School of Computing and Information Systems
The University of Melbourne
COMP30027 Machine Learning (Semester 1, 2019)

Tutorial sample solutions: Week 10

1. A single–layer neural network is often referred to as a **perceptron**.

   (a) Why is a perceptron (which uses a **sigmoid** activation function) equivalent to **logistic regression**?

   - A perceptron has a weight associated with each input (attribute); the output is acquired by applying the activation function ($f(x) = \frac{1}{1+e^{-x}}$) to the linear combination of inputs ($w_0 a_0 + w_1 a_1 + \cdots$), which simplifies to $f(\sum w_i a_i)$ — this is the same as the logistic regression function.

   (b) What makes a (feed–forward) neural network more interesting?

   - Basically, each node in the first layer is a logistic regression model. However, these values are themselves the input to another layer, so that we are effectively progressively stacking lots of logistic regression models.

2. Why is a neural network suitable for **deep learning**? What is significant about the representation that we attempt to learn?

   - Hypothetically, the weights across the network describe some useful properties of the model. In effect, we hope to auto–magically engineer the necessary features to solve our problem, based only on the simplest inputs (where we hopefully haven't already introduced a bias).
   - The most interesting about the representation is that it is simultaneously useful for solving the problem and un-interpretable by humans. Although the claims of the automaton "learning" are probably over–blown, it can indeed discover properties of the data that were previously unknown to be useful (given enough data).
   - The subsequent "embedding" (weights of the final hidden layer of neurons) can also be used as a dense instance representation, which is sometimes helpful in unexpected problems.

3. In the context of `word embeddings`, why do we need **negative sampling**? Word embeddings are vector representations of words where similar words are closer together. During training of `CBOW` one of word2vec variations, we try to predict a word from its context. The loss function is:

$$J = \frac{1}{T}\sum_{i=1}^{T} \log \frac{\exp\left(\mathbf{w}_i^\top \sum_{j\in[-c,+c],j\neq 0} \tilde{\mathbf{w}}_{i+j}\right)}{\sum_{k=1}^{V} \exp\left(\mathbf{w}_k^\top \sum_{j\in[-c,+c],j\neq 0} \tilde{\mathbf{w}}_{i+j}\right)}$$

   where w_i is a word and $\tilde{w}$ is its context. As you can see in the formula, we are using softmax to normalise the word-context similarity into a probability distribution. In the denominator we have a some over all vocabulary which is very costly to calculate. To avoid this costly computation, researchers have proposed negative sampling as an approximation. In negative sampling, instead of trying to maximise the probability of the target word given its context, we change the loss function so that the probability of the target word in its context is more than the probability of a non-target word in that context.

$$J = -\sum_{w_i \in V} [\log\sigma(\mathbf{w}_i^\top c_i) + \sum_{j=1}^{k} \log\sigma(\mathbf{w}_j^\top c_i)]$$

   where k is the number of negative samples, and $c_i = \sum_{j\in[-c,+c],j\neq 0} \tilde{\mathbf{w}}_{i+j}$.

   `word2vec` implementation:
   Consider a set D of correct word-context pairs, and a set $\tilde{D}$ of incorrect word-context pairs. The

goal of the algorithm is to estimate the probability $P(D = 1|w, c)$ that the word-context pair came from the correct set D. This should be high (1) for pairs from D and low (0) for pairs from $\tilde{D}$. The probability constraint dictates that $P(D = 1|w, c) = 1 - P(D = 0|w, c)$. The probability function is modelled as a sigmoid over score $s(w, c)$:

$$P(D = 1|w, c) = \frac{1}{1 + e^{-s(w,c)}}$$

where $s(w, c) = w.c$.

The corpus level objective is:

$$\mathcal{L}(\Theta; D, \overline{D}) = \sum_{(w,c)\in D} \log P(D = 1|w, c) + \sum_{(w,c)\in\overline{D}} \log P(D = 0|w, c)$$

So we start sampling positive and negative word and context words, extract their embeddings, and try to maximise the probability of positive examples, and minimise the probability of negative examples.

4. Describe the mathematical formula of a multilayer perceptron with 1 hidden layer. Assume the input size is 1000, the hidden layer size is 100, and the output size is 20. Identify the parameters of the model, and their size.

   The mathematical formula for a MLP with one hidden layer is: $f(x) = g_2(g_1(x * W_1 + b_1) * W_2 + b_2)$, where $x \in (1, 1000)$, $W_1 \in (1000, 100)$, $b_1 \in (1, 100)$, $W_2 \in (100, 20)$, $b_2 \in (1, 20)$, and $g1$ is an activation function such as ReLU or tanh, and g_2 is softmax.