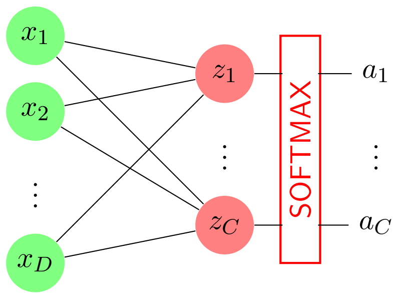
UNIVERSITY OF OXFORD

Machine Learning
Michaelmas Term 2021
Week 8

Problem Sheet 4

1 Backpropagation for Multiclass Logistic Regression

We can view multiclass logistic regression as a simple neural network with no hidden layer. Let us suppose that the inputs are D dimensional and that there are C classes. Then the input layer has D units, simply the inputs $x_1, \ldots, x_D$. The output layer has C units; there is full connection between the layers, *i.e.,* every input unit is connected to every output unit. Bias terms for each unit in the output layer are added separately. Finally, there is a non-linear activation function on the output layer, the softmax function. Pictorially, we may represent the neural network as follows:



Dropping superscripts to denote layers, as they are unnecessary in this case, and using the notation in the lectures, we have:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{1.1}$$
$$\mathbf{a} = \text{softmax}(\mathbf{z}) \tag{1.2}$$

Above $\mathbf{W}$ is a $C \times D$ matrix, and $\mathbf{b}$ is a column vector in C dimensions. Recall that the softmax function is defined as:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}} \tag{1.3}$$

For a point $(\mathbf{x}, y)$ in the training data, $y \in \{1, \ldots, C\}$. The model parameters to be learnt are $\mathbf{W}$ and $\mathbf{b}$. We will use the cross entropy loss function, so the contribution of the point $(\mathbf{x}, y)$ to the objective function is given by:

$$\ell(\mathbf{W}, \mathbf{b}; \mathbf{x}, y) = -\log a_y \tag{1.4}$$

Write the expressions for $\frac{\partial \ell}{\partial \mathbf{a}}$, $\frac{\partial \ell}{\partial \mathbf{z}}$, $\frac{\partial \ell}{\partial \mathbf{W}}$ and $\frac{\partial \ell}{\partial \mathbf{b}}$. You should use the backpropagation equations to obtain these derivatives. You should express these derivatives for a single training point $(\mathbf{x}, y)$. When optimising, you'll usually average the gradient over a mini-batch before actually updating the parameters using a gradient step.

2 Digit Classification using MLPs

In this problem, we will consider a neural network to classify handwritten digits. You will also use this dataset in your practical this week. In the practical, you will implement a convolutional neural network. Here, we consider networks with fully connected layers and a different way to encode the targets (and the outputs of the network). You are strongly encouraged to implement these networks in tensorflow as well, to help you answer this question.

The inputs are vectors of length 784 (obtained from 28×28 grey pixel images). Rather than output a class, the network will output a vector $\widehat{\mathbf{y}} \in \mathbb{R}^{10}$. The target will be represented as a one-hot encoding, *e.g.,* if the label is 3, we will use the vector $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]^\mathsf{T}$ (by convention the digit 0 will be the last component in the encoding, not the first).

The data fed into the neural network consists of $\langle(\mathbf{x}_i, \mathbf{y}_i)\rangle$, where $\mathbf{x}_i \in \mathbb{R}^{784}$ and $\mathbf{y}_i \in \{0, 1\}^{10}$ is the one-hot encoding of the digit. The output of the neural network is also a vector $\widehat{\mathbf{y}} \in \mathbb{R}^{10}$. The parameters in the network are the weights $\mathbf{W}^i$ and biases $\mathbf{b}^i$ for layers in the network. We'll train the networks by minimizing the squared loss objective (with respect to the parameters of the neural network):

$$\mathcal{L}(\mathbf{W}^i, \mathbf{b}^i) = \sum_{i=1}^{N} (\widehat{\mathbf{y}}_i - \mathbf{y}_i)^2$$

The neural network to be used is shown in Figure 1. There are three layers, one input layer, one *hidden* layer and one output layer. The gradient of the loss function with respect to the parameters can be computed using the backpropagation algorithm as we've seen in the lectures. Answer the following questions:

1. To reduce the size of the network and increase efficiency, your colleague suggests using a binary encoding for the outputs instead of one-hot encoding; so 0 is encoded as 0000, 1 as 0001, and so on. In this case, the output layer only has four neurons rather than 10. Perhaps you could also reduce the number of neurons in the middle layer. What do you think about this suggestion?

2. Show that if you have an already trained neural network that has high accuracy with the *one-hot* encoding, you can design a network that uses binary encoding and achieves roughly the same error. *(Hint: You may need to add an additional layer.)*

3. What do you think would happen if you tried to train the neural network you suggested in the previous part directly (rather than adding the last layer by design)?

