Draft Version

# MACHINE LEARNING YEARNING

Technical Strategy for AI Engineers, In the Era of Deep Learning

ANDREW NG

deeplearning.ai

Machine Learning Yearning is a
deeplearning.ai project.

# End-to-end deep learning

# 47 The rise of end-to-end learning

Suppose you want to build a system to examine online product reviews and automatically tell you if the writer liked or disliked that product. For example, you hope to recognize the following review as highly positive:

> This is a great mop!

and the following as highly negative:

> This mop is low quality--I regret buying it.

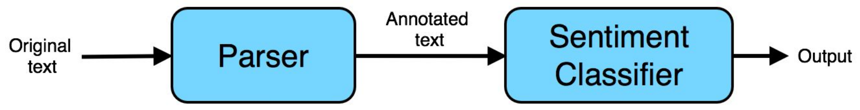
The problem of recognizing positive vs. negative opinions is called "sentiment classification." To build this system, you might build a "pipeline" of two components:

1. Parser: A system that annotates the text with information identifying the most important words.[1] For example, you might use the parser to label all the adjectives and nouns. You would therefore get the following annotated text:

   This is a great$_{\text{Adjective}}$ mop$_{\text{Noun}}$!

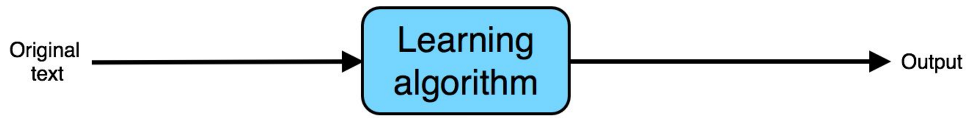
2. Sentiment classifier: A learning algorithm that takes as input the annotated text and predicts the overall sentiment. The parser's annotation could help this learning algorithm greatly: By giving adjectives a higher weight, your algorithm will be able to quickly hone in on the important words such as "great," and ignore less important words such as "this."

We can visualize your "pipeline" of two components as follows:



There has been a recent trend toward replacing pipeline systems with a single learning algorithm. An **end-to-end learning algorithm** for this task would simply take as input the raw, original text "This is a great mop!", and try to directly recognize the sentiment:

[1] A parser gives a much richer annotation of the text than this, but this simplified description will suffice for explaining end-to-end deep learning.

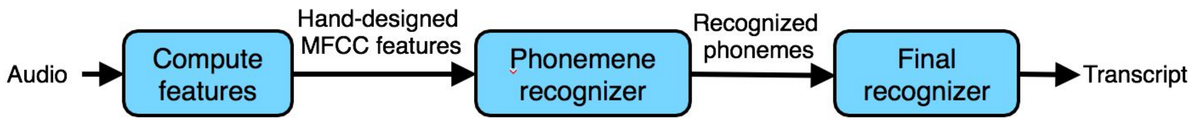


Neural networks are commonly used in end-to-end learning systems. The term "end-to-end" refers to the fact that we are asking the learning algorithm to go directly from the input to the desired output. I.e., the learning algorithm directly connects the "input end" of the system to the "output end."

In problems where data is abundant, end-to-end systems have been remarkably successful. But they are not always a good choice. The next few chapters will give more examples of end-to-end systems as well as give advice on when you should and should not use them.

# 48 More end-to-end learning examples

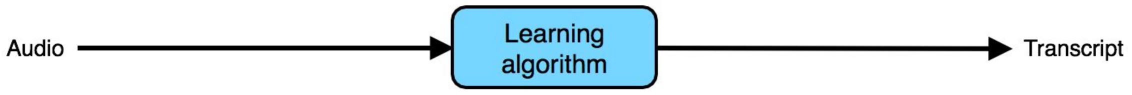
Suppose you want to build a speech recognition system. You might build a system with three components:



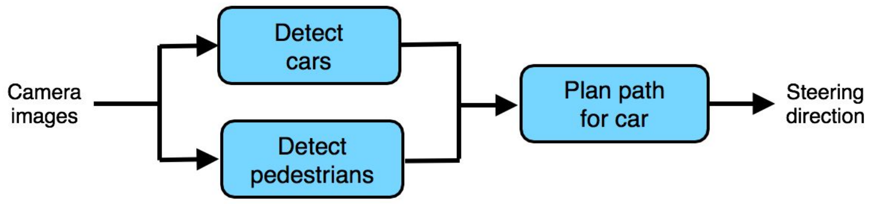
The components work as follows:

1. Compute features: Extract hand-designed features, such as MFCC (Mel-frequency cepstrum coefficients) features, which try to capture the content of an utterance while disregarding less relevant properties, such as the speaker's pitch.
2. Phoneme recognizer: Some linguists believe that there are basic units of sound called "phonemes." For example, the initial "k" sound in "keep" is the same phoneme as the "c" sound in "cake." This system tries to recognize the phonemes in the audio clip.
3. Final recognizer: Take the sequence of recognized phonemes, and try to string them together into an output transcript.

