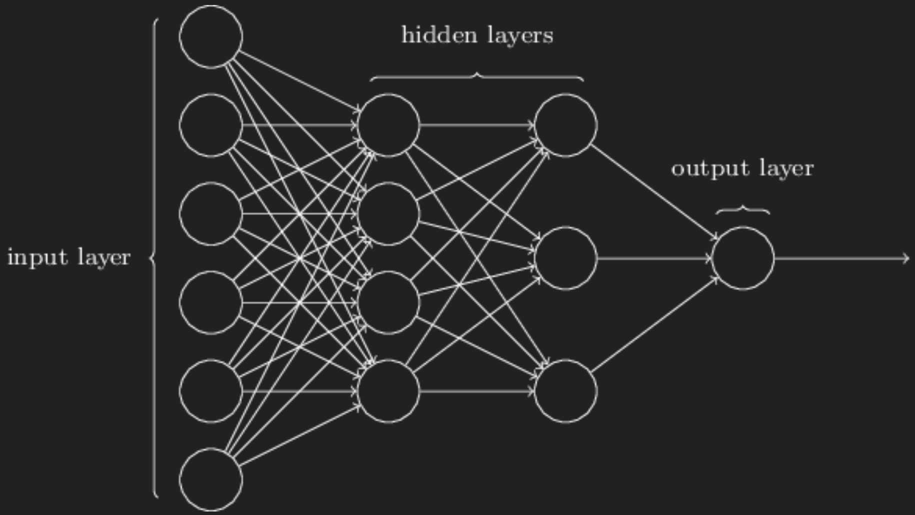
# Neural Networks

Patrick Astorga

## Background

## Architecture of Neural Networks

- Each neuron performs linear regression on its inputs.
- Then, an activation function is applied to introduce nonlinearity.
- On the right a Multi-Layer Perceptron is shown, but Neural Networks can take more complicated shapes.
    - e.g. CNN's, RNN's, LSTM (form of RNN), Transformers, etc.



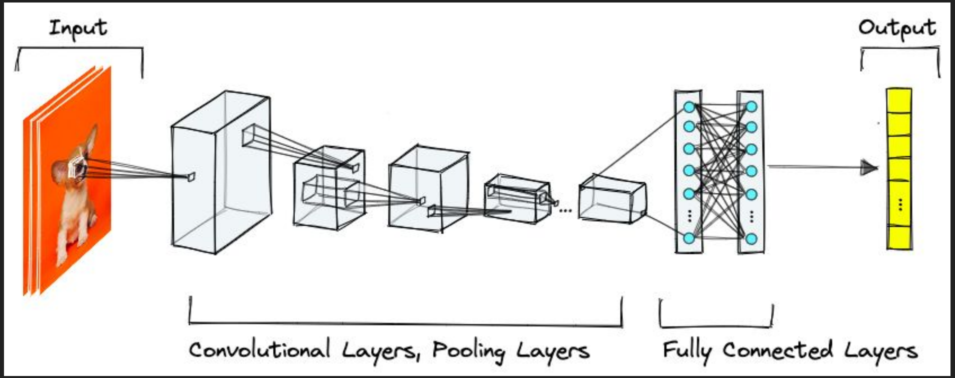
## Chess Position Evaluation

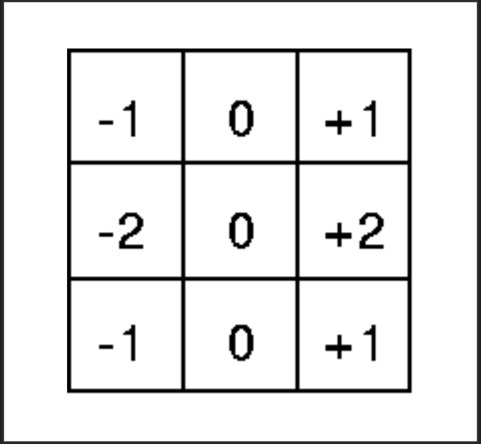
## Convolutional Neural Network (CNN)

Fully connected feedforward neural networks can be impractical for large inputs

Convolution reduces the number of free parameters while maintaining spacial awareness

