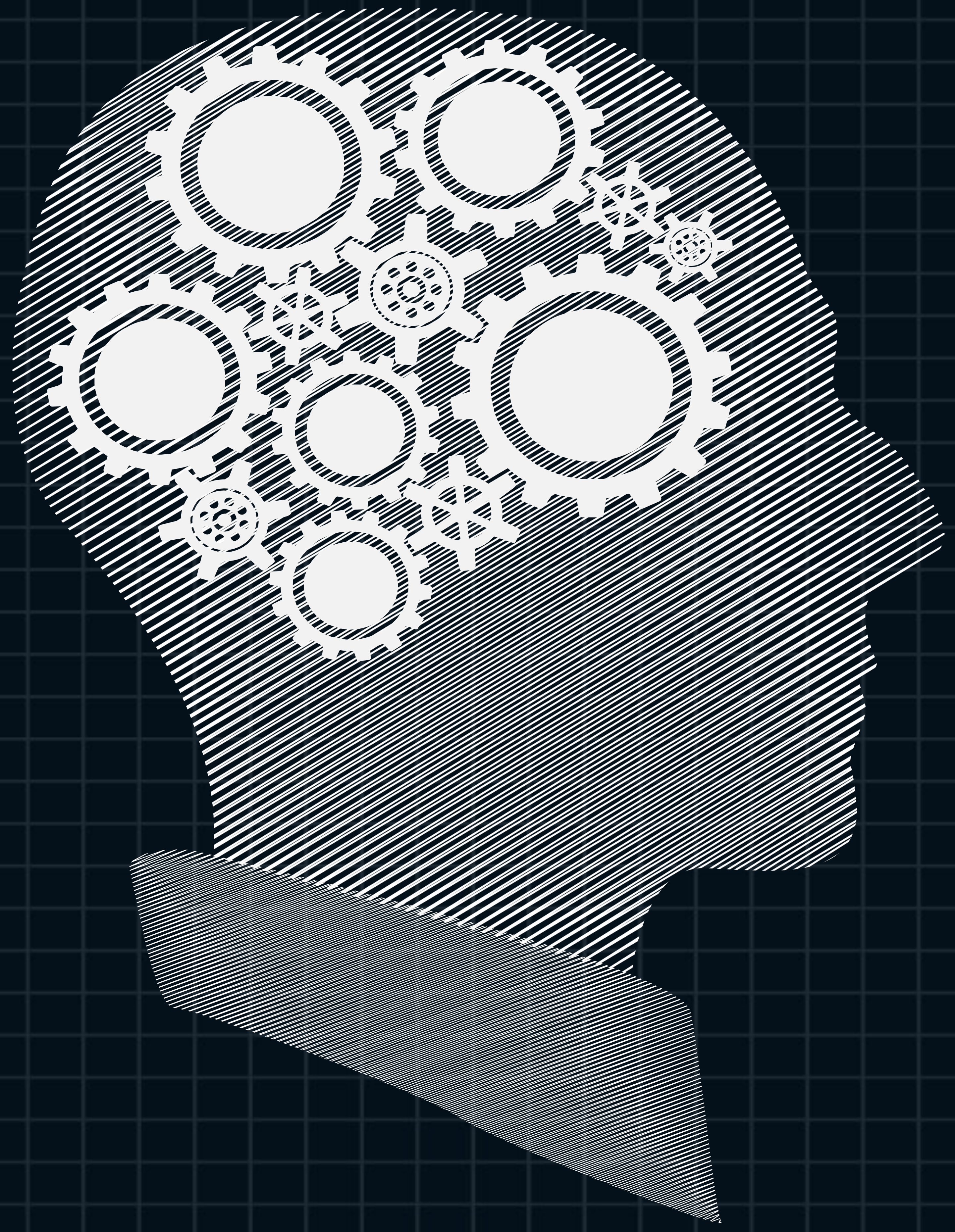


Backpropagation and Jacobian Matrices

Lecture.11 Applications of Expanded Jacobians

Lecture.11 Applications of Expanded Jacobians - MNIST Classifier Model

$J \in \mathbb{R}$

L_{CE} $\quad J = \mathscr{L}_{CCEE}(Y, \hat{Y})$ **Loss Calculator**

$\hat{Y} \in \mathbb{R}^{N \times K}$

$L_{Softmax}$ $\quad \hat{Y} = Softmax(L)$

$L \in \mathbb{R}^{N \times l_3}$

