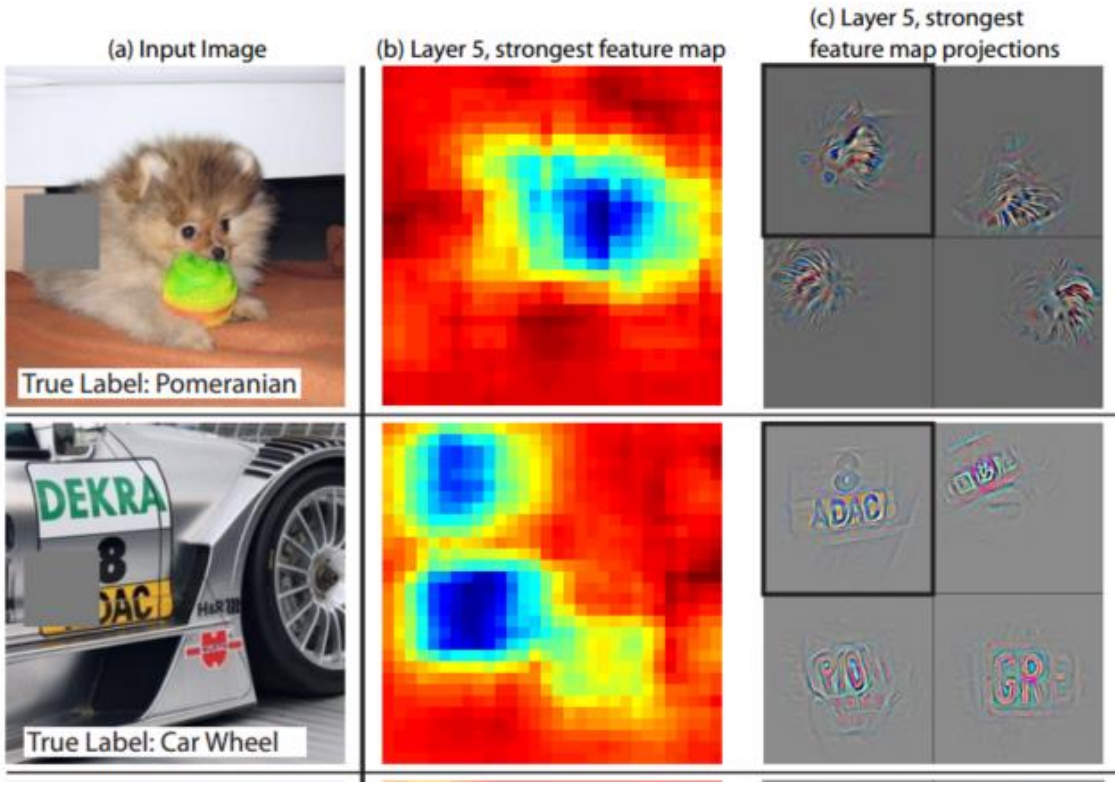
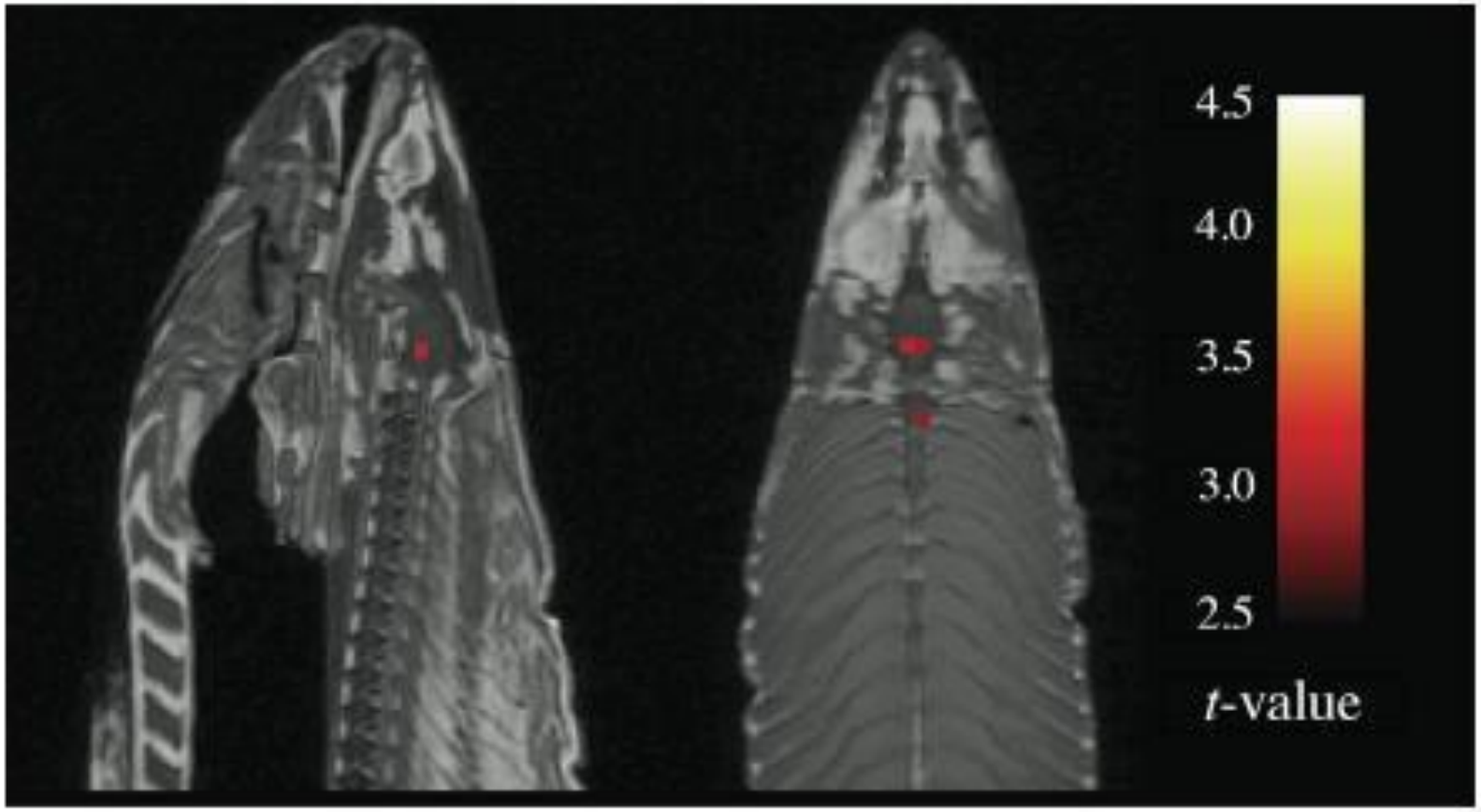
How Neural Networks See (Part 1)



Matthew Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks" (ECCV 2014)

A Diversion: Neural correlates of interspecies perspective taking in the post-mortem Atlantic Salmon



METHODS

Subject. One mature Atlantic Salmon (Salmo salar) participated in the fMRI study. The salmon was approximately 18 inches long, weighed 3.8 lbs, and was not alive at the time of scanning.

