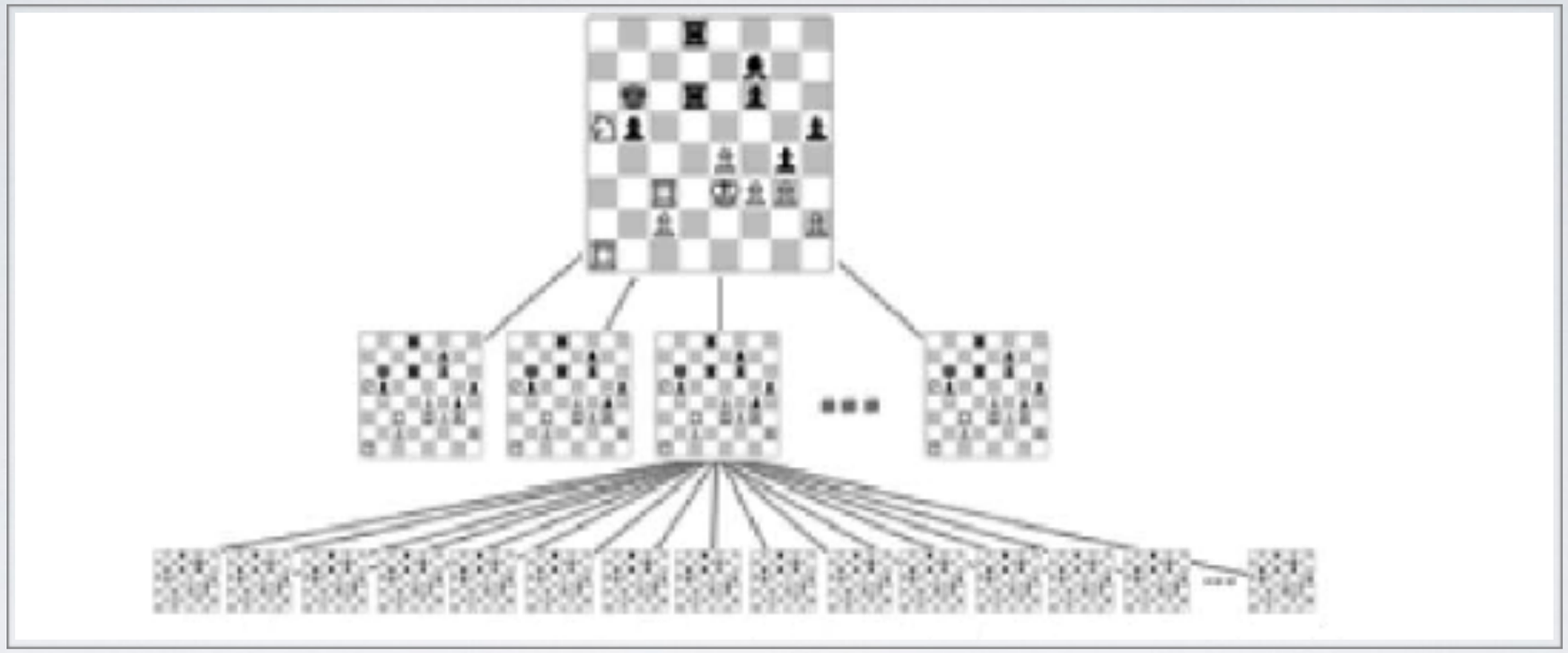
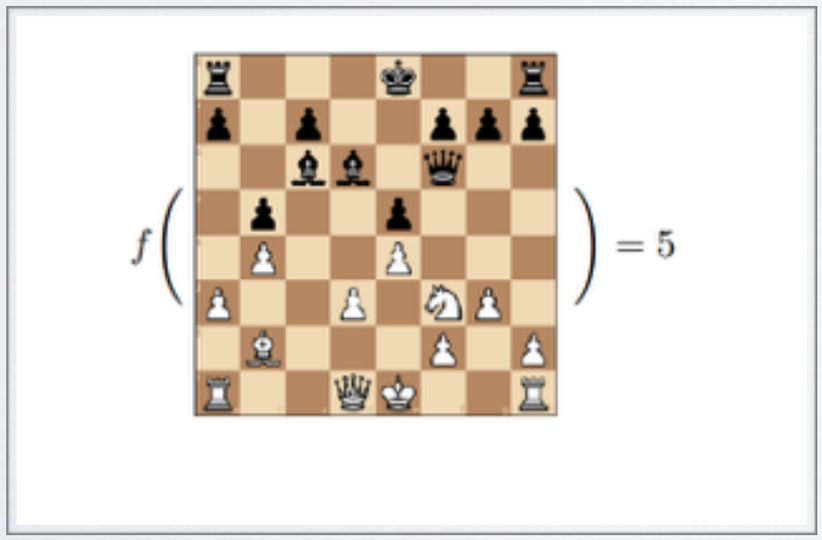
LEARNING CHESS USING CONVOLUTIONAL NEURAL NETWORKS

