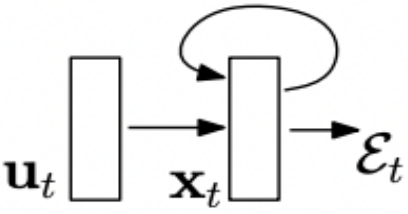
Recurrent Neural Networks-a theoretical analysis



Recurrent neural networks, or RNNs, are a family of neural networks for processing sequential data. They can process variable-length sequences and quickly scale up to longer sequences not seen during training. They use parameter sharing as their building block, enabling it to extend and apply the model to examples of different forms and generalize across them. If we had separate parameters during training, we could not generalize well nor share statistical strength across different sequence lengths and temporal positions. Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence. For example, in the sentence "I went to the fare to eat pizza and pasta" if we were to ask the machine learning model to read the sentence and infer the reason why he went to the fare, we would like it to recognize the keywords namely pizza and pasta as the relevant piece of information appears in the 8th or the 10th word of the sentence. A traditional fully connected feedforward network would have to learn separate parameters for each position of the sentence, while the RNNs can do the same more efficiently through weight sharing across several time steps. We use the same transitions function f with the same parameters at every time step.

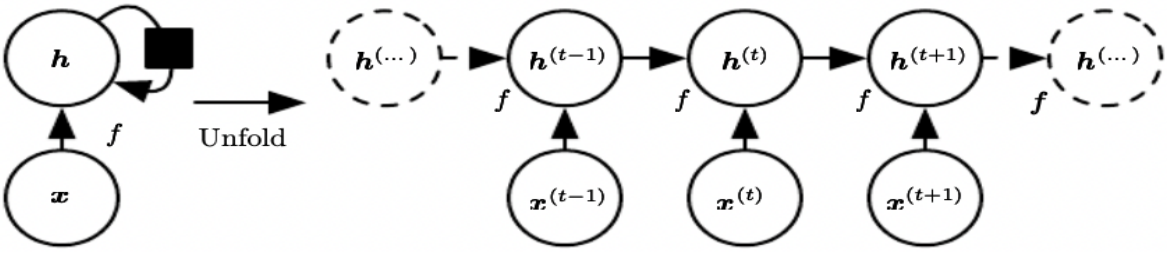


Figure 1: This recurrent neural network processes information from the input by incorporating it into the state (a summary vector) that is passed forward in time.

A recurrent neural network can be expressed as a computational graph that processes information through a set of nonlinear computations (transformations) by incorporating it into the state passed forward in time. The state vector is thought of as the summary vector of the input sequence up to the current timestep. It is required to selectively store the relevant aspects and ignore the others needed for the given language task. For example, if the RNN is used in statistical language modeling, typically to predict the next word given previous words, storing all the information in the input sequence up to time t may not be necessary, storing only enough information to predict the rest of the sentence is sufficient. The state vector must be rich enough to allow one to approximately recover the input sequence,as in autoencoder frameworks.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

The network typically learns to use $h^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past input sequence of inputs upto t. This summary maps an arbitrary length input sequence to a fixed length vector $h^{(t)}$.

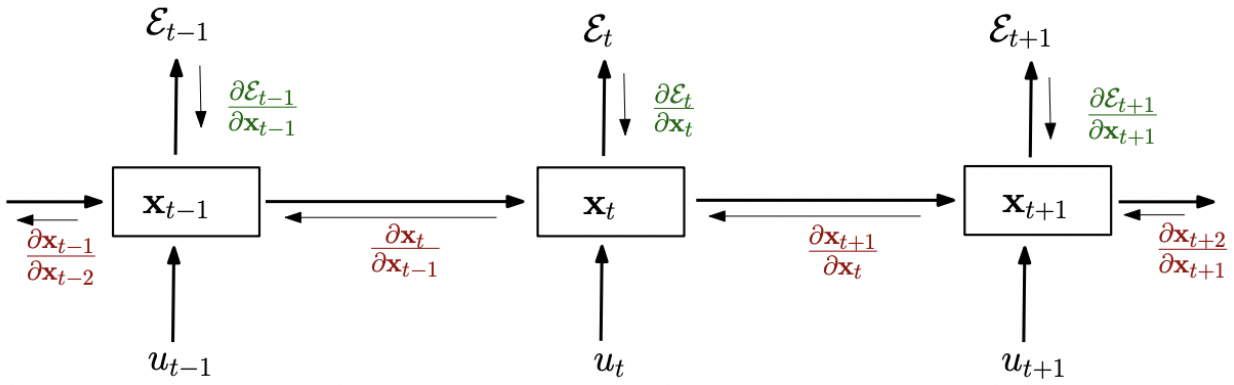


Figure 2: Unrolling an RNN through time by creating a copy of the model at every time step. We denote by xt the hidden state of the network at time t, by ut the input of the network at time t, and by t the error obtained from the output at time t.

The structure of the RNN is very similar to a multilayer perceptron, with the distinction that we allow connections among hidden units associated with a time delay. Through these connections, the model can retain information about past inputs, enabling it to discover temporal correlations between events that are possibly far away from each other.

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta) \qquad (1)$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \qquad (2)$$

The parameters of the model are given by the recurrent weight matrix Wrec, the biases b and the input weight matrix Win, collected in θ for the general case. x0 is provided by the user, set to zero or learned, and σ is an elementwise function (usually the tanh or sigmoid).
We compute the gradients using Backpropagation through time (BPTT) where the recurrent model is represented as a deep multilayer one, and BackpropagationBackpropagation is applied on the unrolled model.

Exploding and Vanishing gradients

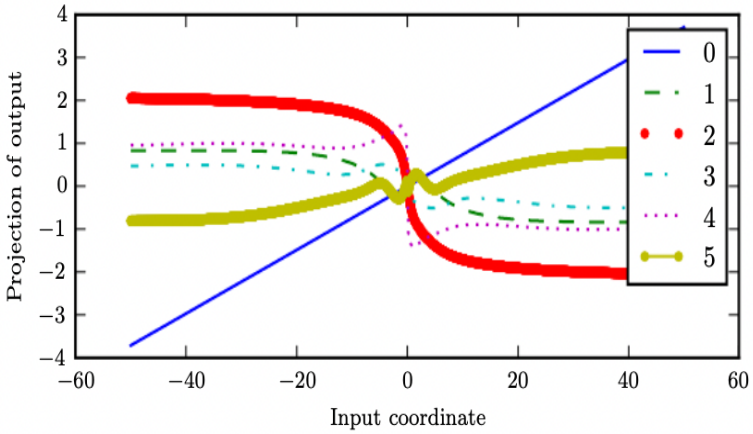


Figure 3: When composing many nonlinear functions (like the linear-tanh layer shown here), the result is highly nonlinear, typically with most of the values associated with a tiny derivative and some with a large derivative, and many alternations between increasing and decreasing. The mathematical challenge of training a recurrent neural net is mainly vanishing and exploding gradient problems. RNNs construct very deep computational graphs by repeatedly applying the same function at each time step of a long temporal sequence. For example, suppose we consider the path in the computational graph that consists of repeatedly multiplying by a matrix W.

After t steps, this recurrence relation describes the power method.

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top} \boldsymbol{h}^{(0)}$$

If W has an eigendecomposition of the form,

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}$$

With orthogonal Q, the recurrence can be simplified further to,

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}$$

The eigenvalues are raised to the power t, and thus any eigenvalue whose magnitude is less than one will decay, and those with a magnitude greater than one will explode. Since the gradients are also scales according to t, it causes the gradients to vanish and explode, thus a significant hindrance to learning long-term dependencies. As Bengio et al.(1994) introduced, exploding gradient problem refers to the substantial increase in the norm of the gradient during training. Such events are caused by the explosion of long-term components, which can grow exponentially more than short-term ones. The vanishing gradient problem refers to the opposite behavior when the long term goes exponentially fast to norm 0, making it impossible for the model to learn the correlation between temporally distant events.

As stated in Bengio et al.(1993), the RNN must enter a region of parameter space where the gradients vanish to store memories in a way that is robust to small noise. Specifically, whenever the model can represent long-term dependencies, the gradient of the long-term interaction has an exponentially smaller magnitude than that of a short-term interaction. Thus, the gradient-based optimization becomes increasingly difficult, with the probability of successful training of a traditional RNN rapidly reaching 0 even for sequences of length 10 or 20.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \tag{3}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^{+} \mathbf{x}_k}{\partial \theta} \right) \tag{4}$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^{T} diag(\sigma'(\mathbf{x}_{i-1})) \tag{5}$$

These equations were obtained by writing the gradients in the sum-of-products form. The ∂+xk/∂θ refers to the immediate partial derivative of the state xk with respect to. Equation (5) is equivalent to the product of t-k Jacobian matrices. In the same way, a product of t-k real numbers can shrink to zero or explode to infinity, and so does this product of matrices (along some direction v). Suppose we simplify this and consider a linear version

of the models (i.e., set σ to the identity function in equation (2)). In that case, we can use the power iteration method to formally analyze this product of Jacobian matrices and obtain tight conditions for when the gradients explode or vanish.

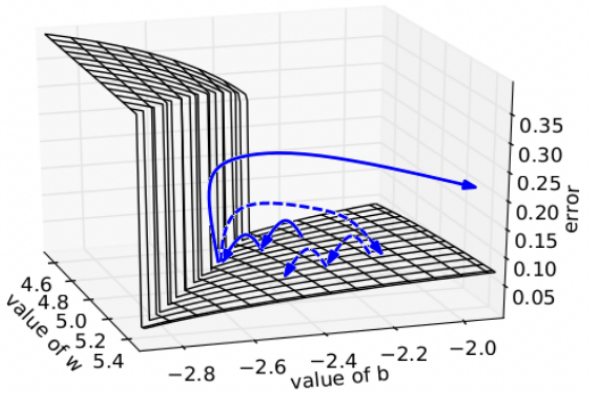


Figure 4: The error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid line depicts standard trajectories the gradient descent might follow.

The error surface is as follows:-

$$\mathcal{E}_{50} = (\sigma(x_{50}) - 0.7)^2$$

It is sufficient for the largest eigenvalue λ1 of the recurrent weight matrix W_{rec} to be smaller than 1 for the long term component to vanish(as t→ ∞) and necessary for it to be larger than 1 for gradients to explode. To analyze this problem, let us consider a simple one hidden unit model given by:-

$$x_t = w\sigma(x_{t-1}) + b$$

We can more easily analyze the behavior of this model by further simplifying it to linear (σ then being the identity function), with b=0. The equation now simplified as follows:-

$$x_t = x_0 w^t$$

The first-order and second-order derivatives now simplifies to equations given by:-

$$\frac{\partial x_t}{\partial w} = t x_0 w^{t-1} \quad \frac{\partial^2 x_t}{\partial w^2} = t(t-1) x_0 w^{t-2}$$

implying that when the first derivative explodes, so does the second derivative. In the general case, when the gradients explode, they do so along some directions v. This says that there exists a vector v such that,

$$\frac{\partial \mathcal{E}_t}{\partial \theta}\mathbf{v} \geq C\alpha^t$$

for C, αϵR and α>1. For the linear case, v is the eigenvector corresponding to the largest eigenvalue of W_{rec}.

In general when gradients explode, so does the curvature along v, leading to a wall in the error surface. We assume that the valley is wide, as we have a large region around where if we land, we can rely on first order methods to move towards the local minima. Suppose both the gradient and the leading eigenvector of the curvature are aligned with the exploding direction v. In that case, it follows that the error surface has a steep wall perpendicular to v (and consequently to the gradient). This means that when stochastic gradient descent (SGD) reaches the well and does a gradient descent step, it will be forced to jump across the valley, moving perpendicular to the steep walls, possibly leaving the valley and disrupting the learning process. It is important to note that the problem here is not the direction SGD is taking but the step size along that direction. Thus, when the gradients explode, the curvature and the higher-order derivatives also explode, and we are faced with a single steep wall in the error surface. To deal with the vanishing gradient problem, we use a regularization

term that forces the error signal not to vanish as it travels back in time. This regularization term forces the Jacobian matrices ∂xi/∂xi−1 to preserve the norm only in relevant directions. For the exploding gradient problem, we could perform gradient clipping or scale down the gradients whenever the norm of the gradient exceeds a given threshold (it can be thought of as adapting the learning rate based on the norm of the gradient). It can handle abrupt changes in the norm and thus helps in the training process of the recurrent neural network for longer sequences.

References

1. Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5.2 (1994): 157-166.
2. Bengio, Yoshua, Paolo Frasconi, and Patrice Simard. "The problem of learning long-term dependencies in recurrent networks." *IEEE international conference on neural networks*. IEEE, 1993.
3. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
4. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
5. Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International conference on machine learning*. PMLR, 2013.