Task. The task administered to the salmon involved completing an open-ended mentalizing task. The salmon was shown a series of photographs depicting human individuals in social situations with a specified emotional valence. The salmon was asked to determine what emotion the individual in the photo must have been experiencing.

Design. Stimuli were presented in a block design with each photo presented for 10 seconds followed by 12 seconds of rest. A total of 15 photos were displayed. Total scan time was 5.5 minutes.

Preprocessing. Image processing was completed using SPM2. Preprocessing steps for the functional imaging data included a 6-parameter rigid-body affine realignment of the fMRI timeseries, coregistration of the data to a T_1-weighted anatomical image, and 8 mm full-width at half-maximum (FWHM) Gaussian smoothing.

Analysis. Voxelwise statistics on the salmon data were calculated through an ordinary least-squares estimation of the general linear model (GLM). Predictors of the hemodynamic response were modeled by a boxcar function convolved with a canonical hemodynamic response. A temporal high pass filter of 128 seconds was include to account for low frequency drift. No autocorrelation correction was applied.

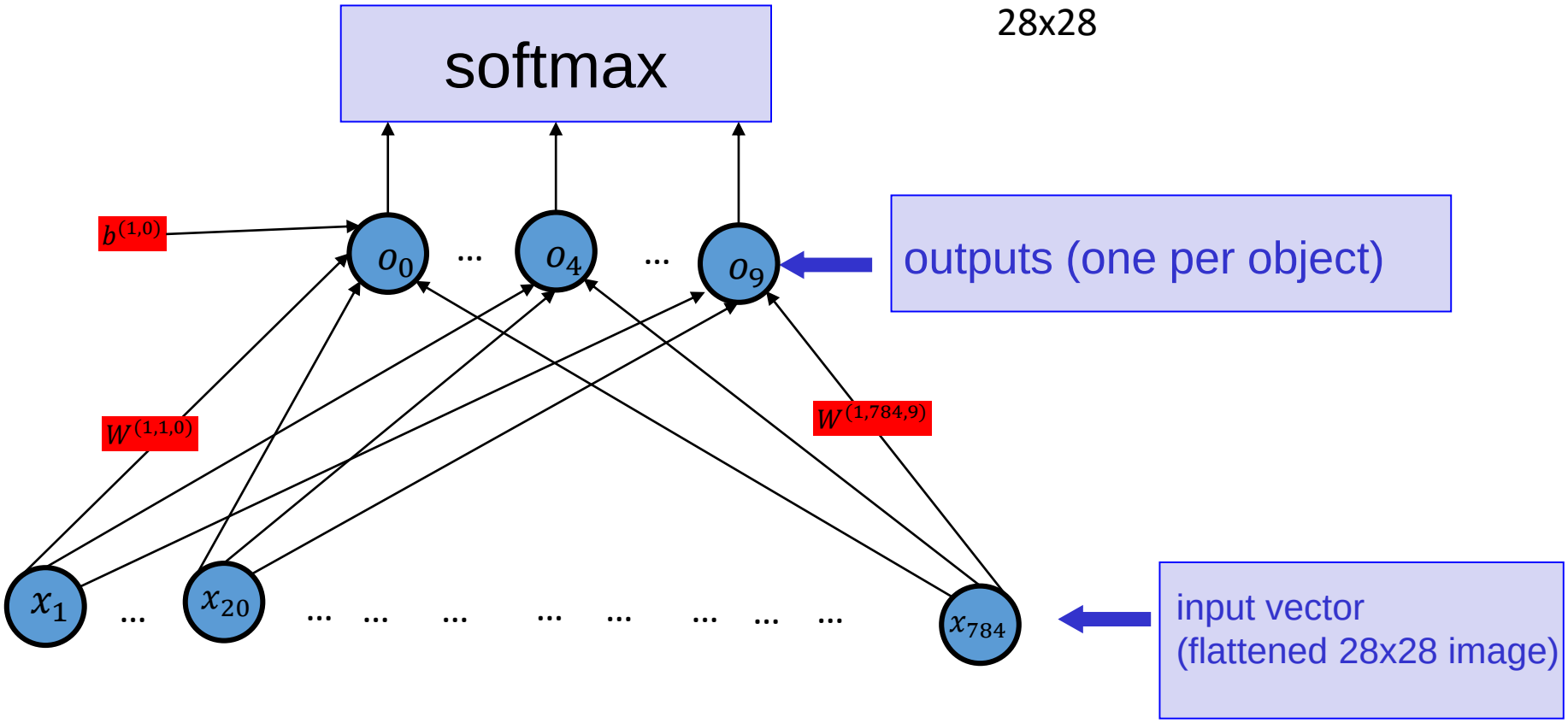
Voxel Selection. Two methods were used for the correction of multiple comparisons in the fMRI results. The first method controlled the overall false discovery rate (FDR) and was based on a method defined by Benjamini and Hochberg (1995). The second method controlled the overall familywise error rate (FWER) through the use of Gaussian random field theory. This was done using algorithms originally devised by Friston et al. (1994).

http://prefrontal.org/files/posters/Bennett-Salmon-2009.pdf

http://www.improbable.com/ig/winners/

Two-Layer Neural Networks for Image Classification

10 objects, all resized to 28x28



(a.k.a. Multinomial Logistic Regression)

