1 Backward Propagation

Do one of the following three problems.

**Problem 4.1.1**

Recurrent neural network is proven effective in learning time-dependent signals. In addition to a feed-forward structure, a recurrent neural network also has a feedback loop that feeds the output of the neural network back to the bottom layer as an input in the next time frame. Technically it is no longer a feed-forward network, but if we regard nodes at different time as different nodes, then it is something close to a feedforward structure, and backward propagation algorithm applies. The algorithm is called backward propagation through time.

Let’s look at a simplest case, where there is one input layer, one hidden layer and one output node.

\[
\begin{align*}
\mathbf{a}^{(n)}_1 &= \mathbf{b}^{(n)}_1 + W_1 \mathbf{x}^{(n)}_0 \\
\mathbf{x}^{(n)}_1 &= \tanh \left( \mathbf{a}^{(n)}_1 \right) \\
\mathbf{a}^{(n)}_2 &= \mathbf{b}^{(n)}_2 + \mathbf{w}^T \mathbf{x}^{(n)}_1 \\
\mathbf{y}^{(n)} &= \tanh \left( \mathbf{a}^{(n)}_2 \right)
\end{align*}
\]

where superscript \((n)\) is the time index, and

\[
\tanh (x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]

The output is fed back as an input for the next time

\[
\mathbf{x}^{(n+1)}_0(1) = \mathbf{y}^{(n)}
\]

where \(\mathbf{x}^{(n+1)}_0(1)\) is the first element of the vector \(\mathbf{x}^{(n+1)}_0\).

The loss function is defined as

\[
\mathcal{L} = \sum_{n=1}^{N} \left( \mathbf{y}^{(n)} - \mathbf{t}^{(n)} \right)^2
\]

where \(\mathbf{t}^{(n)}\) is some label and \(N\) is the total number of time bins.

Define

\[
\delta^{(n)}_2 = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(n)}_2} \\
\delta^{(n)}_1 = \nabla_{\mathbf{a}^{(n)}_1} \mathcal{L}
\]
Homework 4

Express $\delta_2^{(n)}$ in terms of $\delta_1^{(n+1)}$, $\ell^{(n)}$, $y^{(n)}$, $a_2^{(n)}$, $W_1$ and $w_2$. No gradient or derivative operators should show up in your answer.

**Problem 4.1.2**

Convolutional neural network is designed to leverage spatial and temporal translation invariance. A hidden nodes in a upper layer accepts only nodes from the lower layer that are adjacent in time and/or space. Shared weights are introduced to impose translation invariance assumption.

Here, a convolutional neural network that is greatly simplified and slightly modified is presented to test your understanding about backward propagation. There is one input layer, one hidden layer and one output node.

\[
\begin{align*}
a_1^{(n)} &= b_1^{(n)} + \sum_{m=-\tau_0}^{\tau_0} W_1^{(m)} x_0^{(n+m)} \\
x_1^{(n)} &= \tanh(a_1^{(n)}) \\
a_2^{(n)} &= b_2 + \sum_{m=-\tau_1}^{\tau_1} w_2^{(m)T} x_1^{(n+m)} \\
y^{(n)} &= \tanh(a_2^{(n)})
\end{align*}
\]

where superscript $(n)$ is the time index, and

\[\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}\]

The loss function is defined as

\[L = \sum_{n=1}^{N} \left(y^{(n)} - \ell^{(n)}\right)^2\]

where $\ell^{(n)}$ is some label and $N$ is the total number of time bins.

Define

\[
\begin{align*}
\delta_2^{(n)} &= \frac{\partial L}{\partial a_2^{(n)}} \\
\delta_1^{(n)} &= \nabla_{a_1^{(n)}} L
\end{align*}
\]

Express $\delta_1^{(n)}$ in terms of $a_1^{(n)}$, $\delta_2^{(n-\tau_2)}, \ldots, \delta_2^{(n+\tau_2)}$ and $w_2^{(-\tau_2)}, \ldots, w_2^{(\tau_2)}$. No gradient or derivative operators should show up in your answer.

