백서

MATLAB을 활용한 신호 처리 딥러닝

개요

지금까지는 주로 딥러닝 네트워크를 이미지 분류에 활용해왔지만, 딥러닝 네트워크는 신호 처리에도 활용할 수 있는 강력한 도구입니다. 어떤 신호 특징을 모델에 넣을지 고민하지 않아도, 딥러닝 네트워크를 이용하면 수학 모델을 완벽히 대체하실 수 있습니다. 네트워크 자체가 학습 과정에서 그러한 신호 특징을 결정합니다.

입력 데이터가 1차원 신호, 시계열 데이터 또는 심지어 텍스트인 경우에도 컨벌루션 뉴럴 네트워크(CNN)와 같은 딥러닝 네트워크를 이용하면 데이터를 새로운 방식으로 처리할 수 있게 됩니다. 비전문가여도 문제없습니다. CNN을 통해 신호를 실행하기 위해 신호 처리 전문가가 될 필요가 없고, 아주 빠르게 정확한 결과를 산출할 수 있습니다.

이 백서에서는 먼저 딥러닝의 기초를 알아보고, 세 가지 신호 처리 사례를 살펴보겠습니다.

- 음성 명령 인식
- RUL(잔여 수명) 예측
- 신호 노이즈 제거

위의 사례에서는 MATLAB®을 활용한 딥러닝이 어떻게 더 빠르고 정확하게 신호 처리 작업을 할 수 있는지 알아보겠습니다.

딥러닝 기본 사항

딥러닝은 일종의 머신 러닝입니다. 딥러닝에서는 모델이 이미지, 텍스트 또는 사운드를 직접 학습하여 분류와 회귀 작업을 수행하게 됩니다. 머신 러닝이나 과거의 신호 처리에서는 데이터에 있는 관련 특징을 수작업으로 선택해야 했습니다. 딥러닝을 이용하면 데이터가 네트워크를 통과하면서 모델이 관련 정보를 자동으로 학습하고 요약합니다.

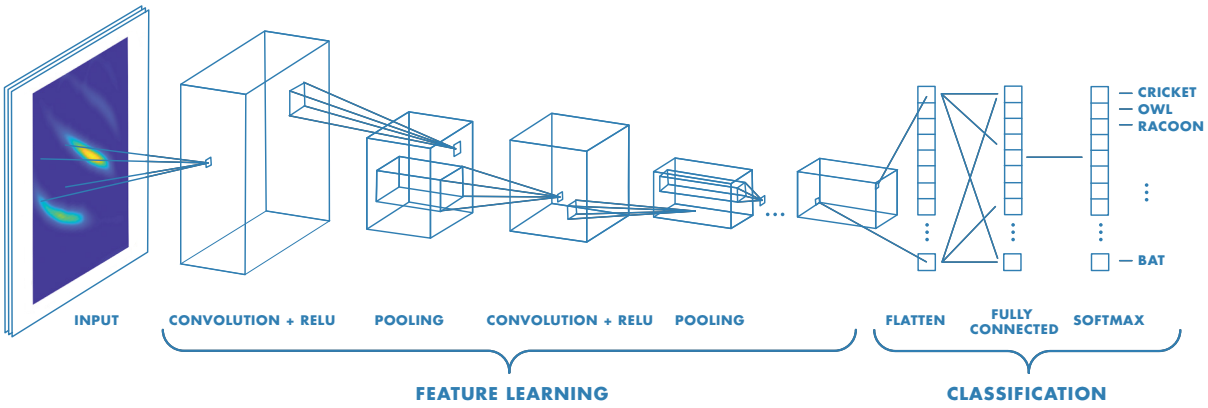
일반적으로 딥러닝은 뉴럴 네트워크 아키텍처를 사용하여 구현합니다. "딥(deep)"이라는 용어는 네트워크의 계층 수를 나타냅니다. 계층이 많을수록 네트워크가 깊어집니다. 계층은 이전 계층의 출력을 입력으로 사용하여 노드 또는 뉴런을 통해 각각의 은닉 계층과 상호 연결됩니다.

딥러닝 네트워크

가장 널리 사용되는 딥러닝 네트워크는 다음과 같습니다.

- 컨벌루션 뉴럴 네트워크(CNN 또는 ConvNet)
- 장기 단기 메모리(LSTM)

CNN은 입력 계층, 출력 계층 및 두 계층 사이의 여러 은닉 계층으로 구성됩니다. 가장 널리 사용되는 계층 세 가지는 컨볼루션, 액티베이션 또는 ReLU, 풀링입니다. 이러한 세 가지 연산이 수십 개 또는 수백 개의 계층에서 반복되며, 각 계층은 입력 데이터에 있는 다른 특징들을 검출하는 방법을 학습하게 됩니다.



CNN 아키텍처는 컨볼루션, ReLU, 풀링처럼 널리 사용되는 계층으로 구성되어 있습니다.

LSTM은 시퀀스 데이터의 시간 스텝들 사이의 장기적 의존성을 학습할 수 있는 일종의 순환 신경망(RNN)입니다. CNN과는 달리, LSTM은 예측 간에 네트워크의 상태를 기억할 수 있습니다.

LSTM 네트워크의 핵심 컴포넌트는 시퀀스 입력 계층과 LSTM 계층입니다. 시퀀스 입력 계층은 시계열 데이터를 네트워크에 입력합니다. LSTM 계층은 시간의 흐름에 따른 시퀀스 데이터 시간 스텝들 사이의 장기적인 의존성을 학습합니다.

네트워크 선택하기

CNN은 일반적으로 신호와 시계열 예측에 사용하며, 이때 입력 신호를 1차원에서 2차원 표현으로 변환시켜, 입력 신호가 '이미지'로 변환됩니다.

LSTM은 시퀀스 데이터와 시계열 데이터를 분류할 때 적당하고, 이때 네트워크 예측 또는 네트워크 출력은 이미 기억되어 있는 데이터 포인트 시퀀스를 기초로 해야 합니다.

신호 데이터를 다룰 때 고려할 사항

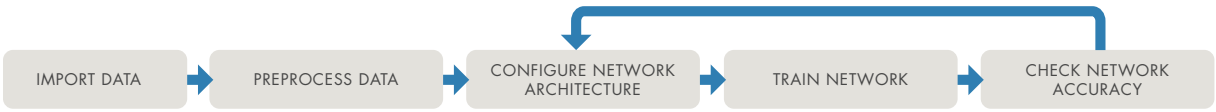
신호 데이터를 이용하여 딥러닝을 할 때는 일반적으로 이미지 데이터를 이용하여 딥러닝을 할 때보다 전처리와 특징 추출을 더 많이 해야 합니다. 보통은 원시 이미지를 네트워크에 넣어서 결과를 얻을 수 있지만, 신호 데이터를 다룰 때는 그러한 워크플로를 사용하는 경우가 매우 드뭅니다. 원시 신호 데이터는 보통 노이즈가 많고 가변적이며, 이미지 데이터보다 크기가 작습니다. 가용한 데이터를 세심하게 전처리해야 최상의 모델을 얻을 수 있습니다.

머신 러닝을 대신하여 사용해야 할까요?

데이터를 이해하고 있지만 양이 제한되어 있는 경우 머신 러닝을 이용하고 수작업으로 가장 관련 있는 특징을 추출하면 더 나은 결과를 얻을 수 있습니다. 결과가 인상적이라 할지라도 이러한 접근은 딥러닝에 비해 더 많은 신호 처리 지식이 요구됩니다.

