Overfitting, Capacity Control, and Training Neural Networks

John Klossner, *The New Yorker*

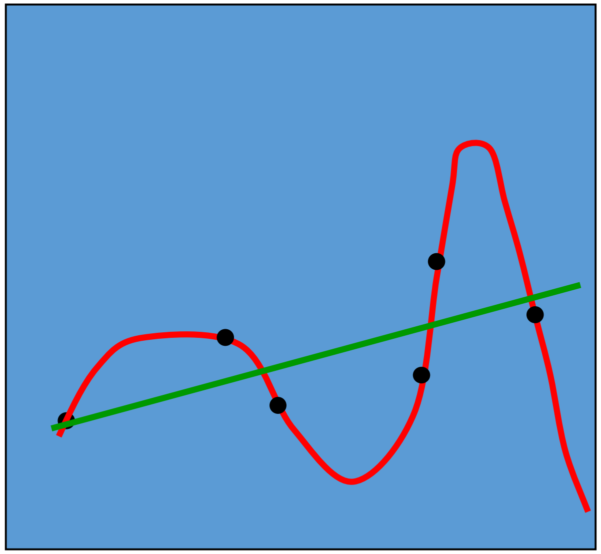
SML310: Research Projects in Data Science, Fall 2019

Michael Guerzhoy

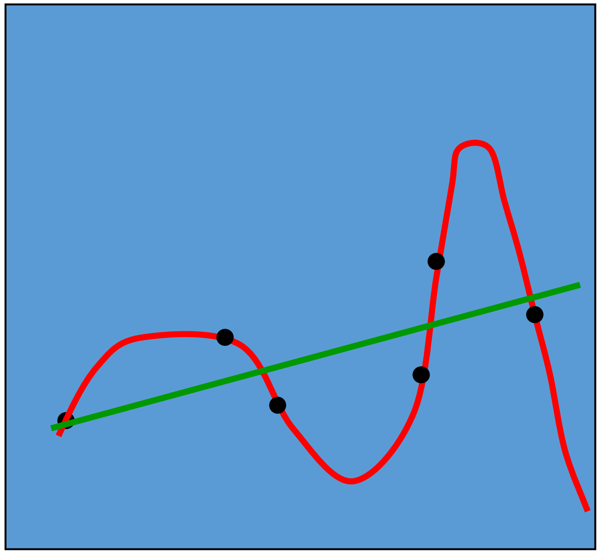
Some slides from Geoffrey Hinton

Overfitting

- Overfitting happens when the model (e.g., a Neural Network, or k-NN, or…) models the specific training set rather than the underlying data from which the training set is taken
  - I.e., because the training set is too small, the network can do extremely well on the training set by modelling its peculiarities

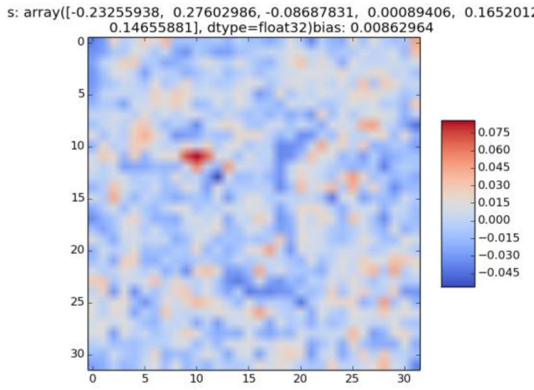


A Simple Example of Overfitting

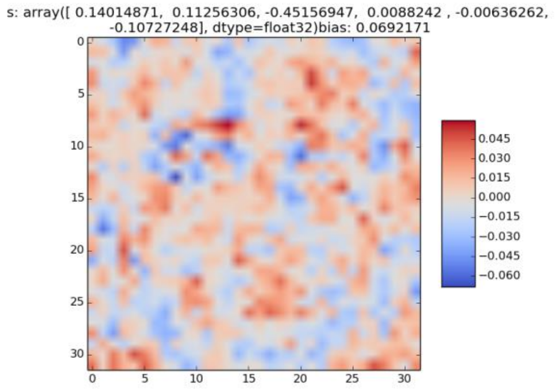


- Which model do you believe?
  - The complicated model fits the data better.
  - But it is not economical
- A model is convincing when it fits a lot of data surprisingly well.
  - It is not surprising that a complicated model can fit a small amount of data.

Overfitting and Faces



```
s: array([-0.23255938,  0.27602986, -0.08687831,  0.00089406,  0.1652012 ,
        0.14655881], dtype=float32)bias: 0.00862964
```



```
s: array([ 0.14014871,  0.11256306, -0.45156947,  0.0088242 , -0.00636262,
       -0.10727248], dtype=float32)bias: 0.0692171
```

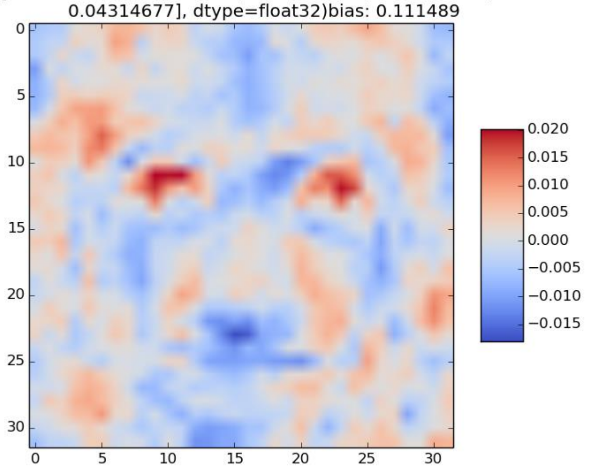
- Above you see examples of W^0 that give near-100% performance on the training set
- The random spots you see are random regularities in the small training set being exploited – exploiting them on the test set won't work, and will possibly lead to bad performance

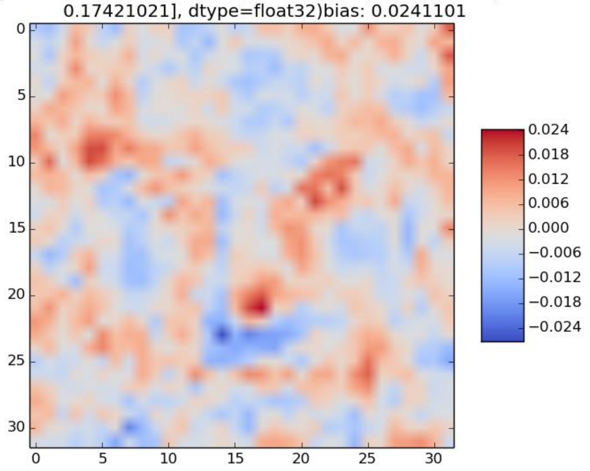
300 Hidden Units, 3000 epochs, no regularization, 40 example per actor, 6 actors

