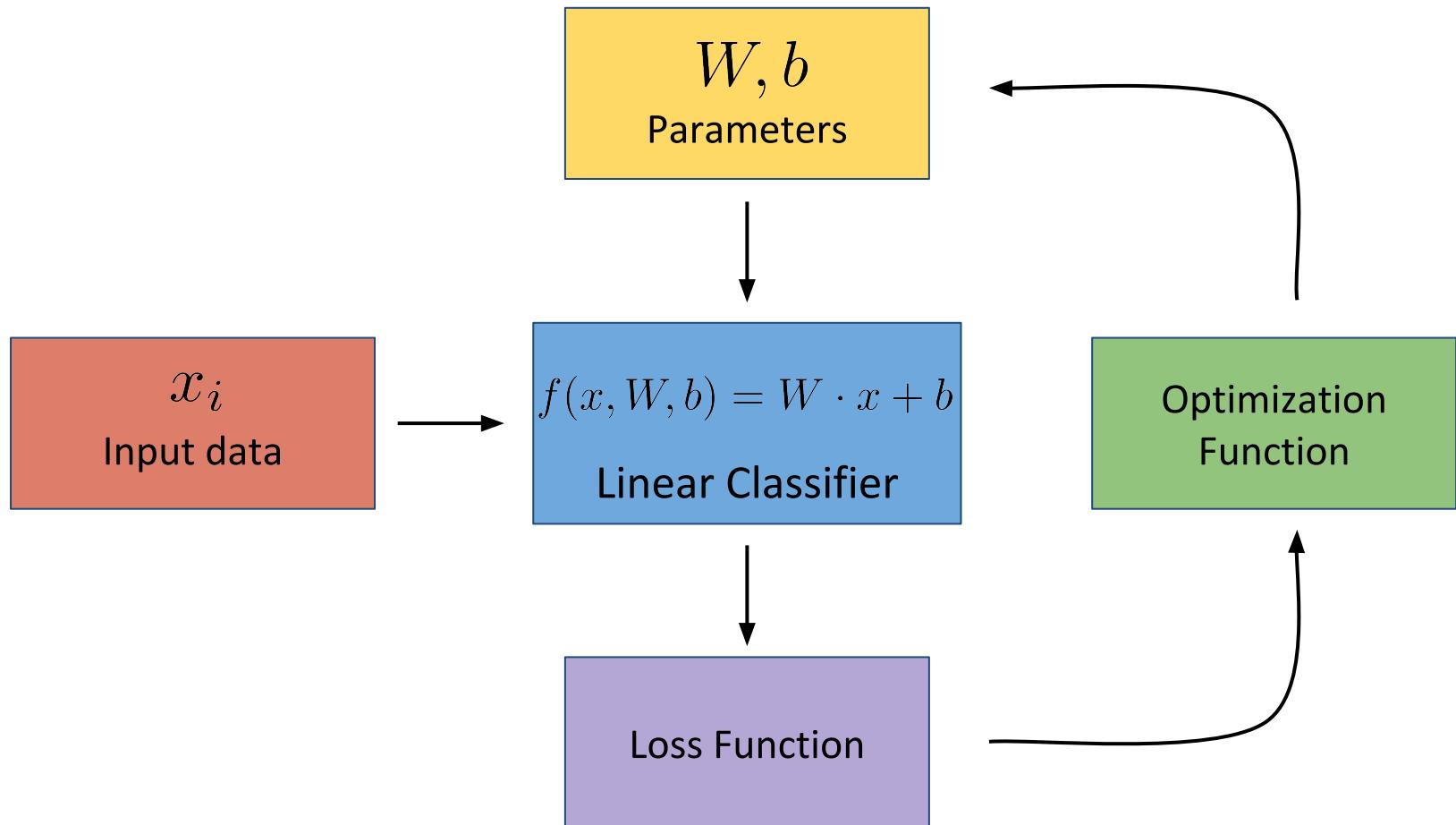


ML Lab with Keras

Ruvan Weerasinghe

(With thanks for material heavily borrowed from Fahim Dalvi of QCRI)

Recap



Overview

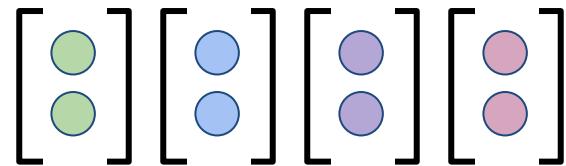
- Linear Algebra in five minutes
- Python Setup
- Jupyter Notebooks
- Introduction to Python & Numpy
- Data Representation
- Linear classifier in Keras

