高速な埋め込み: Spark* NLP と OpenVINO™ を使用して大規模で高速な BERT 推論を実現

この記事は、Medium に公開されている「Supercharged Embeddings: Unlocking Faster BERT Inference at Scale with Spark NLP and OpenVINO™」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

はじめに

Java* および Spark* アプリケーション内でのディープラーニング推論に OpenVINO™ ランタイムと Spark* NLP を活用するテクニカル・ブログ・シリーズのパート 3 へようこそ。前の 2 つのブログ [パート 1 (英語)、パート 2 (英語)] では、基礎および基本的な概念を説明しました。パート 1 では、Java* で OpenVINO™ ランタイムを使用してディープラーニング推論を実行する方法に重点を置いて、環境の設定、モデルのロード、推論の実行方法をステップごとに紹介しました。パート 2 では、OpenVINO™ ランタイムを使用して Spark* NLP エコシステム内で NLP 推論を高速化する方法を説明しました。

このパート 3 では、人気の高い bert-base-cased モデルを使用してトークンレベルの埋め込みを生成する Spark* NLP パイプラインを構築する方法を説明します。OpenVINO™ の機能を BertEmbeddings アノテーターのバックエンドとして活用し、効率的で最適化された埋め込みの抽出を実現します。

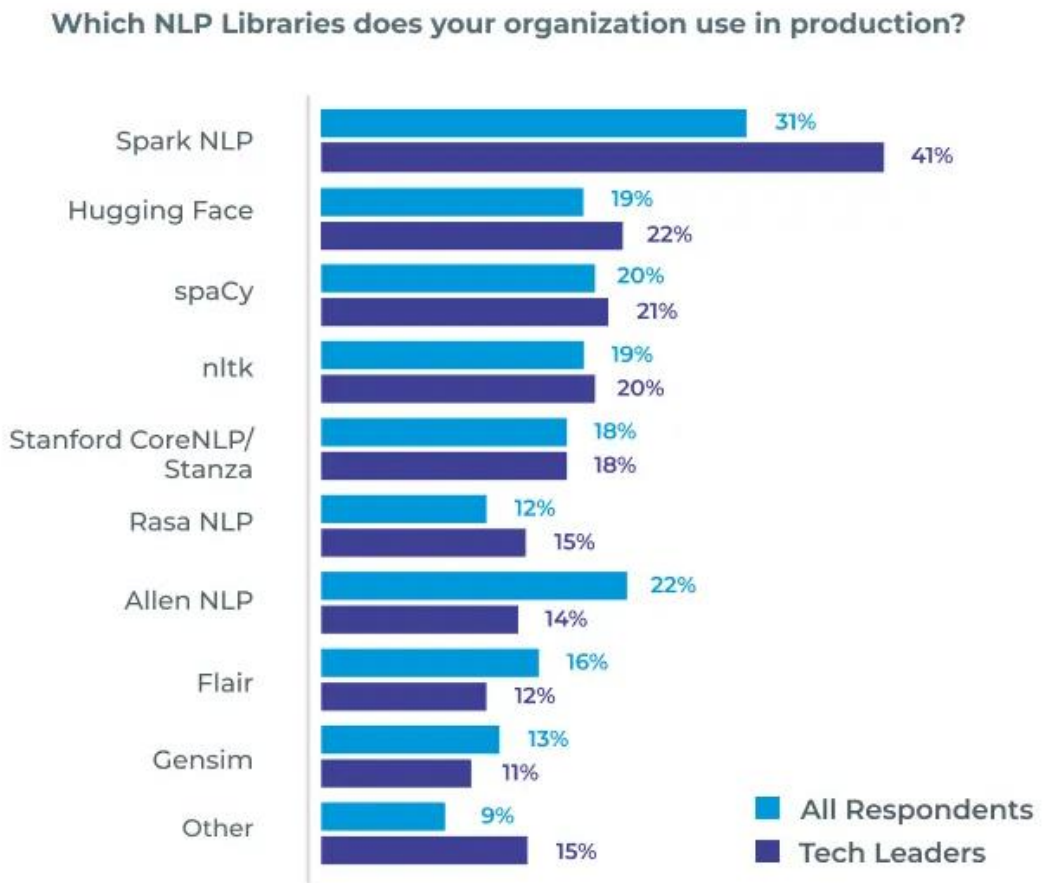
このシリーズのパート 1 とパート 2 をまだお読みになっていない場合は、この先に進む前に 2 つのブログで基本的な概念を確認することを推奨します。