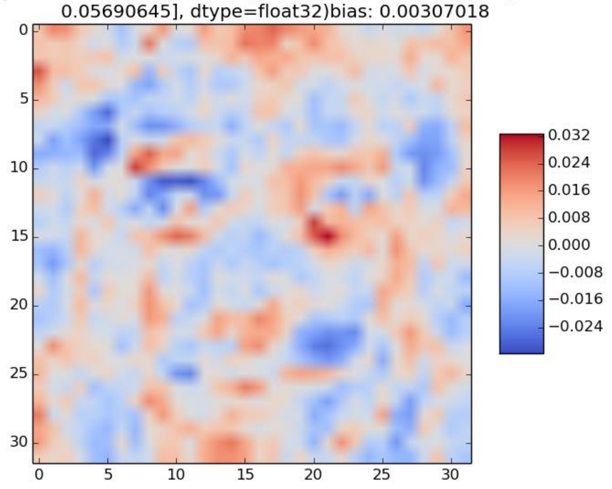
Overfitting: Summary

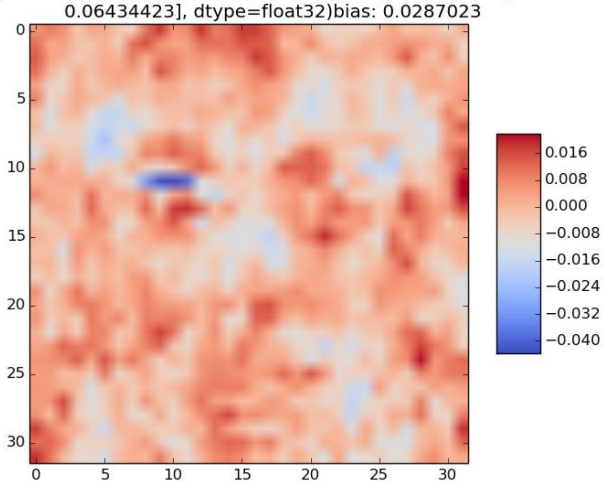
- The training data contains information about the regularities in the mapping from input to output. But it also contains noise
  - The target values may be unreliable.
  - There is sampling error: there will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well.
- Overfitting: a model *making predictions based on accidental regularities in the training set*

Reminder: one-hidden-layer neural networks and faces

```
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
```


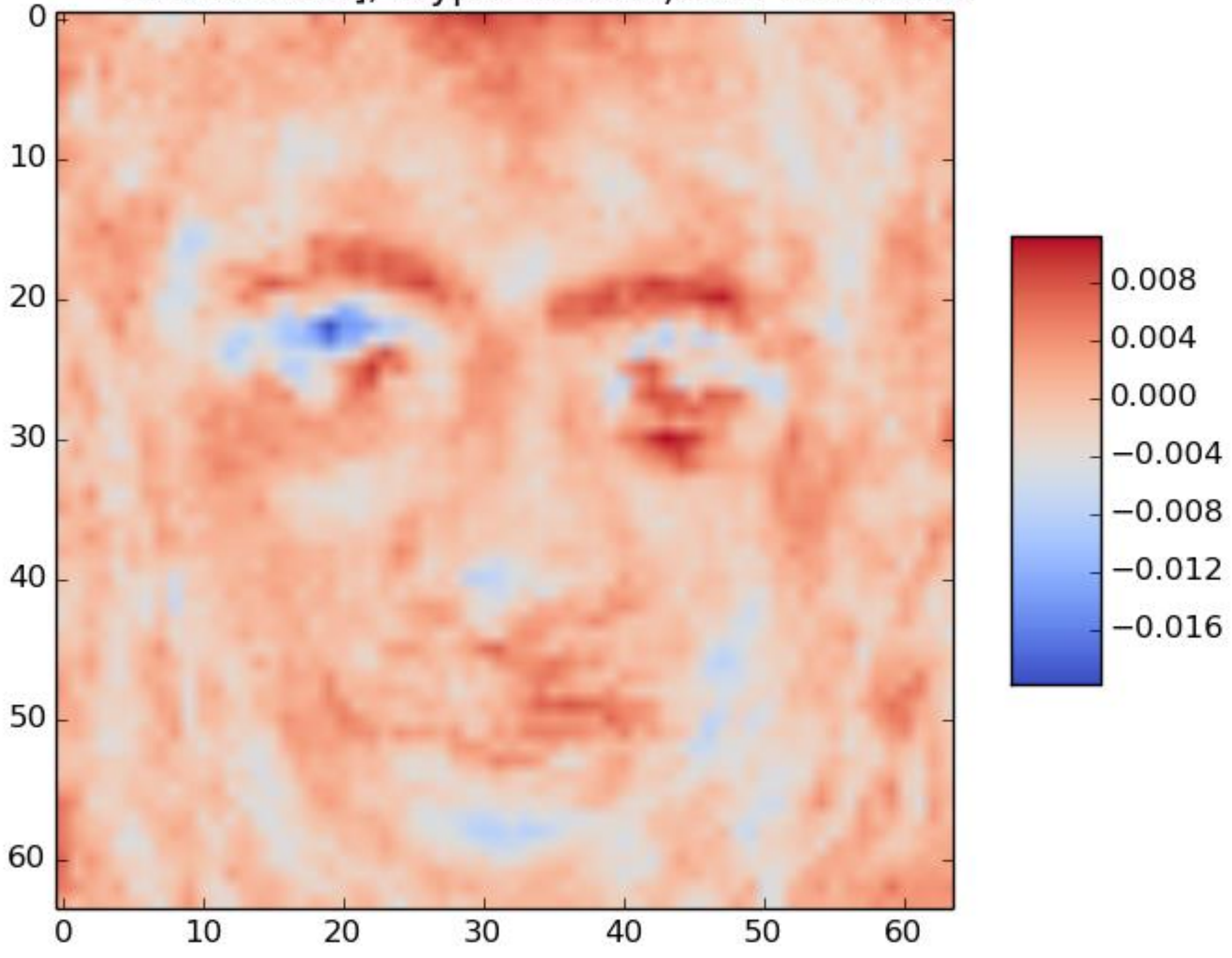
```
s: array([ 0.03922304,  0.05484759,  0.06025519,  0.02333124, -0.26381665,
        0.05690645], dtype=float32)bias: 0.00307018
```


```
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']

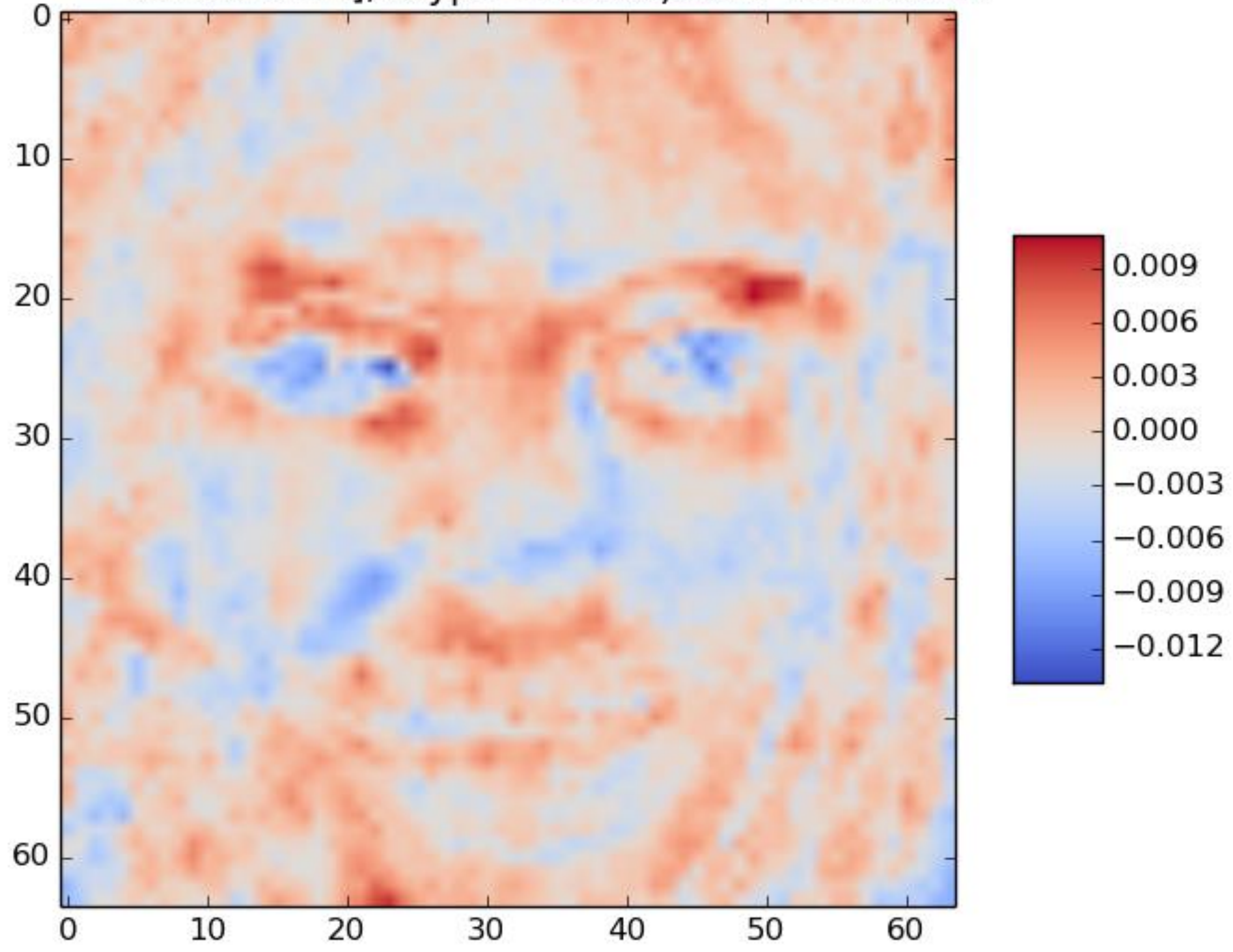
300 hidden units, 6 actors, 40 examples each, L2-penalized