워크플로

딥러닝 워크플로의 주요한 단계는 다음과 같습니다.



사용하는 네트워크 설정 또는 딥러닝으로 해결하려는 과제와는 무관하게, 과정이 항상 반복적으로 이루어집니다. 바라는 것보다 네트워크의 정확도가 낮을 경우에는 되돌아가 설정을 변경해서 정확도를 높일 수 있는지 알아봅니다.

다음의 사례는 이러한 기본적인 워크플로를 따르고 있습니다.

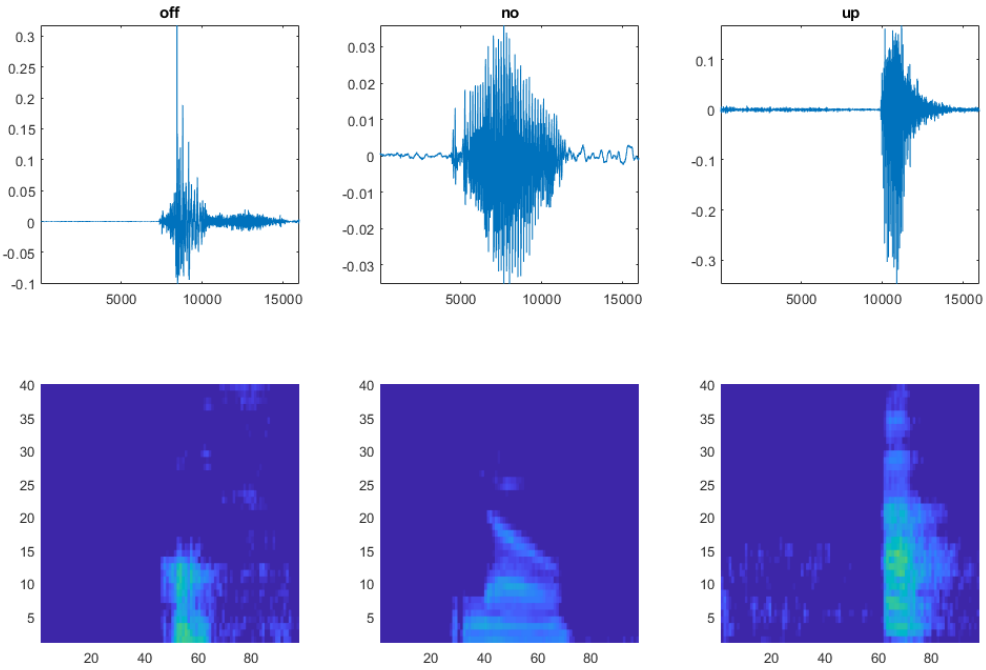
사례 1. 음성 명령 인식: 음성 오디오 파일 분류하기

간단한 딥러닝 모델을 학습시켜서 오디오 레코딩에 있는 개별 음성 명령들을 인식하고 식별하려고 합니다.

기존 신호 처리에서는 신호에서 관련 특징(음성을 식별하는 데 사용되는 신호 요소)을 식별하고 추출해서 디코더 모델로 보내야 했습니다. 예를 들어 배경 노이즈나 반향을 줄이는 등 신호를 세심하게 전처리하고 특정한 특징들을 추출해도, 과제에 맞게 전체 시스템을 완벽하게 적응시키지 않으면 여전히 좋지 않은 결과가 산출될 수 있었습니다.

딥러닝을 사용해도 여전히 관련 음성 특징을 신호에서 추출해야 하지만, 그다음에는 이미지 분류 문제처럼 접근할 수 있습니다. 첫 단계는 오디오 파일을 스펙트로그램으로 변환하는 단계입니다. 그리고 스펙트로그램은 1차원 오디오 파일을 2차원으로 보여주는 것이기 때문에, 실제 이미지를 사용하는 것처럼 그것을 CNN에 입력값으로 사용할 수 있습니다.

신호의 뉘앙스를 처리할 수 있는 강건한 모델을 얻게 됩니다. 딥러닝이 없다면 신호 처리 엔지니어는 수작업으로 신호에서 관련 정보를 식별하고 확인하고, 처리해야 할 것입니다.



위: 원본 오디오 신호 아래: 오디오 신호에 대응되는 스펙트로그램

데이터 가져오기

이 사례에서는 고성능 수치연산에 사용되는 오픈소스 소프트웨어 라이브러리인 TensorFlow™에서 나온 데이터 세트를 이용합니다.

MATLAB 실습: *더 많은 사례와 데이터 세트 링크*

데이터 준비하기

CNN을 효율적으로 학습시키기 위한 데이터를 준비하기 위해 MATLAB에 있는 더 간단한 스펙트로그램 함수를 변형한 **`melSpectrogram`** 명령을 이용하여 음성 파형을 스펙트로그램으로 변환합니다. 음성 처리는 오디오 처리의 특수한 형태로, 특징(음성을 식별하는 데 사용되는 신호의 컴포넌트)이 특정한 주파수에 몰려 있습니다. 음성과 가장 관련성이 높은 주파수 영역에 CNN이 집중하기를 원하기 때문에, 주파수에 대하여 Mel 스페이싱을 사용합니다. 이렇게 하면 인간이 듣는 것과 유사하게 민감도를 분산시키게 됩니다.

스펙트로그램은 원시 신호 데이터를 2차원 이미지로 표현하기 위한 많은 시간-주파수 변환 중 하나일 뿐입니다. 널리 사용되는 다른 기법에는 웨이블릿 변환이 있습니다.

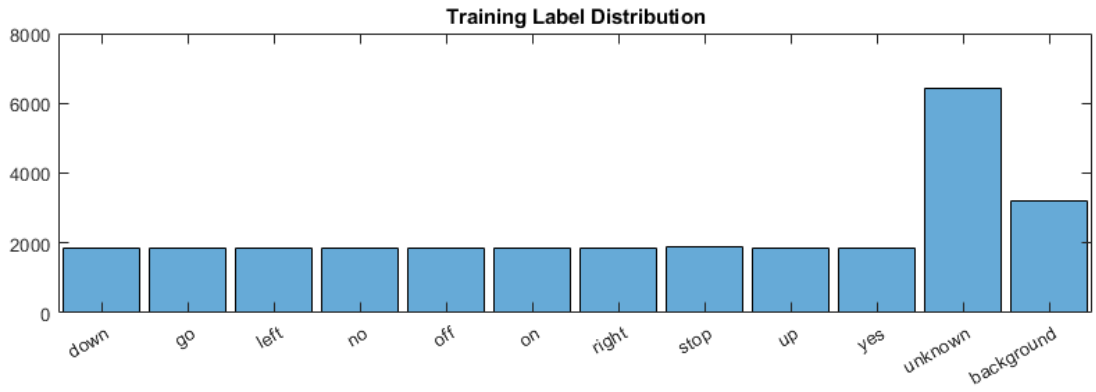
스캘로그램이나 연속 웨이블릿 변환과 같은 웨이블릿 변환을 이용하면, 특히 비주기 신호 컴포넌트를 시간 분해능이 높은 스펙트로그램과 비슷한 2차원 표현으로 변환할 수 있습니다.

