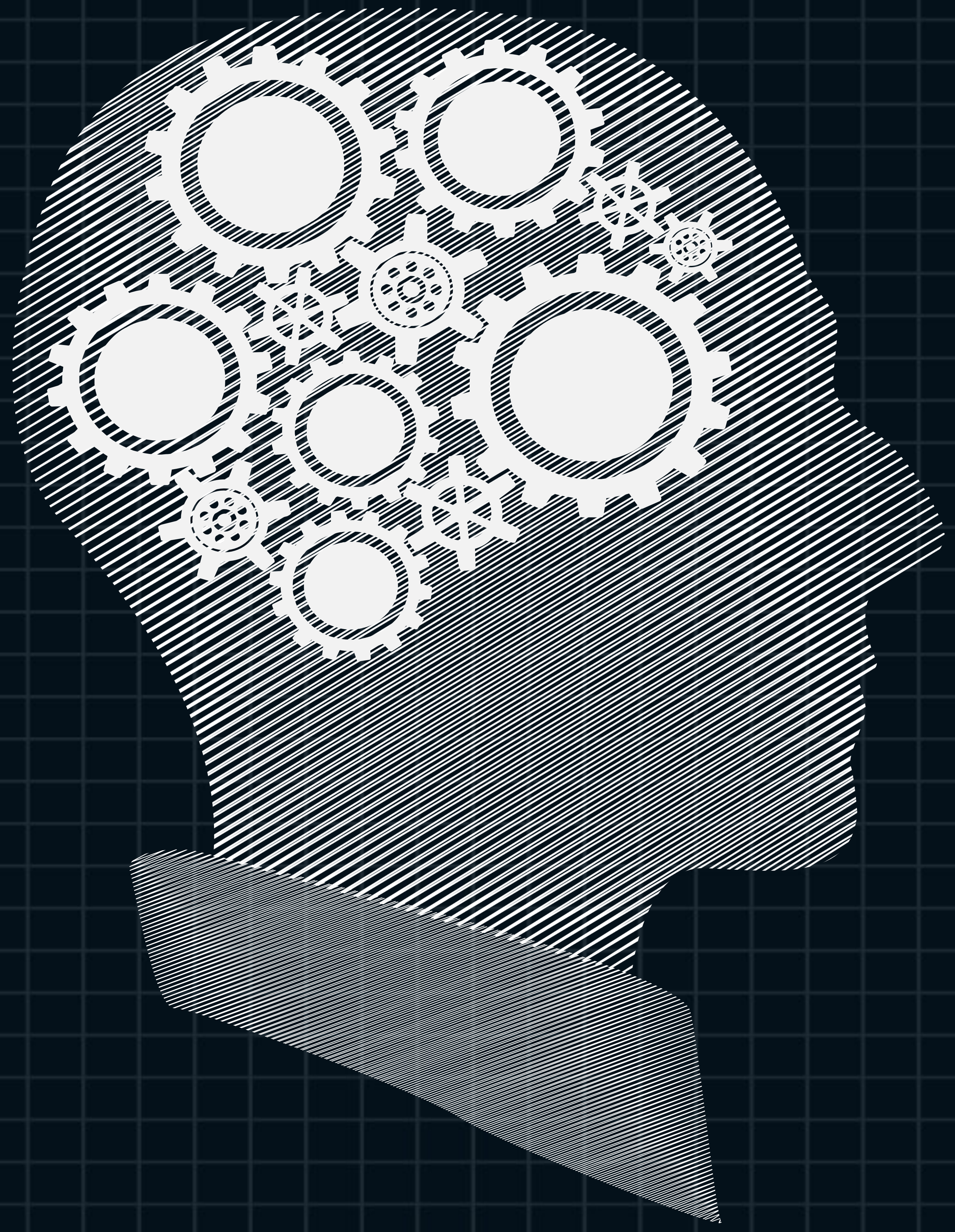
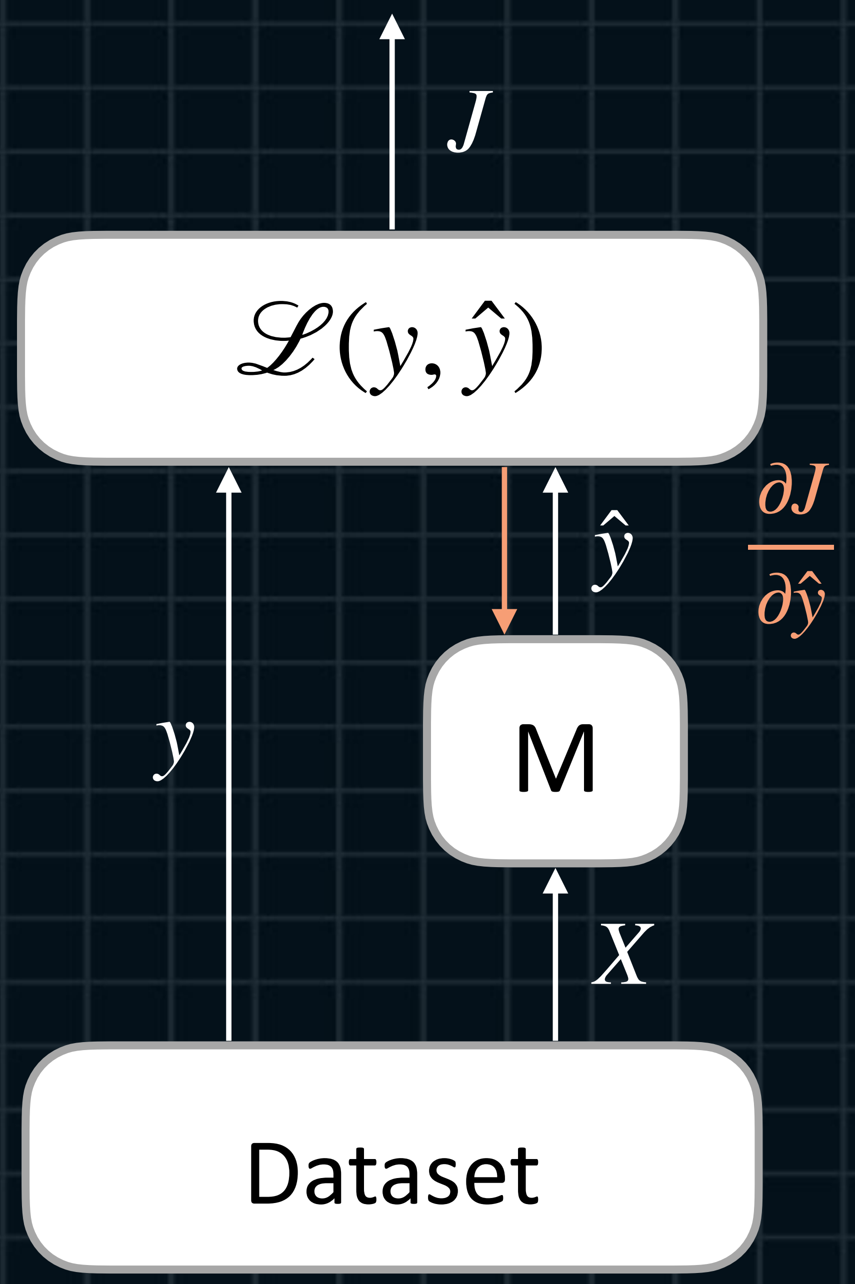


