naturalnego 16

Temat: Szkolenie AI w zakresie tworzenia poezji 17

Lekcja: Programowanie TensorFlow 18

Temat: Wprowadzenie do kodowania TensorFlow 18

Temat: Wprowadzenie do TensorFlow Lite 20

Lekcja: Wprowadzenie do TensorFlow

Temat: Podstawy uczenia maszynowego

TensorFlow to potężne środowisko, które umożliwia programistom wydajne tworzenie i trenowanie modeli uczenia maszynowego. Zapewnia elastyczną i skalowalną architekturę, która może być używana w szerokim zakresie aplikacji. TensorFlow opiera się na koncepcji tensorów, które są wielowymiarowymi tablicami. Te tensory przepływają przez graf obliczeniowy, gdzie wykonywane są na nich operacje matematyczne w celu wygenerowania pożądanego wyniku. To podejście oparte na grafie umożliwia wydajne równoległe wykonywanie zarówno na procesorach CPU, jak i GPU, dzięki czemu nadaje się do zadań uczenia maszynowego na dużą skalę.

Aby lepiej zrozumieć TensorFlow, rozważmy podstawy uczenia maszynowego. **Uczenie maszynowe to poddziedzina AI, która koncentruje się na rozwijaniu algorytmów, które mogą uczyć się z danych i formułować prognozy lub podejmować decyzje bez wyraźnego programowania. Polega na trenowaniu modelu na zestawie danych, który składa się z cech wejściowych i odpowiadających im etykiet wyjściowych. Model uczy się wzorców i relacji w danych i wykorzystuje tę wiedzę do formułowania prognoz na podstawie nowych, niewidzianych danych.**

TensorFlow zapewnia kompleksowy zestaw narzędzi i interfejsów API do implementacji algorytmów uczenia maszynowego. Obsługuje szeroki zakres architektur sieci neuronowych, w tym splotowe sieci neuronowe (CNN) do rozpoznawania obrazów, rekurencyjne sieci neuronowe (RNN) do danych sekwencyjnych i generatywne sieci przeciwstawne (GAN) do generowania nowych próbek danych. TensorFlow oferuje również interfejsy API wysokiego poziomu, takie jak Keras, które upraszczają proces budowania i trenowania modeli głębokiego uczenia.

Jedną z kluczowych zalet TensorFlow jest jego zdolność do efektywnego wykorzystania akceleratorów sprzętowych, takich jak GPU, w celu przyspieszenia procesu szkolenia i wnioskowania. GPU to wysoce równoległe procesory, które doskonale radzą sobie z wykonywaniem operacji macierzowych, które są podstawą wielu algorytmów uczenia maszynowego. TensorFlow automatycznie wykrywa i wykorzystuje dostępne GPU, umożliwiając szybsze szkolenie i wnioskowanie modeli.

Uczenie maszynowe polega na trenowaniu komputera, aby rozpoznawał wzorce w danych. Pokazując komputerowi mnóstwo zdjęć kamieni, papieru i nożyczek, możemy nauczyć go identyfikować te obiekty, znajdując pasujące do nich wzorce. Ten proces trenowania komputera, aby rozpoznawał wzorce, jest istotą uczenia maszynowego.

Zanim zagłębimy się w złożone zadania, takie jak rozpoznawanie kamienia, papieru i nożyc, zacznijmy od prostszego przykładu. Rozważmy zbiór liczb z relacją między wartościami X i Y. Obserwując wzór, możemy wywnioskować relację jako Y = 2X – 1. Ta zasada rozpoznawania wzorców stanowi podstawę wszelkiego uczenia maszynowego.

Aby zilustrować tę koncepcję, przedstawiamy fragment kodu, który tworzy model uczenia maszynowego do dopasowywania tych liczb. Kod definiuje model sieci neuronowej przy użyciu biblioteki Keras firmy TensorFlow. Model ten składa się z pojedynczej warstwy z pojedynczym neuronem. Dane wejściowe do sieci neuronowej to wartość X, a sieć przewiduje odpowiadającą jej wartość Y. Następnie model jest kompilowany przy użyciu funkcji straty i optymalizatora, które są ważnymi składnikami uczenia maszynowego.

Podczas treningu model dokonuje wstępnych przypuszczeń dotyczących relacji między liczbami, np. Y = 5X + 5. Funkcja straty kwantyfikuje dokładność tych przypuszczeń, umożliwiając modelowi uczenie się na błędach i ulepszanie przewidywań w miarę upływu czasu.

Ten przykład pokazuje podstawy uczenia maszynowego przy użyciu TensorFlow. Poprzez trenowanie modeli w celu rozpoznawania wzorców w danych możemy umożliwić komputerom wykonywanie zadań, które naśladują ludzką inteligencję. W miarę postępów w nauce będziesz odkrywać bardziej zaawansowane koncepcje i zastosowania uczenia maszynowego.

W kontekście uczenia maszynowego TensorFlow jest potężną biblioteką typu open source, która pozwala nam budować i trenować sztuczne sieci neuronowe. Jedną z podstawowych koncepcji w TensorFlow jest proces optymalizacji, który

obejmuje iteracyjne dostosowywanie parametrów modelu w celu zminimalizowania różnicy między jego przewidywaniami a rzeczywistymi danymi.

Aby zilustrować tę koncepcję, rozważmy prosty przykład. Wyobraźmy sobie, że mamy zbiór punktów danych, z których każdy składa się z wartości wejściowej (X) i wartości wyjściowej (Y). Naszym celem jest znalezienie wzoru matematycznego, który może dokładnie przewidzieć wartość wyjściową dla dowolnej danej wartości wejściowej. Innymi słowy, chcemy znaleźć funkcję, która mapuje X na Y.

Aby to osiągnąć, możemy utworzyć model sieci neuronowej przy użyciu TensorFlow. Początkowo model będzie losowo zgadywał formułę. Następnie użyje funkcji optymalizatora, aby wygenerować nową zgadywankę na podstawie różnicy między przewidywaniami modelu a rzeczywistymi danymi. Powtarzając ten proces wielokrotnie, model stopniowo poprawia swoją zgadywankę, aż osiągnie dokładniejszą formułę.

W naszym przykładzie dane są reprezentowane jako tablica X i Y. Proces dopasowywania wartości wejściowych i wyjściowych jest wykonywany przez metodę „fit" modelu. Wywołując tę metodę i określając liczbę iteracji (w tym przypadku 500), trenujemy model, aby znaleźć najlepszą formułę pasującą do podanych danych.

Po wytrenowaniu modelu możemy go użyć do przewidywania wartości wyjściowej (Y) dla danej wartości wejściowej (X). Należy jednak pamiętać, że przewidywania modelu nie zawsze mogą być dokładne. W naszym przykładzie, jeśli spróbujemy przewidzieć wartość wyjściową dla X=10, możemy oczekiwać odpowiedzi 19. Jednak ze względu na ograniczone dane treningowe, przewidywanie modelu może być nieznacznie inne, na przykład 18,9998. Ta rozbieżność występuje, ponieważ model został wytrenowany tylko na niewielkiej liczbie punktów danych, a jego zdolność do generalizowania na nowe wartości jest ograniczona.

W uczeniu maszynowym często zdarzają się sytuacje, w których przewidywania modelu nie są dokładne, lecz raczej bliskie przybliżenia. Dzieje się tak, ponieważ model dokonuje przewidywań probabilistycznych w oparciu o dostępne dane treningowe. Chociaż istnieje duże prawdopodobieństwo, że relacja między X i Y jest linią prostą, nie możemy być tego pewni. Dlatego przewidywanie modelu odzwierciedla tę niepewność, podając wartość bardzo bliską oczekiwanemu wynikowi.

Aby dalej eksplorować i eksperymentować z tą koncepcją, możesz spróbować uruchomić dostarczony kod, korzystając z łącza podanego w opisie. Pozwoli ci to zobaczyć z pierwszej ręki, jak prognozy modelu różnią się dla różnych wartości wejściowych.

TensorFlow zapewnia potężne ramy do budowania i trenowania modeli uczenia maszynowego. Poprzez iteracyjną optymalizację parametrów modelu możemy ulepszyć jego przewidywania i znaleźć najlepszą formułę, która pasuje do danych. Chociaż przewidywania modelu mogą nie zawsze być dokładne, są one bliskimi przybliżeniami, które uwzględniają inherentną niepewność danych.

Tradycyjne programowanie, znane również jako programowanie oparte na regułach, obejmuje wyraźne definiowanie reguł i instrukcji, których ma przestrzegać program komputerowy. Reguły te są zazwyczaj tworzone przez programistów, którzy mają głębokie zrozumienie dziedziny problemu. Programista analizuje problem, identyfikuje niezbędne reguły i pisze kod zgodnie z nimi. Kod określa, w jaki sposób dane wejściowe powinny być przetwarzane i jakie dane wyjściowe powinny być generowane na podstawie wstępnie zdefiniowanych warunków i logicznych stwierdzeń.

Na przykład rozważmy prosty problem klasyfikowania wiadomości e-mail jako spam lub nie-spam. W tradycyjnym programowaniu programista może zdefiniować reguły, takie jak sprawdzanie określonych słów kluczowych, analizowanie adresu nadawcy i ocenianie zawartości wiadomości e-mail. Następnie program zastosuje te reguły do przychodzących wiadomości e-mail, aby określić ich klasyfikację.

Z drugiej strony uczenie maszynowe przyjmuje inne podejście. Zamiast jawnie definiować reguły, algorytmy uczenia maszynowego uczą się z danych, aby automatycznie odkrywać wzorce i formułować przewidywania lub podejmować decyzje. To podejście jest szczególnie przydatne w przypadku złożonych problemów, w których trudno jest jawnie zdefiniować reguły lub gdy reguły mogą się zmieniać w czasie.

W uczeniu maszynowym model jest trenowany przy użyciu dużej ilości oznaczonych danych. Model uczy się rozpoznawać wzorce i relacje w danych, co pozwala mu na dokonywanie przewidywań lub podejmowanie decyzji na

podstawie nowych, niewidzianych danych. Proces trenowania modelu uczenia maszynowego obejmuje iteracyjne dostosowywanie jego parametrów wewnętrznych w celu zminimalizowania różnicy między przewidywanymi wynikami a rzeczywistymi etykietami w danych treningowych.