$W^{[3]} \in \mathbb{R}^{l_2 \times l_3}$, $\vec{b}^{[3]} \in \mathbb{R}^{l_3}$ $\quad L^{[3]}_{Dense}$ $\quad L = A^{[2]} \cdot W^{[3]} + \left(\vec{b}^{[3]}\right)^T$

$A^{[2]} \in \mathbb{R}^{N \times l_2}$

Classifier

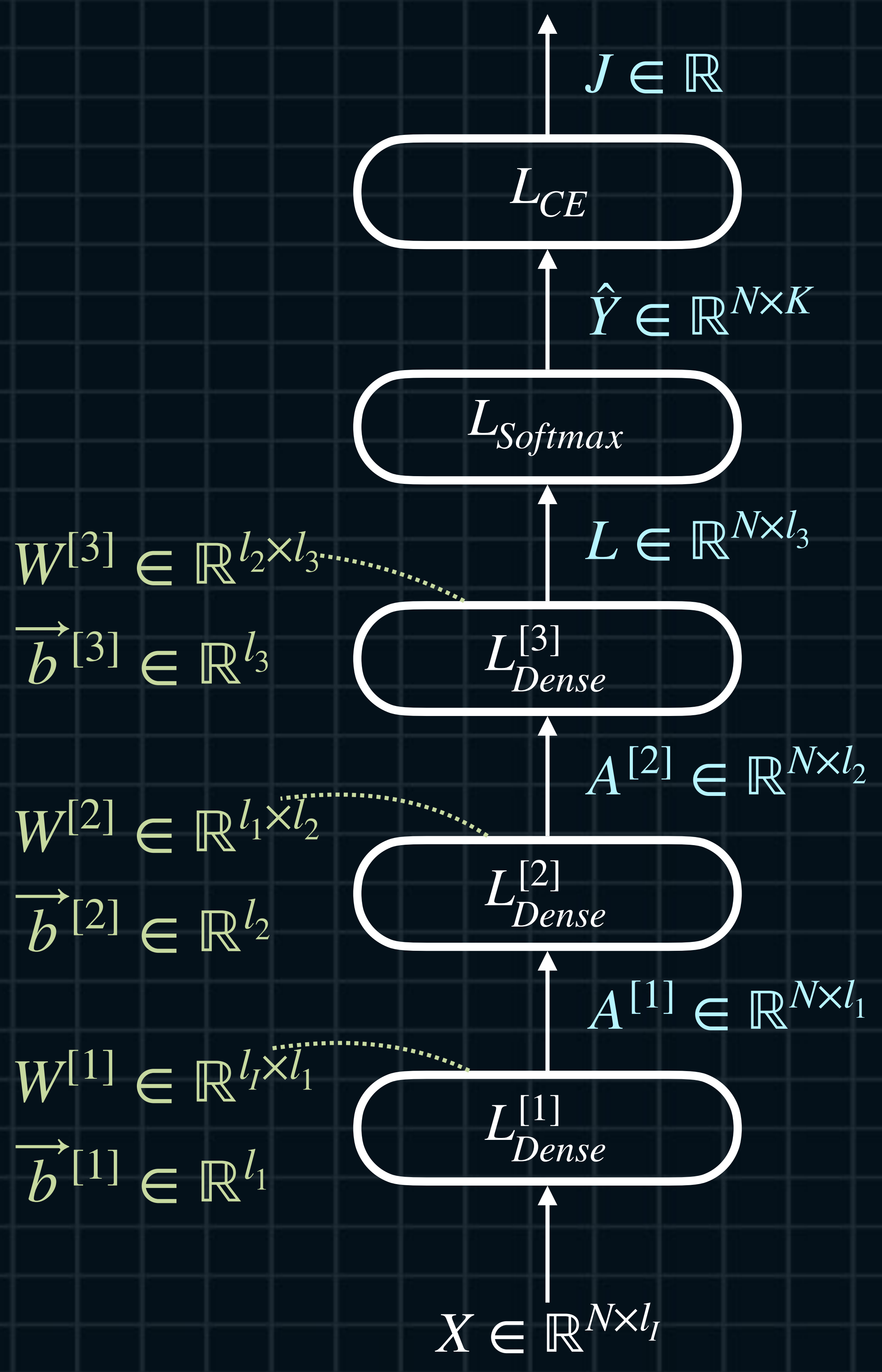
$W^{[2]} \in \mathbb{R}^{l_1 \times l_2}$, $\vec{b}^{[2]} \in \mathbb{R}^{l_2}$ $\quad L^{[2]}_{Dense}$ $\quad A^{[2]} = g\left(Z^{[2]}\right)$, $Z^{[2]} = A^{[1]} \cdot W^{[2]} + \left(\vec{b}^{[2]}\right)^T$