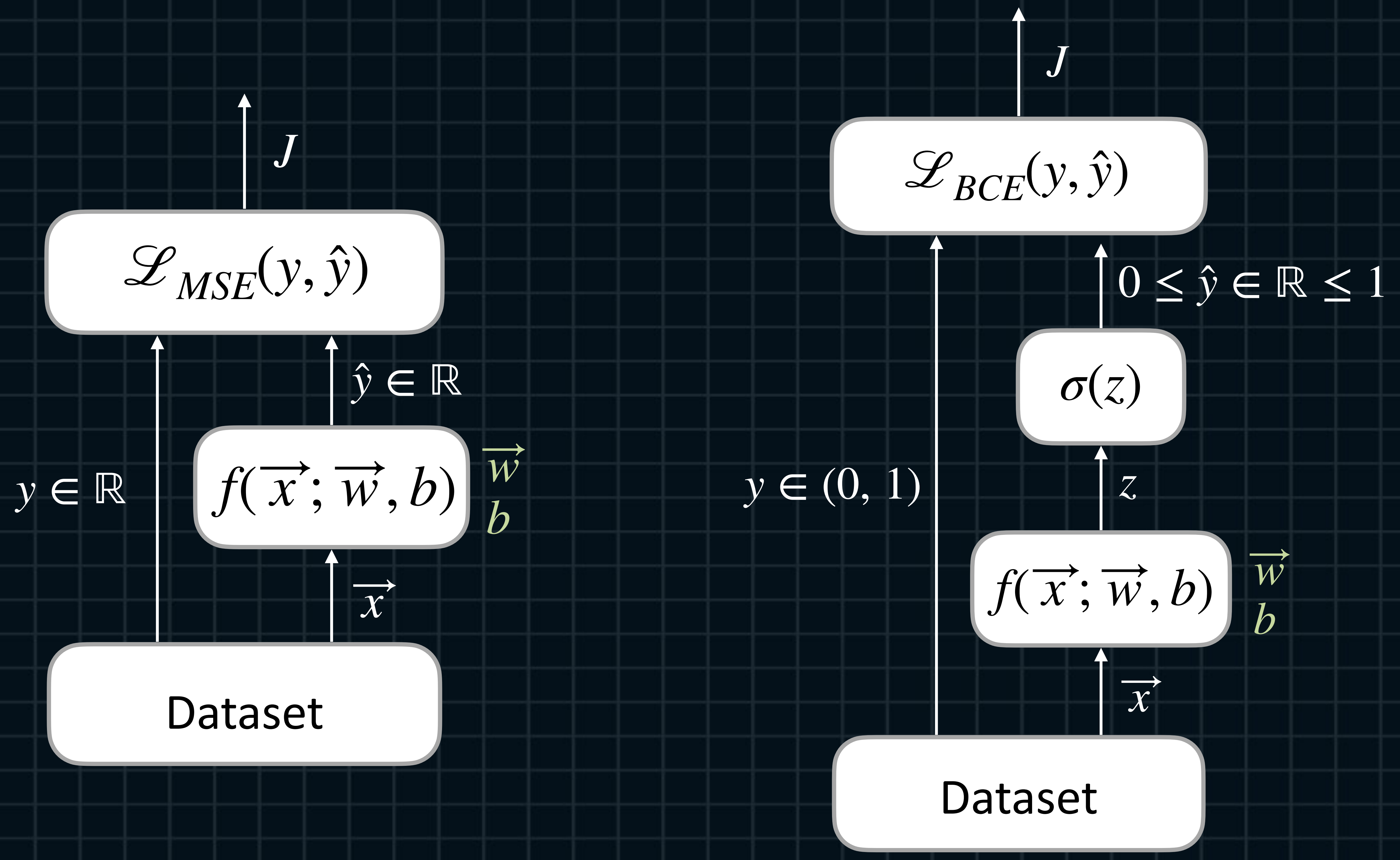
Backpropagation and Jacobian Matrices

Lecture.4
Linear/Logistic Regression(1)

Linear/Logistic Regression



Linear/Logistic Regression



$$\vec{w} := \vec{w} - \alpha \left(\frac{\partial J}{\partial \vec{w}} \right)^T, \quad b := b - \alpha \frac{\partial J}{\partial b}$$

Forward Propagation and Partial Derivatives

$$J$$

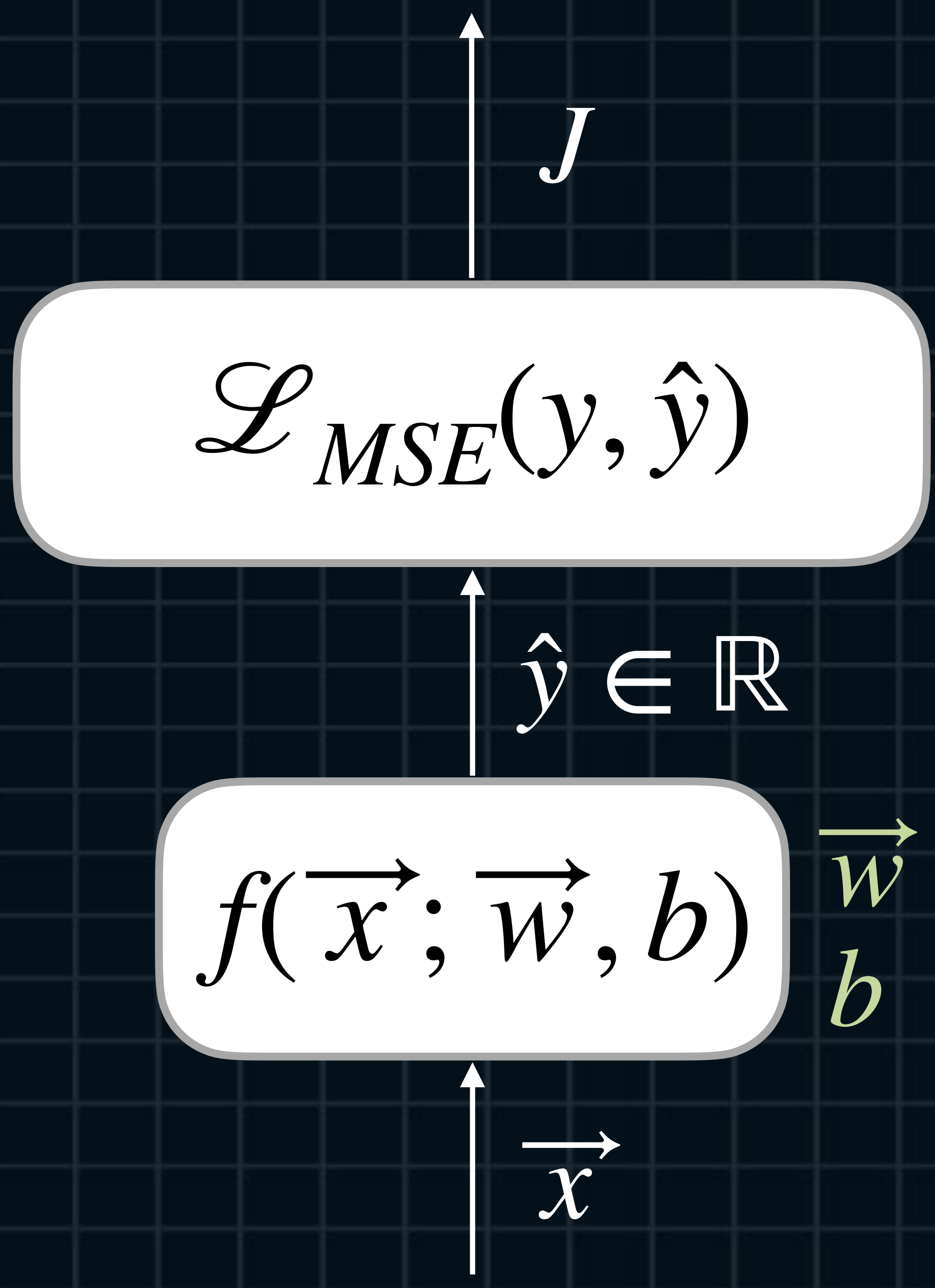
$$J = (y - \hat{y})^2 \qquad \boxed{\mathcal{L}_{MSE}(y, \hat{y})} \qquad \frac{\partial J}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\hat{y} \in \mathbb{R}$$