In contrast, an end-to-end system might input an audio clip, and try to directly output the transcript:



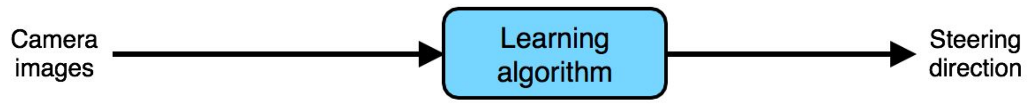
So far, we have only described machine learning "pipelines" that are completely linear: the output is sequentially passed from one staged to the next. Pipelines can be more complex. For example, here is a simple architecture for an autonomous car:



It has three components: One detects other cars using the camera images; one detects pedestrians; then a final component plans a path for our own car that avoids the cars and pedestrians.

Not every component in a pipeline has to be learned. For example, the literature on “robot motion planning” has numerous algorithms for the final path planning step for the car. Many of these algorithms do not involve learning.

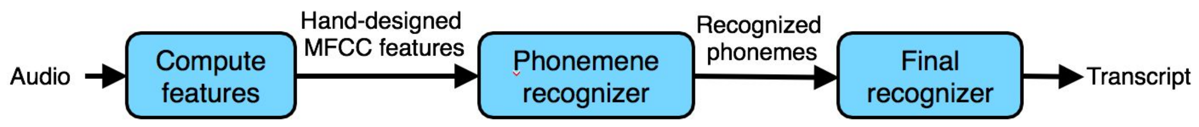
In contrast, and end-to-end approach might try to take in the sensor inputs and directly output the steering direction:



Even though end-to-end learning has seen many successes, it is not always the best approach. For example, end-to-end speech recognition works well. But I’m skeptical about end-to-end learning for autonomous driving. The next few chapters explain why.

# 49 Pros and cons of end-to-end learning

Consider the same speech pipeline from our earlier example:



Many parts of this pipeline were “hand-engineered”:

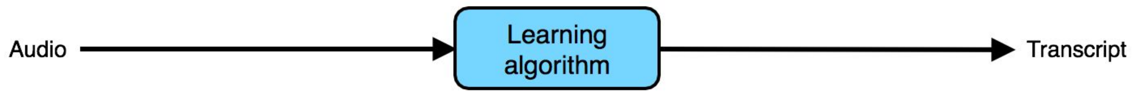
- MFCCs are a set of hand-designed audio features. Although they provide a reasonable summary of the audio input, they also simplify the input signal by throwing some information away.
- Phonemes are an invention of linguists. They are an imperfect representation of speech sounds. To the extent that phonemes are a poor approximation of reality, forcing an algorithm to use a phoneme representation will limit the speech system’s performance.

These hand-engineered components limit the potential performance of the speech system. However, allowing hand-engineered components also has some advantages:

- The MFCC features are robust to some properties of speech that do not affect the content, such as speaker pitch. Thus, they help simplify the problem for the learning algorithm.
- To the extent that phonemes are a reasonable representation of speech, they can also help the learning algorithm understand basic sound components and therefore improve its performance.

Having more hand-engineered components generally allows a speech system to learn with less data. The hand-engineered knowledge captured by MFCCs and phonemes “supplements” the knowledge our algorithm acquires from data. When we don’t have much data, this knowledge is useful.

Now, consider the end-to-end system:



This system lacks the hand-engineered knowledge. Thus, when the training set is small, it might do worse than the hand-engineered pipeline.

However, when the training set is large, then it is not hampered by the limitations of an MFCC or phoneme-based representation. If the learning algorithm is a large-enough neural network and if it is trained with enough training data, it has the potential to do very well, and perhaps even approach the optimal error rate.

End-to-end learning systems tend to do well when there is a lot of labeled data for “both ends”—the input end and the output end. In this example, we require a large dataset of (audio, transcript) pairs. When this type of data is not available, approach end-to-end learning with great caution.

If you are working on a machine learning problem where the training set is very small, most of your algorithm’s knowledge will have to come from your human insight. I.e., from your “hand engineering” components.

If you choose not to use an end-to-end system, you will have to decide what are the steps in your pipeline, and how they should plug together. In the next few chapters, we’ll give some suggestions for designing such pipelines.