- パート 1: Java* で OpenVINO™ ランタイムを使用したディープラーニング推論 (英語)
- パート 2: スピードの必要性: Spark* NLP で OpenVINO™ ランタイムを使用した NLP 推論の高速化 (英語)

Apache Spark* エコシステム

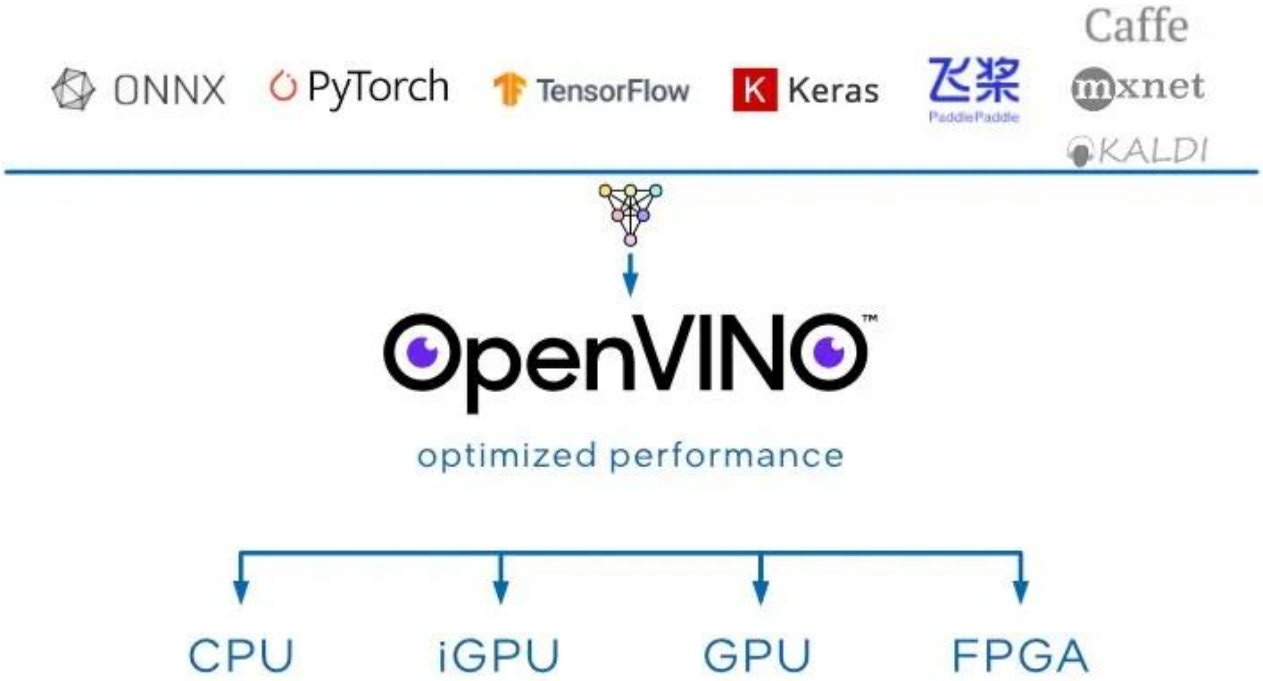
Apache Spark* は、汎用の分散型インメモリー・データ処理フレームワークです。大規模なデータセットの処理タスクを素早く効率良く実行して、バッチ・アプリケーション、対話型クエリー、ストリーム処理を含む幅広いワークロードを処理できます。これらの特性により、ビッグデータとマシンラーニングの分野で幅広く利用されています。Spark* には独自の ML ライブラリー、SparkML (英語) があり、独自のマシンラーニング・パイプラインを作成してチューニングできる API のセットを提供しています。現時点ではすべての NLP タスクには対応していないため、外部 NLP フレームワークを統合すると、文字列データのシリアル化とコピーの効率が低下します。