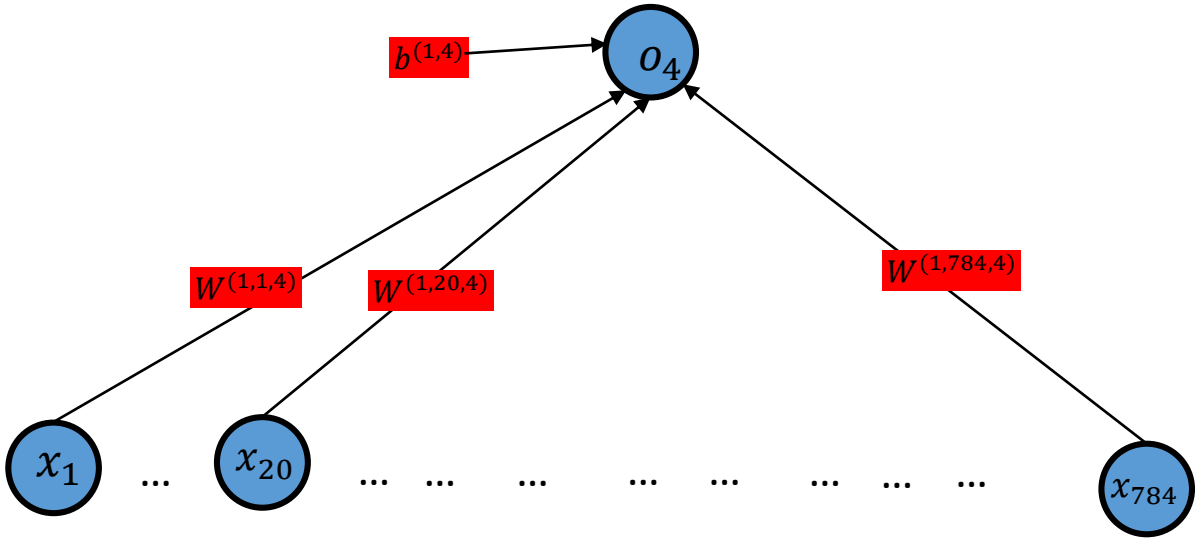
Reminder: Optimizing Neural Networks

- Use Backpropagation to compute the gradient of the cost function (e.g., the –log prob. of the correct answer (a.k.a. cross entropy) w.r.t. the W's and b's for the whole training set, or for a mini-batch of training examples
- Use gradient descent to find the W's and b's that minimize the cost function
- When classifying images, compute the output of the network for
  x=the input image
  and the W's and b's we found minimizing the cost function
- Find which output is the largest, or interpret the outputs of the softmax as the probability estimates for the different objects

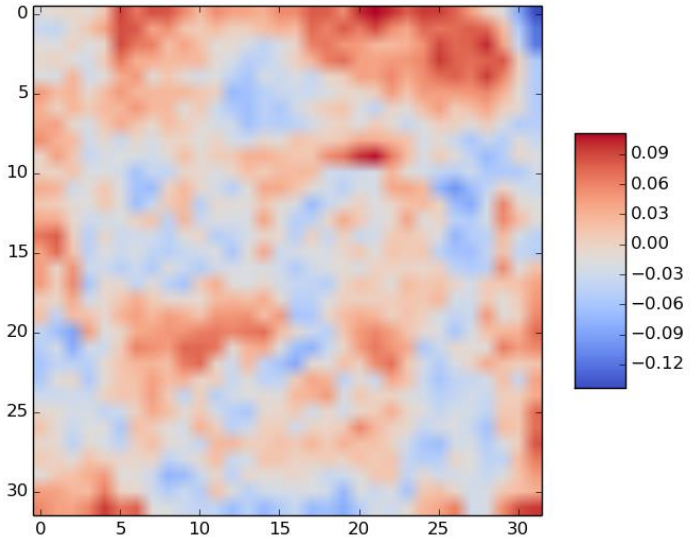
What kind of W's would minimize the cost function?

- E.g., the task is the same as in Project 1: classify an image as one of the 6 actors

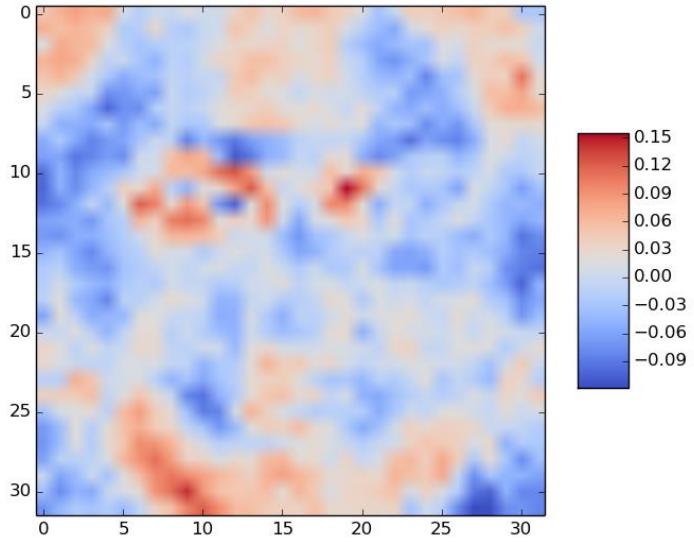
Visualizing the W's



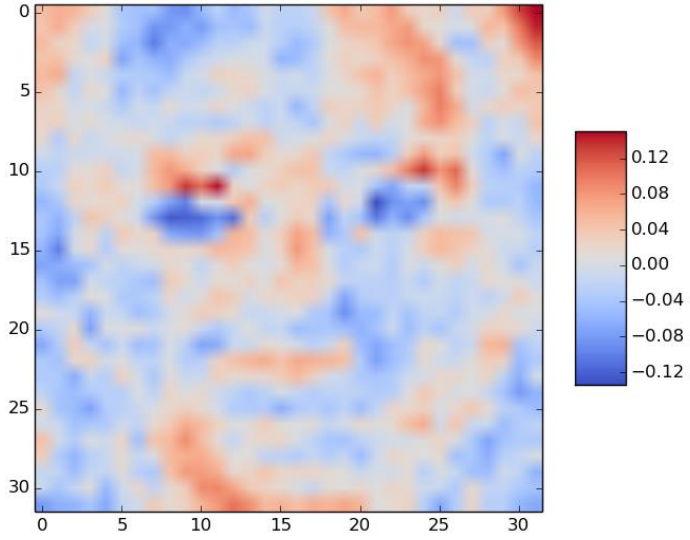
- For a given output unit, we have the strength of the connections from each of the inputs
- To understand what the network is doing, we can think of the $W^{(1,i,4)}$ as an image



