CS37300 Homework	— Homework 6—	Last update: November 20, 2017
https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017		Due: Dec 2, 2017 (06:00am)

Instructions and Policy: Each student should write up their own solutions independently. You need to indicate the names of the people you discussed a problem with; ideally you should discuss with no more than two other people.

- YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK
- The answers (without the python scripts) MUST be in submitted in a single PDF file via Blackboard.
- The python scripts will be submitted separately via turnin at data.cs.purdue.edu.
- Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.
- Theoretical questions MUST include the intermediate steps to the final answer.
- Zero points in any question where the python code answer doesn't match the answer in the PDF.
- Python code answers MUST adhere to the format described below.

There are TWO (2) questions in this homework.

Python code guidelines

Turn in each question of your homework in a separate python file named `hw6-X.py`, where X is the question number.

Your code is REQUIRED to run on either Python 2 or Python 3 at scholar.rcac.purdue.edu. Preferably use Python 3 (Python 2 will also be accepted). The TA's will help you with the use of the scholar cluster. If the name of the executable is incorrect, it won't be graded. Please make sure you didn't use any library/source explicitly forbidden to use. If such library/source code is used, you will get 0 pt for the coding part of the assignment. If your code doesn't run on scholar.rcac.purdue.edu, then even if it compiles in another computer, your code will still be considered not-running and the respective part of the assignment will receive 0 pt.

Q1 (50 pts): [Deep Learning/Activations]

In class we have seen the training of a one hidden layer feedforward neural network using Stochastic Gradient Descent (SGD)
`https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017/lectures/MiniBatch_GD_FeedForward_ReLU.html`
We will now dive deeper into issues with training ReLU activated neurons (ReLU neurons).

Inner product layer (fully connected layer) As the name suggests, every output neuron of inner product layer is fully connected to the input neurons. The output is a vector resulting from the multiplication of the input with a weight matrix W plus a bias offset, i.e.:

$$\mathbf{z}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{1}$$

where the input $\mathbf{x}$ is d dimensional column vector, and $\mathbf{W}$ is a $d \times n$ matrix and $\mathbf{b}$ is n dimensional column vector. This is just a linear transformation of the input. Often, we will concatenate the bias $\mathbf{b}$ into W by concatenating the constant 1 to $\mathbf{x}$. The weight parameter $\mathbf{W}$ and bias parameter $\mathbf{b}$ are learnable in this layer.

Activation. We add nonlinear activation functions after the inner product layers to model non-linearity. Here are some of the popular choices for non-linear activation:

- **Sigmoid**: $\sigma(z) = \frac{1}{(1+e^{-z})}$
- **tanh**: $\tanh(z) = \frac{(e^{2z}-1)}{(e^{2z}+1)}$
- **ReLU**: $\text{relu}(z) = \max(0, z)$
- **ELU**: $\text{elu}(z) = \begin{cases} z & \text{if } z \geq 0, \\ \alpha(\exp(z) - 1) & \text{if } z < 0, \end{cases}$ where α is a small constant.

If the input is a vector $\mathbf{z}$*, then each function is applied to each element of the vector.* Rectified Linear Units (ReLU) and Exponential Linear Units (ELUs) have been found to work well in vision related problems. In this homework, you will understand when ReLU works, its problems, and why ELU[1] is a better alternative.

Consider (minibatch) stochastic gradient descent (SGD) as the learning algorithm used to train the neural network.

(a) (10pts) Calculate the derivatives of the ReLU and ELU units with respect to variable z.

(b) (10pts) Let $\mathbf{h} = \text{relu}(\mathbf{z})$, where $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Compute $\partial\mathbf{h}(r)/\partial\mathbf{W}(r, j)$, where $\mathbf{h}(r)$ is the r-th output of the ReLU and $\mathbf{W}(r, j)$ is the element r, j of the weight matrix (you will need to use the chain rule). This may help: `https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017/lectures/MiniBatch_GD_FeedForward_ReLU.html.`

(c) (20pts) In the iPython notebook used in the lecture of Oct 30th
`https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017/lectures/MiniBatch_GD_FeedForward_ReLU.html`
"Define backpropagation", we have seen how the gradient of the weight matrix W is computed by averaging the gradient computed for each training example in the minibatch. Clearly, however, if the gradient of a parameter is zero, this parameter will not change in the next gradient step. Let the training dataset $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the minibatches be of size N/K, where K is a constant (assuming N is a multiple of K). Define the conditions under which the gradient of a ReLU unit $\mathbf{h}(r)$ is zero for all minibatches. When this happens, we say that the neuron has died.

[1] Clevert, Djork-Arn, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)." ICLR 2016.

Hints: (1) Pay attention to the fact that the gradient of a parameter is the average of the gradient over all training examples of the minibatch. (2) Also note that a minibatch is a sampled from the training data. (3) The values of K and N are not relevant to your answer.

(d) (10pts) Explain using the answers in Q1(a) and Q1(b), why ELU neurons cannot die like ReLU neurons.

Q2 (50 pts): [Deep Learning / Generative Models] Run the Generative Adversarial Network (GAN) code at
`https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017/lectures/GAN_Generator_Moons.html`

(a) **(10 pts)** (1) Describe what `num_epochs` does in the code. (2) Run the code multiple times, varying `num_epochs` as 10, 30, 50, 100, 200. Note that you will need to reset the GAN every time you change `num_epochs`. The easiest way is to re-run the entire python script for every new value of `num_epochs`. For each `num_epochs` value, report the plot in line 18 in your PDF (make them small in your PDF and organize them sorted in ascending order by `num_epochs`). (3) Describe what the GAN is learning to generate. (4) Does the GAN think it has been successful at 200 epochs?

(b) **(10 pts)** (1) Describe what `d_steps` does in the code. (2) Change `d_steps = 10` in the code. Report the plot in line 18 for `num_epochs = 200` in your PDF. (3) Does GAN think it has been successful? (4) What is wrong with this training procedure, why has it failed?
(5) Describe what `g_steps` does in the code. (6) Change `g_steps = 10` in the code. Report the plot in line 18 for `num_epochs = 200` in your PDF. (7) Does GAN think it has been successful? (8) What is wrong with this training procedure, why has it failed?
(9) Change `d_steps = 10` and `g_steps = 10`. Report the plot in line 18 for `num_epochs = 200` in your PDF. Do you think the GAN is learning to generate the data?
Hint: Remember that we are training the classifier and the generator together, and they influence each other.

(c) **(20 pts)** GAN is a type of Noise Contrastive Estimation (NCE) procedure that seeks to learn better noise (proposal) distributions than just "standard" random noise (Gaussian, Uniform). Here, however, we will use the standard NCE procedure (Lecture 19, slide 10). Write a pseudo-code (in the PDF) that uses the standard NCE with 2-dimensional Gaussian noise to generate some 2-dimensional data, like the data in the above examples.
Hint: A GAN will deform the Gaussian noise to increase the probability the classifier will mistake the true data from the noise. Here, we are not deforming the Gaussian input noise. First, you need to train the classifier. Then, you can use the classifier in the input noise to decide which noise examples "look like the data" and output them as the "generated data".

(d) **(Extra credit +10 pts)** Write Q2(c) in python and show (using plots in the PDF) that this procedure generates better data (without mode collapsing) than the original GAN procedure. Upload your python code with turnin. Your code should ONLY output 100 generate examples as "x1,x2" , where x1 and x2 are the coordinates of the point.
Also argue why GANs are used in practical problems like image super-scaling while standard NCE is not. What is the drawback of standard NCE for high-dimensional data?