Artificial Intelligence and Machine Learning

Unit II

Perceptron and Logistic Regression

My own latex definitions

In [19]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('seaborn-whitegrid')

font = {'family' : 'Times',
        'weight' : 'bold',
        'size'   : 12}

matplotlib.rc('font', **font)


# Aux functions

def plot_grid(Xs, Ys, axs=None):
    ''' Aux function to plot a grid'''
    t = np.arange(Xs.size) # define progression of int for indexing colormap
    if axs:
        axs.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        axs.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        axs.axis('scaled') # axis scaled
    else:
        plt.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        plt.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        plt.axis('scaled') # axis scaled

def linear_map(A, Xs, Ys):
    '''Map src points with A'''
    # [NxN,NxN] -> NxNx2 # add 3-rd axis, like adding another layer
    src = np.stack((Xs,Ys), axis=Xs.ndim)
    # flatten first two dimension
    # (NN)x2
    src_r = src.reshape(-1,src.shape[-1]) #ask reshape to keep last dimension and adjust the rest
    # 2x2 @ 2x(NN)
    dst = A @ src_r.T # 2xNN
    #(NN)x2 and then reshape as NxNx2
    dst = (dst.T).reshape(src.shape)
    # Access X and Y
    return dst[...,0], dst[...,1]


def plot_points(ax, Xs, Ys, col='red', unit=None, linestyle='solid'):
    '''Plots points'''
    ax.set_aspect('equal')
    ax.grid(True, which='both')
    ax.axhline(y=0, color='gray', linestyle="--")
    ax.axvline(x=0, color='gray',  linestyle="--")
    ax.plot(Xs, Ys, color=col)
    if unit is None:
        plotVectors(ax, [[0,1],[1,0]], ['gray']*2, alpha=1, linestyle=linestyle)
    else:
        plotVectors(ax, unit, [col]*2, alpha=1, linestyle=linestyle)

def plotVectors(ax, vecs, cols, alpha=1, linestyle='solid'):
    '''Plot set of vectors.'''
    for i in range(len(vecs)):
        x = np.concatenate([[0,0], vecs[i]])
        ax.quiver([x[0]],
                  [x[1]],
                  [x[2]],
                  [x[3]],
                  angles='xy', scale_units='xy', scale=1, color=cols[i],
                  alpha=alpha, linestyle=linestyle, linewidth=2)
```

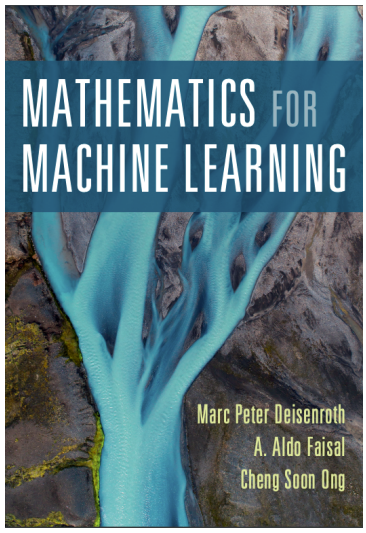
```
/var/folders/rt/lg7n4lt1489270pz_18qn1_c0000gp/T/ipykernel_18217/1496334134.py:5: MatplotlibDeprecationWarning:
The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles s
hipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use t
he seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```

In [20]:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('seaborn-whitegrid')

font = {'family' : 'Times',
        'weight' : 'bold',
        'size'   : 12}

matplotlib.rc('font', **font)


# Aux functions

def plot_grid(Xs, Ys, axs=None):
    ''' Aux function to plot a grid'''
    t = np.arange(Xs.size) # define progression of int for indexing colormap
    if axs:
        axs.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        axs.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        axs.axis('scaled') # axis scaled
    else:
        plt.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        plt.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        plt.axis('scaled') # axis scaled

def linear_map(A, Xs, Ys):
    '''Map src points with A'''
    # [NxN,NxN] -> NxNx2 # add 3-rd axis, like adding another layer
    src = np.stack((Xs,Ys), axis=Xs.ndim)
    # flatten first two dimension
    # (NN)x2
    src_r = src.reshape(-1,src.shape[-1]) #ask reshape to keep last dimension and adjust the rest
    # 2x2 @ 2x(NN)
    dst = A @ src_r.T # 2xNN
    #(NN)x2 and then reshape as NxNx2
    dst = (dst.T).reshape(src.shape)
    # Access X and Y
    return dst[...,0], dst[...,1]


def plot_points(ax, Xs, Ys, col='red', unit=None, linestyle='solid'):
    '''Plots points'''
    ax.set_aspect('equal')
    ax.grid(True, which='both')
    ax.axhline(y=0, color='gray', linestyle="--")
    ax.axvline(x=0, color='gray',  linestyle="--")
    ax.plot(Xs, Ys, color=col)
    if unit is None:
        plotVectors(ax, [[0,1],[1,0]], ['gray']*2, alpha=1, linestyle=linestyle)
    else:
        plotVectors(ax, unit, [col]*2, alpha=1, linestyle=linestyle)

def plotVectors(ax, vecs, cols, alpha=1, linestyle='solid'):
    '''Plot set of vectors.'''
    for i in range(len(vecs)):
        x = np.concatenate([[0,0], vecs[i]])
        ax.quiver([x[0]],
                  [x[1]],
                  [x[2]],
                  [x[3]],
                  angles='xy', scale_units='xy', scale=1, color=cols[i],
                  alpha=alpha, linestyle=linestyle, linewidth=2)
```

```
/var/folders/rt/lg7n4lt1489270pz_18qn1_c0000gp/T/ipykernel_18217/1496334134.py:5: MatplotlibDeprecationWarning:
The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles s
hipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use t
he seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```

Yet Another Text Book



Today's lecture

Supervised, Parametric Models

Propaedeutic part for Deep Learning

1) The Perceptron Algorithm

2) Logistic Regression

ℹ️Where we are now

Topic	Hours
Intro, Math Recap	-
Unsupervised Learning	
Dimensionality Reduction (PCA, Eigenvectors, SVD)	-
Clustering (kmeans, GMM)	-
Supervised Learning, Non-parametric	
Nearest Neighbours	-
Decision trees	-
Supervised Learning, Parametric	
Linear Regression with Least Squares	-
Polynomial regression, under/overfitting	-
~~Kernel Methods and SVM~~	-
⋆ Perceptron, Logistic Regression (LR)	-
Deep Learning	
⋆ Perceptron, Logistic Regression (LR) from LR to Neural Nets	-
Total	60

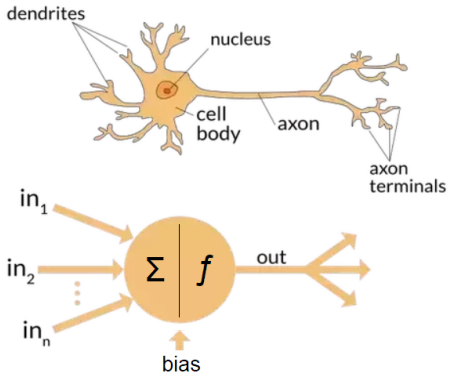
This lecture material is taken from

- Mostly from Stanford Note
- Mostly from Stanford class
- Bishop - Chapter 4 - Section 4.3.2 page 205 (not very well explained)
- Tibshirani - Chapter 4 - Section 4.4 page 119

The Perceptron Algorithm

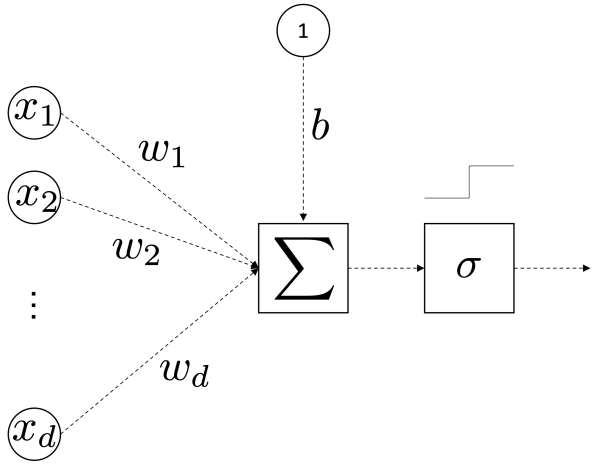
The Perceptron Algorithm

- We study this algorithm mostly for **historical importance and didactic duty**
- Nowadays it is not practically used anymore.
- Very loosely inspired by the concept of **neurons** (that backed the whole idea of Neural Nets)
- I do not like the word "Neural", creates false belief/hype



"Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962

The Perceptron Algorithm



The Perceptron Algorithm

$\mathbf{x}_i \in \mathrm{R}^d$ and $y \in \{0, 1\}$

$$f_{\theta}(\mathbf{x}) \doteq \sigma\left(\theta^T \mathbf{x}\right)$$

where:

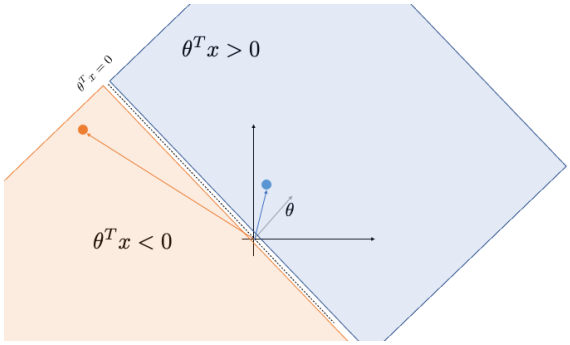
$$\sigma(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

{{import numpy as np;import matplotlib.pyplot as plt;x = np.arange(-20.0, 20.0, 0.1);y = np.maximum(np.sign(x),0);_=plt.plot(x,y);}}

Observations

A few observations:

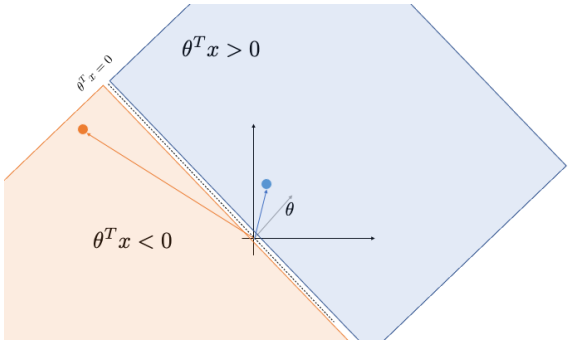
- The magnitude of the dot product $\boldsymbol{\theta}^T\mathbf{x}$ does not matter. What matters is the **sign of the dot product**.
- $\boldsymbol{\theta}^T\mathbf{x} > 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **less** than 90 degree.



Observations

A few observations:

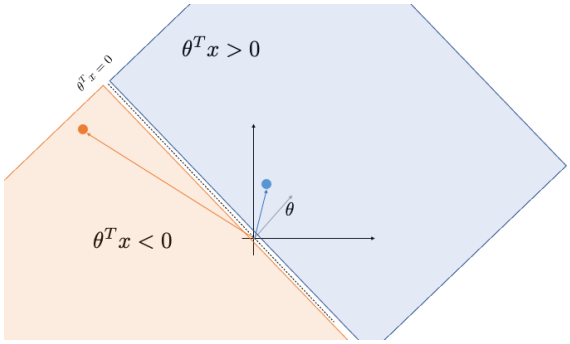
- The magnitude of the dot product $\boldsymbol{\theta}^T\mathbf{x}$ does not matter. What matters is the **sign of the dot product**.
- $\boldsymbol{\theta}^T\mathbf{x} > 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **less** than 90 degree.
- $\boldsymbol{\theta}^T\mathbf{x} < 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **more** than 90 degree.



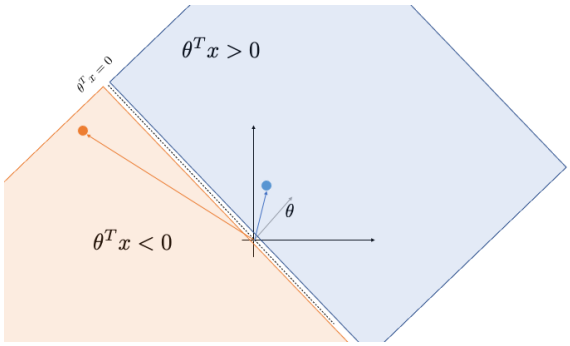
Observations

A few observations:

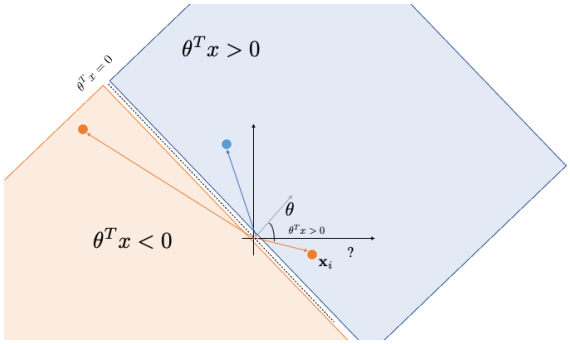
- The magnitude of the dot product $\boldsymbol{\theta}^T\mathbf{x}$ does not matter. What matters is the **sign of the dot product**.
- $\boldsymbol{\theta}^T\mathbf{x} > 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **less** than 90 degree.
- $\boldsymbol{\theta}^T\mathbf{x} < 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **more** than 90 degree.
- $\boldsymbol{\theta}^T\mathbf{x} = 0$ when this condition is geometrically satisfied? The angle between $\boldsymbol{\theta}, \mathbf{x}$ is **exactly** 90 degree (decision boundary).



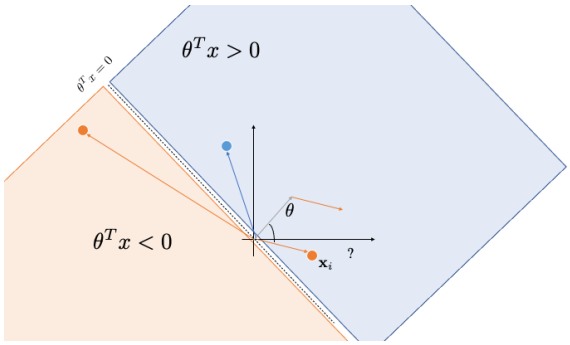
Learning the Perceptron



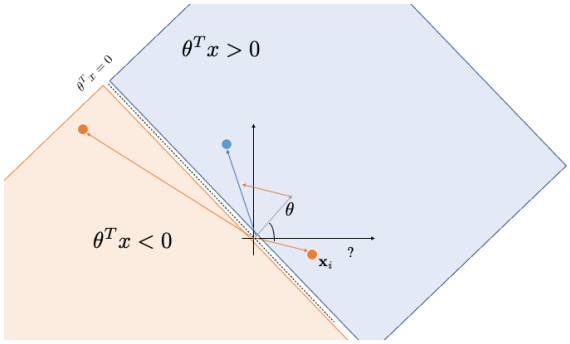
Learning the Perceptron



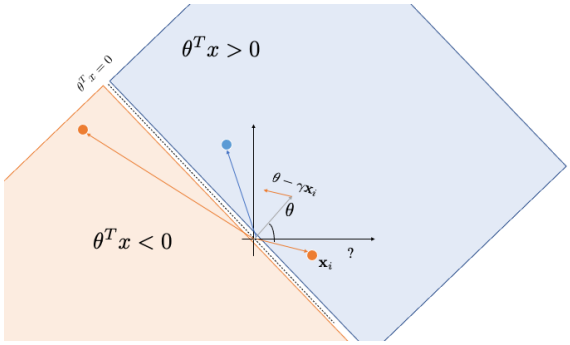
Learning the Perceptron



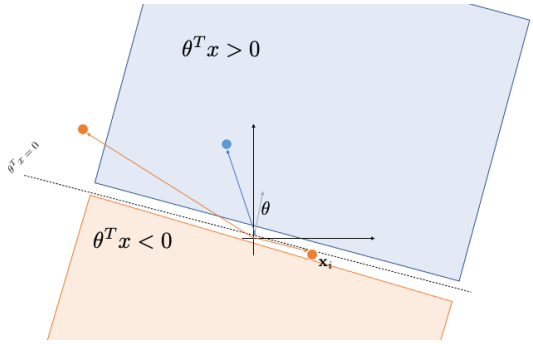
Learning the Perceptron



Learning the Perceptron



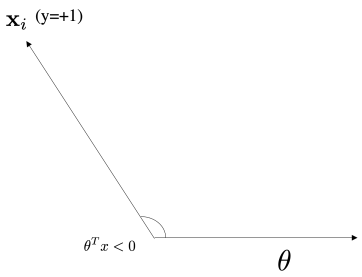
Learning the Perceptron



Key Observation

- if the **misclassified** point is **negative** ($y = 0$) `the angle is less than 90 deg`, change $\theta \leftarrow \theta - \gamma \mathbf{x}$
- if the **misclassified** point is **positive** ($y = 1$) `the angle is more than 90 deg`, change $\theta \leftarrow \theta + \gamma \mathbf{x}$

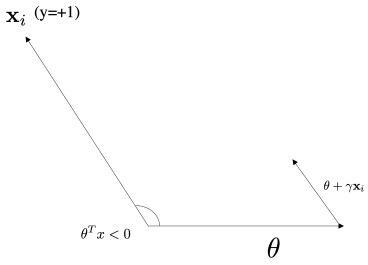
$$\theta^T \mathbf{x} \leq (\theta + \gamma \mathbf{x})^T \mathbf{x} = \theta^T \mathbf{x} + \underbrace{\gamma \mathbf{x}^T \mathbf{x}}_{\text{always positive}}$$



Key Observation

- if the **misclassified** point is **negative** ($y = 0$) `the angle is less than 90 deg`, change $\theta \leftarrow \theta - \gamma \mathbf{x}$
- if the **misclassified** point is **positive** ($y = +1$) `the angle is more than 90 deg`, change $\theta \leftarrow \theta + \gamma \mathbf{x}$

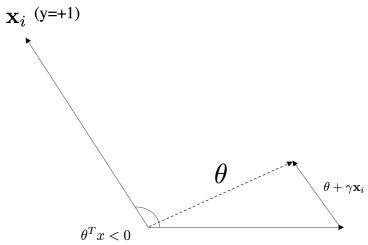
$$\theta^T \mathbf{x} \leq (\theta + \gamma \mathbf{x})^T \mathbf{x} = \theta^T \mathbf{x} + \underbrace{\gamma \mathbf{x}^T \mathbf{x}}_{\text{always positive}}$$



Key Observation

- if the **misclassified** point is **negative** ($y = 0$) `the angle is less than 90 deg`, change $\theta \leftarrow \theta - \gamma \mathbf{x}$
- if the **misclassified** point is **positive** ($y = +1$) `the angle is more than 90 deg`, change $\theta \leftarrow \theta + \gamma \mathbf{x}$

$$\theta^T \mathbf{x} \leq (\theta + \gamma \mathbf{x})^T \mathbf{x} = \theta^T \mathbf{x} + \underbrace{\gamma \mathbf{x}^T \mathbf{x}}_{\text{always positive}}$$



Learning the Perceptron

1. **Initialization - Set all params to zero**

   $$\boldsymbol{\theta} \doteq \mathbf{0}^T$$

   Set it to all zeros or random initialization from a distribution.
2. Repeat until **convergence**:

   $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \Big(\sigma(\theta^T \mathbf{x}) - y \Big) \mathbf{x}$$

   OR

   $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma \Big(y - \sigma(\theta^T \mathbf{x}) \Big) \mathbf{x}$$
3. When convergence is reached, you final estimate is in $\boldsymbol{\theta}$

Perceptron Update Rule: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma \Big(y - \sigma(\theta^T \mathbf{x}) \Big) \mathbf{x}$

- In the correct case (both negative and positive point), $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}$ is kept the same.
- Given $(\mathbf{x}, y = 1)$ but $\sigma\Big(\theta^T \mathbf{x}\Big) = 0$ then update θ with a bit of $+\mathbf{x}$. (add a bit of x to θ).
- Given $(\mathbf{x}, y = 0)$ but $\sigma\Big(\theta^T \mathbf{x}\Big) = 1$ then update θ with a bit of $-\mathbf{x}$. (subtract a bit of x to θ).