Transforms feature maps using convolution kernel





| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

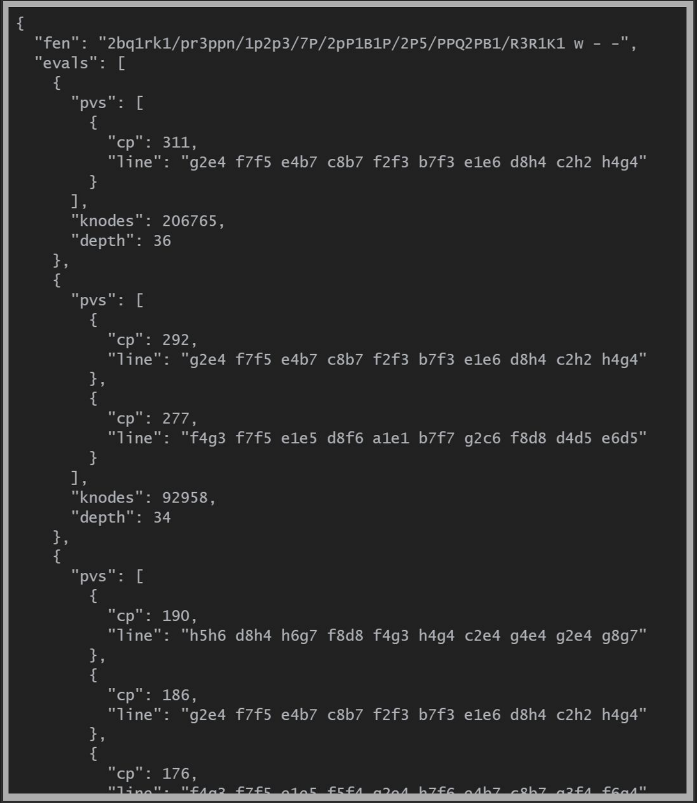
## Datasets: More data = More better

**Lichess open database evaluations:**

- Pros: Easy to get started
- Cons: Textual format and limites size

**Lc0 Training Data:**

- Pros: Billions of filtered evaluations
- Cons: Highly compressed format, required writing custom data loader in C++ to read batches of data



```
{
  "fen": "2bq1rk1/pr3ppn/1p2p3/7P/2pP1B1P/2P5/PPQ2PB1/R3R1K1 w - -",
  "evals": [
    {
      "pvs": [
        {
          "cp": 311,
          "line": "g2e4 f7f5 e4b7 c8b7 f2f3 b7f3 e1e6 d8h4 c2h2 h4g4"
        }
      ],
      "knodes": 206765,
      "depth": 36
    },
    {
      "pvs": [
        {
          "cp": 292,
          "line": "g2e4 f7f5 e4b7 c8b7 f2f3 b7f3 e1e6 d8h4 c2h2 h4g4"
        },
        {
          "cp": 277,
          "line": "f4g3 f7f5 e1e5 d8f6 a1e1 b7f7 g2c6 f8d8 d4d5 e6d5"
        }
      ],
      "knodes": 92958,
      "depth": 34
    },
    {
      "pvs": [
        {
          "cp": 190,
          "line": "h5h6 d8h4 h6g7 f8d8 f4g3 h4g4 c2e4 g4e4 g2e4 g8g7"
        },
        {
          "cp": 186,
          "line": "g2e4 f7f5 e4b7 c8b7 f2f3 b7f3 e1e6 d8h4 c2h2 h4g4"
        },
        {
          "cp": 176,
          "line": "f4g3 f7f5 e1e5 f5f4 g2e4 b7f6 e4b7 c8b7 g3f4 f6g4"
```

Lichess Open Database Evaluations

## Challenge: Lc0 Training Data

**Managing the large, highly compressed, training data format was a challenge:**

- Wrote a C++ library to load batched data from file
- File stores varying length position chains, reading data is sequential by nature

**Stochastic Gradient Descent:**

- Stochastic gradient descent is more stable when randomness is incorporated into the training (don't just iterate in the same order every time)
- Added “drop”: chance that data loader skips over a position in the chain

## Centipawn Eval / Result Interpolation

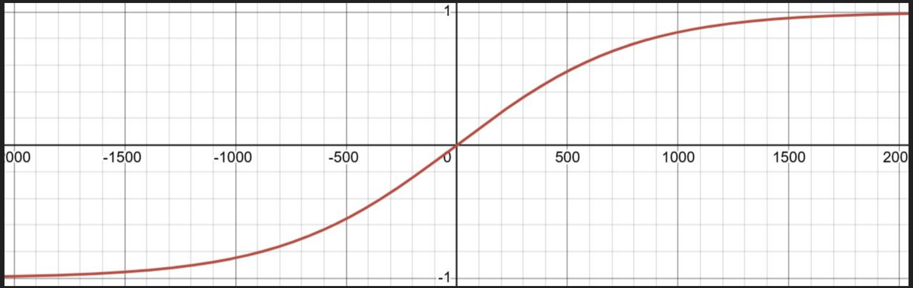
**The output layer of the network is linear function giving the eval (centipawn)**