Debjani Banerjee

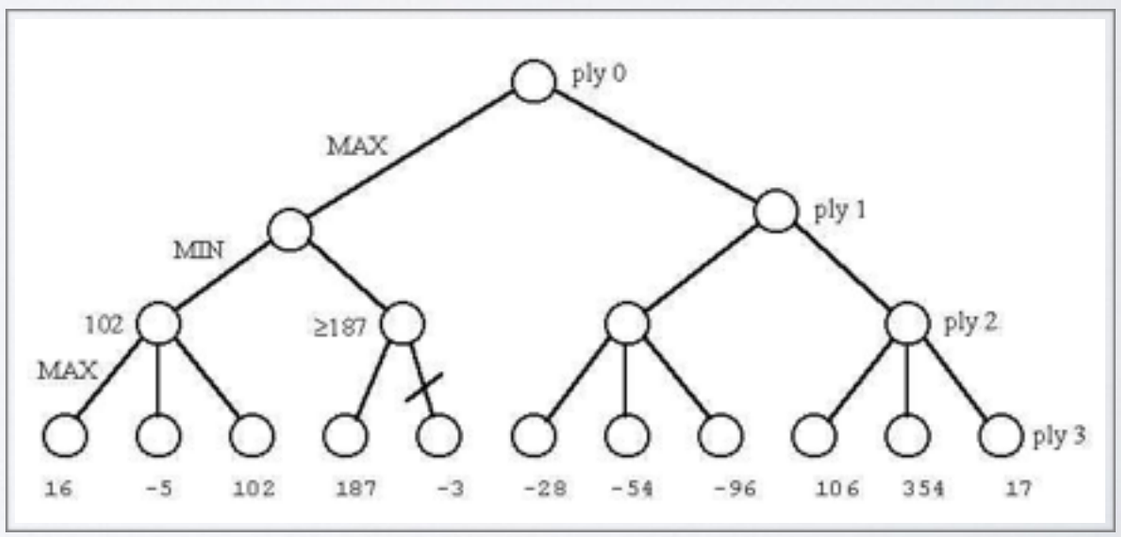
CHESS AI



Chess Search Tree

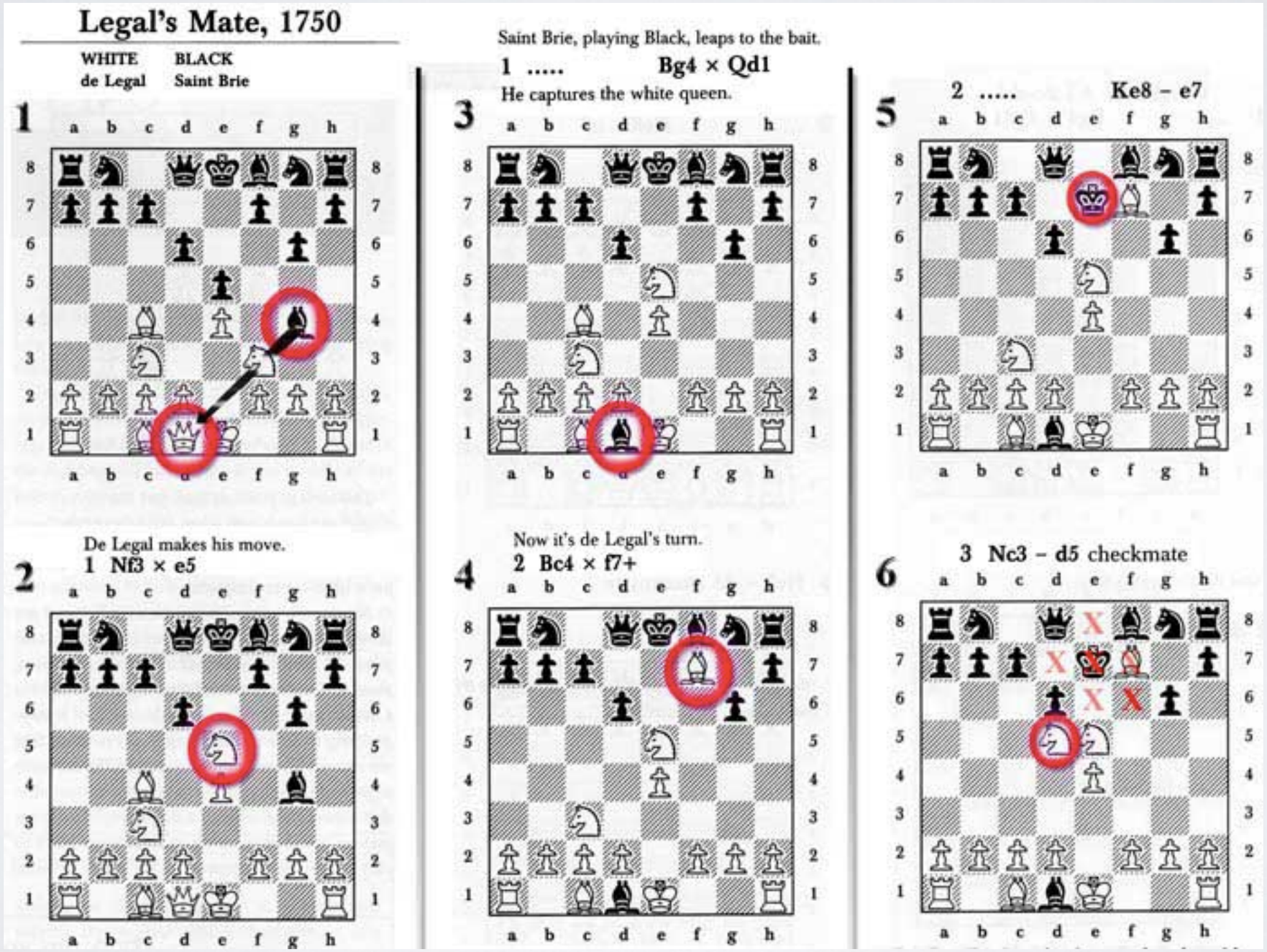


Evaluation Function



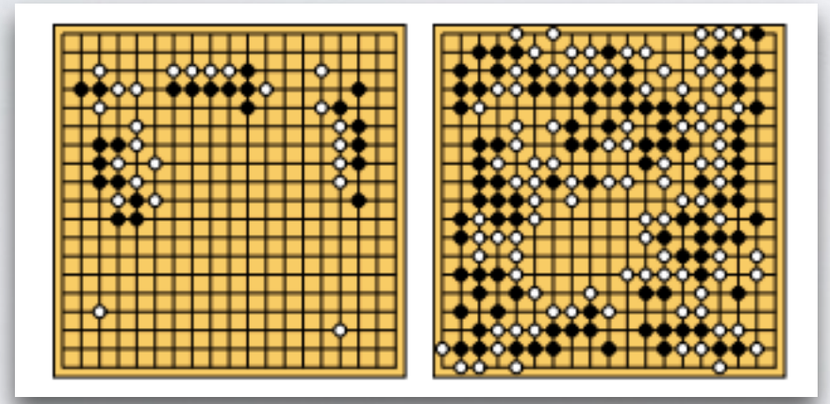
Minimax Function

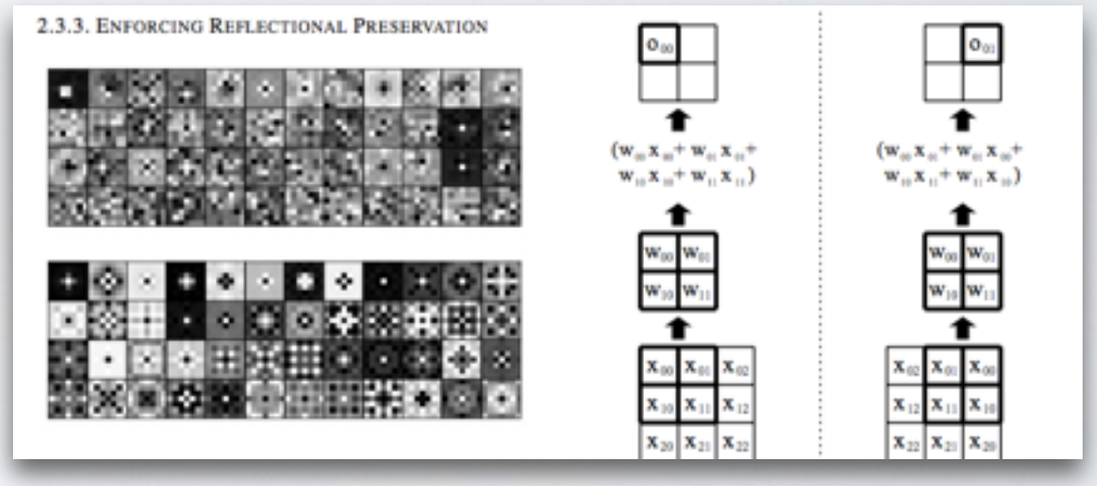
CHESS AS PATTERN RECOGNITION



USING CONVOLUTIONAL NETWORKS IN GAMES

- Based on Clark and Storkey's Teaching "Deep Convolutional Neural Networks to Play Go"

