**Msc Neuroscience Computational Neuroscience Module Practical 4: Deep Networks**
Instructor: Andrew Saxe

In this practical you will write your own backpropagation code to train a deep network. Starter code and data for the assignment is provided on Canvas. The second question is purely for interest and should not be submitted. Submit all code you write, and the plots you produce to generate your answers.

1. **Backpropagation and deep learning.** Here you will write your own deep network training code and run it on a small scale problem: handwritten digit recognition. Starter code for this problem is available in `backprop.m`. The goal is to train a network to differentiate between handwritten 7's and 9's drawn from the MNIST dataset, a widely used proving ground for new learning algorithms before they are scaled up. This data is supplied in `mnist_sevens_nines.mat`, which will define the variables `X_train`, `y_train`, `X_test`, and `y_test` when loaded. There are 1800 training examples and 200 test examples. Sevens have been assigned target values of $-1$, while nines have been assigned target values of 1.

   We will implement a fully connected two hidden-layer network with ReLU nonlinearities, linear output neurons, and the mean squared error loss function. This network will have three matrices of tunable synapses $W_1 \in R^{N_{h_1} \times 784}, W_2 \in R^{N_{h_2} \times N_{h_1}}, W_3 \in R^{1 \times N_{h_2}}$, where $N_{h_1}$ and $N_{h_2}$ are the number of hidden units in the first and second hidden layers respectively, and there is a single output neuron. These weights will be initialized as unit Gaussian random variables scaled by a `weight_scale` factor.

   Let's briefly review the mathematics of the backpropagation algorithm in this context. Given a dataset of $P$ examples $\{x^\mu, y^\mu\}, \mu = 1, \cdots, P$, the loss function we wish to minimize is is

   $$\mathcal{L}(W_3, W_2, W_1) = \frac{1}{2} \sum_{\mu=1}^{P} \|y^\mu - W_3 f(W_2 f(W_1 x^\mu)\|_2^2,$$

   where $f$ is the ReLU nonlinearity applied elementwise.

   We wish to compute the gradient of this loss function with respect to each weight matrix. To do so, first, we compute a forward pass through the network to obtain its current prediction on an input. Given an input (column) vector $x \in R^{784}$, the first hidden layer activity is computed as $h_1 = f(W_1 x)$ where $f$ is the ReLU nonlinearity is applied elementwise, and simply clips negative values to zero (i.e. $f(u) = max(u, 0)$). The second hidden layer activity is $h_2 = f(W_2 h_1)$. The output is $\hat{y} = W_3 h_2$ (which here is just a scalar). Now, we begin the backward pass through the network. We compare the network's output to the target value $y$ to compute the error, $e = y - \hat{y}$, and this is the starting value of delta for backpropagation, $\delta_3 = e$. The backpropagation equations are

   $$\delta_{l-1} = \left[W_l^T \delta_l\right] \circ f'(u_{l-1}),$$

   where $\circ$ denotes elementwise multiplication (`.*` in matlab). Here $f'(u_{l-1})$ is the derivative of the activation function evaluated at the activity level evoked by the example. In our case, the derivative of the ReLU function is simply 1 if the neuron's activity is greater than zero, and zero otherwise (we ignore the discontinuity at zero). Hence $f'(u_{l-1}) = \text{step}(h_{l-1})$ where step$(\cdot)$ is zero if its input is less than zero and 1 otherwise, and step is applied elementwise to a vector input. Finally, the gradient for the weights of layer $l$ is

   $$\frac{\partial \mathcal{L}}{\partial W_l} = -\delta_l h_{l-1}^T,$$

   where we define $h_0$ to be $x$.

   This procedure describes how to compute the gradient on one specific example $\{x, y\}$. To compute the full gradient, we just sum the gradients across examples. In fact, this summation can be handled as a matrix multiplication: all of the above steps work if $x$ is a $784 \times P$ matrix and $y$ is a $1 \times P$ row vector where $P$ is the number of examples. Implementing the above steps will now yield matrices rather than vectors, and the final operation $\delta_{l+1} h_l^T$ will implement the summation across examples (you can also just do a simple for loop but it will be slower).

Finally, we update the weights by taking a small step in the negative gradient direction,

$$W \leftarrow W + \alpha \delta_l h_{l-1}^T,$$

where $\alpha$ is a learning rate.

(a) Implement gradient descent. Run 5000 iterations of gradient descent with a learning rate of $\alpha = 0.1/P$ (where $P$ is the number of examples), and a `weight_scale` of 0.01. Plot the loss on the training examples and the loss on the testing examples over the course of learning, and the training/testing accuracy.

(b) How accurate a classifier does this yield? Is there substantial overfitting? Would stopping training early help the test accuracy?

(c) Now try a larger `weight_scale` of 0.1. Does this impact the final training accuracy? Does this impact the final testing accuracy? Would stopping training early help the test accuracy? Does a smaller or larger initial weight variance seem better for the purposes of generalization based on your findings?

