Understanding Gradient Based Adversarial Attacks

Adversarial perturbations are small changes to an machine learning model's input with the goal of pushing that input over the model's decision boundary. Visualizing decision boundaries and understanding concepts from gradient-based adversarial attacks can be challenging in high dimensional spaces. This homework is adapted from the notebook https://github.com/adrian-botta/understanding_adversarial_examples/blob/master/adversarial_examples_logistic_regressior (https://github.com/adrian-botta/understanding_adversarial_examples/blob/master/adversarial_examples_logistic_regressior

We'll start with a simple classification example with a subset of the Iris dataset - (inspired from: https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac (https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac))

In [ ]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn import datasets
from matplotlib.lines import Line2D

sns.set(style="white")
figure_size = [11,7.5]
```

In [ ]:
```
iris = datasets.load_iris()
X = iris.data[:, :2]
y = (iris.target != 0) * 1
```

In [ ]:
```
colors = np.asarray(['r']*len(y))
colors[y==1] = 'b'
```

In [ ]:
```
f, ax = plt.subplots(figsize=(figure_size[0], figure_size[1]))
ax.scatter(X[:,0], X[::, 1], c=colors, edgecolor="white", linewidth=1)
plt.xlabel("X1"),plt.ylabel("X2")
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='r', markersize=5),
                   Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='b', markersize=5)]
ax.legend(handles=legend_elements)
```

Above, we have a scatter plot showing two classes in the sample dataset. We will use a logistic regression as our classifier and can use its coefficients to draw its decision boundary and create a probability distribution of the class of a data point given the X1 & X2 variables.

We'll fit a logistic regression to X values by changing Theta

In [ ]:
```
clf = LogisticRegression().fit(X, y)
w = clf.coef_[0]
a = -w[0] / w[1]
xx_mod = np.linspace(3.5, 8.4)
yy_mod = a * xx_mod - (clf.intercept_[0]) / w[1]
```

In [ ]:
```
f, ax = plt.subplots(figsize=(figure_size[0], figure_size[1]))
ax.scatter(X[:,0], X[:, 1], c=colors,edgecolor="white", linewidth=1)
plt.xlabel("X1"),plt.ylabel("X2")
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='r', markersize=5),
                   Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='b', markersize=5),
                   Line2D([0], [0], marker='None', color='black', label=
                          markerfacecolor='black', markersize=5)]
ax.legend(handles=legend_elements)
ax.plot(xx_mod, yy_mod, 'k-')
```

In [ ]:
```
xx, yy = np.mgrid[3.5:8.5:.01, 1.5:5.5:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
f, ax = plt.subplots(figsize=(figure_size[0], figure_size[1]))
contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",
                      vmin=0, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label("$P(y = 1)$")
ax_c.set_ticks([0, .25, .5, .75, 1])

ax.scatter(X[:,0], X[::, 1], c=y, s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

legend_elements = [Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='r', markersize=5),
                   Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='b', markersize=5),
                   Line2D([0], [0], marker='None', color='black', label=
                          markerfacecolor='black', markersize=5)]
legend = ax.legend(handles=legend_elements, frameon=True, framealpha=1)
legend.get_frame().set_facecolor('white')
plt.xlabel("X1"),plt.ylabel("X2")
ax.plot(xx_mod, yy_mod, 'k-')
```

When we train our machine learning models, we're defining a loss function that we then aim to minimize

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{M} \log(1 + \exp(-y_i \theta^T x_i)),$$

where y_i is the label of the ith sample, and x_i is a two-dimensional vector representing the feature of the ith sample.

In [ ]:
```
theta = clf.coef_
def loss(theta, X, y):


    return
```

In [ ]:
```
print("This is our model's training loss:", loss(theta, X, y))
```

To update the values of theta, we take the partial derivative of our loss function with respect to (w.r.t) theta. The partial derivative tells us how to change the values of theta to change the loss

$$\frac{\partial J(\theta)}{\partial \theta} = ??$$

(please evaluate)

In [ ]:
```
def gradient_wrt_theta(theta, X, y):
    """
    Compute the gradient of logistic loss with respect to theta.

    Args:
        theta: (1,2) numpy array, parameter vector
        X: (m, 2) numpy array, input features
        y: (m,) numpy array, labels

    Returns:
        grad: (1,2) numpy array, gradient vector
    """





    return
```

In [ ]:
```
print("Grads w.r.t. Theta",gradient_wrt_theta(theta, X, y))
```

With gradient-based adversarial attacks, we want to use the gradients to determine how to change X such that the loss increases (or decreases with a targeted attack). Implement the fast gradient sign method-based adversarial sample generation, i.e.,

$$X' = X + \eta = X + \epsilon \times \text{Sign}\left(\frac{\partial J(\theta)}{\partial X}\right)$$

(please evaluate)

In [ ]:
```
import numpy as np

def sign_gradient_wrt_X(theta, y):

    return
```

```
*** Creating the adversarial perturbation ***
```

```
To create the adversarial X values. Finish the cell below.
```

In [ ]:
```
epsilon = 0.5
sign_dX = sign_gradient_wrt_X(theta, y)
X_advs = ???
```

In [ ]:
```
f, ax = plt.subplots(figsize=(figure_size[0], figure_size[1]))
colors = np.asarray(['gray']*len(y))
ax.scatter(X[:50,0], X[:50, 1], c='grey', s=50,#y[:1], s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)
ax.scatter(X[:1,0], X[:1, 1], c='red', s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="black", linewidth=1)
ax.scatter(X_advs[:1,0], X_advs[:1, 1], c='blue', s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="black", linewidth=1)
plt.arrow(X[0,0], X[0,1], (X_advs[0,0]-X[0,0])*0.8, (X_advs[0,1]-X[0,1])
         head_width = 0.05, color="black")
ax.plot(xx_mod, yy_mod, 'k-')
legend_elements = [Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='grey', markersize=5),
                   Line2D([0], [0], marker='o', color='w', label='Origin
                          markeredgewidth=1, markerfacecolor='red', mark
                   Line2D([0], [0], marker='o', color='w', label='Advers
                          markeredgewidth=1, markerfacecolor='blue', mar
                   Line2D([0], [0], marker='None', color='black', label=
                          markerfacecolor='black', markersize=5)]
ax.legend(handles= legend_elements)
plt.xlabel("X1"),plt.ylabel("X2")
```

From the figure above, we're able to see that we chose the correct step size (epsilon) to cross our victim model's decision boundary. In the figure below, we can see that this step size works for several points in our dataset.

In [ ]:
```
sample = np.asarray([1,2,3,7,8,9,12,18,20,23,25,26,27,28,
                     29,30,31,34,35,36,37,38,39,45,47,49])

f, ax = plt.subplots(figsize=(figure_size[0], figure_size[1]))
colors = np.asarray(['gray']*len(y))
ax.scatter(X[:50,0], X[:50, 1], c=colors[:50], s=50,#y[:1], s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

ax.scatter(X[sample,0], X[sample, 1], c='red', s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="black", linewidth=1)

ax.scatter(X_advs[sample,0], X_advs[sample, 1], c='blue', s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="black", linewidth=1)

for arr in sample:
    plt.arrow(X[arr,0], X[arr,1],
          (X_advs[arr,0]-X[arr,0])*0.80,
          (X_advs[arr,1]-X[arr,1])*0.80,
          head_width = 0.05, color="black")

legend_elements = [Line2D([0], [0], marker='o', color='w', label='Class:
                          markerfacecolor='grey', markersize=5),
                   Line2D([0], [0], marker='o', color='w', markeredgecolor =
                          markeredgewidth = 1, markerfacecolor='red', marker
                   Line2D([0], [0], marker='o', color='w', markeredgecolor =
                          markeredgewidth = 1, markerfacecolor='blue', marke
                   Line2D([0], [0], marker='None', color='black', label='Dec
                          markerfacecolor='black', markersize=5)]
ax.legend(handles= legend_elements)
plt.xlabel("X1"),plt.ylabel("X2")
ax.plot(xx_mod, yy_mod, 'k-')
```

*** Checking predictions ***

We can see that crossing the decision boundary indeed changed our model's predictions

In [ ]:
```
print("victim model predictions", clf.predict(X_advs[sample]))
print("original labels:        ", y[sample])
```