웨이블릿 스캐터링(wavelet scattering) 또는 "불변 스캐터링 컨볼루션 네트워크(invariant scattering convolutional networks)"라고 부르는 자동 특징 추출 기법에도 웨이블릿을 사용합니다. 이 기법은 CNN의 첫 몇 개의 계층에서 학습한 특징을 모방합니다. 그러나 이 기법은 고정된 가중치 패턴을 이용하는데, 데이터에서 학습할 필요가 없기 때문에 네트워크 복잡성과 학습 데이터 크기 요건이 크게 줄어듭니다.

다음으로 데이터를 세 세트로 나눕니다.

- **학습 데이터:** 원본 입력 데이터로, 네트워크가 이 데이터를 이용하여 특징을 학습하고 이해하게 됩니다.
- **검증 데이터:** 네트워크가 학습할 때 사용하여 알고리즘이 특징을 학습하는지 검증하고, 데이터를 이해한 내용도 일반화합니다.
- **시험 데이터:** 네트워크가 지금까지 보지 못한 데이터로, 학습한 후에 네트워크에 넣어서 네트워크의 성능을 평가하고 결과에 편향성이 없는지 확인합니다.

분류하고자 하는 단어 클래스에 고르게 학습 데이터를 분산시킵니다.



고르게 분산된 학습 데이터.

긍정 오류를 줄이기 위해, 의도한 범주와 혼동될 수 있는 단어 범주를 포함시킵니다. 예를 들어 의도한 단어가 "on"인 경우, "mom", "dawn", "won" 등을 "unknown" 범주에 넣습니다. 네트워크가 이러한 단어들을 알 필요는 없고, 인식하려는 단어가 아니라는 것만 알면 됩니다.

네트워크 아키텍처 설정하기

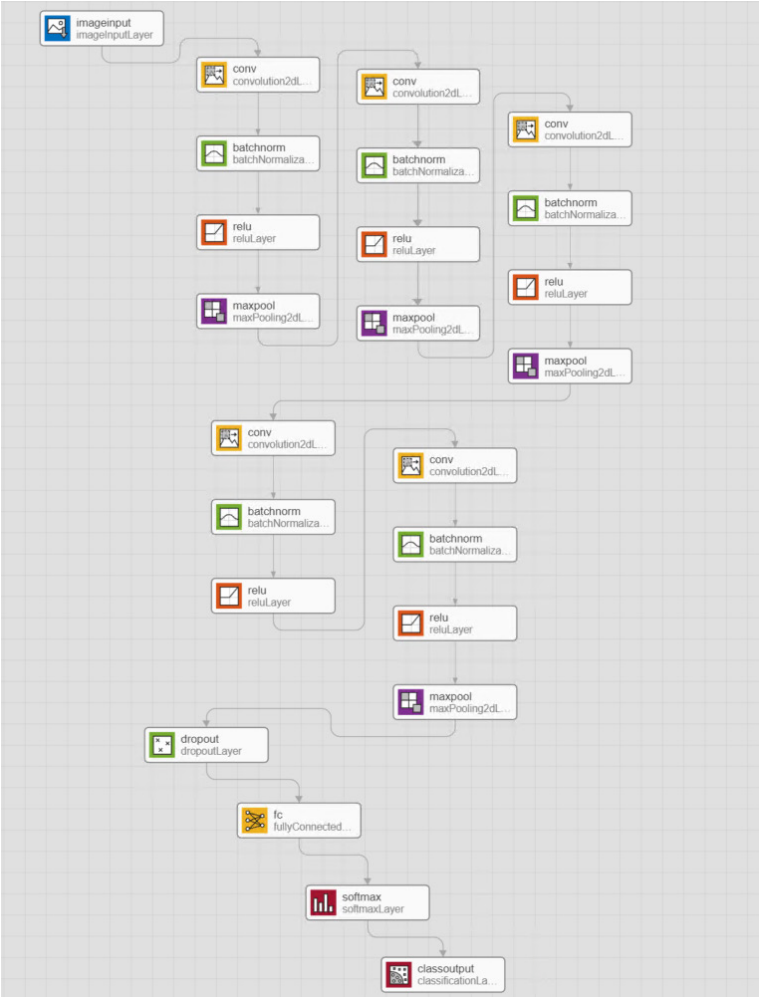
스펙트로그램을 입력값으로 활용하기 때문에 CNN의 구조는 이미지 분류에 사용한 다른 네트워크의 구조와 비슷할 수 있습니다. 이 애플리케이션에서는 48개 계층으로 이루어진 네트워크를 활용합니다.

48개 계층으로 구성된 네트워크라면 복잡할 것 같지만, 기본 구조는 단순합니다. 컨볼루션, 배치 정규화, ReLU 계층이 계속 반복되어 있습니다.

네트워크 아키텍처를 어떻게 정의하나요?

널리 사용되는 방법은 연구논문이나 이미 발행된 출처에 있는 네트워크로 시작해서 필요한 대로 수정하는 방법입니다.

전체 모델을 다음과 같이 표현할 수 있습니다.



컨볼루션, 배치 정규화, ReLU를 얼마나 반복하여 실행할지는 네트워크 설계자가 결정하지만, 네트워크가 복잡해지면 학습하고 시험하기 위해 더 많은 데이터가 필요하다는 점을 기억해야 합니다.

네트워크 학습시키기

학습을 시작하기 전에 다음과 같은 학습 옵션을 지정합니다.

- 에포크 개수(전체 학습 데이터의 반복 횟수)
- 학습 속도(학습하는 속도)
- 프로세서(일반적으로 CPU 또는 GPU)

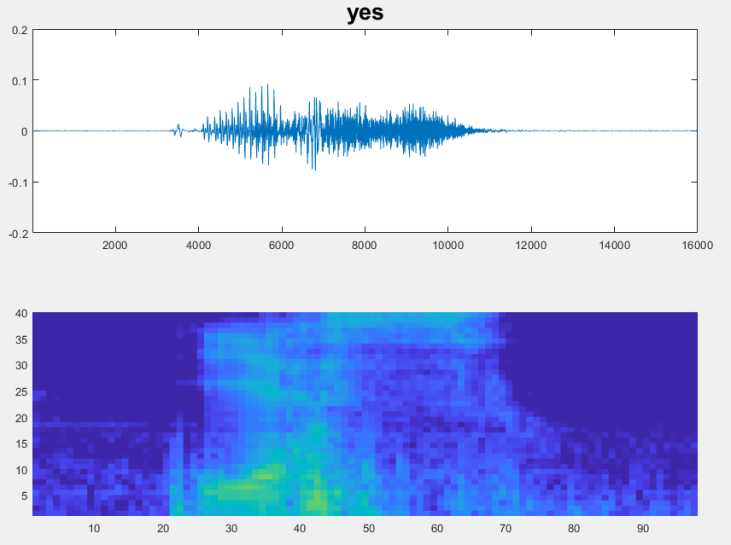
다음으로 신호 샘플을 네트워크에 넣고, 학습이 진행되는 과정을 플로팅합니다.

네트워크가 이 샘플에 있는 고유한 특징을 식별하고, 그에 맞춰 분류하게 됩니다. 스펙트로그램 데이터를 다루기 때문에 학습 과정은 본질적으로 이미지 분류와 같습니다. 네트워크의 초반 계층에서는 색상, 두께, 선과 형상의 방향 등 스펙트로그램의 대략적인 특징을 학습합니다. 학습이 계속되면서 네트워크는 개별 신호의 더 세밀한 특징을 식별하게 됩니다.

네트워크 정확도 점검하기

지정한 에포스 횟수를 완료하거나 지정된 정지 포인트에 도달하여 모델이 학습을 마치면, 모델이 입력 이미지(스펙트로그램)를 적절한 범주로 분류할 것입니다.