$$\hat{y} = \overrightarrow{x}^T \cdot \overrightarrow{w} + b \qquad \boxed{f(\overrightarrow{x}; \overrightarrow{w}, b)} \quad \begin{matrix} \overrightarrow{w} \\ b \end{matrix} \qquad \frac{\partial \hat{y}}{\partial \overrightarrow{w}} = (x_1 \quad x_2 \quad \dots \quad x_n) = \overrightarrow{x}^T, \quad \frac{\partial \hat{y}}{\partial b} = 1$$

$$\overrightarrow{x}$$

Backpropagation



$$\frac{\partial J}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial \vec{w}} = (x_1 \quad x_2 \quad \cdots \quad x_n) = \vec{x}^T, \quad \frac{\partial \hat{y}}{\partial b} = 1$$

$$\frac{\partial J}{\partial \vec{w}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{w}} = -2(y - \hat{y}) \cdot \vec{x}^T$$

$$\vec{w} := \vec{w} - \alpha \left(\frac{\partial J}{\partial \vec{w}} \right)^T$$

$$\vec{w} := \vec{w} + 2\alpha(y - \hat{y}) \cdot \vec{x}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} = -2(y - \hat{y})$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

$$b := b + 2\alpha(y - \hat{y})$$

Lecture.4 Linear/Logistic Regression(1) - Implementations

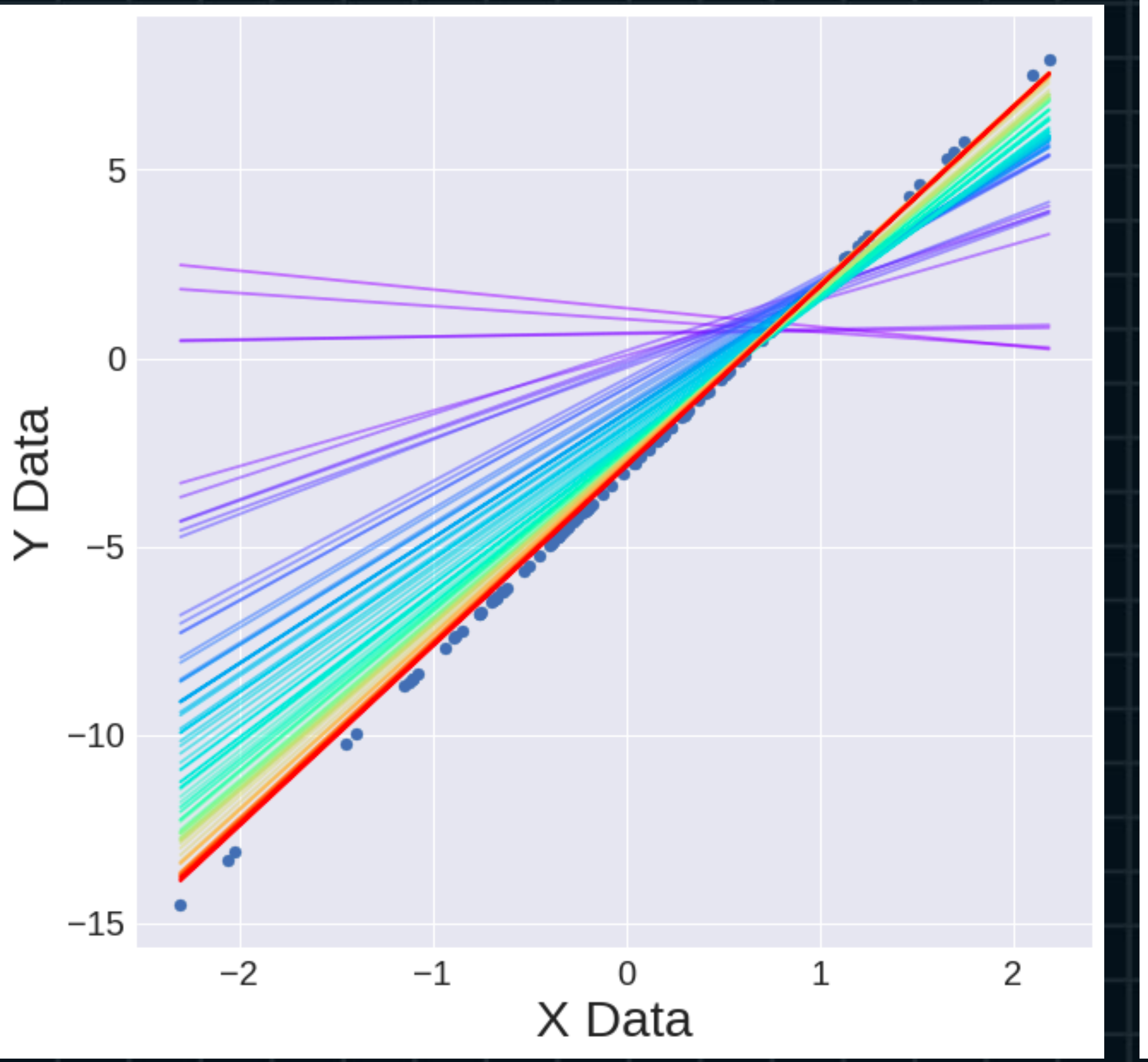
Linear Regression(1 Feature)