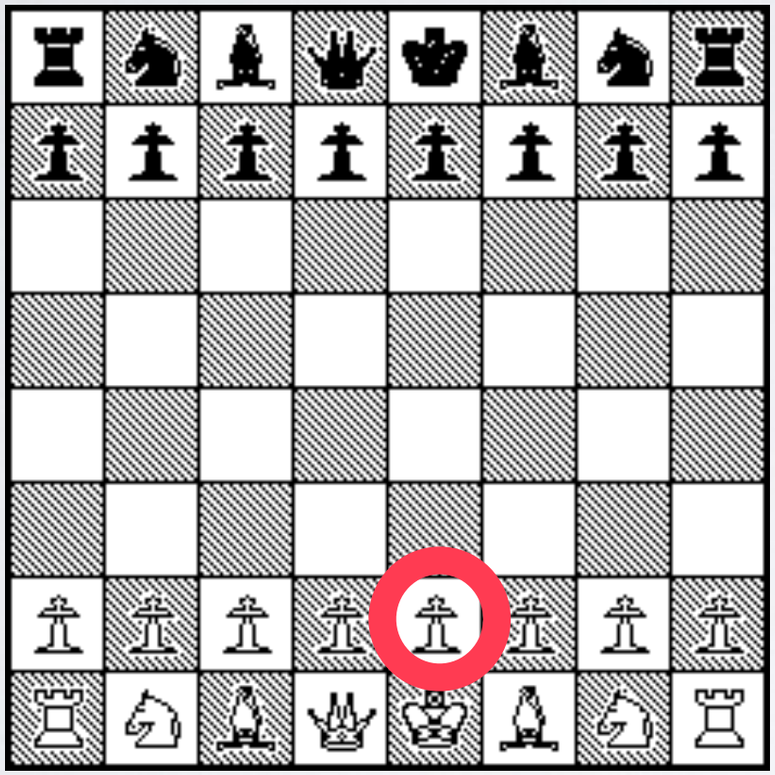




- Previous Go algorithms used evaluation functions and search algorithms like chess

- **Using pattern recognition** in Go networks were able to achieve **move prediction accuracies of 41.1% and 44.4%** on two different Go datasets, **surpassing previous state of the art machines** on this task by significant margins

