ホワイトペーパー

MATLAB による
信号処理向けディープラーニング

MathWorks®

はじめに

ディープラーニングネットワークは、長年にわたり画像分類に使用されてきましたが、信号データにも強力なツールです。モデルに取り込むべき信号の特徴が明確でなくても、ディープラーニングネットワークは数学モデルで可能なすべてのことを実行できます。ネットワーク自体が学習プロセスの中で自ら特徴を決めていきます。

入力データが 1 次元信号、時系列データ、さらにはテキストであっても、畳み込みニューラルネットワーク (CNN) などのディープラーニングネットワークにより、新しい方法でデータを処理することが可能です。ディープラーニングネットワークが非専門家への扉を開いたといえるでしょう。CNN を使用して信号を処理するのに、信号処理の専門家になる必要はありません。CNN により正確な結果を非常に早く得ることができます。

このホワイトペーパーでは、ディープラーニングの基礎を概説した後、3 つの信号処理の例をご紹介します。

- 音声コマンド認識
- 残存耐用時間 (RUL) の予測
- 信号のノイズ除去

これらの例を通して、信号処理タスクをよりすばやく行い、より正確な結果を得る上で、MATLAB を使用したディープラーニングがどのように役立つかを解説していきます。

ディープラーニングの基礎

ディープラーニングは機械学習の手法のひとつです。ディープラーニングのモデルは、画像、テキスト、または音声データから直接分類や回帰タスクを実行することを学習します。機械学習または従来の信号処理では、データ内の関連する特徴を手動で選択していました。ディープラーニングを使うことで、データがネットワークを通過するときにモデルが自動的に関連情報について学習および抽象化を行います。

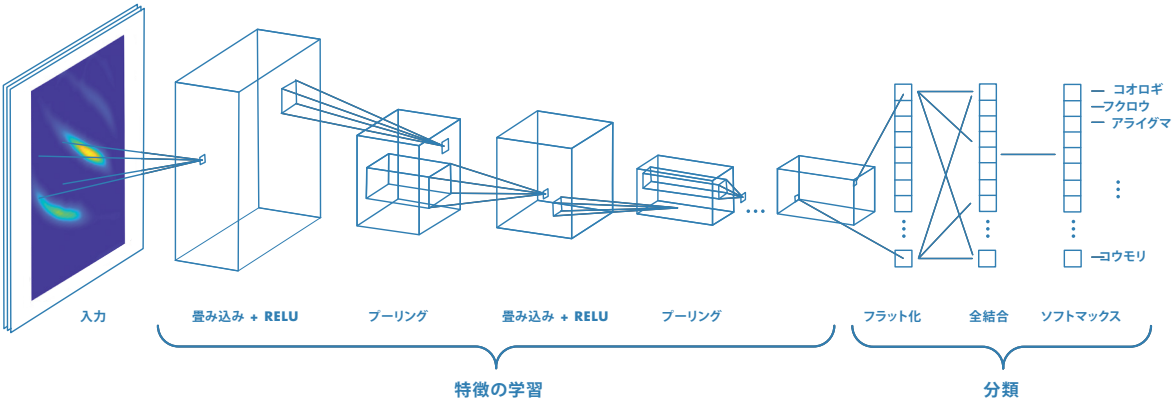
ディープラーニングは通常、ニューラルネットワーク アーキテクチャを使用して実装されます。「ディープ」という用語は、ネットワーク内の層の数を表しています。層が多くなればなるほど、ネットワークはより深くなります。前の層の出力を入力として用いて、ノードまたはニューロン間は相互接続されています。

ディープラーニングネットワーク

最も一般的なディープラーニングネットワークは次の 2 つです。

- 畳み込みニューラルネットワーク (CNN、または ConvNet)
- 長短期記憶 (LSTM)

CNN は、入力層、出力層、そしてその間にある複数の隠れ層で構成されています。最も一般的な 3 つの層は、畳み込み層、活性化層または ReLU 層、プーリング層です。各層が入力データから異なる特徴を検出するように学習しながら、これらの動作は数十または数百の層にわたって繰り返されます。



畳み込み、ReLU、プーリングなどの一般的に使用されるレイヤを含む CNN のアーキテクチャ

LSTM は、シーケンスデータのタイムステップ間にみられる長期的な依存関係を学習できる再帰型ニューラルネットワーク (RNN) の一種です。CNN とは異なり、LSTM は予測間のネットワークの状態を記憶できます。

LSTM ネットワークの主要なコンポーネントは、シーケンス入力層と LSTM 層です。シーケンス入力層は時系列データをネットワークに取り込みます。LSTM 層は、時間経過に伴うシーケンスデータのタイムステップ間の長期的な依存関係を学習します。

ネットワークの選択

一般的に、CNN は入力信号を 1 次元から 2 次元表現に変換して得られる「画像」情報に置き換えることで、信号や時系列の予測に使用されます。

一方、LSTM はネットワークの予測または出力が、記憶を伴う一連のデータ列に基づいている必要がある場合、シーケンスデータと時系列データの分類に適しています。

信号データに関して考慮すべきこと

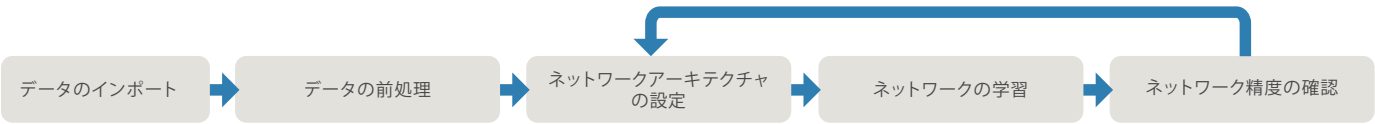
一般的に、信号データを用いたディープラーニングは、画像データを用いたディープラーニングより多くの前処理や特徴抽出を必要とします。生の画像データをネットワークに渡して結果を得られることはよくありますが、同じワークフローで信号データを処理することはほとんど不可能でしょう。それは生の信号データの大部分は、ノイズが多く、可変で、画像データよりも小さいからです。利用可能なデータを慎重に前処理することで、可能な限り最良のモデルを得ることができます。

機械学習を代わりに使うべきでしょうか?

あなたがデータについては理解しているが、データ量が限られている場合は、機械学習を使用して最も関連性の高い特徴を手作業で抽出することで、より良い結果が得られるでしょう。しかしながらこのアプローチは優れた結果を得られる一方で、ディープラーニングよりも多くの信号処理の知識を必要とします。

ワークフロー

ディープラーニングのワークフローの主な手順は次のとおりです。



どのネットワーク構成を使用していても、どのような課題を解決するためにディープラーニングを使用していても、繰り返し作業は常に発生します。期待している精度を得られない場合は、精度を向上させるために戻って設定を変更する必要があります。