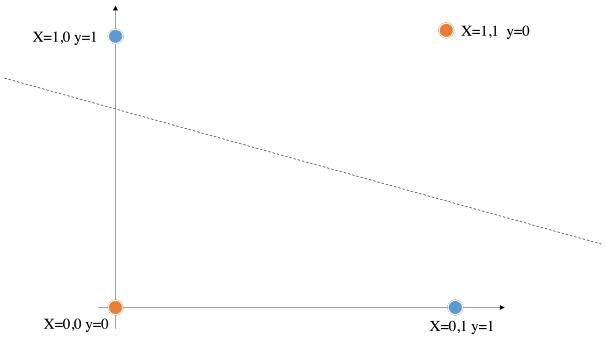
θ is a linear combination of training samples $\{\mathbf{x}_i, y_i\}$

Theory:

Given a linearly separable training set, the **Perceptron** will find a generic solution of the infinite many that classifies correctly all the points (separate all the points neatly).

1. It will converge (convergence speed will depend on the `margin` between points). Small margins require more time to converge.
2. A non-optimal solution is guaranteed to be found.

Perceptron cannot learn XOR logic function



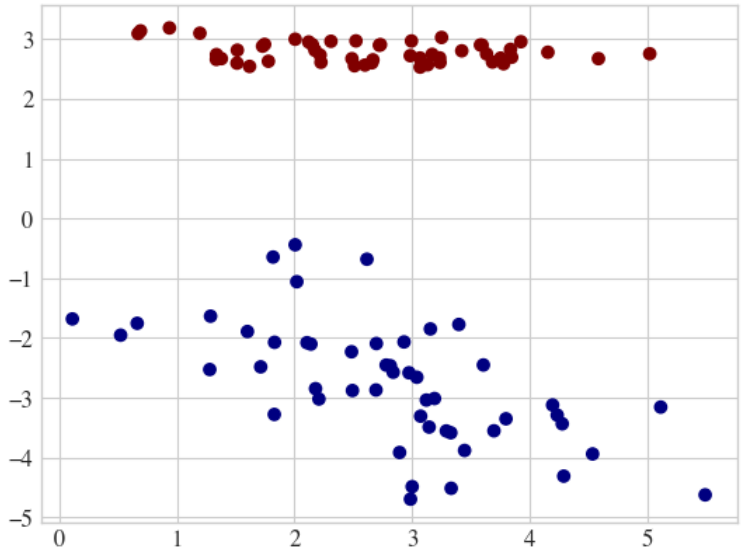
In [21]:
```
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import PolynomialFeatures
import numpy as np

lift_features = True

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = X[:, 0] ^ X[:, 1]

if lift_features:
    print('X before=',X, sep='\n')
    X = PolynomialFeatures(interaction_only=True).fit_transform(X).astype(int)
    print('X before=',X,y.T, sep='\n')
clf = Perceptron(fit_intercept=False, max_iter=10, tol=None,
                 shuffle=False).fit(X, y)
clf.predict(X)
clf.score(X, y)
```

```
X before=
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
X before=
[[1 0 0 0]
 [1 0 1 0]
 [1 1 0 0]
 [1 1 1 1]]
[0 1 1 0]
```

Out[21]:
```
1.0
```

In [22]:
```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_features=2, n_redundant=0, n_informative=2,
    n_clusters_per_class=1, random_state=0, class_sep=2.8)
```

In [23]:
```
plt.scatter(*X.T,c=y,cmap='jet');
```

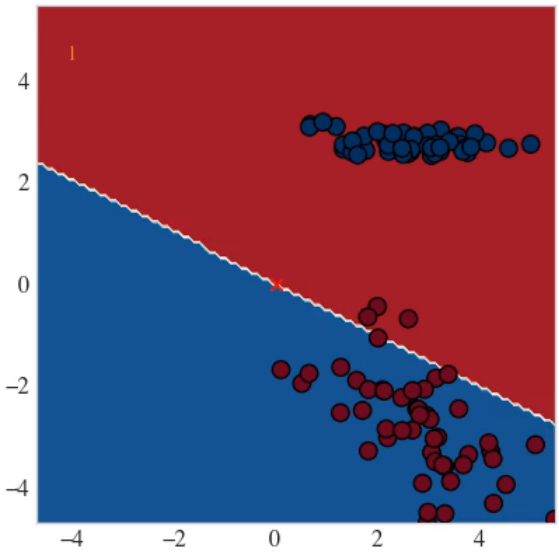


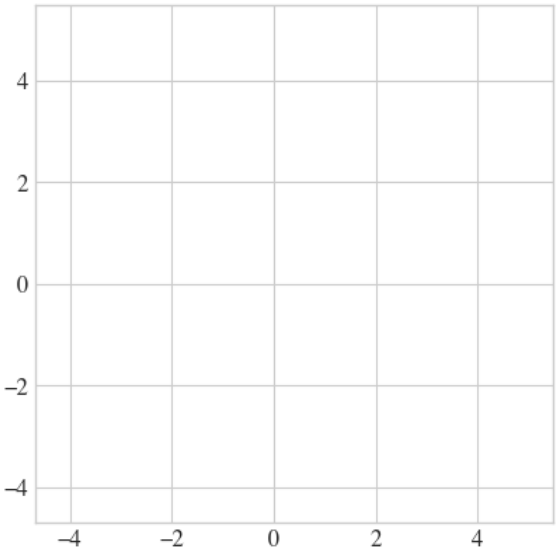
Awesome demo coming up

In [24]:

```
from sklearn.datasets import make_classification
from matplotlib import cm
from celluloid import Camera
from IPython.display import HTML

def get_support(minx, maxx):
    support = np.linspace(minx, maxx, 100)
    xx, yy = np.meshgrid(support, support)
    points = np.stack((xx.flatten(), yy.flatten()), axis=1)
    points = add_ones(points)
    return points, xx, yy

def plot_separating_plane(w, minx, maxx, points, xx, yy):
    dist = perceptron(w, points.T)
    dist = dist.reshape(xx.shape)
    plt.contourf(xx, yy, dist, cmap=cm.get_cmap("RdBu"))
    plt.plot(0, 0, 'rx')
    plt.axis('scaled')
    plt.xlim(minx, maxx)


def perceptron(w, xi):
    return np.maximum(np.sign(np.dot(w, xi)), 0)


def add_ones(X):
    bias = np.ones((X.shape[0], 1))
    X = np.hstack((X, bias))
    return X

def done(w, X):
    return np.all(y == perceptron(w, X.T))


def plot_classification(theta_curr, minx, maxx, support, xx, yy, X):
    plot_separating_plane(theta_curr, minx, maxx, support, xx, yy)
    plt.scatter(X[:, 0], X[:, 1], s=80,c=y,
            facecolors="none",
            zorder=10,
            edgecolors="k",
            cmap=cm.get_cmap("RdBu"))

### CAMERA ##########
fig = plt.figure()
camera = Camera(fig)

################## DATA ####################
#X, y = make_classification(
#    n_features=2, n_redundant=0, n_informative=2,
#    n_clusters_per_class=1,
#    random_state=0, class_sep=2.8)
minx, maxx = X.min(), X.max()
support, xx, yy = get_support(minx, maxx)
X = add_ones(X)
############### PARAMS ########################
theta_curr = np.array([[-1, -2, 0]])
gamma = 1e-2
i = 0
text_kwargs = dict(ha='center', va='center', fontsize=10, color='C1')
###############################################
plot_separating_plane(theta_curr, minx, maxx, support, xx, yy)
exit = False
# while not all are classified
while not done(theta_curr,X) and i < 300:
    # for all points
    for xi, yi in zip(X, y):
        prediction = perceptron(theta_curr, xi)
        # if prediction is wrong, update
        if prediction != yi:
            diff = yi - perceptron(theta_curr, xi)
            theta_curr = theta_curr + gamma * diff * xi
            plot_classification(theta_curr, minx, maxx, support, xx, yy, X)
            i += 1
            plt.text(-4, 4.5, str(i), **text_kwargs)
            camera.snap()
# write 20 frames with the last result for memory
for i in range(20):
    plot_classification(theta_curr, minx, maxx, support, xx, yy, X)
    camera.snap()
animation = camera.animate()
HTML(animation.to_html5_video())
```

Out[24]:





Perceptron Weakness

- It loops forever if the data is **NOT** linearly separable
- Even if the data is linearly separable, it may take an amount of time that is **infeasible** in practice to use it
- Does not converge to an optimal solution such as SVM, but it just returns a generic separating hyper-plane

Perceptron History

Figure 4.8 Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focussed onto a 20×20 array of cadmium sulphide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

Logistic Regression

Logistic Regression

- The name is a bit *misleading*: it is used as a **discriminative classifier**
- Discriminative means we model $p(y \mid x)$ *what is the probability for the label* $prob(Y = y)$ *given* x*?*
- Very commonly used algorithm

Logistic Regression

- $\mathbf{x} \in \mathrm{R}^d \quad y \in \{0, 1\}$

$$\begin{cases} y = 1 & \text{positive example} \\ y = 0 & \text{negative example} \end{cases}$$

Logistic Regression (also called Logit)

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \doteq \sigma\left(\boldsymbol{\theta}^T \mathbf{x}\right)$$

where:

$$\sigma(z) = \frac{1}{1 + \exp^{-z}} \quad \text{sigmoid or logistic function}$$

Smooth and Differentiable alternative to sign

{{import numpy as np;import matplotlib.pyplot as plt;x = np.arange(-20.0, 20.0, 0.1);y = 1/(1+np.exp(-x));_=plt.plot(x,y);}}

Logistic Regression - Probabilistic View

We model **conditional probability of** $y \mid x$**:**

$$\begin{cases} p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta}) = f_{\boldsymbol{\theta}} \\ p(y = 0 \mid \mathbf{x}; \boldsymbol{\theta}) = \ ? \end{cases}$$

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \doteq \sigma\left(\boldsymbol{\theta}^T \mathbf{x}\right)$$

where:

$$\sigma(z) = \frac{1}{1 + \exp^{-z}} \quad \text{sigmoid or logistic function}$$

Logistic Regression - Probabilistic View

We model **conditional probability of** $y \mid x$**:**

$$\begin{cases} p(y=1 \mid \mathbf{x}; \boldsymbol{\theta}) = f_{\boldsymbol{\theta}} \\ p(y=0 \mid \mathbf{x}; \boldsymbol{\theta}) = 1 - f_{\boldsymbol{\theta}} \end{cases}$$

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \doteq \sigma\left(\boldsymbol{\theta}^T \mathbf{x}\right)$$

where:

$$\sigma(z) = \frac{1}{1+\exp^{-z}} \quad \text{sigmoid or logistic function}$$

$$\lim_{z \mapsto \infty} \sigma(z) = 1$$

$$\lim_{z \mapsto -\infty} \sigma(z) = 0$$

Logistic Regression - Probabilistic View

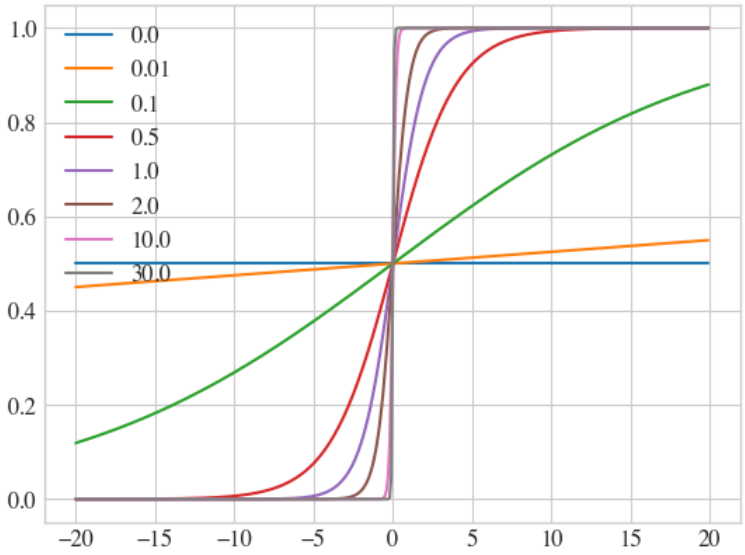
We model **conditional probability of** $y \mid x$**:**

$$\begin{cases} p(y=1 \mid \mathbf{x}; \boldsymbol{\theta}) = f_{\boldsymbol{\theta}} \\ p(y=0 \mid \mathbf{x}; \boldsymbol{\theta}) = 1 - f_{\boldsymbol{\theta}} \end{cases}$$

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \doteq \frac{1}{1+\exp^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

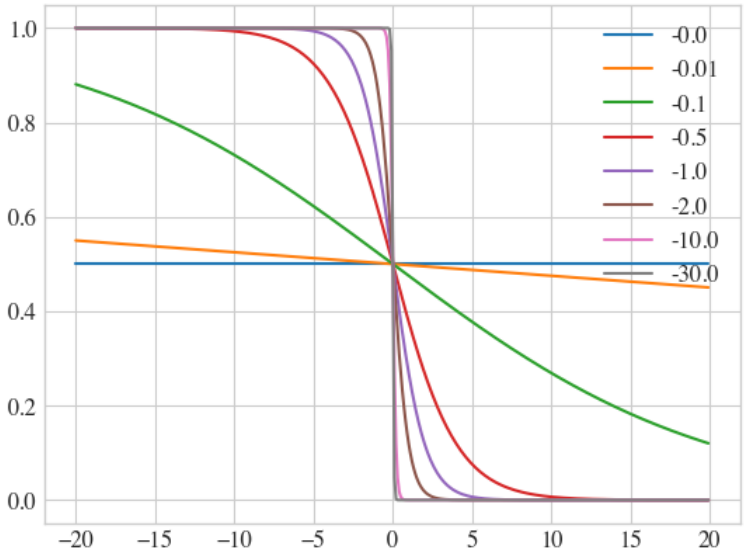
How changing w changes the output of the function

In [125]:
```
import numpy as np;
import matplotlib.pyplot as plt;
x = np.arange(-20.0, 20.0, 0.1);
ws = np.array([0, 0.01,0.1,0.5,1,2,10,30])
for w in ws:
    y = 1/(1+np.exp(-w*x));
    plt.plot(x,y);
plt.legend(ws);
```



How changing $\mathbf{w}$ changes the output of the function

In [26]:
```
import numpy as np;
import matplotlib.pyplot as plt;
x = np.arange(-20.0, 20.0, 0.1);
ws = np.array([0, 0.01,0.1,0.5,1,2,10,30])*-1
for w in ws:
    y = 1/(1+np.exp(-w*x));
    plt.plot(x,y);
plt.legend(ws);
```



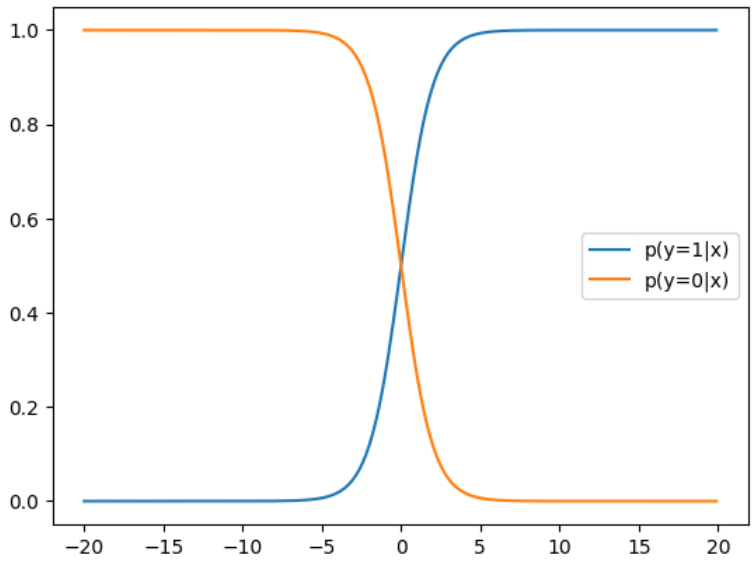
What is the probability of points on the decision boundary?