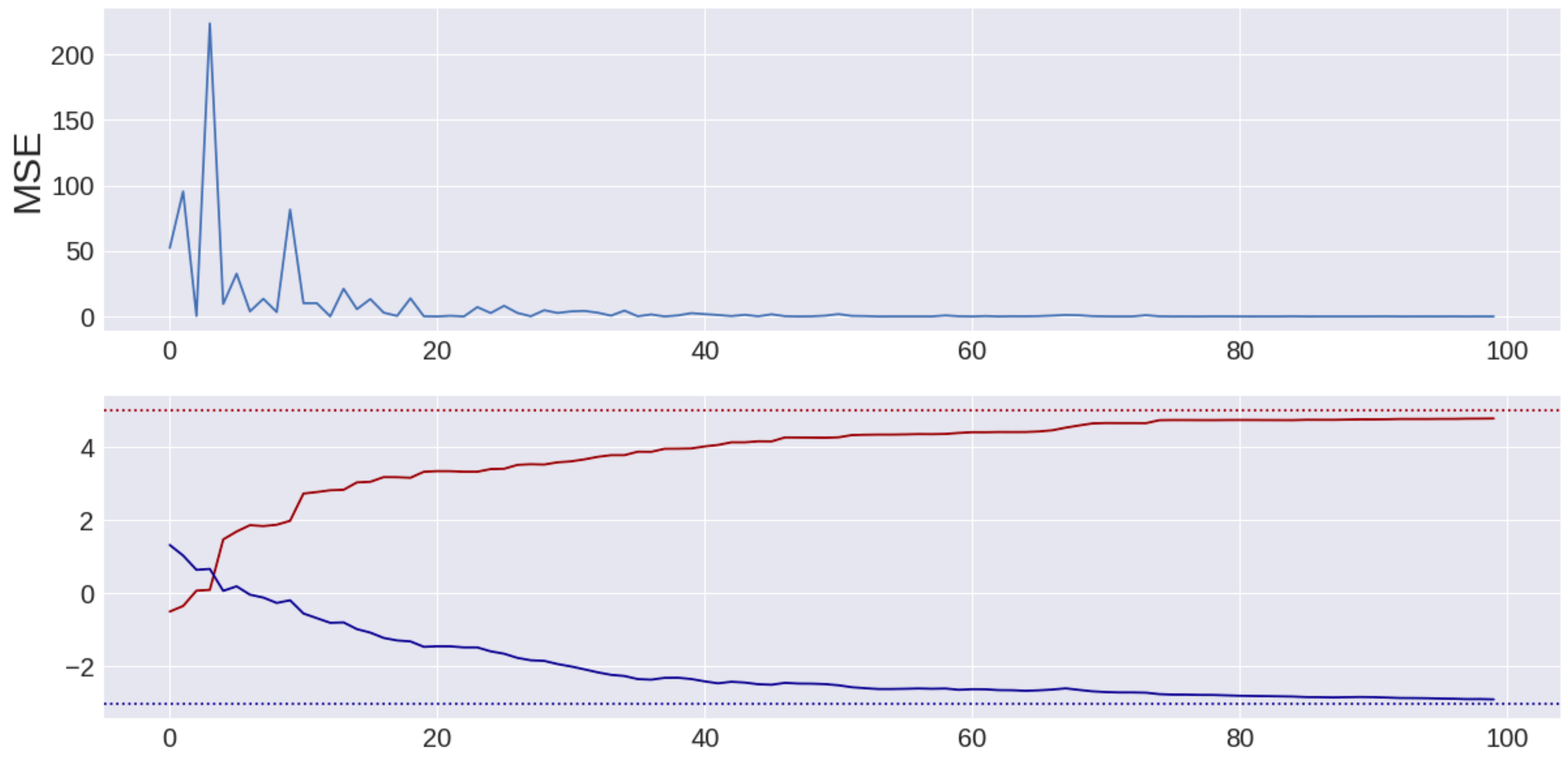
```python
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)
plt.style.use('seaborn')

# set params
n_data = 100
lr = 0.01
t_w, t_b = 5, -3
w, b = np.random.uniform(-3, 3, 2) # initial weight, bias

# generate dataset
x_data = np.random.randn(n_data, )
y_data = t_w*x_data + t_b
# y_data = t_w*x_data + t_b + np.random.randn(n_data, )

# visualize dataset
cmap = plt.get_cmap('rainbow', lut=n_data)

fig, ax = plt.subplots(figsize=(10, 10))
ax.scatter(x_data, y_data)

ax.set_xlabel('X Data', fontsize=30)
ax.set_ylabel('Y Data', fontsize=30)
ax.tick_params(labelsize=20)

# set x range for visualization of model
x_range = np.array([x_data.min(), x_data.max()])
```

```python
# train model and visualize updated model
J_track = list()
w_track, b_track = list(), list()
for data_idx, (x, y) in enumerate(zip(x_data, y_data)):
  w_track.append(w)
  b_track.append(b)

  # visualize updated model
  y_range = w*x_range + b
  ax.plot(x_range, y_range, color=cmap(data_idx), alpha=0.5)

  # loss calculation
  pred = x*w + b
  J = (y - pred)**2
  J_track.append(J)

  # jacobians
  dJ_dpred = -2*(y - pred)
  dpred_dw = x
  dpred_db = 1

  # backpropagation
  dJ_dw = dJ_dpred * dpred_dw
  dJ_db = dJ_dpred * dpred_db

  w = w - 2*lr*dJ_dw
  b = b - 2*lr*dJ_db
```

Linear Regression(1 Feature)

```
# visualize loss
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
axes[0].plot(J_track)
axes[0].set_ylabel('MSE', fontsize=30)
axes[0].tick_params(labelsize=20)

axes[1].axhline(y=t_w, color='darkred', linestyle=':')
axes[1].plot(w_track, color='darkred')
axes[1].axhline(y=t_b, color='darkblue', linestyle=':')
axes[1].plot(b_track, color='darkblue')
axes[1].tick_params(labelsize=20)
```





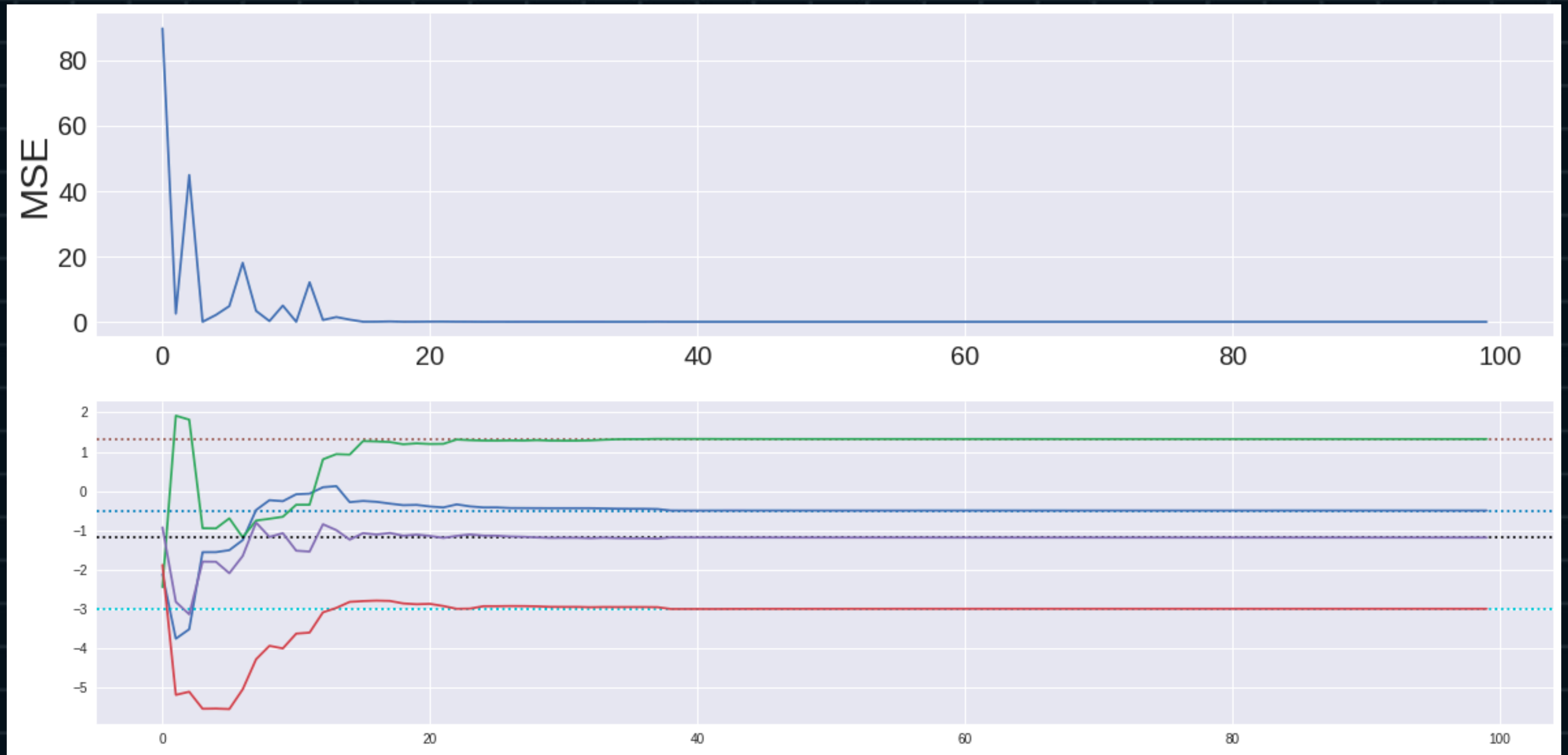
Linear Regression(n Features)

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
np.random.seed(1)
plt.style.use('seaborn')