Kontynuując przykład klasyfikacji wiadomości e-mail, w uczeniu maszynowym dostarczylibyśmy algorytmowi duży zbiór danych oznaczonych wiadomości e-mail. Następnie algorytm nauczyłby się identyfikować wzorce w danych, które odróżniają wiadomości spam od wiadomości niebędących spamem. Po wytrenowaniu modelu można go używać do klasyfikowania nowych wiadomości e-mail bez wyraźnego definiowania reguł.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

# Definicja modelu
model = keras.Sequential([
    keras.Input(shape=(1,)), layers.Dense(units=1) ])

# Kompilacja modelu
model.compile(optimizer='sgd', loss='mean_squared_error')

# Dane wejściowe i wyjściowe
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

# Trenowanie modelu
model.fit(xs, ys, epochs=500)
```

Temat: Podstawy widzenia komputerowego z ML

Sztuczna inteligencja (AI) zrewolucjonizowała różne branże, w tym widzenie komputerowe. TensorFlow, framework uczenia maszynowego (ML) typu open source, stał się popularnym narzędziem do budowania modeli AI. W tym materiale dydaktycznym przyjrzymy się podstawom TensorFlow i jego zastosowaniu w podstawowych zadaniach widzenia komputerowego.

TensorFlow to potężna biblioteka ML opracowana przez Google Brain. Zapewnia elastyczną i wydajną platformę do wdrażania i wdrażania modeli ML. TensorFlow umożliwia użytkownikom definiowanie i trenowanie sieci neuronowych, co czyni ją szczególnie przydatną do zadań związanych z komputerowym widzeniem.

Wizja komputerowa obejmuje ekstrakcję, analizę i zrozumienie informacji wizualnych z obrazów lub filmów. TensorFlow upraszcza ten proces, zapewniając wstępnie zbudowane funkcje i narzędzia do przetwarzania i analizy obrazu. Dzięki temu programiści mogą skupić się na architekturze modelu i szkoleniu, a nie na niskopoziomowej manipulacji obrazem.

Aby rozpocząć pracę z TensorFlow, konieczne jest zrozumienie jego podstawowych komponentów. **Podstawowym elementem TensorFlow jest tensor, który reprezentuje wielowymiarowe tablice danych. Tensory mogą być skalarami, wektorami, macierzami lub tablicami o wyższym wymiarze. Operacje TensorFlow, znane jako ops, manipulują tymi tensorami w celu wykonywania obliczeń.**

TensorFlow wprowadza również koncepcję grafu obliczeniowego. Graf obliczeniowy to seria operacji TensorFlow ułożonych w strukturę podobną do grafu. Każdy węzeł w grafie reprezentuje operację, podczas gdy krawędzie reprezentują przepływ danych między operacjami. To podejście oparte na grafie pozwala TensorFlow na efektywne rozprowadzanie obliczeń na wiele urządzeń, takich jak procesory CPU lub GPU.

Aby zilustrować podstawowe koncepcje TensorFlow, rozważmy proste zadanie z zakresu widzenia komputerowego: klasyfikację obrazów. W klasyfikacji obrazów celem jest przypisanie etykiety do obrazu wejściowego z zestawu wstępnie

zdefiniowanych kategorii. TensorFlow udostępnia interfejs API wysokiego poziomu o nazwie Keras, który upraszcza proces budowania i trenowania sieci neuronowych.

Najpierw musimy przygotować nasze dane. Wiąże się to z załadowaniem i wstępnym przetworzeniem obrazów. TensorFlow udostępnia funkcje do odczytu i dekodowania plików obrazów, a także narzędzia do zmiany rozmiaru, normalizacji i rozszerzania danych. Wstępne przetwarzanie jest ważne, aby upewnić się, że obrazy wejściowe są w odpowiednim formacie do szkolenia.

Następnie definiujemy architekturę naszej sieci neuronowej. Wiąże się to z wyborem odpowiednich warstw i konfiguracją ich parametrów. TensorFlow oferuje szeroki zakres wstępnie zbudowanych warstw, takich jak warstwy splotowe, pulowe i w pełni połączone, które są powszechnie stosowane w modelach wizji komputerowej. Warstwy te można układać w stosy, aby utworzyć model sekwencyjny.

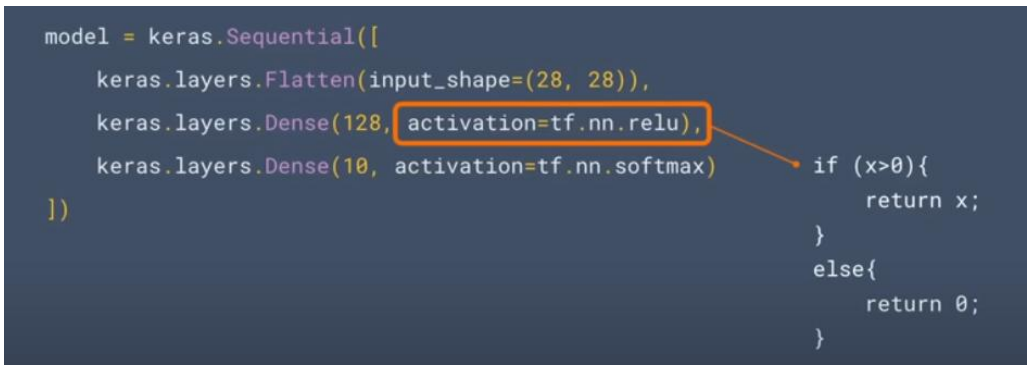
Po zdefiniowaniu architektury modelu kompilujemy model, określając funkcję straty, optymalizator i metryki oceny. Funkcja straty kwantyfikuje różnicę między przewidywanymi i prawdziwymi etykietami, kierując model w stronę lepszych prognoz. Optymalizator aktualizuje parametry modelu na podstawie obliczonych gradientów, optymalizując funkcję straty. Metryki oceny zapewniają dodatkowe miary wydajności, takie jak dokładność lub precyzja.

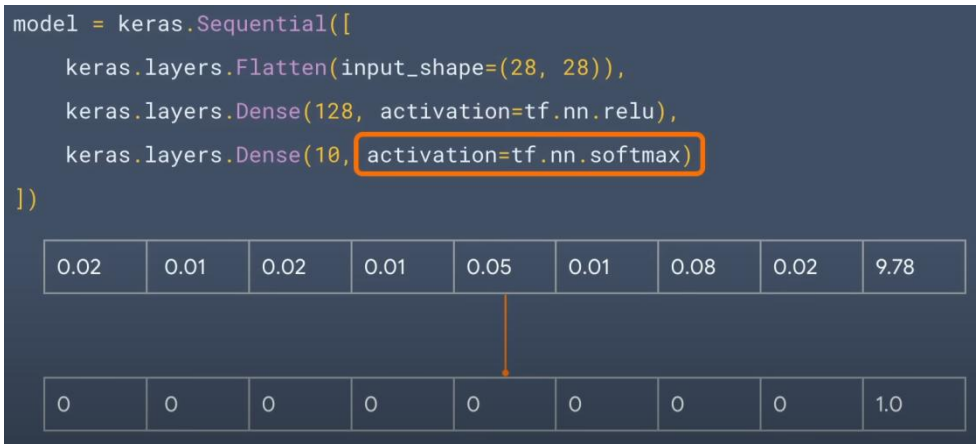
Po skompilowaniu modelu możemy go teraz trenować, używając naszego oznaczonego zestawu danych. TensorFlow udostępnia funkcje do iterowania danych treningowych w partiach, umożliwiając wydajne przetwarzanie dużych zestawów danych. Podczas treningu model uczy się dostosowywać swoje parametry, aby zminimalizować funkcję straty, stopniowo ulepszając swoje przewidywania.

Po szkoleniu możemy ocenić wydajność naszego modelu na oddzielnym zestawie danych walidacyjnych lub testowych. TensorFlow udostępnia funkcje do obliczania różnych metryk, takich jak dokładność, precyzja, odwołanie i wynik F1. Te metryki dostarczają wglądu w wydajność modelu i pomagają zidentyfikować obszary do poprawy.

Gdy jesteśmy zadowoleni z wydajności modelu, możemy użyć go do tworzenia prognoz na nowych, niewidzianych obrazach. TensorFlow udostępnia funkcje do ładowania i wstępnego przetwarzania nowych obrazów, a także narzędzia do interpretowania prognoz modelu. Pozwala nam to wdrażać nasz model w rzeczywistych aplikacjach, takich jak rozpoznawanie obiektów lub autonomiczne systemy napędowe.

TensorFlow to potężne środowisko do implementacji modeli widzenia komputerowego. Jego elastyczna i wydajna konstrukcja, wraz z API wysokiego poziomu Keras, upraszcza proces budowania i trenowania sieci neuronowych. Wykorzystując możliwości TensorFlow, programiści mogą zająć się szerokim zakresem zadań widzenia komputerowego, od klasyfikacji obrazów po wykrywanie i segmentację obiektów.





Temat: Wprowadzenie do sieci neuronowych splotowych

Sztuczna inteligencja – Podstawy TensorFlow – Wprowadzenie do TensorFlow – Wprowadzenie do sieci neuronowych splotowych

Sieci neuronowe splotowe (CNN) to rodzaj algorytmu głębokiego uczenia szeroko stosowanego w dziedzinie widzenia komputerowego i rozpoznawania obrazów. Są one specjalnie zaprojektowane do przetwarzania i analizowania danych wizualnych, co czyni je wysoce skutecznymi w takich zadaniach, jak wykrywanie obiektów, klasyfikacja obrazów i rozpoznawanie twarzy. W tej sekcji przedstawimy koncepcję sieci neuronowych CNN i zbadamy, w jaki sposób są one implementowane przy użyciu TensorFlow, popularnego frameworka uczenia maszynowego typu open source.

W swojej istocie CNN składa się z wielu warstw, które wykonują różne operacje na danych wejściowych. Kluczowym komponentem CNN jest warstwa splotowa, która stosuje zestaw filtrów do obrazu wejściowego w celu wyodrębnienia znaczących cech. Te filtry to małe macierze, które są splecione z obrazem wejściowym, tworząc mapę cech, która wyróżnia określone wzorce lub struktury obecne w obrazie.

W TensorFlow warstwa splotowa jest implementowana przy użyciu klasy tf.keras.layers.Conv2D. Ta klasa umożliwia określenie liczby filtrów, rozmiaru filtra i innych parametrów. Każdy filtr w warstwie splotowej uczy się wykrywać określoną cechę, dostosowując swoje wagi podczas procesu szkolenia. Poprzez układanie wielu warstw splotowych razem, sieci neuronowe mogą uczyć się hierarchicznych reprezentacji danych wizualnych, przechwytując zarówno cechy niskiego poziomu, takie jak krawędzie, jak i cechy wysokiego poziomu, takie jak kształty i obiekty.

Po warstwach splotowych sieci CNN zazwyczaj obejmują warstwy pulowania, aby zmniejszyć wymiary przestrzenne map cech. Pulowanie pomaga wyodrębnić najważniejsze cechy, jednocześnie odrzucając nieistotne szczegóły, dzięki czemu sieć jest bardziej odporna na zmiany w danych wejściowych. Najczęstszą operacją pulowania jest maksymalne pulowanie, które pobiera maksymalną wartość w małym oknie i zmniejsza próbkowanie mapy cech.

W TensorFlow warstwę pooling można dodać za pomocą klasy tf.keras.layers.MaxPooling2D. Można określić rozmiar poolingu i kroki, które odpowiednio określają rozmiar okna poolingu i ilość downsamplingu. Poprzez wielokrotne stosowanie warstw splotowych i poolingu, sieci neuronowe mogą stopniowo uczyć się bardziej złożonych cech i abstrakcyjnych reprezentacji, co prowadzi do większej dokładności zadań wizualnych.

Po zakończeniu ekstrakcji cech, dane wyjściowe ostatniej warstwy splotowej lub puli są spłaszczane do wektora 1D. Następnie wektor ten jest przepuszczany przez jedną lub więcej w pełni połączonych warstw, znanych również jako gęste warstwy, aby wykonać ostateczną klasyfikację lub regresję. Gęste warstwy łączą każdy neuron w jednej warstwie z każdym neuronem w następnej warstwie, umożliwiając bardziej złożone relacje między cechami.

W TensorFlow gęsta warstwa jest implementowana przy użyciu klasy tf.keras.layers.Dense. Możesz określić liczbę neuronów i funkcję aktywacji dla gęstej warstwy. Funkcja aktywacji wprowadza nieliniowość do sieci, umożliwiając jej naukę złożonych granic decyzyjnych. Typowe wybory funkcji aktywacji obejmują ReLU (Rectified Linear Unit) i sigmoid.

Aby wytrenować model CNN w TensorFlow, potrzebujesz oznaczonego zestawu danych obrazów. Model jest trenowany przez iteracyjne dostosowywanie wag filtrów i neuronów w celu zminimalizowania różnicy między przewidywanym wynikiem a prawdziwymi etykietami. Ten proces, znany jako propagacja wsteczna, wykorzystuje algorytm optymalizacji, taki jak stochastyczny gradient zejścia (SGD), aby zaktualizować wagi na podstawie gradientu funkcji straty.

Sieci neuronowe splotowe są potężnym narzędziem do analizy danych wizualnych. Wykorzystując hierarchiczną strukturę CNN, TensorFlow umożliwia wydajną implementację i szkolenie tych sieci. Zrozumienie podstaw CNN i ich implementacji w TensorFlow jest ważne dla każdego, kto pracuje w dziedzinie widzenia komputerowego i rozpoznawania obrazów.

