Backprop for recurrent networks

Sebastian Seung
Other flavors of backprop

• Recurrent backprop
  – for training steady states of a recurrent network

• Backprop-through-time
  – for training trajectories of a recurrent network
Steady state of a recurrent network

• Recurrent network

\[ x_i = f \left( \sum_j W_{ij} x_j + b_i \right) \]

– feedforward network is a special case

• Goal: maximize some function of the activity vector

\[ \max_{W,b} R(x) \]
Implicit function theorem

- $x$ is an implicit function of $W$ and $b$
- defined as the solution of $F(x, W, b) = 0$
- where $F(x, W, b) \equiv x - f(Wx + b)$
- assuming nonsingular Jacobian $\partial F_i / \partial x_j$
- This is a local definition.
Recurrent backpropagation

• Find steady state \( x = f(Wx + b) \)

• Calculate slopes \( D = \text{diag}\{f'(Wx + b)\} \)

• Solve for sensitivity \( (D^{-1} - W^T)\hat{u} = \frac{\partial R}{\partial x} \)

• Weight update \( \Delta W = \eta \hat{u}x^T \)
Sensitivity lemma

\[ \frac{\partial R}{\partial W_{ij}} = \frac{\partial R}{\partial b_i} x_j \]

- It is sufficient to calculate the derivative with respect to the biases.
- Recurrent backprop is a way of calculating the sensitivities

\[ \frac{\partial R}{\partial b_i} \equiv \hat{u}_i \]
Input as a function of output

• What input $b$ is required to make $x$ a steady state?

$$b_i = f^{-1}(x_i) - \sum_j W_{ij} x_j$$

• This is unique, even when output is not a unique function of the input!
Jacobian matrix

\[ b_i = f^{-1}(x_i) - \sum_j W_{ij} x_j \]

\[ \frac{\partial b_i}{\partial x_j} = f^{-1}(x_i) \delta_{ij} - W_{ij} \]

\[ = \left( D^{-1} - W \right)_{ij} \]
Composition of functions

• This composition of functions may seem more natural
  \[ b \rightarrow x \rightarrow R \]

• But this composition can also be defined
  \[ x \rightarrow b \rightarrow R \]

• Both definitions are locally valid.
Chain rule

\[
x \rightarrow b \rightarrow R
\]

\[
\frac{\partial R}{\partial x_j} = \sum_i \frac{\partial R}{\partial b_i} \frac{\partial b_i}{\partial x_j}
\]

\[
= \sum_i \frac{\partial R}{\partial b_i} \left( D^{-1} - W \right)_{ij}
\]

\[
\frac{\partial R}{\partial x} = \left( D^{-1} - W^T \right) \frac{\partial R}{\partial b}
\]
Trajectory learning

• Initialize at $x(0)$, iterate for $T$ time steps

\[ x_i(t) = f \left( \sum_j W_{ij} x_j(t-1) + b_i \right) \]

• Goal: maximize some function of the time series of activity vectors

\[ \max_{W,b} R(x(1), \ldots, x(T)) \]
Backpropagation through time

- Multilayer perceptron
  - Same number of neurons in each layer
  - Same weights and biases in each layer (weight-sharing)
Forward pass

• Initial condition $x(0)$

\[ u(t) = Wx(t - 1) + b(t) \]

\[ x(t) = f(u(t)) \]
Backward pass

- Final condition \( \hat{u}(T + 1) = 0 \)

\[
\hat{x}(t) = W^T \hat{u}(t + 1) + \frac{\partial R}{\partial x(t)}
\]

\[
\hat{u}(t) = D(t) \hat{x}(t)
\]
Weight update

\[ \Delta W = \eta \sum_t \hat{u}(t)x(t - 1)^T \quad \Delta b = \eta \sum_t \hat{u}(t) \]
Sensitivity lemma