- Directly applying the loss to this yields less stability (much higher gradients)

**I applied a sigmoid like curve (or tanh) to the output**

- This effectively squashes the extreme values
- This also allows for interpolation with game results (-1, 0, 1)

$y = \tanh(x / 800)$



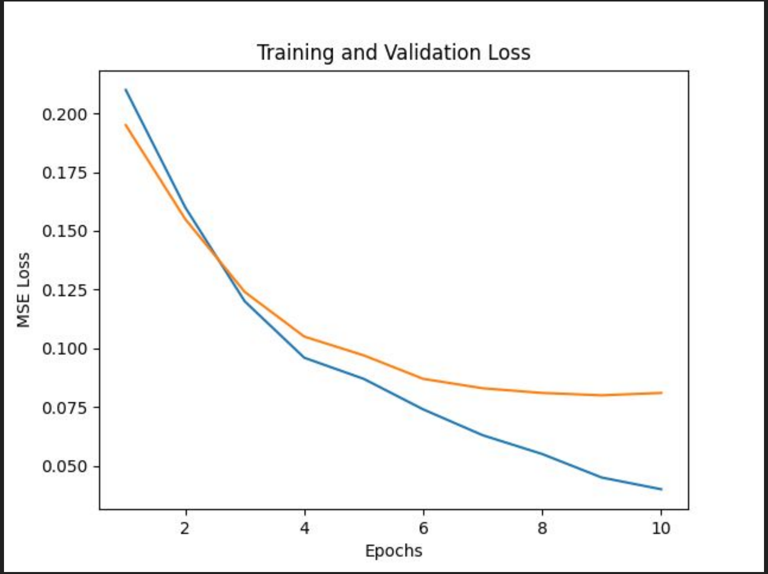
## Model Architecture and Results

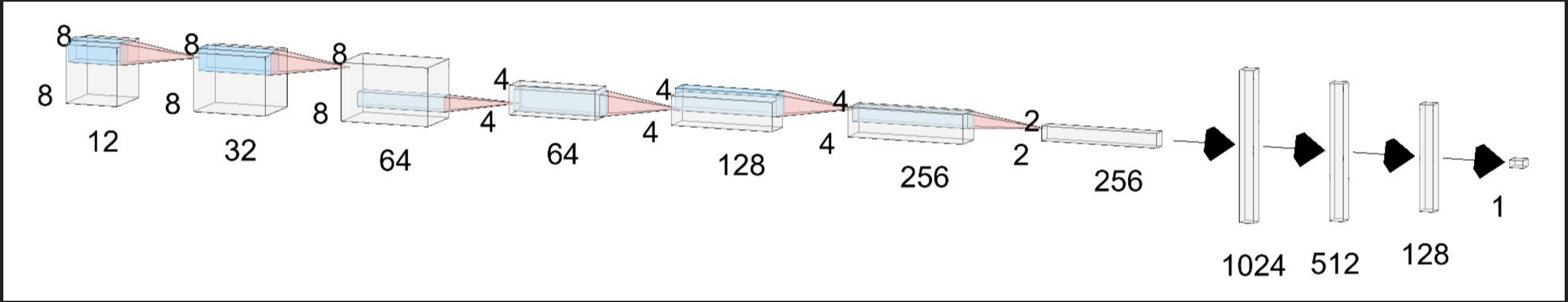
4 Convolutional layers with ReLU activations

2 Max pooling layers

3 Fully connected layers with ReLU activations

8 x 8 x 12 input (rank, file, piece type) → evaluation

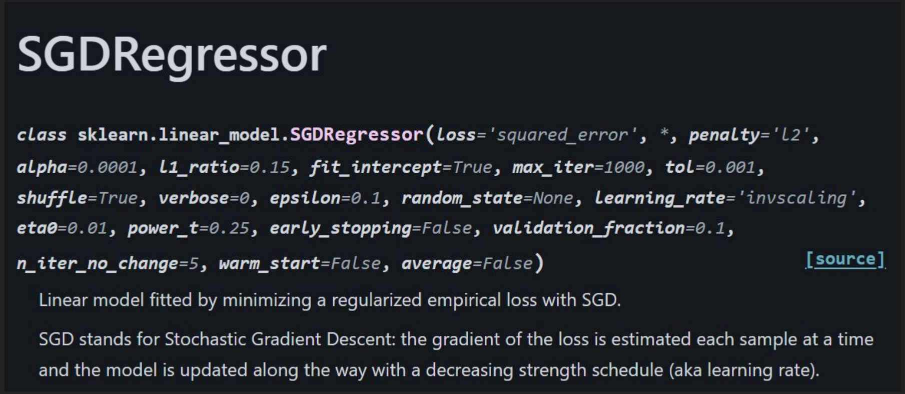




## Linear Regression: Stochastic Gradient Descent

My dataset is too large to perform explicit linear regression.