Temat: Budowanie klasyfikatora obrazów

Na początek zapoznajmy się z koncepcją tensorów, które są podstawowymi elementami składowymi TensorFlow. **Tensor można postrzegać jako wielowymiarową tablicę lub obiekt matematyczny, który uogólnia skalary, wektory i macierze. Reprezentuje on dane przepływające przez graf obliczeniowy. TensorFlow działa na tensorach, wykonując operacje takie jak dodawanie, mnożenie i bardziej złożone obliczenia.**

Jedną z kluczowych cech TensorFlow jest możliwość automatycznego obliczania gradientów. Jest to ważne w przypadku trenowania modeli uczenia maszynowego przy użyciu technik takich jak propagacja wsteczna. Automatyczne różnicowanie TensorFlow pozwala programistom skupić się na budowaniu i optymalizacji modeli bez martwienia się o zawiłości ręcznego obliczania gradientów.

Aby zbudować klasyfikator obrazów przy użyciu TensorFlow, musimy zrozumieć proces trenowania modelu. Pierwszym krokiem jest zebranie oznaczonego zestawu danych obrazów, gdzie każdy obraz jest powiązany z określoną klasą lub kategorią. Ten zestaw danych zostanie użyty do trenowania modelu w celu rozpoznawania i klasyfikowania nowych obrazów.

Następnie wstępnie przetwarzamy obrazy, aby upewnić się, że mają odpowiedni format do treningu. Może to obejmować zmianę rozmiaru obrazów, normalizację wartości pikseli i rozszerzenie zestawu danych poprzez zastosowanie transformacji, takich jak obroty lub odwrócenia. Wstępne przetwarzanie pomaga poprawić zdolność modelu do generalizowania i dobrego działania na niewidzianych danych.

Po przygotowaniu zestawu danych możemy zdefiniować architekturę naszego klasyfikatora obrazów za pomocą API wysokiego poziomu TensorFlow, Keras. Keras upraszcza proces budowania sieci neuronowych, zapewniając przyjazny dla użytkownika interfejs do definiowania warstw, funkcji aktywacji i algorytmów optymalizacji.

Architektura klasyfikatora obrazu zazwyczaj składa się z wielu warstw, w tym warstw splotowych, warstw grupujących i warstw w pełni połączonych. Warstwy splotowe wyodrębniają cechy z obrazów wejściowych, podczas gdy warstwy grupujące redukują wymiarowość wyodrębnionych cech. Warstwy w pełni połączone, znane również jako warstwy gęste, wykonują ostateczną klasyfikację na podstawie wyodrębnionych cech.

Po zdefiniowaniu architektury kompilujemy model, określając funkcję straty, optymalizator i metryki oceny. Funkcja straty kwantyfikuje rozbieżność między przewidywanymi i rzeczywistymi etykietami, podczas gdy optymalizator aktualizuje parametry modelu, aby zminimalizować tę rozbieżność. Metryki oceny, takie jak dokładność, służą do pomiaru wydajności modelu podczas treningu i oceny.

Po skompilowaniu modelu możemy rozpocząć proces szkolenia. Obejmuje on wprowadzanie oznaczonych obrazów z zestawu danych do modelu i iteracyjnie dostosowywanie parametrów modelu w celu zminimalizowania strat. TensorFlow udostępnia różne techniki szkoleniowe, takie jak stochastyczny gradient zejścia (SGD) i adaptacyjne algorytmy optymalizacji, takie jak Adam, w celu wydajnej aktualizacji parametrów modelu.

Podczas treningu istotne jest monitorowanie wydajności modelu na oddzielnym zestawie danych walidacyjnych, aby uniknąć nadmiernego dopasowania. Nadmierne dopasowanie występuje, gdy model dobrze radzi sobie z danymi treningowymi, ale nie uogólnia się na nowe, niewidziane dane. **Techniki regularyzacji, takie jak dropout i weight decay, można stosować w celu złagodzenia nadmiernego dopasowania i poprawy zdolności modelu do generalizacji.**

Po wytrenowaniu modelu możemy ocenić jego wydajność na oddzielnym zestawie danych testowych. Ten zestaw danych powinien być odrębny od zestawów danych treningowych i walidacyjnych, aby zapewnić bezstronną ocenę dokładności modelu. Ocena wydajności modelu na niewidzianych danych pomaga nam ocenić jego zdolność do uogólniania i formułowania prognoz na podstawie obrazów ze świata rzeczywistego.

Lekcja: Uczenie się struktur neuronowych z TensorFlow

Temat: Omówienie struktury uczenia się opartego na strukturze neuronowej

Neural Structured Learning (NSL) to rozszerzenie TensorFlow, które umożliwia integrację danych o strukturze grafu z procesem szkolenia. Grafy to potężny sposób na reprezentowanie relacji między jednostkami, a NSL wykorzystuje to, włączając techniki regularyzacji oparte na grafach. Ta struktura jest szczególnie przydatna w scenariuszach, w których dane zawierają wrodzone struktury grafu, takie jak sieci społecznościowe, sieci cytowań lub sieci biologiczne.

Głównym celem NSL jest poprawa generalizacji i solidności modeli uczenia maszynowego poprzez wykorzystanie informacji strukturalnych obecnych w danych. Poprzez włączenie regularizacji grafu, NSL zachęca model do uczenia się wzorców nie tylko z cech wejściowych, ale także z relacji między encjami na grafie. Pomaga to modelowi lepiej generalizować, zwłaszcza w przypadku danych rozproszonych lub zaszumionych.

NSL zapewnia kilka kluczowych komponentów, które ułatwiają integrację danych o strukturze grafu z procesem szkoleniowym. Komponenty te obejmują:

1. Regularyzacja grafu: NSL wprowadza termin regularyzacji grafu do funkcji straty, co zachęca model do uczenia się ze struktury grafu. Ten termin regularyzacji karze odchylenia od oczekiwanego zachowania na podstawie relacji grafu.

2. Konstrukcja grafu: NSL zapewnia mechanizm do konstruowania grafów z danych wejściowych. Można to zrobić na podstawie wstępnie zdefiniowanych reguł lub przy użyciu algorytmu konstrukcji grafu, który automatycznie wnioskuje relacje między encjami.

3. Trening przeciwstawny: NSL obejmuje proces treningu przeciwstawnego, którego celem jest uczynienie modelu odpornym na ataki przeciwstawne. Poprzez dodanie zakłóceń do danych wejściowych NSL zmusza model do nauki bardziej odpornych reprezentacji.

4. Uczenie się grafów neuronowych: NSL wprowadza koncepcję uczenia się grafów neuronowych, w której model uczy się przewidywać samą strukturę grafu. Dzięki temu model może skuteczniej uchwycić relacje między encjami.

Aby użyć NSL z TensorFlow, programiści muszą wykonać kilka kroków. Najpierw muszą skonstruować graf z danych wejściowych, korzystając z dostarczonych mechanizmów konstrukcji grafu. Następnie włączają termin regularizacji grafu do funkcji straty podczas treningu. Na koniec mogą zastosować techniki treningu adwersarskiego i uczenia się grafów neuronowych, aby jeszcze bardziej zwiększyć wydajność modelu.

Temat: Szkolenie z grafami naturalnymi

W kontekście AI graf jest reprezentacją matematyczną składającą się z węzłów i krawędzi. Węzły reprezentują jednostki, podczas gdy krawędzie reprezentują relacje między tymi jednostkami. Neural Structured Learning with TensorFlow wykorzystuje tę strukturę grafu, aby poprawić wydajność modeli AI. Poprzez włączenie informacji o grafie do procesu szkolenia modele mogą uczyć się zarówno z danych oznaczonych, jak i nieoznaczonych, co prowadzi do lepszej generalizacji i zwiększonej dokładności.

Szkolenie z naturalnymi grafami polega na użyciu rzeczywistych danych, które naturalnie tworzą strukturę grafu. Może to obejmować dane z sieci społecznościowych, sieci cytowań lub dowolnej innej domeny, w której jednostki są połączone za pomocą relacji. Celem jest wykorzystanie wewnętrznej struktury danych w celu usprawnienia procesu uczenia się.

Po przygotowaniu danych następnym krokiem jest zdefiniowanie architektury modelu. Neural Structured Learning umożliwia integrację technik regularizacji grafu z modelem. Regularizacja grafu zachęca model do uczenia się płynnych przewidywań w połączonych węzłach, promując spójność przewidywań dla podobnych jednostek. Ta regularyzacja może pomóc zapobiec nadmiernemu dopasowaniu i poprawić możliwości generalizacji modelu.

Podczas procesu szkolenia model jest optymalizowany przy użyciu funkcji straty, która mierzy rozbieżność między przewidywanymi wynikami a etykietami prawdy podstawowej. W przypadku grafów naturalnych można uwzględnić dodatkowe terminy straty, aby zachęcić model do uczenia się ze struktury grafu. Na przykład termin straty może karać prognozy, które są niezgodne z prognozami sąsiednich węzłów w grafie.

Aby skutecznie trenować model, TensorFlow udostępnia różne algorytmy optymalizacji, takie jak stochastyczny gradient zstępujący (SGD) lub Adam. Algorytmy te iteracyjnie aktualizują parametry modelu na podstawie gradientów funkcji straty. Poprzez dostosowanie szybkości uczenia się i innych hiperparametrów, programiści mogą dostroić proces trenowania, aby osiągnąć optymalną wydajność.

Po wytrenowaniu modelu można go ocenić na oddzielnym zestawie testowym, aby ocenić jego wydajność. **Metryki takie jak dokładność, precyzja, odwołanie i wynik F1 można wykorzystać do pomiaru skuteczności modelu w tworzeniu przewidywań. Wykorzystując strukturę grafu, Neural Structured Learning ma na celu poprawę tych metryk wydajności w porównaniu z tradycyjnymi podejściami szkoleniowymi**.

Neural Structured Learning with TensorFlow umożliwia trenowanie modeli AI przy użyciu naturalnych grafów. Poprzez włączenie struktury grafu do procesu trenowania modele mogą uczyć się zarówno z oznaczonych, jak i nieoznaczonych danych, co prowadzi do poprawy dokładności i generalizacji. To podejście jest szczególnie przydatne w domenach, w których dane naturalnie tworzą strukturę grafu, takich jak sieci społecznościowe lub sieci cytowań.

Temat: Szkolenie z grafami syntezowanymi

Neural Structured Learning (NSL) to biblioteka TensorFlow, która umożliwia trenowanie sieci neuronowych przy użyciu sygnałów strukturalnych. Umożliwia ona włączenie danych o strukturze grafu do procesu trenowania, co może być szczególnie przydatne w scenariuszach, w których instancje danych są ze sobą połączone. NSL wykorzystuje moc sieci neuronowych grafów, aby uczyć się z tych sygnałów strukturalnych i poprawiać wydajność modeli.