Spark* NLP は John Snow Labs (英語) により主導および後援されているオープンソース・プロジェクトで、分散型クラスターで簡単に拡張できる NLP アノテーションを提供します。Apache Spark* と SparkML の上に構築されており、生のテキストを構造化された特徴に変換し、独自の NLP パイプラインを実行して Scala と Python* の簡単な API を使用して結果を下流の ML パイプラインにフィードできる、プロダクション・グレードの統合ソリューションを提供します。Spark* NLP Models Hub で提供されている事前トレーニング済みモデルに加えて、カスタムモデルを同等の Spark* NLP アノテーター (バージョン 3.1.0 以降) にインポートすることもできます。サポートされているモデルのリストと、Hugging Face および TF Hub からモデルを Spark* NLP にインポートする方法を説明したサンプル・ノートブックのリストは、こちら (英語) を参照してください。



出典: John Snow Labs Spark* NLP (英語)

OpenVINO™ を Spark* NLP と統合 (英語) すると、OpenVINO™ バックエンドを使用して Spark* NLP にインポートしたモデルを実行できるようになります。OpenVINO™ は、Tensorflow*、Tensorflow* Lite、ONNX*、PaddlePaddle* 形式のモデルを処理できます。モデルを OpenVINO™ 中間表現 (IR) 形式に変換して、ツールキットの最適化および量子化機能を利用することもできます。詳細は、公式ドキュメント (英語) で提供されている、OpenVINO™ ツールキットを使用してデプロイのためにモデルを準備をする方法を参照してください。



出典: OpenVINO™ ツールキット (英語)

この記事では、次の手順を説明します。

1. Hugging Face から bert-base-cased モデルをエクスポートします。
2. Spark* NLP をセットアップします。
3. Spark* NLP パイプラインの BertEmbeddings アノテーターにモデルをロードします。
4. OpenVINO™ バックエンドを使用して Spark* NLP パイプラインを実行します。

モデルのエクスポート

Hugging Face (英語) は、強力なマシンラーニング・モデルを構築、トレーニング、デプロイできるツールへのアクセスを提供する AI コミュニティーとプラットフォームです。オープンソースのトランスフォーマー・フレームワークにより、Hugging Face は、自然言語処理、コンピューター・ビジョン、オーディオを含む異なる形式のさまざまなタスク向けに、最先端のトランスフォーマー・アーキテクチャーに基づく 20,000 以上の事前トレーニング済みモデルへのアクセスを提供する API とツールを提供します。

このデモでは、マスクされた言語モデリングや次の文の予測に使用できる生の BERT モデルである bert-base-cased (英語) モデルをエクスポートします。Hugging Face からモデルをエクスポートするには、PyPi (英語) 経由で利用できる `transformers` ライブラリーを使用します。

注: BertEmbeddings アノテーターは、Hugging Face の `Fill Mask` カテゴリーの BERT モデルをサポートしています。トークン分類などの特定のタスクでトレーニングまたは微調整されたモデルは使用できません。

必要条件

- Python* 3.6 以降

モデルをエクスポートする手順

- 依存ファイルの互換性問題を回避するため、まず Python* 仮想環境を作成して有効化します。

```
python -m venv .env
source .env/bin/activate
```

