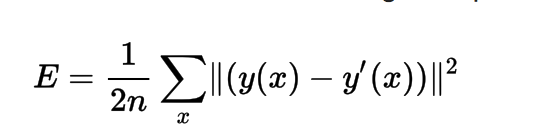
# Assignment answer:

**Sunglass recogniser:**

* What did I change:
  + The if statement in imagenet.c
  + Instead of comparing userid to “Glickman”, it is now comparing eyes to “sunglasses”.
* Training set:
  + Maximum accuracy achieved was 100%.
  + This happened at epoch 30.
* Validation set:
  + Maximum accuracy achieved was 100%.
  + This happened at epoch 20.
* Training set:
  + Maximum accuracy achieved was 94.2308%.
  + This happened at epoch 45.

**Face recogniser:**

* What did I change:
  + I change imagenet.c to add an appropriate target vector to each image.
    - Implemented a for loop which iterates through all of the names
  + Then I changed the evaluate performance () method to iterate through all of the targets and see if it guessed correctly.
  + Created a method output\_result\_on\_imagelist ()
    - This outputs for each image in the list if the function guessed correctly.
  + Error function changed to
  + 
* Training set:
  + Maximum accuracy achieved was 100%.
  + This happened at epoch 54.
* Validation set:
  + Maximum accuracy achieved was 94.4%.
  + This happened at epoch 92.
* Training set:
  + Maximum accuracy achieved was 87.5%.
  + This happened at epoch 69.
* As you can see from the image wrongly\_guessed\_images, all of them are very similar, and that there is little distinguishing them from each other.

**Pose recogniser:**

* I encoded the output the same way, only with focus on poses instead of userid.
* Training set:
  + Maximum accuracy achieved was 99.639%.
  + This happened at epoch 48.
* Validation set:
  + Maximum accuracy achieved was 85.6115%.
  + This happened at epoch 99.
* Training set:
  + Maximum accuracy achieved was 91.3462%.
  + This happened at epoch 51.
* Weight tuning discussed:
  + I did two training sessions before looking at the hidden units weight. One training session on full resolution and one on a quarter of the resolution.
  + You can find the images showing the weights in the folder “weights of hidden units”.
  + The hidden units are definitely weighing particular parts of the image differently. If you compare unit 1 and unit 3 for example, then you can see that they are almost the same, only reversed. (left side weights of unit 1 has the same transformation as right side of unit 3).
  + Comparing different units, we can see that some is focusing on the difference of the pixels compared (edges). While some are focused on the concentration.

# Faces:

* 20 directories
  + One per person – named by user ID
* Naming convention - userid\_pose\_expression\_eyes\_scale.pgm
  + User ID:
    - 20 different values (number of persons)
  + Pose:
    - Head position of the person
    - 4 values – straight, right, left, up
  + Expression
    - 4 values – neutral, happy, sad, angry
  + Eyes
    - 2 values - open or sunglasses
  + Scale
    - How big is the image
      * 1: 128cx120r
      * 2: 64x60
      * 4: 32x30
    - Will be using the quarter resolution (4) for the training to spare time
  + NB: some images have a bad suffix. Meaning they contains glitches.

# Documentation:

* **Files not to modify:** 
  + Pgmimage.c – read and write of pgm image files
    - Data structures:
      * IMAGE and IMAGELIST (array of pointers to IMAGE)
  + Backprop.c – neural network package.
    - Supports three layer fully connected feed forward network.
    - Uses back propagation algorithm for weight tuning
    - Routines for creating, training and using networks.
  + Hidtopgm.c
    - Hidden unit weight visualization utility
    - Interesting to explore some of the possible alternate visualization schemes.
* **Files to modify:** 
  + Imagenet.c
    - Interface routines for loading images into the input units of a network and setting up target vectors for training.
    - Modify load\_target when implementing the face and pose recognizer, so that it contains appropriate target vectors!
  + Facetrain.c
    - The top-level program that uses all of the other modules.
    - Modify to change network sizes and learning parameters
    - Also modify performance\_on\_imagelist () and evaluate\_performance () for the face and pose recognizer.

# Facetrain

* **Running**:
  + ./facetrain –n <network file> -e <number of epochs> -T <test only> -s <seed> -S <number of epochs between saves> -t <training image list> -1<test set 1 list> -2 <test set 2 list>
  + Network file – loads or creates a file to save the network
  + Number of epochs – specifies number of training epochs (default = 100)
  + Test only – performance reported on all three sets.
  + Seed – input used as seed for random number generator (default = 102194)
    - Allows you to reproduce experiments if necessary
  + Number of epochs between saves – default= 100, thus, if you train for 100 epochs, then the document is only saved after training is completed.
  + Training image list
    - If this option is not specified, it is assumed that no training will take place epochs = 0.
    - In this case the statistics for the training set will all be zeros
  + 2 test sets serves for the purpose of having one set to train and test on, and when the performance on the test set begins to degrade you can use the second set for a “real” test.
* **Output:**
* For each epoch the following performance measures are output
  + **Epoch delta trainperf trainerr t perf t err t perf t err**
* **Epoch:** number of epoch just completed
* **Delta:**  sum of all delta values on the hidden and output units as computed during back propagation, over all training examples for that epoch.
* **Trainperf:** Percentage of examples correctly classified in the training set
* **Trainerr**: Average, over all training examples, of the error function
  + 0.5\*sigma (ti – oi)^2
* **T1perf**: percentage of examples in test set 1 correctly classified
* **T2err:** Average, over all training examples, of the error function
* **T2perf: --“”---** set 2.
* **T2err**: ---“”--- set 2.