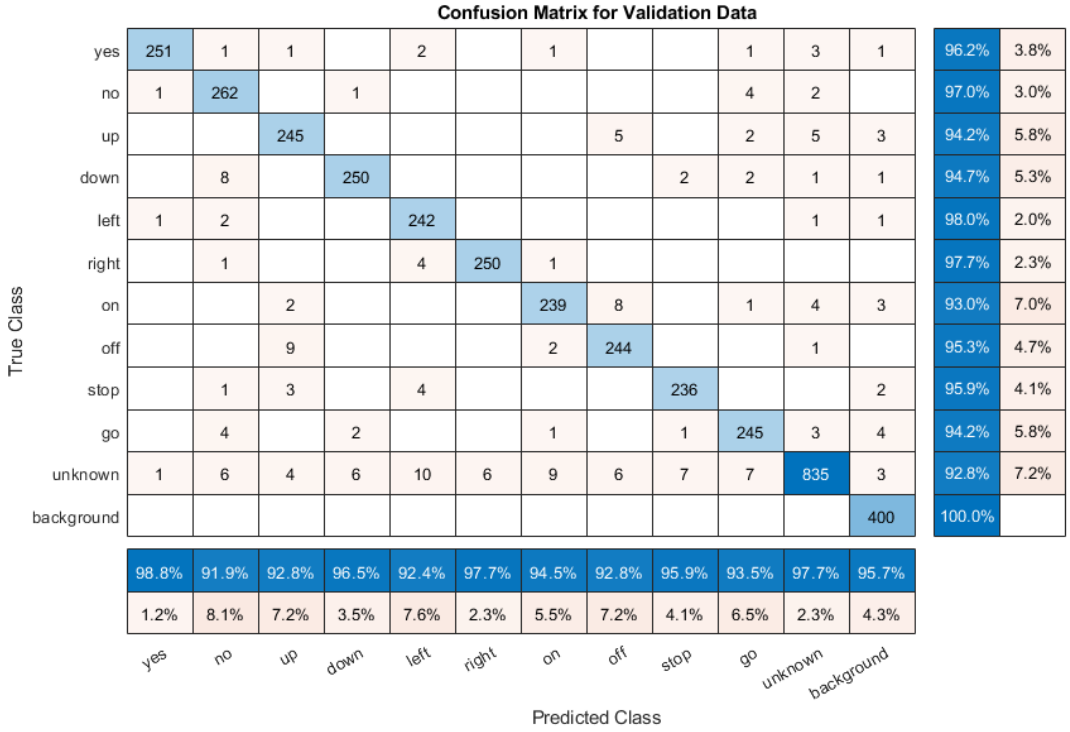
결과를 시험하기 위해 네트워크 예측 결과와 그에 대응된 오디오 클립을 비교합니다. 검증 세트의 정확도는 약 96% 입니다.



"yes"로 분류된 단어의 그래프와 스펙트로그램

간단한 음성 인식 작업이라면 이 정도 결과로 충분합니다만, 예를 들어 보안 시스템이나 음성 작동 기기에서 모델을 구현하려면 정확도가 더 높아야 할 것입니다.

여기에서 딥러닝의 진가를 확인하실 수 있습니다. 딥러닝은 정확도를 높이기 위한 다양한 방법을 제공합니다. 네트워크에 학습 샘플을 추가하기만 해도 결과를 개선할 수 있습니다. 네트워크의 예측이 틀린 곳에 대하여 더 깊은 통찰력을 얻으려면 정오분류표를 만들 수 있습니다.



정오분류표에는 네트워크가 예측한 클래스(단어)와 실제 클래스가 비교되어 제시됩니다. 파란색 음영은 정확한 예측이고, 베이지색은 틀린 예측입니다.

원하는 정확도에 도달할 때까지 딥러닝을 계속할 수 있습니다. 종래의 신호 처리에서는 신호의 뉘앙스를 이해하려면 상당히 많은 전문지식이 필요했습니다.

사례 2. 잔여 수명 예측. 시퀀스-투-시퀀스 회귀

이 사례에서는 엔진이 고장 나기 전 남은 수명, 즉 *잔여 수명(RUL)*을 예측하려 합니다 RUL은 엔진의 정비 주기를 정하고, 일정에 없는 지연 사태를 방지하는 중요한 척도입니다.

많은 딥러닝 네트워크가 클래스 또는 범주를 예측하지만, 원하는 결과는 숫자입니다. 즉 엔진이 고장 날 때까지 남은 사이클 횟수이기 때문에 이 문제는 회귀 문제가 됩니다.

'고장 날 때까지 작동시킨(run-to-failure)' 데이터를 이용하여 RUL을 계산하며, 이 데이터에는 시스템의 상대적인 상태에 따라 센서 값이 어떻게 변하는지 나타납니다.

딥러닝을 사용하지 않을 경우에는 문제에 접근하기 위한 워크플로는 다음과 같을 것입니다.

1. 시스템 상태를 가장 잘 나타내는 센서를 선택합니다.
2. 그러한 센서들을 조합하여 척도를 도출합니다.
3. 계속적으로 데이터를 추적하여, 시스템이 고장 날 위험이 있는지 확인합니다.
4. 고장 확률 또는 고장이 발생할 때까지 걸리는 시간을 반환하는 모델을 구성합니다.

첫 단계인 센서 선택(특징 선택)은 쉽지 않은 작업입니다. 시스템에 있는 센서는 50개가 넘는 경우가 흔하고, 시스템이 고장에 접근한다고 해서 모든 센서가 스트레스 신호를 나타내지도 않습니다. 딥러닝을 이용하면 모델이 특징을 추출합니다. 네트워크는 어떤 센서가 문제를 가장 잘 예측하는지 식별합니다.

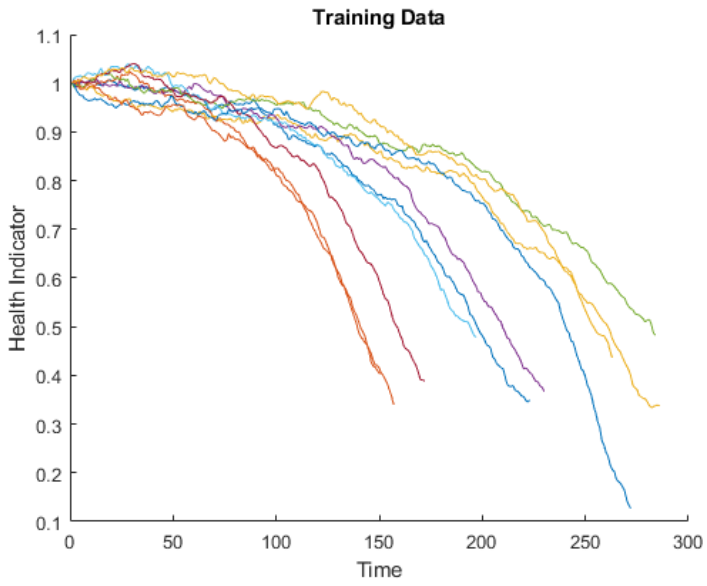
데이터 가져오기

먼저 NASA 진단 데이터 저장소에 있는 터보팬 엔진 성능 저하 시뮬레이션 데이터를 이용합니다.

MATLAB 실습: *더 많은 사례와 데이터 세트 링크*

이 데이터 세트에는 검증에 활용할 수 있는 실제 RUL 값 벡터뿐만 아니라 학습 및 시험 관측값이 포함되어 있습니다.