2. ***Optional, not for credit:*** **Learning Dynamics in Deep Linear Neural Networks**

Consider a simple neural network consisting of a serial chain of $D$ neurons (that is, each layer contains just a single neuron). Given scalar input $x^\mu$, the network produces a scalar output

$$\hat{y}^\mu = w_{D-1} w_{D-2} \cdots w_2 w_1 x^\mu$$

where $w_i, i = 1, \cdots, D-1$ are the (scalar) weights between neurons. Here a depth $D = 2$ network corresponds to a 'shallow' network with no hidden layers; and a depth $D > 2$ network corresponds to a 'deep' network with one or more hidden layers.

The network is trained on a dataset of $P$ examples $\{x^\mu, y^\mu\}, \mu = 1, \cdots, P$ by minimizing the mean squared error between the network's output and the target outputs,

$$E(w_1, \cdots, w_{D-1}) = \frac{1}{P} \sum_{i=1}^{P} \left(y^\mu - \hat{y}^\mu\right)^2, \tag{1}$$

yielding the optimization problem

$$\min_{w_1, w_2, \cdots, w_{D-1}} E\left(w_1, \cdots, w_{D-1}\right). \tag{2}$$

To perform this minimization we use continuous time batch gradient descent, such that the dynamics for the weights are

$$\tau \frac{d}{dt} w_i = -\frac{\partial E(w_1, \cdots, w_{D-1})}{\partial w_i}, \quad i = 1, \cdots, D-1. \tag{3}$$

a) Show that a linear network of any depth can be expressed as a shallow network with appropriate weight $w_s$. Hence for linear networks, deeper networks do not have greater expressive power than shallow networks. (This is a one-liner, but it is conceptually important.)

b) Show that the error in Eqn. (1) depends only on the second order statistics of the training data, namely the output correlations $\sigma^y = \langle (y^\mu)^2 \rangle$, input-output correlations $\sigma^{yx} = \langle y^\mu x^\mu \rangle$, and input correlations $\sigma^x = \langle (x^\mu)^2 \rangle$. Express the error as a function of these correlations and the overall total weight $w_{tot} = w_{D-1} w_{D-2} \cdots w_2 w_1$.

c) Compute the gradient descent equations (3).

d) Find the fixed points of the dynamics. In particular, show that a shallow network ($D = 2$) has exactly one fixed point so long as $\sigma^x > 0$, whereas a deep network ($D > 2$) has two sets of fixed points: one set corresponding to the same minimum as the shallow network, and another set which is unique to the deep networks. What error is attained by fixed points of each type?

e) We will now construct a phase portrait for the case $D = 3$, i.e., when the network has one hidden layer and two weights $w_1, w_2$. Consider a dataset with $\sigma^y = 4.5$, $\sigma^{yx} = 2$, and $\sigma^x = 1$. Plot the error $E$ as a function of $w_1$ and $w_2$ (let $w_1$ and $w_2$ vary between $[-2.5, 2.5]$). Plot both types of fixed points (one type will yield a curve, the other type yields just a single point in this case). By inspection of the error surface, what kind of fixed points are these (e.g., local/global minima/maxima, saddle points)?

f) Plot three sample trajectories of gradient descent by numerically integrating the dynamics. Start one trajectory with the initial condition $w_1(0) = w_2(0) = 0.001$, the second trajectory with $w_1(0) = 2.1, w_2(0) = -2$, and the third trajectory with $w_1(0) = 2, w_2(0) = -2$. Take $\tau = 1$. You may use any method you like but simple Euler integration will work. This is a discretization of the continuous time gradient dynamics yielding the update

$$w_i[t+1] \leftarrow w_i[t] - \frac{1}{\tau}\frac{\partial E(w_1, \cdots, w_{D-1})}{\partial w_i}\Delta t, \quad i = 1, \cdots, D-1,$$

where $\Delta t$ is the discretization step size (try $\Delta t = 0.01$ and perform 1000 steps). Plot these trajectories over time. Additionally, plot them in phase space (that is, $w_1(t)$ against $w_2(t)$) on your phase portrait from part (e). Does training ever fail?

g) Show that if all weights start out to be equal ($w_i(0) = w_0$, $i = 1, \cdots, D-1$), then they remain so under the gradient descent dynamics. In this 'balanced' regime we therefore can track the dynamics of just one of the weights over time. Denote this weight as $w$ (i.e., $w(t) = w_i(t)$ for all $i$ and $t$). What is the error $E$ as a function of $w$? What is the gradient descent equation for $w$? What are the fixed points? Plot the error $E$ as a function of $w$ for $w \in [0, 2.5]$ (note that the range excludes negative values of $w$ since these are harder to interpret because, for odd depths, they flip the sign of the output), and for depths $D = 2, \cdots, 6$ (using the same dataset statistics as part (e)). Set the upper y-axis limit to make the behavior around the origin easy to see. Suppose we initialize networks with $w(0) = 1/2, w(0) = 1$, or $w(0) = 1.1$. Plot the gradient at the start of training as a function of depth (let $D = 2, \cdots, 20$) for these three initial conditions. Qualitatively, how do you expect each of these initializations to fare as the depth of the network increases?