APPROACH



Piece Selector

Move Selector

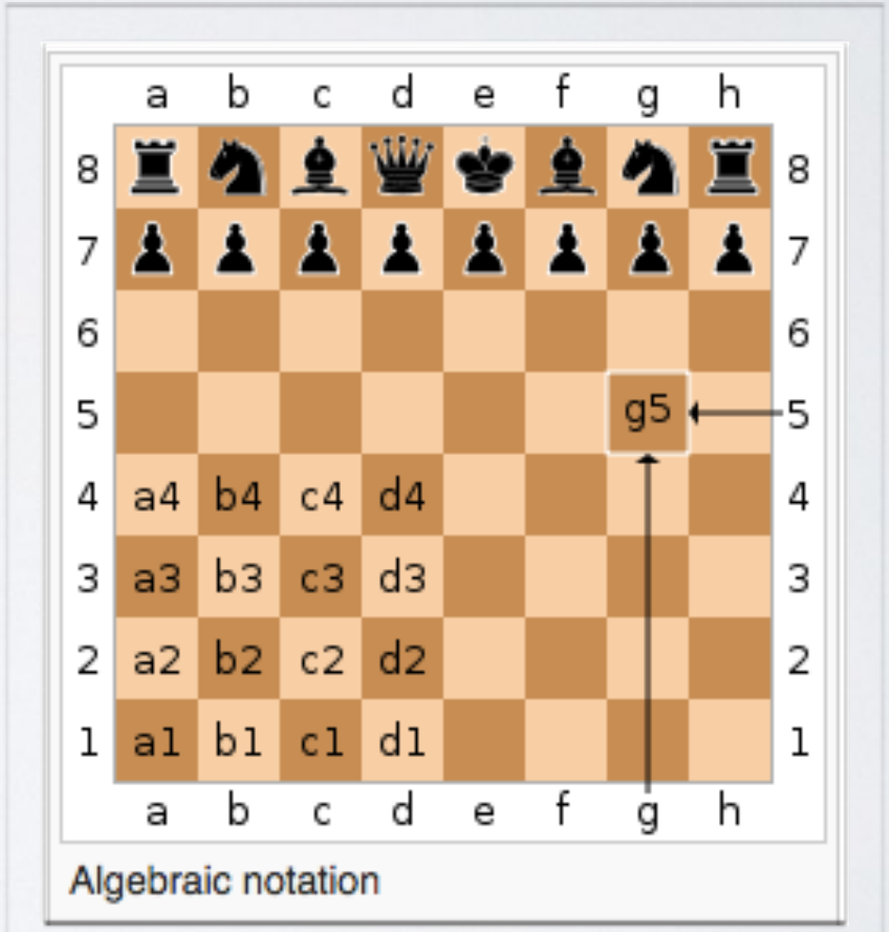
DATASET

DATASET USED WAS THE **FICS GAMES DATABASE** FOR JANUARY 2015

Data Format

1. Nf3 d5 2. d4 Nf6 3. e3 e6 4. Bd3 Be7 5. Nbd2 O-O
6. Qe2 b6 7. e4 Bb7 8. e5 Nfd7 9. h4 h6
10. Ng5 Bxg5
11. hxg5 Qxg5 12. Nf3 Qe7 13. Ng5 Rd8
14. Qf3 Nf8
15. Qh5 Nbd7 16. Nf3 f5 17. Bg5 hxg5
18. Nxg5 Qb4+
19. c3 Qxb2 20. Qh8#
{ Black checkmated } 1-0[Event "FICS rated standard game"]

SAN Notation



DATA REPRESENTATION

Input Representation

```
[[[ 0.  0.  0.  0.  0.  0.  0.  0.]
  [-1. -1. -1. -1. -1. -1. -1. -1.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 1.  1.  1.  1.  1.  1.  1.  1.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]]

 .......

 [[ 0.  0.  0.  0. -1.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  1.  0.  0.  0.]]]
```

Output Representation

62

Piece Selector

Input Representation

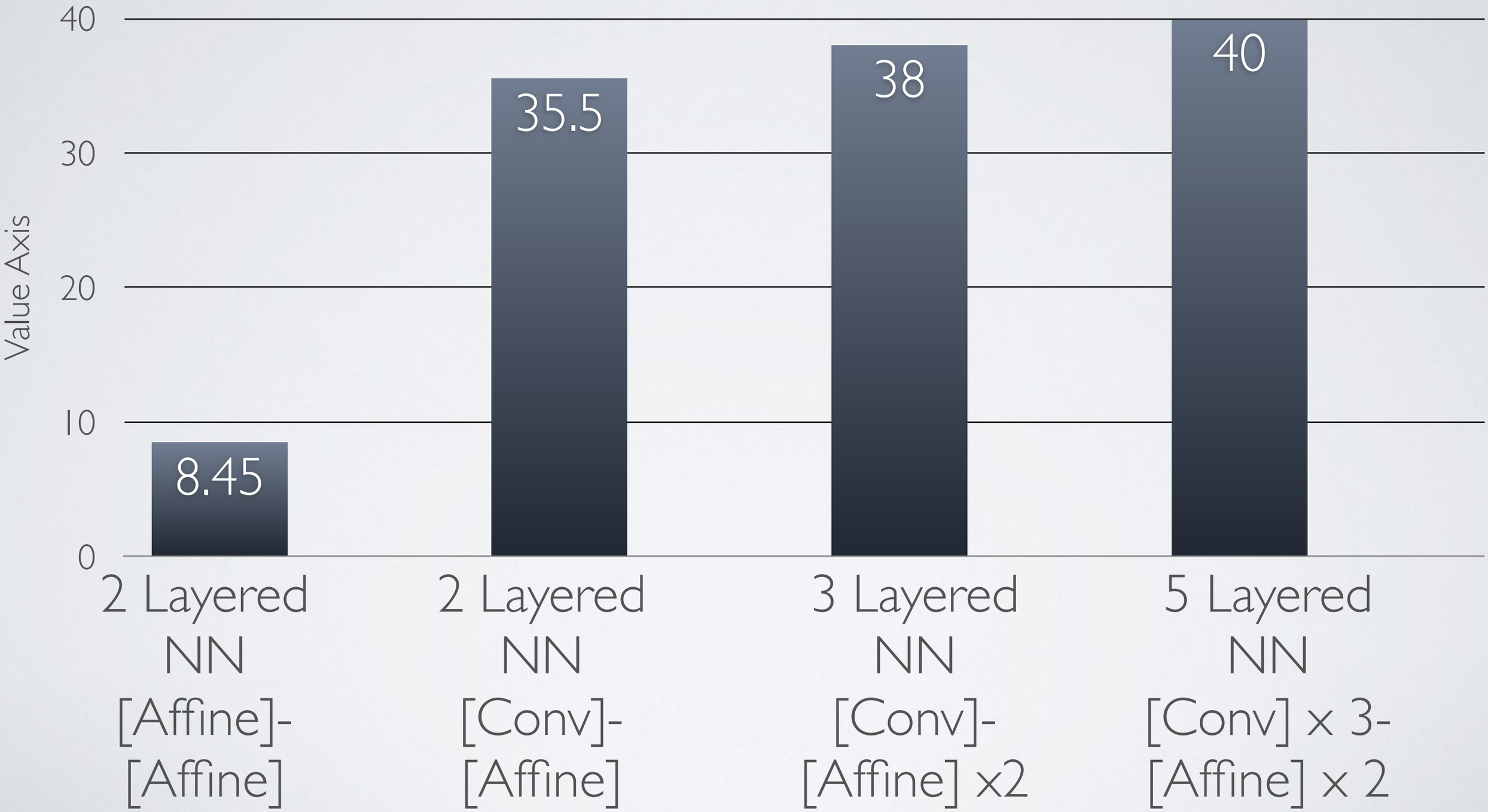
```
[[[ 0.  0.  0.  0.  0.  0.  0.  0.]
  [-1. -1. -1. -1. -1. -1. -1. -1.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 1.  1.  1.  1.  1.  1.  1.  1.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]]

 .......

 [[ 0.  0.  0.  0. -1.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.]
  [ 0.  0.  0.  0.  1.  0.  0.  0.]]]
```