```
s: array([ 0.09187977, -0.01672127, -0.0360681 ,  0.02101913, -0.12962481,
         0.02085598], dtype=float32)bias: 0.0037237
```

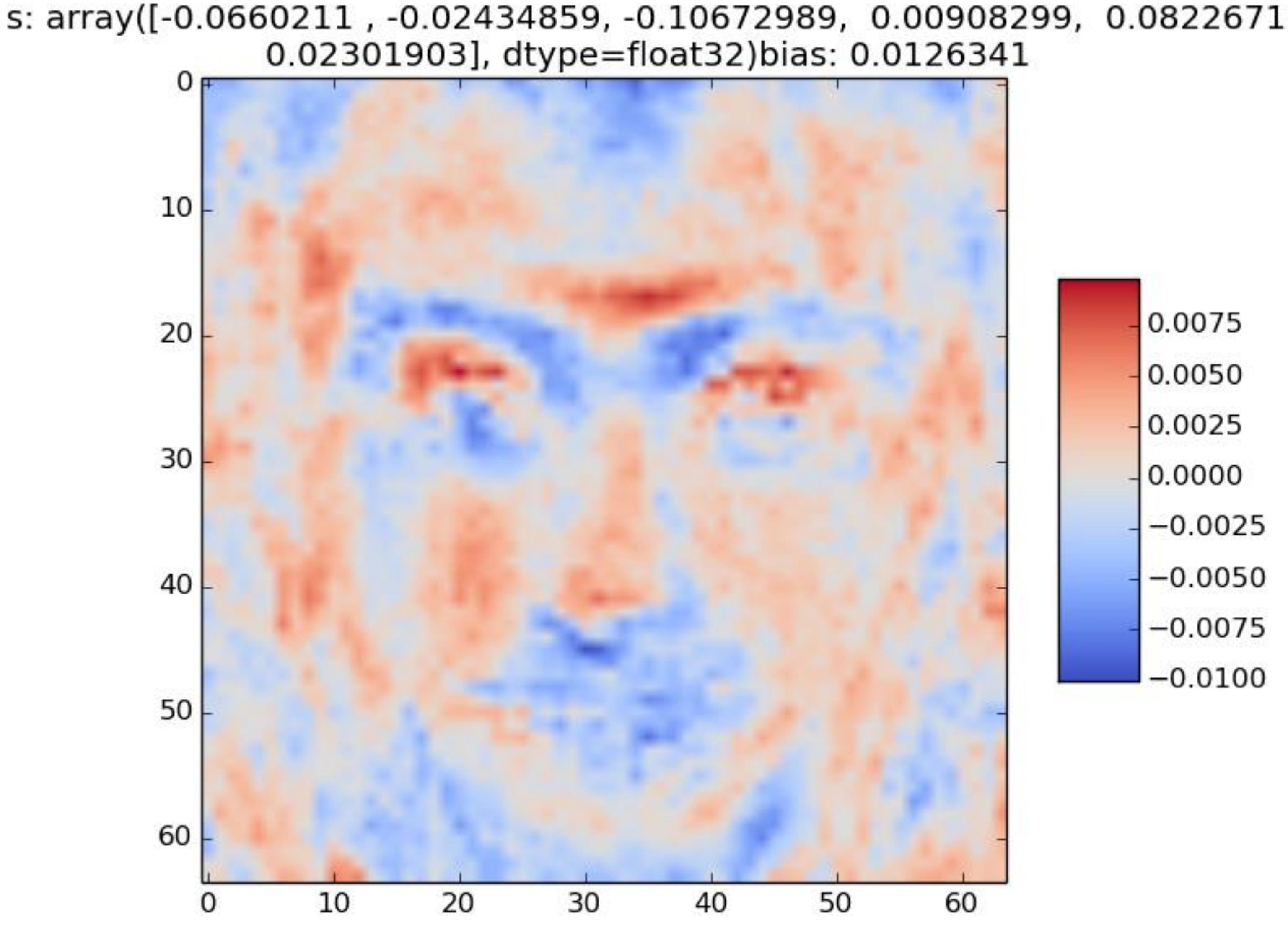


300 hidden units, 6 actors, 40 examples each, L2-penalized, 128x128 images

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



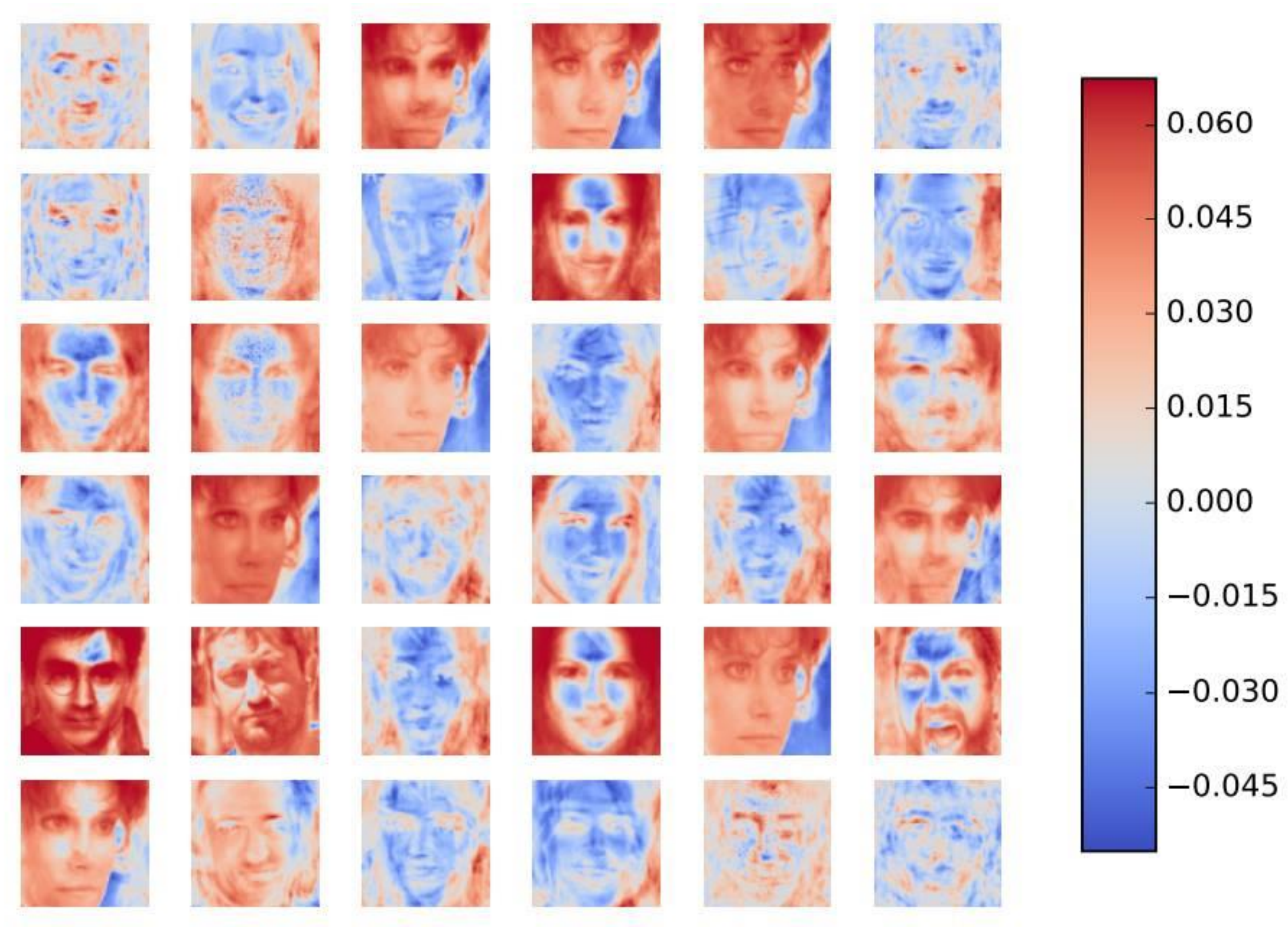
300 hidden units, 6 actors, 40 examples each, L2-penalized, 128x128 images



s: array([-0.0660211 , -0.02434859, -0.10672989, 0.00908299, 0.08226717, 0.02301903], dtype=float32)bias: 0.0126341

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

Overfitting with a hidden layer



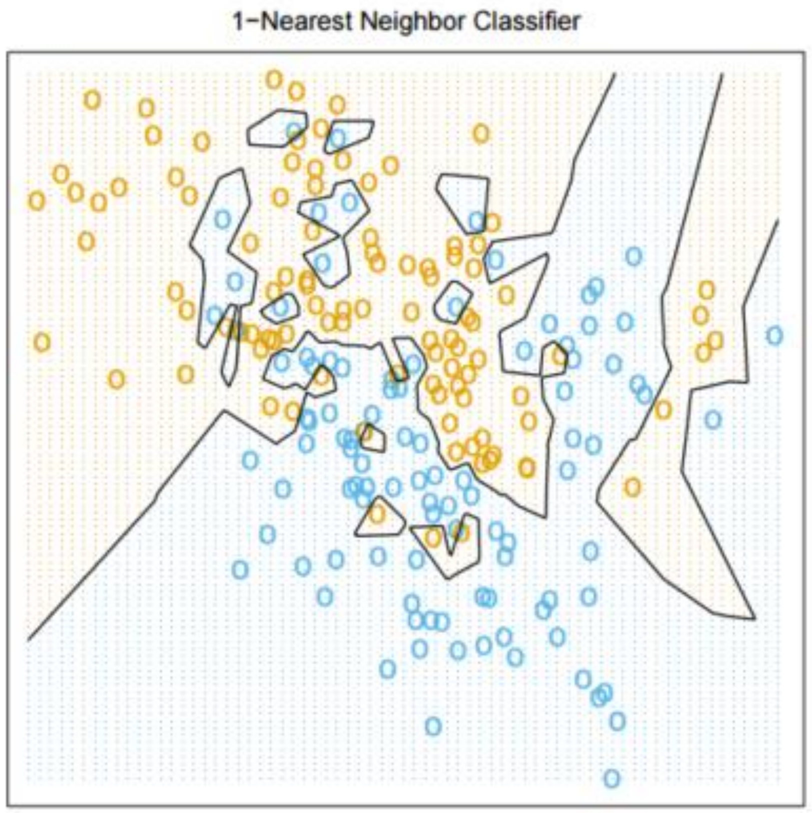
300 units + heavy-duty optimization

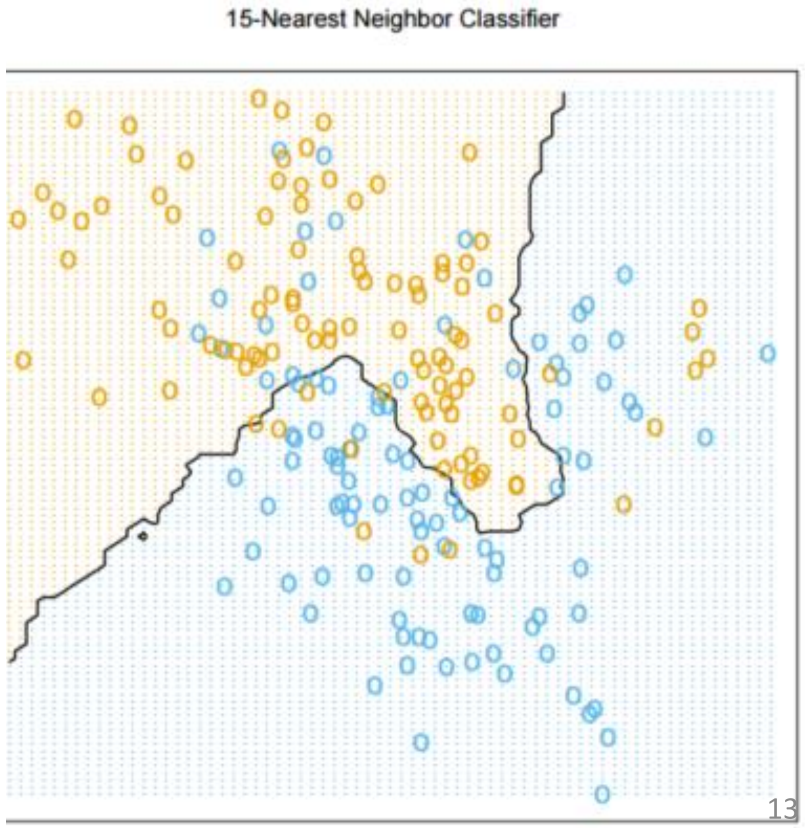
Preventing overfitting

- Use a model that has the right capacity:
  - Capacity: ability to produce different outputs depending on the input
    - Need enough to model the true regularities
    - Want to not have enough capacity to also model the spurious regularities (assuming they are weaker)
- Fitting curves in 2D:
  - Only fit lines, not higher-degree polynomials (example on the board)
  - Only fit quadratics, not higher degree polynomials

Reminder: Nearest Neighbours

- More nearest-neighbours $\Rightarrow$ less capacity
  - More complicated decision surfaces are not possible





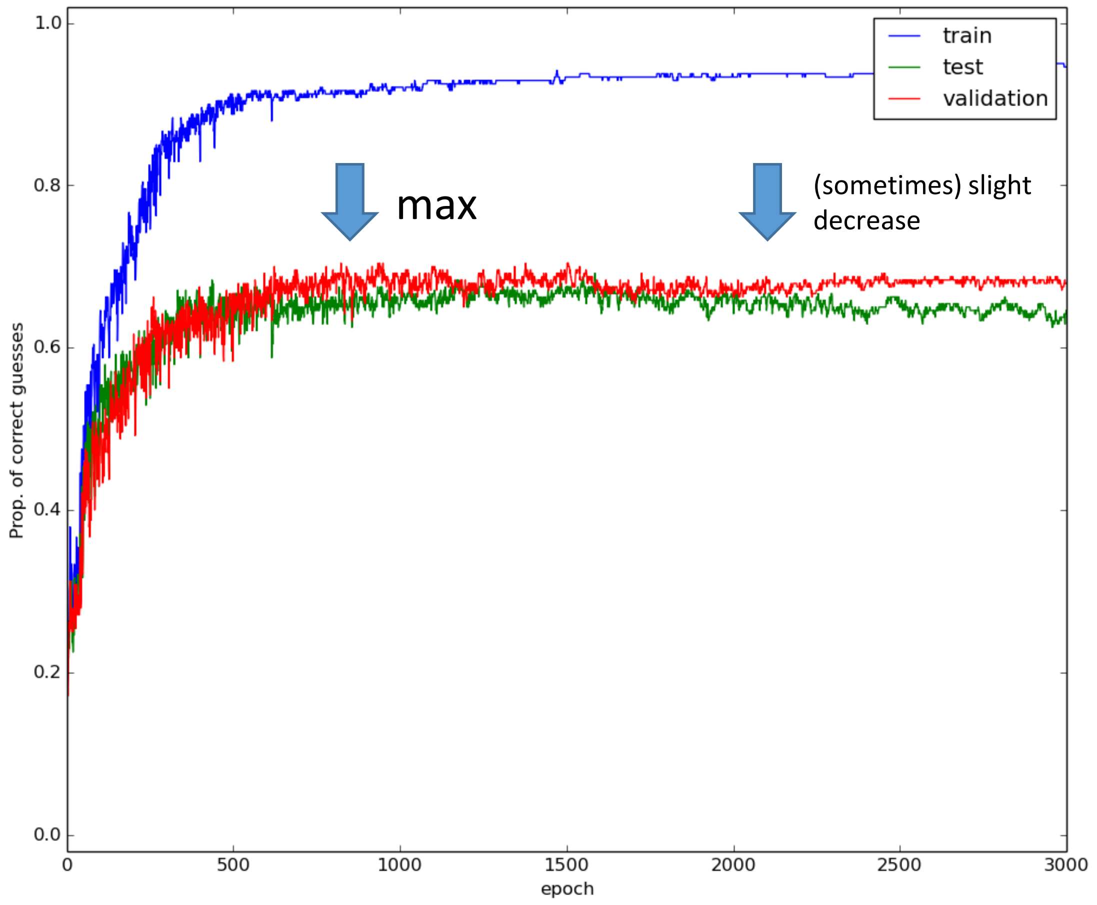
Limiting the Capacity of a Neural Network

- Limit the number of hidden units
- Limit the size of the weights
  - Works for most classifiers
- Stop the learning before we have time to overfit
  - Works for many classifiers
- Combine multiple networks
  - Works for most classifiers
- Dropout

Learning Curves

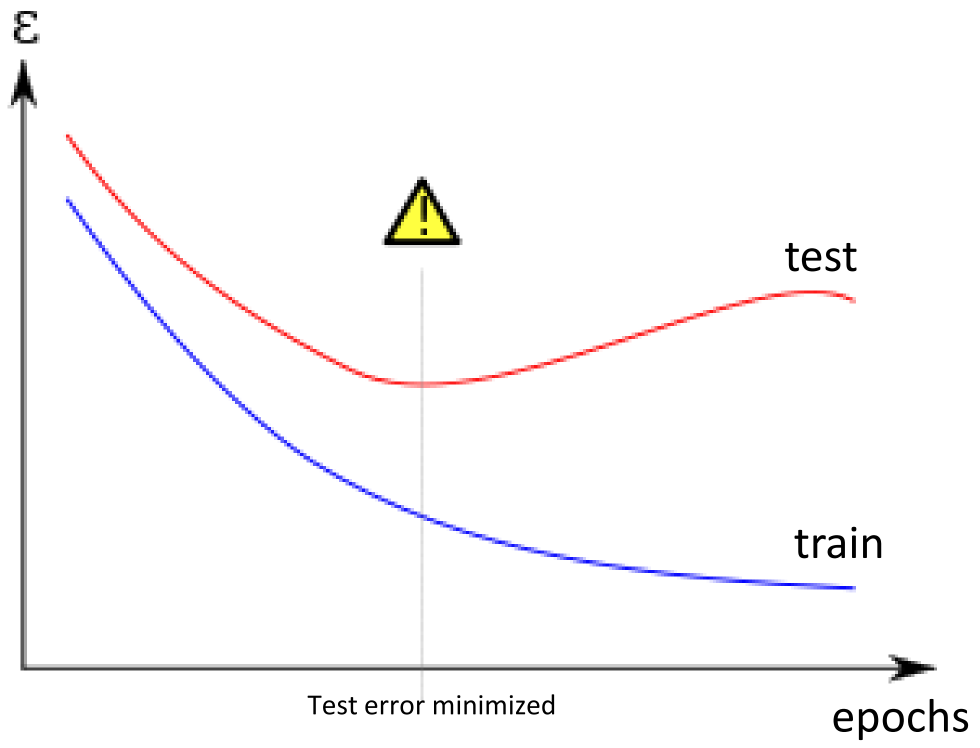
- Split the data into a training set, validation set, and test set.
- Minimize the cost function on the training set, measure performance on all sets
- Plot the performance on the three sets vs. the number of optimization iterations performed
  - Optimization iteration i:
    $$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla cost(\theta, x_{train}, y_{train})$$
    - More details later

“Typical” Learning Curves



300-unit hidden layer. 6 people, 80 examples each. Best test performance: 68%

Wikipedia version



(Basically a fairytale: the moral of the story is kind of true, but things rarely look this nice)

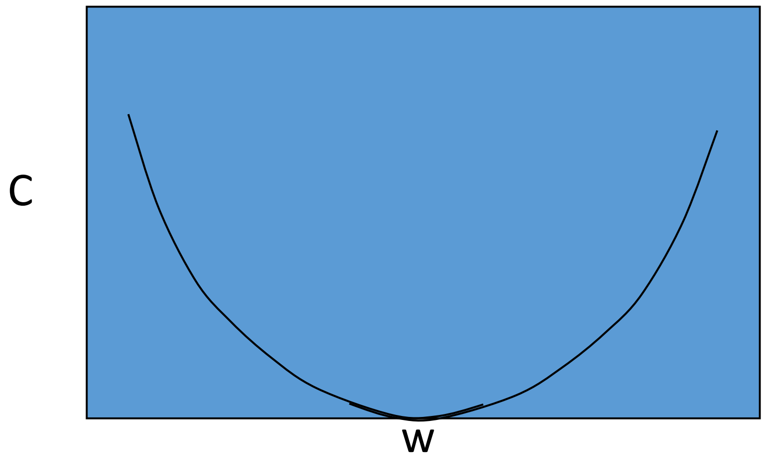
Learning Curves

- The training performance always increases
  - (Assuming the performance is closely enough related to the cost we're optimizing – we sometimes also plot the cost directly)
- The test and validation curve should be the same, up to sampling error (i.e., variation due to the fact that the sets are small and sometimes things work better on one of them by chance)
- The training and validation performance sometimes initially decreases and the decreases as we minimize the cost function

Weight Decay: Limiting the size of the weights

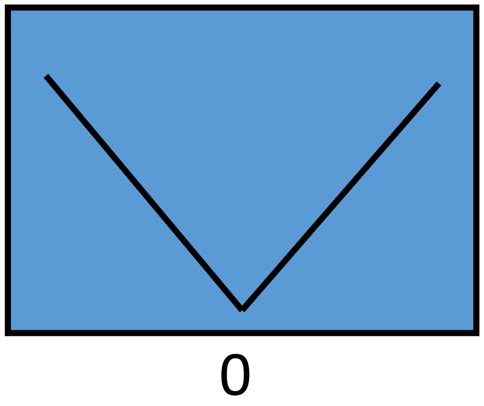
- Weight-decay involves adding an extra term to the cost function that penalizes the squared weights.
  - Keeps weights small unless they have big error derivatives.

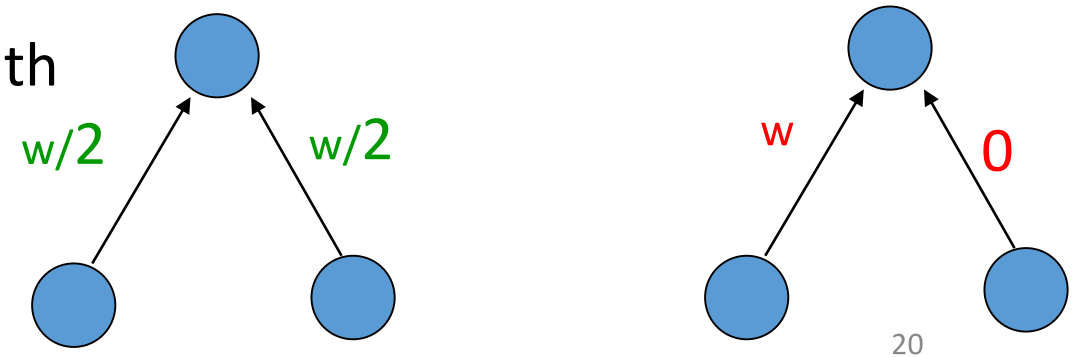
$$cost_{WD} = cost + \frac{\lambda}{2}\sum_{i,j,k}(W^{(k,i,j)})^2$$



Other kinds of weight penalty

- Sometimes it works better to penalize the absolute values of the weights. (I.e., we penalized the L1 norm of the weights rather than the L2 norm)
- Sometimes leads to smaller test errors
- Makes some weights zero
- Compared to the square penalty, which would not tend to do that
- This is sometimes helpful with interpreting the features



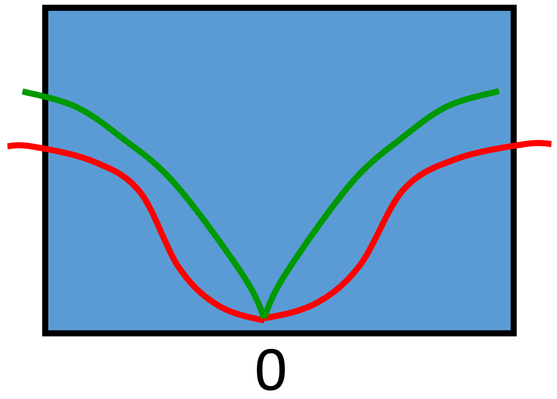


Terminology

- L2 regularization: “ridge regression”
- L1 regularization: “LASSO”
    - Least Absolute Shrinkage and Selection Operator

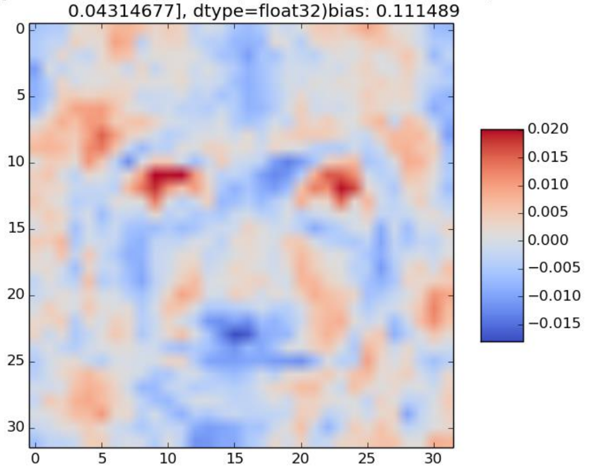
Another kind of Weight penalty

- Sometimes it works better to use a weight penalty that has negligible effect on large weights.
  - Some weights *need* to be large for the neural network to work correctly!

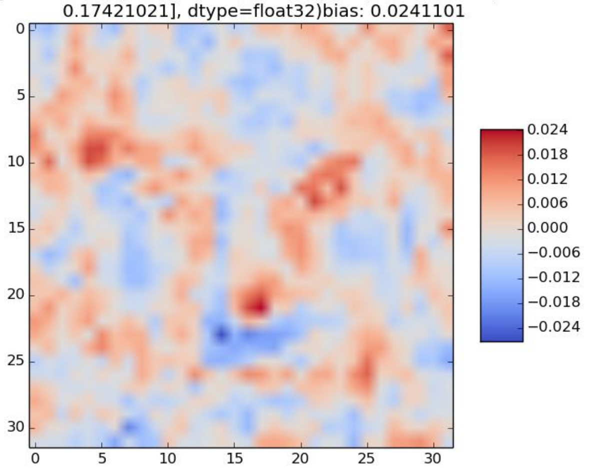


Geman-McClure loss

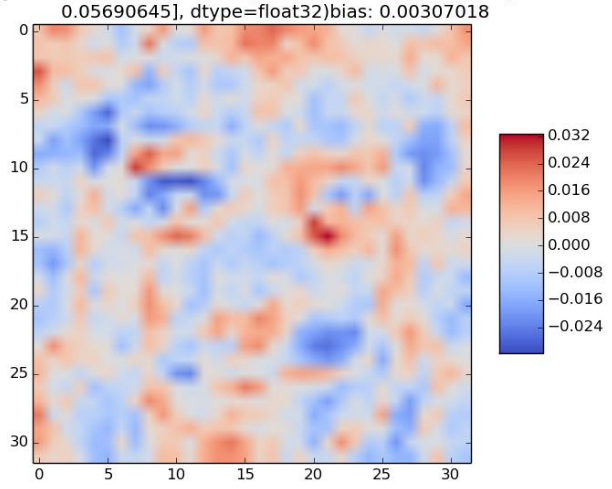
```
s: array([-0.1032981 , -0.02623156, -0.04492124,  0.04031333,  0.09555781,
       0.04314677], dtype=float32)bias: 0.111489
```



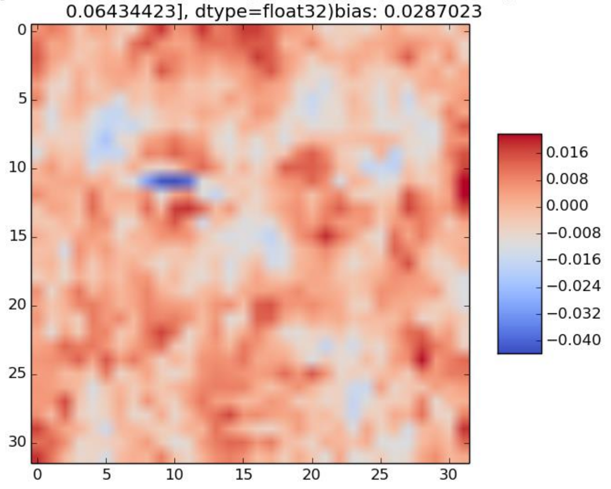
```
s: array([-0.05847707,  0.02304747, -0.04514949, -0.06355965,  0.02980999,
       0.17421021], dtype=float32)bias: 0.0241101
```



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

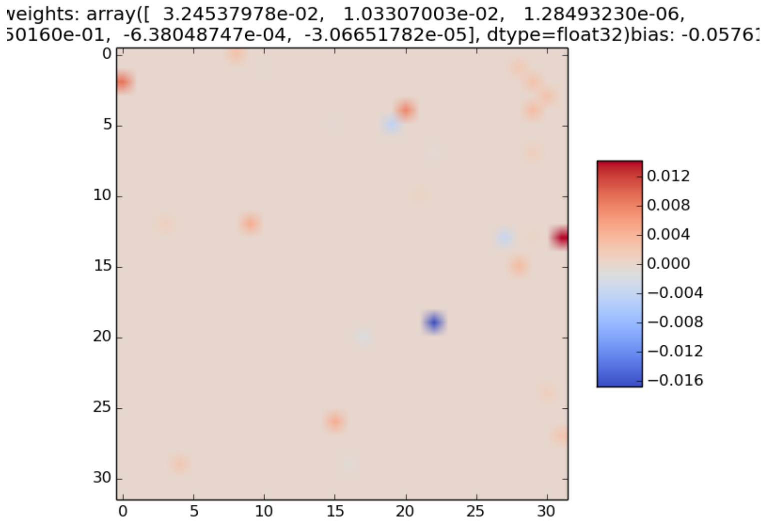


```
s: array([ 0.06559545, -0.14167207,  0.06504502,  0.01543506, -0.14153987,
       0.06434423], dtype=float32)bias: 0.0287023
```

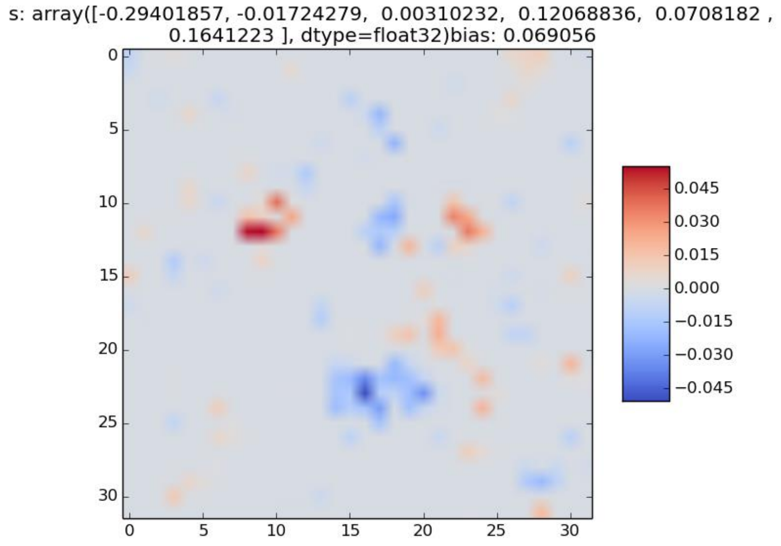


act = ['Angie Harmon‘, 'Peri Gilpin‘, 'Lorraine Bracco', 'Michael Vartan', 'Daniel Radcliffe', 'Gerard Butler']

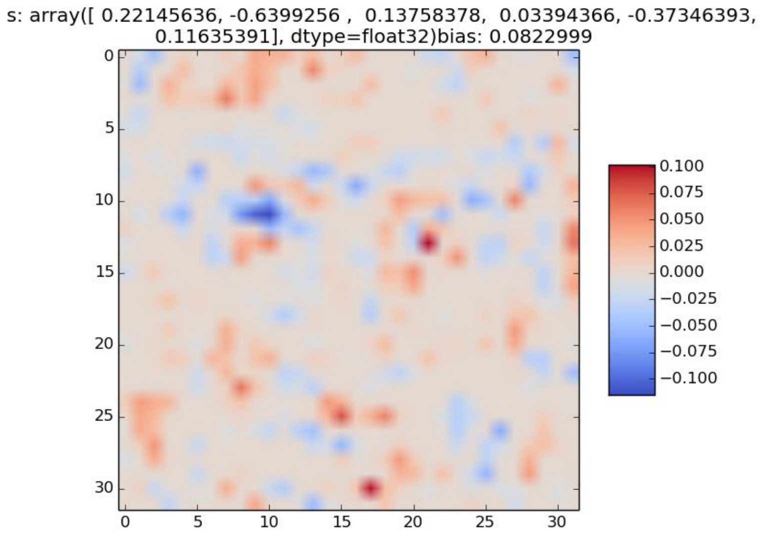
300 hidden units, 6 actors, 40 examples each, L2-penalized



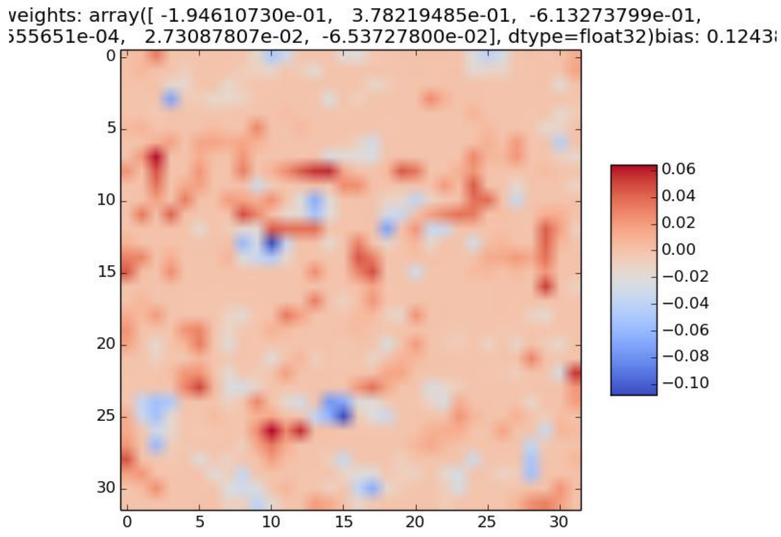
```
veights: array([  3.24537978e-02,   1.03307003e-02,   1.28493230e-06,
50160e-01,  -6.38048747e-04,  -3.06651782e-05], dtype=float32)bias: -0.0576
```