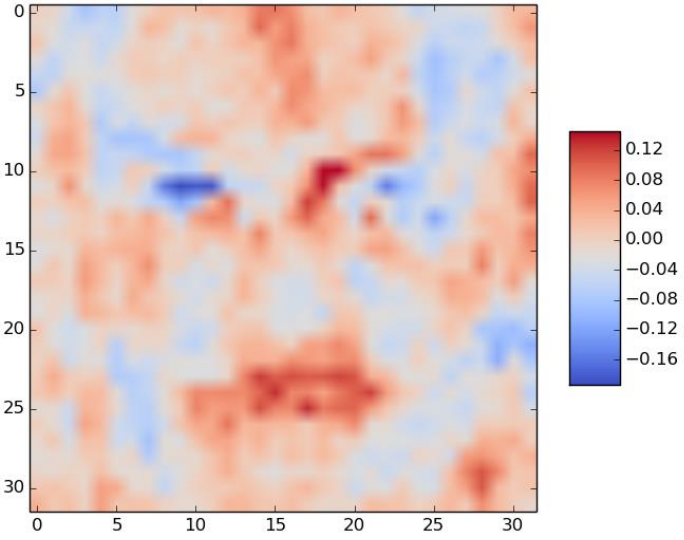
Bracco



Vartan



Radcliffe



Gilpin

Multinomial Logistic Regression with Early Stopping, 40 examples each

The Dot Product $W^{(1,*,j)} \cdot x$

- Note that the input to the unit o_j is

$$W^{(1,*,j)} \cdot x + b^{(1,j)}$$

- For a vector x of a given magnitude, $W^{(1,*,j)} \cdot x$ is as large as possible when $x = \alpha W^{(1,*,j)}$
  - I.e., when x and $W^{(1,*,j)}$ point in the same direction
  - (Explanation on the board: the dot product $u \cdot v$ is the length of the projection of u onto v)
  - That means that o_j is larger when x looks like $W^{(1,*,j)}$, viewed as images
    - (Note: it also means we should make sure all our input x's are of similar magnitudes)

Aside: all the input x's should have the same magnitude

- If $\mathrm{x}^{(1)} = \alpha x^{(2)}$, they are basically the same image, just with different contrast and maximum brightness
- The output of the neural network for x_1 and x_2 should be the same
- Solution: always *normalize* any input x before putting it in the dataset

$$x \rightarrow \frac{x}{|x|}$$

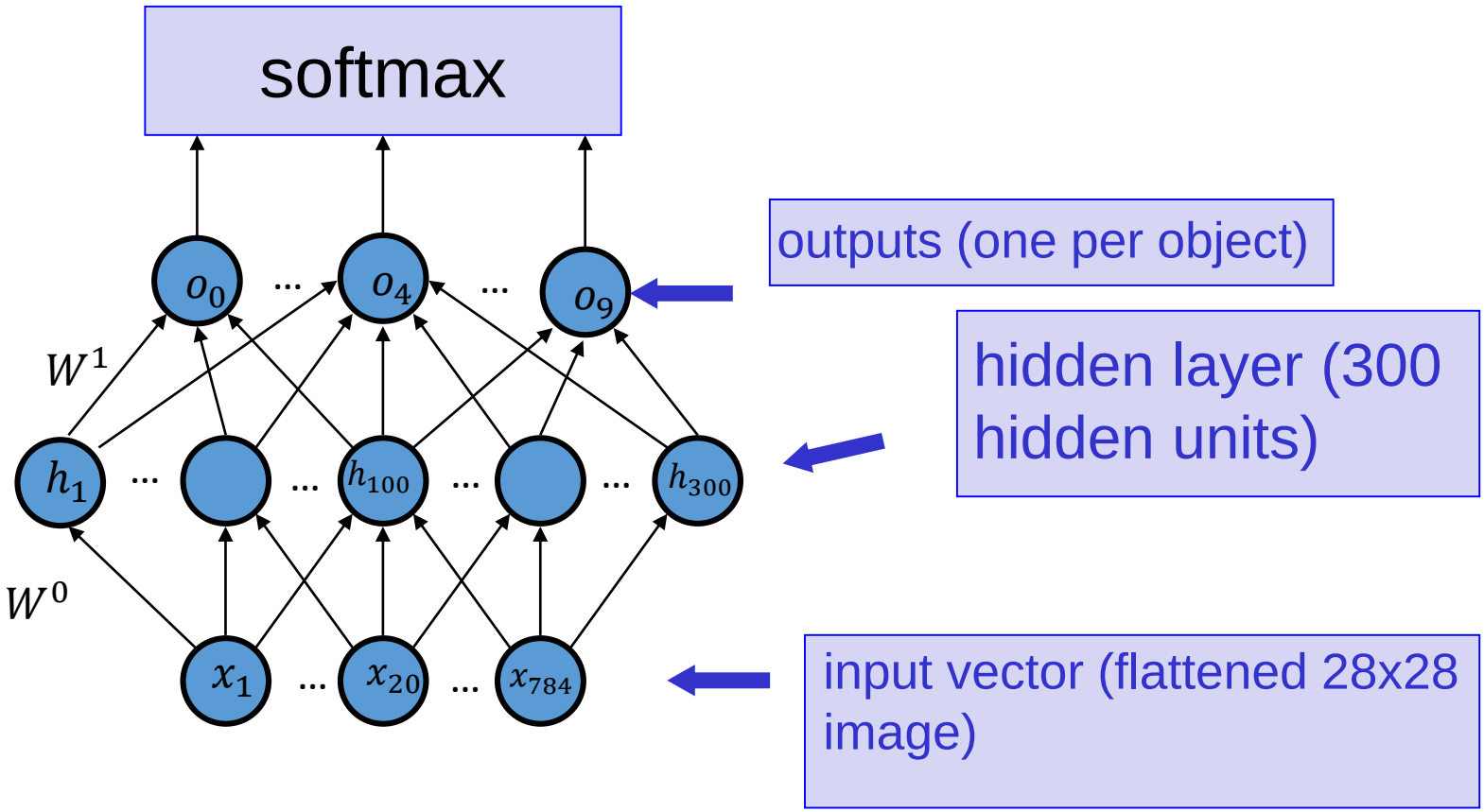
Aside: Normalizing Data

- Usually, we also want for the mean of all the entries in x to be 0
  - Helps prevent dead neurons (reminder of why on the board)
  - $x \to (x - mean(x))$
- Transformation:

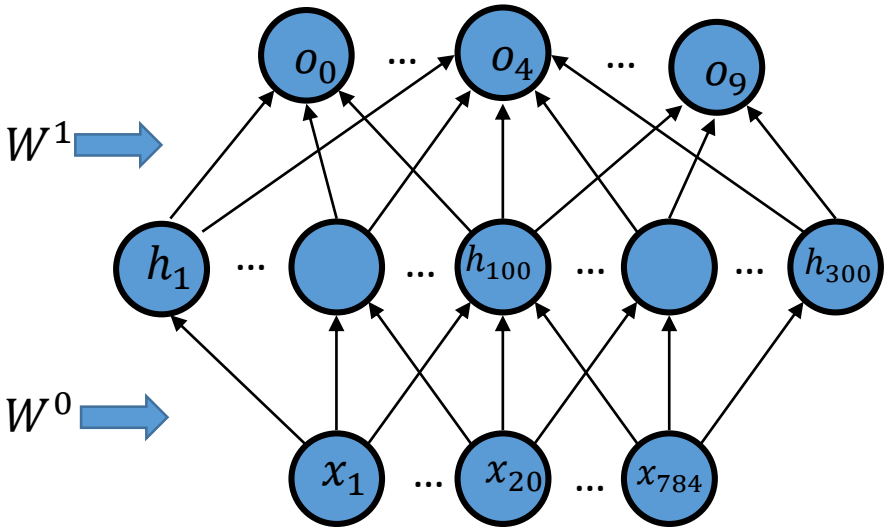
$$x \to \frac{x - mean(x)}{sd(x)}$$

- (Note: for $\text{mean}(x) = 0, sd(x) = |x|/sqrt(\dim(x))$)

Neural Networks with Hidden Layers

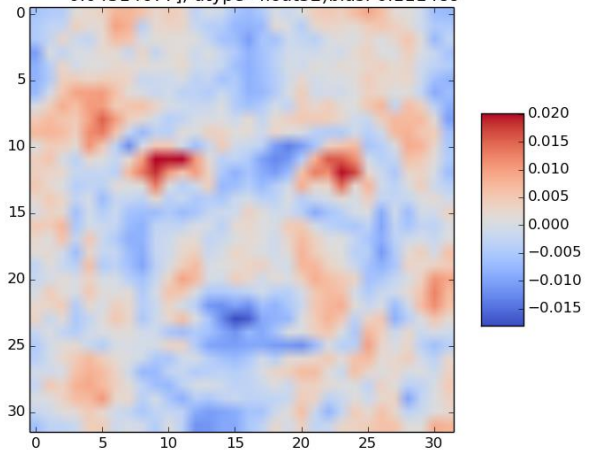


Understanding Hidden Layers

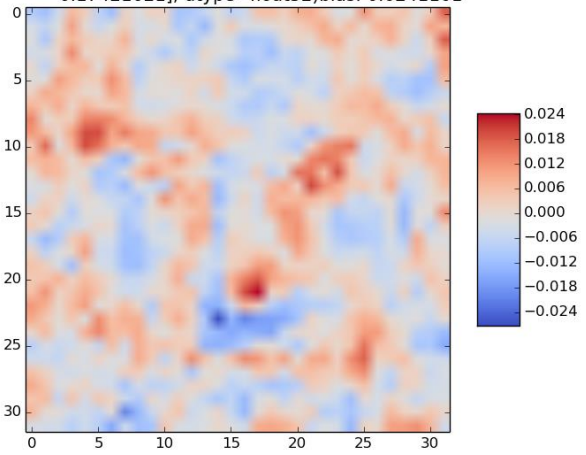


- Can visualize W^0 like before
- But what does it mean for the input to e.g. h_5 to be high?
  - Depends on how h_5 is connected to the output layer!

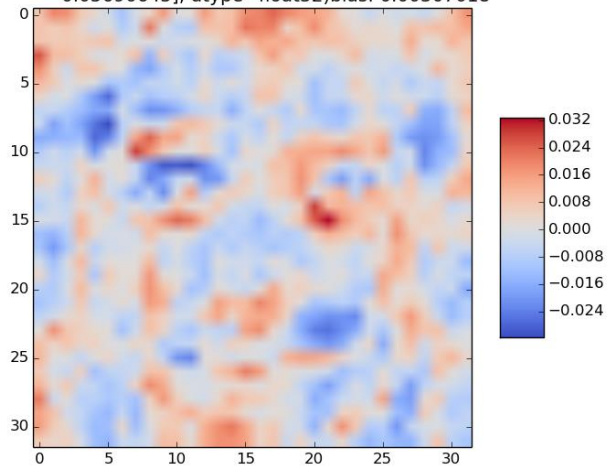
s: array([-0.1032981 , -0.02623156, -0.04492124, 0.04031333, 0.09555781,
0.04314677], dtype=float32)bias: 0.111489



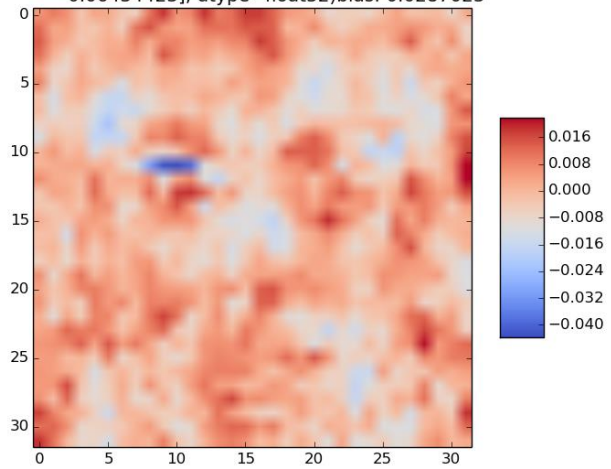
s: array([-0.05847707, 0.02304747, -0.04514949, -0.06355965, 0.02980999,
0.17421021], dtype=float32)bias: 0.0241101



s: array([0.03922304, 0.05484759, 0.06025519, 0.02333124, -0.26381665,
0.05690645], dtype=float32)bias: 0.00307018



