Final Presentation

COMP 6630
Matthew Freestone, Will Humphlett, Matthew Shiplett

Problem Description

- Sentiment Classification:
  - Classify a body of text as one of a set of sentiments
    - Classify text from Amazon reviews into their overall star rating

- Possible Applications:
  - Allow for analysis of large corpus of text quickly

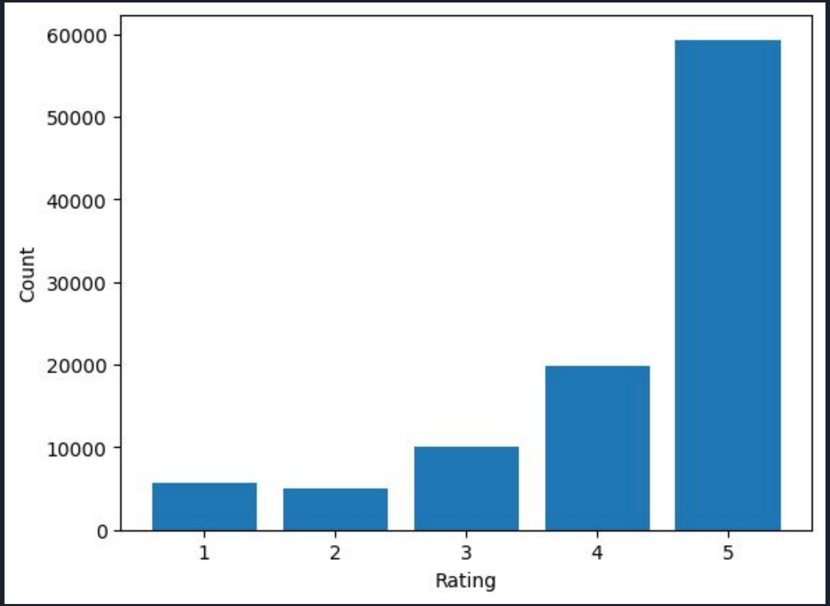
- Dataset Problems:
  - IMDb movie dataset did not have neutral reviews
  - Twitter dataset claimed to have neutral tweets but did not
  - Amazon reviews skew heavily towards 5 star reviews.

Dataset Change

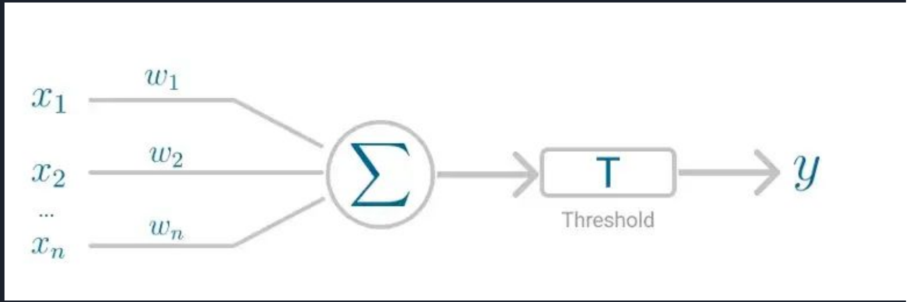
- No Neutral Reviews

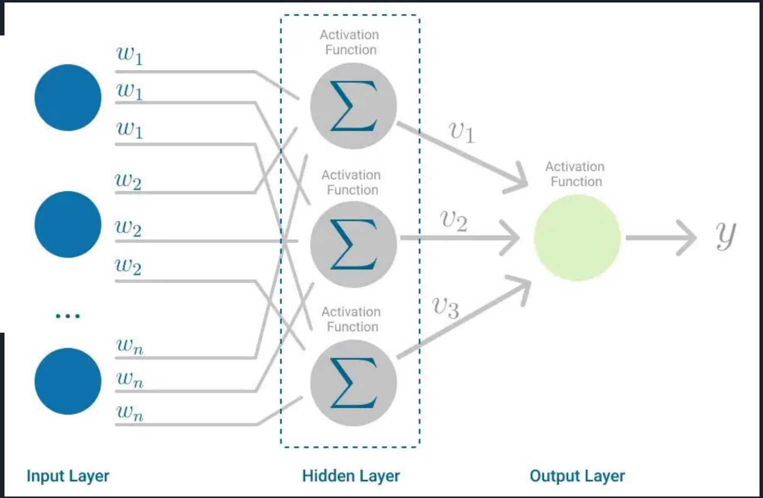
https://nijianmo.github.io/amazon/index.html