# set params
n_data, n_feature = 100, 3
lr = 0.1
t_W = np.random.uniform(-3, 3, (n_feature, 1))
t_b = np.random.uniform(-3, 3, (1, ))

W = np.random.uniform(-3, 3, (n_feature, 1))
b = np.random.uniform(-3, 3, (1, ))


# generate dataset
x_data = np.random.randn(n_data, n_feature)
y_data = x_data @ t_W + t_b
```

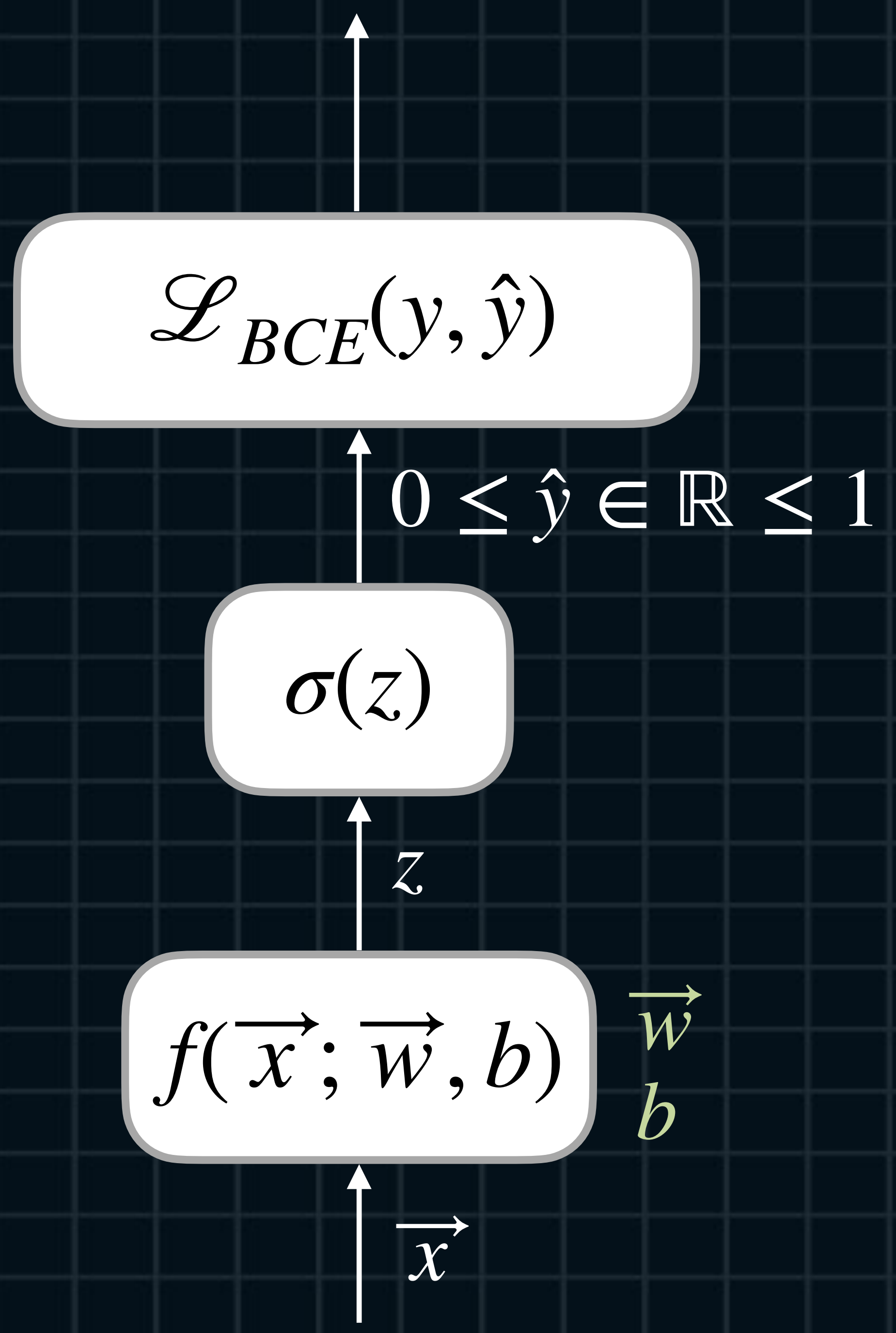
```python
J_track = list()
W_track, b_track = list(), list()
for data_idx, (X, y) in enumerate(zip(x_data, y_data)):
  W_track.append(W)
  b_track.append(b)

  # forward propagation
  pred = X @ W + b
  J = (y - pred)**2
  J_track.append(J)

  # jacobians
  dJ_dpred = -2*(y - pred)
  dpred_dW = X.reshape(1, -1)
  dpred_b = 1

  # backpropagation
  dJ_dW = dJ_dpred * dpred_dW
  dJ_db = dJ_dpred * dpred_db

  # paramter update
  # print(W.shape, dJ_dW.shape)
  # print(b.shape, dJ_db.shape)
  W = W - lr*dJ_dW.T
  b = b - lr*dJ_db

W_track = np.hstack(W_track)
b_track = np.concatenate(b_track)
```

Linear Regression(n Features)

```
# visualize loss
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
axes[0].plot(J_track)
axes[0].set_ylabel('MSE', fontsize=30)
axes[0].tick_params(labelsize=20)

cmap = cm.get_cmap('tab10', lut=n_feature)
for w_idx, (t_w, w) in enumerate(zip(t_W, W_track)):
  axes[1].axhline(y=t_w, color=cmap(w_idx), linestyle=':')
  axes[1].plot(w)
axes[1].axhline(y=t_b, color='black', linestyle=':')
axes[1].plot(b_track)
axes[0].tick_params(labelsize=20)
```



Logistic Regression Model



Forward Propagation and Partial Derivatives

$$J = -\left[y log(\hat{y}) + (1-y) log(1-\hat{y})\right] \qquad \mathscr{L}_{BCE}(y, \hat{y}) \qquad \frac{\partial J}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$0 \le \hat{y} \in \mathbb{R} \le 1$$

$$\hat{y} = \sigma(z) \qquad \sigma(z) \qquad \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

$$z$$

$$z = \vec{x}^T \cdot \vec{w} + b \qquad f(\vec{x}; \vec{w}, b) \qquad \vec{w} \quad b \qquad \frac{\partial z}{\partial \vec{w}} = (x_1 \quad x_2 \quad \dots \quad x_n) = \vec{x}^T, \quad \frac{\partial z}{\partial b} = 1$$

$$\vec{x}$$

- Logistic Regression

Backpropagation

$$\mathcal{L}_{BCE}(y, \hat{y}) \qquad \frac{\partial J}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$0 \leq \hat{y} \in \mathbb{R} \leq 1$$

$$\sigma(z) \qquad \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

$$z$$

$$f(\overrightarrow{x}; \overrightarrow{w}, b) \qquad \begin{matrix} \overrightarrow{w} \\ b \end{matrix} \qquad \frac{\partial z}{\partial \overrightarrow{w}} = (x_1 \quad x_2 \quad \cdots \quad x_n) = \overrightarrow{x}^T, \quad \frac{\partial z}{\partial b} = 1$$

$$\overrightarrow{x}$$

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y})$$

$$= \hat{y} - y = -(y - \hat{y}) \qquad cf)\ \frac{\partial J}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\frac{\partial J}{\partial \overrightarrow{w}} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial \overrightarrow{w}} = -(y - \hat{y}) \cdot \overrightarrow{x}^T$$

$$\overrightarrow{w} := \overrightarrow{w} - \alpha\left(\frac{\partial J}{\partial \overrightarrow{w}}\right)^T$$

$$\overrightarrow{w} := \overrightarrow{w} + \alpha(y - \hat{y}) \cdot \overrightarrow{x}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial b} = -(y - \hat{y})$$

$$b := b - \alpha\frac{\partial J}{\partial b}$$

$$b := b + \alpha(y - \hat{y})$$

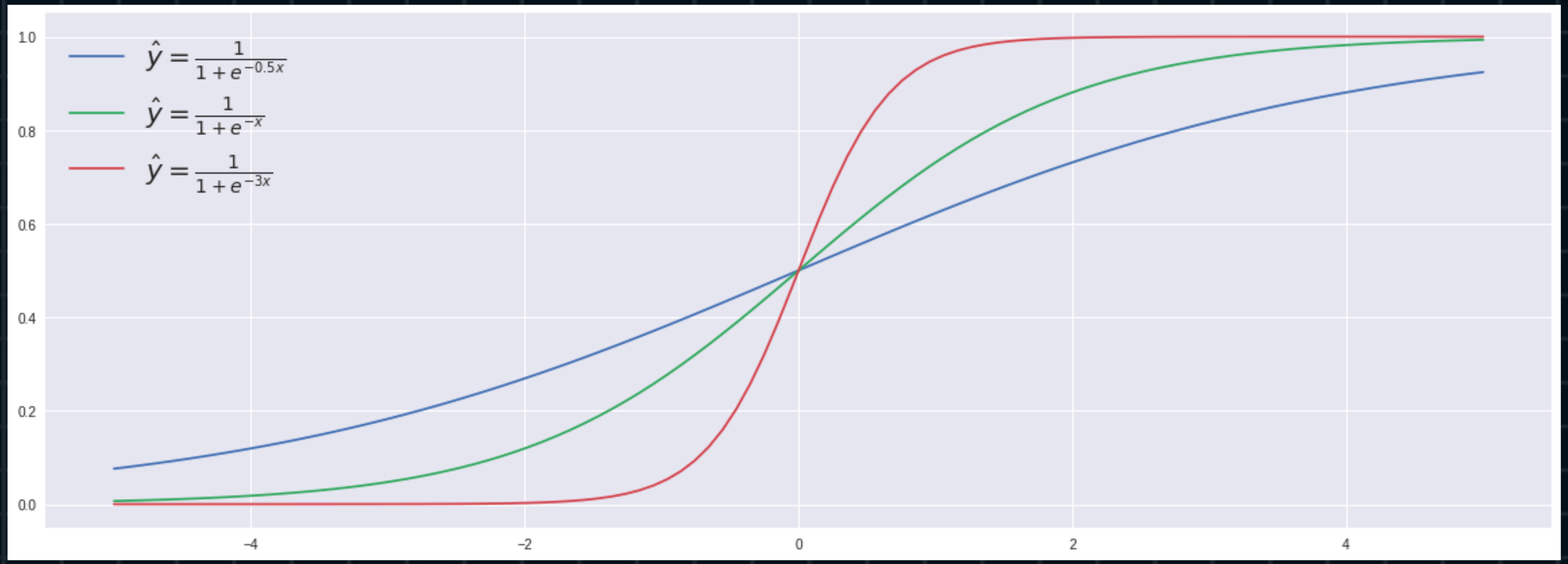
Lecture.4 Linear/Logistic Regression(1) - Logistic Regression

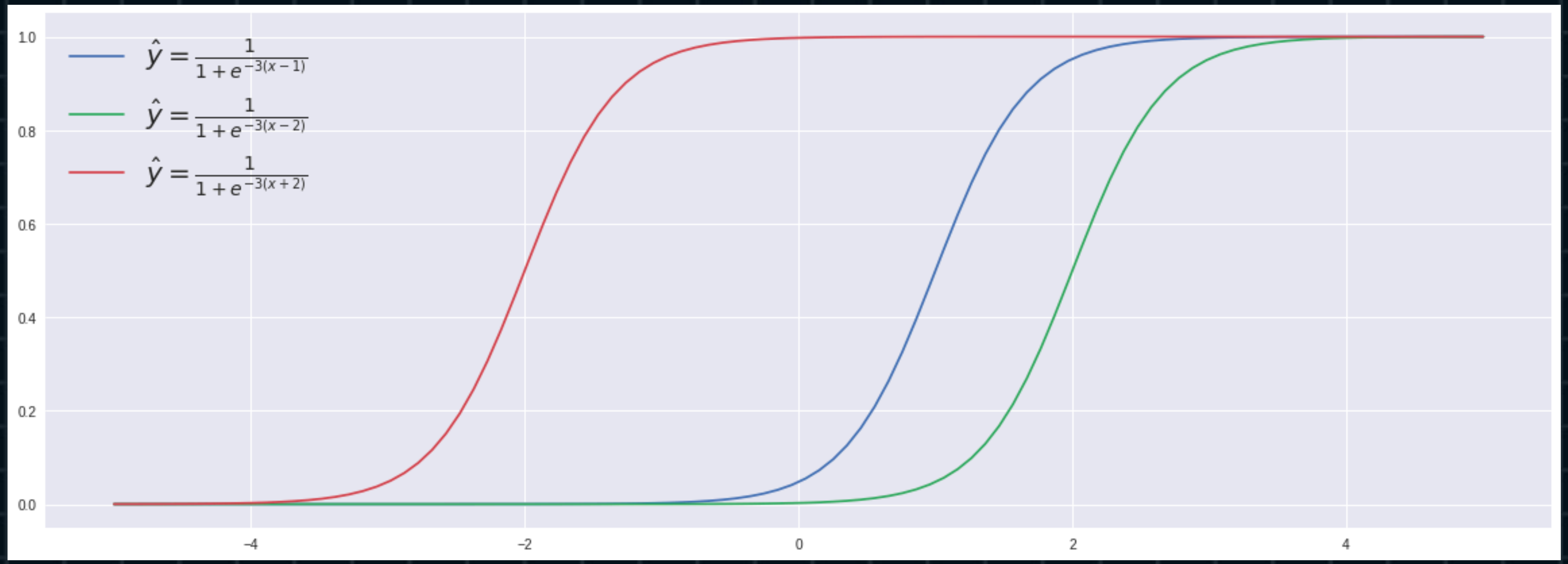
Sigmoid's Params

$$\hat{y} = \sigma(z) = \sigma(xw + b)$$
$$= \frac{1}{1 + e^{-(xw+b)}}$$
$$\hat{y} = \frac{1}{1 + e^{-w(x-(-\frac{b}{w}))}}$$

$$\hat{y} = \frac{1}{1 + e^{-wx}}$$

$$\hat{y} = \frac{1}{1 + e^{-w(x-(-\frac{b}{w}))}}$$

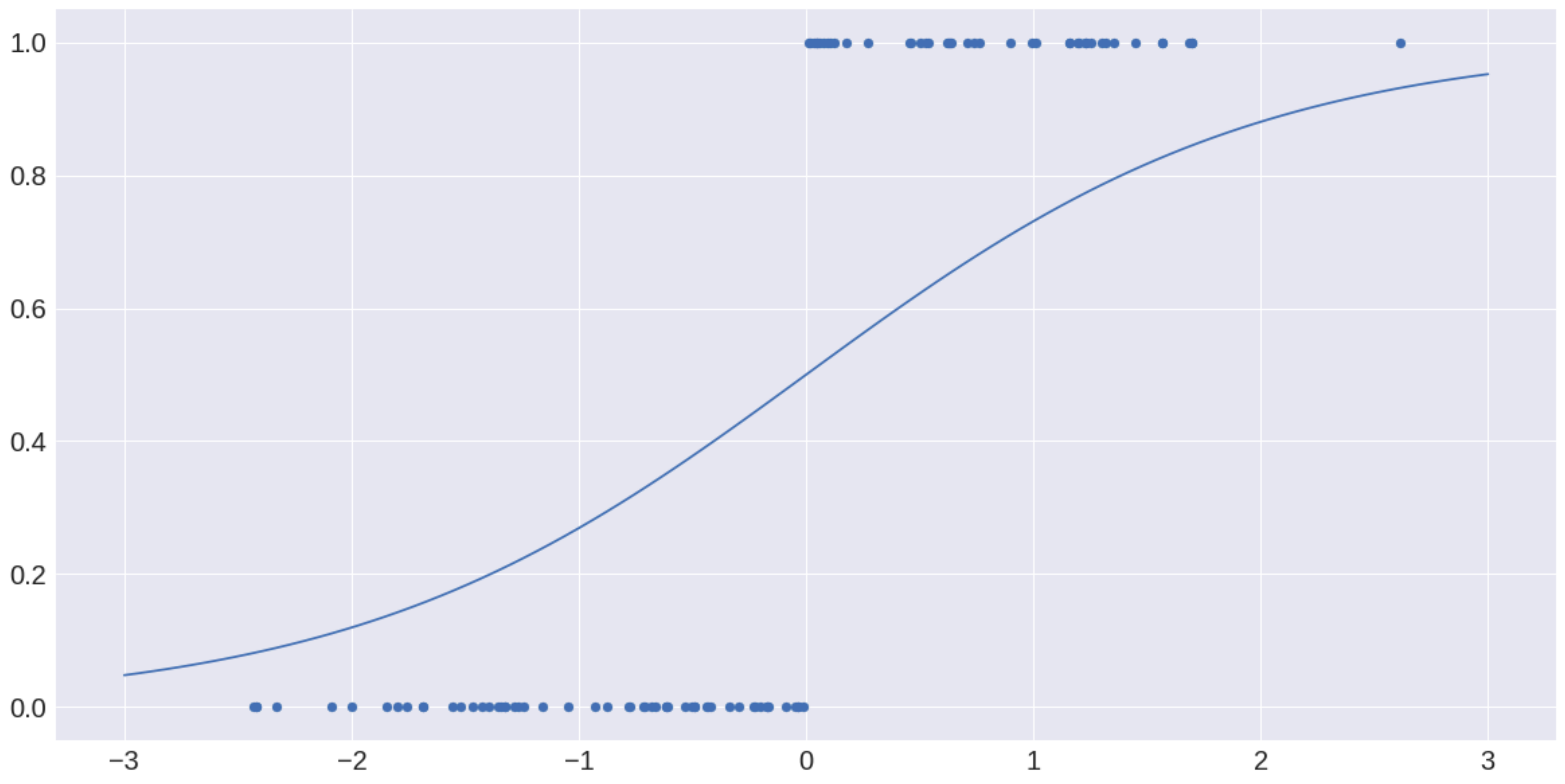




- Logistic Regression

Decision Boundary and Parameter Update

$$J = -\left[ylog(\hat{y}) + (1-y)log(1-\hat{y})\right]$$



Lecture.4
Linear/Logistic Regression(1) - Implementations

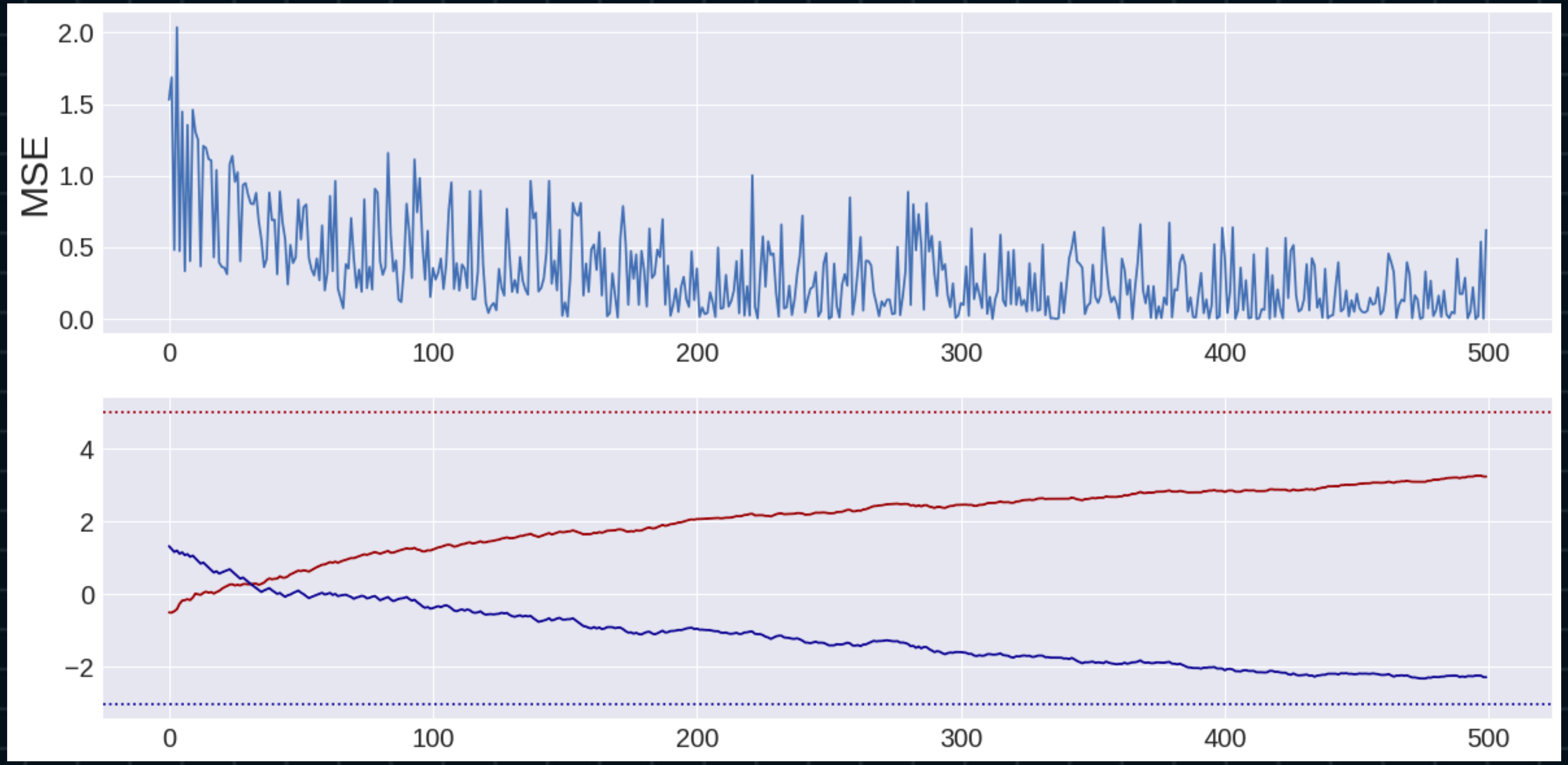
Logistic Regression(1 Feature)

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)
plt.style.use('seaborn')

# set params
n_data = 500
lr = 0.1
t_w, t_b = 5, -3
w, b = np.random.uniform(-3, 3, 2) # initial weight, bias

# generate dataset
t_decision_boundary = -t_b/t_w
x_data = np.random.normal(t_decision_boundary, 1,
                          (n_data, ))
y_data = x_data * t_w + t_b
y_data = (y_data > t_decision_boundary).astype(np.float32)

# visualize dataset
cmap = plt.get_cmap('rainbow', lut=n_data)

fig, ax = plt.subplots(figsize=(20, 7))
ax.scatter(x_data, y_data)
ax.set_xlabel('X Data', fontsize=30)
ax.set_ylabel('Y Data', fontsize=30)
ax.tick_params(labelsize=20)
ax.set_ylim([-0.2, 1.2])

# set x range for visualization of model
x_range = np.linspace(-2, 4, 100)
```

```
# train model and visualize updated model
J_track = list()
w_track, b_track = list(), list()
for data_idx, (x, y) in enumerate(zip(x_data, y_data)):
  w_track.append(w)
  b_track.append(b)

  # visualize updated model
  y_range = w*x_range + b
  y_range = 1/(1 + np.exp(-y_range))
  ax.plot(x_range, y_range, color=cmap(data_idx), alpha=0.3)

  # loss calculation
  pred = x*w + b
  pred = 1/(1 + np.exp(-pred))
  J = -(y*np.log(pred) + (1-y)*np.log(1-pred))
  J_track.append(J)

  # jacobians
  dJ_dpred = (pred - y)/(pred*(1-pred))
  dpred_dz = pred*(1-pred)
  dz_dw = x
  dz_db = 1

  # backpropagation
  dJ_dz = dJ_dpred * dpred_dz
  dJ_dw = dJ_dz * dz_dw
  dJ_db = dJ_dz * dz_db

  # train model
  w = w - lr*dJ_dw
  b = b - lr*dJ_db
```