Python Setup

```
# Install dependencies  
> conda install keras numpy matplotlib jupyter  
scikit-learn pydot graphviz nltk nb_conda  
> conda install -c conda-forge ffmpeg
```

Data Representation

Data Representations

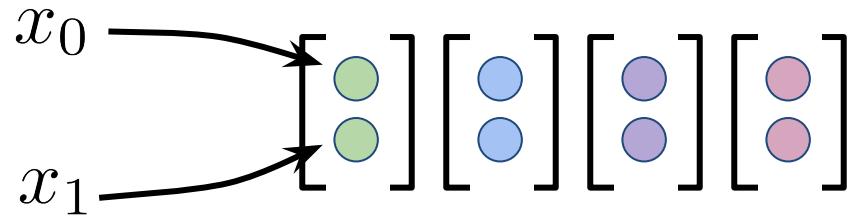


Dataset

4 examples

2 features

Data Representations



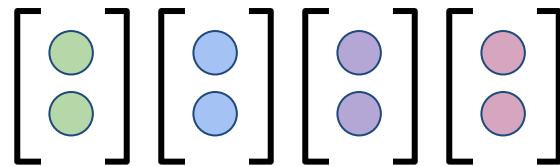
Dataset

4 examples

2 features

Data Representations

Dataset
4 examples
2 features



$$w_0 \begin{bmatrix} w_1 & w_1 \end{bmatrix} \begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} + b = \text{orange}$$

$f(x, w, b) = w \cdot x + b$

Vector

Real number

Linear Regression

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{red} & \text{red} \end{bmatrix} \begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} + \text{red} = \text{orange}$$

$$f(x, w, b) = w \cdot x + b$$

Linear Regression

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \quad \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \quad \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \quad \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$W \rightarrow \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} \begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} = \begin{bmatrix} \text{orange} \\ \text{orange} \end{bmatrix} \leftarrow \text{scores}$$

b

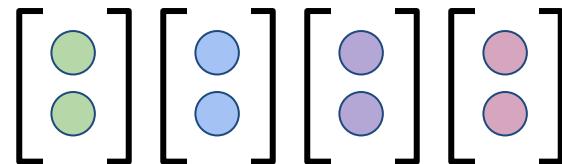
$$f(x, W, b) = \boxed{W} \cdot \boxed{x} + \boxed{b}$$

Matrix Vector

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features



Number of features

Number of classes

$$\left[\begin{array}{cc} \text{red} & \text{red} \\ \text{red} & \text{red} \end{array} \right] \left[\begin{array}{c} \text{green} \\ \text{green} \end{array} \right] + \left[\begin{array}{c} \text{red} \\ \text{red} \end{array} \right]$$

Number of classes

A diagram illustrating the dimensions of a weight matrix for multi-class linear classification. A green bracket labeled "Number of features" covers the first column of a 2x2 matrix of red circles. A red bracket labeled "Number of classes" covers the second column of the same matrix. To the right of the matrix is another red bracket labeled "Number of classes" covering a 2x1 vector of red circles.

Matrix

Vector

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

3 class classification

$$\begin{array}{c} \text{Number of features} \\ \overbrace{\quad\quad\quad}^{\text{Number of classes}} \left[\begin{matrix} \textcolor{red}{\bullet} & \textcolor{red}{\bullet} \\ \textcolor{red}{\bullet} & \textcolor{red}{\bullet} \\ \textcolor{red}{\bullet} & \textcolor{red}{\bullet} \end{matrix} \right] \left[\begin{matrix} \textcolor{teal}{\bullet} \\ \textcolor{teal}{\bullet} \end{matrix} \right] + \left[\begin{matrix} \textcolor{red}{\bullet} \\ \textcolor{red}{\bullet} \\ \textcolor{red}{\bullet} \end{matrix} \right] \overbrace{\quad\quad\quad}^{\text{Number of classes}} \end{array}$$

$[3 \times 2]$ $[2 \times 1]$ $[3 \times 1]$

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{red} & \text{red} \\ \text{red} & \text{red} \end{bmatrix} \begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} = \begin{bmatrix} \text{orange} \\ \text{orange} \end{bmatrix}$$

In this case, we are performing the above computation *per example*

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{red} & \text{red} \\ \text{red} & \text{red} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} = \begin{bmatrix} \text{orange} \\ \text{orange} \end{bmatrix}$$