What is the probability of points on the decision boundary?

Points on the decision boundary: $\theta^T \mathbf{x} = 0$

$$f_{\theta}(\mathbf{x}) \doteq \frac{1}{1+\exp^{-\theta^T \mathbf{x}}} = \frac{1}{1+\exp^{0}} = \frac{1}{1+1} = \frac{1}{2}$$

In [1]:
```
import numpy as np;import matplotlib.pyplot as plt;x = np.arange(-20.0, 20.0, 0.1);y = 1/(1+np.exp(-x));plt.plo
```

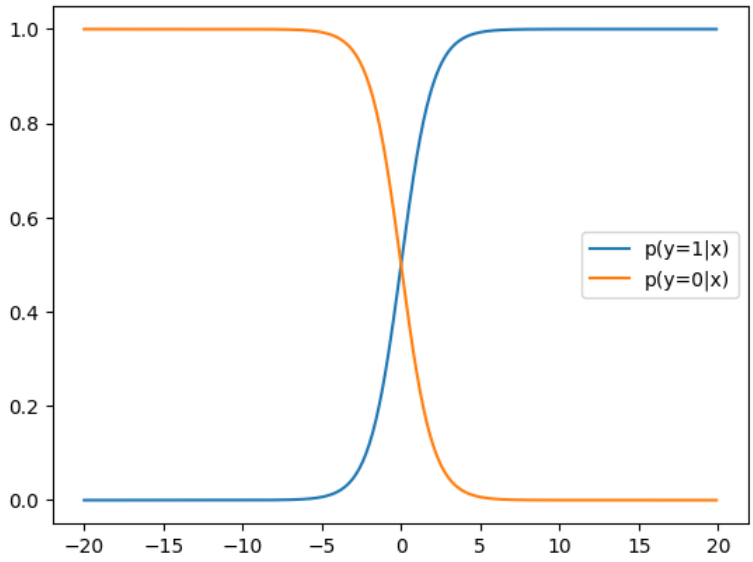


Logistic Regression + threshold for probability = classifier

In general the threshold is **0.5** to map it to a predicted class.

$$\frac{1}{1+\exp^{-\theta^T \mathbf{x}}} > 0.5$$

In [2]:
```
import numpy as np;import matplotlib.pyplot as plt;x = np.arange(-20.0, 20.0, 0.1);y = 1/(1+np.exp(-x));plt.plo
```



Learning Logistic Regression - No closed form solution

We model **conditional probability of $y|x$:**

$$\begin{cases} p(y=1 \mid \mathbf{x}; \boldsymbol{\theta}) = f_{\boldsymbol{\theta}} \\ p(y=0 \mid \mathbf{x}; \boldsymbol{\theta}) = 1 - f_{\boldsymbol{\theta}} \end{cases}$$

For a single point, the `if else` above based on the y can be written compactly as:

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = (f_{\boldsymbol{\theta}})^{y} (1 - f_{\boldsymbol{\theta}})^{1-y}$$

Which discrete distribution does the equation above resemble?

Bernoulli Distribution

$$p(y \mid \mathbf{x}; \boldsymbol{\theta}) = (p(y=1 \mid \mathbf{x}; \boldsymbol{\theta}))^{y} (1 - p(y=1 \mid \mathbf{x}; \boldsymbol{\theta}))^{1-y}$$

Bernoulli Distribution

$$f(k; p) = \begin{cases} p & \text{if } k = 1, \\ q = 1 - p & \text{if } k = 0. \end{cases}$$

Before moving to optimizing Logistic Regression

let's have a "bird's eye" view

Generalized Linear Models (GLM) - $\sigma(\boldsymbol{\theta}^T\mathbf{x})$

Data Type y	Expo. Family	Name/ML Topic
R	Gaussian LaPlace	Regression
$\{0, 1\}$	Bernoulli	Binary Classification
$\{1, K\}$	Categorical	Multi-class Classification
N_+	Poisson	Poisson Regression (Counts)
Categorical	Dirichlet	More advanced Topics

Generalized Linear Models (GLM) - $\sigma(\boldsymbol{\theta}^T\mathbf{x})$

Property of GLM

1. Make a choice on the distribution based on the label data type y (support compatibile to the data type y)
2. Express the chosen distribution with **Exponential Family** *(that we did not cover here)*
   A. We model the conditional distribution $f_{\boldsymbol{\theta}} = \mathrm{E}(y \,|\, x; \theta)$
   B. This will give you your hypothesis $f_{\boldsymbol{\theta}}(\mathbf{x}) \doteq \sigma\left(\boldsymbol{\theta}^T\mathbf{x}\right)$ where σ depends on what you chose before.
3. **Unified GD Update Rule for all GLM:**

$$\theta := \theta + \gamma(y - \sigma(\boldsymbol{\theta}^T\mathbf{x}))\mathbf{x}$$

Data Type y	Expo. Family	Name
R	Gaussian LaPlace	Regression
$\{0, 1\}$	Bernoulli	Binary Classification
$\{1, K\}$	Categorical	Multi-class Classification
N_+	Poisson	Poisson Regression
Categorical	Dirichlet	More advanced Topics

Linear Regression Revisited

Data Type y	Expo. Family	Name
R	Gaussian LaPlace	Regression

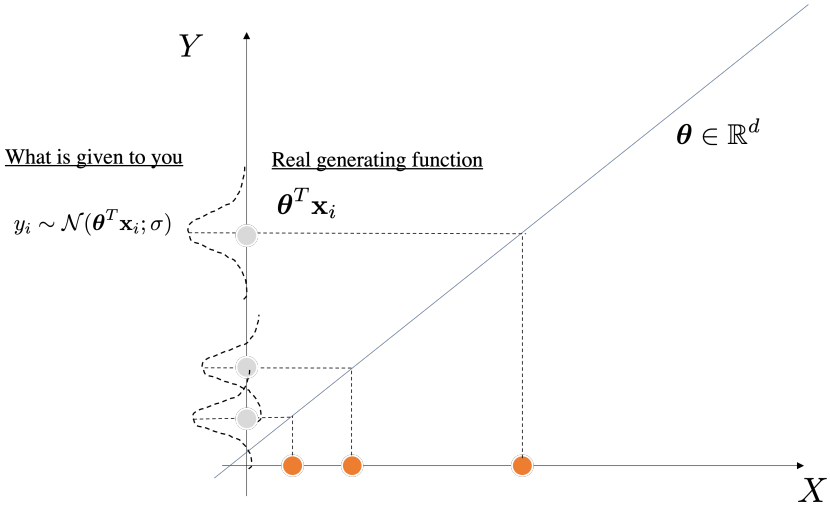
Generation process:

$$y = \mathrm{N}(\boldsymbol{\theta}^T\mathbf{x}; \sigma^2) \;\text{ same as }\; y = \theta^T\mathbf{x} + \epsilon \;\text{ with }\; \epsilon \sim \mathrm{N}(0; \sigma^2)$$

Linear Regression Revisited

Generation process:

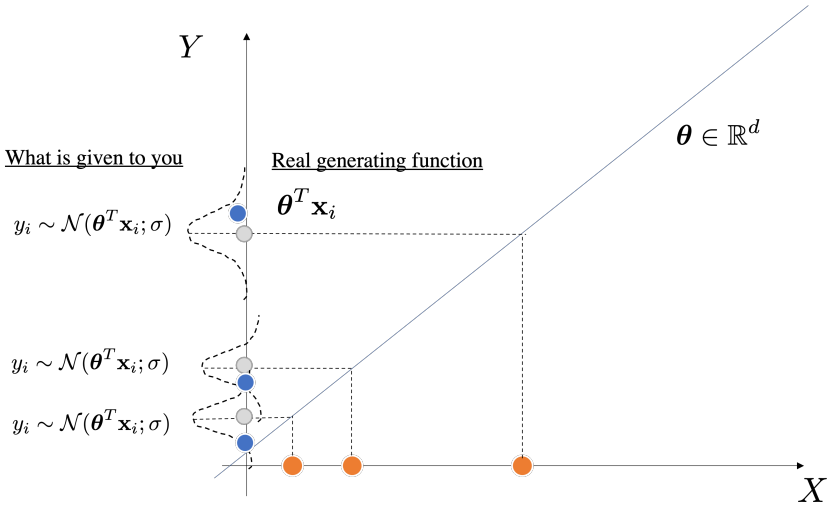
$y = \mathrm{N}(\boldsymbol{\theta}^T\mathbf{x}; \sigma^2)$ same as $y = \theta^T\mathbf{x} + \epsilon$ with $\epsilon \sim \mathrm{N}(0; \sigma^2)$



Linear Regression Revisited

Generation process:

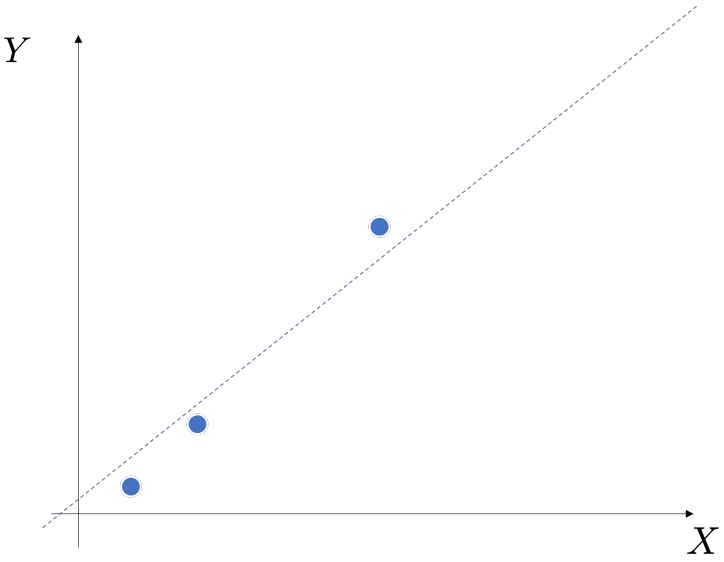
$y = \mathrm{N}(\boldsymbol{\theta}^T\mathbf{x}; \sigma^2)$ same as $y = \theta^T\mathbf{x} + \epsilon$ with $\epsilon \sim \mathrm{N}(0; \sigma^2)$



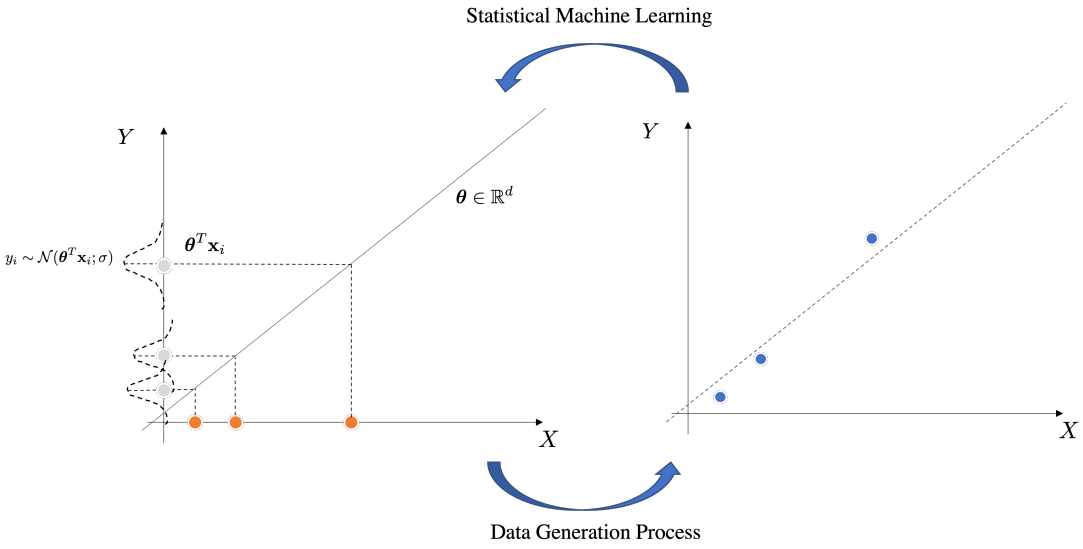
Linear Regression Revisited: What we see when we start

Generation process:

$$y = \mathrm{N}(\boldsymbol{\theta}^T\mathbf{x}; \sigma^2) \text{ same as } y = \theta^T\mathbf{x} + \epsilon \text{ with } \epsilon \sim \mathrm{N}(0; \sigma^2)$$



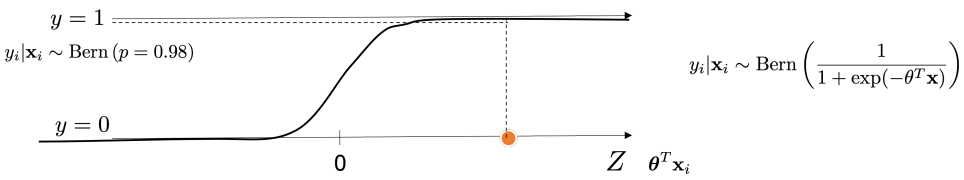
Linear Regression Revisited: What we have to invert

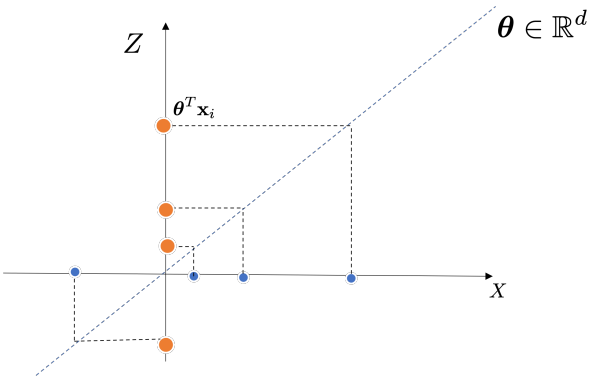


Something similar holds for Logistic Regression

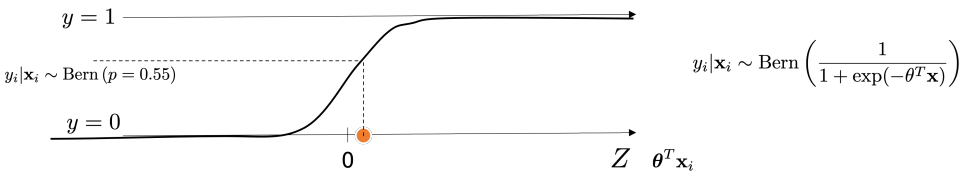
$$y_i \,|\, \mathbf{x}_i \sim \mathrm{Bern}\left(\frac{1}{1 + \exp(-\theta^T\mathbf{x}_i)}\right)$$

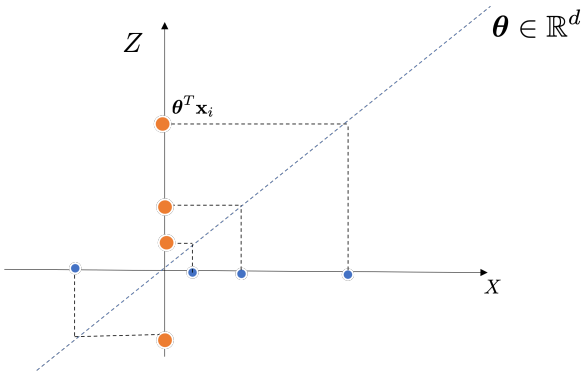
Something similar holds for Logistic Regression



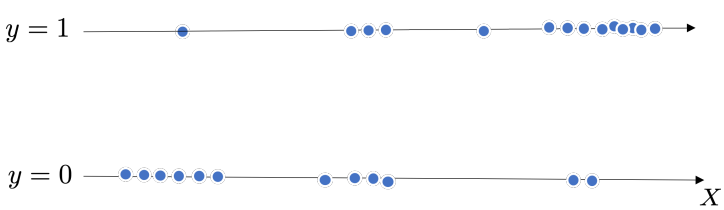


Something similar holds for Logistic Regression





Logistic Regression Data



Learning Logistic Regression - Maximizing Log Likelihood

Learning Logistic Regression - Maximizing Log Likelihood

We model **conditional probability of $y|x$:**

$$\begin{cases} p(y=1|\mathbf{x};\boldsymbol{\theta}) = f_{\boldsymbol{\theta}} \\ p(y=0|\mathbf{x};\boldsymbol{\theta}) = 1 - f_{\boldsymbol{\theta}} \end{cases}$$

For multiple training points, using I.I.D. assumptions:

$$\begin{aligned} L(\theta;\mathbf{X};\mathbf{y}) &= p(\vec{y} \mid X;\theta) = \\ L(\theta;\mathbf{X};\mathbf{y}) &= p(\vec{y} \mid \mathbf{x}_1,\ldots,\mathbf{x}_n;\theta) = [IID] \\ &= \prod_{i=1}^{n} p\left(y^{(i)} \mid x^{(i)};\theta\right) \\ &= \prod_{i=1}^{n} \left(f_{\theta}\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - f_{\theta}\left(x^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned}$$

Learning Logistic Regression - Maximizing Log Likelihood

We get the **Log-loss** that we have seen in the lecture on **evaluating the models**!

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{n} y^{(i)} \log f\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}\right)\right) \end{aligned}$$

Learning Logistic Regression - Gradient of Log Likelihood

We get the **Log-loss** that we have seen in the lecture on **evaluating the models**!

$$\begin{aligned} &\nabla_{\theta} \log L(\theta) = \\ &= \nabla_{\theta} \sum_{i=1}^{n} y^{(i)} \log f_{\theta}\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f_{\theta}\left(x^{(i)}\right)\right) \end{aligned}$$

but before let's compute the gradient of the logistic function.