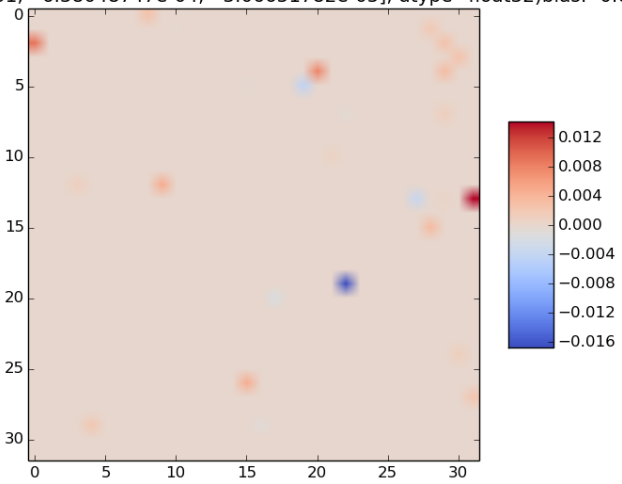
s: array([0.06559545, -0.14167207, 0.06504502, 0.01543506, -0.14153987,
0.06434423], dtype=float32)bias: 0.0287023



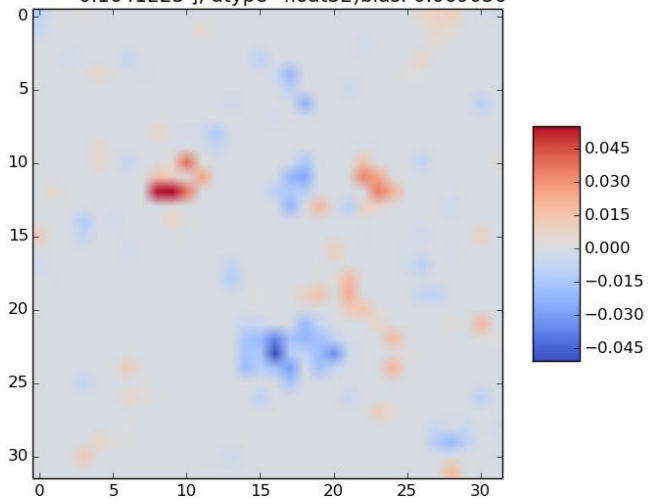
act = ['Angie Harmon‘, 'Peri Gilpin‘, 'Lorraine Bracco', 'Michael Vartan', 'Daniel Radcliffe', 'Gerard Butler']

300 hidden units, 6 actors, 40 examples each, L2-penalized regularization

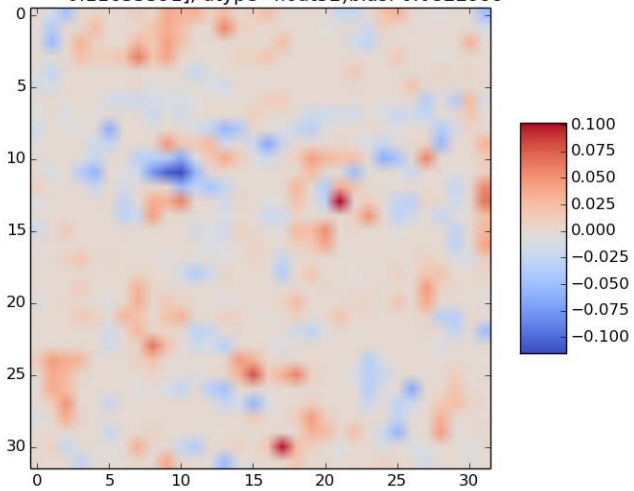
veights: array([3.24537978e-02, 1.03307003e-02, 1.28493230e-06,
50160e-01, -6.38048747e-04, -3.06651782e-05], dtype=float32)bias: -0.0576



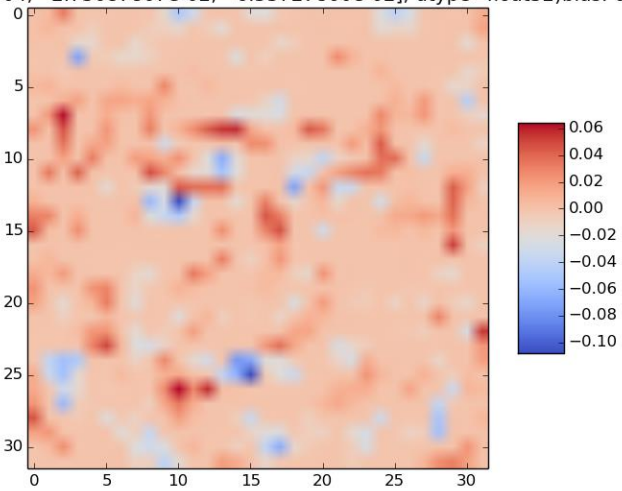
s: array([-0.29401857, -0.01724279, 0.00310232, 0.12068836, 0.0708182 ,
0.1641223], dtype=float32)bias: 0.069056