**Problem 4.1.3**

“Spiral network” is a brand new architecture proposed by Prof. Hasegawa-Johnson for the midterm exam of ECE544 in 2013. Here is a slightly modified version. A spiral network has $M$ number of nodes, denoted as $x(1), x(2), \ldots, x(M)$, and each node is defined as

\[
a(i) = \sum_{j=\max(i-N,1)}^{i-1} W(i, j)x(j)
\]

\[
x(i) = \tanh[a(i)] = \frac{e^{a(i)} - e^{-a(i)}}{e^{a(i)} + e^{-a(i)}}
\]
where \(N\) is an integer smaller than \(M\). For simplicity, the superscript \((n)\) as appears in the course slides is dropped.

The loss function is defined as

\[
\mathcal{L} = \sum_{i=1}^{N} (x(i) - t(i))^2
\]

where \(t(i)\) is some label.

Define

\[
\delta(i) = \frac{\partial \mathcal{L}}{\partial a(i)}
\]

Express \(\delta(i)\) in terms of \(\delta(i + 1), \ldots, \delta(N), a(i), t(i)\) and \(W\). No gradient or derivative operators should show up in your answer.

## 2 Minimum Cross Entropy Criterion and Softmax Nonlinearity

Do one of the following three problems.

**Problem 4.2.1**

Consider a \(M\)-class classification problem, where \((x, c)\) are input feature random vector and class random variable respectively. The class variable \(c\) takes the value from 1 to \(M\).

\[
P(c = i) = \pi_i
\]

The joint probability of the feature vector, conditional on each class, is a multivariate Gaussian with fixed covariance matrix:

\[
p_{x|c}(x'|i) = \frac{1}{\sqrt{(2\pi)^M |\Sigma|}} \exp \left( -\frac{(x'-\mu_i)^T \Sigma^{-1} (x'-\mu_i)}{2} \right)
\]

A set of training tokens \(\{(x^{(n)}, c^{(n)})\}\) are independently drawn from the joint distribution of \((x, c)\).

To solve the classification problem, we build a simple linear classifier with multiple outputs:

\[
a^{(n)} = b + Wx^{(n)}
\]

where \(a^{(n)}\) is an \(M\)-by-1 vector. The output of the classifier \(y^{(n)}\), which is also an \(M\)-by-1 vector, is obtained by passing \(a^{(n)}\) through the softmax nonlinearity function:

\[
y^{(n)}(i) = \frac{\exp(a^{(n)}(i))}{\sum_{j=1}^{M} \exp(a^{(n)}(j))}
\]

We apply the cross entropy as the loss function

\[
\mathcal{L} = \sum_n \sum_i t^{(n)}(i) \log \left( y^{(n)}(i) \right)
\]

where \(t^{(n)}(i)\) is the class indicator vector, which equals 1 iff \(c^{(n)} = i\) and 0 otherwise.

Suppose there are sufficient number of training tokens. Find the optimal \(W\) and \(b\). You need to justify your answer using Bayesian framework.

**Hint:** This simple structure can achieve Bayes optimal result, namely the posterior expectation.
Problem 4.2.2

Same as 1.1, but the observations are $d$-variate Bernoulli:

$$p_{x'|c}(x'|i) = \prod_{j=1}^{d} \lambda_{ji}^{x'_j} (1 - \lambda_{ji})^{1-x'_j}$$

where $x'_j$ is the $j$-th element of $x'(j)$.

Problem 4.2.3

Same as 1.1, but the observations are $d$-variate exponential:

$$p_{x'|c}(x'|i) = \prod_{j=1}^{d} \lambda_{ji} e^{-\lambda_{ji} x'_j}$$

where $x'_j$ is the $j$-th element of $x'(j)$. 