Gradient/Derivative of Logistic Function

$$\sigma(z) = \frac{1}{1+\exp^{-z}} = (1+\exp^{-z})^{-1}$$

$$\sigma(z)' = -1(1+\exp^{-z})^{-1-1} \cdot (-\exp^{-z})$$

$$\sigma(z)' = \frac{-1}{(1+\exp^{-z})^2} \cdot (-\exp^{-z})$$

$$\sigma(z)' = \frac{-1}{(1+\exp^{-z})} \cdot \frac{-\exp^{-z}}{(1+\exp^{-z})}$$

$$\sigma(z)' = \frac{1}{(1+\exp^{-z})} \cdot \frac{\exp^{-z}}{(1+\exp^{-z})}$$

$$\sigma(z)' = \frac{1}{(1+\exp^{-z})} \cdot \frac{1-1+\exp^{-z}}{(1+\exp^{-z})}$$

$$\sigma(z)' = \frac{1}{(1+\exp^{-z})} \cdot \frac{1+\exp^{-z}-1}{(1+\exp^{-z})}$$

Gradient of Logistic Function

$$\sigma(z)' = \frac{1}{(1+\exp^{-z})} \cdot \frac{1+\exp^{-z}-1}{(1+\exp^{-z})}$$

$$\sigma(z)' = \frac{1}{(1+\exp^{-z})} \cdot \left(1 - \frac{1}{(1+\exp^{-z})}\right)$$

$$\sigma(z)' = \underbrace{\frac{1}{(1+\exp^{-z})}}_{sigmoid} \cdot \left(1 - \underbrace{\frac{1}{(1+\exp^{-z})}}_{sigmoid}\right)$$

$$\sigma(z)' = \sigma(z)(1-\sigma(z))$$

Learning Logistic Regression - Gradient of Log Likelihood

We get the **Log-loss** that we have seen in the lecture on **evaluating the models**!

$$\nabla_\theta \log L(\theta) =$$
$$= \nabla_\theta \sum_{i=1}^{n} y^{(i)} \log f_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-f_\theta\left(x^{(i)}\right)\right)$$

Learning Logistic Regression - Gradient of Log Likelihood

We get the **Log-loss** that we have seen in the lecture on **evaluating the models**!

$$\nabla_\theta \log L(\theta) =$$
$$= \nabla_\theta \sum_{i=1}^{n} y^{(i)} \log[\sigma(\theta^T x^{(i)})] + \left(1-y^{(i)}\right)\log[1-\sigma(\theta^T x^{(i)})]$$

Learning Logistic Regression - Gradient of Log Likelihood

$$\sum_{i=1}^{n} y^{(i)} \frac{1}{\sigma(\theta^T x^{(i)})} \cdot \sigma(\theta^T x^{(i)})' x^{(i)} + (1-y)^{(i)} \left(\frac{1}{1-\sigma(\theta^T x^{(i)})}\right)\left(-\sigma(\theta^T x^{(i)})' x^{(i)}\right)$$

Remember:

$$\sigma(z)' = \sigma(z)(1-\sigma(z))$$

Learning Logistic Regression - Gradient of Log Likelihood

$$\sum_{i=1}^{n} y^{(i)} \frac{1}{\sigma(\theta^T x^{(i)})} \cdot \left(\sigma(\theta^T x^{(i)})(1-\sigma(\theta^T x^{(i)}))\right) x^{(i)} + (1-y)^{(i)} \left(\frac{1}{1-\sigma(\theta^T x^{(i)})}\right)\left(-\left(\sigma(\theta^T x^{(i)})(1-\sigma(\theta^T x^{(i)}))\right) x^{(i)}\right)$$

Learning Logistic Regression - Gradient of Log Likelihood

$$\sum_{i=1}^{n} y^{(i)} \left((1-\sigma(\theta^T x^{(i)}))\right) x^{(i)} + (1-y)^{(i)} \left(-\left(\sigma(\theta^T x^{(i)})\right) x^{(i)}\right)$$

Learning Logistic Regression - Gradient of Log Likelihood

$$\sum_{i=1}^{n}[y^{(i)}\left((1-\sigma(\theta^T x^{(i)}))\right)+(1-y)^{(i)}\left(-\left(\sigma(\theta^T x^{(i)})\right)\right)]x^{(i)}$$

$$\sum_{i=1}^{n}[y^{(i)}-y^{(i)}\sigma(\theta^T x^{(i)})-\sigma(\theta^T x^{(i)})+y^{(i)}\sigma(\theta^T x^{(i)})]x^{(i)}$$

$$\sum_{i=1}^{n}[y^{(i)}-\sigma(\theta^T x^{(i)})]x^{(i)}$$

Gradient Ascent for Logistic Regression since maximizing Log Likelihood

$$\boldsymbol{\theta}:=\boldsymbol{\theta}+\gamma\sum_{i=1}^{n}[y^{(i)}-\sigma(\theta^T x^{(i)})]\mathbf{x^{(i)}}$$

$$\boldsymbol{\theta}:=\boldsymbol{\theta}+\gamma\sum_{i=1}^{n}[y^{(i)}-f_\theta\left(x^{(i)}\right)]\mathbf{x^{(i)}}$$

Same update rule for linear regression but $f_\theta\left(x^{(i)}\right)$ changes!

Because of the property of GLM!

A note on convexity

All latest parametric supervised problems we have discussed are **convex**.

The loss in function of the parameters is convex.

- Least squares, robust regression, logistic regression, Support Vector Machines, multi-class logistic etc.
- All of the above with L2-regularization.

GD gives you a global minimum/maximum on convex functions (optimization is easy)

Logistic Regression vs Least Squares (Linear Regression)

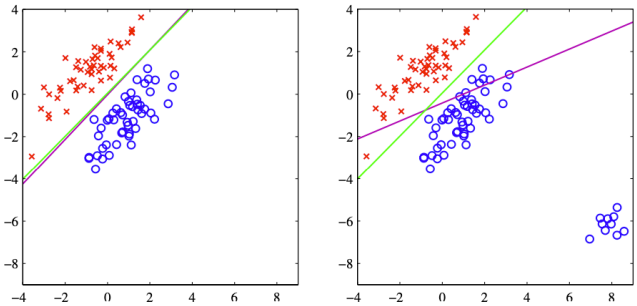
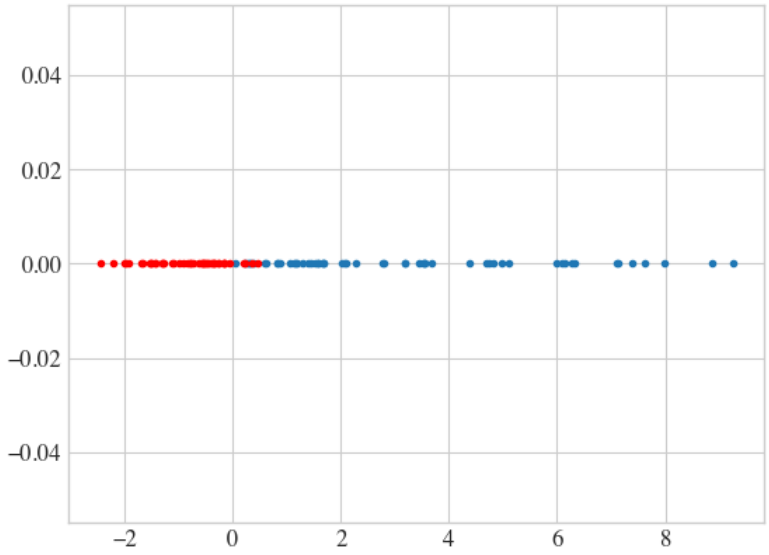
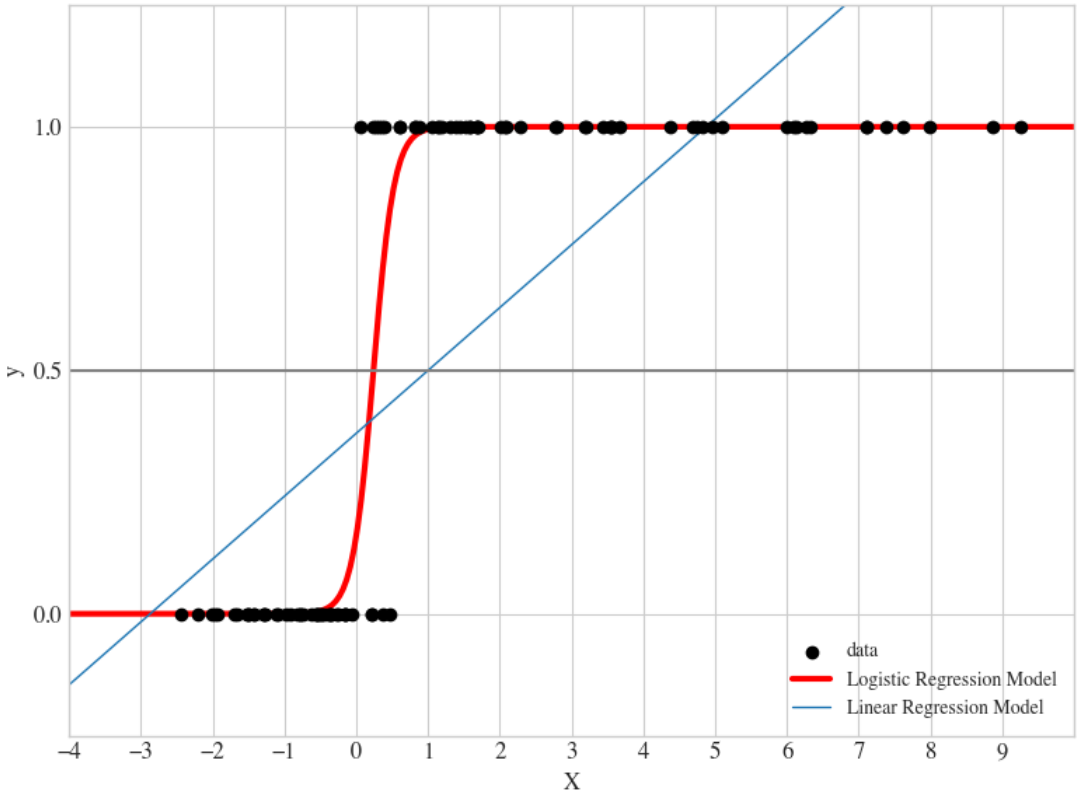


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

In [27]:
```
# Generate a toy dataset, it's just a straight line with some Gaussian noise:
xmin, xmax = -5, 5
n_samples = 100
np.random.seed(0)
X = np.random.normal(size=n_samples)
y = (X > 0).astype(float)
X[X > 0] *= 4
X += 0.3 * np.random.normal(size=n_samples)

X = X[:, np.newaxis]

plt.plot(X[y==1],[0]*sum(y==1),'.')
plt.plot(X[y==0],[0]*sum(y==0),'r.');
```

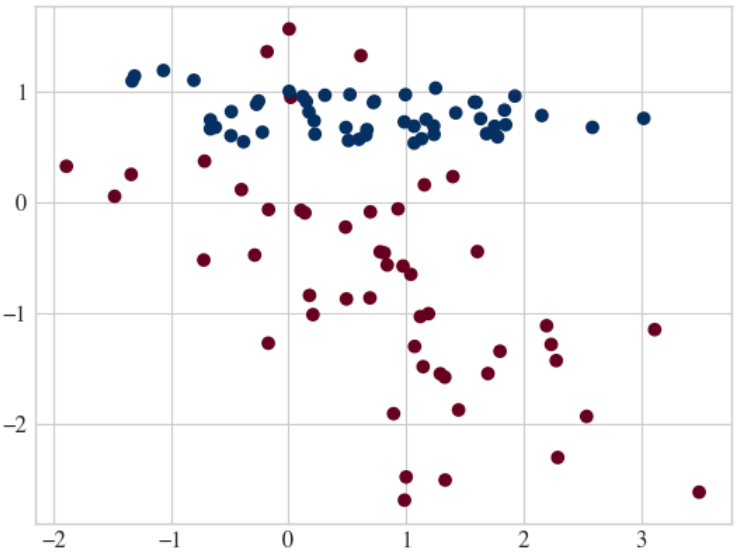


In [28]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression, LinearRegression
from scipy.special import expit



# Fit the classifier
clf = LogisticRegression(C=1e5)
clf.fit(X, y)

# and plot the result
plt.figure(1, figsize=(8, 6))
plt.clf()
plt.scatter(X.ravel(), y, color="black", zorder=20)
X_test = np.linspace(-5, 10, 300)

loss = expit(X_test * clf.coef_ + clf.intercept_).ravel()
plt.plot(X_test, loss, color="red", linewidth=3)

ols = LinearRegression()
ols.fit(X, y)
plt.plot(X_test, ols.coef_ * X_test + ols.intercept_, linewidth=1)
plt.axhline(0.5, color=".5")