Jedną z kluczowych cech NSL jest możliwość trenowania modeli przy użyciu syntetyzowanych grafów. Syntetyzowane grafy są tworzone przez rozszerzenie oryginalnych danych o dodatkowe krawędzie lub połączenia grafów. Połączenia te mogą być oparte na różnych czynnikach, takich jak podobieństwo między wystąpieniami danych, wiedza o domenie lub inne istotne kryteria. Poprzez włączenie tych syntetyzowanych grafów do procesu trenowania, modele NSL mogą uczyć się zarówno z oryginalnych danych, jak i rozszerzonej struktury grafów, co prowadzi do poprawy wydajności i generalizacji.

Proces szkolenia z syntetyzowanymi grafami przy użyciu NSL obejmuje kilka kroków. Najpierw oryginalne dane są wstępnie przetwarzane w celu utworzenia reprezentacji grafu. Można to zrobić, definiując węzły i krawędzie grafu na podstawie wystąpień danych i ich relacji. Po utworzeniu reprezentacji grafu dodatkowe krawędzie są syntetyzowane na podstawie żądanych kryteriów. Te syntetyzowane krawędzie można dodawać do grafu w sposób kontrolowany w celu rozszerzenia oryginalnej struktury.

Po przygotowaniu reprezentacji grafu następnym krokiem jest zdefiniowanie architektury sieci neuronowej. Obejmuje to określenie warstw, funkcji aktywacji i innych parametrów modelu. NSL udostępnia zestaw interfejsów API i narzędzi, których można użyć do zdefiniowania i dostosowania architektury sieci neuronowej w oparciu o konkretne wymagania danego problemu.

Po zdefiniowaniu architektury model jest trenowany przy użyciu zsyntetyzowanych danych grafu. NSL zapewnia algorytmy i techniki treningowe, które uwzględniają strukturę grafu podczas procesu uczenia się. Algorytmy te umożliwiają modelowi uczenie się zarówno z oryginalnych danych, jak i rozszerzonego grafu, co prowadzi do poprawy wydajności i niezawodności.

Podczas procesu szkolenia modele NSL mogą wykorzystywać strukturę grafu do przechwytywania relacji i zależności między instancjami danych. Może to być szczególnie korzystne w scenariuszach, w których instancje danych są ze sobą połączone, takich jak sieci społecznościowe, systemy rekomendacji lub sieci biologiczne. Poprzez włączenie struktury grafu do procesu uczenia się modele NSL mogą lepiej przechwytywać podstawowe wzorce i podejmować dokładniejsze prognozy lub decyzje.

Neural Structured Learning with TensorFlow umożliwia szkolenie sieci neuronowych przy użyciu syntetyzowanych grafów. Poprzez włączenie danych o strukturze grafowej do procesu uczenia się, modele NSL mogą poprawić wydajność i generalizację. Możliwość wykorzystania struktury grafowej może być szczególnie przydatna w scenariuszach, w których instancje danych są ze sobą połączone. TensorFlow, dzięki swoim rozległym możliwościom i wsparciu społeczności, zapewnia potężną platformę do wdrażania NSL i eksplorowania potencjału uczenia o strukturze grafowej w aplikacjach AI.

Temat: Uczenie się adwersarskie w celu klasyfikacji obrazów

Kluczowym pomysłem stojącym za NSL jest przedstawienie danych jako grafu, w którym węzły reprezentują próbki, a krawędzie przechwytują relacje między nimi. Poprzez włączenie tej struktury grafu do procesu uczenia się, NSL zachęca modele do uczenia się z podstawowych połączeń między punktami danych. To podejście pomaga modelom lepiej uogólniać i zwiększa ich odporność na ataki przeciwnika.

Uczenie się przez przeciwnika, poddziedzina AI, koncentruje się na zrozumieniu i obronie przed atakami przeciwnika. W kontekście klasyfikacji obrazów ataki przeciwnika obejmują wprowadzanie małych, niezauważalnych zmian w obrazie wejściowym, aby oszukać model i zmusić go do błędnej klasyfikacji. Ataki przeciwnika stanowią poważne wyzwanie dla niezawodności modeli AI, ponieważ można je łatwo oszukać za pomocą starannie opracowanych danych wejściowych.

Aby sprostać temu wyzwaniu, NSL włącza techniki uczenia się adwersarzy do procesu szkolenia. Poprzez rozszerzenie danych szkoleniowych o przykłady adwersarzy, NSL umożliwia modelom naukę solidnych cech, które są mniej podatne na ataki adwersarzy. To podejście zwiększa zdolność modelu do prawidłowej klasyfikacji zarówno czystych, jak i adwersarzy obrazów.

Integracja NSL z TensorFlow zapewnia potężne ramy do trenowania solidnych modeli klasyfikacji obrazów. Wykorzystując techniki regularyzacji oparte na grafach NSL, programiści mogą udoskonalić możliwości generalizacji swoich modeli i zwiększyć ich odporność na ataki przeciwników. Ta kombinacja technik umożliwia modelom AI osiągnięcie większej dokładności i niezawodności w rzeczywistych zastosowaniach.

Neural Structured Learning firmy TensorFlow oferuje cenne podejście do poprawy wydajności modeli AI, szczególnie w kontekście klasyfikacji obrazów. Poprzez włączenie regularyzacji opartej na grafach i technik uczenia się adwersarskiego, NSL zwiększa możliwości generalizacji i solidność modeli. Ta integracja umożliwia programistom tworzenie bardziej niezawodnych i dokładnych systemów AI.

Lekcja: Przetwarzanie języka naturalnego z TensorFlow

Temat: Tokenizacja

Sztuczna inteligencja (AI) zrewolucjonizowała różne dziedziny, w tym przetwarzanie języka naturalnego (NLP). NLP koncentruje się na umożliwieniu maszynom rozumienia i przetwarzania języka ludzkiego. TensorFlow, popularny framework uczenia maszynowego typu open source, zapewnia potężne narzędzia i biblioteki do budowania modeli AI, w

tym tych do zadań NLP. Jednym z ważnych kroków w NLP jest tokenizacja, która polega na rozbiciu tekstu na mniejsze jednostki zwane tokenami. W tym materiale dydaktycznym zbadamy podstawy tokenizacji przy użyciu TensorFlow do zastosowań NLP.

Tokenizacja to proces dzielenia tekstu na pojedyncze słowa, frazy lub inne znaczące jednostki znane jako tokeny. Te tokeny służą jako elementy składowe dla kolejnych zadań NLP, takich jak analiza sentymentu, rozpoznawanie nazwanych jednostek i tłumaczenie maszynowe. TensorFlow oferuje kilka technik tokenizacji, które odpowiadają różnym wymaganiom i zastosowaniom.

Jednym z powszechnych podejść do tokenizacji jest tokenizacja słów, w której tekst jest dzielony na pojedyncze słowa. **TensorFlow udostępnia moduł `tokenizer`, który obejmuje różne tokenizery, takie jak `WhitespaceTokenizer`, `WordTokenizer` i `UnicodeScriptTokenizer`. Te tokenizery stosują różne strategie dzielenia tekstu na podstawie odstępów, znaków interpunkcyjnych lub skryptów Unicode.**

Na przykład **`WhitespaceTokenizer` dzieli tekst na podstawie znaków odstępu, takich jak spacje i tabulatory**. Kolejne znaki niebędące odstępami traktuje jako pojedynczy token. Z drugiej strony **`WordTokenizer` tokenizuje tekst, traktując znaki interpunkcyjne jako oddzielne tokeny.** Ten tokenizer jest przydatny, gdy interpunkcja niesie znaczenie semantyczne, np. w analizie sentymentu, gdzie obecność wykrzykników może wskazywać na silne emocje.

Oprócz tokenizacji słów, TensorFlow obsługuje również tokenizację podsłowną. Tokenizacja podsłowna dzieli tekst na mniejsze jednostki podsłowne, takie jak prefiksy, sufiksy lub rdzenie słów. Ta technika jest szczególnie przydatna do obsługi słów spoza słownika lub języków o bogatej morfologii. TensorFlow zapewnia `SubwordTokenizer` do tokenizacji podsłownej, który wykorzystuje algorytmy takie jak Byte Pair Encoding (BPE) lub WordPiece.

Byte Pair Encoding to technika kompresji danych, która iteracyjnie zastępuje najczęściej występującą parę bajtów nowym bajtem, którego nie ma w oryginalnych danych. W NLP BPE jest przystosowany do generowania jednostek podsłownych. `SubwordTokenizer` używając BPE uczy się słownictwa jednostek podsłownych z danych treningowych. Podczas tokenizacji zastępuje rzadkie lub nieznane słowa ich jednostkami podsłownymi, umożliwiając modelowi skuteczne radzenie sobie z niewidzianymi słowami.

Tokenizacja jest ważnym etapem wstępnego przetwarzania w NLP, ponieważ umożliwia reprezentację tekstu w formacie odpowiednim dla modeli uczenia maszynowego. Możliwości tokenizacji TensorFlow zapewniają elastyczność i możliwość dostosowania do różnych zadań NLP. Wykorzystując odpowiedni tokenizer, programiści mogą zwiększyć wydajność i dokładność swoich modeli AI.

TensorFlow oferuje potężne narzędzia do tokenizacji w aplikacjach NLP. Niezależnie od tego, czy chodzi o tokenizację słów, czy podsłów, TensorFlow zapewnia szereg tokenizatorów, które spełniają różne wymagania. Dzięki zrozumieniu i wykorzystaniu tych technik tokenizacji programiści mogą skutecznie wstępnie przetwarzać dane tekstowe, torując drogę do dokładniejszych i bardziej znaczących modeli AI w dziedzinie NLP.

Na początek rozważmy słowo „listen”. Słowo to składa się z ciągu liter, które można przedstawić za pomocą liczb, korzystając ze schematu kodowania, takiego jak ASCII. Jednak słowo „silent” ma te same litery w innej kolejności, co utrudnia zrozumienie sentymentu słowa wyłącznie na podstawie jego liter.

Zamiast kodować litery, łatwiej może być zakodować same słowa. Na przykład w zdaniu „Kocham mojego psa” możemy przypisać słowu „ja” liczbę 1, a zdanie jako całość będzie reprezentowane jako 1, 2, 3, 4. Jeśli weźmiemy inne zdanie, takie jak „Kocham mojego kota”, możemy je zakodować jako 1, 2, 3, 5, gdzie „Kocham mojego” ma już przypisane liczby i musimy zakodować tylko słowo „kot”. Porównując zakodowane sekwencje dwóch zdań, możemy zaobserwować podobieństwo między nimi, ponieważ oba wyrażają miłość do zwierzaka.

Teraz przyjrzyjmy się, jak możemy wdrożyć tokenizację za pomocą TensorFlow. Istnieje API dostępne dla tokenizacji i pokażemy, jak go używać z Pythonem. Oto przykład kodu, który tokenizuje zdania:

```
python

from tensorflow.keras.preprocessing.text import Tokenizer
```

```
sentences = ["I love my dog", "I love my cat"]

tokenizer = Tokenizer(num_words=100)
tokenizer.fit_on_texts(sentences)

word_index = tokenizer.word_index
print(word_index)
```

W powyższym kodzie importujemy niezbędny interfejs API tokenizatora z TensorFlow Keras. Następnie tworzymy wystąpienie obiektu tokenizatora, określając maksymalną liczbę słów do zachowania (w tym przypadku 100). Następnie tokenizator jest dopasowywany do podanych zdań i możemy uzyskać dostęp do właściwości indeksu słów, aby uzyskać słownik, w którym kluczami są słowa, a wartościami odpowiadające im tokeny.

Tokenizer jest również wystarczająco inteligentny, aby obsługiwać wyjątki. Na przykład, jeśli dodamy trzecie zdanie, takie jak „Kocham mojego psa!", gdzie po „piesku" następuje wykrzyknik, tokenizer rozpozna, że nie powinien tworzyć nowego tokena dla „pies wykrzyknik", ale raczej traktować go jako istniejący token dla „pies". Ponadto utworzy nowy token dla słowa „ty", jeśli pojawi się ono w tekście.

Jeśli chcesz wypróbować kod samodzielnie, znajdziesz go w dostarczonym notatniku Colab. Tokenizując słowa i zdania, wykonałeś ważny krok w przygotowaniu danych do przetwarzania przez sieć neuronową. W następnym odcinku przyjrzymy się narzędziom dostępnym w TensorFlow do zarządzania sekwencjonowaniem liczb w celu reprezentowania zdań. Nie zapomnij zasubskrybować, aby uzyskać więcej materiałów edukacyjnych.

Example Task

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
sentences = [
    'I love my dog',
    'I love my cat'
]

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
{'i': 1, 'my': 3, 'dog': 4, 'cat': 5, 'love': 2}
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!'
]
```

```
{'i': 3, 'my': 2, 'you': 6, 'love': 1, 'cat': 5, 'dog': 4}
```

Temat: Sekwencjonowanie – przekształcanie zdań w dane

Sztuczna inteligencja (AI) zrewolucjonizowała różne dziedziny, w tym przetwarzanie języka naturalnego (NLP). NLP obejmuje interakcję między komputerami a językiem ludzkim, umożliwiając maszynom rozumienie, interpretowanie i generowanie języka ludzkiego. TensorFlow, popularna biblioteka typu open source, zapewnia potężne narzędzia do wdrażania zadań NLP. W tym materiale dydaktycznym zbadamy podstawy TensorFlow i sposób, w jaki można go używać do sekwencjonowania, w szczególności do przekształcania zdań w dane.

Sekwencjonowanie jest ważnym aspektem NLP, ponieważ obejmuje konwersję danych tekstowych do formatu, który może być przetwarzany przez modele uczenia maszynowego. TensorFlow oferuje kilka technik, aby to osiągnąć, takich jak tokenizacja, kodowanie i osadzanie. Tokenizacja obejmuje rozbijanie zdań na pojedyncze słowa lub podsłowa, które są następnie reprezentowane jako tokeny. Ten proces pozwala maszynom zrozumieć strukturę i znaczenie zdań.

Kodowanie to kolejny krok w sekwencjonowaniu, w którym tokeny są mapowane na wartości liczbowe. Ta konwersja umożliwia maszynom wykonywanie operacji matematycznych na danych**. TensorFlow zapewnia różne techniki kodowania, takie jak kodowanie one-hot, kodowanie integer i kodowanie wordpiece. Kodowanie one-hot przedstawia każdy token jako wektor binarny, z wartością 1 dla pozycji tokena i 0 dla wszystkich innych pozycji. Kodowanie integer przypisuje unikalną wartość całkowitą do każdego tokena. Kodowanie wordpiece dzieli słowa na podsłowa, umożliwiając modelowi wydajne radzenie sobie ze słowami spoza słownika.**

Po tokenizacji i zakodowaniu zdań TensorFlow oferuje techniki osadzania, aby reprezentować tokeny w gęstej przestrzeni wektorowej. Osadzanie słów przechwytuje relacje semantyczne między słowami, umożliwiając maszynom zrozumienie znaczenia i kontekstu zdań. **Popularne metody osadzania obejmują Word2Vec, GloVe i BERT. Te wstępnie wyszkolone modele można łatwo zintegrować z potokami TensorFlow, zapewniając potężne reprezentacje dla zadań NLP.**