Output Representation

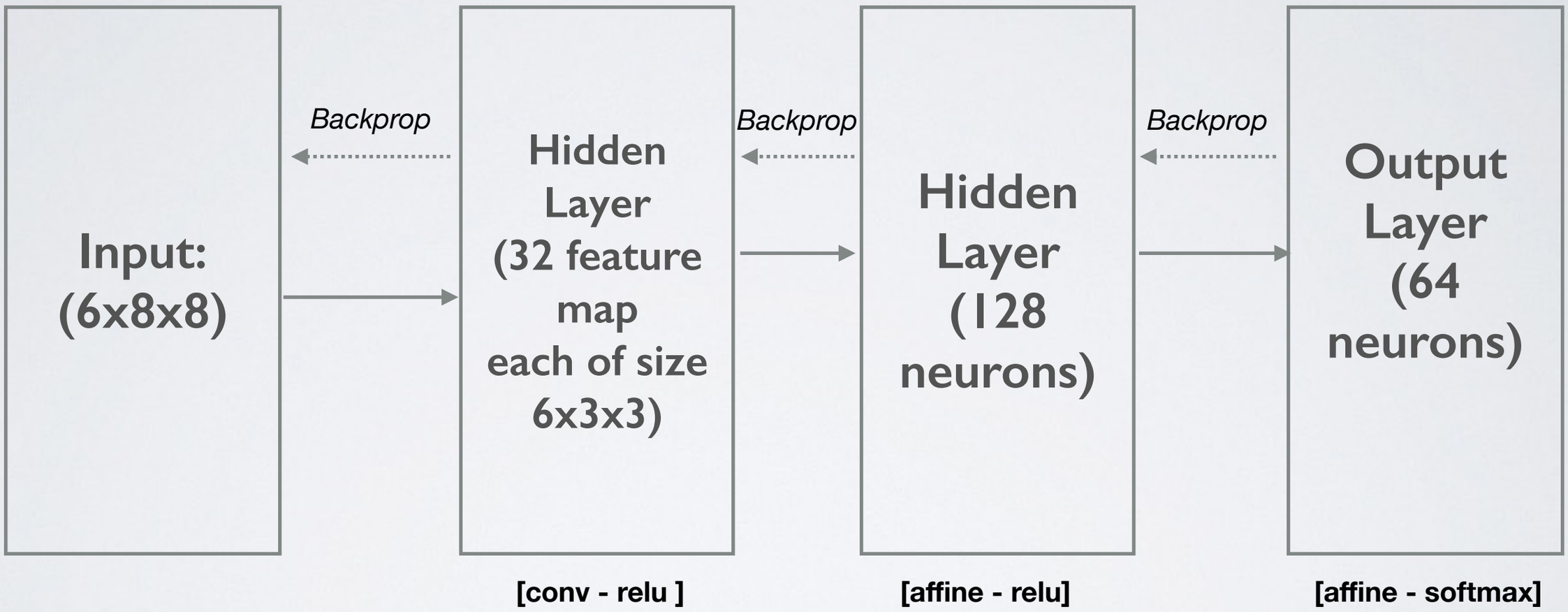
35

Move Selector

Validation Accuracy Across Networks



CNN ARCHITECTURE



Total: 41078; Training size: 32862; Testing size: 8215

SETTING UP DATA AND LOSS

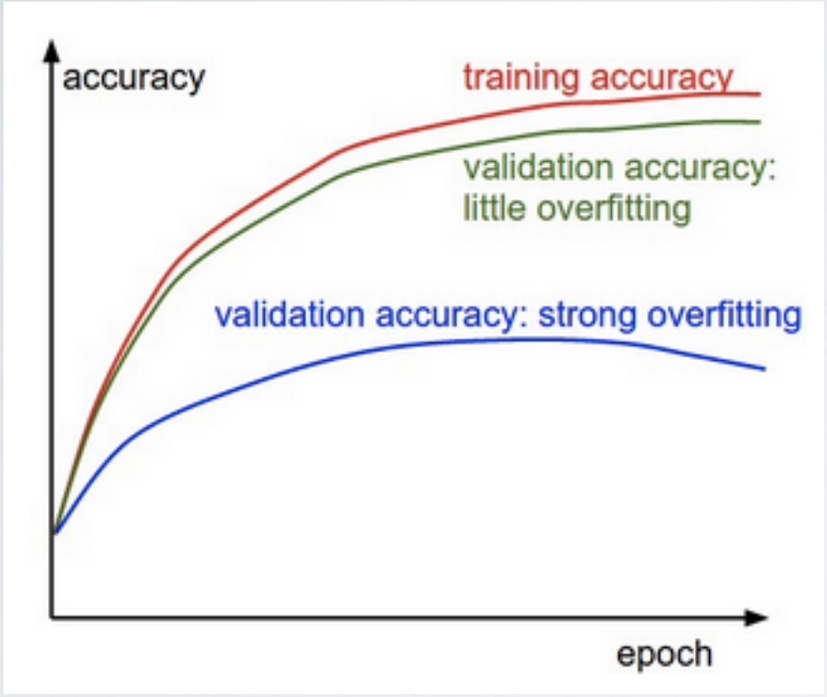
- Weight Initialization:
  - **`W = 0.001* np.random.randn(D,H)`**
- Regularization: For every weight in the network we add a small

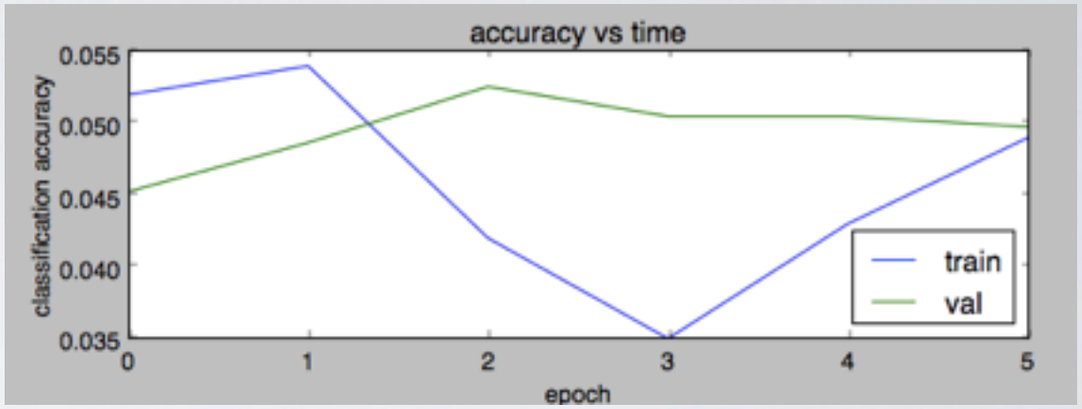
$$\min \ \mathrm{H} = \frac{1}{2m}\sum_{i=1}^{m}\left\|h(x^{(i)}) - y^{(i)}\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\left\|w^{(l)}\right\|_F^2$$

