A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach

-Herbert Jaeger

Presented by: Shaowei Wang
Outline

• Recurrent neural network (RNN)
• Algorithms on training RNN
• Echo state network – special case of RNN
Recurrent neural network (RNN)

- Used to do sequence processing
- The output is fed back as input to others
- Allows loop

Figure 1.1: Typical structure of a feedforward network (left) and a recurrent network (right).
## Short characterization comparison

<table>
<thead>
<tr>
<th>Structure</th>
<th>Recurrent neural network</th>
<th>Feedforward network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>At least contain cycles</td>
<td>No cycle and the layers are clear</td>
</tr>
<tr>
<td></td>
<td>Can “remember” states</td>
<td></td>
</tr>
<tr>
<td>Input-output</td>
<td>time sequential data</td>
<td>no time series</td>
</tr>
<tr>
<td>publications</td>
<td>5%</td>
<td>95%</td>
</tr>
<tr>
<td>Training approaches</td>
<td>No clear winner</td>
<td>Most popular: backpropagation algorithm</td>
</tr>
</tbody>
</table>
Category

• **Discrete-time recurrent neural network (DTRNN)**
  – The processing occurs in discrete steps, as if the network was driven by an external clock

• **Continuous-time recurrent neural network (CTRNN)**
  – The processing occurs in continuous time
The difference in activation updating could be understood as time step.
Training approaches

- Backpropagation though time (BPTT)
- Real-time recurrent learning
- Extended Kalman filter
Backpropagation though time

- *Unfold the* discrete-time recurrent neural network into a multilayer feedforward neural network (FFNN) each time a sequence is processed.

- FFNN has a layer for each ``time step'' in the sequence, as if the ``time step'' is the index of the layer.

- Redirect the connection between layers.

- Use the standard backpropagation algorithm to train each FFNN form top to bottom.
Algorithm to update weights

\begin{align*}
\text{new } w_{ij} &= w_{ij} + \gamma \sum_{n=1}^{T} \delta_i(n) x_j(n-1) \quad [\text{use } x_j(n-1) = 0 \text{ for } n = 1] \\
\text{new } w_{ij}^{in} &= w_{ij}^{in} + \gamma \sum_{n=1}^{T} \delta_i(n) u_j(n) \\
\text{new } w_{ij}^{out} &= w_{ij}^{out} + \gamma \times \begin{cases} 
\sum_{n=1}^{T} \delta_i(n) u_j(n), & \text{if } j \text{ refers to input unit} \\
\sum_{n=1}^{T} \delta_i(n) x_j(n-1), & \text{if } j \text{ refers to hidden unit} 
\end{cases} \\
\text{new } w_{ij}^{back} &= w_{ij}^{back} + \gamma \sum_{n=1}^{T} \delta_i(n) y_j(n-1) \quad [\text{use } y_j(n-1) = 0 \text{ for } n = 1]
\end{align*}

• Since layers have been obtained by replicating the DTRNN over and over, weights in all layers should be the same.
• BPTT updates all equivalent weights using the sum of the gradients obtained for weights in all layers.
Drawback

• It’s hard to be used in the application where online adaption is required as the entire time series must be used.
  – One option (p-BPTT): truncate part of time instead of entire time.
    • Drawback: the ‘memory’ beyond truncated time can’t be captured by the model
Training approaches

• Backpropagation though time (BPTT)
• Real-time recurrent learning
• Extended Kalman filter
Real-time recurrent learning

- Compute the error gradient and update weights for each time step.
- During forward step, it compute the gradient of internal and output nodes with respects to all weights as the network.

![Diagram of a recurrent neural network with K input units, N internal units, and L output units. The diagram shows bidirectional connections between units, indicating the flow of information over time.](image)
Some of the units in $U$ are output units, for which a target is defined. A target may *not* be defined for every single input however. For example, if we are presenting a string to the network to be classified as either grammatical or ungrammatical, we may provide a target only for the last symbol in the string. In defining an error over the outputs, therefore, we need to make the error time dependent too, so that it can be undefined (or 0) for an output unit for which no target exists at present. Let $T(t)$ be the set of indices $k$ in $U$ for which there exists a target value $d_k(t)$ at time $t$. We are forced to use the notation $d_k$ instead of $t$ here, as $t$ now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases}$$

and define our error function for a single time step as

$$E(t) = \frac{1}{2} \sum_{k \in U} [e_k(t)]^2$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{total}(t_0, t_1) = \sum_{\tau = t_0 + 1}^{t_1} E(\tau)$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla_W E_{total}(t_0, t + 1) = \nabla_W E_{total}(t_0, t) + \nabla_W E(t + 1)$$

As a time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}}$$

http://www.willamette.edu/~gorr/classes/cs449/rtl.html
Training approaches

- Backpropagation though time (BPTT)
- Real-time recurrent learning
- Extended Kalman filter
Extended Kalman filter

• Kalman filter
  – a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error on linear system.