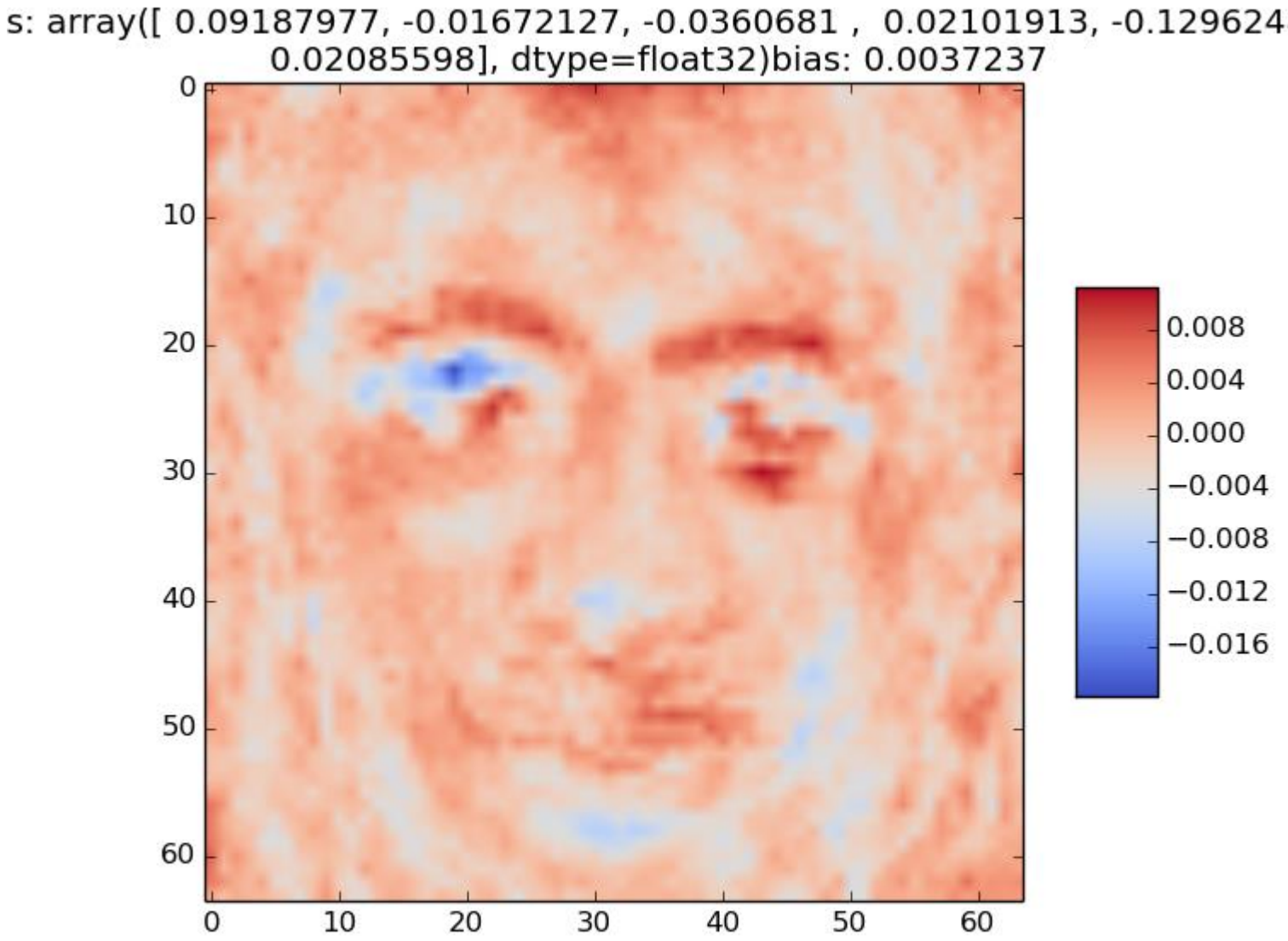
s: array([0.22145636, -0.6399256 , 0.13758378, 0.03394366, -0.37346393,
0.11635391], dtype=float32)bias: 0.0822999



veights: array([-1.94610730e-01, 3.78219485e-01, -6.13273799e-01,
55651e-04, 2.73087807e-02, -6.53727800e-02], dtype=float32)bias: 0.1243