- Iterate over the dataset in batches, and partially fit the model for each batch
- Repeat with a diminishing learning rate
- scikit learn package continues training until validation loss no longer increases



### SGDRegressor

```
class sklearn.linear_model.SGDRegressor(loss='squared_error', *, penalty='l2',
alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001,
shuffle=True, verbose=0, epsilon=0.1, random_state=None, learning_rate='invscaling',
eta0=0.01, power_t=0.25, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, warm_start=False, average=False)
```

[source]

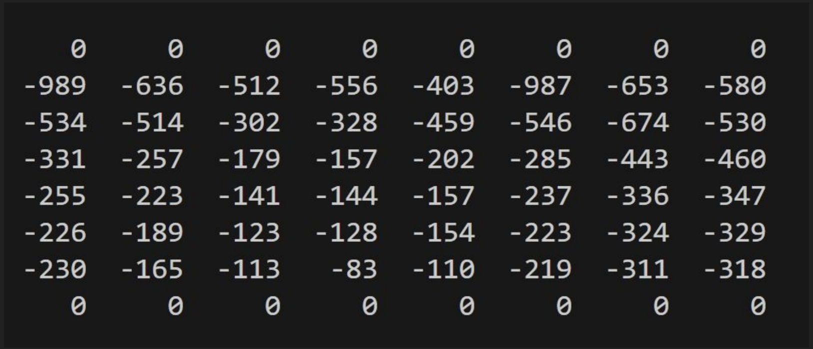
Linear model fitted by minimizing a regularized empirical loss with SGD.

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

```
# Initialize the LinearRegression model
model = SGDRegressor(
    loss='squared_error',
    penalty=None,
    alpha=0.0001,
    fit_intercept=False,
    max_iter=1000,
    tol=1e-3,
    learning_rate='invscaling',
    early_stopping=True
)
```

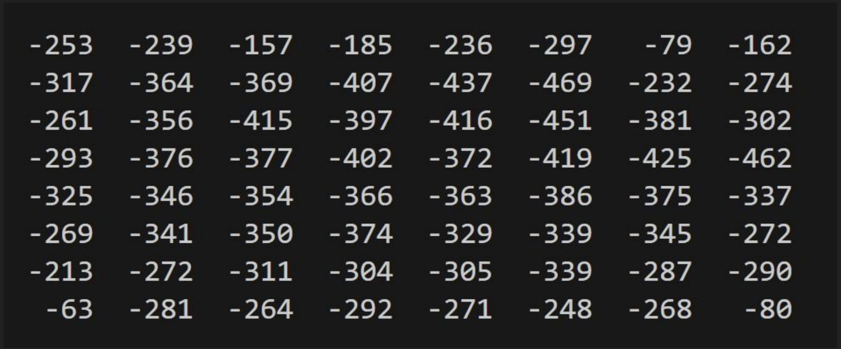
# Linear Regression Results: Maximum Interpretability

## Black Pawns



```
   0     0     0     0     0     0     0     0
-989  -636  -512  -556  -403  -987  -653  -580
-534  -514  -302  -328  -459  -546  -674  -530
-331  -257  -179  -157  -202  -285  -443  -460
-255  -223  -141  -144  -157  -237  -336  -347
-226  -189  -123  -128  -154  -223  -324  -329
-230  -165  -113   -83  -110  -219  -311  -318
   0     0     0     0     0     0     0     0
```

## Black Knights



```
-253  -239  -157  -185  -236  -297   -79  -162
-317  -364  -369  -407  -437  -469  -232  -274
-261  -356  -415  -397  -416  -451  -381  -302
-293  -376  -377  -402  -372  -419  -425  -462
-325  -346  -354  -366  -363  -386  -375  -337
-269  -341  -350  -374  -329  -339  -345  -272
-213  -272  -311  -304  -305  -339  -287  -290
 -63  -281  -264  -292  -271  -248  -268   -80
```

Linear regression coefficients gives the exact contribution each piece makes on a specific square!