• Extented Kalman filter
  – A version working on nonlinear system
Objective function of Kalman filter

\[ \hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \]

\[ e_k = x_k - \hat{x}_k. \]

\[ P_k = E[e_k e_k^T]. \]

\[ K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \]

\[ = \frac{P_k^- H^T}{HP_k^- H^T + R}. \]

\( X_k \) the state of network at time-step \( k \),
\( X_k^\wedge \) is the posteriori state estimate of \( X_k \),
\( X_k^- \) is the priori state estimate of \( X_k \),
\( Z_k \) is the actual measurement, \( H X_k^- \) predict measurement
\( e_k \) is the posteriori estimate error

Minimize \( P_k \) (error covariance) as the objective function and get \( K_k \)

Discrete Kalman filter cycle

**Time Update ("Predict")**

1. Project the state ahead
   \[ \hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \]

2. Project the error covariance ahead
   \[ P_k^- = AP_{k-1}A^T + Q \]

**Measurement Update ("Correct")**

1. Compute the Kalman gain
   \[ K_k = P_k^-H^T(HP_k^-H^T + R)^{-1} \]

2. Update estimate with measurement \( z_k \)
   \[ \hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \]

3. Update the error covariance
   \[ P_k = (I - K_kH)P_k^- \]

Initial estimates for \( \hat{x}_{k-1} \) and \( P_{k-1} \)
Discrete extended Kalman filter cycle

Nonlinear function

**Time Update (“Predict”)**

1. Project the state ahead
   \[ \hat{x}_k^* = f(\hat{x}_{k-1}, u_{k-1}, 0) \]
2. Project the error covariance ahead
   \[ P_k^* = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \]

**Measurement Update (“Correct”)**

1. Compute the Kalman gain
   \[ K_k = P_k^* H_k^T (H_k P_k^* H_k^T + V_k R_k V_k^T)^{-1} \]
2. Update estimate with measurement \( z_k \)
   \[ \hat{x}_k = \hat{x}_k^* + K_k (z_k - h(\hat{x}_k^*, 0)) \]
3. Update the error covariance
   \[ P_k = (I - K_k H_k) P_k^* \]
Adapt to RNN

Time update (predict)

\[ w(n+1) = w(n) + q(n) \]
\[ d(n) = h(w, u(0), ..., u(n)) \]
\[ e = z(n) - d(n) \]

Measurement update (correct)

\[ K(n) = P(n)H(n)[H(n)' P(n)H(n)]^{-1} \]
\[ \hat{w}(n+1) = \hat{w}(n) + K(n)\xi(n) \]
\[ P(n+1) = P(n) - K(n)H(n)' P(n) + Q(n) \]

\[ K(n) = P(n)H(n)[(1/\eta)I + H(n)' P(n)H(n)]^{-1}, \]

where \( d(n) \) is the desired output, \( w(n) \) is the weights. \( n \) is the time step. \( H(n) \) is the derivative of \( h() \). The information of \( d(n) \) is incorporated into \( P(n) \) to update the Kalman gain function \( K(n) \).

Interpret the weights \( w \) of the RNN as the state of a dynamical system.
Outline

• Recurrent neural networks
• Algorithms on training RNN
• Echo state network – special case of RNN
Example – sinewave

- $D(n) = \frac{1}{2} \sin(n/4)$
- Task: remember the $d(n)$ by using 300-step sequence of $d(n)$ as the teacher signal (training data), without input.

Figure 6.2: Schematic setup of ESN for training a sinewave generator.
* The echo of sinewave $d(n)$ is kept in the echo state network.