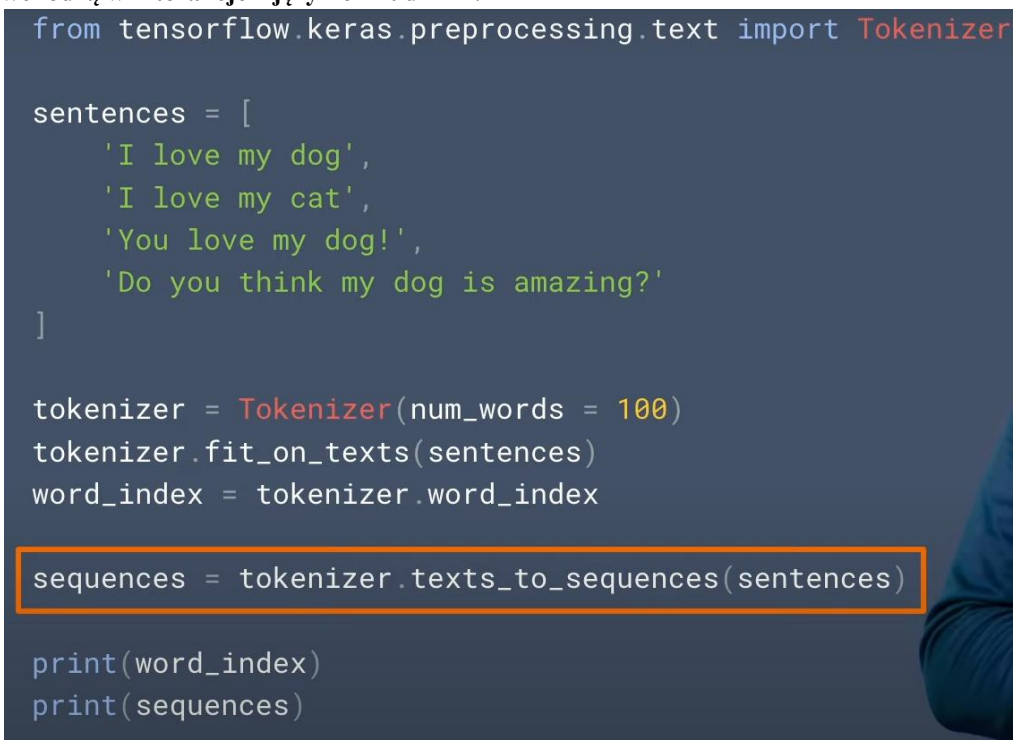
W TensorFlow proces przekształcania zdań w dane obejmuje tworzenie sekwencyjnego modelu, który przyjmuje zakodowane tokeny jako dane wejściowe. Model ten można skonstruować przy użyciu różnych warstw, takich jak warstwy osadzania, rekurencyjne sieci neuronowe (RNN) lub transformatory. Warstwy osadzania konwertują zakodowane tokeny na gęste wektory, przechwytując informacje semantyczne. Sieci RNN, takie jak LSTM lub GRU, są powszechnie używane do modelowania sekwencji, ponieważ mogą przechwytywać długoterminowe zależności w zdaniach. Transformatory z kolei zyskały popularność ze względu na ich zdolność do obsługi przetwarzania równoległego i wydajnego przechwytywania globalnych zależności.

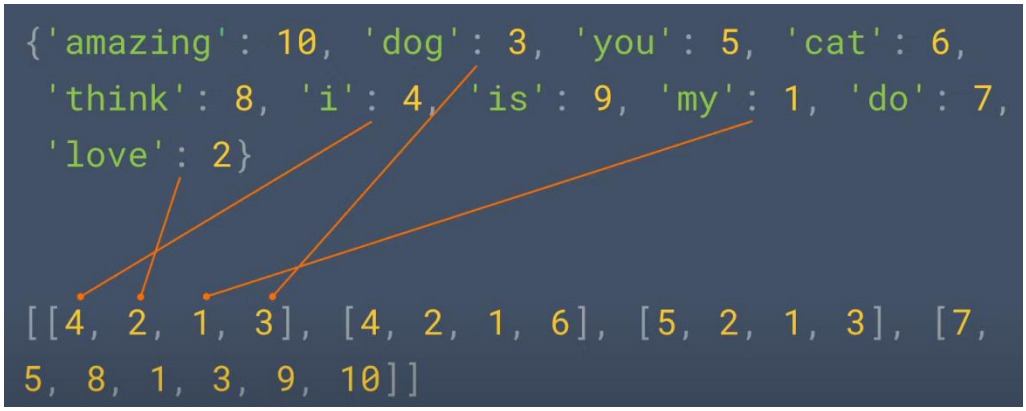
Aby wytrenować model sekwencyjny, TensorFlow udostępnia techniki optymalizacji, takie jak gradient zstępujący i propagacja wsteczna. Te algorytmy dostosowują parametry modelu, aby zminimalizować różnicę między przewidywanym wynikiem a prawdą. Proces szkolenia obejmuje zasilanie modelu oznaczonymi danymi, obliczenie straty i odpowiednią aktualizację wag. Możliwości automatycznego różnicowania TensorFlow upraszczają implementację tych technik optymalizacji.

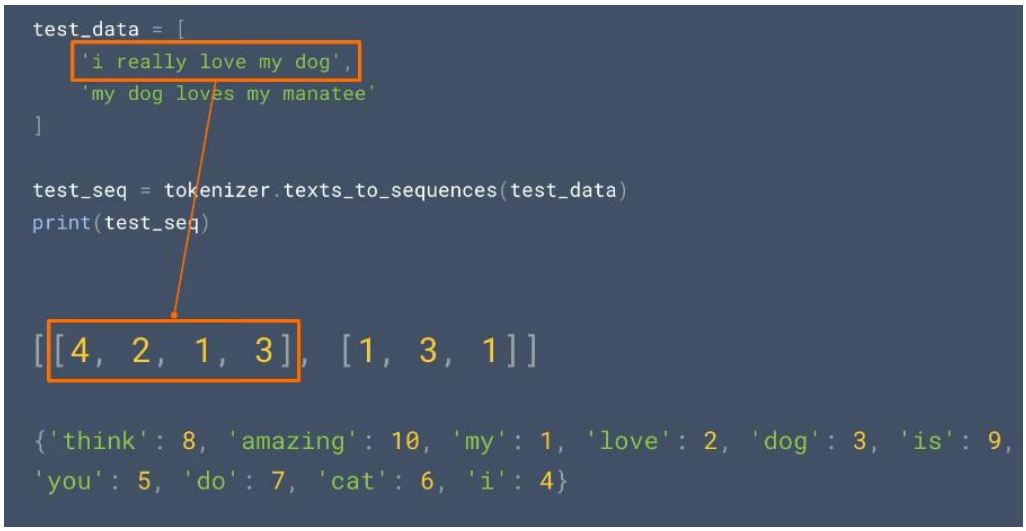
Po wytrenowaniu modelu sekwencyjnego można go używać do różnych zadań NLP, takich jak analiza sentymentów, klasyfikacja tekstu, rozpoznawanie nazwanych jednostek, tłumaczenie maszynowe i wiele innych. Elastyczność TensorFlow

pozwala badaczom i deweloperom z łatwością budować złożone potoki NLP. Wykorzystując moc głębokiego uczenia i rozbudowany zestaw narzędzi TensorFlow, możliwości przetwarzania języka naturalnego są nieograniczone.

TensorFlow zapewnia solidne ramy do wdrażania zadań przetwarzania języka naturalnego, w tym sekwencjonowania. Dzięki tokenizacji, kodowaniu i osadzaniu zdań maszyny mogą skutecznie rozumieć i przetwarzać dane tekstowe. Dzięki różnym warstwom i technikom optymalizacji TensorFlow umożliwia tworzenie potężnych modeli sekwencyjnych dla szerokiej gamy aplikacji NLP. Wykorzystując możliwości AI i TensorFlow, możemy odblokować potencjał przetwarzania języka naturalnego i zrewolucjonizować sposób, w jaki maszyny wchodzą w interakcje z językiem ludzkim.







Temat: Szkolenie modelu w celu rozpoznawania sentymentów w tekście

Aby wytrenować model do analizy sentymentu, najpierw potrzebujemy zbioru danych. Zbiór danych analizy sentymentu zazwyczaj składa się z próbek tekstu oznaczonych odpowiadającym im sentymentem (np. pozytywnym, negatywnym, neutralnym). TensorFlow udostępnia różne techniki wstępnego przetwarzania i przygotowywania danych. Może to obejmować tokenizację tekstu, konwersję go na reprezentacje numeryczne i podział na zestawy treningowe i testowe.

Gdy dane są już przygotowane, możemy zacząć budować model przy użyciu API wysokiego poziomu TensorFlow, Keras. Keras zapewnia przyjazny dla użytkownika interfejs do definiowania i trenowania sieci neuronowych. W analizie sentymentów powszechnym podejściem jest użycie rekurencyjnej sieci neuronowej (RNN) lub jej wariantu zwanego siecią o długiej pamięci krótkotrwałej (LSTM). Sieci te są dobrze przystosowane do przetwarzania danych sekwencyjnych, takich jak tekst.

Architektura modelu analizy sentymentu zazwyczaj składa się z warstwy osadzania, która konwertuje dane tekstowe na gęste wektory liczbowe, a następnie z jednej lub więcej warstw LSTM. Warstwy LSTM przechwytują informacje kontekstowe i uczą się rozpoznawać wzorce wskazujące na sentyment. Na koniec gęsta warstwa z aktywacją softmax jest używana do przewidywania klasy sentymentu.

Szkolenie modelu obejmuje optymalizację jego parametrów w celu zminimalizowania błędu predykcji. TensorFlow udostępnia różne algorytmy optymalizacji, takie jak stochastyczny gradient zstępujący (SGD) i Adam, które dostosowują wagi modelu na podstawie gradientów obliczonych podczas propagacji wstecznej. Proces szkolenia obejmuje zasilanie modelu partiami oznaczonych danych, obliczenie straty i aktualizację wag przy użyciu wybranego algorytmu optymalizacji.

Aby ocenić wydajność wytrenowanego modelu, możemy użyć metryk, takich jak dokładność, precyzja, odwołanie i wynik F1. Te metryki dostarczają informacji o tym, jak dobrze model generalizuje niewidziane dane. Ponadto możemy wizualizować postępy w szkoleniu za pomocą TensorBoard TensorFlow, co pozwala nam monitorować różne metryki i wizualizować architekturę modelu.