$A^{[1]} \in \mathbb{R}^{N \times l_1}$

$W^{[1]} \in \mathbb{R}^{l_I \times l_1}$, $\vec{b}^{[1]} \in \mathbb{R}^{l_1}$ $\quad L^{[1]}_{Dense}$ $\quad A^{[1]} = g\left(Z^{[1]}\right)$, $Z^{[1]} = X \cdot W^{[1]} + \left(\vec{b}^{[1]}\right)^T$

$X \in \mathbb{R}^{N \times l_I}$

Lecture.11 Applications of Expanded Jacobians - MNIST Classifier and Backpropagation



$J = \mathscr{L}_{CCEE}(Y, \hat{Y})$

$\hat{Y} = Softmax(L)$

$dL = -\dfrac{1}{N}(Y - \hat{Y})$

$L = A^{[2]} \cdot W^{[3]} + \left(\vec{b}^{[3]}\right)^T$

$dA^{[2]} = dL \cdot \left(W^{[3]}\right)^T$ $\quad \mathbb{R}^{N \times l_3} \times \mathbb{R}^{l_3 \times l_2}$

$dW^{[3]} = \left(A^{[2]}\right)^T \cdot dL$ $\quad \mathbb{R}^{l_2 \times N} \times \mathbb{R}^{N \times l_3}$

$dB^{[3]} = sum(dL, axis = 0)$ $\quad \mathbb{R}^{N \times l_3}$

$A^{[2]} = g\left(Z^{[2]}\right)$

$dZ^{[2]} = dA^{[2]} * A^{[2]} * \left(1 - A^{[2]}\right)$

$Z^{[2]} = A^{[1]} \cdot W^{[2]} + \left(\vec{b}^{[2]}\right)^T$

$dA^{[1]} = dZ^{[2]} \cdot \left(W^{[2]}\right)^T$ $\quad \mathbb{R}^{N \times l_2} \times \mathbb{R}^{l_2 \times l_1}$

$dW^{[2]} = \left(A^{[1]}\right)^T \cdot dZ^{[2]}$ $\quad \mathbb{R}^{l_1 \times N} \times \mathbb{R}^{N \times l_2}$

$dB^{[2]} = sum(dZ^{[2]}, axis = 0)$ $\quad \mathbb{R}^{N \times l_2}$

$A^{[1]} = g\left(Z^{[1]}\right)$

$dZ^{[1]} = dA^{[1]} * A^{[1]} * \left(1 - A^{[1]}\right)$

$Z^{[1]} = X \cdot W^{[1]} + \left(\vec{b}^{[1]}\right)^T$

$dW^{[1]} = X^T \cdot dZ^{[1]}$ $\quad \mathbb{R}^{l_I \times N} \times \mathbb{R}^{N \times l_1}$

$d\vec{b}^{[1]} = sum(dZ^{[1]}, axis = 0)$ $\quad \mathbb{R}^{N \times l_1}$

Lecture.11 Applications of Expanded Jacobians - MNIST Classifier Implementation

Learning Env. Setting

```
import numpy as np
from numpy.random import normal
from numpy import zeros

from termcolor import colored
import matplotlib.pyplot as plt
plt.style.use('seaborn')

from tensorflow.keras.datasets.mnist import load_data

(train_images, train_labels), test_ds = load_data()

# set test env.
n_data = train_images.shape[0]
n_feature = train_images.shape[1]*train_images.shape[2]
b_size = 64 # batch size
n_batch = n_data // b_size
epochs = 20
lr = 0.03
units = [64, 32, 10]
```

```
# initialize w, b
W1 = normal(0, 1, (n_feature, units[0]))
B1 = zeros((units[0]))

W2 = normal(0, 1, (units[0], units[1]))
B2 = zeros((units[1]))

W3 = normal(0, 1, (units[1], units[2]))
B3 = zeros((units[2]))

print(colored("W/B Shapes", 'green'))
print(f"W1/B1: {W1.shape}/{B1.shape}")
print(f"W2/B2: {W2.shape}/{B2.shape}")
print(f"W3/B3: {W3.shape}/{B3.shape}\n")
```

Lecture.11 Applications of Expanded Jacobians - MNIST Classifier and Backpropagation

Training

```
losses, accs = list(), list()
for epoch in range(epochs):
  n_correct, n_data = 0, 0
  for b_idx in range(n_batch):
    # get mini-batch
    images = train_images[b_idx*b_size : (b_idx + 1)*b_size, ...]
    X = images.reshape(b_size, -1) / 255.
    Y = train_labels[b_idx*b_size : (b_idx + 1)*b_size]
    # print(X.shape, Y.shape)

    ### forward propagation
    # dense1
    Z1 = X @ W1 + B1
    A1 = 1/(1 + np.exp(-Z1))
    # dense2
    Z2 = A1 @ W2 + B2
    A2 = 1/(1 + np.exp(-Z2))
    # dense3
    L = A2 @ W3 + B3
    Pred = np.exp(L)/np.sum(np.exp(L), axis=1, keepdims=True)
    # loss
    J = np.mean(-np.log(Pred[np.arange(b_size), Y]))
    losses.append(J)

    # calculate accuracy
    Pred_label = np.argmax(Pred, axis=1)
    n_correct += np.sum(Pred_label == Y)
    n_data += b_size

    ### backpropagation
    labels = Y.copy()
    Y = np.zeros_like(Pred)
    Y[np.arange(b_size), labels] = 1
    # loss
    dL = -1/b_size*(Y - Pred)
    # dense3
    dA2 = dL @ W3.T
    dW3 = A2.T @ dL
    dB3 = np.sum(dL, axis=0)
    # dense2
    dZ2 = dA2 * A2*(1-A2)
    dA1 = dZ2 @ W2.T
    dW2 = A1.T @ dZ2
    dB2 = np.sum(dZ2, axis=0)
    # dense1
    dZ1 = dA1 * A1*(1-A1)
    dW1 = X.T @ dZ1
    dB1 = np.sum(dZ1, axis=0)

    # parameter update
    W3, B3 = W3-lr*dW3, B3-lr*dB3
    W2, B2 = W2-lr*dW2, B2-lr*dB2
    W1, B1 = W1-lr*dW1, B1-lr*dB1
  accs.append(n_correct/n_data)
```

