Module 3, Lecture 4:
Recurrent LSTM Neural Networks

M. Vidyasagar

Distinguished Professor, IIT Hyderabad
Email: m.vidyasagar@iith.ac.in
Website: www.iith.ac.in/∼m_vidyasagar/

Outline

Outline

Outline

Memoryless Nature of Feedforward Neural Networks

The neural networks studied thus far are usually referred to as Feedforward Neural Networks (FFNNs). Multi-layer perceptron networks (MLPNs) are an example of FFNNs.

Such networks are essentially *memoryless*. A FFNN defines a nonlinear map $f(\mathbf{w}, \mathbf{x})$ that maps an input vector $\mathbf{x}$ into an output vector $f(\mathbf{w}, \mathbf{x})$, where $\mathbf{w}$ is the set of "weights" in the NN.

If we input a temporal sequence of inputs $\{\mathbf{x}(t)\}_{t \geq 0}$ to the FFNN,, then at time t the FFNN merely outputs $\{f(\mathbf{w}, \mathbf{x}(t))\}_{t \geq 0}$, *without regard to the previous history.*

The Need for Memory in Processing Temporal Signals

Take two sentences:

- "Would you like to have coffee?"
- "What kind of wood is in this furniture?"

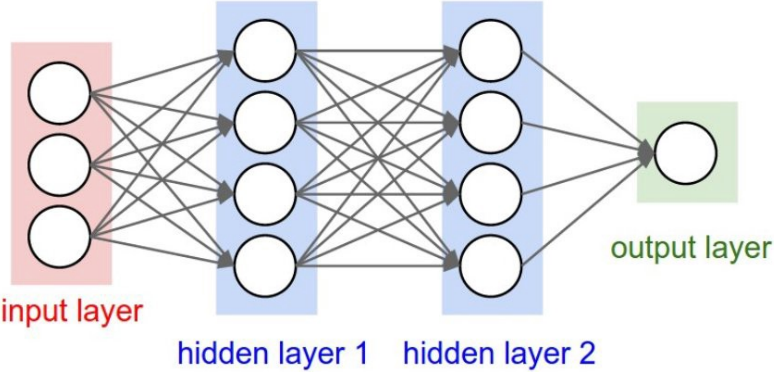
Most people would pronounce "would" and "wood" in an almost identical fashion. But the context makes it clear which is which.

Without taking the preceding (and succeeding) words into account, a naive speech processing system would fail to distinguish between the two. The same is true of image interpretation.

Recurrent Neural Networks (RNNs) provide for feeding back past (but not future) inputs in order to improve performance.

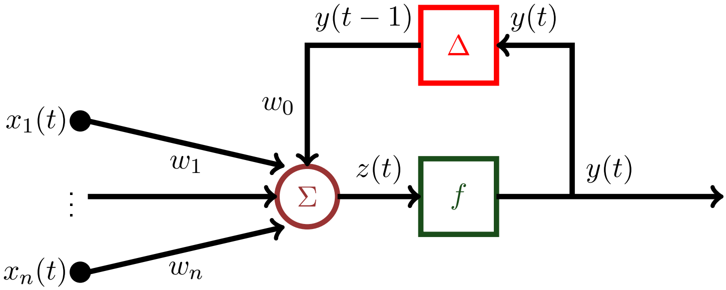
Outline

Feedforward Neural Network



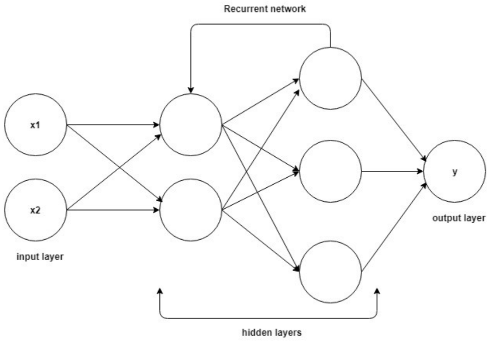
Source

Recurrent Neural Networks (Single Level)



Σ = Summing junction, f = Neuron , Δ = One-step delay.

Recurrent Neural Networks (Multiple Level)



Source

The feedback can go down multiple levels, and there can be multiple such feedbacks.

Outline

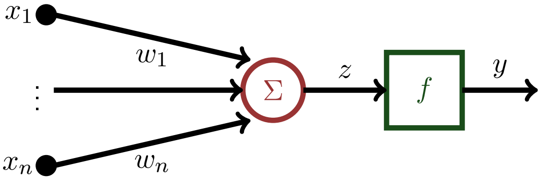
Back-Propagation Revisited

Start with a one-layer neural network (next slide).

Suppose each processing element is *continuously differentiable*.

This assumption holds with a sigmoid, but not with ReLU, perceptron, etc. The characteristics in those units need to smoothened out.

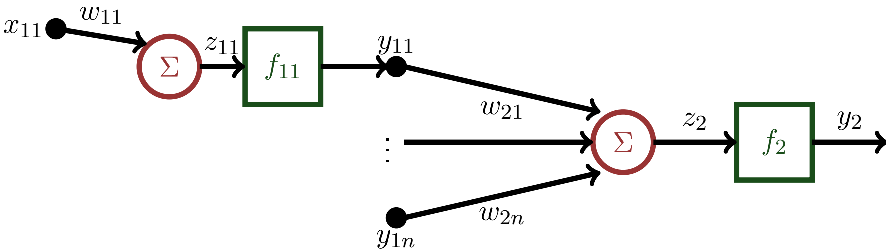
One-Layer Network



Chain rule:

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i} = f'(z) x_i.$$

Multi-Layer Network



Apply chain rule repeatedly!

$$\frac{\partial y_2}{\partial w_{11}} = \frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_{11}}\frac{\partial y_{11}}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{11}} = f_2'(z_2) w_{21} f_{11}'(z_{11}) x_{11}.$$

Back-Propagation Rule

Caution: From a given input, there might be *multiple paths* to the final output.

A partial derivative needs to be computed along *each path*, and then added up.

One way to do this: *Reverse* all the arrows in the FFNN, and trace out all the paths *from* the output *to* the node in question; then apply the previous formula.

This explains the name "back-propagation."

Back-Propagation for Minimizing the Least-Squares Error

Let $f(\mathbf{w}, \mathbf{x})$ denote the output of a FFNN with input $\mathbf{x}$ and parameter vector $\mathbf{w}$. The back-propagation procedure allows us to compute $\partial f(\mathbf{w}, \mathbf{x})/\partial w_i$ for each input $\mathbf{x}$.

Given labelled samples $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, m$, the usual procedure is to define the **least-squares error**

$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(\mathbf{w}, \mathbf{x}_i))^2,$$

and choose $\mathbf{w}$ to minimize J.

Back-Propagation (Cont'd)

Note that

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(\mathbf{w}, \mathbf{x}_i)) \frac{\partial f(\mathbf{w}, \mathbf{x})}{\partial w_i}.$$

Now we can use steepest descent, or add a momentum term, or whatever we wish.

Back-Propagation Through Time (BPTT)

In a recurrent neural network, instead of having pairs of *vectors* $(\mathbf{x}_i, y_i)$ to specify the desired output, we have instead pairs of *sequences vectors* indexed by time.

So the mean-squared error must also sum over time:

$$J(\mathbf{w}) = \frac{1}{m}\frac{1}{T}\sum_{i=1}^{m}\sum_{t=1}^{T}(y_i(t) - f(\mathbf{w}, \mathbf{x}_i(1:t)))^2.$$

Here the notation $\mathbf{x}_i(1:t)$ denotes the i-th input sequence $\mathbf{x}_i(1), \ldots, \mathbf{x}_i(t)$. The output $y_i(t)$ depends not only on the *current* input value $\mathbf{x}_i(t)$, but also the *past* input values.

Back-Propagation Through Time (Cont'd)

To apply back-propagation to recurrent networks, the same logic applies. We can compute

$$\frac{\partial y_t}{\partial y_{t-1}}$$

just as we can compute the partial derivatives with respect to any weights. Using this, we can compute $\partial J(\mathbf{w})/\partial w_i$.

Apart from more messy notation, the idea is the same as before: Compute the gradient of J with respect to the weights w_i (which are *not* changing with time), and then optimize. Link

IIT Hyderabad

Outline

Outline

The Vanishing Gradient Problem

Let $J(\mathbf{w})$ denote the mean-squared error to be minimized with respect to $\mathbf{w}$. If $\partial J(\mathbf{w})/\partial w_i \approx 0$, then gradient-based methods do not change w_i, and the training "gets stuck" at that value.

Because of back-propagation, the gradient $\partial J(\mathbf{w})/\partial w_i$ is a product of a whole lot of individual gradients.

If *any one* of these gradient terms is very small, the overall gradient $\partial J(\mathbf{w})/\partial w_i \approx 0$.

With standard sigmoid, the gradient is indeed small if the input is large (next slide).

