Just like before, your homework submission must contain the following to receive full credit:

- A single script called "hmwk3.m" or "hmwk3.py" that requires no arguments and prints the things I ask for in bold below. Each output should be labeled.
- A pdf called hmwk3_results.pdf with your solutions to all theory problems and also the text output from the hmwk1 script. See the posted homework example.
- As many other code files as you wish - but only things you wrote or solutions to previous homeworks.

1. (a) Write a function with signature

```
function iscorrect = check_gradient(f, grad, x0)
```

Arguments:

- `f` : A function $\mathbb{R}^\Omega \to \mathbb{R}$ that maps an array or arbitrary dimensions into the reals.
- `grad` : A function $\mathbb{R}^\Omega \to \mathbb{R}^\Omega$ that maps an array or arbitrary dimensions into an array of the same dimensions.
- `x0` : A vector/array in $\mathbb{R}^\Omega$.

Returns: A boolean that is true if `grad` generates the gradient of `f`.

Details: Your code should test whether `grad` generates the gradient of `f`. It should do this by generating a random perturbation δ and then testing whether

$$f(x+\delta) - f(x) \approx \langle \delta, \nabla f(x) \rangle.$$

The method should generate a *random Gaussian* δ with the same norm as `x0`, check the gradient condition, and then replace $\delta \leftarrow \delta/10$. Do this for 10 different orders of magnitude of δ. For each order, print out the ratio $\|\delta\|/\|x0\|$, and print out the *relative* error between the left and right side of the above equation. Finally, the last line of output should print the minimum relative error achieved. All of the outputs should be labeled. The method returns `true` if the gradient is correct up to reasonable error, and `false` otherwise.

Note: you can use one `for` loop (but only one) in your solution.

(b) **In your hmwk1 script, test your gradient checker on the function**

$$f(x) = \frac{1}{2}\|Ax - b\|^2$$

for random $A \in \mathbb{R}^{4\times 3}$ and $b \in \mathbb{R}^{4\times 5}$, and capture the output in the console.

2. (a) Create a function with signature

```
function y = logreg_grad(x,D,c)
```

that produces the gradient of the function `logreg_objective` from your last homework assignment. No `for` loops! Note: you can solve this with a few lines of code. Don't exponentiate positive numbers!

(b) **Test your gradient by calling `create_classification_problem(1000,10,5)` to produce test data and labels. Then feed the resulting function and gradient into your function `check_gradient`. Capture the console output.**

3. All of your solutions to these problems should only require a few lines of code. You may not use any `for` loops!

   (a) Write a function with signature
   ```
   function f = l1_eps(x,eps)
   ```
   that returns the smoothed ℓ_1 norm
   $$|x|_\epsilon = \sum_i \sqrt{x_i^2 + \epsilon^2}.$$
   Your function must handle arguments `x` of arbitrary dimensions (i.e. 1d, 2d, 3d, etc...).

   (b) Write a function with signature
   ```
   function f = l1_grad(x,eps)
   ```
   that returns the gradient of the smoothed ℓ_1 norm you created in part (a).

   (c) Write a function with signature
   ```
   function f = tv_objective(x, b, mu, eps)
   ```
   Arguments:

   - `x` : A 2d array.
   - `b` : A 2d array containing a noisy image.
   - `mu` : A scalar, non-negative scaling parameter.

   Returns: The value of
   $$f(x) = \mu|\nabla_d x|_\epsilon + \frac{1}{2}\|x - b\|^2$$
   where $|\cdot|_\epsilon$ is the smoothed ℓ_1 norm, and ∇ denotes the discrete gradient. This function should only be a few lines of code (possibly just 1), and you should use the function `grad2d` from your last homework assignment.

   (d) Write a function with signature
   ```
   function f = tv_grad(x, b, mu, eps)
   ```
   that produces the gradient $\nabla f(x)$. You should use your `grad2d` and `div2d` functions from the last assignment. Remember - this should only be a few lines of code or you did something wrong.

   (e) Build a noisy test image `b` (call `phantom(64)` in Matlab, or create a black image with a white square in the center), a random 2d input `x`, and set `mu=0.5`. Build a function handle that accepts `x`, and returns the objective $f(x)$. Build another function handle that accepts `x`, and returns the gradient. **Using these handles, test your gradient by calling `check_gradient`, and capture results in the console.**

   If your method doesn't pass the gradient checker, then make sure you didn't make any of the following common errors:

   - You remembered to square the norm in the objective, right?
   - You're using the Frobenius norm, not the 2 norm, right?
   - You remembered to multiply by μ in both objective and gradient functions, right?
   - You remembered the factor of $\frac{1}{2}$ in the objective, right?

   The most common reason to fail the gradient test is because your implementations for ∇ and ∇^T are not correct adjoints. You should have verified your adjoint using your adjoint checker in the last assignment.

4. (a) Create a function to implement the gradient of the neural net objective function you created in your last homework assignment. Your function should have prototype

```
function dW =  net_grad(W,D,L)
```

and the following specifications:

- The function should return a cell-array or list of matrices containing the gradients with respect to the weights. The dimensions of these gradients should match those of the weight matrices handed in as argument `W`.
- You can never evaluate the exponential of a positive number.
- Your function may contain only 2 `for` loops (to loop over layers of the net during the forward and backward pass). You may NOT loop over feature vectors.

(b) Load the mnist dataset, and create a random neural network for this problem with 2 hidden layers, the first with 20 neurons and the second with 15. Build a function handle that accepts a vector of 16130 weights, and returns the loss value of a network with these weights. Build a function handle that accepts a list of 16130 weights, and returns a gradient with 16130 entries. These functions should use the entire MNIST training dataset to evaluate the objective/gradient.

Verify your gradient by calling `check_gradient` using these two function handles, and capture the output in the console.

Hint: I have provided utility functions for converting between a vector and a cell-array/list of weight matrices. Download `cell_to_vec` and `vec_to_cell` in Matlab, or `utility.py` for python. These functions will accept a list of matrices, and dump the contents into one long vector, and take a long vector and reshape it into a list of matrices. With these helper functions, this problem should also require a few lines of code.