Po wytrenowaniu i ocenie modelu możemy go użyć do przewidywania sentymentu nowych, niewidzianych próbek tekstu. TensorFlow zapewnia wygodne metody dokonywania przewidywań przy użyciu wytrenowanego modelu. Wprowadzając tekst do modelu, możemy uzyskać przewidywaną klasę sentymentu wraz z powiązanymi wynikami ufności.

TensorFlow to potężne narzędzie do trenowania modeli w celu rozpoznawania sentymentu w tekście. Wykorzystując jego możliwości, możemy budować i trenować zaawansowane modele, które mogą dokładnie analizować sentyment wyrażony w danych tekstowych. Zrozumienie podstaw TensorFlow i NLP jest ważne dla opracowywania aplikacji AI, które mogą przetwarzać i rozumieć język ludzki.

Temat: ML z rekurencyjnymi sieciami neuronowymi

Jednym z kluczowych komponentów NLP z TensorFlow jest wykorzystanie rekurencyjnych sieci neuronowych (RNN). RNN to klasa sztucznych sieci neuronowych, które doskonale radzą sobie z przetwarzaniem danych sekwencyjnych, co czyni je dobrze przystosowanymi do zadań NLP. Mają zdolność do wychwytywania zależności czasowych w sekwencji słów, umożliwiając modelowi zrozumienie kontekstu i znaczenia tekstu.

W TensorFlow sieci neuronowe RNN można implementować za pomocą interfejsu API TensorFlow do budowania sieci neuronowych, znanego jako Keras. Keras zapewnia interfejs wysokiego poziomu do konstruowania i trenowania modeli głębokiego uczenia, w tym sieci neuronowych RNN. Upraszcza proces budowania złożonych architektur, zapewniając wstępnie zdefiniowane warstwy i moduły, które można łatwo ze sobą łączyć.

Aby zrozumieć działanie NLP z TensorFlow i RNN, rozważmy przykład analizy sentymentu. **Analiza sentymentu obejmuje określenie sentymentu lub emocji wyrażonych w tekście, takich jak pozytywny, negatywny lub neutralny**. Za pomocą TensorFlow możemy wytrenować model oparty na RNN, aby klasyfikować sentyment danego tekstu.

Oprócz analizy sentymentów, TensorFlow może być używany do szerokiego zakresu zadań NLP. Na przykład do tłumaczenia maszynowego możemy używać modeli sekwencja-sekwencja z mechanizmami uwagi. Do odpowiadania na pytania możemy stosować modele wykorzystujące architekturę transformatora. Elastyczność TensorFlow i rozbudowane wsparcie bibliotek sprawiają, że jest to potężne narzędzie do eksploracji i implementacji różnych aplikacji NLP.

TensorFlow zapewnia kompleksową platformę do przetwarzania języka naturalnego za pomocą rekurencyjnych sieci neuronowych. Jego rozbudowane funkcjonalności w połączeniu z elastycznością API Keras umożliwiają programistom budowanie i trenowanie modeli głębokiego uczenia dla szerokiego zakresu zadań NLP. Wykorzystując moc TensorFlow, możemy odblokować potencjał AI i zwiększyć naszą zdolność do rozumienia i przetwarzania języka ludzkiego.

Temat: Pamięć długotrwała i krótkotrwała dla przetwarzania języka naturalnego

Aby zastosować TensorFlow w zadaniach NLP, najpierw musimy wstępnie przetworzyć dane tekstowe. Obejmuje to tokenizację, w której dzielimy tekst na pojedyncze słowa lub podsłowa. **Tokenizacja jest ważna dla tworzenia reprezentacji liczbowej tekstu, która może zostać wprowadzona do modelu LSTM. TensorFlow zapewnia różne techniki tokenizacji, takie jak tokenizacja oparta na słowach przy użyciu interfejsu API Tokenizer lub tokenizacja oparta na podsłowie przy użyciu tokenizatora BERT.**

Po tokenizacji danych tekstowych możemy przystąpić do budowania modelu LSTM przy użyciu TensorFlow. LSTM to typ RNN, który rozwiązuje problem zanikającego gradientu, umożliwiając modelowi przechwytywanie długoterminowych zależności w sekwencyjnych danych. Składa się z komórek pamięci, które przechowują i aktualizują informacje w czasie, co czyni go idealnym do przetwarzania danych tekstowych. TensorFlow udostępnia warstwę LSTM jako część swojego API Keras, co ułatwia włączenie jej do naszych modeli NLP.

Oprócz warstwy LSTM możemy zwiększyć wydajność naszych modeli NLP, włączając inne warstwy, takie jak warstwy osadzania, warstwy dropout i warstwy gęste. Warstwa osadzania konwertuje tokenizowany tekst na gęste wektory, przechwytując relacje semantyczne między słowami. Warstwy dropout pomagają zapobiegać nadmiernemu dopasowaniu poprzez losowe wyłączanie części neuronów podczas treningu. Gęste warstwy to w pełni połączone warstwy, które wykonują zadania klasyfikacji lub regresji w oparciu o dane wyjściowe warstwy LSTM.

Trening modelu NLP za pomocą TensorFlow obejmuje zdefiniowanie architektury modelu, skompilowanie go z odpowiednimi funkcjami strat i optymalizacji oraz dopasowanie go do danych treningowych. TensorFlow zapewnia szeroki zakres funkcji strat dla różnych zadań NLP, takich jak kategorialna entropia krzyżowa do klasyfikacji wieloklasowej lub binarna entropia krzyżowa do analizy sentymentu. Funkcje optymalizacyjne, takie jak Adam lub RMSprop, mogą być używane do aktualizacji parametrów modelu podczas treningu.

Po wytrenowaniu modelu możemy ocenić jego wydajność na podstawie niewidzianych danych, używając metryk, takich jak dokładność, precyzja, odwołanie lub wynik F1. TensorFlow zapewnia wygodne funkcje do obliczania tych metryk. Ponadto możemy wizualizować wydajność modelu, używając narzędzi, takich jak TensorBoard, które pozwalają nam monitorować proces szkolenia, wizualizować architekturę modelu i analizować osadzenia.

Temat: Szkolenie AI w zakresie tworzenia poezji

Jednym z kluczowych komponentów NLP jest wstępne przetwarzanie tekstu, które polega na przekształcaniu surowych danych tekstowych do formatu odpowiedniego do analizy. Zazwyczaj obejmuje to kroki takie jak tokenizacja, gdzie tekst jest dzielony na pojedyncze słowa lub znaki, oraz normalizacja, gdzie słowa są konwertowane do ich formy bazowej (np. konwersja „running" na „run"). TensorFlow udostępnia różne funkcje i moduły ułatwiające te kroki wstępnego przetwarzania, ułatwiając czyszczenie i przygotowywanie danych tekstowych do dalszej analizy.

Po wstępnym przetworzeniu danych tekstowych można ich użyć do trenowania modeli AI do określonych zadań NLP. W przypadku generowania poezji celem jest trenowanie modelu, który może generować spójne i estetycznie przyjemne wiersze. TensorFlow oferuje kilka algorytmów i architektur, które można wykorzystać w tym celu, w tym rekurencyjne sieci neuronowe (RNN) i modele transformatorowe.

Sieci RNN są szczególnie dobrze przystosowane do danych sekwencyjnych, takich jak tekst, ponieważ mogą przechwytywać zależności czasowe i generować dane wyjściowe na podstawie poprzednich danych wejściowych. Zostały pomyślnie zastosowane w różnych zadaniach NLP, w tym modelowaniu języka i generowaniu tekstu. TensorFlow zapewnia interfejs API wysokiego poziomu o nazwie Keras, który ułatwia budowanie i trenowanie modeli opartych na sieciach RNN do generowania poezji.

Inną potężną architekturą dla zadań NLP jest model transformatora. Transformatory zyskały popularność w ostatnich latach ze względu na ich zdolność do przechwytywania zależności dalekiego zasięgu w danych tekstowych. Składają się ze struktury kodera-dekodera i wykorzystują mechanizmy uwagi, aby skupić się na odpowiednich częściach sekwencji wejściowej. Oficjalna implementacja modelu transformatora TensorFlow, znana jako „Transformer" w repozytorium modeli TensorFlow, może być używana do generowania poezji.

Szkolenie modelu AI do generowania poezji polega na zasilaniu go dużym korpusem istniejących wierszy. Korpus ten służy jako dane szkoleniowe, umożliwiając modelowi naukę wzorców, stylów i struktur obecnych w poezji. TensorFlow zapewnia wydajne narzędzia do obsługi dużych zestawów danych, w tym potoki danych i rozproszone szkolenie, co może przyspieszyć proces szkolenia.

Podczas treningu model AI uczy się generować wiersze, przewidując następne słowo lub wers na podstawie podanego kontekstu. Osiąga się to poprzez proces zwany „wymuszaniem przez nauczyciela", w którym model otrzymuje poprawne dane wyjściowe na każdym etapie. Jednak podczas wnioskowania model generuje dane wyjściowe autonomicznie, bez żadnego wymuszania przez nauczyciela. To właśnie tutaj ujawnia się prawdziwy potencjał twórczy modelu AI, ponieważ może on generować unikalne i oryginalne wiersze na podstawie wzorców, których się nauczył.

Aby ocenić jakość wygenerowanych wierszy, można użyć różnych metryk, takich jak perplexity, coherence i ocena ludzka. Perplexity mierzy, jak dobrze model przewiduje następne słowo, przy czym niższe wartości wskazują na lepszą wydajność. Coherence ocenia logiczny przepływ i łączność wygenerowanego tekstu. Ocena ludzka obejmuje pozyskiwanie opinii od sędziów ludzkich w celu oceny jakości estetycznej i ogólnej kreatywności wygenerowanych wierszy.