- **Loss function: Softmax**

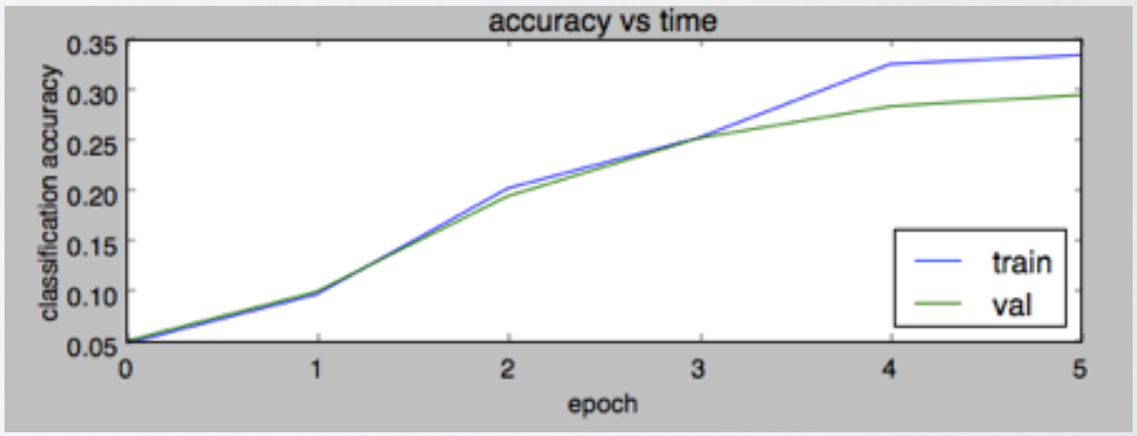
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