plt.ylabel("y")
plt.xlabel("X")
plt.xticks(range(-5, 10))
plt.yticks([0, 0.5, 1])
plt.ylim(-0.25, 1.25)
plt.xlim(-4, 10)
plt.legend(
    ("data","Logistic Regression Model", "Linear Regression Model"),
    loc="lower right",
    fontsize="small",
)
plt.tight_layout()
plt.show()
```



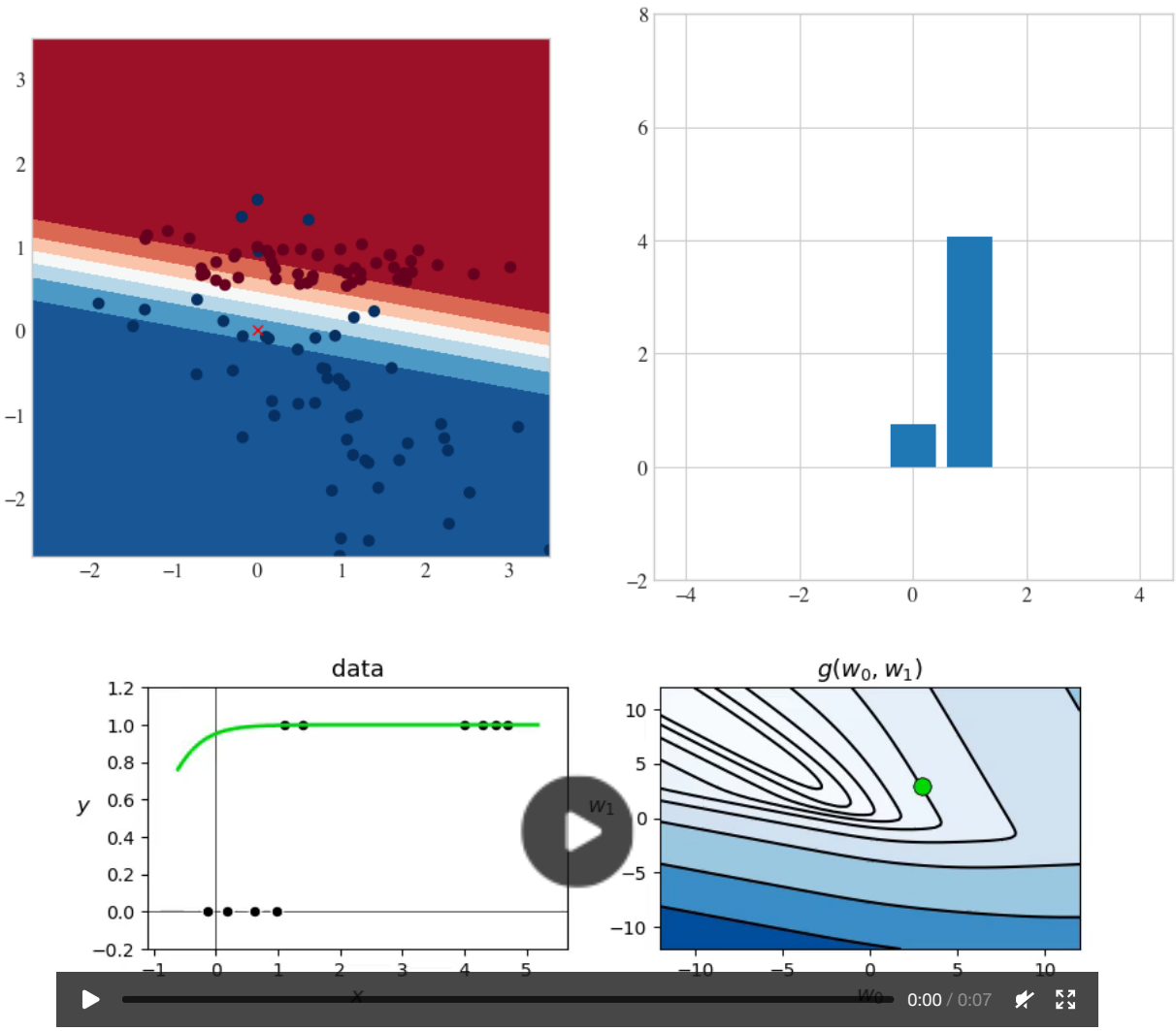
In [29]:
```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_features=2, n_redundant=0, n_informative=2, n_clusters_per_class=1, random_state=0, class_sep=0.8)
```

In [30]:
```
plt.scatter(*X.T,c=y,cmap=cm.get_cmap("RdBu"));
```



$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

In [31]:
```
clf = LogisticRegression(penalty='none')
clf.fit(X, y); minx, maxx = X.min(), X.max()
support = np.linspace(minx, maxx, 100)
xx, yy = np.meshgrid(support, support)
points = np.stack((xx.flatten(), yy.flatten()), axis=1)
prob_mesh = clf.predict_proba(points)
prob_mesh = prob_mesh[:,0].reshape(xx.shape)
fig, axes = plt.subplots(1,2,figsize=(12,6))
axes[0].contourf(xx,yy,prob_mesh,cmap=cm.get_cmap("RdBu"));axes[0].plot(0,0,'rx')
axes[0].scatter(*X.T,c=(1-y),cmap=cm.get_cmap("RdBu"));
axes[0].axis('scaled');axes[1].bar([0,1],clf.coef_[0]);axes[1].axis('equal');axes[1].set_xlim((-5,5));axes[1].s
```



Artificial Intelligence and Machine Learning

Unit II

From Logistic Regression to Softmax Classifier (multi-class)

Information Theory View on Log Loss

Information Theory View on Log Loss

$$\log L(\theta) = \\ = \sum_{i=1}^{n} y^{(i)} \log f\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}\right)\right)$$

Information Theory View on Log Loss

$$\log L(\theta) = \\ = \sum_{i=1}^{n} y^{(i)} \log p\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - p\left(x^{(i)}\right)\right)$$

How do we relate Log Loss with Information Theory?

Information Theory View on Log Loss

- The problem is still binary classification $y \in \{0, 1\}$
- Instead of modeling $y \in \{0, 1\}$ let's say $\vec{\mathbf{y}}$ is a vector of the same dimension of the number of classes.
- $\mathbf{y} \in \{0, 1\}^K$ with $\mathbf{y}$ being a **Categorical distribution** where the mass is all concentrated in the index of the ground-truth label.
  - Only `[1,0]` and `[0,1]` are possible for 2 classes since it has to be a prob. distribution
  - `[1,1]` is not a prob. distribution

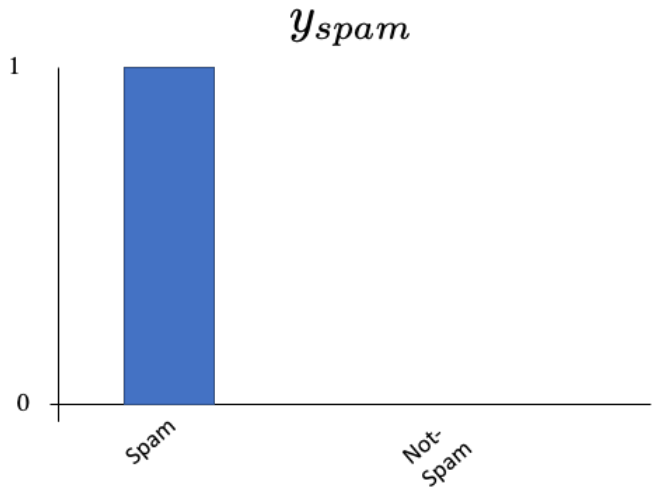
So for a $K = 2$ binary class problem,e.g. (spam vs not-spam) the possible labels could be:

$$y_{spam} = [1, 0]^T$$

$$y_{not-spam} = [0, 1]^T$$

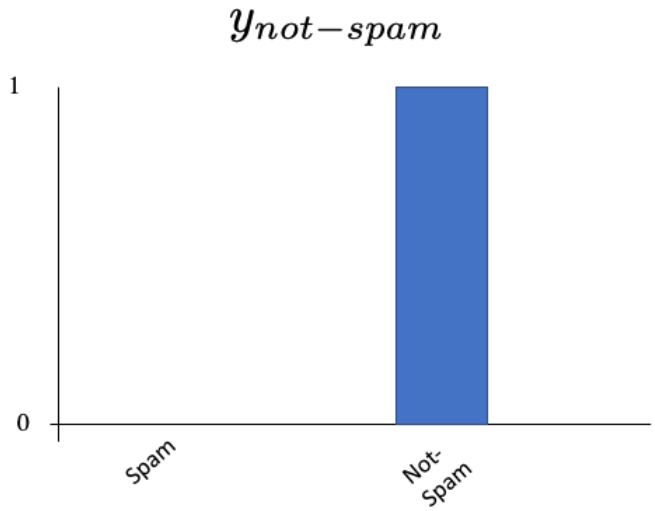
One-Hot Encoding

$$y_{spam} = [1, 0]^T$$



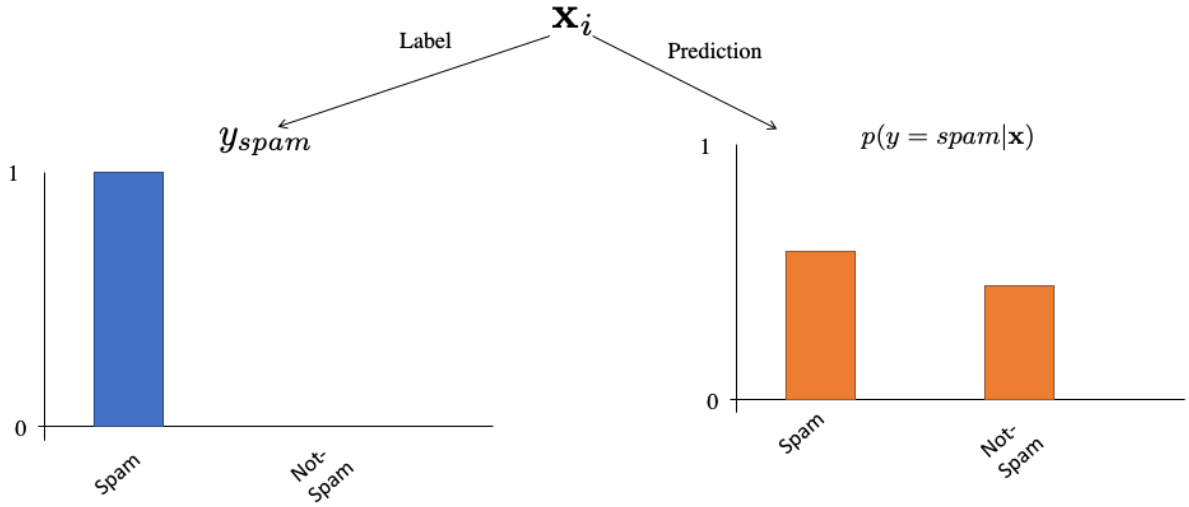
One-Hot Encoding

$$y_{not-spam} = [0, 1]^T$$



Logistic Regression Output vs One-Hot Encoding

Who can tell me how we can compare two *[discrete]* probability distributions?



"Divergence" between two [discrete] distributions

Objective: Estimate a sort of "distance" (or better **divergence**) between two distributions $\boldsymbol{p(x)}$ **vs** $\boldsymbol{q(x)}$.

- We have an ~~**unknown distribution**~~ **distribution from the labels** $p(x) = [1, 0]$
- We want to model it using an **approximating distribution from your model** $q(x) = [0.2, 0.8]$.

If we use $q(x)$ to construct a coding scheme for the purpose of transmitting values of x to a receiver instead of $p(x)$, then the **average additional amount of information** required to specify the value of x as a result of using $q(x)$ instead of the true distribution $p(x)$ is given by:

$$\underbrace{H(P, Q)}_{\text{q not p, so extra}} - \underbrace{H(P)}_{\text{best we can do}}$$

Idea: if you use **q** instead of **p**, but the underlying process is governed by **p**, then you need to pay an extra price in transmission a bit more of information. "The bit more" is the equation above.

Measuring the "divergence" between two distributions

$$H(P, Q) - H(P) = \underbrace{-\sum_{x \in X} p(x) \log q(x)}_{\text{cross-entropy}} - \underbrace{\left(-\sum_{x \in X} p(x) \log p(x)\right)}_{\text{entropy}}$$

Measuring the "divergence" between two distributions

- $p(x) = [1, 0]$ is the **one-hot encoding of the labels**
- $q(x) = [0.2, 0.8]$ is the output from your model. We want q to match p (prediction to match label)

We can use KL divergence!

$$KL(P||Q) = H(P, Q) - H(P) = \underbrace{-\sum_{x \in X} p(x) \log q(x)}_{\text{cross-entropy}} - \underbrace{\left(-\sum_{x \in X} p(x) \log p(x)\right)}_{\text{entropy}}$$

Relative entropy or Kullback-Leibler (KL) divergence

$$KL(P||Q) = \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right)$$

Note that:

- $KL(P||Q) \neq KL(Q||P)$ so it is **NOT a distance metric**, but thankfully the following holds:
- $KL(P||Q) = 0 \longleftrightarrow p = q$

KL divergence reverts back to Cross-Entropy with One-Hot Encoding

Wait a second but....**how much is the entropy of the "labels"** aka entropy of $[0, 1]$ or $[1, 0]$?

$$H(P, Q) - H(P) = \underbrace{-\sum_{x \in X} p(x) \log q(x)}_{\text{cross-entropy}} - \underbrace{\left(-\sum_{x \in X} p(x) \log p(x)\right)}_{\text{entropy of labels is zero}}$$

$$H(P, Q) = \underbrace{-\sum_{x \in X} p(x) \log q(x)}_{\text{cross-entropy}}$$

Training by minimizing cross-entropy (CE)

- $p(y|x) = [1, 0]$ is the **one-hot encoding of the labels**
- $q(y|x) = [0.2, 0.8]$ is the output from your model. We want q to match p.

$$H(P, Q) - H(P) = \underbrace{-\sum_{y \in Y} p(y|x) \log q(y|x)}_{\text{cross-entropy}}$$

$$-1 \cdot \ln(0.2) - 0 \cdot \ln(0.8) = -\ln(0.2) \approx 1.6$$

Information Theory View on Log Loss

Maximizing Log Likelihood - I removed the sum over training samples for clarity

$$\log L(\theta) = y^{(i)} \log p\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - p\left(x^{(i)}\right)\right)$$

Minimizing Cross-Entropy

$$H(P, Q) - H(P) = \underbrace{-\sum_{y \in Y} p(y|x) \log q(y|x)}_{\text{cross-entropy}}$$

$$-\left(p(y = 1) \log q(y = 1|x) + p(y = 0) \log q(y = 0|x)\right)$$

$$-\left(p(y = 1) \log q(y = 1|x) + (1 - p(y = 1)) \log(1 - q(y = 1|x))\right)$$

$$-\left(y \log q(y = 1|x) + (1 - y) \log(1 - q(y = 1|x))\right)$$

Information Theory View on Log Loss

Maximizing Log Likelihood

$$y\log p(x) + (1-y)\log(1-p(x))$$

Minimizing Cross-Entropy

$$-\Big(y\log q(y=1\,|\,x) + (1-y)\log(1-q(y=1\,|\,x))\Big)$$

Which loss, if you have "soft" labels?

Use the "full" KL divergence not cross-entropy!

Does Cross-Entropy works for Multi-Class?

Minimizing cross-entropy (CE) works even in a multi-class case

- $p(y\,|\,x) = [0, 0, 0, 1, 0]$ is the **one-hot encoding of the labels**
- $q(y\,|\,x) = [0.1, 0.1, 0.1, 0.35, 0.35]$ is the output from your model. We want q to match p.

$$H(P, Q) - H(P) = -\sum_{y \in Y} \underbrace{p(y\,|\,x)}\log q(y\,|\,x)$$

cross-entropy

$$-0 \cdot \ln(0.1) - 0 \cdot \ln(0.1) - 0 \cdot \ln(0.1) - 1 \cdot \underbrace{\ln(0.35)} - 0 \cdot \ln(0.35) = -1 \cdot \ln(0.35) \approx 1.04$$

only this matter

One-hot encoding is a selector!

- $p(y\,|\,x) = [0, 0, 0, 1, 0]$ is the **one-hot encoding of the labels** works as a selector of probability of the model!
- Of the probabilities returned by the model select that for which the index gt corresponds to the `1` in the label.

$$\mathrm{L} = -\log q(y\,|\,x)_{gt}$$

- **This is the loss function used to train Neural Networks for Multi-Class classification (image recognition).**

Todays lecture

Supervised, Parametric Models

0) More Sample Questions similar to the exam

1) Multi-class Classification

2) SoftMax Regression (for multi-class classification)

I am still deciding but probably 3 types of questions

1. Simple **exercises** like the ones we saw for Decision Trees, KNN etc. but also on other methods (SVM, linear regression, SoftMax etc.)
2. **Definitions + Proof Sketch** like the ones that we show in class.
3. Answering critical open questions but you have to show **some kind of math or graph/plot sketch behind your rationale**

- Still deciding about adding `multiple choice questions`
- Questions will be spread over the entire program *(math, norms, clustering, supervised non-parametric, parametric models, neural nets etc)*.

The questions will be of increasing difficulties and calibrated to the test time.

K-NN

Given the following training data for a $\{0, 1\}$ binary classifier

- $(x_1 = 0.8; y_1 = 1)$
- $(x_2 = 0.4; y_2 = 0)$
- $(x_3 = 0.6; y_3 = 1)$

Determine the output of a **K Nearest Neighbour (K-NN) classifier** for all points on the interval $0 \leq x \leq 1$ using:

- 1-NN
- 3-NN

K-NN

Given the following training data for a $\{0, 1\}$ binary classifier

- $(x_1 = 0.8; y_1 = 1)$
- $(x_2 = 0.4; y_2 = 0)$
- $(x_3 = 0.6; y_3 = 1)$

1-NN

K-NN

Given the following training data for a $\{0, 1\}$ binary classifier

- $(x_1 = 0.8; y_1 = 1)$
- $(x_2 = 0.4; y_2 = 0)$
- $(x_3 = 0.6; y_3 = 1)$

3-NN

Do we have actually to compute it?

Will be majority vote of class labels since we classify using all training points. **Aka, we have two "1" and one "0". So always 1!**

K-NN - Do it yourself

A regressor algorithm is defined using the mean of the **K Nearest Neighbours** of a test point. Determine the ouput on the interval $0 \leq x \leq 1$ using the training data in the previous example for $K = 2$.

Definitions + Proof Sketch seen in the lectures

- Define what is the entropy of a discrete random variable.
- How do you compare two discrete probability distributions?
- Show that minimizing the entropy of a distribution P is the same as finding a function of P that is as far as possible from another distribution Q that is **uniform distribution across K classes**?

Learning a decision tree is about reducing impurity

- So we want to reduce 1) Misclassification or 2)Gini Impurity or 3) **? What else ?**
- **Know your enemy**: the uniform distributions $\mathbf{x} \sim \mathrm{U}$
- If we have K classes then $\{q_1, \ldots, q_k\} = \{1/K, \ldots, 1/K\}$

We want find a function $f(p)$ so that $f(p)$ is very distant from uniform distribution Q, so we can use KL divergence for this:

$$KL(P||Q) = \sum_{k=1}^{K} p(x)\log\Big(\frac{p(x)}{1/K}\Big) = \sum_{k=1}^{K} p(x)\log\big(p(x)\big) - p(x)\log\big(1/K\big)$$

Critical Questions (Example on K-means)

You and your best friend found a trick to save bandwidth over the internet. Instead of sharing a total of $N = 10M$ grayscale images directly over the Internet, you took the N images and learned $K = 1000$ centroids $\{\mu_1, \ldots, \mu_K\}$ using **K-means**. Each input image is defined over the set $\mathbf{x} \in [0, 255]^{H \times W \times 1}$. Then you shared once for all the K centroids $\{\mu_k\}_{k=1}^{K}$ with your friend.

- Every time you want to send one of the N images $\mathbf{x}$, you send your friend the **associated assignment** y found by K-means.
- Along with y, you also send the **Euclidean distance** d between the image $\mathbf{x}$ and the associated centroid.

Critical Questions (Example on K-means)

Easy Questions: (check if you know the basic notions)

- What is the dimensionality of μ_k? Is it equal for all centroids?
- What is the dimensionality of the assignment y associated to $\mathbf{x}$.
- Which values the assignment y can take?