Logistic Regression(1 Feature)

```
# visualize loss
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
axes[0].plot(J_track)
axes[0].set_ylabel('MSE', fontsize=30)
axes[0].tick_params(labelsize=20)

axes[1].axhline(y=t_w, color='darkred', linestyle=':')
axes[1].plot(w_track, color='darkred')
axes[1].axhline(y=t_b, color='darkblue', linestyle=':')
axes[1].plot(b_track, color='darkblue')
axes[1].tick_params(labelsize=20)
```



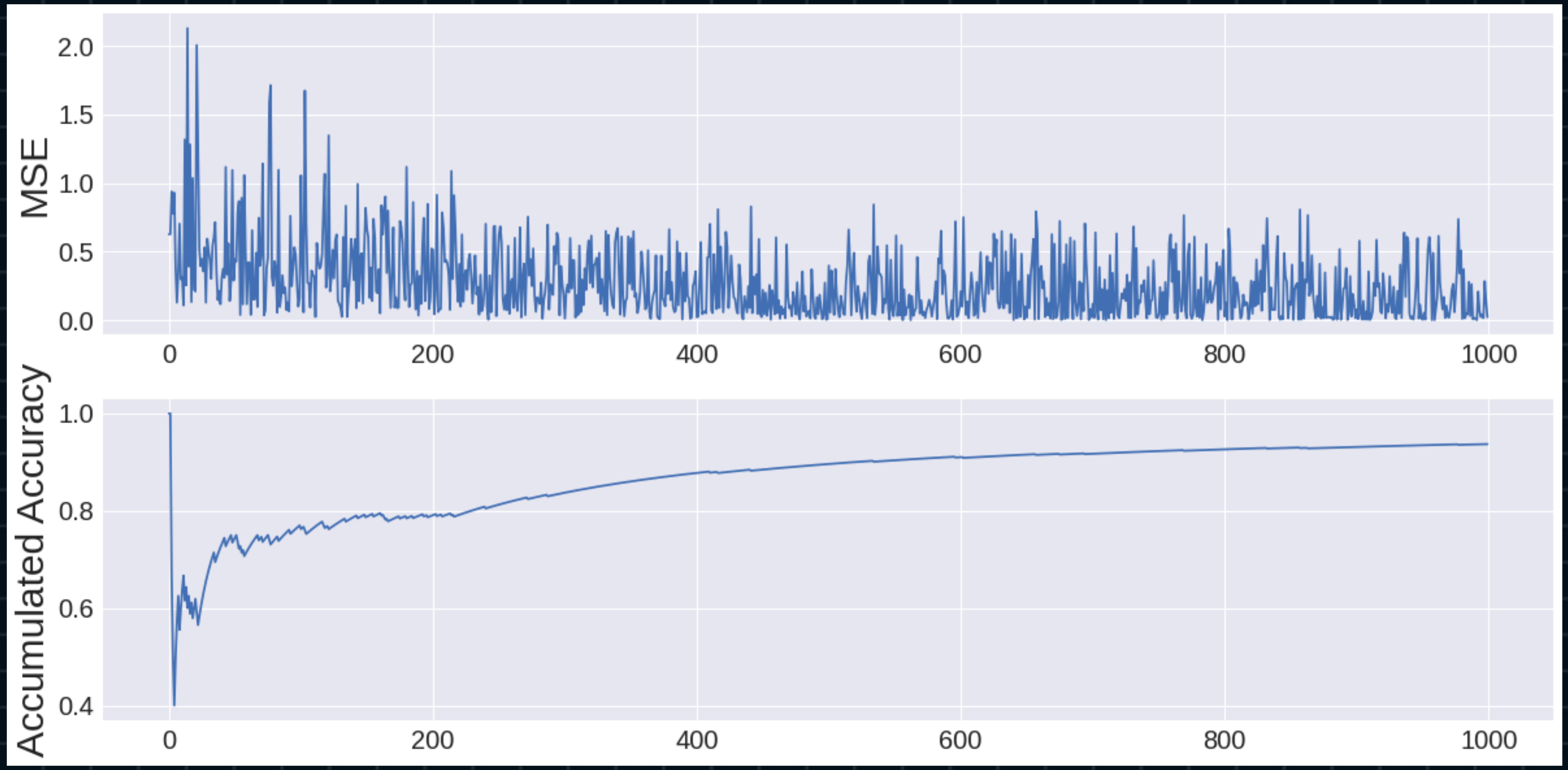
Logistic Regression(n Features)

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)
plt.style.use('seaborn')

# set params
n_data, n_feature = 1000, 3
lr = 0.03
t_W = np.random.uniform(-1, 1, (n_feature, 1)) # target weights
t_b = np.random.uniform(-1, 1, (1, )) # target bias
W = np.random.uniform(-1, 1, (n_feature, 1)) # initial weights
b = np.random.uniform(-1, 1, (1, )) # initial biass

# generate dataset
x_data = np.random.randn(n_data, n_feature)
y_data = x_data @ t_W + t_b
y_data = 1/(1 + np.exp(-y_data))
y_data = (y_data > 0.5).astype(np.int)

J_track, acc_track = list(), list()
n_correct = 0
```

```
for data_idx, (X, y) in enumerate(zip(x_data, y_data)):
  # train model
  pred = X @ W + b
  pred = 1/(1 + np.exp(-pred))

  J = -(y*np.log(pred) + (1-y)*np.log(1-pred))
  J_track.append(J)

  pred_ = (pred > 0.5).astype(np.int)
  if pred_ == y:
    n_correct += 1
  acc_track.append(n_correct/(data_idx + 1))

  # jacobians
  dJ_dpred = (pred - y)/(pred*(1-pred))
  dpred_dz = pred*(1-pred)
  dz_dW = X.reshape(1, -1)
  dz_db = 1

  # backpropagation
  dJ_dz = dJ_dpred * dpred_dz
  dJ_dW = dJ_dz * dz_dW
  dJ_db = dJ_dz * dz_db

  # train model
  W = W - lr*dJ_dW.T
  b = b - lr*dJ_db
```

Logistic Regression(n Features)

```python
# visualize weight/bias
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
axes[0].plot(J_track)
axes[1].plot(acc_track)

axes[0].set_ylabel('MSE', fontsize=30)
axes[0].tick_params(labelsize=20)
axes[1].set_ylabel('Accumulated Accuracy', fontsize=30)
axes[1].tick_params(labelsize=20)
```



Backpropagation and Jacobian Matrices

Lecture.4
Linear/Logistic Regression(1)