LEARNING PROCESS -REGULARIZATION



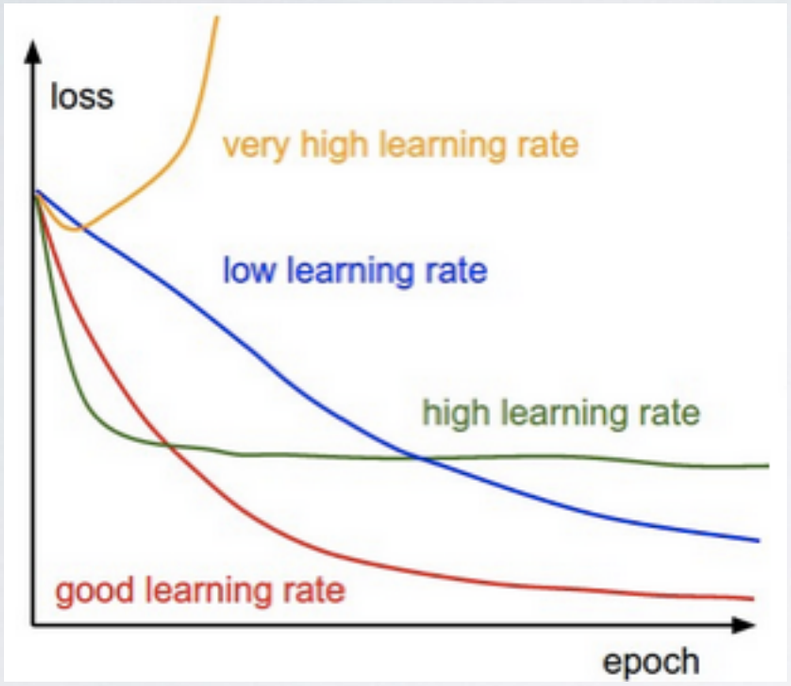


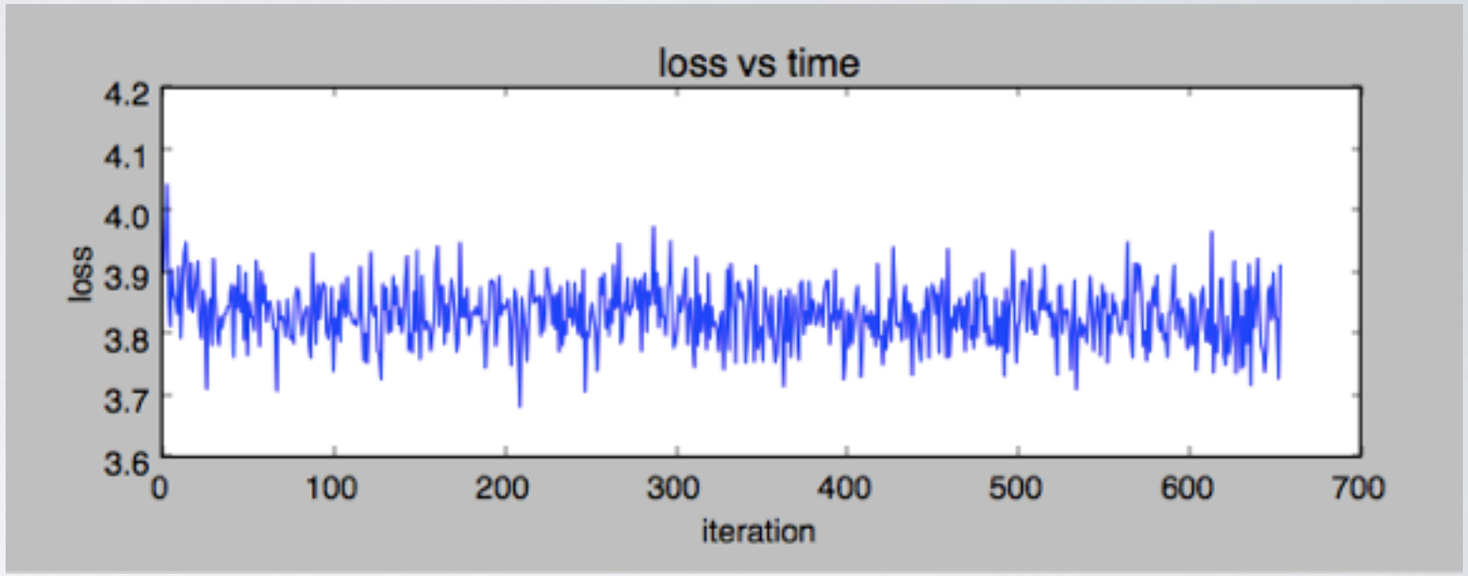
With regularization



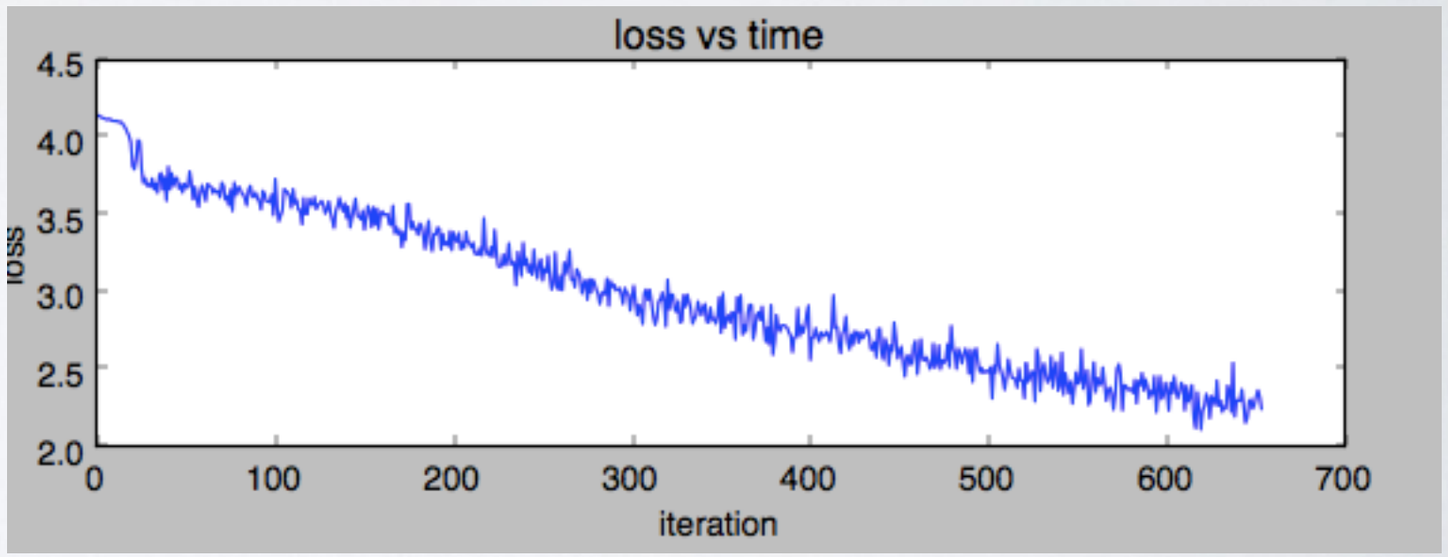
Without regularization

LEARNING PROCESS - ADJUSTING LEARNING RATE





high learning rate

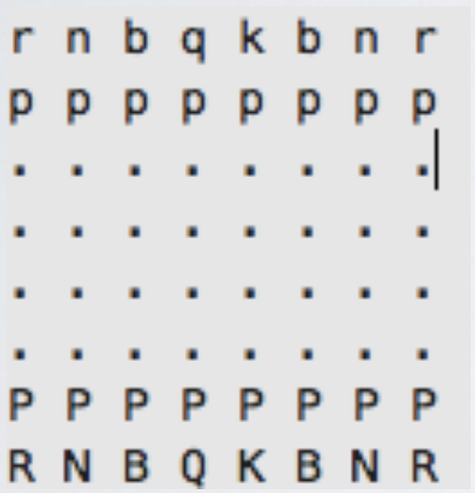


good learning rate (final model)

RESULTS

```
Piece Type:  P [[ 0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          6.96715893  0.          0.          0.          0.          0.
   0.          0.          3.43588313  0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
```