More Difficult Questions: (check if you can reason on problems)

- Does your friend have a way to restore the original image $\mathbf{x}$, given d and y?
- If yes, how he/she can do it? If no, can she/he retrieve a set of images in which $\mathbf{x}$ could be inside?
- How many parameters (scalar numbers) you have to pass if you share an image $\mathbf{x}$? What about sharing d and y?

You send y associated to the image $\mathbf{x}$, so y takes value in $[1, K]$ and you friend can use y to select the centroid nearest to the point $\mathbf{x}$.

In fact, the point $\mathbf{x}$ was clustered in the cluster given by y.

So one could stop here and say that the best approximation for $\mathbf{x}$ is μ_y ($\mathbf{x} \approx \mu_y$) but we can do more.

Given that we have the scalar euclidean distance between $\mathbf{x}$ and the associated centroid aka μ_y we have:

$$d = ||\mathbf{x} - \mu_y||_2$$

We have only d and μ_y, the image $\mathbf{x}$ that was sent is in the set:

$$\{\forall\, \mathbf{v} \in [0, 255]^{H \times W \times 1} : ||\mathbf{v} - \mu_y||_2 = d\}$$

which is all the points in the hyper-sphere in $[0, 255]^{H \times W \times 1}$ of radius d and centered at μ_y

Critical Questions (Example on Norms) - A bit harder

You are hired by a famous company that has to compute ℓ_2^2 norm over two feature vectors $\mathbf{x} \in \mathrm{R}^D$ and $\mathbf{y} \in \mathrm{R}^D$.

The company works with a machine learning algorithm that **guarantees** that the features $\mathbf{x}$ and $\mathbf{y}$ lie on a unit hyper-sphere (_Hint: same as $||x||_2^2 = 1$)_

$$d^2 = ||\mathbf{x} - \mathbf{y}||_2^2 = \sum_{i=1}^{D} (x_i - y_i)^2$$

- Can you get away computing this without computing the squared difference, i.e. avoiding computing $(\diamond - \diamond)^2$?
- What is the minimum and **maximum** value that d can take?

(Hint: develop the definition of the norm written above)

$$\sum_{i=1}^{D} (x_i - y_i)^2 = \sum_{i=1}^{D} x_i^2 + y_i^2 - 2x_i \cdot y_i = \sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} y_i^2 - 2\sum_{i=1}^{D} x_i \cdot y_i =$$

$$||\mathbf{x}||_2^2 + ||\mathbf{y}||_2^2 - 2\mathbf{x}^T\mathbf{y}$$

Using unit hyper-sphere:

$$1 + 1 - 2\mathbf{x}^T\mathbf{y} = 2 - 2\mathbf{x}^T\mathbf{y}$$

So, yes, they can simply computing the dot product multiply by -2 and add 2.

- Minimum value of d is zero since ℓ_2 is bounded by zero.
- Maximum is when vectors are parallel but facing opposite directions, in this case $\mathbf{x}^T\mathbf{y} = -1$ thus $d^2 = 2 - 2(-1) = 4 \implies d = 2$, which makes sense because two vectors on the unit hyper-sphere are at maximum distance when match the diameter.

Multi-class Classification

Multi-class Classification as GLM

Data Type y	Expo. Family	Name
$\{1, K\}$	Categorical	Multi-class Classification

Multi-Class: One vs Rest (left) and One vs One (right)

Regions of the space remains unclassified

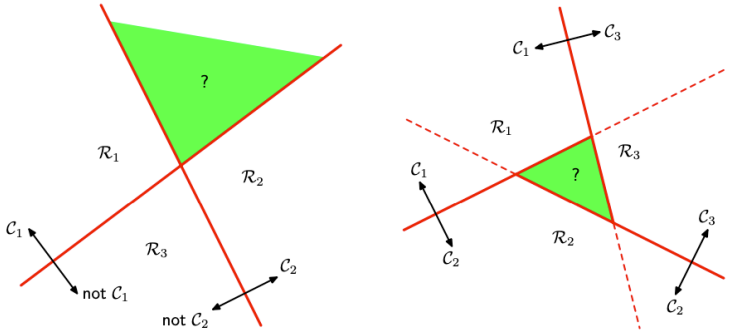
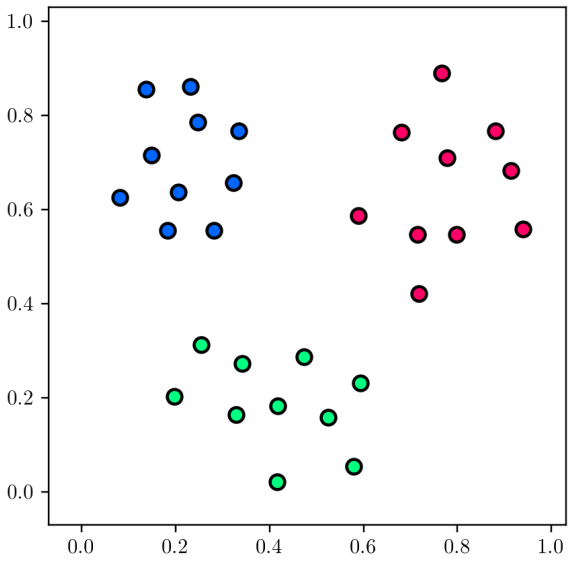
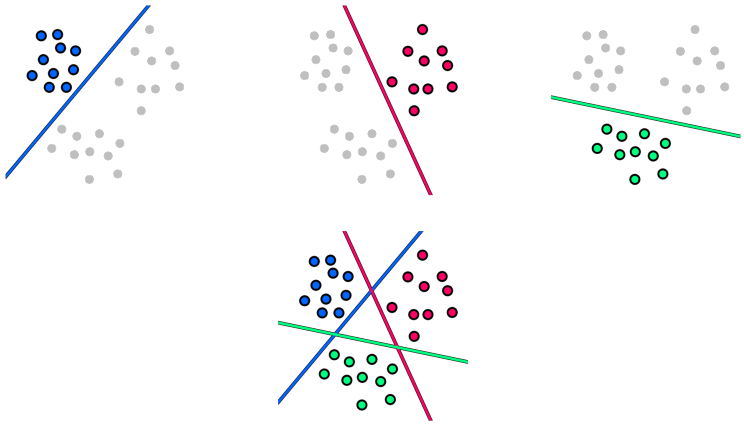


Figure 4.2 Attempting to construct a K class discriminant from a set of two class discriminants leads to ambiguous regions, shown in green. On the left is an example involving the use of two discriminants designed to distinguish points in class $\mathcal{C}_k$ from points not in class $\mathcal{C}_k$. On the right is an example involving three discriminant functions each of which is used to separate a pair of classes $\mathcal{C}_k$ and $\mathcal{C}_j$.

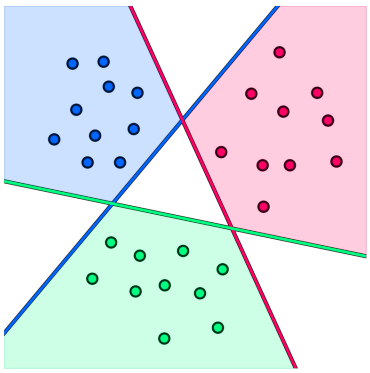
Multi-class classifier via One vs Rest



Multi-class via K One vs Rest Binary Classifers



Multi-class via K One vs Rest Binary Classifers

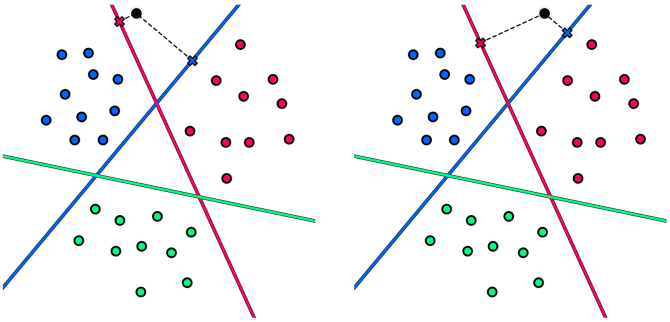


One vs Rest Binary Classifers - Positive Overlap

$$\mathbf{w}_1^T\mathbf{x} + b_1 > 0 \quad \text{and} \quad \mathbf{w}_3^T\mathbf{x} + b_3 > 0$$

$\mathbf{x}$ is assigned to point with higher distance to the hyper-plane (because it is more confident to be that class).

- Left case: blue class; Right case: red class

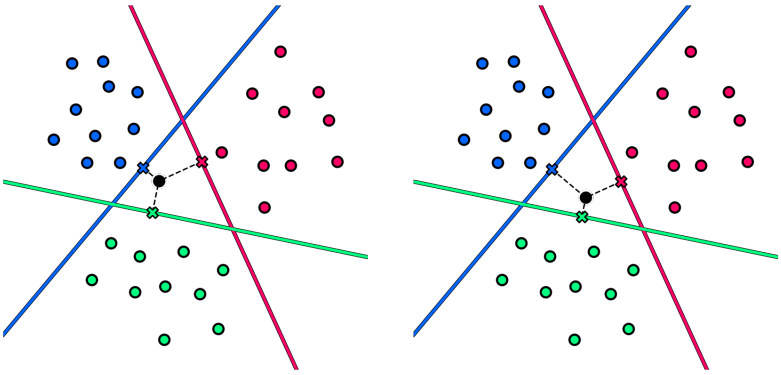


One vs Rest Binary Classifers - Negative Region

$$\mathbf{w}_j^T\mathbf{x} + b_j < 0 \quad \forall j \in \{1, \ldots, K\}$$

$\mathbf{x}$ is assigned to point with **lower** distance to the hyper-plane (because it the least unsure).

- Left case: blue class; Right case: green class

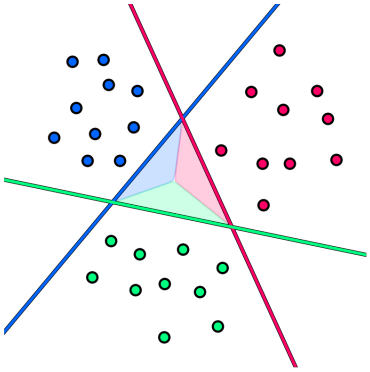


One vs Rest Binary Classifers - Negative Region

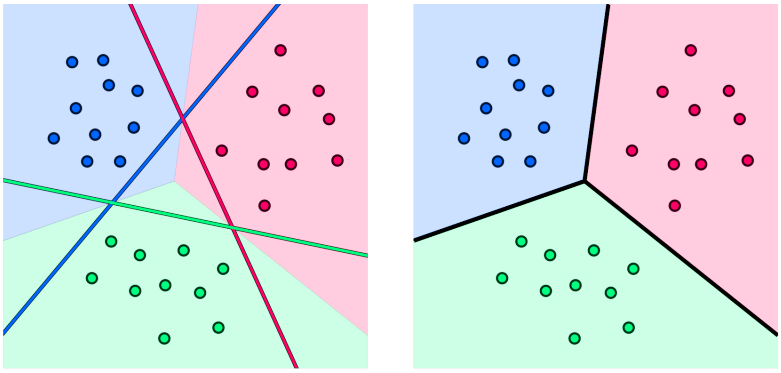
$$\mathbf{w}_j^T\mathbf{x} + b_j < 0 \quad \forall j \in \{1, \ldots, K\}$$

$\mathbf{x}$ is assigned to point with **lower** distance to the hyper-plane (because it the least unsure).

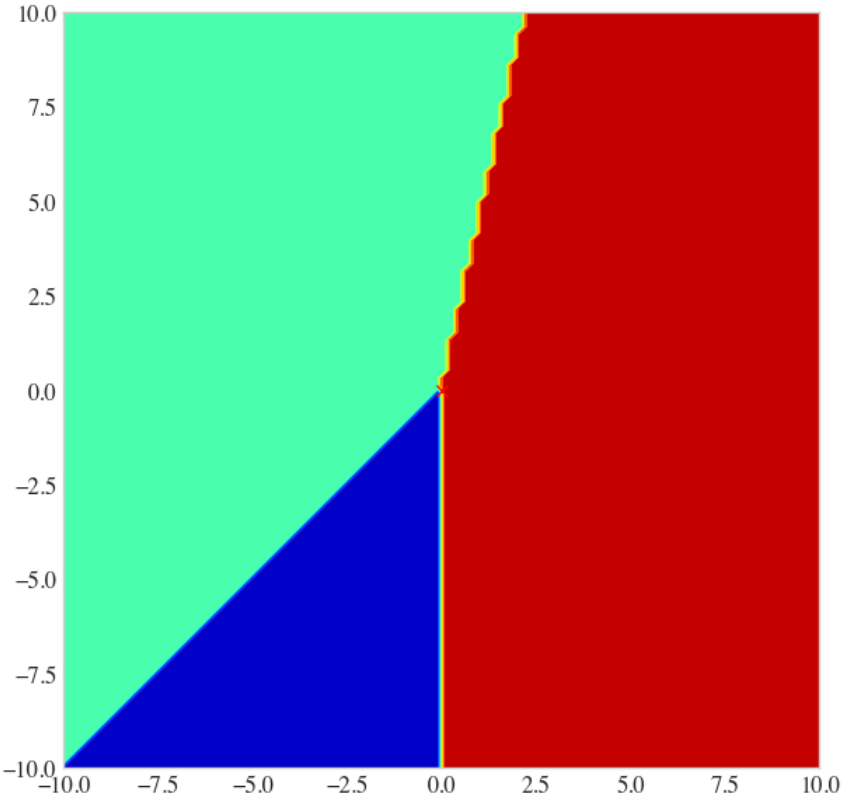
- Left case: blue class; Right case: green class



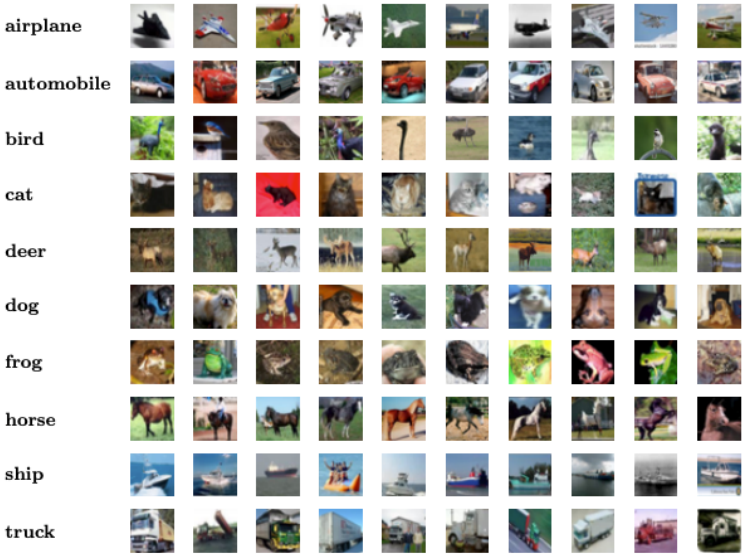
How do we go from left to right?



In [32]:
```
import numpy as np
ww = [[1, 2], [-1, 4], [8, 2]]
support = np.linspace(-10, 10, 100)
xx, yy = np.meshgrid(support, support)
W = np.array(ww)
dim = xx.shape
points = np.stack((xx.flatten(), yy.flatten()), axis=1)
dist = np.argmax((W@points.T), axis=0)
dist = dist.reshape(dim)
plt.figure(figsize=(7,7))
plt.contourf(xx, yy, dist, cmap='jet')
plt.plot(0, 0, 'rx')
plt.axis('scaled')
#plt.colorbar()
plt.xlim(-10, 10);
```



Multi-Class Classification Example: CIFAR-10



From Binary to Multi-Class

Binary case:

$$\text{sign}\left(\mathbf{w}^T\mathbf{x}_i + b\right) > 0$$

$$\text{sign}(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{if } z < 0 \end{cases}$$

let's consider the distance as:

We want:

$$\begin{cases} \mathbf{w}^T\mathbf{x} + b > 0 & y = +1 \\ \mathbf{w}^T\mathbf{x} + b < 0 & y = -1 \end{cases}$$

Multi-Class Decision Boundary

Decision boundary $\{\mathbf{w}_k, b_k\}_{k=1}^K$ per class given K classes.

$$\begin{aligned} f_{\{\mathbf{w}_1, b_1\}} &= \mathbf{w}_1^T\mathbf{x} + b_1 \quad y = 1 \\ f_{\{\mathbf{w}_2, b_2\}} &= \mathbf{w}_2^T\mathbf{x} + b_2 \quad y = 2 \\ &\dots \\ f_{\{\mathbf{w}_K, b_K\}} &= \mathbf{w}_K^T\mathbf{x} + b_K \quad y = K \end{aligned}$$

Multi-Class Decision Boundary

Decision boundary $\{\mathbf{w}_k, b_k\}_{k=1}^K$ per class given K classes.

A data point $\mathbf{x}^\star$ is assigned to class $\hat{k}$ iff:

$$f_{\{\mathbf{w}_{\hat{k}}, b_{\hat{k}}\}}\left(\mathbf{x}^\star\right) > f_{\{\mathbf{w}_j, b_j\}}\left(\mathbf{x}^\star\right) \quad \forall j \in \{1, \dots K\} \setminus \{\hat{k}\}$$

Decision Boundary between 2 classes: *j vs k*

$$f_{\{\mathbf{w}_k, b_k\}}\left(\mathbf{x}^\star\right) = f_{\{\mathbf{w}_j, b_j\}}\left(\mathbf{x}^\star\right)$$

let's write it explicitly

$$\mathbf{w}_k^T\mathbf{x} + b_k = \mathbf{w}_j^T\mathbf{x} + b_j$$

$$(\mathbf{w}_k - \mathbf{w}_j)^T\mathbf{x} + (b_k - b_j) = 0$$

This has the same form as the decision boundary for the two-class case discussed before.

Decision Boundary Convexity

The decision regions of such a discriminant are always singly connected and convex.

Assuming $\mathbf{x}_A$ and $\mathbf{x}_B$ are in R_k:

$$\begin{aligned} (\mathbf{w}_k - \mathbf{w}_j)^T\mathbf{x}_A + (b_k - b_j) > 0 \quad \forall j \neq k \\ (\mathbf{w}_k - \mathbf{w}_j)^T\mathbf{x}_B + (b_k - b_j) > 0 \quad \forall j \neq k \end{aligned}$$

Decision Boundary Convexity

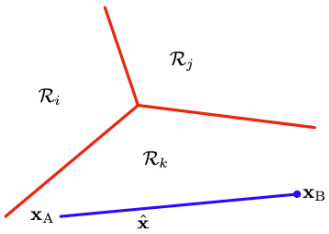
and also *(definition of convexity)*:

$$\hat{\mathbf{x}} = \lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B \quad \text{with} \quad 0 \leq \lambda \leq 1$$

Then also $\hat{\mathbf{x}}$ is in R_k, where $f_k(\cdot)$ is the hyper-plane:

$$f_k(\hat{\mathbf{x}}) > f_j(\hat{\mathbf{x}}) \quad \forall j \neq k$$

Figure 4.3 Illustration of the decision regions for a multiclass linear discriminant, with the decision boundaries shown in red. If two points $\mathbf{x}_\mathrm{A}$ and $\mathbf{x}_\mathrm{B}$ both lie inside the same decision region $\mathcal{R}_k$, then any point $\hat{\mathbf{x}}$ that lies on the line connecting these two points must also lie in $\mathcal{R}_k$, and hence the decision region must be singly connected and convex.



Proof Sketch

We have to show that:

$$f_k(\hat{\mathbf{x}}) > f_j(\hat{\mathbf{x}}) \quad \forall j \neq k$$

and we know that $f_k(\mathbf{x}_A) > f_j(\mathbf{x}_A)$ and $f_k(\mathbf{x}_B) > f_j(\mathbf{x}_B) \;\; \forall j \neq k$ holds.

We apply the definition and use linearity of hyper-plane (dot product):

$$\hat{\mathbf{x}} = \lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B \quad \text{with} \quad 0 \leq \lambda \leq 1$$

$$f_k(\hat{\mathbf{x}}) = f_k\Big(\lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B\Big) = \lambda f_k(\mathbf{x}_A) + (1-\lambda) f_k(\mathbf{x}_B)$$

Proof Sketch

Same hold for:

$$f_j(\hat{\mathbf{x}}) = f_k\Big(\lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B\Big) = \lambda f_j(\mathbf{x}_A) + (1-\lambda) f_j(\mathbf{x}_B)$$

Now this holds by assumption

$$f_k(\mathbf{x}_A) > f_j(\mathbf{x}_A) \quad \forall j \neq k$$

given that $f_k(\mathbf{x}_B) > f_j(\mathbf{x}_B)$, it must be:

$$f_k(\mathbf{x}_A) + f_k(\mathbf{x}_B) > f_j(\mathbf{x}_A) + f_j(\mathbf{x}_B) \quad \forall j \neq k$$

We have to prove:

$$f_k(\hat{\mathbf{x}}) = f_k\Big(\lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B\Big) = \lambda f_k(\lambda \mathbf{x}_A) + (1-\lambda) f_k(\lambda \mathbf{x}_B)$$

We arrived here:

$$f_k(\mathbf{x}_A) + f_k(\mathbf{x}_B) > f_j(\mathbf{x}_A) + f_j(\mathbf{x}_B) \quad \forall j \neq k$$

Remember $0 \leq \lambda \leq 1$ thus both $\lambda \geq 0$ and $(1-\lambda) \geq 0$, thus:

$$\lambda f_k(\mathbf{x}_A) + \underbrace{(1-\lambda) f_k(\mathbf{x}_B)}_{f_k(\hat{\mathbf{x}})} > \lambda f_j(\mathbf{x}_A) + \underbrace{(1-\lambda) f_j(\mathbf{x}_B)}_{f_j(\hat{\mathbf{x}})} \quad \forall j \neq k$$

Thus convex linear combination $\hat{\mathbf{x}}$ still satisfy the same decision boundary:

$$f_k(\hat{\mathbf{x}}) > f_j(\hat{\mathbf{x}}) \quad \forall j \neq k$$

Multi-Class: $\mathbf{W}$, $\mathbf{b}$. From a vector to a matrix

Decision boundary $\{\mathbf{w}_k, b_k\}_{k=1}^K$ per class given K classes.

$$
\begin{aligned}
f_{\{\mathbf{w}_1, b_1\}} &= \mathbf{w}_1^T \mathbf{x} + b_1 \quad y = 1 \\
f_{\{\mathbf{w}_2, b_2\}} &= \mathbf{w}_2^T \mathbf{x} + b_2 \quad y = 2 \\
&\dots \\
f_{\{\mathbf{w}_K, b_K\}} &= \mathbf{w}_K^T \mathbf{x} + b_K \quad y = K
\end{aligned}
$$

Multi-Class: $\mathbf{W}$, $\mathbf{b}$. From a vector to a matrix

Decision boundary $\{\mathbf{w}_k, b_k\}_{k=1}^K$ per class given K classes.

We can model directly everything with a matrix:

$$
\underbrace{\mathbf{y}}_{\mathrm{R}^{Kx1}} = \underbrace{\mathbf{W}}_{\mathrm{R}^{K \times d}} \underbrace{\mathbf{x}}_{\mathrm{R}^{d \times 1}} + \underbrace{\mathbf{b}}_{\mathrm{R}^{K}}
$$

What if you want to classify n points directly, not just one?

Will this work?

$$
\underbrace{\mathbf{Y}}_{\mathrm{R}^{Kxn}} = \underbrace{\mathbf{W}}_{\mathrm{R}^{K \times d}} \underbrace{\mathbf{X}}_{\mathrm{R}^{d \times n}} + \underbrace{\mathbf{b}}_{\mathrm{R}^{K}}
$$

What if you want to classify n points directly, not just one?

OK there is broadcasting on $\mathbf{b}$!

Can I just do