• Suppose that weights and biases are functions of time:

\[ x_i(t) = f\left(\sum_j W_{ij}(t)x_j(t-1) + b_i(t)\right) \]

\[ \frac{\partial R}{\partial W_{ij}(t)} = \frac{\partial R}{\partial b_i(t)} x_j(t) \]
Sensitivity lemma

• If the weights and biases are constant in time,

\[
\frac{\partial R}{\partial W_{ij}} = \sum_t \frac{\partial R}{\partial W_{ij}(t)} \quad \frac{\partial R}{\partial b_i} = \sum_t \frac{\partial R}{\partial b_i(t)}
\]

\[
= \sum_t \frac{\partial R}{\partial b_i(t)} x_j(t)
\]
Input as a function of output

\[ x(t) = f(Wx(t-1) + b(t)) \]
\[ b(t) = f^{-1}(x(t)) - Wx(t-1) \]

\[ x(1), x(2), \ldots, x(T-1), x(T) \]
\[ b(1), b(2), \ldots, b(T-1), b(T) \]
Jacobian matrix

\[ b(t) = f^{-1}(x(t)) - Wx(t - 1) \]

\[ \frac{\partial b_i(t)}{\partial x_j(t')} = \delta_{tt'}(D^{-1}(t))_{ij} - W_{ij} \delta_{t-1,t'} \]

\[ D(t) = \text{diag}\{f'(Wx(t - 1) + b(t))\} \]
Chain rule

\[
\frac{\partial R}{\partial x_j(t')} = \sum_{i,t} \frac{\partial R}{\partial b_i(t)} \frac{\partial b_i(t)}{\partial x_j(t')} = \sum_i \hat{u}_i(t')(D^{-1}(t'))_{ij} - \sum_i \hat{u}_i(t' + 1) W_{ij}
\]

\[
\frac{\partial R}{\partial x(t)} = D^{-1}(t)\hat{u}(t) - W^T\hat{u}(t + 1)
\]
Lagrangian method

• Application of the chain rule can be confusing.

• Lagrange multipliers provide
  – a “turn the crank” method of calculating gradients
  – another interpretation of the backward pass
Lagrangian

\[ L(x, \hat{u}, W, b) = R(x) - \hat{u}^T [f^{-1}(x) - Wx - b] \]
Stationary point

\( x^*(W, b) \) and \( \hat{u}^*(W, b) \) such that

\[
0 = -\frac{\partial L}{\partial \hat{u}} = f^{-1}(x) - Wx - b
\]

\[
0 = \frac{\partial L}{\partial x} = \frac{\partial R}{\partial x} - (D^{-1} - WT)\hat{u}
\]
The Lagrangian equals the reward function at a stationary point

\[ R(x^*(W, b)) = L(x^*(W, b), \hat{u}^*(W, b), W, b) \]
Sensitivities of the Lagrangian and reward function

• All derivatives are evaluated at the stationary point.

\[
\frac{\partial R}{\partial b_j} = \sum_i \frac{\partial L}{\partial x_i} \frac{\partial x_i^*}{\partial b_j} + \sum_i \frac{\partial L}{\partial \hat{u}_i} \frac{\partial \hat{u}_i^*}{\partial b_j} + \frac{\partial L}{\partial b_j}
\]

• Similarly

\[
\frac{\partial R}{\partial W_{ij}} = \frac{\partial L}{\partial W_{ij}}
\]
The reward gradients

\[
\frac{\partial}{\partial W} R(\mathbf{x}^*(W, b)) = \frac{\partial L}{\partial W} = \hat{\mathbf{u}} \mathbf{x}^T
\]

\[
\frac{\partial}{\partial b} R(\mathbf{x}^*(W, b)) = \frac{\partial L}{\partial b} = \hat{\mathbf{u}}
\]

• where it’s understood that

\[
\hat{\mathbf{u}} = \hat{\mathbf{u}}^*(W, b) \quad \mathbf{x} = \mathbf{x}^*(W, b)
\]