- Unbalanced Dataset



Multilayer Perceptron Outline





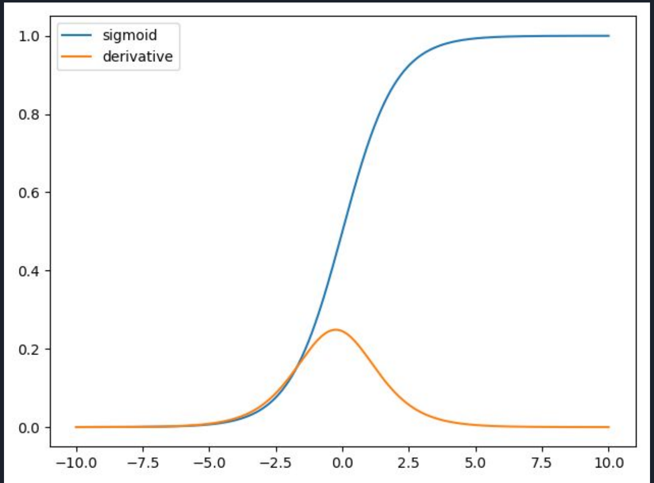
Core Mathematical Ideas

$$\frac{\partial C}{\partial w^{[3]}} = \frac{\partial C}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial w^{[3]}}$$

$$\frac{\partial C}{\partial b^{[3]}} = \frac{\partial C}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial b^{[3]}}$$

Partial Derivatives

Sigmoid Function



Forward Pass Theory and Implementation

```
def _fast_forward_pass(self, X: np.ndarray) -> np.ndarray:
    curr = X
    for i in range(self._num_layers - 1):
        curr = curr @ self._weights[i]
        curr += self._biases[i].T
        curr = self.activation(curr) if i < self._num_layers - 2 else self._output_activation(curr)
    return curr
```

X: (n_example, n_features)

At least layer, W is (prev_layer_size, next_layer_size)

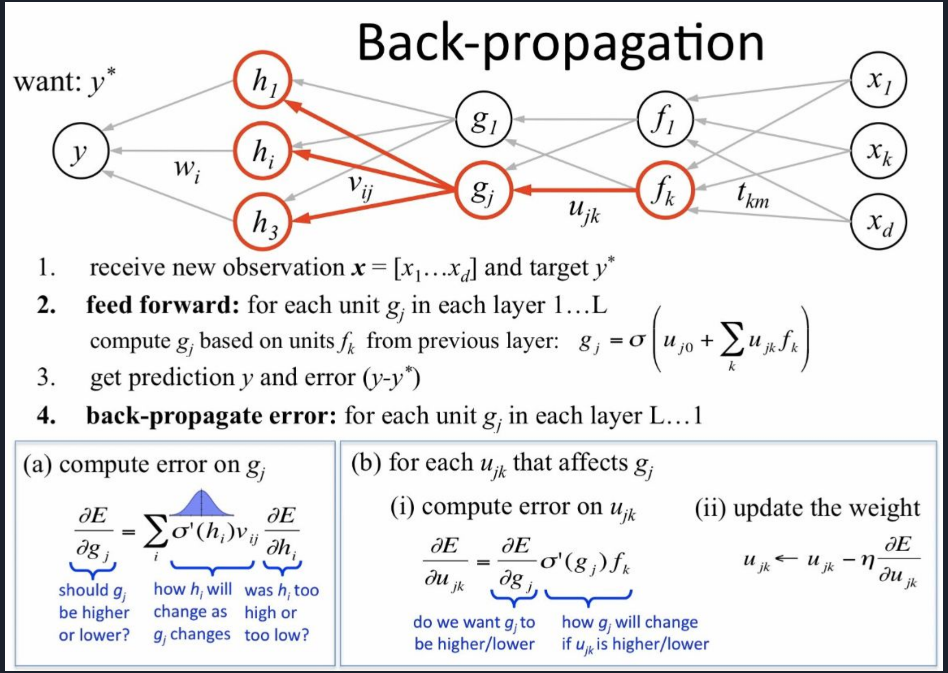
So curr is (n_examples, next_layer_size)

At last step, apply activation (softmax)

Backpropagation and Gradient Descent Math

Each layer receives a 'delta' from the one after it. We first use y_true to determine how different our prediction is from the actual value

After the change is computed,
It is applied proportional to learning rate



Backpropagation Implementation

```python
def _backprop(self, X: np.ndarray, y: np.ndarray) -> tuple[list[np.ndarray], list[np.ndarray]]:
    dBias = [np.zeros(b.shape) for b in self._biases]
    dWeights = [np.zeros(w.shape) for w in self._weights]
    n_samples = X.shape[0]

    # do forward pass, store all activations and net values
    layer_raw = []
    layer_activations = []
    a = X
    for i, (b, W) in enumerate(zip(self._biases, self._weights)):
        z = a @ W + b.T
        # z = self._safe_sparse_dot(a,W) + b.T
        a = self.activation(z) if i < self._num_layers - 2 else self._output_activation(z)
        layer_raw.append(z)
        layer_activations.append(a)
```

Backpropagation Implementation

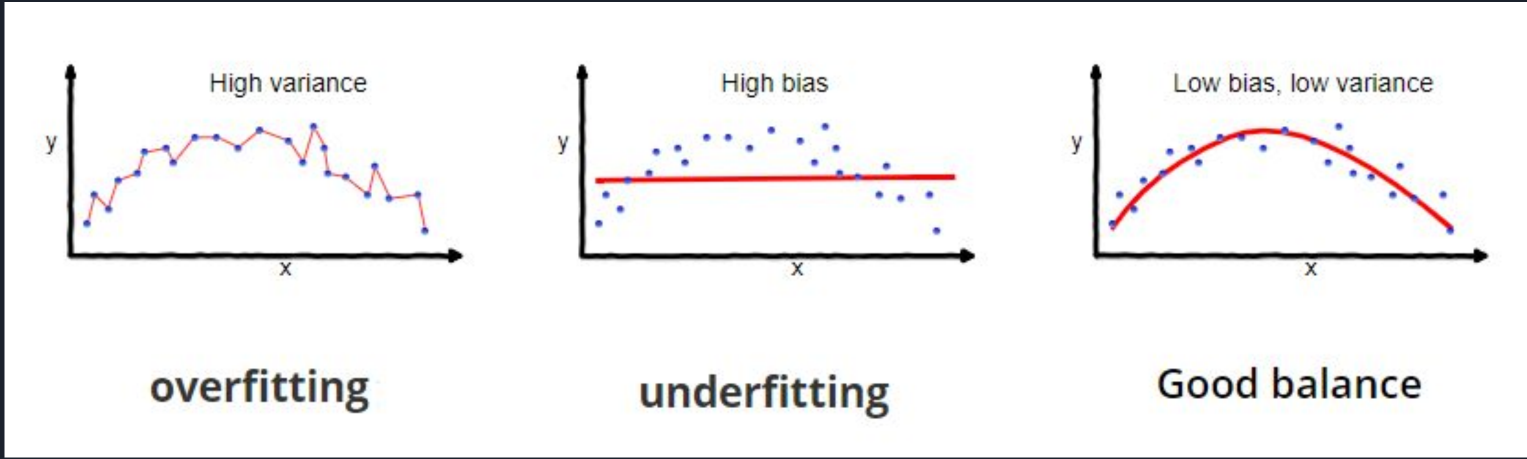
(.2, .1, .2, .1, .4)   (0, 0, 0, 0, 1)

```
delta = (layer_activations[last_hidden] - y) * self.dActivation(layer_raw[last_hidden])
dBias[last_hidden] = np.mean(delta, axis=0)
dWeights[last_hidden] = layer_activations[last_hidden-1].T @ delta

# For all hidden layers, compare to layer after it
for L in range(last_hidden-1, 0, -1):
    delta = (delta @ self._weights[L+1].T) * self.dActivation(layer_raw[L])
    dBias[L] = np.mean(delta, axis=0)
    dWeights[L] = layer_activations[L-1].T @ delta


# For input layer, update W according to input, not previous layer
delta = (delta @ self._weights[1].T) * self.dActivation(layer_raw[0])
dBias[0] = np.mean(delta, axis=0)
dWeights[0] = X.T @ delta
```

Regularization, L1 and L2

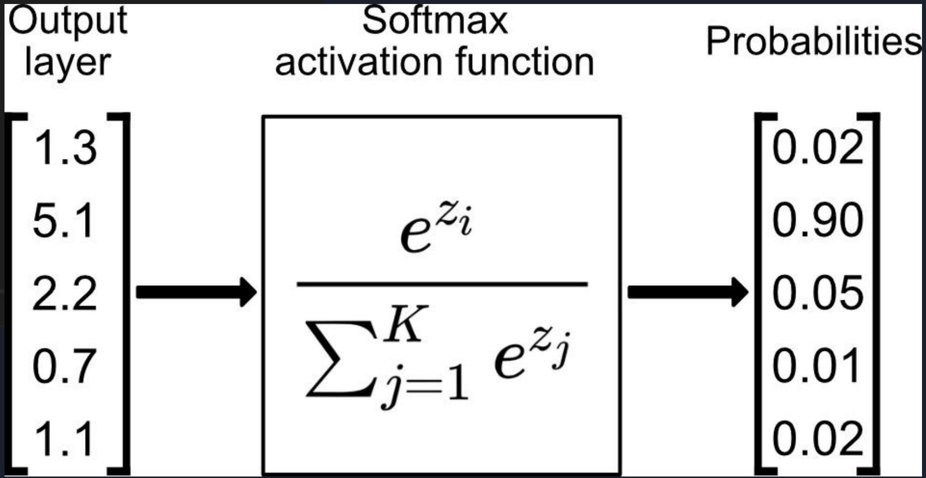



```
# add a second dimension to the bias gradient to make it compatible with the bias shape
dBias = [db[:,np.newaxis] for db in dBias]
# divide by number of samples to get the average gradient
dWeights = [dw/n_samples for dw in dWeights]
if self.regularization:
    dWeights = [dw + self.reg_const * r for dw, r in zip(dWeights, self._grad_reg(self._weights))]
return (dBias, dWeights, loss)
```

Representing Multiple Classes

```python
def _format_labels(self, y: np.ndarray) -> np.ndarray:
    if len(y.shape) == 2 and y.shape[1] == 1:
        y = y.ravel()
    elif len(y.shape) == 2 and y.shape[1] > 1 or len(y.shape) > 2:
        raise ValueError("Invalid shape for y")

    self._yencoder.fit(y)
    if len(self._yencoder.classes_) == 2:
        self._output_activation = sigmoid
    else:
        self._output_activation = softmax
    return self._yencoder.transform(y)
```



MLP Fit

```
for epoch_num, lr in tqdm(self.epochs(), total=self.num_epochs):
    train_loss = 0
    for i in range(0, X.shape[0], batch_size):
        X_batch = X[i:i+batch_size]
        y_batch = y[i:i+batch_size]
        dJdB, dJdW, c_loss = self._backprop(X_batch, y_batch)
        train_loss += c_loss

        self._biases = [b - lr * db for b, db in zip(self._biases, dJdB)]
        self._weights = [w - lr * dw for w, dw in zip(self._weights, dJdW)]
    num_batches = X.shape[0] // batch_size
    self.train_loss_curve.append(train_loss / num_batches)
    if use_val:
        val_loss = self._calc_loss(y_val, self._fast_forward_pass(X_val))
        self.val_loss_curve.append(val_loss)
```

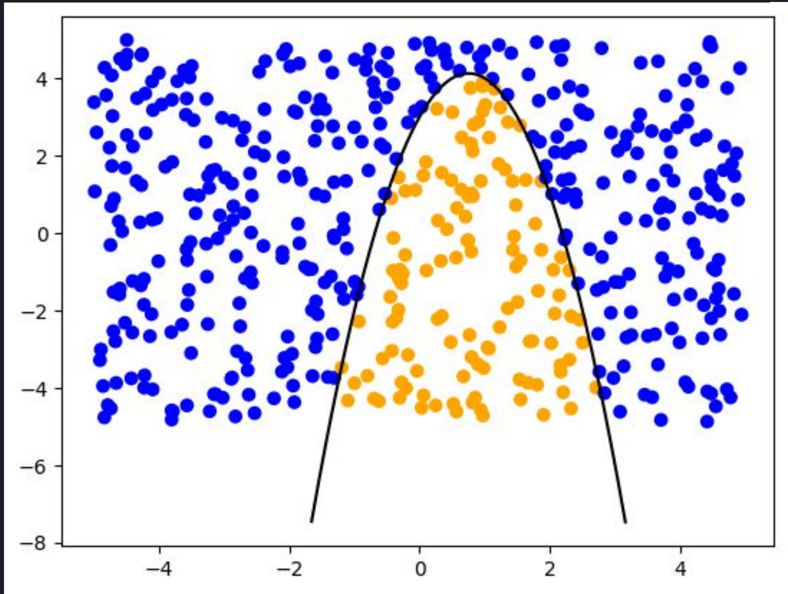
Proving Functionality

```
p = TwoDimProblem(value_range=5)
X, y = p.createData(soln_rank=2, noise_frac=0, samples=500)
```

Custom class to create a visualizable problem

Useful to ensuring that fit actually occurs

Non-linear planes (quadratic)



Proving Functionality

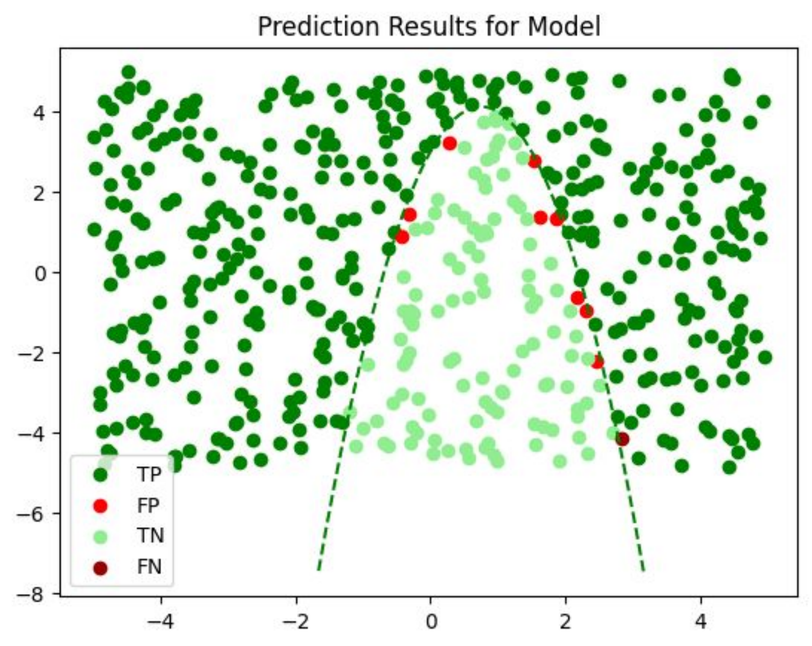
```python
from mlp import MultiLayerPerceptron

mlp = MultiLayerPerceptron(
    epochs=150,
    lr=0.1,
    activation='sigmoid',
    hidden_layers=[5],
)
mlp.fit(X, y, batch_size=10)
print(p.plotPred(mlp.predict(X)))
```

[8] ✓ 0.9s

```
100%|██████████| 150/150 [00:00<00:00, 231.84it/s]

Accuracy = 0.98
```



Using the code

Live Demo

Preprocessing

Count Vectorizer - Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
from preprocessing import get_subsets

word_vectorizer = CountVectorizer(
    min_df=0.001,
    max_df=0.7
)
X = df["reviewText"].to_numpy()
X = word_vectorizer.fit_transform(X)
y = df['overall'].to_numpy()

X_train, X_val, X_test, y_train, y_val, y_test = get_subsets(X,y, train_split=0.8, val_split=0.1, test_split=0.1)
```

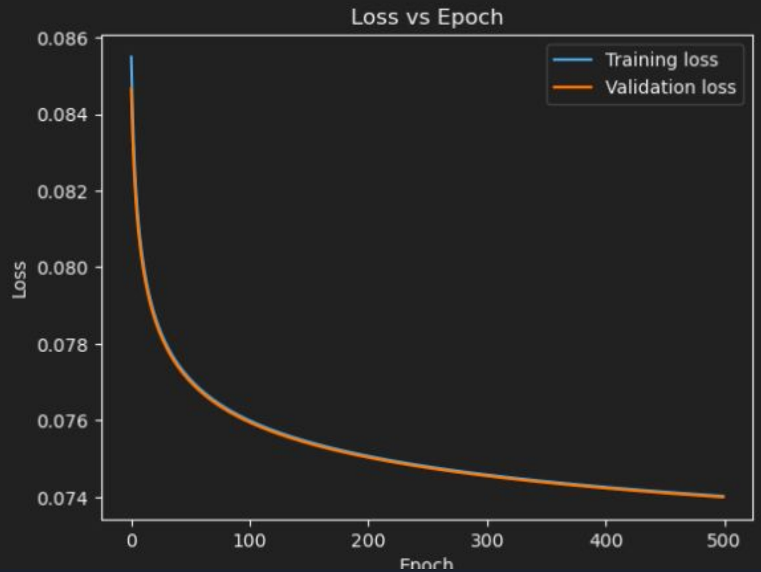
✓ 6.2s

Results - Our implementation on Unbalanced Dataset

```
Training accuracy: 0.43961161273790045
Testing accuracy: 0.44094984506992474
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	10495
2	0.07	0.00	0.00	10286
3	0.09	0.13	0.11	20119
4	0.26	0.05	0.09	38161
5	0.53	0.77	0.63	90693
accuracy			0.44	169754
macro avg	0.19	0.19	0.16	169754
weighted avg	0.35	0.44	0.37	169754

```
100%|██████████| 500/500 [1:55:46<00:00, 13.89s/it]
```



Benchmarking

Tensorflow, Dense Layers with Vectorization built in - **66.54**

```
pred = model.evaluate(X_test, y_test)
```

[13] ✓ 13.9s

```
3125/3125 [==============================] - 14s 4ms/step - loss: 1.0896 - accuracy: 0.6654 - mse: 10.5934
```

Sklearn, Random Forest -

Sklearn, MLPClassifier - **67.37**

```
Training accuracy: 0.7097509979091428
Testing accuracy: 0.6737895158063225
```

	precision	recall	f1-score	support
1	0.53	0.51	0.52	699
2	0.40	0.00	0.01	570
3	0.44	0.41	0.43	1310
4	0.46	0.28	0.35	2456
5	0.75	0.92	0.83	7460
accuracy			0.67	12495
macro avg	0.52	0.42	0.43	12495
weighted avg	0.63	0.67	0.63	12495

Running Challenges

Non-gpu optimized, can’t train on 1 million point dataset due to time constraints

Numpy arrays do not work on data this big, but scipy sparse does

```
MemoryError: Unable to allocate 429. GiB for an array with shape (3410019,) and data type <U33759
```

Dataset is unbalanced, more than 50% 5 stars

Making interfaces intuitive and useable between developers

Future Work

Numba, a JIT compiler for python. Allows CUDA GPU runs

Variable Learning Rate

Pulling data into memory incrementally

Embedding using Glove, Bert, Word2vec etc

Finishing SkLearn API to make our models usable in all their functions

Manually Balancing the dataset

Tests on performance for non-movie entries

Lessons Learned

Batching backprop calc leads to much better performance at little effect to how well it works

Variable Learning Rates and optimizers, like Adam, greatly improve performance

Imbalanced datasets are difficult to use

Use tensorflow - Open source good

Don’t reinvent the wheel