```
y = np.argmax(W@points.T, axis=0)
```

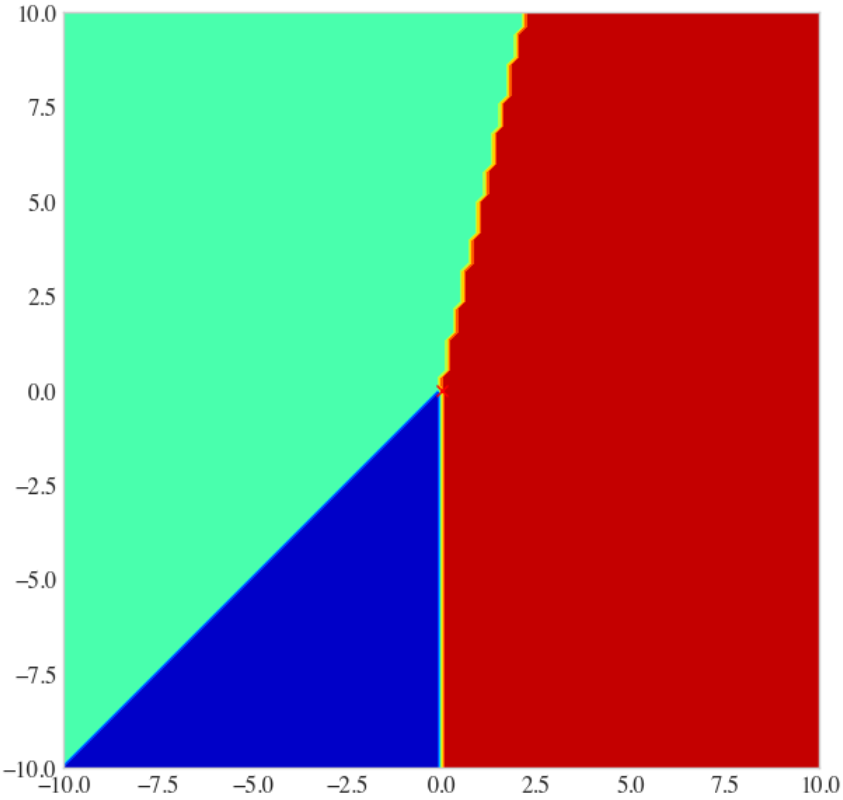
if I want to train a model?

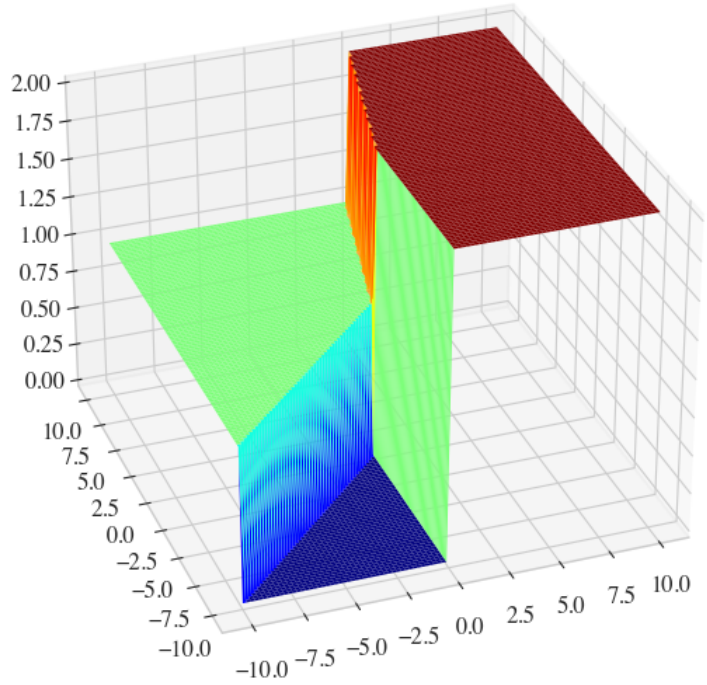
$$
\arg\max_{\mathbf{W}, \mathbf{b}} \left[\arg\max_{k} \left(\mathbf{W}\mathbf{X} + \mathbf{b} \right) \right]
$$

In other words: is `argmax` itself differentiable?

$$
\nabla_{\mathbf{W}, \mathbf{b}} \arg\max_{k} \left(\mathbf{W}\mathbf{X} + \mathbf{b} \right)
$$

In [33]:
```
import numpy as np
from scipy.special import softmax
_softmax = False
ww = [[1, 2], [-1, 4], [8, 2]];support = np.linspace(-10, 10, 100)
xx, yy = np.meshgrid(support, support)
W = np.array(ww)
dim = xx.shape
points = np.stack((xx.flatten(), yy.flatten()), axis=1)
prob = np.argmax(W@points.T, axis=0) if not _softmax else np.max(np.tile(
    np.arange(0, 3).T, (points.shape[0], 1)).T*softmax(0.2*W@points.T, axis=0), axis=0)
prob = prob.reshape(dim)
plt.figure(figsize=(7, 7));
plt.contourf(xx, yy, prob, cmap='jet');plt.plot(0, 0, 'rx');plt.axis('scaled');plt.xlim(-10, 10);plt.figure(fig
ax = plt.axes(projection='3d');ax.plot_surface(xx, yy, prob,rstride=1, cstride=1,cmap='jet', edgecolor='none');
```





In other words: is `argmax` itself differentiable? Nope!

NO! we cannot get gradients from a step function with discontinuities

The problem is not the discontinuities, the problem is that gradients are zero almost everywhere!

$$\nabla_{\mathbf{W},\mathbf{b}} \arg\max_k \left(\mathbf{WX} + \mathbf{b}\right)$$

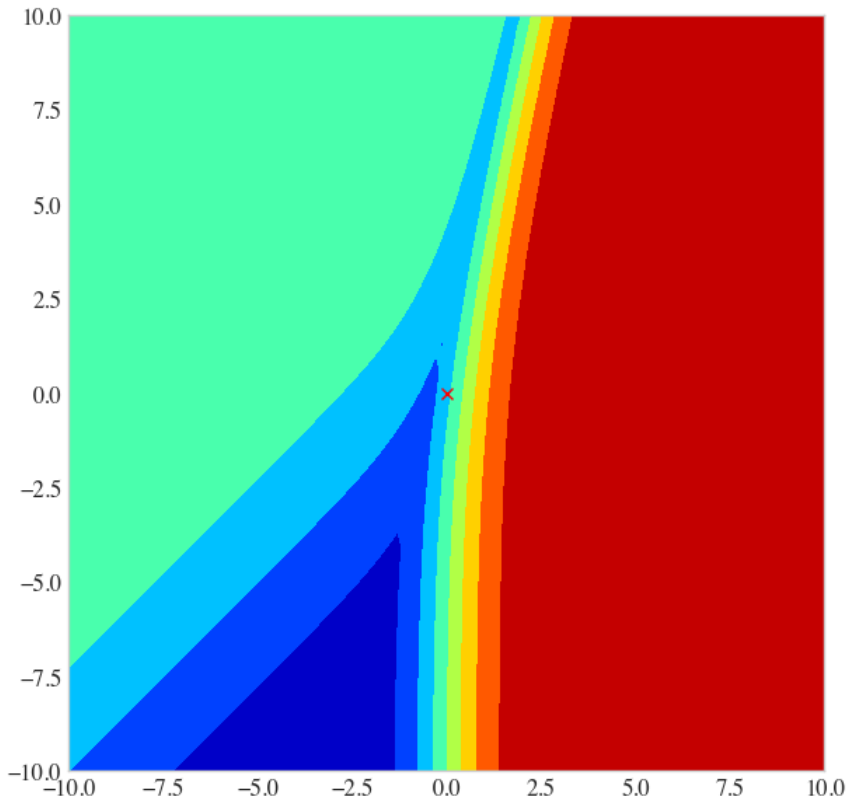
Are we doomed?

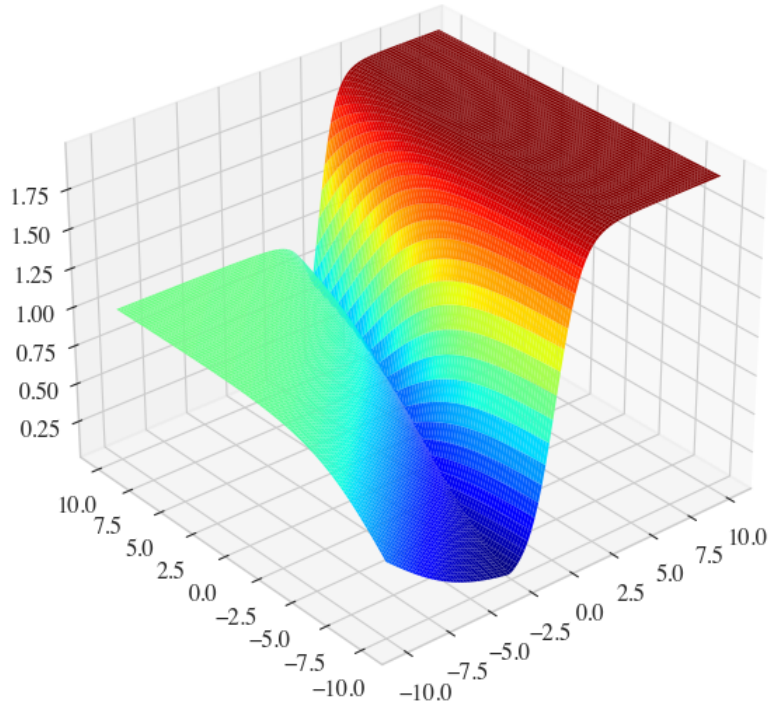
We replace `argmax` with a smooth differentiable version called ~~Softargmax~~ SoftMax

> SoftMax is rather a smooth approximation to the arg max function: the function whose value is which index has the maximum, modeled as One-Hot Encoding

Think Softmax as a differentiable selector!

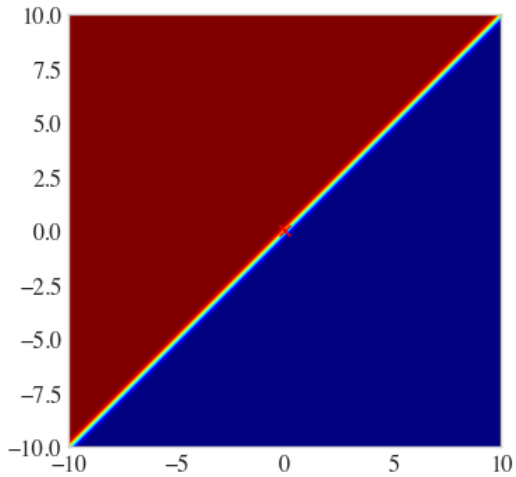
In [34]:
```
import numpy as np
from scipy.special import softmax
_softmax = True
ww = [[1, 2], [-1, 4], [8, 2]]
support = np.linspace(-10, 10, 100)
xx, yy = np.meshgrid(support, support)
W = np.array(ww)
dim = xx.shape
points = np.stack((xx.flatten(), yy.flatten()), axis=1)
prob = np.argmax(W@points.T, axis=0) if not _softmax else np.max(np.tile(
    np.arange(0, 3).T, (points.shape[0], 1)).T*softmax(0.2*W@points.T, axis=0), axis=0)
prob = prob.reshape(dim)
plt.figure(figsize=(7, 7))
plt.contourf(xx, yy, prob, cmap='jet')
plt.plot(0, 0, 'rx')
plt.axis('scaled')
# plt.colorbar()
plt.xlim(-10, 10);
plt.figure(figsize=(7, 7))
ax = plt.axes(projection='3d')
ax.plot_surface(xx, yy, prob,rstride=1, cstride=1,
                cmap='jet', edgecolor='none')
ax.view_init(30, -130)
```

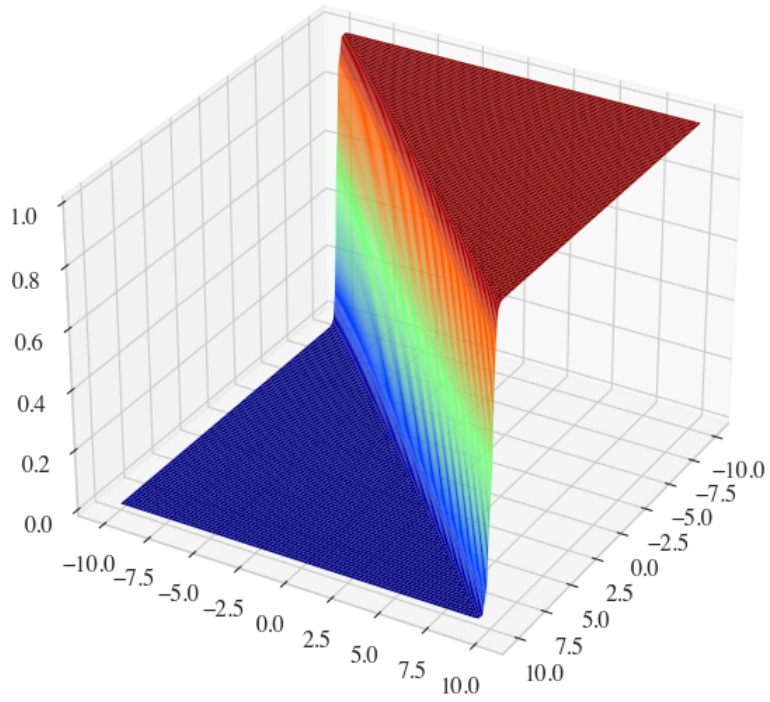




Another example of non-differentiability of `argmax`

In [35]:
```
import numpy as np
from scipy.special import softmax
_softmax = True
temperature = 10
ww = [[1, 2], [-1, 4], [8, 2]]
support = np.linspace(-10, 10, 100)
xx, yy = np.meshgrid(support, support)
W = np.array(ww)
dim = xx.shape
points = np.stack((xx.flatten(), yy.flatten()), axis=1)
prob = np.argmax(points, axis=1) if not _softmax else np.max(np.tile(
    np.arange(0, 2).T, (points.shape[0], 1))*softmax(temperature*points, axis=1), axis=1)
prob = prob.reshape(dim)
plt.figure(figsize=(4, 4))
plt.contourf(xx, yy, prob, cmap='jet', levels=500)
plt.plot(0, 0, 'rx');plt.axis('scaled');plt.xlim(-10, 10);plt.figure(figsize=(7, 7));ax = plt.axes(projection='
ax.plot_surface(xx, yy, prob,rstride=1, cstride=1,
                cmap='jet', edgecolor='none')
ax.view_init(30, 30)
```





SoftMax Regression + Cross Entropy (CE) Loss =

Linear Classifier in Deep Learning =

Multinomial Logistic Regression

SoftMax Regression

SoftMax Regression

We model **conditional probability of $y|x$:**

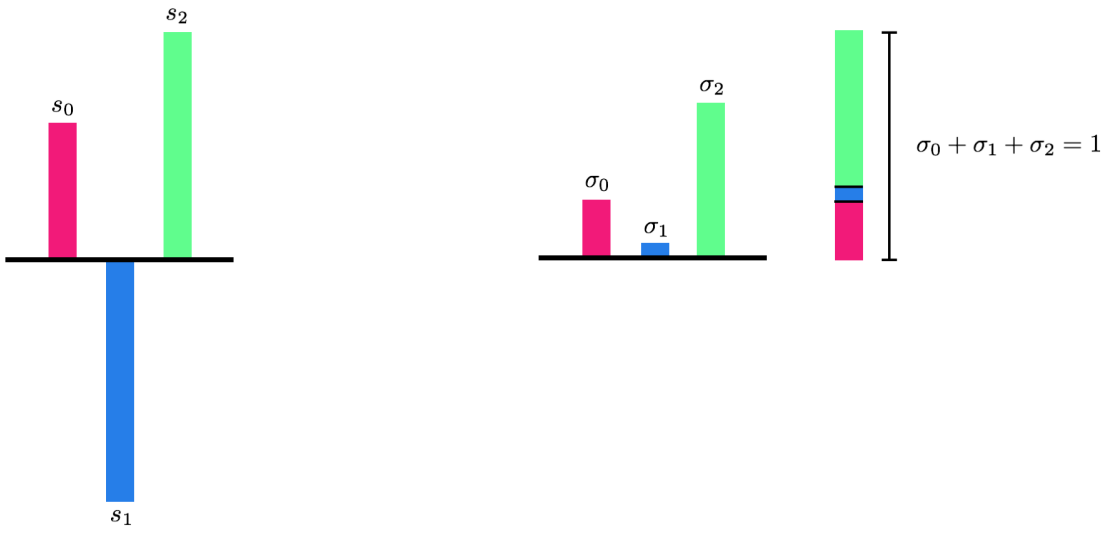
$$\begin{cases} p(y=1\,|\,\mathbf{x};\mathbf{W},\mathbf{b}) = p_1 = \sigma_1(\mathbf{WX}+\mathbf{b}) \\ p(y=2\,|\,\mathbf{x};\mathbf{W},\mathbf{b}) = p_2 = \sigma_2(\mathbf{WX}+\mathbf{b}) \\ \cdots \\ p(y=K\,|\,\mathbf{x};\mathbf{W},\mathbf{b}) = p_K = \sigma_K(\mathbf{WX}+\mathbf{b}) \end{cases}$$

$$f_{\theta}(\mathbf{x}) \doteq \sigma(\mathbf{WX}+\mathbf{b})$$

where:

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{Softmax function}$$

SoftMax Regression



SoftMax Regression, More Compact Form

you can think $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ as **unormalized log-probability** of each class.

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{W}\mathbf{x} + \mathbf{b})}{\sum_{k=1}^{K} \exp(\mathbf{W}_k \mathbf{x} + \mathbf{b}_k)}$$

SoftMax at Work

you can think $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ as **unormalized log-probability** of each class.

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{W}\mathbf{x} + \mathbf{b})}{\sum_{k=1}^{K} \exp(\mathbf{W}_k \mathbf{x} + \mathbf{b}_k)}$$

unormalized log-probability {{np.random.seed(0);z = np.random.randn(10);_=plt.bar(range(z.size),z);}}

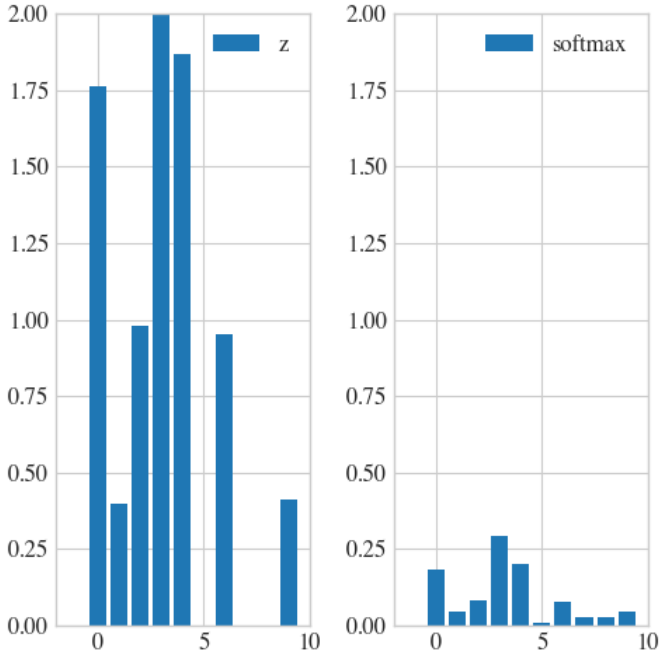
Unormalized Log-Prob z

{{np.random.seed(0);z = np.random.randn(10);_=plt.bar(range(z.size),z);}}

Softmax(z)