Sigmoidal Nonlinearity

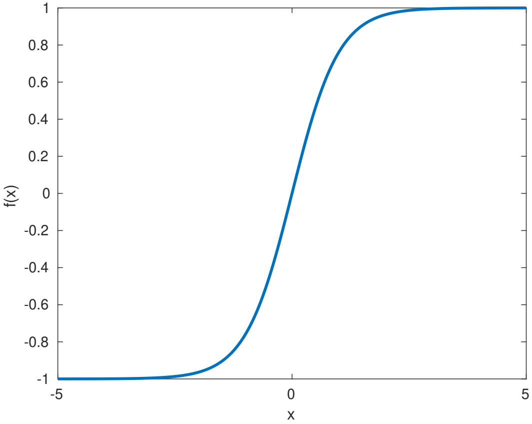


Figure: Sigmoidal Nonlinearity

Possible Remedies

Regularization: Add a "regularizing term" to the mean-squared error, and choose the weight vector $\mathbf{w}$ so as to minimize

$$J(\mathbf{w}) + \mu \|\mathbf{w}\|_2^2, \text{ where } \|\mathbf{w}\|_2^2 = \sum_{i=1}^{l} w_i^2.$$

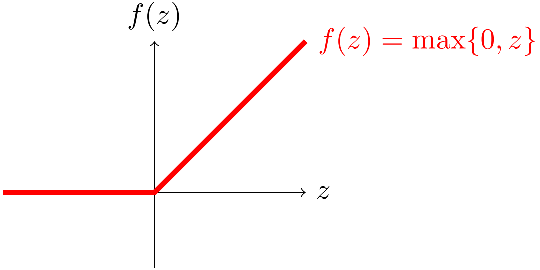
Here the weight μ gives a tradeoff between getting a good fit to training data and ensuring that the weights remain small.

Or we can try to ensure that the gradient does not vanish by replacing the sigmoid by a ReLU or other such neuron.

The Rectified Linear Unit (ReLU): Reprise

The Rectified Linear Unit (ReLU) is defined by

$$y = \begin{cases} z & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases}$$



Outline

Recurrent Neural Networks Revisited

A general RNN with feedbacks everywhere looks like this: ▸ Link

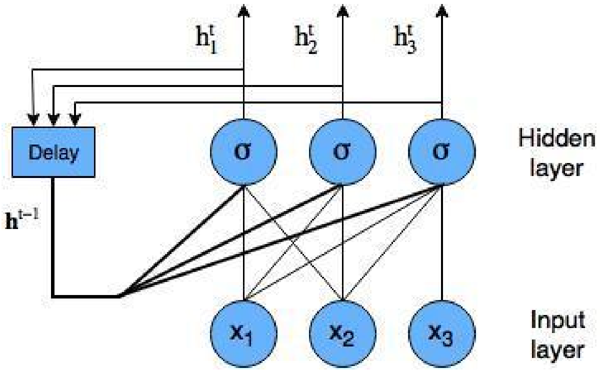


Figure: A "Complete" Recursive Neural Network

"Unrolled" Recursive Neural Network

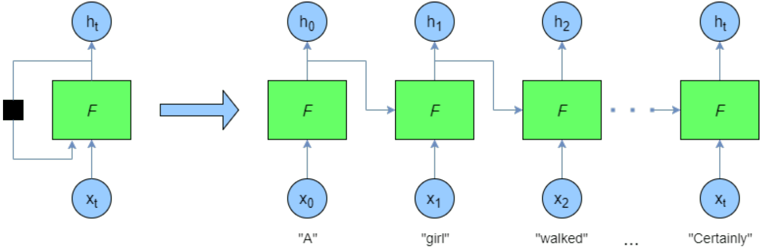


Figure: An "Unrolled" Recursive Neural Network

There are still too many connections! Link

Long Short-Term Memory (LSTM) Architecture

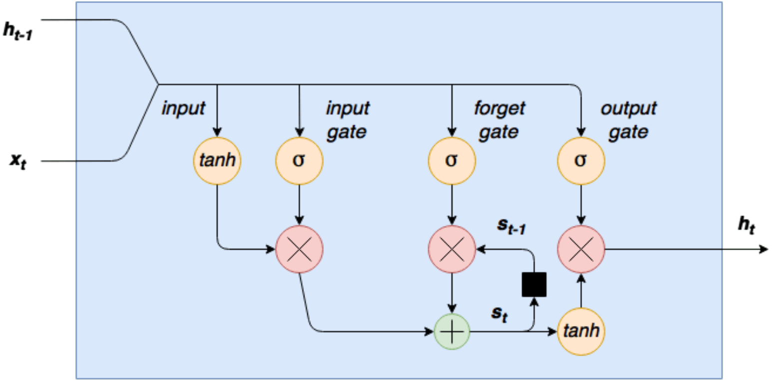


Figure: LSTM Architecture

There are far fewer connections. Link

Outline

1. Recurrent Neural Networks
   - The Role of Time and Context in Neural Networks
   - Architecture of Recurrent Neural Networks
   - Training RNNs: Back-Propagation Through Time
2. Long Short-Term Memory (LSTM) Neural Networks
   - Limitations of Recurrent Neural Networks
   - Architecture of LSTM Neural Networks
3. Gated Recurrent Units (GRUs)
4. Some Resources

IIT Hyderabad

Outline

Some Resources on RNNs and LSTM NNs

- An introduction to RNNs Link
- Another introduction to RNNs Link (Requires the creation of a free account)
- A description of five "enterprise" (companywide) applications of RNNs. Link

IIT Hyderabad