In this case, we are performing the above computation *per example*

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{red} & \text{red} \\ \text{red} & \text{red} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} = \begin{bmatrix} \text{orange} \\ \text{orange} \end{bmatrix}$$

In this case, we are performing the above computation *per example*

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features

$$\begin{bmatrix} \text{green} \\ \text{green} \end{bmatrix} \begin{bmatrix} \text{blue} \\ \text{blue} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} \begin{bmatrix} \text{pink} \\ \text{pink} \end{bmatrix}$$

$$\begin{bmatrix} \text{red} & \text{red} \\ \text{red} & \text{red} \end{bmatrix} \begin{bmatrix} \text{purple} \\ \text{purple} \end{bmatrix} + \begin{bmatrix} \text{red} \\ \text{red} \end{bmatrix} = \begin{bmatrix} \text{orange} \\ \text{orange} \end{bmatrix}$$

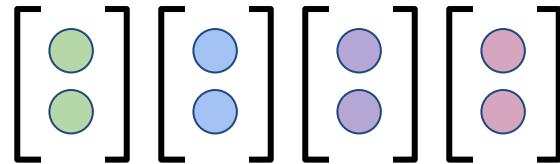
In this case, we are performing the above computation *per example*

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features



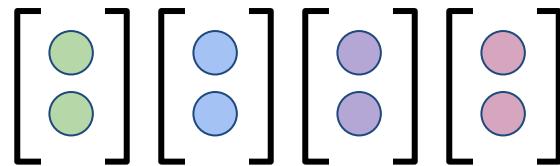
What if we can process all the examples in one go?

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features



What if we can process all the examples in one go?

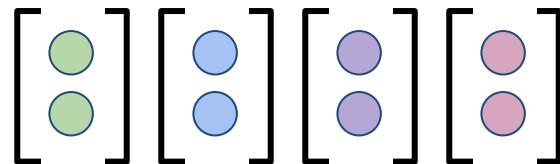
How: Stack all examples into one big matrix!

$$f(x, W, b) = W \cdot x + b$$

Multi-class Linear Classification

Data Representations

Dataset
4 examples
2 features



scores for all examples
per column

$$W \rightarrow [2 \times 2] \begin{bmatrix} [2 \times 4] \\ [2 \times 4] \end{bmatrix} + [2 \times 1] = [2 \times 4]$$

The diagram illustrates the matrix-vector multiplication for linear classification. A weight matrix W of size $[2 \times 2]$ is multiplied by a feature matrix X of size $[2 \times 4]$. The result is then added to a bias vector b of size $[2 \times 1]$. The final output is a score matrix of size $[2 \times 4]$, where each row contains the scores for all examples per column.

$$f(X, W, b) = W \cdot X + b$$

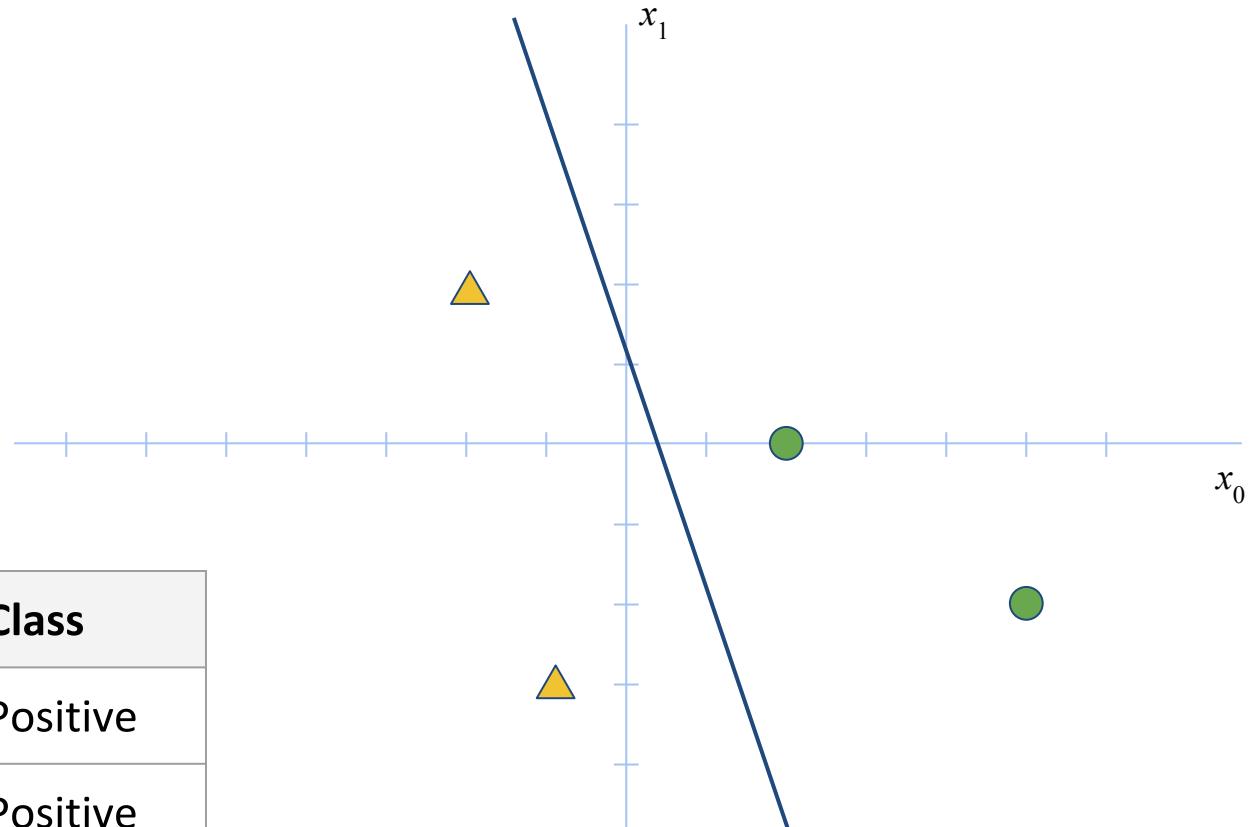
Matrix Vector

The equation $f(X, W, b) = W \cdot X + b$ represents the function for multi-class linear classification. The terms W , X , and b are highlighted with red boxes and labeled as Matrix, Vector, and Matrix respectively.

Efficient Multi-class Linear Classification

Linear Classifier in Keras

Linear Classifier using Regression



x_0	x_1	Class
2	0	Positive
5	-2	Positive
-2	2	Negative
-1	-3	Negative

Linear Classifier in Keras

Data setup

```
data = [(2,0),(5,-2),(-2,2),(-1,-3)]
labels = [-1,-1,1,1]
```

- Usually data is loaded from an external source
- Eventually, all data is represented in some structured form like in matrices
- Data for supervised learning is normally composed of the actual data points and the labels for each point

Linear Classifier in Keras

Data setup

```
x = np.array(data)  
y = np.array(labels)
```

- Eventually, all data is represented in some structured form like in matrices
- Here, we convert all of our data and labels into Numpy arrays

Linear Classifier in Keras

Model definition

```
model = Sequential()
model.add(Dense(1, input_shape=(2,)))

model.compile(loss="mse", optimizer="sgd", metrics=['acc'])
model.summary()
```

- In this case, Dense is the objective function for a linear classifier
- Dense corresponds to the equation of f which is $Wx + b$
- loss computes *mean squared error*

$$f(x, W, b) = w_0 \cdot x_0 + w_1 \cdot x_1 + b$$

$$MSE(x, W, b, y) = (f(x, W, b) - y)^2$$

Linear Classifier in Keras

Model definition

```
model = Sequential()  
model.add(Dense(1, input_shape=(2,)))  
  
model.compile(loss="mse", optimizer="sgd", metrics=['acc'])  
model.summary()
```



Input shape defines the
number of features.
In our case, this is 2

Linear Classifier in Keras

Model definition

```
model = Sequential()  
model.add(Dense(1, input_shape=(2,)))  
  
model.compile(loss="mse", optimizer="sgd", metrics=['acc'])  
model.summary()
```



The number of units of the Dense layer - this corresponds to the number of outputs.

In our case, we only want to output 1 number (regression)

Linear Classifier in Keras

Model definition

```
model = Sequential()  
model.add(Dense(1, input_shape=(2,)))  
  
model.compile(loss="mse", optimizer="sgd", metrics=['acc'])  
model.summary()
```

Mean squared
error loss

Optimization
function is
gradient descent

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    # Recall our classes are [-1,-1,1,1]
    num_correct = 0
    if y_pred[0] < 0: num_correct += 1
    if y_pred[1] < 0: num_correct += 1
    if y_pred[2] > 0: num_correct += 1
    if y_pred[3] > 0: num_correct += 1
    acc = num_correct / 4.0
    loss = loss_history.history['loss'][-1]
    print("Epoch %d: %0.2f (acc) %0.2f (loss)"%(epoch+1, acc, loss))

    # Not mandatory: Save parameters for later analysis
    w, b = model.layers[0].get_weights()
    parameter_history.append((w,b))
```

Optimization loop:
We will run the optimization for 50 epochs

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    # Recall our classes are [-1,-1,1,1]
    num_correct = 0
    if y_pred[0] < 0: num_correct += 1
    if y_pred[1] < 0: num_correct += 1
    if y_pred[2] > 0: num_correct += 1
    if y_pred[3] > 0: num_correct += 1
    acc = num_correct / 4.0
    loss = loss_history.history['loss'][-1]
    print("Epoch %d: %0.2f (acc) %0.2f (loss)" % (epoch, acc, loss))

    # Not mandatory: Save parameters for later
    w, b = model.layers[0].get_weights()
    parameter_history.append((w,b))
```

Here we fit over our data once
In the fit function, Keras automatically computes the objective function, computes the loss and uses the optimizer to adjust the parameters of the model

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    # Recall our classes are [-1,-1,1,1]
    num_correct = 0
    if y_pred[0] < 0: num_correct += 1
    if y_pred[1] < 0: num_correct += 1
    if y_pred[2] > 0: num_correct += 1
    if y_pred[3] > 0: num_correct += 1
    acc = num_correct / 4.0
    loss = loss_history.history['loss'][-1]
    print("Epoch %d: %0.2f (acc) %0.2f (loss)"%(epoch+1, acc, loss))

    # Not mandatory: Save parameters for later analysis
    w, b = model.layers[0].get_weights()
    parameter_history.append((w,b))
```

fit also returns the history of losses for each epoch, along with whatever metrics we requested for when defining the model

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)
```

```
# See how well our model is doing
# Recall our classes are [-1,-1,1,1]
num_correct = 0
if y_pred[0] < 0: num_correct += 1
if y_pred[1] < 0: num_correct += 1
if y_pred[2] > 0: num_correct += 1
if y_pred[3] > 0: num_correct += 1
acc = num_correct / 4.0
loss = loss_history.history['loss'][-1]
print("Epoch %d: %0.2f (acc) %0.2f (loss)" %
```

```
# Not mandatory: Save parameters for later analysis
w, b = model.layers[0].get_weights()
parameter_history.append((w,b))
```

`predict` takes as input some data and returns the value of the objective function (In this case, one value per data point)

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)
```

```
# See how well our model is doing
# Recall our classes are [-1,-1,1,1]
num_correct = 0
if y_pred[0] < 0: num_correct += 1
if y_pred[1] < 0: num_correct += 1
if y_pred[2] > 0: num_correct += 1
if y_pred[3] > 0: num_correct += 1
acc = num_correct / 4.0
loss = loss_history.history['loss'][-1]
print("Epoch %d: %0.2f (acc) %0.2f (loss)" %
```

```
# Not mandatory: Save parameters for later analysis
w, b = model.layers[0].get_weights()
parameter_history.append((w,b))
```

Here, we compare the value of the objective function and assign classes. Recall that a value of < 0 is Class 1, and > 0 is Class 2

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    # Recall our classes are [-1,-1,1,1]
    num_correct = 0
    if y_pred[0] < 0: num_correct += 1
    if y_pred[1] < 0: num_correct += 1
    if y_pred[2] > 0: num_correct += 1
    if y_pred[3] > 0: num_correct += 1
    acc = num_correct / 4.0
    loss = loss_history.history['loss'][-1]
    print("Epoch %d: %0.2f (acc) %0.2f (loss)"%(epoch+1, acc, loss))

    # Not mandatory: Save parameters for later analysis
    w, b = model.layers[0].get_weights()
    parameter_history.append((w,b))
```

Print the progress.
The value of the
loss should go down
with each epoch

Linear Classifier in Keras

Optimization

```
parameter_history = []
for epoch in range(50):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    # Recall our classes are [-1,-1,1,1]
    num_correct = 0
    if y_pred[0] < 0: num_correct += 1
    if y_pred[1] < 0: num_correct += 1
    if y_pred[2] > 0: num_correct += 1
    if y_pred[3] > 0: num_correct += 1
    acc = num_correct / 4.0
    loss = loss_history.history['loss'][-1]
    print("Epoch %d: %0.2f (acc) %0.2f (loss)"%
```

Save parameters
after each epoch to
visualize later

```
# Not mandatory: Save parameters for later analysis
w, b = model.layers[0].get_weights()
parameter_history.append((w,b))
```

Linear Classifier in Keras

Bonus: Plotting

```
plt.scatter([x[0] for x in data],  
           [x[1] for x in data],  
           c=['b','b','r','r'],  
           s=40)
```

```
x1 = np.arange(-20,20,0.1)  
x2 = (-1 * b - (w[0] * x1)) / w[1]  
plt.axis([-15, 15, -6, 6])  
plt.plot(x1,x2)
```

Plot data points

Linear Classifier in Keras

Bonus: Plotting

```
plt.scatter([x[0] for x in data],  
           [x[1] for x in data],  
           c=['b','b','r','r'],  
           s=40)
```

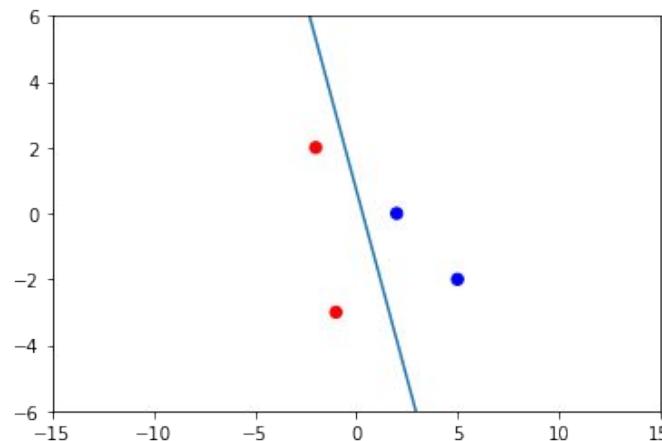
Plot decision boundary

```
x1 = np.arange(-20,20,0.1)  
x2 = (-1 * b - (w[0] * x1)) / w[1]  
plt.axis([-15, 15, -6, 6])  
plt.plot(x1,x2)
```

Linear Classifier in Keras

Bonus: Plotting

```
plt.scatter([x[0] for x in data],  
           [x[1] for x in data],  
           c=['b','b','r','r'],  
           s=40)  
  
x1 = np.arange(-20,20,0.1)  
x2 = (-1 * b - (w[0] * x1)) / w[1]  
plt.axis([-15, 15, -6, 6])  
plt.plot(x1,x2)
```



Linear Classifier in Keras

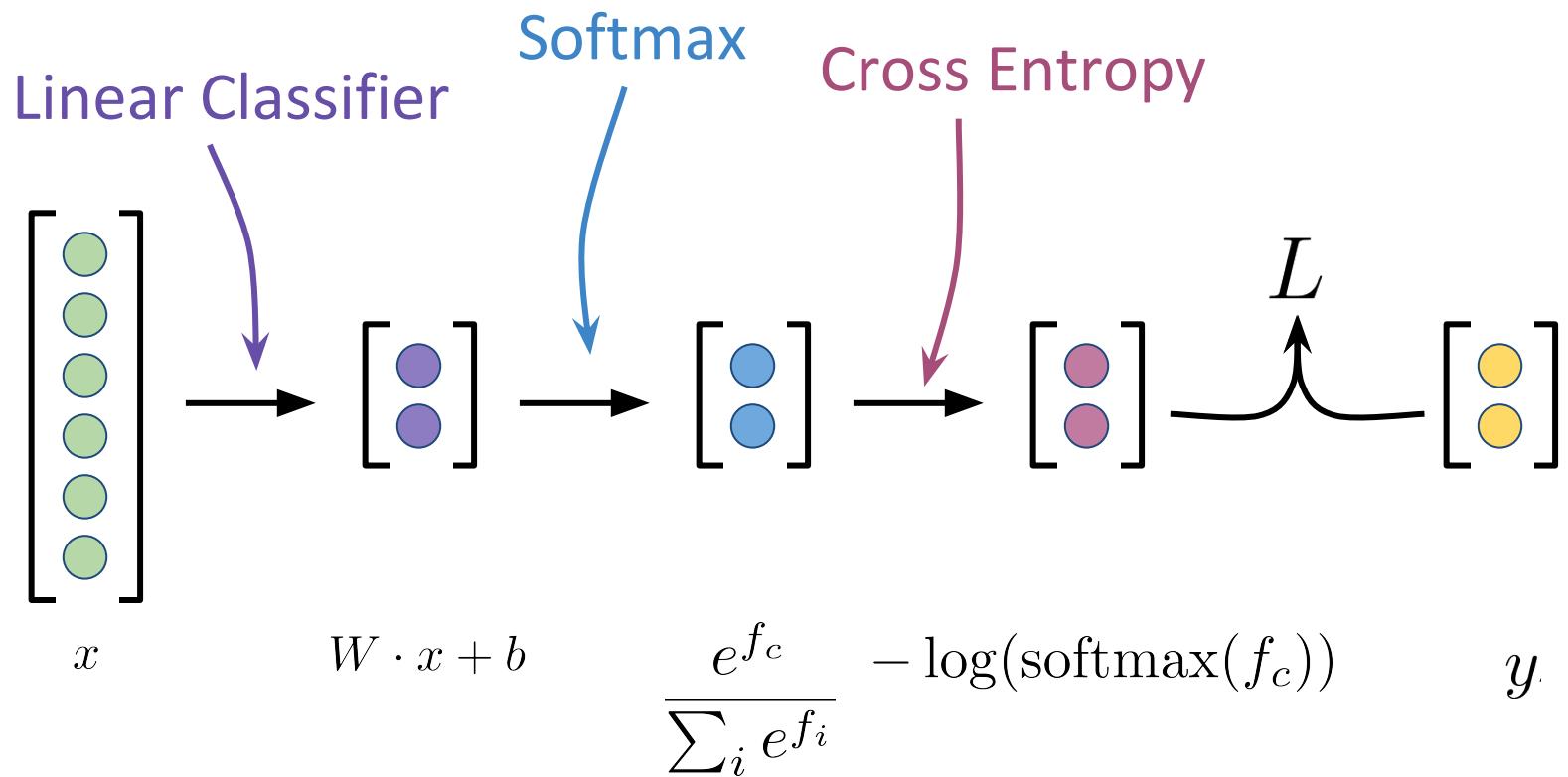
Lets see it in action!



Lecture 1 - Linear Classification by Regression

Binary Linear Classifier in Keras

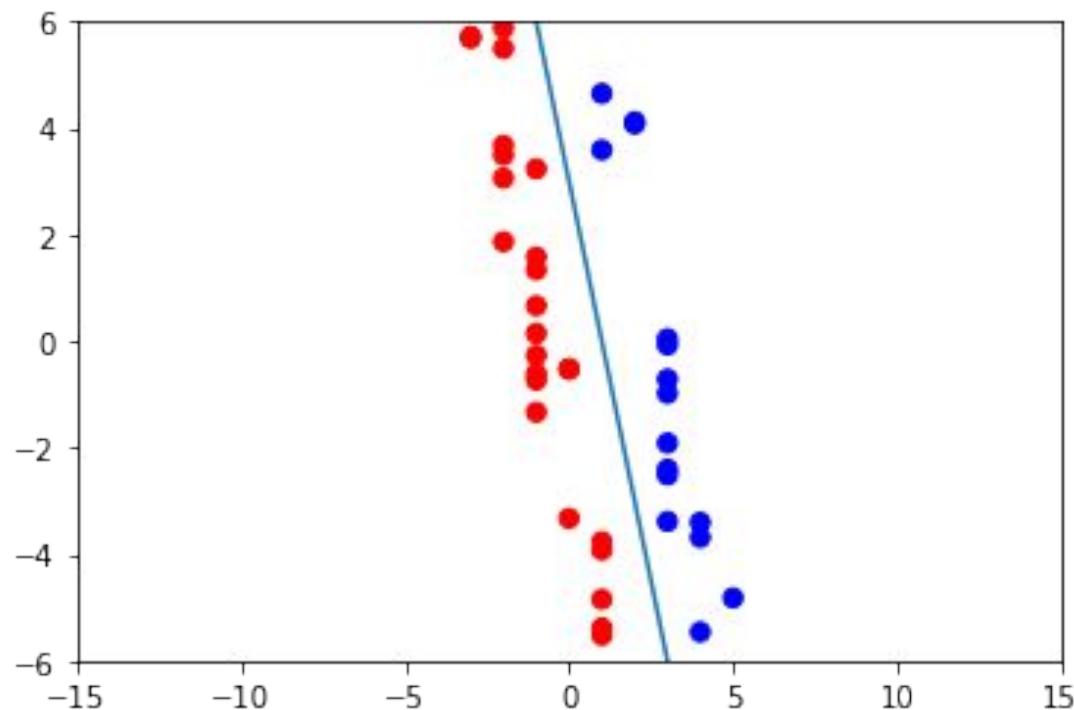
Binary Linear Classifier in Keras



Binary Linear Classifier in Keras

Data setup

This time, we will generate a bigger toy set



Binary Linear Classifier in Keras

Data setup

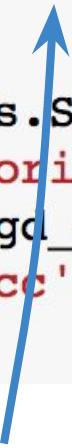
Recall that cross entropy and softmax operate on probability distributions, so our labels now have to be a probability distribution. Therefore we need to do the following mapping:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \xrightarrow{\hspace{1cm}} \begin{bmatrix} \text{Class 1} & \text{Class 2} \\ 0.0 & 1.0 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \\ \vdots & \\ 1.0 & 0.0 \end{bmatrix}$$

Linear Classifier in Keras

Model definition

```
model = Sequential()  
model.add(Dense(2, input_shape=(2,)))  
model.add(Softmax())  
  
sgd_optimizer = optimizers.SGD(lr=0.01)  
model.compile(loss="categorical_crossentropy",  
              optimizer=sgd_optimizer,  
              metrics=['acc'])  
model.summary()
```



The number of features is still 2 in this
case

Linear Classifier in Keras

Model definition

```
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Softmax())

sgd_optimizer = optimizers.SGD(lr=0.01)
model.compile(loss="categorical_crossentropy",
              optimizer=sgd_optimizer,
              metrics=['acc'])
model.summary()
```

Recall that in *binary classification*, we want to output **1 score per class**, hence the number of outputs here will be 2

Linear Classifier in Keras

Model definition

```
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Softmax())
sgd_optimizer = optimizers.SGD(lr=0.01)
model.compile(loss="categorical_crossentropy",
              optimizer=sgd_optimizer,
              metrics=['acc'])
model.summary()
```

Softmax to convert raw scores into
probability distributions

Linear Classifier in Keras

Model definition

```
model = Sequential()  
model.add(Dense(2, input_shape=(2,)))  
model.add(Softmax())  
  
sgd_optimizer = optimizers.SGD(lr=0.01)  
model.compile(loss="categorical_crossentropy",  
              optimizer=sgd_optimizer,  
              metrics=['acc'])  
model.summary()
```

Different from last time, here we explicitly define an optimizer object - this allows us to control parameters such as *learning rate*

Linear Classifier in Keras

Model definition

```
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Softmax())

sgd_optimizer = optimizers.SGD(lr=0.01)
model.compile(loss="categorical_crossentropy",
              optimizer=sgd_optimizer,
              metrics=['acc'])
model.summary()
```



We define the loss as cross entropy

Linear Classifier in Keras

Model definition

```
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Softmax())

sgd_optimizer = optimizers.SGD(lr=0.01)
model.compile(loss="categorical_crossentropy",
              optimizer=sgd_optimizer,
              metrics=['acc'])
model.summary()
```

Use the explicitly initialized optimizer

Linear Classifier in Keras

Optimization

```
for epoch in range(100):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y_probs, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    correct_preds = (np.argmax(y_pred, axis=1) == y)
    num_correct = np.sum(correct_preds)
    acc = num_correct / 100.0
    loss = loss_history.history['loss'][-1]

    if epoch % 10 == 0:
        print("Epoch %d: %.2f (%acc) %.2f (loss)"%(epoch+1, acc, loss))
```

Optimization loop:
Mostly the same as last time

Linear Classifier in Keras

Optimization

```
for epoch in range(100):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y_probs, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    correct_preds = (np.argmax(y_pred, axis=1) == y)
    num_correct = np.sum(correct_preds)
    acc = num_correct / 100.0
    loss = loss_history.history['loss'][-1]

    if epoch % 10 == 0:
        print("Epoch %d: %.2f (%acc) %.2f (loss)"%(epoch+1, acc, loss))
```

Recall: prediction is no more defined as > 0 or < 0 , but rather we pick the class with the maximum score (Using argmax here from numpy)

Linear Classifier in Keras

Optimization

```
for epoch in range(100):
    # Perform one step over the entire dataset
    loss_history = model.fit(X, y_probs, epochs=1, verbose=False)

    # Get predictions (value of the objective function, f)
    y_pred = model.predict(X, verbose=False)

    # See how well our model is doing
    correct_preds = (np.argmax(y_pred, axis=1) == y)
    num_correct = np.sum(correct_preds)
    acc = num_correct / 100.0
    loss = loss_history.history['loss'][-1]

    if epoch % 10 == 0:
        print("Epoch %d: %.2f (%acc) %.2f (loss)"%(epoch+1, acc, loss))
```

We compare the values from argmax against the true class labels