```
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
```

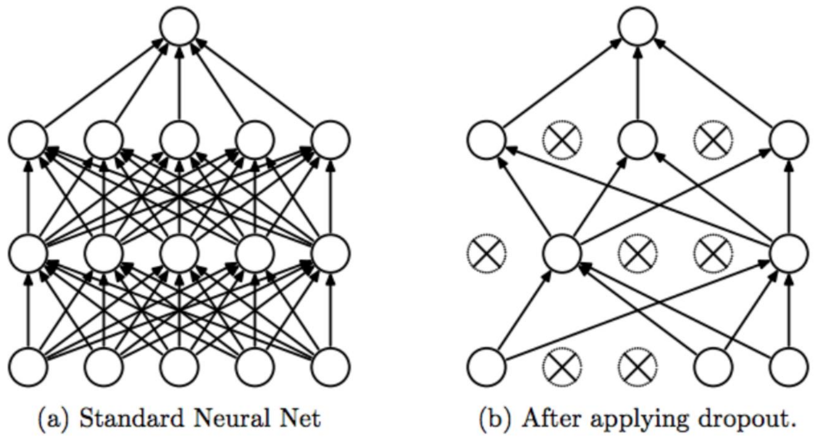
300 hidden units, 6 actors, 40 examples each, L1-penalized

Combining networks

- When the amount of training data is limited, we need to avoid overfitting.
    - Averaging the predictions of many different networks is a good way to do this.
    - It works best if the networks are as different as possible.
- If the data is really a mixture of several different “regimes” it is helpful to identify these regimes and use a separate, simple model for each regime.
    - We want to use the desired outputs to help cluster cases into regimes. (But we don’t know how to do that in advance – need to train the network first)

Dropout

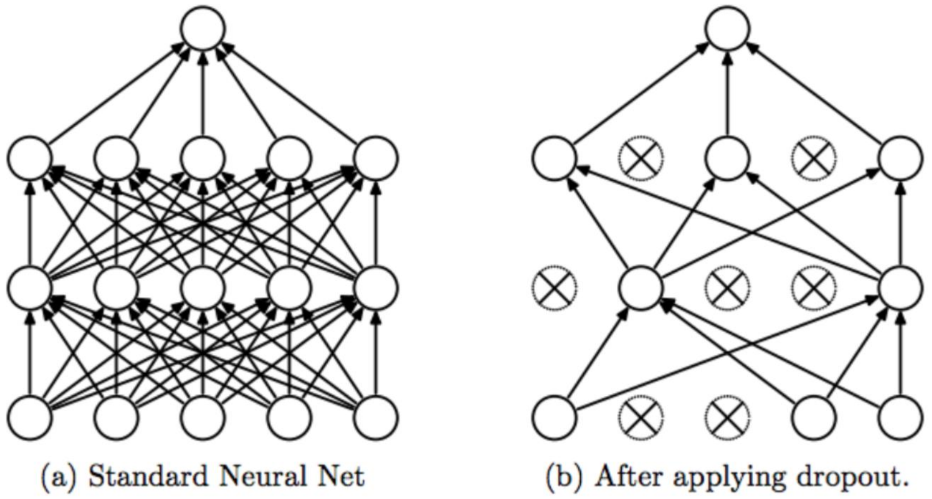
- During training, hidden units are set to 0 with probability $(1 - p)$



- When computing test outputs, scale all activations by the factor of p
    - Keeps the scale of the output consistent, and gives the right output in expectation

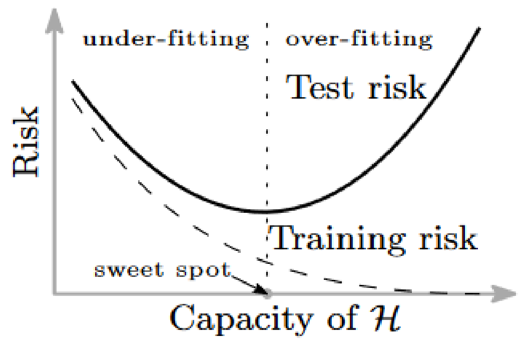
Why does dropout prevent overfitting?

- Prevents dependence between units
  - Each unit must be “useful” independent of other units
  - We cannot learn a network that depends on complex activation patterns

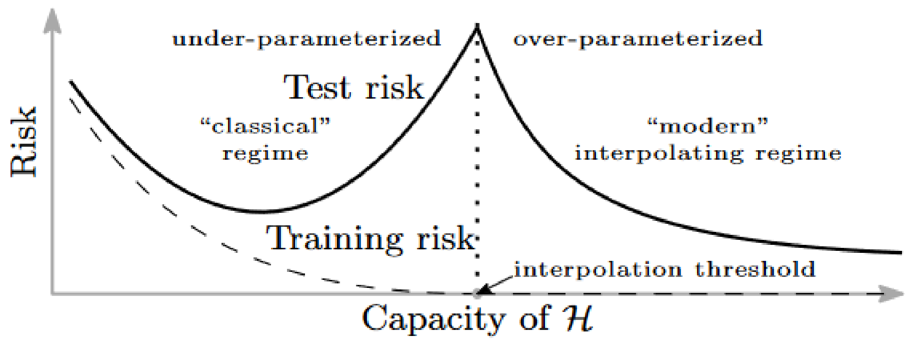


The Risk curve

risk = error



(a)



(b)

https://arxiv.org/pdf/1812.11118.pdf

Belkin et al.'s "double descent" (2019)