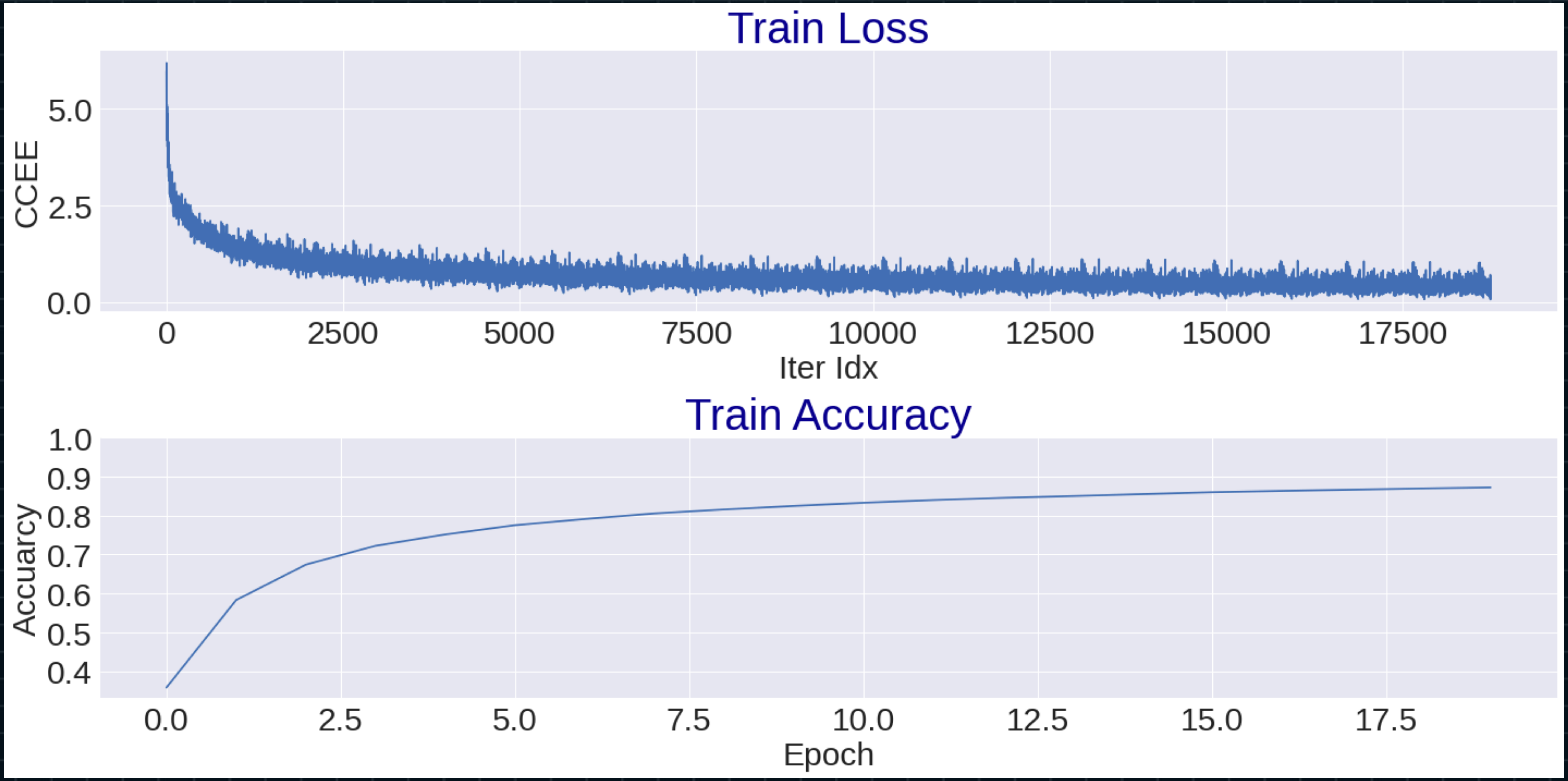
Result Visualization

```
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
axes[0].plot(losses)
axes[1].plot(accs)
axes[0].set_title("Train Loss", color='darkblue', fontsize=40)
axes[0].set_xlabel("Iter Idx", fontsize=30)
axes[0].set_ylabel("CCEE", fontsize=30)

axes[1].set_title("Train Accuracy", color='darkblue', fontsize=40)
axes[1].set_xlabel("Epoch", fontsize=30)
axes[1].set_ylabel("Accuarcy", fontsize=30)
axes[1].set_yticks(np.linspace(0.4, 1.0, 7))

axes[0].tick_params(labelsize=30)
axes[1].tick_params(labelsize=30)
fig.tight_layout()
```

- MNIST Classifier and Backpropagation



Backpropagation and Jacobian Matrices

Lecture.11
Applications of Expanded Jacobians