각각의 시계열이 하나의 엔진을 나타냅니다. 엔진들은 시계열이 시작할 때 정상적으로 작동하고, 시계열 중간의 어떤 시점에서 오류가 발생합니다. 학습 세트에서 오류가 점점 커져서 시스템 고장 시점에 이르게 됩니다.



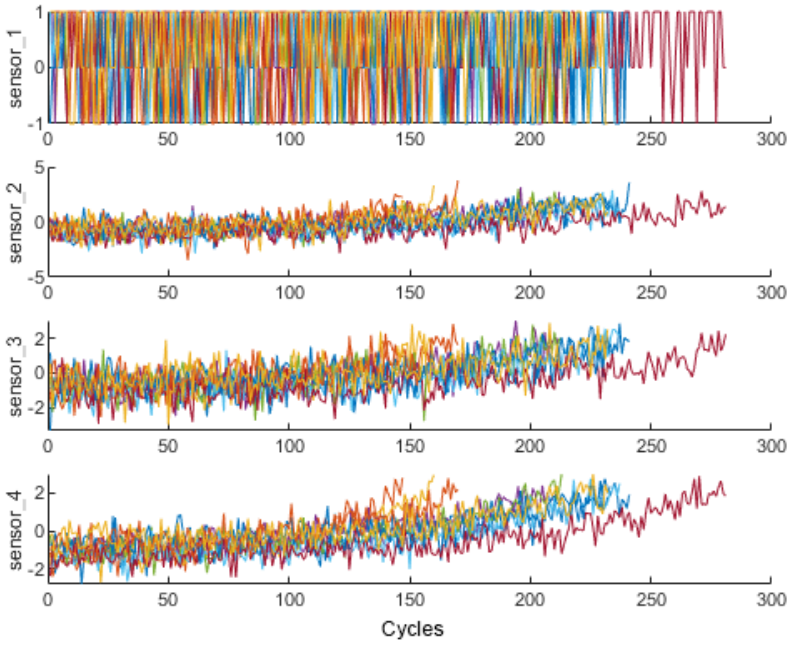
0사이클부터 300사이클까지의 학습 데이터 모습

데이터 준비하기

정확하게 예측하기 위해, 네트워크가 활용할 수 있는 최상의 데이터를 제공하려 합니다. 데이터를 간단히 하고 정리하기 위해 다음을 수행합니다.

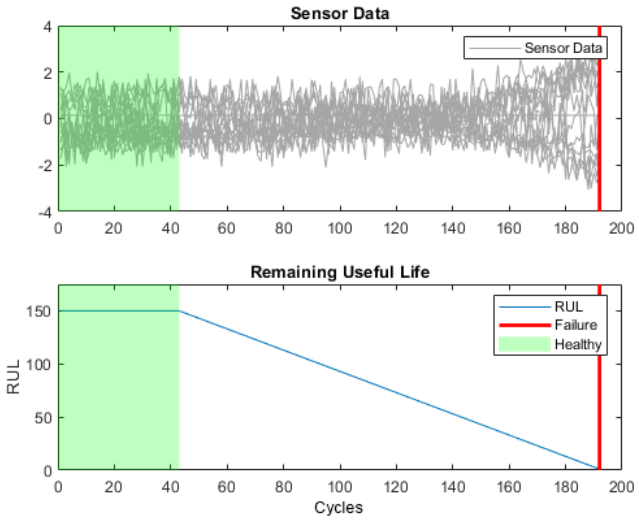
상수 신호를 제거합니다. 모든 시간 스텝에 일정하게 유지되는 신호는 학습된 모델의 정확도에 도움이 되지 않을 것이기 때문에 최솟값과 최댓값이 동일한 신호를 제거할 수 있습니다.

학습 예측변수를 정규화합니다. 다양한 신호가 다양한 범위에 걸쳐 있기 때문에 데이터를 정규화해야 합니다. 이 사례에서 평균이 0이고 구간이 -4와 4 사이라고 가정합니다. 모든 관측값에 걸친 평균과 표준편차를 계산하기 위해 시퀀스 데이터를 수평으로 연결합니다.



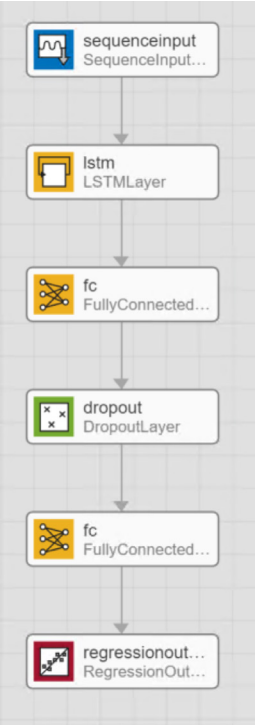
센서 1 내지 센서 4에서 나온 데이터를 정규화한 결과

응답을 클립합니다. 응답을 클립하면 시스템이 고장 나기 시작하는 시점에 네트워크를 집중시킵니다. 아래 그림에는 한 엔진의 센서 데이터(맨 위 그래프)와 각 시점에서 센서 데이터에 상응하는 잔여 수명(아래 그래프)이 표시되어 있습니다. 고장까지 150사이클 이상 남은 신호의 초기값(녹색)에서 잔여 수명 값이 최대 150이어서, 네트워크를 다소 보수적으로 만들었습니다.



위: 한 엔진의 센서 데이터 아래: 각 시점에서 센서 데이터에 상응하는 RUL

네트워크 아키텍처 설정하기



LSTM은 시퀀스 데이터의 시간 스텝 사이의 의존성을 학습할 수 있기 때문에 이러한 분석에는 LSTM이 적합합니다. 센서들이 흔히 서로 연계되어 있고, 두 사이클 전에 발생한 사건이 시스템의 향후 성능에 영향을 미칠 수 있습니다.

은닉 유닛이 200개인 계층과 드롭아웃 확률이 0.5인 드롭아웃 계층으로 구성된 LSTM 네트워크를 정의합니다.

드롭아웃 확률은 과잉피팅을 방지하기 위한 도구입니다. 이 네트워크에서 이 값을 이용하여 일부 데이터를 임의로 건너뛰어서, 학습 세트를 기억하지 않도록 합니다. 은닉 유닛은 보통 100단위의 숫자입니다. 포함시킬 은닉 유닛의 개수를 결정하는 것은 일종의 절충과정입니다. 너무 적으면 모델이 학습할 메모리가 충분하지 않게 됩니다. 너무 많으면 네트워크가 과적합이 될 수 있습니다.

네트워크 학습시키기

먼저 학습 옵션 두 개를 설정합니다.

- 솔버 **'adam'**을 사용하여 사이즈 20인 미니배치가 있는 에포크를 60개로 설정합니다.
- 학습 속도를 0.01로 설정합니다.

