RECURRENT NEURAL NETWORKS

JEAN-MARTIN ALBERT

1. Dealing with sequences

There is more than one way to extend a recurrent function to a set of sequences

2. Back Propagation

In general, in order to minimize a function $f(w)$ of one variable, we make use of the fact that the extreme values of f all have $f'(x) = 0$. If f is a function of two or more variables, then if x is such that $f(x)$ is a local max, or minimum, then all partial derivatives $\frac{\partial f}{\partial w_i}(x) = 0$. If $f : \mathbb{R}^n \to \mathbb{R}$ is a function, then the *gradient* of f is the vector $\nabla f(x) = \left(\frac{\partial f}{\partial w_1}(x), ..., \frac{\partial f}{\partial w_n}(x)\right)$. One interesting property of $\nabla f(x)$ is that, as a vector, it "points" in the direction in which f decreases the most. That is to say, there is a good chance that $f(x - \alpha\nabla f(x)) \leq f(x)$. The chance is better the smaller α is. We see that if we iterate that process, we get a sequence $x_1, ..., x_n, ...$ such that $f(x_{i+1}) \leq f(x_i)$ for every i. Since $m \leq f(x_i) \leq f(x_0)$ for every i, we get a bounded decreasing sequence of real number, which converges.

Caveat: this ideal situation doesn't allways happen.

This iterative process is *(Stocastic) gradient descent*.

Suppose f is a complsition of two functions, and that we can write $f(x, w) = g(h(x, w_1), w_2)$. Then we can write $\nabla_w f(x, w) = \nabla g(h(x, w_1), w_2) \cdot \nabla h(x, w_1)$. Each level of composition corresponds to a layer of neural network. The chain rule transforms function composition into a product. If a network becomes deep, then the derivative of the loss function becomes a long product. When gradients become small in norm (like they do when we approach a min), then the gradient becomes very close to 0, and the update rule for gradient descent stops changing the weights.

This is the vanishing gradient problem for very deep networks, and makes convergence slow.

3. The base of recurrent networks

Consider a function $f : U \times S \to V \times S$, where U, V and S are finite dimensional vector spaces. Note that every vector space has a a distinguished element 0, the zero-vector. Let $u_1, ..., u_n$ be a finite sequence of vectors in U. We construct a sequence of vectors in V as follows. Write $f(u_1, 0) = (v_1, s_1)$, and for every i, if $f(u_i, s_{i-1}) = (v_i, s_i)$. Note that this is a state machine, close to a deterministic finite automaton, except that here the state space S is infinite.

The set $\{0, 1\}^*$ of all finite sequences of 0's and 1's is countable, and S, being a real vector space, is uncountable. Therefore, we can encode every element $w \in$

Date: August 18, 2017.

$\{0,1\}^*$ as a vector in S. Informally, S is enough to encode any finite state space, and every possible value for the content of the tape of a Turing machine. We get:

Recurrent neural networks are Turing-complete.

which explains why recurrent neural networks seem to be able to produce results that other networks can't. Training a recurrent network is the same as producing a Turing machine.

4. Types of Recurrent Cells

4.1. **Basic Recurrent Cell.**

4.2. **Long Short-term Memory Cell.**

4.3. **Gated Recurrent Unit Cell.**

5. Examples

5.1. **Many-to-One.** The Buzzometer sentiment analysis tool uses a bi-directional GRU model.

5.2. **Many-to-many.** Text generator

5.3. **Many-to-one-to-many.** As an example of a recurrent neural network performing a task that ordinary neural network should not be able to perform, we present a sorting function which uses GRU cells. The architecture of the network is the reason we call it a many-to-one-to-many network. One level takes a list of numbers, and produces a vector which we can see as representing all the numbers in the list, and a second layer uses this output vector as input, and produces the original list, sorted in increasing order. The training was done using sequences of natural numbers $n \in \{1, ..., 32\}$, and each of the training sequences of length 32. After about 6 days of training, the accuracy of the network stabilized at around 95%, and it is worth noting that eventhough it took a long time to breach the 90% accuracy mark, very early on in the training did the network output increasing sequences.