- Hugging Face の `transformers` ライブラリーをインストールします。このデモでは、バージョン 4.31.0 を使用します。ソースモデルは Tensorflow* ベースであるため、`tensorflow` バックエンドをインストールする必要があります。次のコマンドを実行します。

```
pip install transformers[tf-cpu]==4.31.0
```

- 必要な依存ファイルをインストールしたら、BERT モデルをダウンロードして、Tensorflow* SavedModel 形式でエクスポートします。

```
from transformers import TFBertModel, BertTokenizer
import tensorflow as tf

model = TFBertModel.from_pretrained(&apos;bert-base-cased&apos;)
tokenizer = BertTokenizer.from_pretrained(&apos;bert-base-cased&apos;)
```

- `TFBertModel` クラスは、本質的に tf.keras.Model (英語) サブクラスであり、特定のヘッドがない生の隠れ状態を出力する最小限の BERT モデルを表します。エクスポートする前に、次のようにモデルのシグネチャーを定義します。

```
@tf.function(
  input_signature=[
      {
          "input_ids": tf.TensorSpec((None, None), tf.int32,
name="input_ids"),
          "attention_mask": tf.TensorSpec((None, None), tf.int32,
name="attention_mask"),
          "token_type_ids": tf.TensorSpec((None, None), tf.int32,
name="token_type_ids"),
      }
  ]
)
def serving_fn(input):
    return model(input)
```

- これで、Hugging Face で提供されている `save_pretrained` 関数を使用してモデルを安全にエクスポートできるようになりました。モデルを SavedModel 形式でエクスポートするため、`saved_model` オプションを有効にします。

```
model.save_pretrained(save_directory="bert-base-cased", saved_model=True,
signatures={"serving_default": serving_fn})
tokenizer.save_vocabulary("bert-base-cased/saved_model/1/assets")
```

- モデルファイルが `bert-base-cased/saved_model/1` ディレクトリーに保存され、Spark* NLP 内のトークン化に必要な語彙ファイルが `assets` ディレクトリーに保存されます。

```
total 6.9M
drwxr-xr-x 2 root root 4.0K Aug 27 18:24 assets
-rw-r--r-- 1 root root   56 Aug 27 18:24 fingerprint.pb
-rw-r--r-- 1 root root 162K Aug 27 18:24 keras_metadata.pb
-rw-r--r-- 1 root root 6.7M Aug 27 18:24 saved_model.pb
drwxr-xr-x 2 root root 4.0K Aug 27 18:24 variables
```

- モデルのインポートに必要なのは `bert-base-cased/saved_model/1` ディレクトリーです。次のコマンドを使用して、このディレクトリーを `/root/models` に移動します。

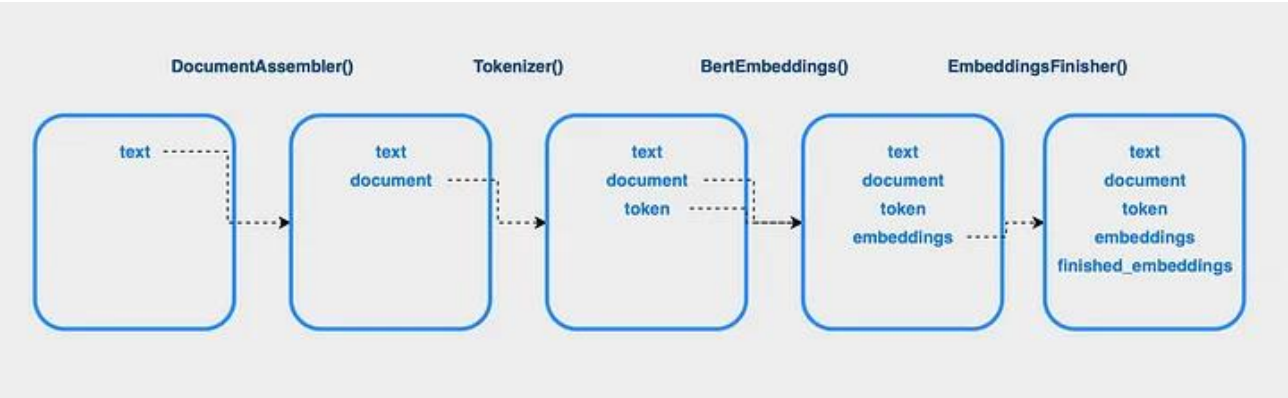
```
mkdir -p /root/models
mv bert-base-cased/saved_model/1 /root/models/bert-base-cased
```

Tensorflow* モデルを直接インポートできます。8 ビット量子化などの OpenVINO™ の最適化機能を利用してモデルのサイズを縮小し、CPU のスループットを向上させることもできます。モデルの最適化の手順とサポートされている手法は、最適化ガイド (英語) を参照してください。

Spark* NLP パイプライン

Spark* ML は、マシンラーニング・アプリケーションを構築するための 2 つの主要コンポーネント、**推定器**と**トランスフォーマー**を提供しています。推定器は `fit()` メソッドを使用してデータの一部のトレーニングを行い、トランスフォーマー (通常はフィッティング・プロセスの結果) は `transform()` メソッドを使用してターゲット DataFrame に変換を適用します。Spark* NLP は、アノテーター (NLP タスクの先頭に立つ主要コンポーネント) を通じてこれらの概念を拡張します。アノテーターには 2 つのタイプがあります。アノテーター・アプローチは、推定器を表し、トレーニング段階が必要です。アノテーター・モデルは、トランスフォーマーを表し、以前のデータを削除または置換することなく現在のアノテーションの結果を入力フレームに追加します。Spark* NLP アノテーターのリストと使用法は、こちら (英語) を参照してください。

パイプラインは、単一のワークフローで複数のアノテーターをチェーンにするメカニズムです。エンドツーエンドの NLP ワークフローは、各ステージがトークン化などの関連する変換を実行するパイプラインとして表すことができます。次の図は、**BertEmbeddings** アノテーターを使用して単語埋め込みを生成および抽出する単純なパイプラインを表しています。



サンプル・パイプラインのステージ

上記のパイプラインを実装する手順を次に示します。

必要条件

- Ubuntu* 20.04 以降
- OpenJDK* 8 以降

```
sudo apt-get update
sudo apt-get install -y default-jdk
```

- OpenVINO™ 2023.0.1 以降

このデモでは、OpenVINO™ バージョン 2023.0.1 を使用します。こちら (英語) の説明のように、OpenVINO™ コンポーネントは `/opt/intel/openvino-2023.0.1` ディレクトリーにインストールされると想定しています。

```
root@ov-java:~# ls -lh /opt/intel/openvino-2023.0.1/
total 32K
drwxr-xr-x 3 root root 4.0K Jun 29 10:05 docs
drwxr-xr-x 2 root root 4.0K Jun 28 12:54 install_dependencies
drwxr-xr-x 7 root root 4.0K Jun 28 12:55 python
drwxr-xr-x 5 root root 4.0K Jun 29 14:24 runtime
drwxr-xr-x 5 root root 4.0K Jun 28 12:54 samples
-rwxr-xr-x 1 root root 6.7K Jun 28 11:39 setupvars.sh
drwxr-xr-x 3 root root 4.0K Jun 28 12:54 tools
```

- Spark* 3.2.3 以降

公式リリースのアーカイブ (英語) から Apache Spark* をダウンロードします。ここでは、Apache Spark* バージョン 3.4.1 を使用します。

```
curl -L https://archive.apache.org/dist/spark/spark-3.4.1/spark-3.4.1-bin-hadoop3.tgz --output spark-3.4.1-bin-hadoop3.tgz
```

次に、ダウンロードしたアーカイブを `opt/spark-3.4.1` に展開します。

```
mkdir /opt/spark-3.4.1
sudo tar -xzf spark-3.4.1-bin-hadoop3.tgz -C /opt/spark-3.4.1 --strip-components=1
```

Apache Spark* 環境変数を設定し、次の行を `.bashrc` ファイルに追加してバイナリーを PATH 変数に追加します。

```
vi ~/.bashrc

# Add the following lines at the end of the .bashrc file
export SPARK_HOME=/opt/spark-3.4.1
export PATH=$PATH:$SPARK_HOME/bin
```

次のコマンドを実行して、これらの環境変数を現在のターミナルセッションにロードします。

```
source ~/.bashrc
```

次のコマンドを実行して、インストールが完了したことを確認します。

```
spark-submit --version
```

- Spark* NLP

この例では、Spark* NLP jar が `~/spark-nlp/python/lib` ディレクトリーに配置されていると想定しています。

注: この記事を執筆している時点では、OpenVINO™ 統合を利用するには、最新の変更を加えたソースから Spark* NLP をビルドする必要があります。こちら (英語) の手順を使用して OpenVINO™ jar をビルドし、OpenVINO™ jar ファイルを Spark* NLP プロジェクトのルートの新しい `lib` ディレクトリーにコピーした後、プロジェクトのルートから `sbt assemblyAndCopy` を実行して Spark* NLP jar を `spark-nlp/python/lib` ディレクトリーにコンパイルします。

Spark* NLP パイプラインを実行する手順

Spark* の対話型シェル (Scala REPL) は、Spark* のステートメントをテストし、アプリケーションのプロトタイプを素早く作成し、API を学習するための、強力かつ便利な方法を提供します。このシェルを使用して、保存した BERT モデルを Spark* NLP にインポートし、サンプル・パイプラインを実行する方法を説明します。サンプルコードはこちらから (英語) 入手できます。

次のコマンドを実行して、Spark* NLP jar を使用して Spark* シェルを起動します。`driver-memory` パラメーターを使用して、利用可能なメモリーを増やします。

```
spark-shell --jars ~/spark-nlp/python/lib/sparknlp.jar --driver-memory=4g
```

シェルが起動したら、パイプラインの構築を開始します。

まず、必要な依存ファイルをインポートします。

```
import com.johnsnowlabs.nlp.embeddings.BertEmbeddings
import com.johnsnowlabs.nlp.base.DocumentAssembler
import com.johnsnowlabs.nlp.annotators.Tokenizer
import org.apache.spark.ml.Pipeline
import com.johnsnowlabs.nlp.EmbeddingsFinisher
import org.apache.spark.sql.functions.explode
```

次に、モデルのパスを変数に格納します。

```
val MODEL_PATH="file:/root/models/bert-base-cased"
```

パイプラインのコンポーネントの定義を開始します。`DocumentAssembler` は、すべての Spark* NLP パイプラインのエントリーポイントです。この Spark* トランスフォーマーは、生の入力テキストを `String` 列として読み取り、Spark* でのさらなる処理に必要な **document** タイプに変換します。

```
val document = new
DocumentAssembler().setInputCol("text").setOutputCol("document")
```

入力テキストを処理する準備ができたら、テキストをモデルが理解できるトークンに分割する tokenizer を定義します。

```
val tokenizer = new
Tokenizer().setInputCols(Array("document")).setOutputCol("token")
```

次に、`BERT` モデルを使用してトークンレベルの埋め込みを生成するために使用する Spark* トランスフォーマーである `BertEmbeddings` アノテーターを定義します。アノテーターは、`pretrained` 関数を使用した事前トレーニング済みモデルのダウンロードと使用をサポートしています。

このアノテーターで使用するカスタムモデルをインポートするには、`loadSavedModel` 関数を使用します。デフォルトでは、TF モデルは Tensorflow* を使用して実行されます。OpenVINO™ バックエンドを使用するには、`useOpenvino` オプションを有効にします。

```
val embeddings = BertEmbeddings.loadSavedModel(MODEL_PATH, spark, useOpenvino
= true).setInputCols("token",
"document").setOutputCol("embeddings").setCaseSensitive(true)
```

最後に、EmbeddingsFinisher アノテーターを使用して生成された埋め込みを抽出します。

```
val embeddingsFinisher = new
EmbeddingsFinisher().setInputCols("embeddings").setOutputCols("finished_embed
dings")
```

これらのステージをパイプラインに組み立てるには、次の Spark* ステートメントを使用します。

```
val pipeline = new Pipeline().setStages(Array(document, tokenizer,
embeddings, embeddingsFinisher))
```

これでおしまいです。次のように、入力テキストを指定してパイプラインをテストしてみましょう。

```
val data = Seq("The quick brown fox jumped over the lazy dog.").toDF("text")
```

生の入力データフレームを適合および変換します。

```
val result = pipeline.fit(data).transform(data)
```

生成されたデータフレームの内容を調べてみましょう。

```
result.select(explode($"finished_embeddings") as "result").show(5, 100)
```

```
scala> result.select(explode($"finished_embeddings") as "result").show(5, 100)
+----------------------------------------------------------------------------------------------------+
|                                                                                              result|
+----------------------------------------------------------------------------------------------------+
|[0.26522264, -0.33549413, 0.45074996, -0.18028228, -0.28208363, 0.32488698, 0.31877044, 0.0827080...|
|[-0.0130853355, -0.015156232, 0.16939071, -0.020362701, 0.12682635, 0.028407631, 0.0012666842, 0....|
|[0.28798717, -0.6005762, -0.041134898, -0.053977165, -0.14034356, -0.018826792, 0.09948694, -0.10...|
|[0.28367367, -0.43907073, 0.32618508, 0.3782992, 0.10845218, 0.2833489, -0.05738289, -0.18555321,...|
|[0.15990195, -0.10428578, 0.61077064, -0.1877716, 0.30766734, 0.1913225, -0.07741632, -0.1984541,...|
+----------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

これらの生の埋め込みは、分類や固有表現認識などの下流タスクに使用できます。Scala アプリケーションのコードは、こちらから (英語) 入手できます。

まとめ

この記事では、BERT と OpenVINO™ を使用してトークン埋め込みを生成する Spark* NLP パイプラインを構築するための主要な手順を説明しました。まず、Hugging Face から人気の高い BERT 基本モデルをエクスポートして、Spark* NLP で使用できるように準備しました。次に、DocumentAssembler、Tokenizer、BertEmbeddings、EmbeddingsFinisher アノテーターを使用して、サンプル Spark* NLP パイプラインをセットアップしました。パイプラインは、任意の入力テキストの埋め込みを生成して、下流の NLP タスクを強化できます。OpenVINO™ のニューラル・ネットワークの最適化とハードウェア・アクセラレーションにより、埋め込みの生成を高速化できました。

注: Spark* NLP Python* API を使用して BERT 埋め込みを生成する手順は、このノートブック (英語) を参照してください。ノートブックを実行するには、この機能ブランチ (英語) から jar と wheel ファイルをビルドする必要があることに注意してください。この記事で説明した手順を使用して Spark* NLP jar を構築した後、`sparknlp/python` ディレクトリーから `pip wheel .` を実行して wheel ファイルをビルドします。

関連情報 (英語)

- Apache Spark* の概念の概要
- OpenVINO™ を使用したモデルの最適化
- Spark* NLP 入門
- Spark* NLP Scala API リファレンス
- BERT 埋め込みのサンプル・アプリケーション
- Java* で OpenVINO™ ランタイムを使用したディープラーニング推論
- スピードの必要性: Spark* NLP で OpenVINO™ ランタイムを使用した NLP 推論の高速化

OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピューター・ビジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン / ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html

法務上の注意書き