3 Neural Nets with Linear Activation Functions

Consider a neural network for multiclass classification. Let us assume that the input is 784 dimensional, we'll us one hidden layer, and there are 10 classes, *e.g.,* digits. For the hidden unit we'll use the "identity" activation function, *i.e.,* $f(x) = x$. Answer the following questions:

1. Suppose you use 25 hidden units and no regularization. For the output layer, we'll use a softmax non-linearity as we did for multiclass logistic regression in Problem 1. An

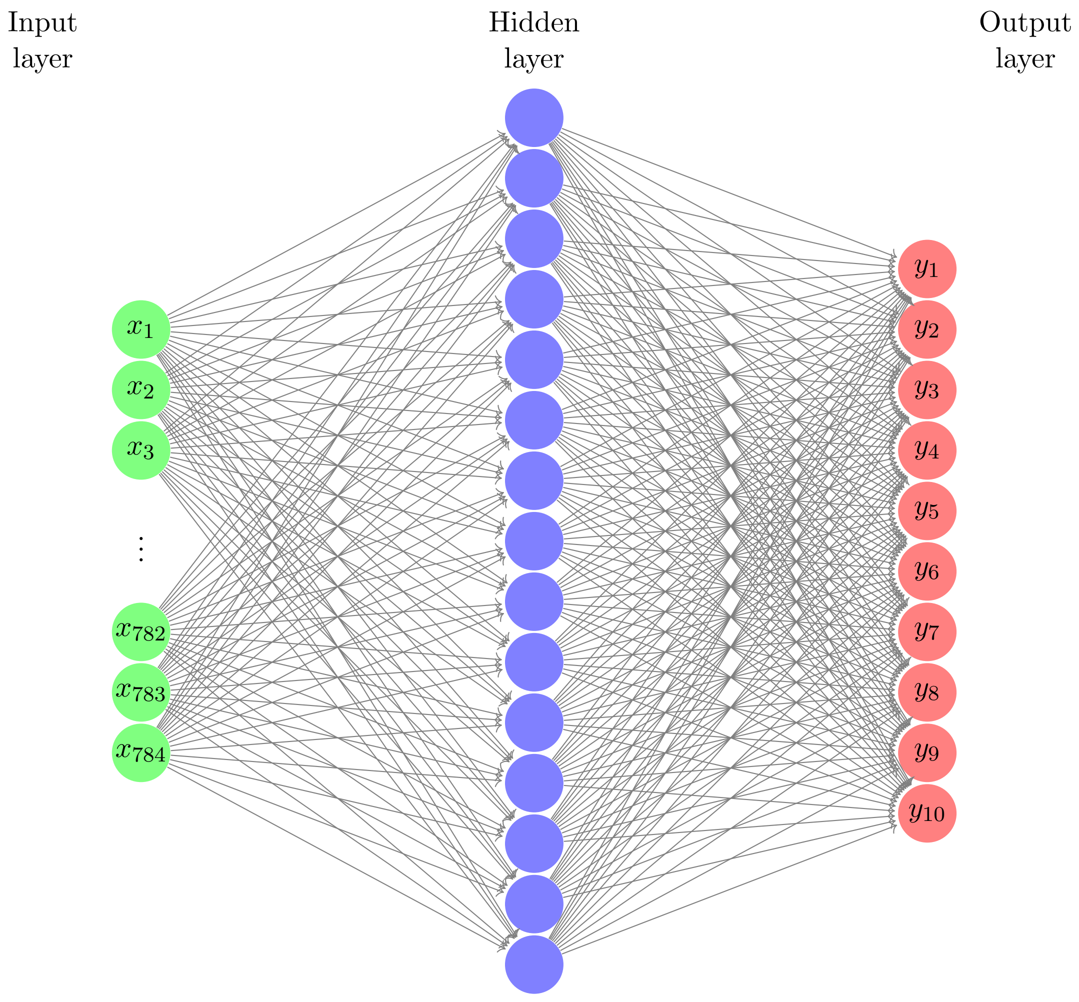


Figure 1: Three-layer neural network for handwritten digit classification.

alternative to such a neural network would be to just use the multiclass logistic regression approach directly. Which of the two models is more powerful, *i.e.*, able to represent possibly more complex relationships between inputs and outputs? (For this part, ignore the difficulty of training the models.)

2. Now suppose you only use 4 hidden units. What can you say about the relative power of the two models? Which model would you prefer?

3. For the network in Part 2, if you use the cross entropy loss function on the output layer, do you get a convex optimisation problem?

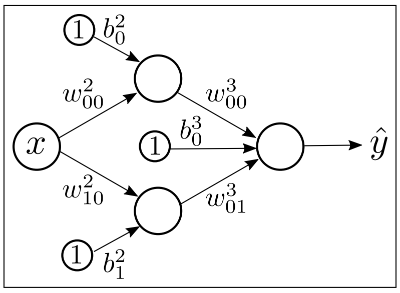
4 Neural Networks as Universal Function Approximators

As remarked in the lectures, neural networks are universal function approximators. In this question, we will discuss some steps for showing this result and what it means to be a universal function approximator a bit further.