**Figure 6.3.** The dynamics within the DR induced by teacher-forcing the sinewave $d(n)$ in the output unit. 50-step traces of the 20 internal DR units and of the teacher signal (last plot) are shown.
Features of ESN

• It has a sparsely connected hidden layer (with typically 1% connectivity).
• The weights of hidden neurons are randomly assigned and are fixed.
• The weights of output neurons can be learned and produce (echo) specific pattern.

http://en.wikipedia.org/wiki/Echo_state_network
Whether a network has echo state property?

• The property is only dependent on W (weights of hidden layers), not dependent on $W_{\text{back}}$ and $W_{\text{in}}$.

• If spectral radius $|\lambda_{\text{max}}| > 1$, no echo state property, where $\lambda_{\text{max}}$ the max value of eigenvector of W.

• If spectral radius $|\lambda_{\text{max}}| < 1$, have echo state property.
How to estimate the $W^{out}$?

• The desired output weights $W^{out}$ are the linear regression weights of the desired outputs $d(n)$ on the states of network $x(n)$

• Minimize the mean squared error (MSE)

$$MSE = \sum_{n=1}^{n=T} (d(n) - \sum_{i=1}^{\mid W^{out} \mid} w^{out}_{i} x_{i}(n))^2$$
Step 1. Procure an untrained DR network \((W^{in}, W, W^{back})\) which has the echo state property, and whose internal units exhibit mutually interestingly different dynamics when excited.

This step involves many heuristics. The way I proceed most often involves the following substeps.

1. Randomly generate an internal weight matrix \(W_0\).
2. Normalize \(W_0\) to a matrix \(W_1\) with unit spectral radius by putting \(W_1 = 1/|\lambda_{\max}| W_0\), where \(|\lambda_{\max}|\) is the spectral radius of \(W_0\). Standard mathematical packages for matrix operations all include routines to determine the eigenvalues of a matrix, so this is a straightforward thing.
3. Scale \(W_1\) to \(W = \alpha W_1\), where \(\alpha < 1\), whereby \(W\) obtains a spectral radius of \(\alpha\).
4. Randomly generate input weights \(W^{in}\) and output backpropagation weights \(W^{back}\). Then, the untrained network \((W^{in}, W, W^{back})\) is (or more honestly, has always been found to be) an echo state network, regardless of how \(W^{in}, W^{back}\) are chosen.

Small \(a\) for fast teacher dynamics, otherwise big \(a\)
Step 2. Sample network training dynamics.

This is a mechanical step, which involves no heuristics. It involves the following operations:

1. Initialize the network state arbitrarily, e.g. to zero state \( x(0) = 0 \).
2. Drive the network by the training data, for times \( n = 0, ..., T \), by presenting the teacher input \( u(n) \), and by teacher-forcing the teacher output \( d(n-1) \), by computing

\[
(6.10) \quad x(n+1) = f(W^{in} u(n+1) + W x(n) + W^{back} d(n))
\]

3. At time \( n = 0 \), where \( d(n) \) is not defined, use \( d(n) = 0 \).
4. For each time larger or equal than an initial washout time \( T_0 \), collect the network state \( x(n) \) as a new row into a state collecting matrix \( M \). In the end, one has obtained a state collecting matrix of size \( (T - T_0 + 1) \times (K + N + L) \).
5. Similarly, for each time larger or equal to \( T_0 \), collect the sigmoid-inverted teacher output \( \tanh^{-1} d(n) \) row-wise into a teacher collection matrix \( T \), to end up with a teacher collecting matrix \( T \) of size \( (T - T_0 + 1) \times L \).

Note: Be careful to collect into \( M \) and \( T \) the vectors \( x(n) \) and \( \tanh^{-1} d(n) \), not \( x(n) \) and \( \tanh^{-1} d(n-1) \)!
Learning algorithm - 3

Step 3: Compute output weights.

1. Concretely, multiply the pseudoinverse of $\mathbf{M}$ with $\mathbf{T}$, to obtain a $(K + N + L) \times L$ sized matrix $(\mathbf{W}_{\text{out}}^\dagger)^t$ whose $i$-th column contains the output weights from all network units to the $i$-th output unit:

\[
(\mathbf{W}_{\text{out}}^\dagger)^t = \mathbf{M}^{-1}\mathbf{T}.
\]

(6.11)

Every programming package of numerical linear algebra has optimized procedures for computing pseudoinverses.

2. Transpose $(\mathbf{W}_{\text{out}}^\dagger)^t$ to $\mathbf{W}_{\text{out}}$ in order to obtain the desired output weight matrix.
Thank you!