Vanilla RNN

Ahmed Métwalli - Alia Elhefny

7/12/2024

1 Introduction

This document presents a detailed example of how a simple Recurrent Neural Network (RNN) processes a sequence of inputs. We start by introducing the mathematical formulation of a basic RNN cell, then work through a small numeric example, and finally present pseudo-code demonstrating how these computations might be implemented.

2 Vanilla RNN Solved Example

A simple RNN operates on an input sequence $(x_1, x_2, \ldots, x_T)$ and produces a sequence of hidden states $(h_1, h_2, \ldots, h_T)$. Each hidden state h_t depends on the current input x_t and the previous hidden state h_{t-1}. Formally:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h).$$

Here:

- $h_t \in \mathbb{R}^{n_h}$ is the hidden state vector at time step t, where n_h is the number of hidden units.
- $x_t \in \mathbb{R}^{n_x}$ is the input vector at time step t, where n_x is the input dimension.
- $W_{hh} \in \mathbb{R}^{n_h \times n_h}$ is the recurrent weight matrix.
- $W_{xh} \in \mathbb{R}^{n_h \times n_x}$ is the input-to-hidden weight matrix.
- $b_h \in \mathbb{R}^{n_h}$ is the bias term.
- $\tanh(\cdot)$ is the hyperbolic tangent activation function.

If we want an output at each time step y_t, we often define:

$$y_t = W_{hy}h_t + b_y,$$

where $W_{hy} \in \mathbb{R}^{n_y \times n_h}$ and $b_y \in \mathbb{R}^{n_y}$ produce the output vector $y_t \in \mathbb{R}^{n_y}$.

3 Side Note: Reasons for Using tanh

3.1 Bounded Output Range

The tanh function maps real numbers into the range $(-1, 1)$. By constraining hidden states to lie within this bounded range, tanh helps maintain numerical stability during training. Without such bounding, hidden states can grow very large (known as the "exploding gradient" problem), causing instability and making the training process difficult.

3.2 Centered Around Zero

Unlike the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, which maps inputs to $(0, 1)$, tanh is symmetric about zero. It produces negative outputs for negative inputs, positive outputs for positive inputs, and outputs near zero for inputs near zero. This zero-centered property often leads to more stable and efficient optimization, as gradients can flow more symmetrically around zero.

3.3 Historical and Conventional Usage

Early work in RNNs and related architectures commonly used tanh (or sigmoid) as the nonlinearity of choice. Although modern recurrent units like LSTMs and GRUs also rely on gating mechanisms involving sigmoid functions, the hidden state transformations often still leverage tanh. Its long-standing use in foundational literature and basic RNN implementations makes it a well-understood and standard choice.

3.4 Gradient Properties

The derivative of tanh at 0 is 1, which is larger than the derivative of sigmoid at 0 (which is 0.25). While tanh does not eliminate the vanishing gradient problem, it provides somewhat stronger gradients around zero compared to the sigmoid. This can be marginally beneficial during training, especially when the hidden states are small in magnitude.

4 A Simple Numerical Example

4.1 Setup

Dimensions: For simplicity, let us consider:

$$n_x = 1, \quad n_h = 1, \quad n_y = 1.$$

This means at each time step we have a single real number as input, a single real number as the hidden state, and a single real number as output.

Parameters: Since $n_h = 1$, all weight matrices and bias terms are scalars. Let us choose:

$$W_{hh} = 0.5, \quad W_{xh} = 1.0, \quad b_h = 0.0,$$

$$W_{hy} = 0.8, \quad b_y = 0.1.$$

4.2 Input Sequence

Suppose our input sequence is:

$$x = (x_1, x_2, x_3) = (0.0, 1.0, -0.5).$$

We will process these three time steps to compute the hidden and output states.

4.3 Initial Conditions

We initialize the first hidden state as:

$$h_0 = 0.$$

4.4 Time Step 1

Input: $x_1 = 0.0$, previous hidden state: $h_0 = 0$.

Compute:

$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1 + b_h) = \tanh(0.5 \cdot 0 + 1.0 \cdot 0.0 + 0.0) = \tanh(0) = 0.$$

Thus $h_1 = 0$.

Now output:

$$y_1 = W_{hy}h_1 + b_y = 0.8 \cdot 0 + 0.1 = 0.1.$$

At time step 1:

$$h_1 = 0, \quad y_1 = 0.1.$$

4.5 Time Step 2

Input: $x_2 = 1.0$, previous hidden state: $h_1 = 0$.

Compute:

$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h) = \tanh(0.5 \cdot 0 + 1.0 \cdot 1.0 + 0.0) = \tanh(1.0) \approx 0.761594.$$

Let $h_2 \approx 0.7616$.

Output:

$$y_2 = W_{hy}h_2 + b_y = 0.8 \cdot 0.7616 + 0.1 \approx 0.60928 + 0.1 = 0.70928.$$

At time step 2:

$$h_2 \approx 0.7616, \quad y_2 \approx 0.7093.$$

4.6 Time Step 3

Input: $x_3 = -0.5$, previous hidden state: $h_2 \approx 0.7616$.

Compute:

$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h) = \tanh(0.5 \cdot 0.7616 + 1.0 \cdot (-0.5) + 0.0).$$

First, compute the argument:

$$0.5 \cdot 0.7616 - 0.5 = 0.3808 - 0.5 = -0.1192.$$

Thus:

$$h_3 = \tanh(-0.1192) \approx -0.1187.$$

Output:

$$y_3 = W_{hy}h_3 + b_y = 0.8 \cdot (-0.1187) + 0.1 \approx -0.09496 + 0.1 = 0.00504.$$

At time step 3:

$$h_3 \approx -0.1187, \quad y_3 \approx 0.0050.$$

4.7 Final Results

For the input sequence $(0.0, 1.0, -0.5)$, the RNN produced:

$$h_1 = 0, \quad h_2 \approx 0.7616, \quad h_3 \approx -0.1187,$$

$$y_1 = 0.1, \quad y_2 \approx 0.7093, \quad y_3 \approx 0.0050.$$

This demonstrates how the hidden state evolves and influences the output over time.

4.8 Intuition

- At time step 1, the network had no meaningful past, so the output was near the bias. - At time step 2, the strong positive input increased the hidden state, leading to a larger output. - At time step 3, a negative input pulled the hidden state down, reducing the output.

In this example, we started with the fundamental equations of a simple RNN cell, chose a small numeric example to make the math transparent, and walked through each time step in detail. This approach provides insight into how the hidden state evolves and how the output is generated at each step. In more complex scenarios with multiple hidden units and longer sequences, RNNs can capture intricate temporal patterns.