[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources. ]

**Due Tuesday 6pm**

## *Coding 1: Fooling of visual classifiers*

- implement the algorithmically improved fooling from the lecture, except the part about the fix for rounding/discretization problems (but with the part ensuring that all image subpixels after multiplying back the standard deviation and adding back the mean are all within the proper bounds for an image)
- submit code and and the modified image of Senor Shout that passes as a strawberry when loaded from disk. Test the prediction when loading the image from disk - include that test routine in the code to be submitted. Dont forget to mention what network you did you use.
- compute the difference between the original and the modified image, rescale it so that it fills up the range maximally (e.g. multiply all rgb subpixels so that the largest subpixel value reaches either 1 or 255), print this as an image too.

More Homework questions:

- Why with this modified gradient the objective function cannot decrease ? It will either increase or at least stay constant.
- Give an example in at most 3 sentences when with this modified gradient the objective function would not increase, that is the algorithm cannot continue in fooling?

## *Coding 2: RNN for classification with LSTM*

Dont worry, next week you will have a beefier task on RNNs.

Two things I want you to learn here:

- how to use an LSTM in its native interfaces
- adapting RNN-code to batchsizes $> 1$

What to do?

- Take the tutorial code from: `https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html` as a starting point

- split the dataset into train and test (no need for validation set here, 70/30)
- Task1: replace there the custom RNN by an LSTM. The hidden state $h_n$ at the last step will be fed into a fully connected layer. See the documentation at `https://pytorch.org/docs/master/nn.html#lstm`
  That is more work than it looks at the first glance. Suggested steps:

  1. it can be useful for clarity to at first take the code, and put all dangling commands and variables that are globally defined in the file into functions for the sake of encapsulation, and just call one function from

  ```
  if __name__ == '__main__':
    run()
  ```

  This may increase readability of the code a lot.
  2. Data access. You can (but not must) write your own python iterator. Below is an example that generates batchsize 1 samples of feature-label-pairs. If you want to construct a batch of 2, then you can overwrite the `return` statement, or call the iterator twice. `def __next__(self):` defines what happens if you call once

  ```
    for element in youriterator:
      #whatever
      pass
  ```

  Here is an example iterator implementation. No perfection guaranteed.

  ```
  class iteratefromdict():
      def __init__(self, adict,all_categories):
        #whatever here

      def num(self):
        return len(self.namlab)

      def __iter__(self):
          return self

      def __next__(self):

          if self.ct==len(self.namlab):
              # reset before raising iteration for reusal
              np.random.shuffle(self.namlab)
  ```

```
            self.ct=0
            #raise
            raise StopIteration()
        else:
            self.ct+=1
            #return feature-label pair here
            return self.namlab[self.ct-1][0],self.namlab[self.ct-1][1]
```

Such an iterator class allows you to iterate in training and in testing loops:

```
for counter,(feature,label) in enumerate(youriterator):
  #whatever
  pass
```

It functions like a dataloader (not dataset) which provides you batchsize-1-pairs of features and labels. This stuff should train reasonably fast on a notebook, AWS should be not necessary unless your notebook is very old or a toy.
**Report performance on the test set in terms of accuracy for 1 and 2 layer LSTMs with at least 3 different sizes of the hidden layer, and submit code of course.**

- Task2 - part A: do it after task1 in case that you cannot finish it, not together. Take your result from task1 and adapt the code to run with a batchsize of at least 2. For this small network it does not matter but for your deep learning tasks after SUTD and for larger tasks in week 6/8 it is necessary.
  The problem about batchsizes here is: sequences have varying length! You cannot just clutch them together. You can use `torch.nn.pack_padded_sequence` or `torch.nn.utils.rnn.pack_sequence` (which I have used in my implementation).

  If you are going to use *output* from the LSTM, and not $(hn, cn)$, then you will also need to use `pad_packed_sequence` on it, to make the output into some readable tensor again.

- Task2 - part B: compare training and test error and test accuracy as a function of epoch for one choice of LSTM (e.g. 1 layer, 200 hidden dimensions) for batchsize 1,10,30. Plot it.