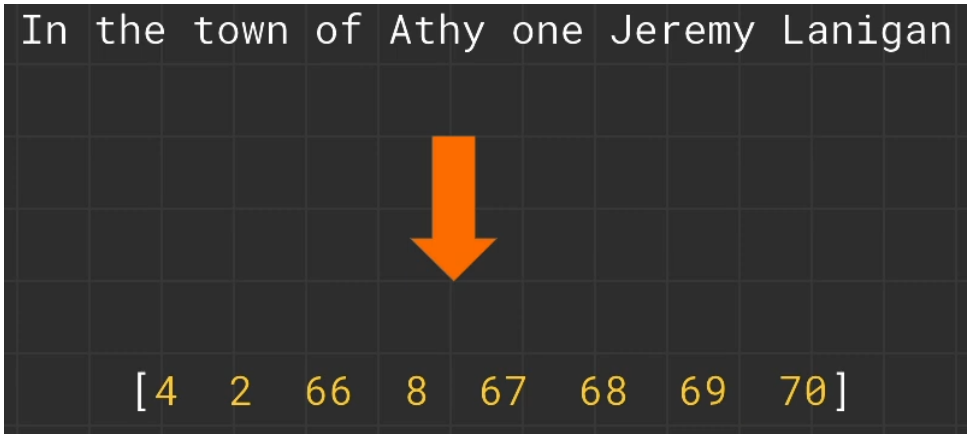
TensorFlow zapewnia solidne ramy do trenowania modeli AI do generowania poezji przy użyciu technik przetwarzania języka naturalnego. Wykorzystując moc narzędzi i algorytmów TensorFlow, programiści i badacze mogą tworzyć systemy AI, które mogą generować arcydzieła poetyckie. Otwiera to ekscytujące możliwości dla kreatywnych zastosowań AI w dziedzinie literatury i sztuki.

```
tokenizer = Tokenizer()

data="In the town of Athy one Jeremy Lanigan \n Battered away ... ..."
corpus = data.lower().split("\n")

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```



Lekcja: Programowanie TensorFlow

Temat: Wprowadzenie do kodowania TensorFlow

TensorFlow to framework uczenia maszynowego typu open source opracowany przez Google. Jest szeroko stosowany do budowania i wdrażania modeli uczenia maszynowego, zwłaszcza tych obejmujących głębokie sieci neuronowe. TensorFlow zapewnia kompleksowy zestaw narzędzi i bibliotek, które umożliwiają programistom wydajną implementację i trenowanie złożonych modeli. W tym materiale dydaktycznym przedstawimy podstawy kodowania TensorFlow, omawiając jego kluczowe komponenty i podając przykłady ilustrujące jego użycie.

Podstawą TensorFlow jest koncepcja grafu obliczeniowego. Graf obliczeniowy to seria połączonych ze sobą węzłów, gdzie każdy węzeł reprezentuje operację, a krawędzie reprezentują przepływ danych. TensorFlow umożliwia użytkownikom wydajne definiowanie i wykonywanie grafów obliczeniowych, co czyni go odpowiednim do zadań uczenia maszynowego na dużą skalę.

Aby rozpocząć programowanie z TensorFlow, musimy zaimportować wymagane biblioteki. Najczęstszym poleceniem importu jest:

```
import tensorflow as tf
```

Po zaimportowaniu możemy zdefiniować nasz graf obliczeniowy za pomocą interfejsu API wysokiego poziomu TensorFlow. Ten interfejs API zapewnia zestaw wstępnie zbudowanych funkcji i klas, które upraszczają proces budowania i trenowania modeli. Na przykład możemy utworzyć prosty graf, który dodaje dwie liczby do siebie w następujący sposób:

```
a = tf.constant(2)
b = tf.constant(3)
c = tf.add(a, b)
```

W powyższym fragmencie kodu definiujemy dwa stałe węzły (`a` i `b`) o wartościach odpowiednio 2 i 3. Następnie używamy funkcji `tf.add`, aby dodać te dwa węzły razem, co skutkuje nowym węzłem `c`. Należy zauważyć, że na tym etapie graf nie jest jeszcze wykonywany; jest to jedynie symboliczna reprezentacja obliczeń.

Aby wykonać graf i uzyskać wynik, musimy utworzyć sesję TensorFlow. Sesja hermetyzuje środowisko wykonawcze do wykonywania grafu. Możemy utworzyć sesję i uruchomić graf w następujący sposób:

```
with tf.Session() as sess:
    result = sess.run(c)
    print(result)
```

W tym fragmencie kodu tworzymy sesję za pomocą konstruktora `tf.Session()`. Instrukcja `with` zapewnia, że sesja zostanie poprawnie zamknięta po wykonaniu grafu. W ramach sesji używamy metody `sess.run`, aby uruchomić graf i uzyskać wartość węzła `c`. Na koniec drukujemy wynik, który w tym przypadku będzie wynosił 5.

TensorFlow oferuje również koncepcję zwaną placeholderami, która pozwala nam wprowadzać dane do grafu w czasie wykonywania. Placeholdery są zazwyczaj używane do reprezentowania danych wejściowych i docelowych w modelach uczenia maszynowego. Placeholder możemy zdefiniować w następujący sposób:

```
x = tf.placeholder(tf.float32, shape=(None, 2))
```

W tym przykładzie definiujemy symbol zastępczy `x`, który oczekuje dwuwymiarowego tensora liczb zmiennoprzecinkowych. Argument `shape` określa kształt tensora, a `None` wskazuje, że rozmiar może się zmieniać wzdłuż tego wymiaru.

Aby wprowadzić dane do symbolu zastępczego, możemy użyć argumentu `feed_dict` podczas uruchamiania grafu. Na przykład:

```
with tf.Session() as sess:
    result = sess.run(c, feed_dict={x: [[2, 3]]})
    print(result)
```

Tutaj podajemy słownik mapujący symbol zastępczy `x` na rzeczywiste dane, które chcemy wprowadzić. W tym przypadku przekazujemy dwuwymiarową tablicę zawierającą wartości 2 i 3. Następnie wykonywany jest wykres, a wynik jest drukowany.

Oprócz podstawowych operacji i symboli zastępczych TensorFlow obsługuje szeroki zakres operacji matematycznych, funkcji aktywacji, funkcji strat i algorytmów optymalizacji. Te narzędzia umożliwiają programistom łatwe budowanie złożonych modeli uczenia maszynowego. Ponadto elastyczność TensorFlow umożliwia rozproszone szkolenie na wielu urządzeniach, dzięki czemu nadaje się do aplikacji na dużą skalę.

TensorFlow zapewnia potężne i elastyczne ramy do programowania modeli uczenia maszynowego. Poprzez definiowanie grafów obliczeniowych, wykonywanie ich w sesjach i wykorzystywanie symboli zastępczych, programiści mogą wydajnie budować i trenować modele. Obszerna biblioteka funkcji i algorytmów TensorFlow jeszcze bardziej upraszcza proces, czyniąc go popularnym wyborem wśród badaczy i praktyków w dziedzinie sztucznej inteligencji.

Temat: Wprowadzenie do TensorFlow Lite

TensorFlow opiera się na koncepcji tensorów, które są wielowymiarowymi tablicami. Zapewnia kompleksowy ekosystem do opracowywania i wdrażania modeli uczenia maszynowego, oferując szereg funkcjonalności dla zadań takich jak wstępne przetwarzanie danych, szkolenie modeli i wnioskowanie. TensorFlow obsługuje zarówno interfejsy API wysokiego, jak i niskiego poziomu, umożliwiając użytkownikom wybór poziomu abstrakcji odpowiadającego ich potrzebom.

Programowanie w TensorFlow obejmuje konstruowanie grafu obliczeniowego, który przedstawia przepływ danych przez model. Graf składa się z węzłów, które wykonują operacje na tensorach i krawędzi, które przedstawiają przepływ danych między węzłami. TensorFlow zapewnia bogaty zestaw operacji dla typowych zadań, takich jak obliczenia matematyczne, manipulacje tablicami i warstwy sieci neuronowych.

Aby zobrazować proces programowania, rozważmy poniższy przykład budowy prostej sieci neuronowej przy użyciu TensorFlow:

```
python
import tensorflow as tf

# Define the model architecture
model = tf.keras.Sequential([
  tf.keras.layers.Dense(64, activation='relu', input_shape=(784,)),
  tf.keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, validation_data=(x_val, y_val))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
```

W tym przykładzie definiujemy model sekwencyjny z dwiema gęstymi warstwami. Pierwsza warstwa ma 64 jednostki i używa funkcji aktywacji ReLU, podczas gdy druga warstwa ma 10 jednostek i używa funkcji aktywacji softmax. Kompilujemy model za pomocą optymalizatora Adam i rzadkiej funkcji straty entropii krzyżowej. Następnie trenujemy model na zestawie danych treningowych i oceniamy jego wydajność na zestawie danych testowych.

TensorFlow Lite to rozszerzenie TensorFlow zaprojektowane specjalnie dla urządzeń mobilnych i wbudowanych. Umożliwia deweloperom wdrażanie modeli TensorFlow na platformach o ograniczonych zasobach, umożliwiając aplikacje AI na urządzeniach o ograniczonej mocy obliczeniowej. TensorFlow Lite osiąga to poprzez optymalizację modeli pod kątem wydajnego wykonywania, zmniejszanie ich rozmiaru i wykorzystywanie przyspieszenia sprzętowego, gdy jest ono dostępne.

Proces korzystania z TensorFlow Lite obejmuje konwersję modelu TensorFlow do formatu odpowiedniego do wdrożenia na urządzeniach mobilnych. Konwersja ta obejmuje kwantyzację modelu, która zmniejsza precyzję wag i aktywacji modelu, aby jeszcze bardziej zwiększyć wydajność. Po konwersji modelu można go zintegrować z aplikacjami mobilnymi i używać do zadań takich jak klasyfikacja obrazów, wykrywanie obiektów i przetwarzanie języka naturalnego.

Podsumowując, TensorFlow to potężne środowisko do budowania i wdrażania modeli AI. Dzięki zrozumieniu podstaw TensorFlow i programowaniu z jego użyciem, deweloperzy mogą wykorzystać jego możliwości do tworzenia zaawansowanych aplikacji uczenia maszynowego. Ponadto TensorFlow Lite rozszerza zasięg TensorFlow na urządzenia mobilne i osadzone, umożliwiając AI na krawędzi.

https://developers.google.com/protocol-buffers