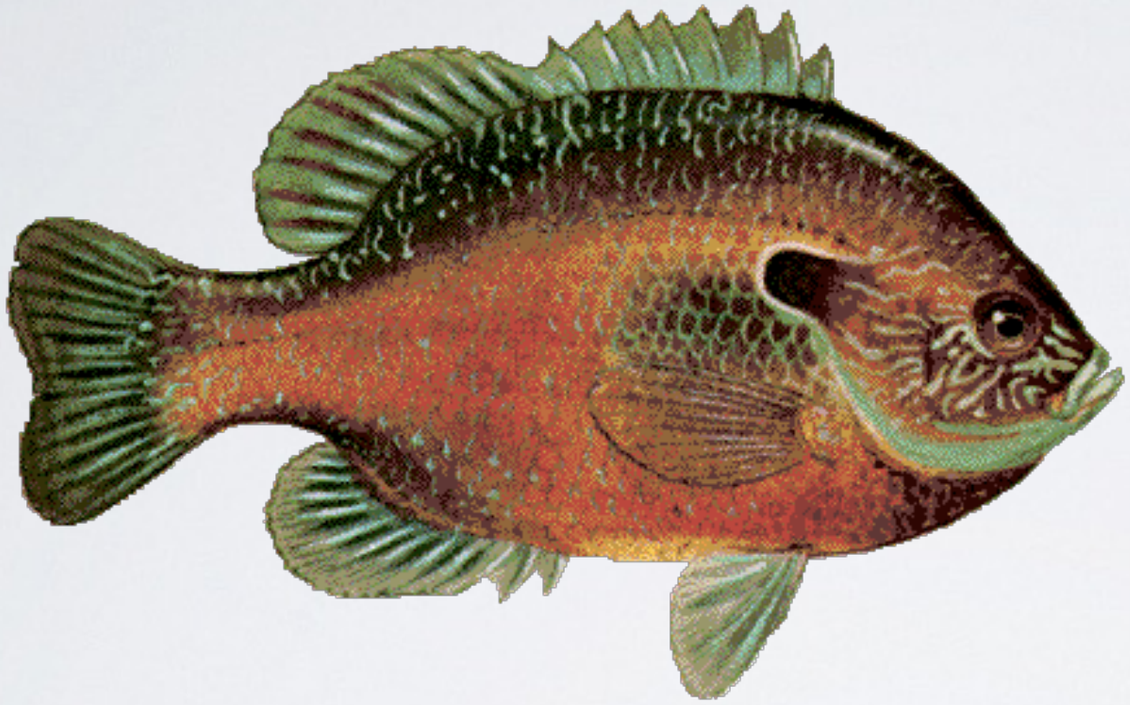
Piece Selector(-legal-enemy)




```
[[-0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.         -0.
  -0.         -0.         -0.         -0.         -0.         -0.
   1.37974401  0.94216092  3.54610081  5.61872896  4.74290027  0.84706229
   2.76923138  0.42216698  0.42847587  2.21811768  0.97981067  0.42374523
  -1.43886676  0.59443686  3.12996321 -3.25227699]]
```

Move Selector(-legal-enemy)

MYOPIC VS SUNFISH*

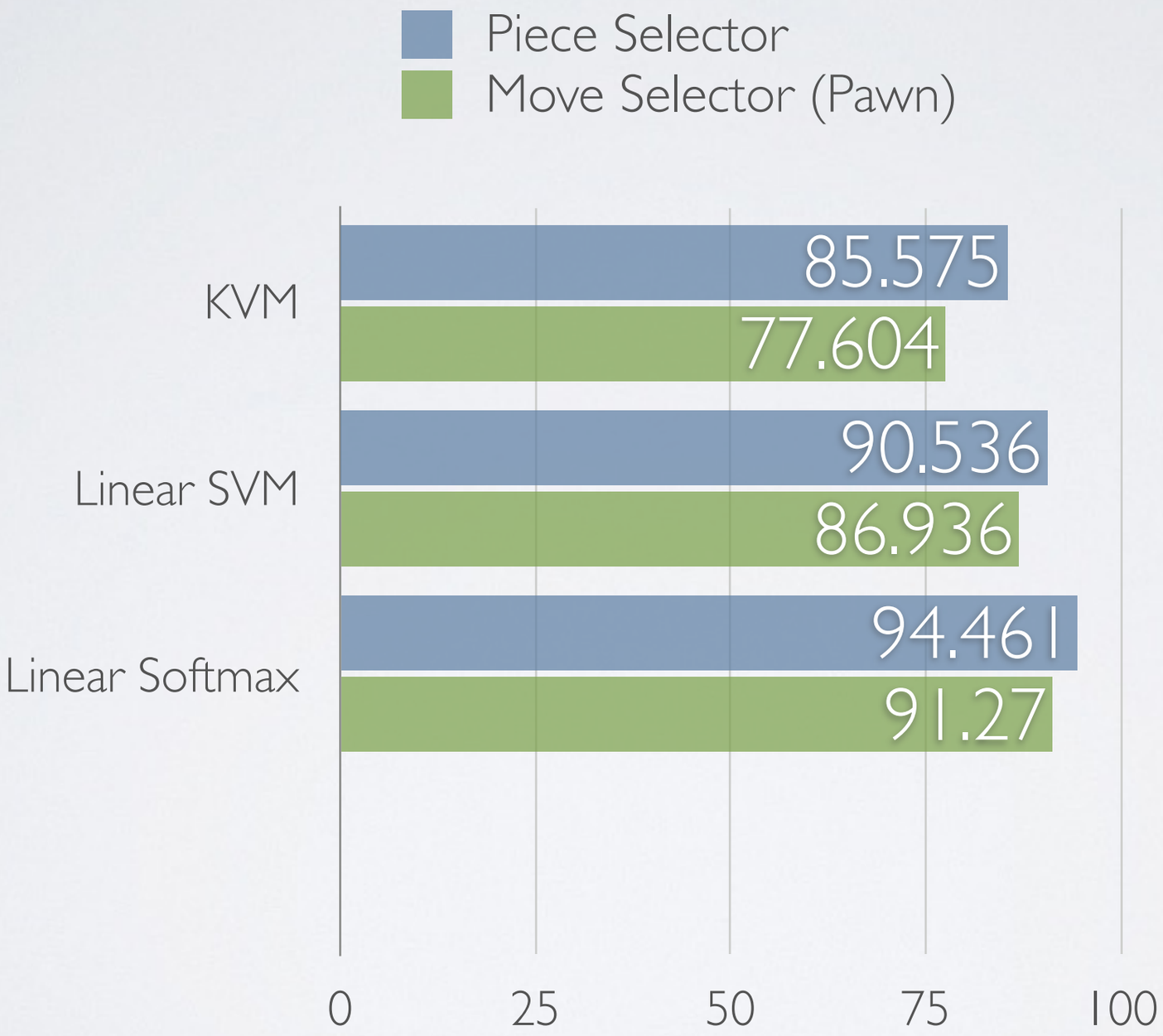


Sunfish wins 90% of the time

Others: Draw or Invalid Move

* Adapted from thomasahle's Sunfish and erikbern's DeepPink engine

OTHER METHODS



REFERENCES

1. Stanford CS Class CS231N: Convolutional Neural Networks for Visual Recognition.

2. Teaching Deep Convolutional Neural Networks to Play Go, Christopher Clark, Amos Storkey

3. Deep Learning for Chess

LIBRARIES USED

- Sunfish (https://github.com/thomasahle/sunfish)
- Stanford Convnet Library for CS231N (http://cs231n.github.io)
- Deep Pink (erikbern.com/2014/11/29/deep-learning-for-chess/)