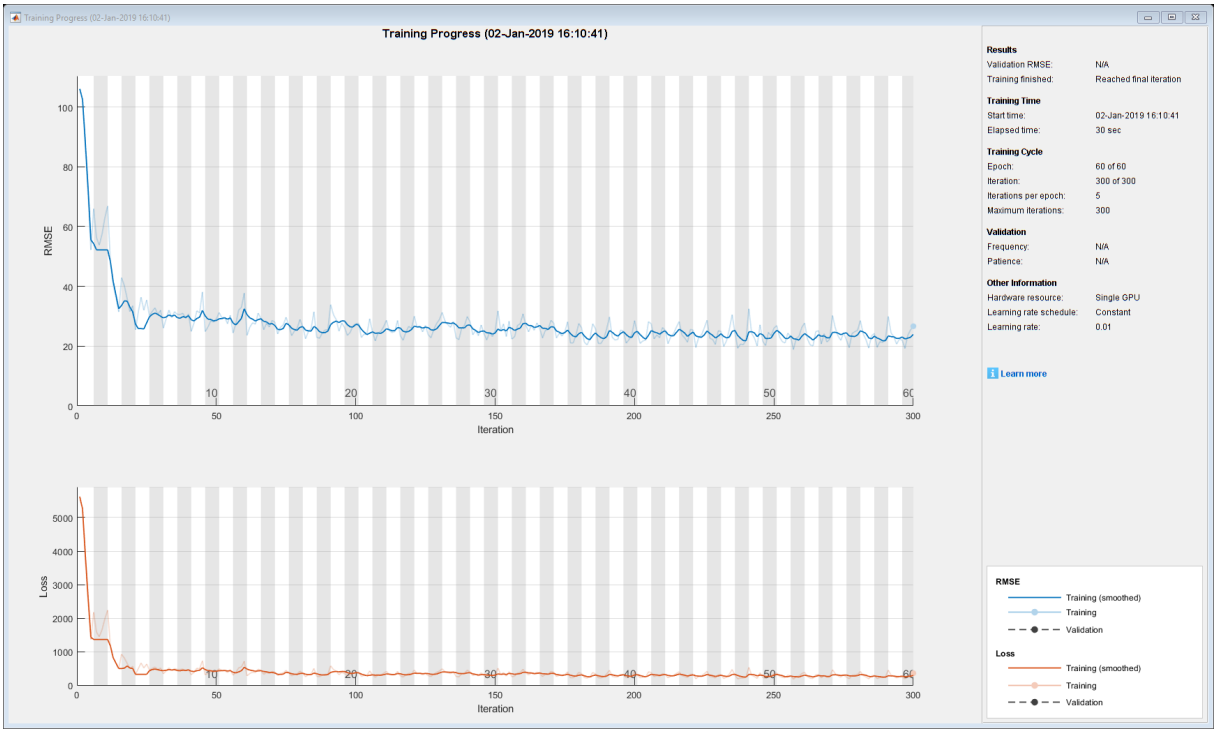
이러한 설정으로 네트워크를 학습시키면 약 2분이 걸립니다. 네트워크가 실행 중인 동안에 학습 진도를 플로팅합니다.

솔버란 무엇일까요?

솔버는 뉴럴 네트워크를 최적화하는 알고리즘입니다.

'adam'(adaptive moment estimation)이 LSTM 네트워크에 가장 널리 사용하는 솔버이며, 보통은 디폴트 설정으로 사용할 수 있습니다. **'adam'**은 파라미터 기울기와 파라미터 제곱 값의 요소별 이동평균을 유지합니다.

'adam'과 기타 솔버에 대해 더 자세히 알아봅시다.**'sgdm'**이나 **'rmsprop'**도 있습니다.



RUL 모델의 학습 결과.

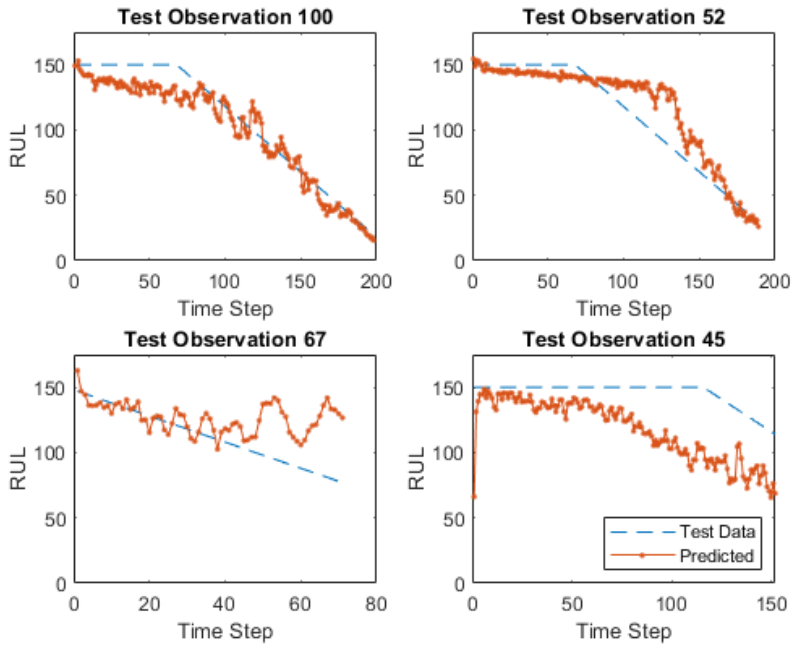
RUL 예측은 회귀 문제이기 때문에 첫 플롯은 정확도가 아닌 제곱평균오차(RMSE)의 회귀를 보여주고 있습니다. 숫자가 작을수록 고장까지 걸릴 것으로 예측된 시간이 실제 값에 가깝습니다.

네트워크 정확도 점검하기

네트워크를 학습시킨 후, 시험 데이터를 이용하여 검증합니다.

LSTM 네트워크는 한 번에 한 시간 스텝씩 부분 시퀀스를 예측합니다. 각 사이클에서 네트워크는 내부 상태를 업데이트하고 고장까지 걸리는 시간의 길이인 RUL 값을 예측합니다.

네트워크의 성능을 보여주기 위해, 4개의 시험용 엔진에서 데이터를 선택하고 예측치와 실제 잔여 수명을 비교하여 플로팅합니다.



실제 RUL(파란색 점선)과 예측된 RUL(오렌지색 선)을 비교한 네 엔진(100, 52, 67, 45)의 모델 예측

LSTM을 사용한 결과, 엔진 100과 엔진 45는 고장까지 걸리는 시간이 비교적 정확하게 예측되었지만 엔진 52과 엔진 67은 정확도가 낮습니다. 이 결과를 보면, 그러한 경우에 엔진의 성능을 나타내는 학습 데이터가 더 많이 필요하다는 것을 알 수 있습니다. 다른 방법으로 네트워크 학습 설정을 수정하고 성능이 개선되는지 살펴볼 수도 있습니다.

이 사례에서는 간단한 LSTM 네트워크를 설계, 학습, 시험하였고, 이 네트워크를 이용하여 잔여 수명을 예측했습니다. 기존의 신호 처리 대신에 딥러닝을 이용해서, 특징 추출이라는 어렵고도 많은 시간이 걸리는 작업을 하지 않게 되었습니다.

사례 3. 완전 연결 뉴럴 네트워크를 이용한 음성 노이즈 제거

신호의 품질과 명료성을 높이는 한편, 음성 신호에서 세탁기 소음을 제거하려 합니다. 엔지니어는 신호에서 노이즈를 제거하여 데이터 품질을 높입니다. 예를 들면 더 나은 레코딩을 클라우드 기반 음성 보조 엔진에 넣거나 ECG에서 신호대 잡음비를 높입니다.

