



Recurrent Networks Are Turing Complete

Perhaps unsurprisingly, NTMs are capable of performing any computation a Turing machine can and are consequently Turing complete. However, a less known fact is that vanilla recurrent neural networks are themselves Turing complete! Put another way, in principle, a recurrent neural network is capable of learning to perform arbitrary computation.

The basic idea is that the transition operator can learn to perform basic reading, writing, and storage operations. The unrolling of the recurrent network over time allows for the performance of complex computations. In some sense, this fact shouldn't be too surprising. The universal approximation theorem already demonstrates that fully connected networks are capable of learning arbitrary functions. Chaining arbitrary functions together over time leads to arbitrary computations. (The technical details required to formally prove this are formidable, though.)

Working with Recurrent Neural Networks in Practice

In this section, you will learn about the use of recurrent neural networks for language modeling on the Penn Treebank dataset, a natural language dataset built from *Wall Street Journal* articles. We will introduce the TensorFlow primitives needed to perform this modeling and will also walk you through the data handling and preprocessing steps needed to prepare data for training. We encourage you to follow along and try running the code in the [GitHub repo associated with the book](#).

Processing the Penn Treebank Corpus

The Penn Treebank contains a million-word corpus of *Wall Street Journal* articles. This corpus can be used for either character-level or word-level modeling (the tasks of predicting the next character or word in a sentence given those preceding). The efficacy of models is measured using the perplexity of trained models (more on this metric later).

The Penn Treebank corpus consists of sentences. How can we transform sentences into a form that can be fed to machine learning systems such as recurrent language models? Recall that machine learning models accept tensors (with recurrent models accepting sequences of tensors) as input. Consequently, we need to transform words into tensors for machine learning.

The simplest method of transforming words into vectors is to use “one-hot” encoding. In this encoding, let's suppose that our language dataset uses a vocabulary that has $|V|$ words. Then each word is transformed into a vector of shape $(|V|)$. All the