Subsequently, in all questions we will exclusively consider neural networks with

- a single input,
- a single hidden layer with an arbitrary number of neurons with ReLU activation functions, and
- an output layer consisting of a single neuron with the identity activation function.

Consequently, the network implements a function $h\colon \mathbb{R} \to \mathbb{R}$. Whenever you are asked below to provide a neural network, your network is required to have those properties. The following picture shows an example of such a network with two neurons in the hidden layer:



1. Write down the model output $\widehat{y}$ as a function of x by application of the forward equations.
2. Set $w_{00}^2 = w_{10}^2 = w_{00}^3 = 1$, $w_{01}^3 = -1$, $b_0^2 = -3$, $b_1^2 = -4$, and $b_0^3 = 0$. Compute $\widehat{y}$ for the inputs $x_1 = 3$ and $x_2 = 4$. State and sketch the function $h\colon \mathbb{R} \to \mathbb{R}$ implemented by the network.
3. Given constants $c < d$, provide a neural network which implements a function $h_{c,d}\colon \mathbb{R} \to \mathbb{R}$ such that for all $x \in \mathbb{R}$

$$h_{c,d}(x) = \begin{cases} 0 & \text{if } x < c \\ \frac{1}{d-c} \cdot x - \frac{c}{d-c} & \text{if } c \le x \le d \\ 1 & \text{if } x > d. \end{cases}$$

4. Let $f\colon \mathbb{R} \to \mathbb{R}$ be any function such that on the interval $[0, 1]$, f is continuous and bounded, i.e., there is some constant $c \ge 0$ such that $|f(x)| \le c$ for all $x \in [0, 1]$. Give a high-level yet mathematically well-founded argument why the class of neural networks considered in this question can approximate f arbitrarily close. More formally, given f and $\varepsilon > 0$, we can provide a neural network implementing some $h\colon \mathbb{R} \to \mathbb{R}$ such that $|f(x) - h(x)| < \varepsilon$ for all $x \in [0, 1]$.

5. Argue that for any choice of weights and biases of the network in the figure above, there exists some x_0 after which h behaves like a linear function, i.e., there is some $f(x) = m{\cdot}x{+}n$ such that for all $x \geq x_0$, $h(x) = f(x)$. Is the same true if there are arbitrarily many neurons in the hidden layer?

6. You have become head of a venture capital firm investing in promising future technologies. A new start-up called BrainDrain is applying for funding at your firm. They claim that using a blockchain-based in-the-cloud approach, they have trained a neural network that, on an input x, outputs a value greater than 0 if x is a prime number and 0 otherwise. Will you invest in this start-up?

5 Recurrent Neural Networks

1. The Truncated Backpropagation Through Time (TBPTT) algorithm for training Recurrent Neural Networks (RRNs) truncates the gradient calculation to a maximum of k steps. Compare a feed forward neural n-gram model trained with SGD and an RNN language model trained with TBPTT truncated to n steps. Does the truncation make the RNN equivalent to the n-gram model or can the RNN learn dependencies longer than n, either in theory or in practice? Discuss.

2. The BPTT algorithm computes the derivates of the unrolled sequence in reverse order, iteratively computing the following recurrence,

$$\nabla_\theta \mathcal{F} = \sum_{n=1}^{N} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_n} \frac{\partial \mathbf{h}_n}{\partial \theta_n} = \sum_{n=1}^{N} \left[\frac{\partial \mathcal{F}}{\partial \mathbf{h}_{n+1}} \frac{\partial \mathbf{h}_{n+1}}{\partial \mathbf{h}_n} + \frac{\partial \mathcal{F}_n}{\partial \mathbf{h}_n} \right] \frac{\partial \mathbf{h}_n}{\partial \theta_n}.$$

   Where $\mathcal{F} = \sum_n \mathcal{F}_n$ and the θ_n notation refers to the copy of the RNN parameters at timestep n in the unrolled computation graph. The complete gradient for the parameters θ is the sum of gradients for each copy.

   An alternative to backpropagation is to compute the partial derivates using an in order recurrence,

$$\nabla_\theta \mathcal{F} = \sum_{n=1}^{N} \frac{\partial \mathcal{F}_n}{\partial \mathbf{h}_n} \frac{\partial \mathbf{h}_n}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial \mathcal{F}_n}{\partial \mathbf{h}_n} [\ldots] .$$

   Complete this equation by filling in the missing partial derivatives in the brackets $[\ldots]$. Using the completed equation, propose an alternative SGD training algorithm to BPTT. What is the computational and memory complexity of this algorithm, and what are its advantages and disadvantages versus BPTT?