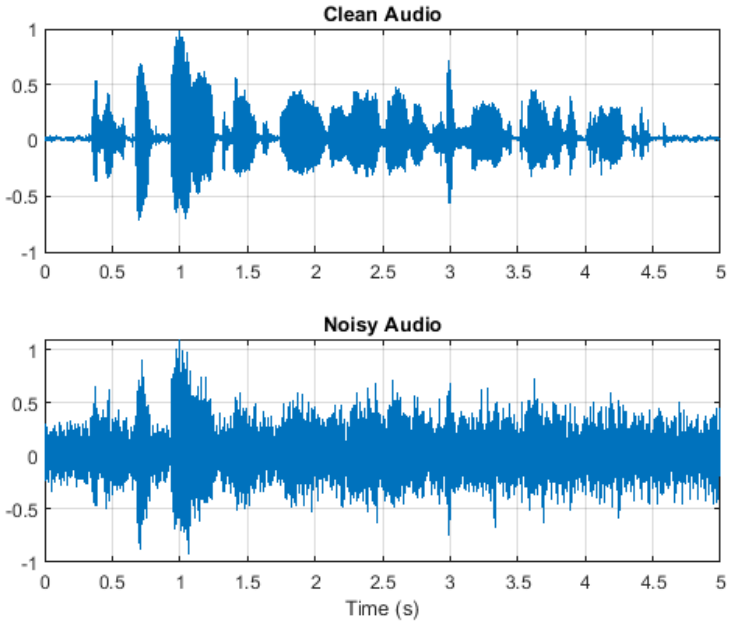
노이즈를 거르기 위해 종래의 신호 처리 기법을 이행하였고, 지금도 그렇게 하고 있습니다. 이러한 방식(스펙트럼 차감, 노이즈 게이팅 등)의 문제점은 양호한 품질의 신호도 종종 의도치 않게 걸러져서 원본 신호의 정보가 유실될 수 있다는 점입니다.

이 사례에서는 음성 노이즈 제거에 딥러닝을 이용하여, 산출된 음성에 있는 불필요한 아티팩트를 최소화하는 한편, 음성 신호에서 세탁기 소음을 제거할 수 있습니다. 이 네트워크 산출물을 이용하면 다른 유사한 시끄러운 상황에서도 소음을 거를 수 있습니다.

데이터 가져오기

깨끗한 음성 신호를 읽고, 이어서 이 깨끗한 신호에서 소음 신호를 생성합니다. 미리 녹음된 wav 파일에서 나온 세탁기 소음을 신호에 추가하는 방법을 이용합니다. 이렇게 하면 실제 깨끗한 신호와 최종적으로 노이즈를 제거한 신호를 비교함으로써, 알고리즘이 얼마나 잘 작동하는지 이해할 수 있게 됩니다.

MATLAB 실습: *사례 파일과 전체 사례 링크*



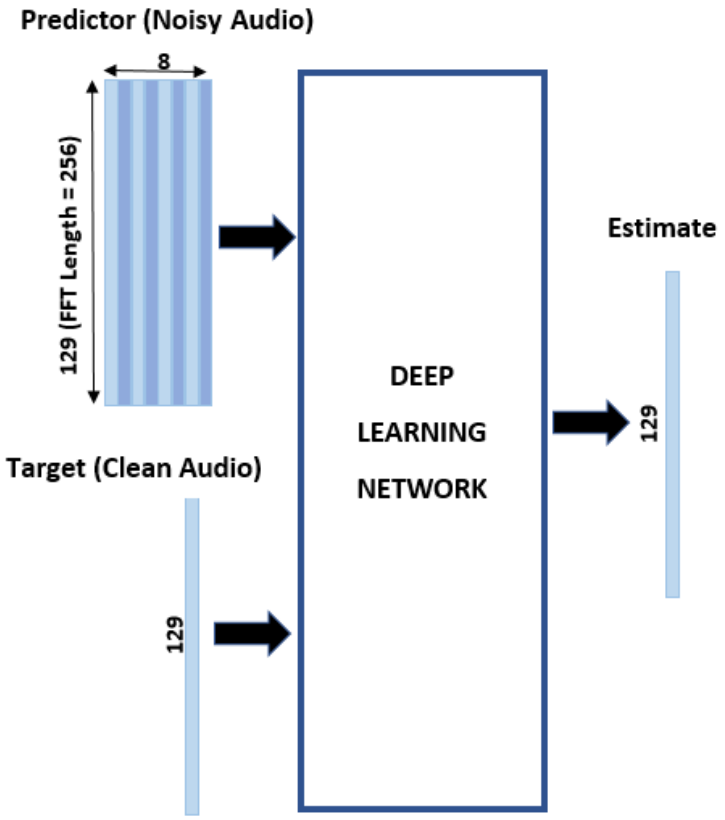
원본 오디오와 소음이 있는 오디오 플롯

데이터 준비하기

학습된 알고리즘을 시험하기 위해, 소음이 있는 음성 사례들을 남겨둡니다.

이 사례에서는 단시간 푸리에 변환(STFT)을 이용하여 신호를 2차원 신호로 변환합니다. 이 기법에서는 신호를 중첩되는 세그먼트들로 나누고, 각 세그먼트의 고속 푸리에 변환(FFT)을 계산합니다. 세그먼트들로 나뉜 스트리밍 신호를 취하기 때문에, 신호를 획득하는 중에 예측결과를 볼 수 있습니다. 입력에서는 8개 세그먼트를 누적하여 네트워크에 일종의 '메모리'를 제공합니다. 이렇게 하여, 네트워크가 현재의 신호 세그먼트만을 보는 것이 아니라 현재의 신호와 밀접한 관계에 있는 과거의 몇몇 신호도 볼 수 있도록 합니다. 출력에서는 네트워크가 한 번에 한 세그먼트씩 예측하도록 합니다.

이 과정이 아래에 나타나 있습니다.



예측변수 입력값은 소음이 있는 연속된 STFT 벡터 8개이며, 따라서 각각의 STFT 출력 예측은 소음이 있는 현재의 STFT 1개 및 맥락을 부여하기 위해 제공된 소음이 있는 과거의 STFT 벡터 7개를 기초로 계산됩니다.

STFT 타겟과 예측변수

단시간 푸리에 변환(STFT)을 이용하여 오디오를 주파수 영역으로 변환할 때, 약간의 시스템 파라미터를 적용해야 합니다.

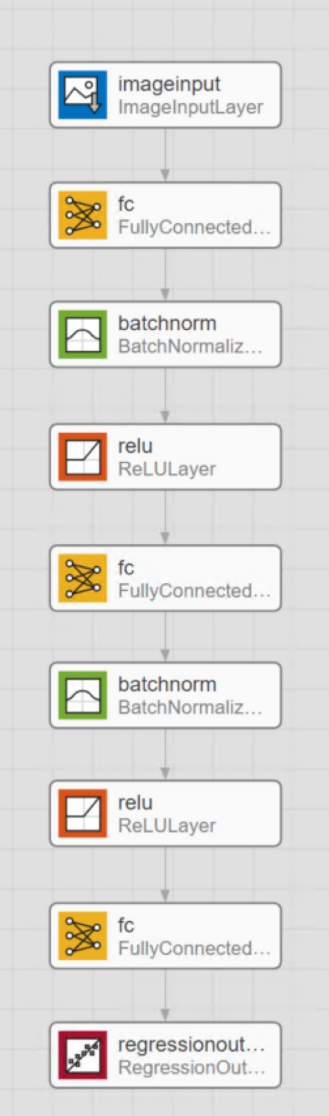
윈도우 길이. 윈도우 길이가 길면 분해능이 높아지고 복잡성이 커집니다. 이 사례에서 256이라는 윈도우 길이는, 가용한 데이터를 감안하여 높은 주파수 분해능과 낮은 계산 복잡성을 합리적으로 절충한 것입니다.

중첩. 중첩을 75%로 설정하여, 네트워크에 더 정밀한 시간 분해능을 제공하고 같은 시간 동안에 네트워크에 정보가 더 많이 통과하도록 합니다.

입력 샘플링 레이트. 이 입력 데이터는 샘플링 레이트 48kHz로 녹음되었습니다.

주파수 샘플링 레이트. 사람 음성은 입력 데이터를 8kHz로 다운샘플링하면 충분할 것이며, 작고 견고한 네트워크를 구축할 수 있습니다.

세그먼트 개수 이 사례에서는 소음이 있는 현재 STFT 1개와 네트워크가 학습하기 위한 맥락이 되는 과거의 세그먼트 7개를 포함하여, 세그먼트 8개를 선택합니다. 각 세그먼트가 75% 중첩되기 때문에 대략적으로 완전한 시간 세그먼트가 3개입니다.



네트워크 아키텍처 설정하기

첫 번째 사례에서 사용한 것과 같은 기본 워크플로를 따를 것입니다. 음성 명령 인식 사례에서는 컨볼루션 계층들을 사용했지만, 이 사례에서는 완전 연결 계층을 사용합니다.

완전 연결 계층은 이전의 계층에 있는 모든 액티베이션에 연결되어 있습니다. 완전 연결 계층은 2차원 공간 특징을 1차원 벡터로 평면화합니다.

배치 정규화 계층은 출력값의 평균과 표준편차를 정규화합니다. 학습 단계에서 파라미터나 이전 계층이 변함에 따라 현재 계층이 새로운 분포로 재조정되어야 하는데, 시간이 걸립니다. 컨벌루션 신경망의 훈련 속도를 높이고 신경망 초기화에 대한 민감도를 줄이기 위해 배치 정규화 계층과 컨벌루션 계층 사이의 ReLU 계층과 같은 비선형성을 사용합니다.

ReLU 계층은 입력값의 각 요소에 대해 임계값 연산을 수행하며, 이때 0보다 작은 모든 값이 0으로 설정됩니다.

회귀 계층으로 마무리하고, 이때 평균제곱오차의 절반인 손실이 산출됩니다. 회귀 계층은 각도, 거리 또는 이 경우에는 노이즈가 제거된 신호와 같이, 연속 데이터에 관한 예측을 산출합니다.

네트워크 학습시키기

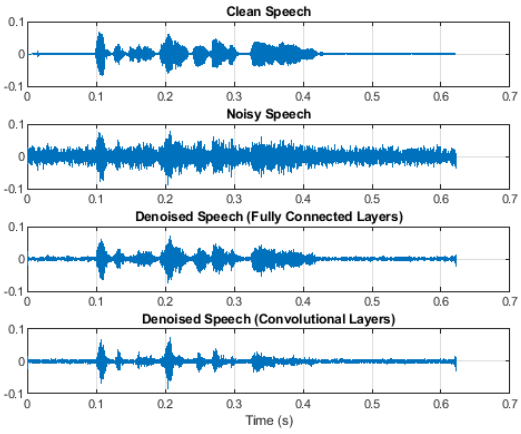
다음으로 학습 옵션을 다음과 같이 설정합니다.

- 최대 에포크를 3으로 설정합니다(네트워크가 학습 데이터를 세 번 반복합니다).
- 미니 배치 사이즈를 128로 설정합니다(네트워크가 한 번에 학습 신호 128개를 보게 됩니다.)
- `'Plots'`를 `'training-progress'`로 설정합니다(학습 그래프를 그리게 됩니다.)
- `'Shuffle'`을 `'every-epoch'`로 설정합니다.
- `'LearnRateSchedule'`을 `'piecewise'`로 설정합니다(에포크를 지날 때마다 지정된 만큼(0.9) 학습 속도가 감소합니다.)

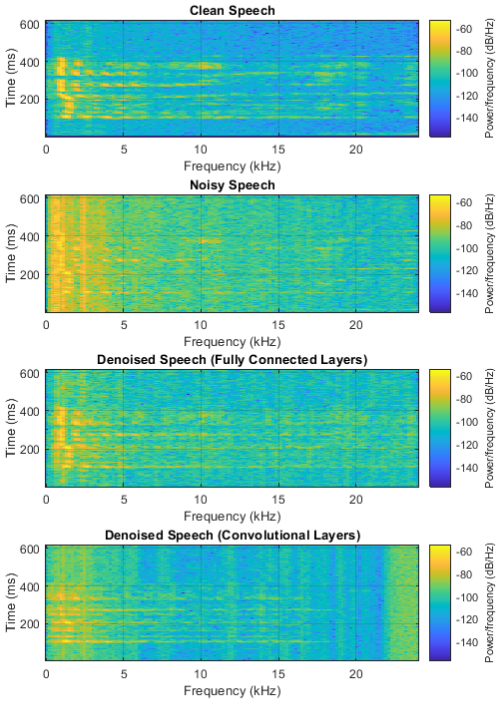
이 크기로 학습을 설정하면 실행하는 데 몇 분 정도 걸립니다.

네트워크 정확도 점검하기

네트워크가 학습을 마친 후, 시험을 위해 남겨 두었던 소음 신호를 네트워크에 넣고 결과를 도시할 수 있습니다. 신호를 플롯 또는 스펙트로그램으로 표시할 수 있습니다.



시간 영역 플롯은 배경 노이즈 수준이 상당히 감소하였음을 나타냅니다.



스펙트로그램은 다양한 주파수에서 모델이 어떻게 작동하는지 자세히 보여주고 있습니다. 예를 들면, 대부분의 음성 스펙트럼이 집중된 낮은 주파수에서 노이즈를 제거할 때 특히 효과적임을 알 수 있습니다.

오디오 데이터이기 때문에 음성을 귀로 들어서 품질을 주관적으로 비교하여, 결과가 만족스러운지 알 수 있습니다.

이제 학습된 네트워크를 실시간으로 활용할 수 있게 되었습니다. 이제 반복적으로 정확하게 음성의 노이즈를 제거할 수 있는 모델을 얻게 되었습니다.

MATLAB에서 **`speechDenoisingRealtimeApp`**을 실행시켜서 노이즈 제거 네트워크의 스트리밍 실시간 버전을 시뮬레이션해보세요.

요약

신호 처리에 딥러닝을 활용하면 엔지니어가 반복해서 사용할 수 있는 정확한 모델을 만드는 데 도움이 되고, 신호에서 특징을 추출하고 선택할 때 필요한 전문지식이 줄어들며, 변경을 통해 모델의 성능을 개선할 수 있습니다.

음성 인식과 신호 노이즈 제거 사례에서는, CNN과 완벽하게 연결된 각각의 네트워크에 사용하기 위해 오디오 데이터를 어떻게 준비하고 변환하는지 알아보았습니다. 모델의 성능을 눈으로 확인하였고, 많은 신호 처리 전문지식이 없이도 개선할 수 있는 방법을 살펴보았습니다.

RUL 사례에서는 LSTM을 사용하였고, 센서 데이터를 전처리하여 네트워크에 최상의 데이터를 제공하려 하였으며 수작업으로 특징을 추출하고 선택하지 않는 실제 모델을 만들었습니다.

추가 리소스

딥러닝을 위한 MATLAB

신호와 소리를 위한 딥러닝