これ以降の各例は、この基本的なワークフローに沿っています。

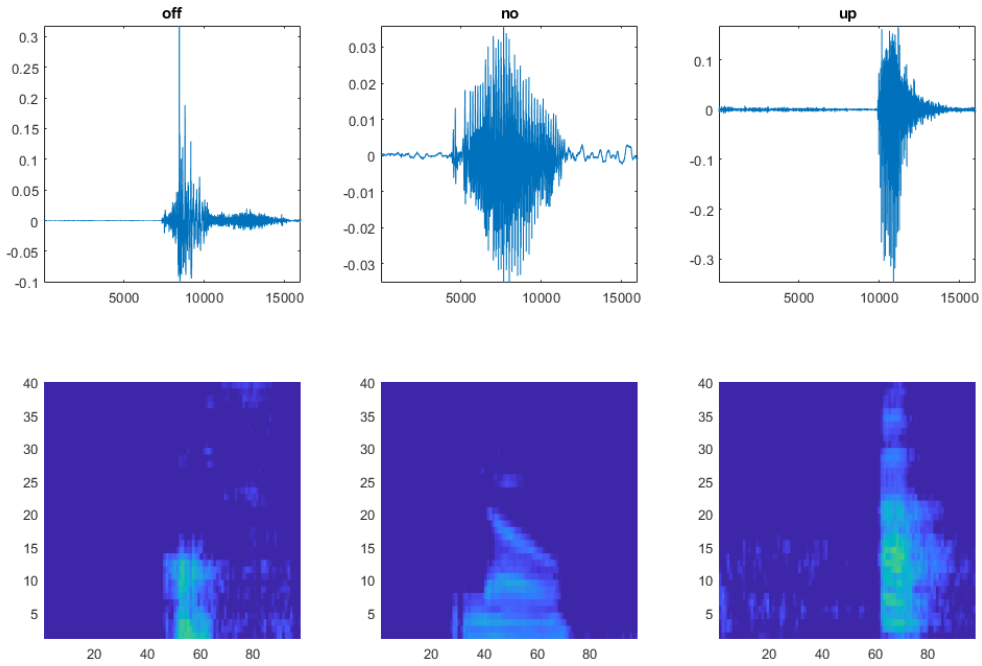
例 1. 音声コマンド認識：音声オーディオファイルの分類

オーディオ録音から個々の音声コマンドを認識し、識別するための単純なディープラーニングモデルの学習を行っていきます。

古典的な信号処理では、信号から関連する特徴 (音声を識別する信号の成分) を識別して抽出し、それらをデコーダモデルに渡す必要があります。システム全体が問題にうまく適応されていない限り、注意深く信号データの前処理を行い (例えばバックグラウンドノイズまたは残響を減らすため)、特定の特徴の組み合わせを選択しても、よい結果が得られないことがあります。

一方、ディープラーニングでは信号から関連する音声特徴量を抽出する必要がありますが、画像分類の問題と同様に取り組むことができます。最初のステップは、オーディオファイルをスペクトログラムに変換することです。スペクトログラムは 1 次元のオーディオファイル内の信号を 2 次元で可視化したものであるため、実際の画像を使用するのと同じように、CNN への入力として使用できます。

結果として、信号の微妙な差異を処理できるロバストなモデルになります。ディープラーニングがなければ、信号処理のエンジニアは信号から関連情報を手作業で識別、発見、処理しなければならないでしょう。



上：オリジナルのオーディオ信号　下：対応するスペクトログラムのオーディオ信号

データのインポート

この例では、高性能数値計算用のオープンソース ソフトウェアライブラリである TensorFlow™ のデータセットを使用します。

MATLAB で試してみる：詳細な例とデータセットへのリンクを入手する

データの準備

CNN の学習を効率的に行えるデータを準備するため、MATLAB のスペクトログラム関数を変化させた **`melSpectrogram`** コマンドを使用して、音声の波形をスペクトログラムに変換します。音声処理は、オーディオ処理の特殊な形であり、特徴 (音声を識別する信号の成分) が特定の周波数に局在化しています。音声に最も関連性の高い周波数領域に CNN の焦点を合わせていくため、周波数はメル尺度を使用します。これにより人間が聞こえるのと同じように感度を分散させることができます。

スペクトログラムは、生の信号データを 2 次元画像として表現するための時間 - 周波数変換の 1 つにすぎません。他の一般的に使用される技術にはウェーブレット変換があります。

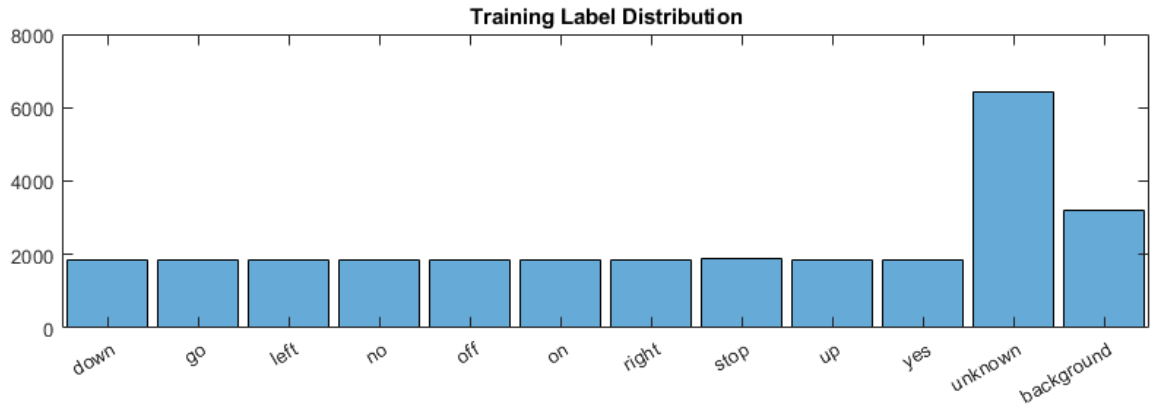
ウェーブレット変換 (スカログラムや連続ウェーブレット変換など) は、特に非周期的な信号の成分に対して、時間分解能を高めたスペクトログラムのような 2 次元表現を生成できます。

ウェーブレットは、ウェーブレット散乱 または「不変散乱畳み込みネットワーク」と呼ばれる自動的な特徴抽出手法でも使用されます。この手法は、CNN の最初の数層で学習された特徴を模倣します。ただし、データから学習する必要がない固定された重みのパターンを使用しているため、ネットワークの複雑さと学習データのサイズの要件が大幅に軽減されます。

次に、データを 3 つのセットに分けます。

- **学習データ：**元の入力データ。ネットワークが特徴を学び理解するために使用します。
- **検証データ：**データの理解を一般化しながら、アルゴリズムが特徴を学習していることを確認するためにネットワークの学習中に使用します。
- **テストデータ：**これまで使用されていないデータ。学習後にネットワークに渡され、結果が偏っていないことを確認するために使用します。

分類したい単語のクラス間で学習データを均等に分配します。



均等に分散された学習データ

誤検出を減らすために、意図したカテゴリと混同される可能性が高い単語のカテゴリを含めるようにします。例えば、意図した単語が「on」である場合、「mom」、「dawn」、「won」のような単語は「unknown」のカテゴリに含まれます。ネットワークはこれらの単語を知っている必要はなく、ただ認識すべき単語ではないというだけです。

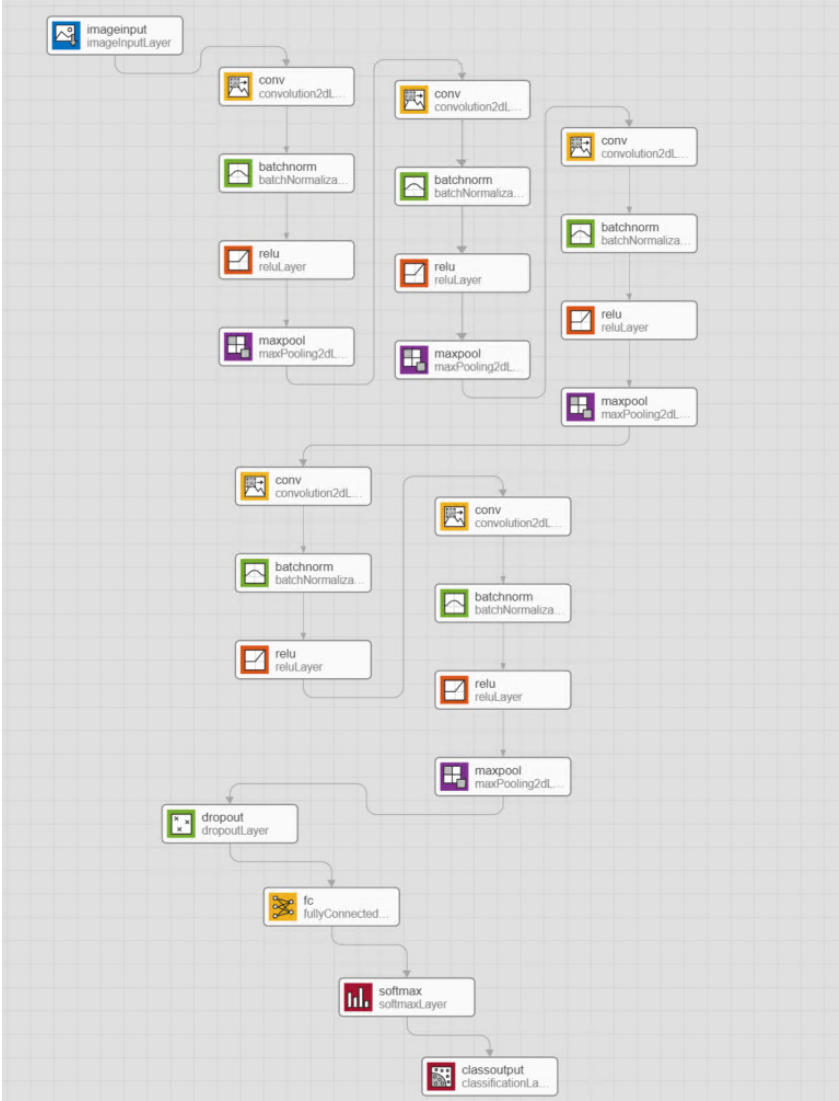
ネットワークアーキテクチャの設定

スペクトログラムを入力として使用しているため、CNN の構造は、画像分類に使用される他のネットワークと似ています。このアプリケーションでは、48 層からなるネットワークを使用します。48 層のネットワークというと複雑に聞こえるかもしれませんが、基本的な構造は単純です。一連の畳み込み層、バッチ正規化層、ReLU 層の繰り返しです。

モデル全体を次のように可視化することができます。

ネットワークアーキテクチャはどのように定義できますか?

一般的なアプローチは、研究論文または他の公開されているソースのネットワークから始めて、必要に応じて修正することです。



畳み込み、バッチ正規化、ReLU の反復回数を決めるのはネットワークの設計者次第ですが、ネットワークが複雑になればなるほど、学習やテストに必要なデータが増えることは覚えておきましょう。

ネットワークの学習

学習を開始する前に、次のような学習オプションを指定します。

- エポック数 (学習データ全体の処理回数)
- 学習率 (学習の速さ)
- プロセッサ (通常は CPU または GPU)

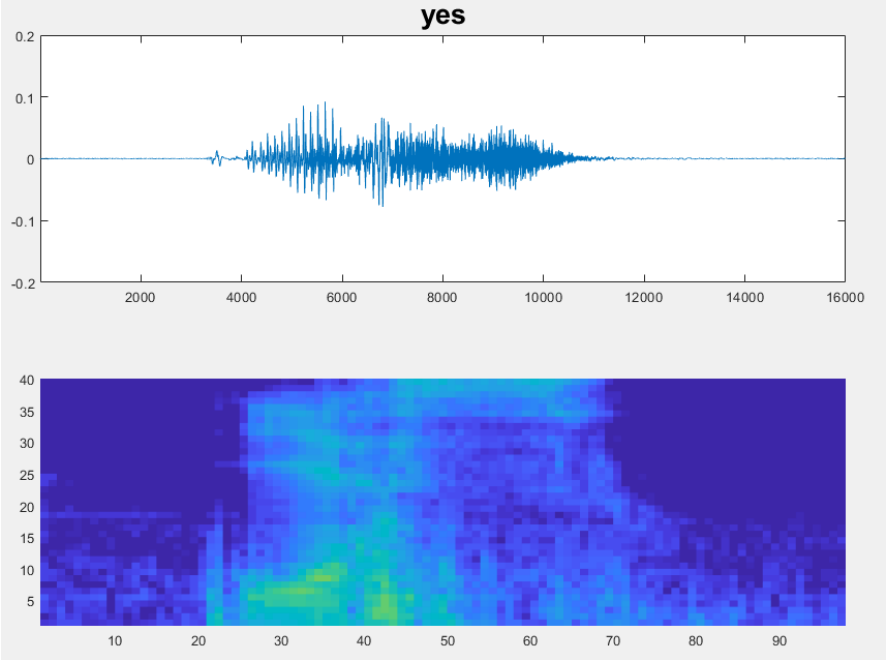
次に、信号のサンプルをネットワークに渡し、学習の進行状況をプロットします。

ネットワークは、サンプル内の固有の特徴を識別し、それに応じて分類することを学習します。スペクトログラムデータを扱う場合、学習プロセスは基本的に画像分類と同じです。ネットワークの最初の層は、色、太さ、線や形の向きなど、スペクトログラムの幅広い特徴を学習します。学習が進むにつれて、ネットワークは個々の信号のより詳細な特徴を識別していきます。

ネットワーク精度の確認

モデルの学習が終わると、指定したエポック数を完了したか、または指定した停止点に達したため、入力画像 (スペクトログラム) が適切なカテゴリに分類されます。

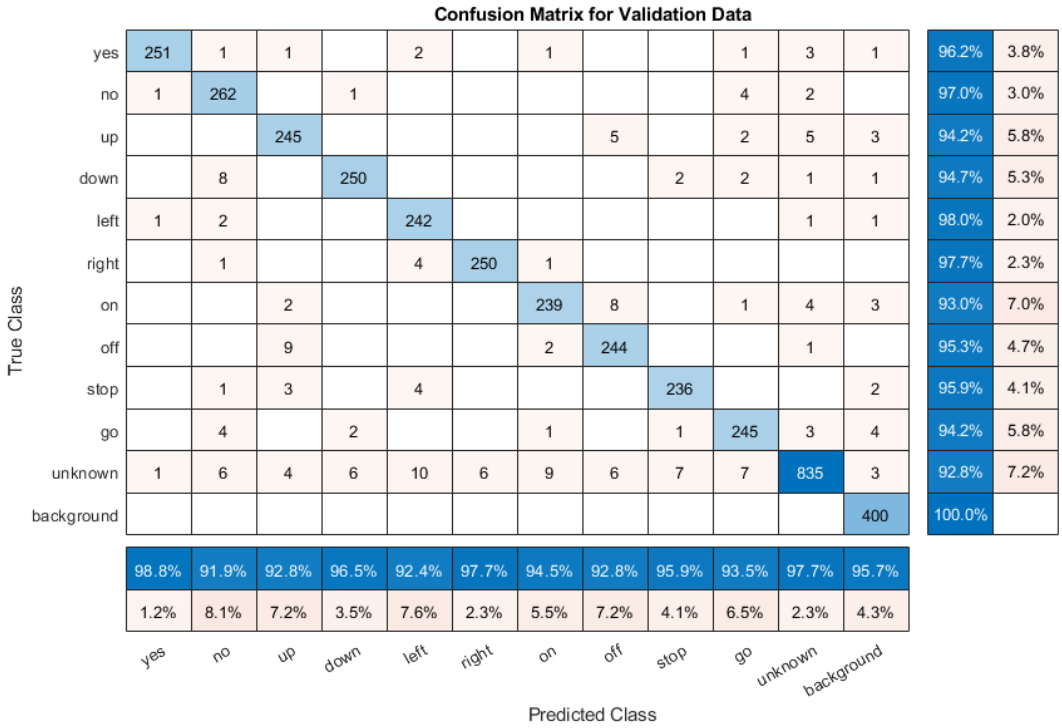
結果をテストするために、ネットワーク予測と対応するオーディオクリップを比較します。この検証セットの精度は約 96% です。



「yes」として分類されている単語のプロットとスペクトログラム

この結果は単純な音声認識には許容できるかもしれませんが、たとえばセキュリティシステムや音声起動デバイスに実装する場合は、より高い精度が必要となるでしょう。

ここからがディープラーニングという手法の価値が見え始めるところです。ディープラーニングには、より高い精度を達成するためのさまざまなオプションがあります。ネットワークに学習サンプルを追加するだけで結果を改善することができます。ネットワークがどこで誤った予測をしているかは、さらなる洞察を得るために、混同行列を可視化することができます。



混同行列は、ネットワークの予測クラス (単語)と実際のクラスを比較したものです。

青は正しい予測を示し、ベージュは誤った予測を示します。

ディープラーニングでは、必要な精度が得られるまで続けることができます。従来の信号処理では、信号の微妙な違いを理解するにはかなりの専門知識が必要となるでしょう。

例 2. 残存耐用時間の予測：シーケンス間回帰

この例では、エンジンが故障するまでの残り時間、すなわち残存耐用時間 (RUL) を予測していきます。RUL は、エンジンメンテナンスのタイミングを調整し、計画外の遅延を回避するための重要な指標です。

多くのディープラーニングネットワークはクラスやカテゴリを予測しますが、ここで期待している出力は数値です。エンジンが故障するまでのサイクル数となるので、回帰問題となります。

故障までの時間データを使用して RUL を計算します。これはシステムの相対的な健全性に応じてセンサーの値がどのように変化するかを示します。

ディープラーニングを使わない場合、この問題に取り組む際のワークフローは次のようになります。

1. どのセンサーがシステムの正常性を示す最適な指標であるかを選択する。
2. これらのセンサーの組み合わせを使用して指標を導き出す。
3. システムに故障が発生する危険性があるか確認するために、データを継続的に追跡する。
4. 故障の確率または故障までの時間を返すモデルを近似する。

最初のステップである、センサーを選択する (特徴選択)ことは、簡単な作業ではありません。1 つのシステムに 50 個以上のセンサーが搭載されていることはよくありますが、システムが故障に近づくにつれて、すべてのセンサーに変化の兆候が見られるわけではないからです。しかしディープラーニングでは、この特徴抽出はモデル自らが行います。つまり、どのセンサーが最適な予測子であるのかは、ネットワークが決めるのです。

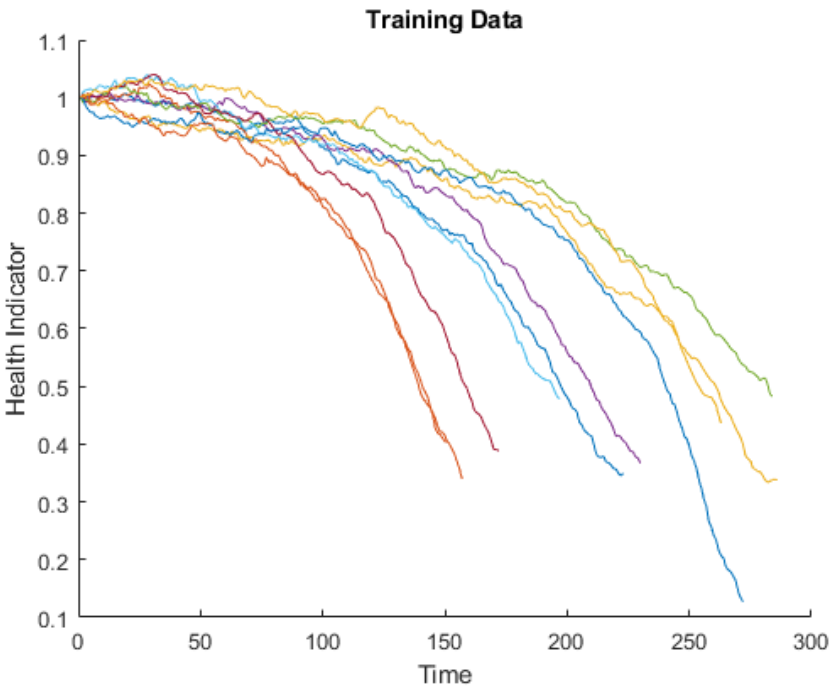
データのインポート

まずは、NASA Prognostic Data Repository の Turbofan Engine Degradation Simulation のデータセットから始めます。

MATLAB で試してみる：詳細な例とデータセットへのリンクを入手する

このデータセットには、学習用とテスト用のそれぞれの観測値と、検証用の実際の RUL 値のベクトルが含まれています。

各時系列は単一のエンジンを表します。各エンジンは時系列の開始時は正常に動作し、時系列の途中のある時点で故障が発生します。学習セットでは、システム障害が発生するまで故障の規模が大きくなります。



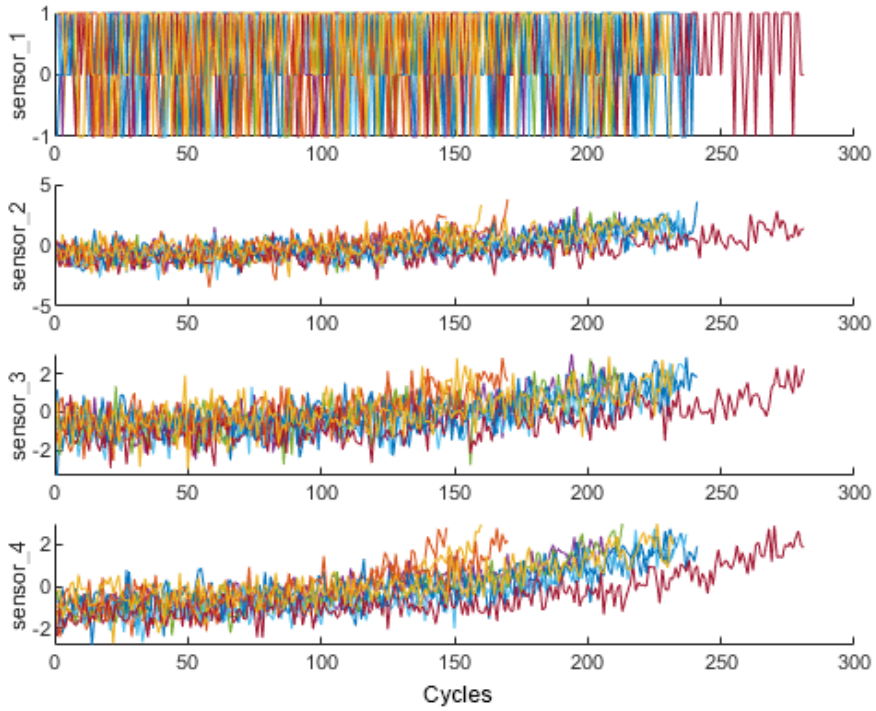
0 から 300 サイクルまでの学習データのビュー

データの準備

正確な予測を確実に行うために、ネットワークに可能な限り最適なデータが提供されるようにします。データを単純化して整理するために、次のことを行います。

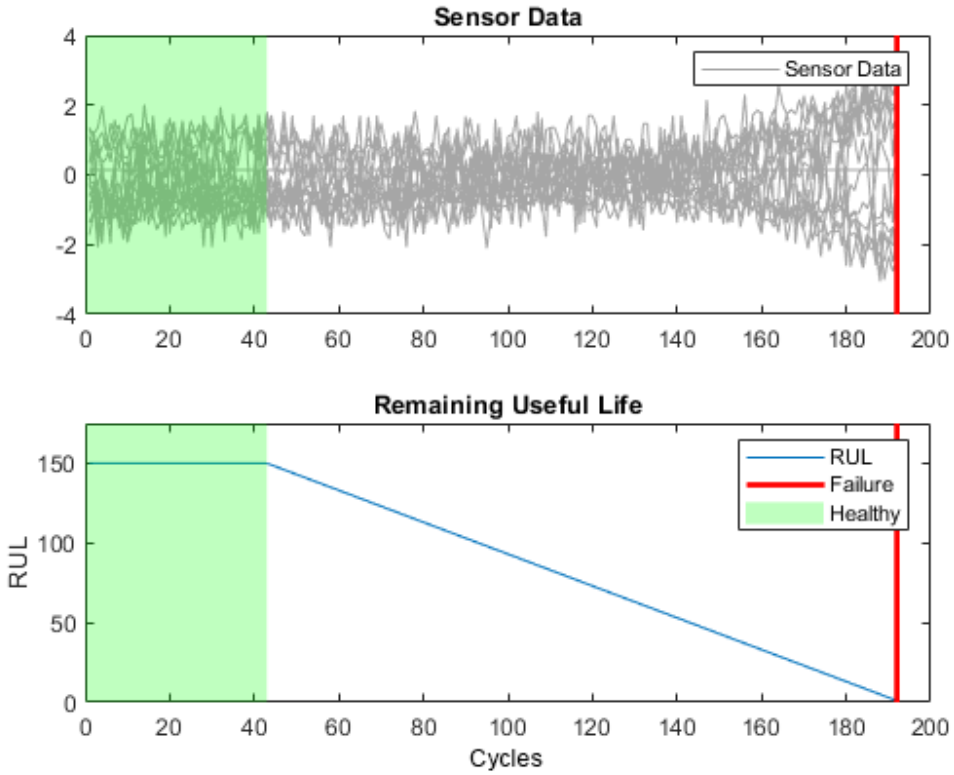
定数値を持つ信号を削除する。すべてのタイムステップで一定のままである信号は、学習済みモデルの精度には寄与しない可能性があるため、最小値と最大値が同じ信号は削除することができます。

学習予測子を正規化する。異なる信号は異なる範囲にまたがるので、データを正規化する必要があります。この例では、ゼロ平均を仮定し、-4 から 4 の間でスケーリングします。すべての観測値の平均と標準偏差を計算するために、シーケンスデータを水平方向に連結します。

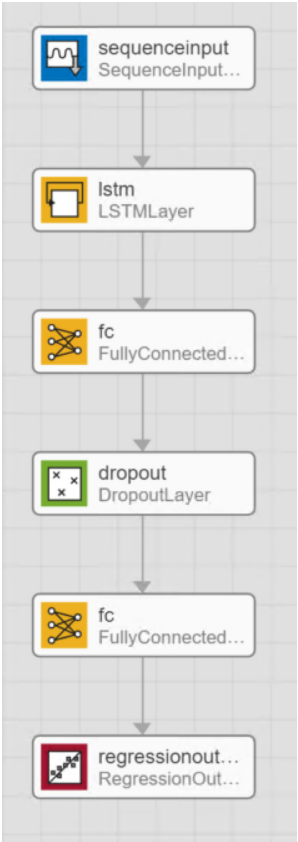


正規化後のセンサー 1-4 のデータ

応答をクリップする。応答をクリップすることで、ネットワークはシステム障害が発生し始める時点に集中します。下の図は、1 つのエンジンのセンサーデータ (上図) と各ポイントのセンサーデータの残存耐用時間 (下図) を示しています。故障から 150 サイクル以上経過した信号の初期値 (緑色で表示) の残存耐用時間の値は 150 に制限されているため、ネットワークはある程度保守的になります。



上：1 つのエンジンのセンサーデータ。下：各点におけるセンサーデータの RUL。



ネットワークアーキテクチャの設定

LSTM はシーケンスデータのタイムステップ間の依存関係を学習できるため、この種のアプリケーションに適しています。センサーは相互接続されていることが多く、2 サイクル前に発生したイベントはシステムの将来のパフォーマンスに影響を与える可能性があります。

隠れユニットが 200 個ある層と、ドロップアウト確率が 0.5 のドロップアウト層で構成された LSTM ネットワークを定義します。

ドロップアウト確率は、過適合を防ぐために使用します。ネットワークはこの確率に基づいて、データをランダムにスキップし、ネットワークが学習セットを記憶してしまうことを抑制します。隠れユニット数は通常数百に及びます。隠れユニットをいくつ含めるか決めることはトレードオフであるといえます。隠れユニットが少なすぎると、モデルの学習に十分な記憶容量がありません。逆に多すぎると、ネットワークが過適合する可能性があります。

ネットワークの学習

まずは、2 つの学習オプションを設定することから始めましょう。

- ソルバー **'adam'** を使用して、サイズ 20 のミニバッチで 60 エポック
- 学習率 0.01

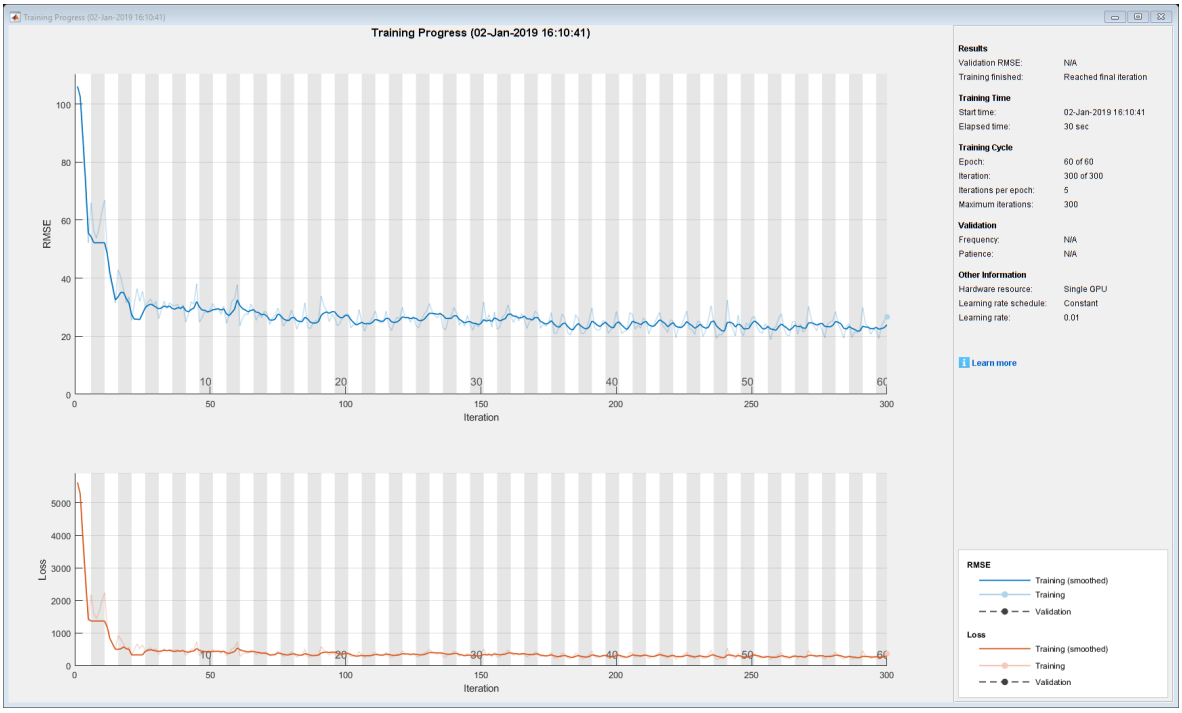
これらの設定では、ネットワークの学習には約 2 分かかります。ネットワークが稼働している間の学習の進行状況をプロットします。

ソルバーとは？

ソルバーは、ニューラルネットワークを最適化するアルゴリズムです。

'adam' (adaptive moment estimation、適応モーメント推定) は LSTM ネットワークで使用される最も一般的なソルバーであり、しばしばデフォルト設定で使用することができます。**'adam'** は、パラメータの勾配と勾配の二乗値のそれぞれについて、移動平均を維持します。

'adam'や他のソルバーに関する詳細情報をご覧ください。（**'sgdm'** と **'rmsprop'** を含む）



RUL モデルの学習結果

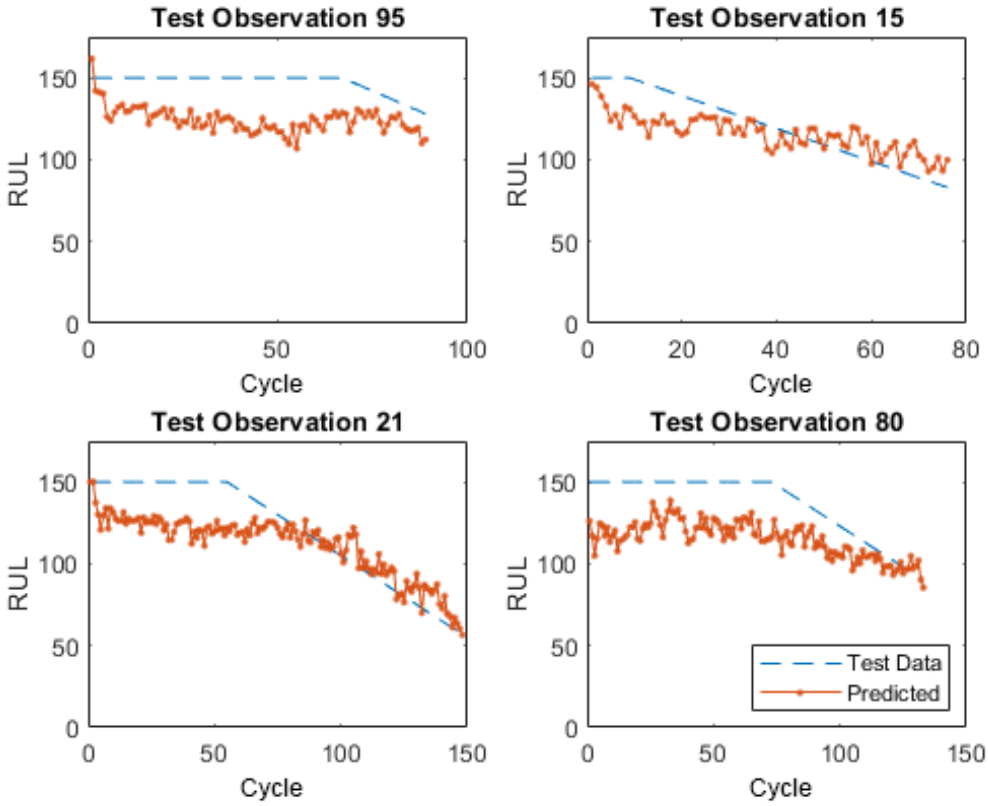
RUL の予測は回帰問題であるため、上のプロットは精度ではなく二乗平均平方根誤差 (RMSE) の進行を示しています。数値が小さいほど、予測される故障までの時間は実際の値に近くなります。

ネットワーク精度の確認

ネットワークの学習が終了したら、テストデータで検証します。

この LSTM ネットワークは、1 タイムステップずつ部分シーケンスについて予測を行います。各サイクルで、ネットワークはその内部状態を更新し、故障が発生するまでの時間、つまり RUL 値を予測します。

ネットワークのパフォーマンスを可視化するために、4 つのテストエンジンからデータを選択し、予測に対して実際の残存耐用時間をプロットします。



実際の RUL (青の点線) と推定 RUL (オレンジの線) を比較する 4 つのエンジン (100、52、67、および 45) のモデル予測。

この LSTM は、エンジン 100 と 45 についてはある程度正確な故障までの推定時間を示していますが、エンジン 52 と 67 については精度が劣ります。この結果は、それらのエンジン性能を表す学習データがさらに必要であることを示している可能性があります。あるいは、ネットワークの設定を調整して、パフォーマンスが向上するかどうかを確認することもできます。

この例では、単純な LSTM ネットワークを設計、学習、テストし、そのネットワークを使用して残存耐用時間を予測しました。従来の信号処理の代わりにディープラーニングを用いることにより、特徴抽出における困難で時間のかかるタスクを省略することができました。

例 3. 全結合ニューラルネットワークを用いた音声のノイズ除去

信号の品質と明瞭度を高めながら、音声信号から洗濯機のノイズを取り除いていきます。エンジニアはデータ品質を改善するために信号のノイズ除去を行います。たとえば、クラウドベースの音声アシスタントエンジンによりよい録音を渡したり、ECG の S/N 比を向上させる場合です。

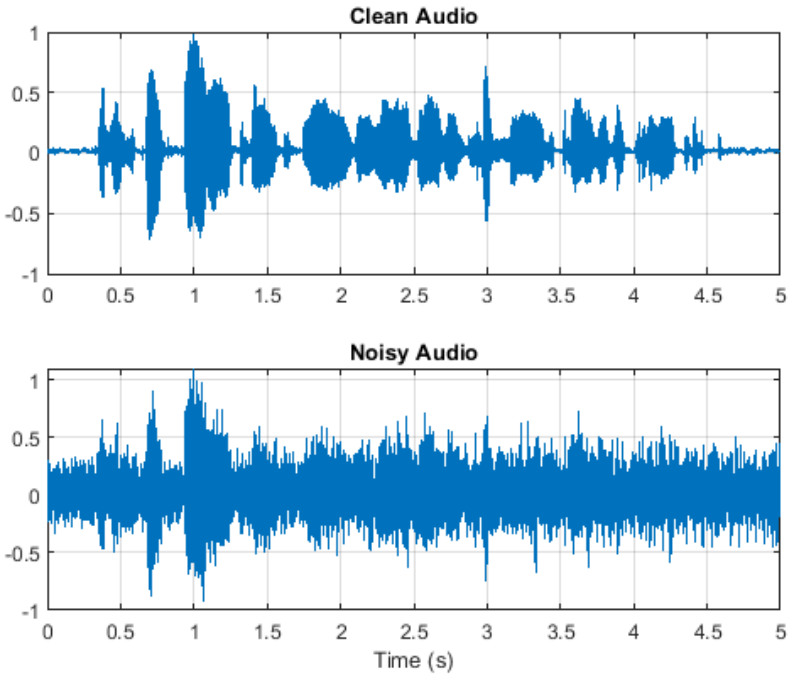
古典的な信号処理技術は、ノイズを除去するために使われていました (現在でも使われています) 。これらのアプローチ (スペクトル減算、ノイズゲーティングなど) の課題は、よい信号の品質が意図せずに除去され、元の信号の一部の情報が失われてしまうことです。

この例では、音声のノイズ除去にディープラーニングを使用することで、出力における望ましくない欠点を最小にしながら、音声信号から洗濯機のノイズを除去することができます。その後、ネットワークの出力を利用して、他の似たようなノイズの多い環境においてもノイズを除去することができます。

データのインポート

クリーンな音声信号を読み取り、このクリーンな信号からノイズの多い信号を作成していきます。これは、録音済みの wav ファイルから洗濯機のノイズを信号に加えることで実現できます。このプロセスにより、真のクリーンな信号と最終的なノイズ除去された信号を比較することで、アルゴリズムがどの程度うまく機能するかを確認できます。

MATLAB で試してみる：例題ファイルと詳細な例題を入手する



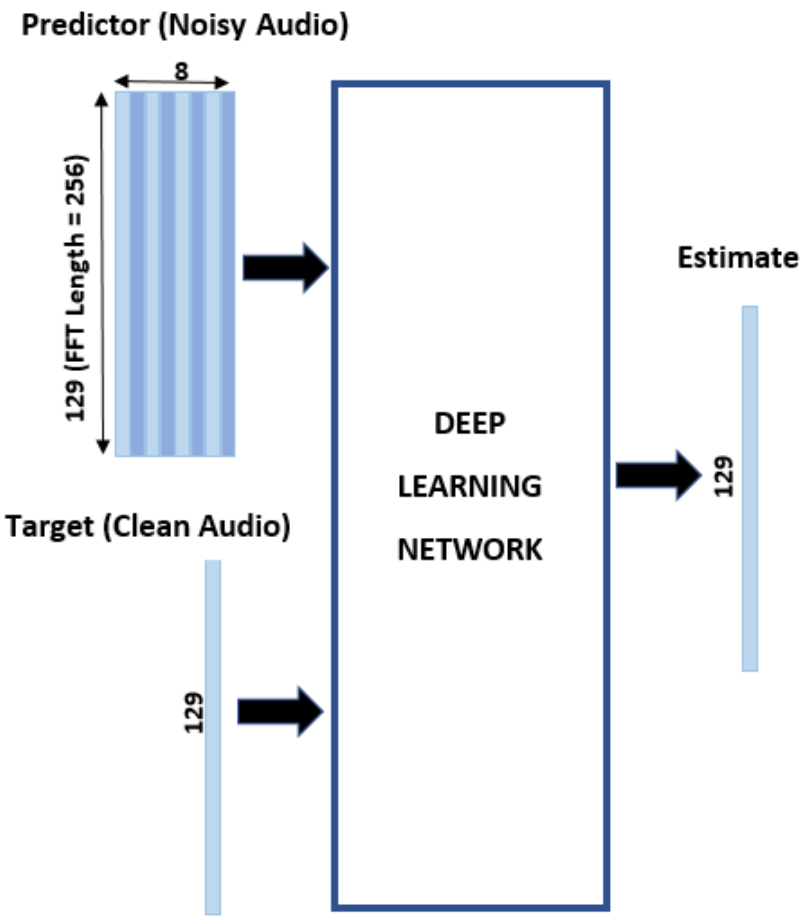
元のオーディオとノイズを加えたオーディオのプロット

データの準備

学習済みアルゴリズムをテストするために、ノイズの多い音声データを他に確保しておきます。

この例では、短時間フーリエ変換 (STFT) を使用して信号を 2 次元信号に変換します。この方法には、信号をオーバーラップしたセグメントに分割し、各セグメントの高速フーリエ変換 (FFT) を計算することが含まれます。ストリーミング信号をセグメント単位で取得することで、信号の取得中に予測を確認できます。入力時に、ネットワークに時間内に「メモリ」を提供するために 8 つのセグメントを蓄積します。これにより現在の信号セグメントだけでなく、現在の信号セグメントと密接に関連するいくつかの過去の信号も見ることができます。出力では、ネットワークは一度に 1 つのセグメントを予測するように求められます。

プロセスは以下のとおりです。



予測子の入力は、8 つの連続したノイズの多い STFT ベクトルからなるので、各 STFT 出力推定値はさらなるコンテキストのために、現在のノイズの多い STFT と 7 つ前のノイズの多い STFT ベクトルに基づいて計算される。

STFT のターゲットと予測子

短時間フーリエ変換 (STFT) を使用してオーディオを周波数領域に変換するときは、いくつかのシステムパラメーターを適用する必要があります。

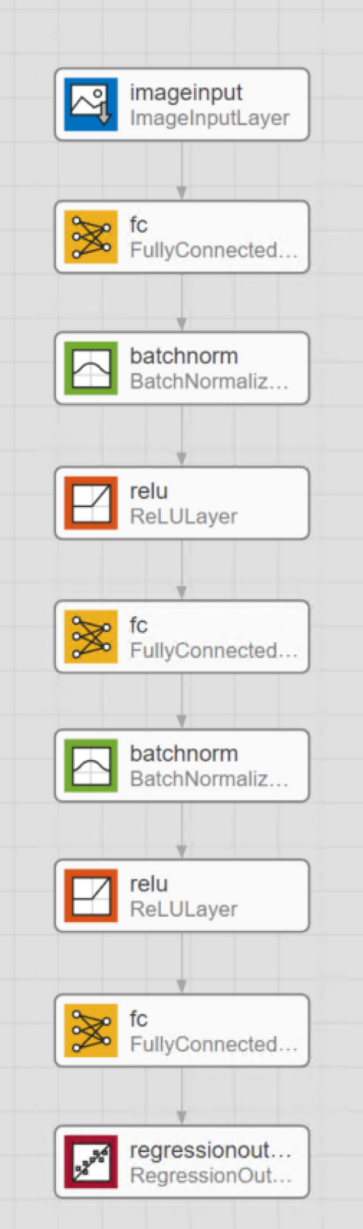
ウィンドウの長さ。ウィンドウの長さが長いほど、解像度と複雑さが増します。この例の 256 のウィンドウの長さは、利用可能なデータを考慮すると、高周波数分解能と低い計算量との間の妥当な妥協点といえます。

オーバーラップ。オーバーラップを 75% に設定して、ネットワークにより正確な時間分解能を与え、より多くの情報が同じ時間内にネットワークを通過できるようにします。

入力サンプルレート。この入力データは 48 KHz のレートと記録されました。

周波数サンプルレート。入力データを 8 KHz にダウンサンプリングすると、人間の音声には十分に機能する、よりコンパクトなネットワークが得られます。

セグメント数。この例では、現在のノイズの多い STFT と、ネットワークが学習に用いるコンテキストとして使用する 7 つ前のセグメントを含む 8 つのセグメントを選択しました。各セグメントは 75% 重複しているため、これは約 3 回の全セグメント時間に相当します。



ネットワークアーキテクチャを構成

最初の例で使用したのと同じ基本的なワークフローに従います。音声コマンド認識の例では一連の畳み込み層を使用しましたが、この例では全結合層を使用しています。

全結合層は、前の層からのすべての活性化に接続されています。全結合層は 2 次元の空間的な特徴を、1 次元のベクトルに「フラット化」します。

バッチ正規化層は、出力の平均と標準偏差を正規化します。パラメーターや前の層が学習段階の途中で変化すると、現在の層は新しい入力分布に対して再調整しなければならず、時間がかかります。全結合層の学習速度を上げ、ネットワークの初期化に対する感度を下げるには、全結合層の間にあるバッチ正規化層と、ReLu 層などの非線形性を利用します。

ReLU 層は、各入力に対してしきい値操作を実行します。そこでは、入力値がゼロよりも小さい値はゼロに設定されます。

そして回帰層で終わります。ここで、半二乗平均誤差の損失が確認できます。回帰層は、角度、距離、またはこの例の場合はノイズ除去された信号など、連続データに関する予測を出力します。

ネットワークの学習

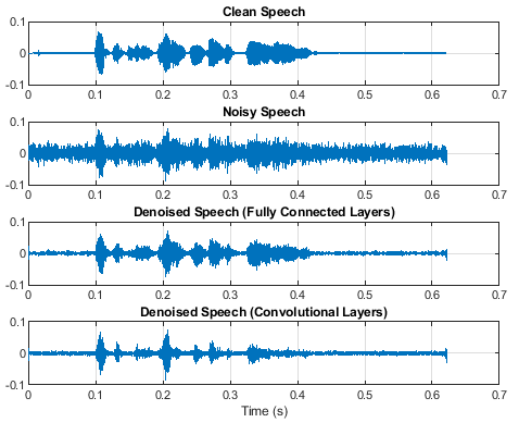
次に、学習オプションを次のように設定します。

- 最大エポック数：3 (ネットワークは学習データを 3 回通過します)
- ミニバッチサイズ：128 (ネットワークは一度に 128 の学習信号を扱います)
- `'Plots'`：`'training-progress'` (学習プロットが表示されます)
- `'Shuffle'`： `'every-epoch'`
- `'LearnRateSchedule'`：`'piecewise'` (エポックが経過するたびに、学習率を指定された係数 (0.9) だけ減少させます)

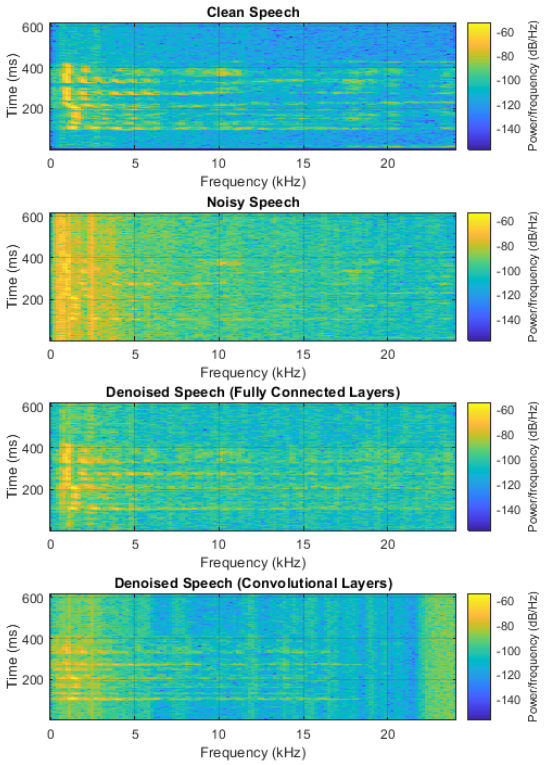
このサイズの学習セットの実行には、数分かかることがあります。

ネットワーク精度の確認

ネットワークの学習が終わったら、テスト用に取っておいたノイズを含んだ信号をネットワークに渡して結果を可視化します。信号はプロットやスペクトログラムとして可視化が可能です。



時間領域プロットは、バックグラウンドノイズレベルが大幅に減少したことを示しています。



スペクトログラムを見ると、モデルによって異なる周波数でどのように振舞うか、詳細にわかります。たとえば、音声スペクトルの大部分が集中している低周波数領域で、ノイズ除去が特に効果的であったことがわかります。

これはオーディオデータであるので、音声を聴いて主観的に品質を比較し、結果が満足できるものかどうかを確認できます。

この学習済みのネットワークはリアルタイムで使用できます。ここでは、音声のノイズ除去を行う、精度が高く再利用可能なモデルを作成することができました。

MATLAB で **`speechDenoisingRealtimeApp`** を実行し、ストリーミングリアルタイムバージョンのノイズ除去ネットワークをシミュレーションします。

まとめ

信号処理向けのディープラーニングにより、エンジニアは (1) 精度が高く、再利用可能なモデルを作成すること、(2) 信号の特徴抽出・選択を行うための専門知識を必要とせずに、モデルの性能を向上させるための変更を加えること、が容易になりました。

音声認識と信号のノイズ除去の例では、CNN と全結合ネットワークでそれぞれ使用するために音声データを準備、変換する方法をご紹介しました。モデルのパフォーマンスを可視化し、信号処理に関する豊富な専門知識がなくても変更を加えられる方法を検討しました。

また、RUL の例では、LSTM を使用し、ネットワークに可能な限り最良のデータを提供するためにセンサーデータを前処理し、手動の特徴抽出と選択を行わずに動作するモデルを作成しました。

その他のリソース

MATLAB によるディープラーニング

信号と音声向けディープラーニング