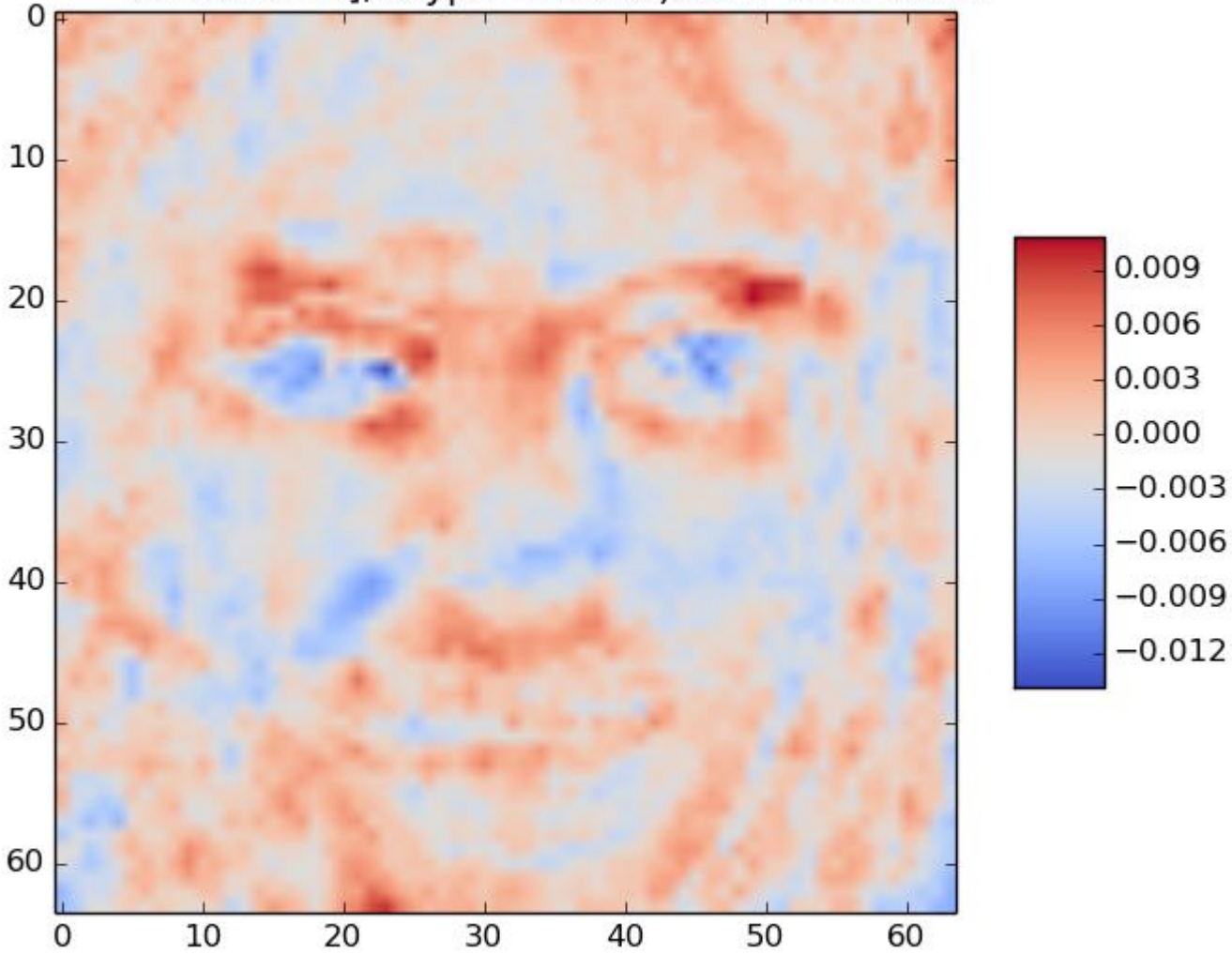


300 hidden units, 6 actors, 40 examples each, L1-penalized regularization

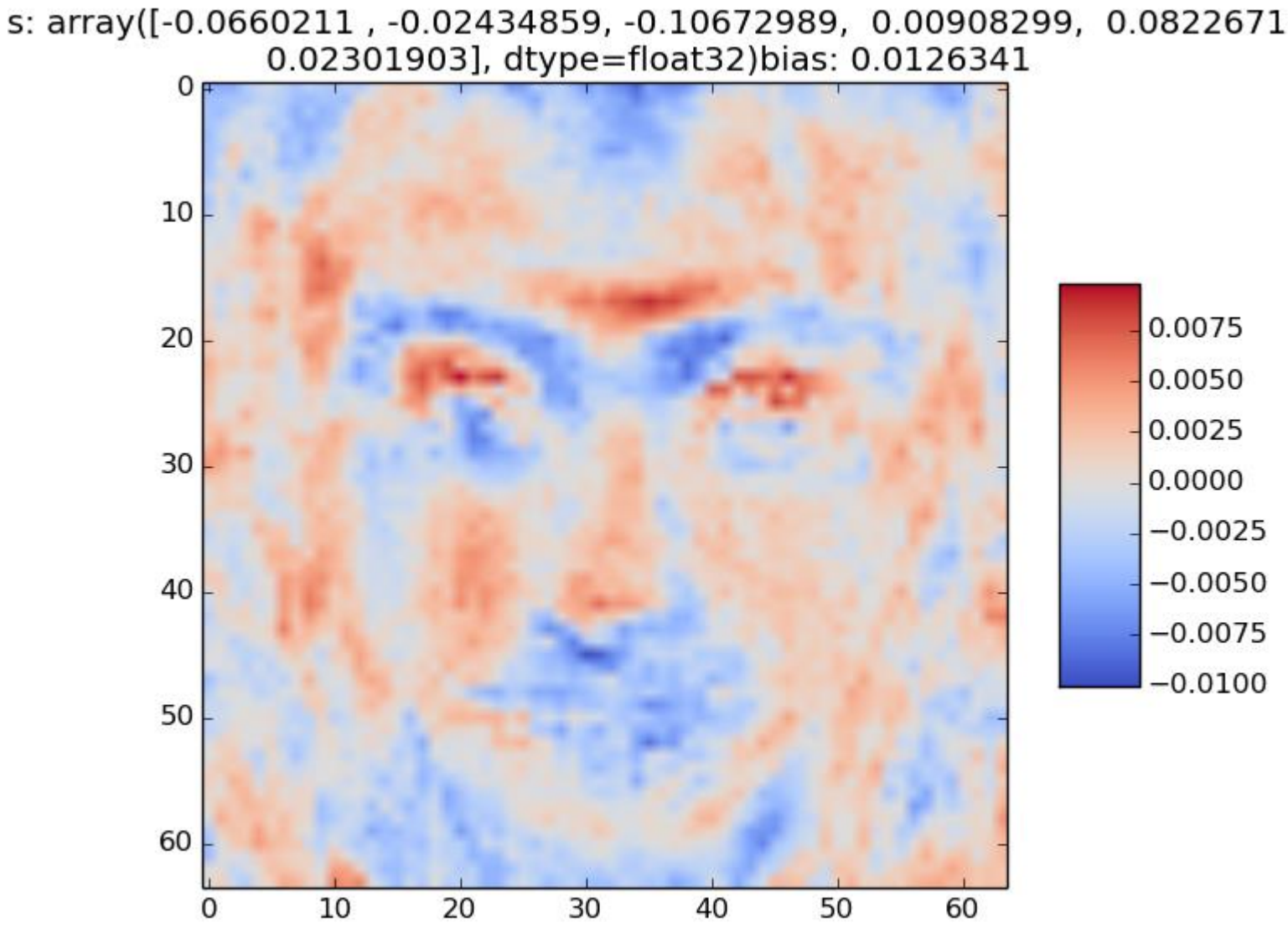


300 hidden units, 6 actors, 40 examples each, L2-penalized regularization, 128x128 images

:s: array([0.031698 , 0.14668576, 0.03825208, 0.01261172, -0.01688866,
-0.02065944], dtype=float32)bias: -0.0516133



300 hidden units, 6 actors, 40 examples each, L2-penalized regularization, 128x128 images



300 hidden units, 6 actors, 40 examples each, L2-penalized regularization, 128x128 images

Hidden Layer Units as Features

- Once we train the neural network, the values units in the hidden layer should be useful for computing the output units.
- The weights W^0 between the input layer and the hidden layer are such that the hidden units are useful
- Think of the hidden units as "features" of the data – summaries of the data that are useful for computing the outputs
- In networks with no hidden layer, we simply compute as many features as there are outputs
  - So the "features" should look like the inputs that we are looking for
- (Recall the XOR example: we computed the feature "x1>.5" and the feature "x2>.5" using hidden units)