{{np.random.seed(0);z = np.random.randn(10);_=plt.bar(range(z.size),softmax(z),color='r');}}

In [36]:
```
np.random.seed(0)
fig, axes = plt.subplots(1, 2, figsize=(5, 5))
z = np.random.randn(10)
axes[0].bar(range(z.size), z)
axes[1].bar(range(z.size), softmax(z))
# axes[0].axis('equal')
# axes[1].axis('equal')
axes[0].set(xlim=(-2,10), ylim=(0, 2))
axes[1].set(xlim=(-2,10), ylim=(0, 2))
axes[0].legend(['z'])
axes[1].legend(['softmax']);
fig.tight_layout()
```



Cross-Entropy in function of p

$$L(\mathrm{D}, W, \mathbf{b}) = \frac{1}{|\mathrm{D}|} \sum_{(\mathbf{x},y)\in \mathrm{D}} \underbrace{-\log\left(\frac{\exp\left(s_y\right)}{\sum_{k=1}^{C}\exp\left(s_k\right)}\right)}_{\text{cross-entropy loss for } (\mathbf{x},y)}$$

{{import numpy as np;import matplotlib.pyplot as plt;p = np.arange(1e-6, 1, 1e-2);y = -np.log(p);plt.figure(figsize=(7,7));plt.plot(p, y);_=plt.legend(['Crossentropy Loss in function of p']);}}

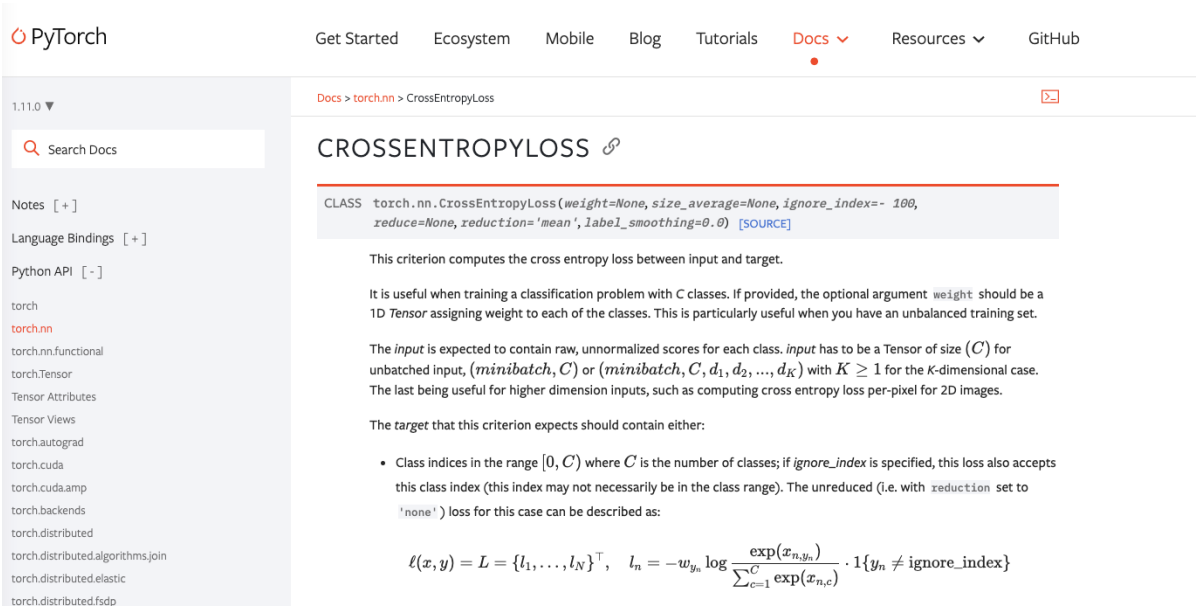
Critical question:

Let's say that you have a model and you sample the weights $\mathbf{W}, \mathbf{b}$ randomly from some distributions so that they **are non zero**. Also assume that the data $\mathbf{x}$ is meaningful (non-zero).

> Can get you `nan` as $+\infty$ in the loss while you **start** training? Or better, which is the value of the loss as soon as you start training?

- If non-zero, in the worst case, your classifier, for a K class problem, it will perform **as bad as a random classifier (the weights are randomly generated)**.
- As such, probably your loss at the beginning of the training will simply be $\approx -\ln(\frac{1}{K})$ so it will **NOT** blow up to `nan` as $+\infty$.
- $-\ln(\frac{1}{10}) \approx 2.3$ when start training.
- In CE we loss **natural log** because softmax exponentiates.

Now we are doing Deep Learning...we just miss the "Deep"



SoftMax with K=2 is Sigmoid?

Yes, almost the same (up to save/wasted space)

$$p(\mathbf{y} \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{W}\mathbf{x} + \mathbf{b})}{\sum_{k=1}^{K} \exp(\mathbf{W}_k \mathbf{x} + \mathbf{b}_k)}$$

Now $K = 2$ and define $\mathbf{z} \doteq \mathbf{W}\mathbf{x} + \mathbf{b}$:

$$p(\mathbf{y} = 1 \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{z}_1)}{\sum_{k=1}^{K} \exp(\mathbf{z}_k)} = \frac{\exp(\mathbf{z}_1)}{\exp(\mathbf{z}_1) + \exp(\mathbf{z}_2)}$$

$$p(\mathbf{y} = 1 \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{z}_1) \cdot}{\sum_{k=1}^{K} \exp(\mathbf{z}_k)} = \frac{\exp(\mathbf{z}_1) \cdot \exp(-\mathbf{z}_1)}{[\exp(\mathbf{z}_1) + \exp(\mathbf{z}_2)] \cdot \exp(-\mathbf{z}_1)} = \frac{1}{1 + \exp(\mathbf{z}_2 - \mathbf{z}_1)}$$

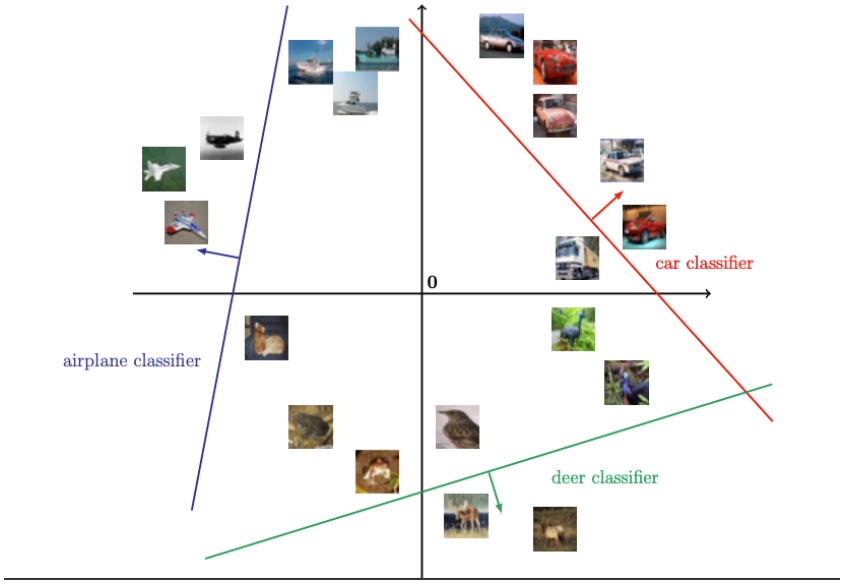
$$p(\mathbf{y} = 1 \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{1}{1 + \exp(-(\mathbf{z}_1 - \mathbf{z}_2))} = \sigma(\mathbf{y}) \text{ where } \mathbf{y} = \mathbf{z}_1 - \mathbf{z}_2$$

Yes it is the same but with sigmoid you spare a vector since $\mathbf{W}$ becomes a single vector $\mathbf{w}$ not a matrix!

We can avoid modeling the other class since $p(\mathbf{y} = 0 \mid \mathbf{x}; \mathbf{W}, \mathbf{b}) = 1 - p(\mathbf{y} = 1 \mid \mathbf{x}; \mathbf{W}, \mathbf{b})$

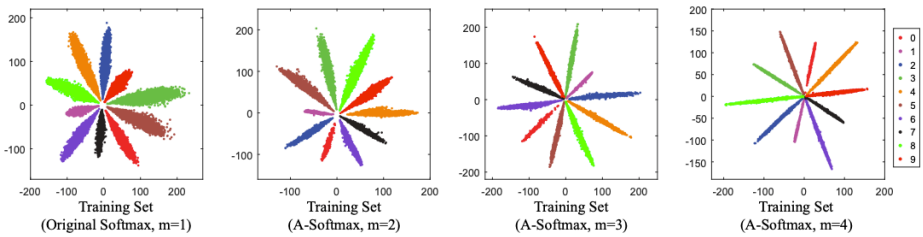
Geometry of the Feature Space of Learned by SoftMax + CE

2D Feature Space Learned with SoftMax + CE



3D Feature Space Learned with SoftMax + CE

From the research side: decreasing intra-class variabiliy is a good idea

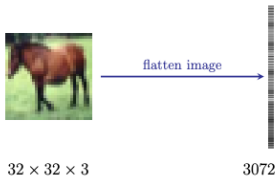


Each $\mathbf{W}_k$ works as a "prototype" of each class

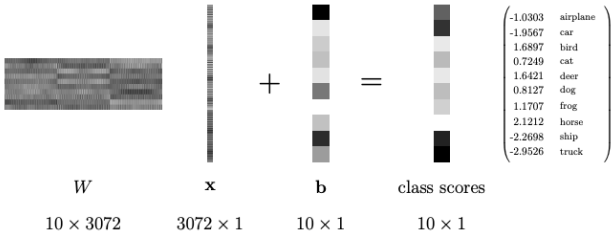
- Learn $\mathbf{W}, \mathbf{b}$ to classify the images in a dataset e.g. CIFAR-10
- Can interpret each row, $\mathbf{W}_k$ of $\mathbf{W}$ as a **template/prototype** for class k
- Below is the visualization of each learnt $\mathbf{W}_k$ for CIFAR-10
- Parallelism between $\boldsymbol{\mu}_k$ learned by K-means and $\mathbf{W}_k$ now but now there is supervision!

Let's see visually the classification

- Have a 2D colour image but can flatten it into a 1D vector $\mathbf{x}$



- Apply classifier: $W\mathbf{x} + \mathbf{b}$ to get a score for each class.



Can you compute the loss value?

input: $\mathbf{x}$ | output | label | loss

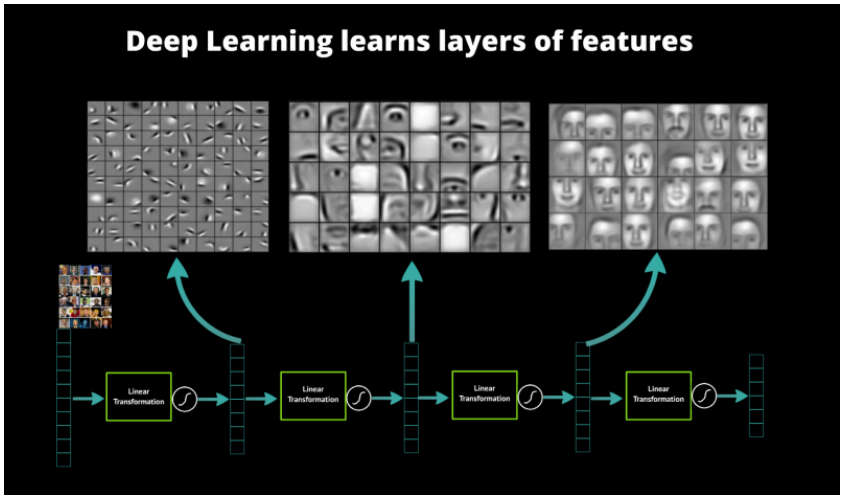
$\mathbf{s} = W\mathbf{x} + \mathbf{b}$ | $y = 8$ | $l = -\log\left(\frac{\exp(s_y)}{\sum \exp(s_k)}\right)$

	Scores	exp(Scores)	Normalized scores
airplane	-0.3166	0.7354	0.0571
car	-0.6609	0.5328	0.0414
bird	0.7058	2.0203	0.1568
cat	0.8538	2.3583	0.1830
deer	0.6525	1.9303	0.1498
dog	0.1874	1.2080	0.0938
frog	0.6072	1.8319	0.1422
horse	0.5134	1.7141	0.1330
ship	-1.3490	0.2585	0.0201
truck	-1.2225	0.2945	0.0229
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$\exp(\mathbf{s})$	$\frac{\exp(\mathbf{s})}{\sum_k \exp(s_k)}$

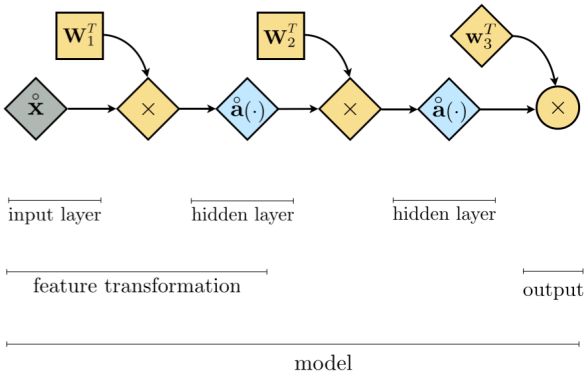
Loss for $\mathbf{x}$: 2.0171

Damn, until now is all linear. So now the "Deep"!

- Damn, until now is all linear.
- Our **beloved SoftMax+CE linear layer** is there **in the end (classifier).**



Neural Net as a Computational Graph that implements Backprop



In [ ]: