Train a small neural network (dataset - MNIST classification)

Compare the optimizers:
1. SGD
2. SGD + Momentum
3. Adam
Plot:
1. Training loss vs epochs
2. Accuracy vs epochs

In [1]:
```
from tensorflow.keras.datasets import mnist
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
2026-01-06 15:42:04.098943: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn
them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-01-06 15:42:04.105967: E external/local_xla/xla/stream_executor/cuda/
cuda_fft.cc:467] Unable to register cuFFT factory: Attempting to register
factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
E0000 00:00:1767694324.115136  185700 cuda_dnn.cc:8579] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
E0000 00:00:1767694324.117636  185700 cuda_blas.cc:1407] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
W0000 00:00:1767694324.123933  185700 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1767694324.123946  185700 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1767694324.123947  185700 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1767694324.123948  185700 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
2026-01-06 15:42:04.126372: I tensorflow/core/platform/
cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use
available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

In [2]:
```
data = mnist.load_data()
(train_images, train_labels), (test_images, test_labels) = data
```

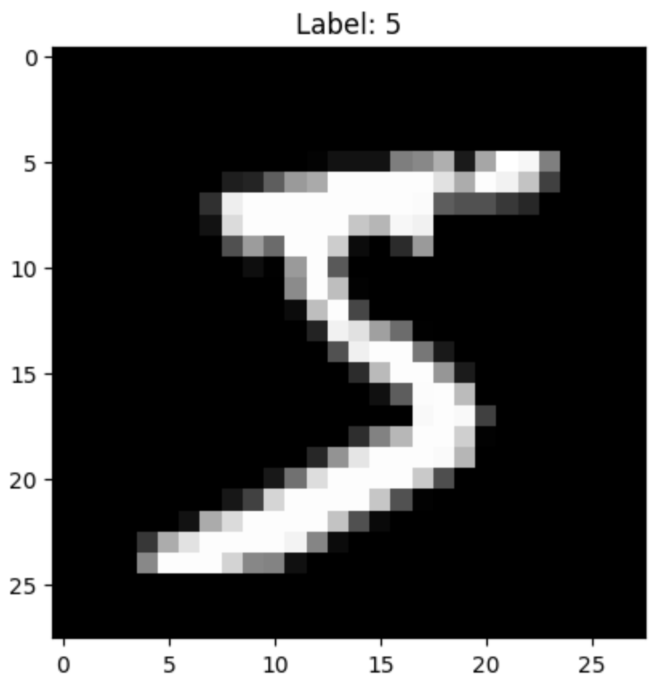
In [3]:
```
img_size =  train_images[0].shape[0]
```

In [4]:
```
num_classes = 10
```

In [5]:
```
train_images = train_images.astype("float32") / 255.0
test_images = test_images.astype("float32") / 255.0
```

In [6]:
```
plt.imshow(train_images[0], cmap='gray')
plt.title(f'Label: {train_labels[0]}')
```

Out[6]:
```
Text(0.5, 1.0, 'Label: 5')
```



In [7]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Flatten
```

In [8]:
```python
from tensorflow.keras.optimizers import SGD, Adam
SGD_nomomentum = SGD()
SGD_momentum = SGD(momentum=0.6)
```

```
---------------------------------------------------------------------------
InternalError                             Traceback (most recent call last)
Cell In[8], line 2
      1 from tensorflow.keras.optimizers import SGD, Adam
----> 2 SGD_nomomentum = SGD()
      3 SGD_momentum = SGD(momentum=0.6)

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/optimizers/sgd.py:60, in SGD.__init__(self, learning_rate,
momentum, nesterov, weight_decay, clipnorm, clipvalue, global_clipnorm,
use_ema, ema_momentum, ema_overwrite_frequency, loss_scale_factor,
gradient_accumulation_steps, name, **kwargs)
     43 def __init__(
     44     self,
     45     learning_rate=0.01,
   (...)
     58     **kwargs,
     59 ):
---> 60     super().__init__(
     61         learning_rate=learning_rate,
     62         name=name,
     63         weight_decay=weight_decay,
     64         clipnorm=clipnorm,
     65         clipvalue=clipvalue,
     66         global_clipnorm=global_clipnorm,
     67         use_ema=use_ema,
     68         ema_momentum=ema_momentum,
     69         ema_overwrite_frequency=ema_overwrite_frequency,
     70         loss_scale_factor=loss_scale_factor,
     71         gradient_accumulation_steps=gradient_accumulation_steps,
     72         **kwargs,
     73     )
     74     if not isinstance(momentum, float) or momentum < 0 or momentum >
1:
     75         raise ValueError("`momentum` must be a float between [0,
1].")

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/backend/tensorflow/optimizer.py:21, in TFOptimizer.__init__(self,
*args, **kwargs)
     20 def __init__(self, *args, **kwargs):
---> 21     super().__init__(*args, **kwargs)
     22     self._distribution_strategy = tf.distribute.get_strategy()

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/optimizers/base_optimizer.py:158, in BaseOptimizer.__init__(self,
learning_rate, weight_decay, clipnorm, clipvalue, global_clipnorm, use_ema,
ema_momentum, ema_overwrite_frequency, loss_scale_factor,
gradient_accumulation_steps, name, **kwargs)
    154 # Create iteration variable
    155 # Note: dtype="int" will resolve to int32 in JAX
    156 # (since int64 is disallowed in JAX) and to int64 in TF.
    157 with backend.name_scope(self.name, caller=self):
--> 158     iterations = backend.Variable(
    159         0,
    160         name="iteration",
    161         dtype="int",
    162         trainable=False,
    163         aggregation="only_first_replica",
    164     )
    165 self._track_variable(iterations)
    166 self._iterations = iterations

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/backend/common/variables.py:173, in Variable.__init__(***failed
resolving arguments***)
    166         raise ValueError(
    167             "When creating a Variable from an initializer, "
    168             "the `shape` argument should be specified. "
    169             f"Received: initializer={initializer} "
    170             f"and shape={shape}"
    171         )
    172 else:
--> 173     initializer = self._convert_to_tensor(initializer, dtype=dtype)
    174     # If dtype is None and `initializer` is an array, use its dtype.
    175     if dtype is None:

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/backend/tensorflow/core.py:73, in Variable._convert_to_tensor(self,
value, dtype)
     72 def _convert_to_tensor(self, value, dtype=None):
---> 73     return convert_to_tensor(value, dtype=dtype)

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
keras/src/backend/tensorflow/core.py:151, in convert_to_tensor(x, dtype,
sparse, ragged)
    146 if not tf.is_tensor(x):
    147     if dtype == "bool" or is_int_dtype(dtype):
    148         # TensorFlow conversion is stricter than other backends, it
does not
    149         # allow ints for bools or floats for ints. We convert without
dtype
    150         # and cast instead.
--> 151         x = tf.convert_to_tensor(x)
    152         return tf.cast(x, dtype)
    153     return tf.convert_to_tensor(x, dtype=dtype)

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
tensorflow/python/util/traceback_utils.py:153, in
filter_traceback.<locals>.error_handler(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153   raise e.with_traceback(filtered_tb) from None
    154 finally:
    155   del filtered_tb

File ~/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-packages/
tensorflow/python/eager/context.py:726, in Context.ensure_initialized(self)
    722   pywrap_tfe.TFE_ContextOptionsSetRunEagerOpAsFunction(opts, True)
    723   pywrap_tfe.TFE_ContextOptionsSetJitCompileRewrite(
    724       opts, self._jit_compile_rewrite
    725   )
--> 726   context_handle = pywrap_tfe.TFE_NewContext(opts)
    727 finally:
    728   pywrap_tfe.TFE_DeleteContextOptions(opts)

InternalError: cudaSetDevice() on GPU:0 failed. Status: out of memory
```

In [ ]:
```
adam_model = Sequential([
    Input(shape =(img_size,img_size) ),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax')
])

adam_model.compile(optimizer = 'adam', loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
adam_history = adam_model.fit(train_images,train_labels, epochs = 15)
```

```
Epoch 1/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 598us/step -
accuracy: 0.8997 - loss: 0.3581
Epoch 2/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 566us/step -
accuracy: 0.9426 - loss: 0.2003
Epoch 3/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 553us/step -
accuracy: 0.9559 - loss: 0.1559
Epoch 4/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 560us/step -
accuracy: 0.9630 - loss: 0.1276
Epoch 5/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 617us/step -
accuracy: 0.9675 - loss: 0.1107
Epoch 6/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 569us/step -
accuracy: 0.9705 - loss: 0.0986
Epoch 7/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 609us/step -
accuracy: 0.9738 - loss: 0.0882
Epoch 8/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 604us/step -
accuracy: 0.9754 - loss: 0.0811
Epoch 9/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 566us/step -
accuracy: 0.9783 - loss: 0.0738
Epoch 10/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 564us/step -
accuracy: 0.9790 - loss: 0.0695
Epoch 11/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 551us/step -
accuracy: 0.9812 - loss: 0.0635
Epoch 12/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 557us/step -
accuracy: 0.9814 - loss: 0.0599
Epoch 13/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 549us/step -
accuracy: 0.9831 - loss: 0.0554
Epoch 14/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 560us/step -
accuracy: 0.9835 - loss: 0.0529
Epoch 15/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 593us/step -
accuracy: 0.9847 - loss: 0.0496
```

In [ ]:
```
SGD_model = Sequential([
    Input(shape =(img_size,img_size) ),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax')
])

SGD_model.compile(optimizer =SGD_nomomentum, loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
SGD_history = SGD_model.fit(train_images,train_labels, epochs = 15)
```

```
Epoch 1/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 550us/step -
accuracy: 0.8143 - loss: 0.7009
Epoch 2/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 557us/step -
accuracy: 0.9014 - loss: 0.3548
Epoch 3/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 548us/step -
accuracy: 0.9131 - loss: 0.3085
Epoch 4/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 531us/step -
accuracy: 0.9200 - loss: 0.2817
Epoch 5/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 519us/step -
accuracy: 0.9255 - loss: 0.2628
Epoch 6/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 515us/step -
accuracy: 0.9296 - loss: 0.2472
Epoch 7/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 508us/step -
accuracy: 0.9340 - loss: 0.2333
Epoch 8/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 523us/step -
accuracy: 0.9377 - loss: 0.2213
Epoch 9/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 519us/step -
accuracy: 0.9409 - loss: 0.2110
Epoch 10/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 515us/step -
accuracy: 0.9428 - loss: 0.2017
Epoch 11/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 513us/step -
accuracy: 0.9453 - loss: 0.1935
Epoch 12/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 518us/step -
accuracy: 0.9469 - loss: 0.1865
Epoch 13/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 516us/step -
accuracy: 0.9495 - loss: 0.1794
Epoch 14/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 511us/step -
accuracy: 0.9505 - loss: 0.1732
Epoch 15/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 518us/step -
accuracy: 0.9529 - loss: 0.1674
```

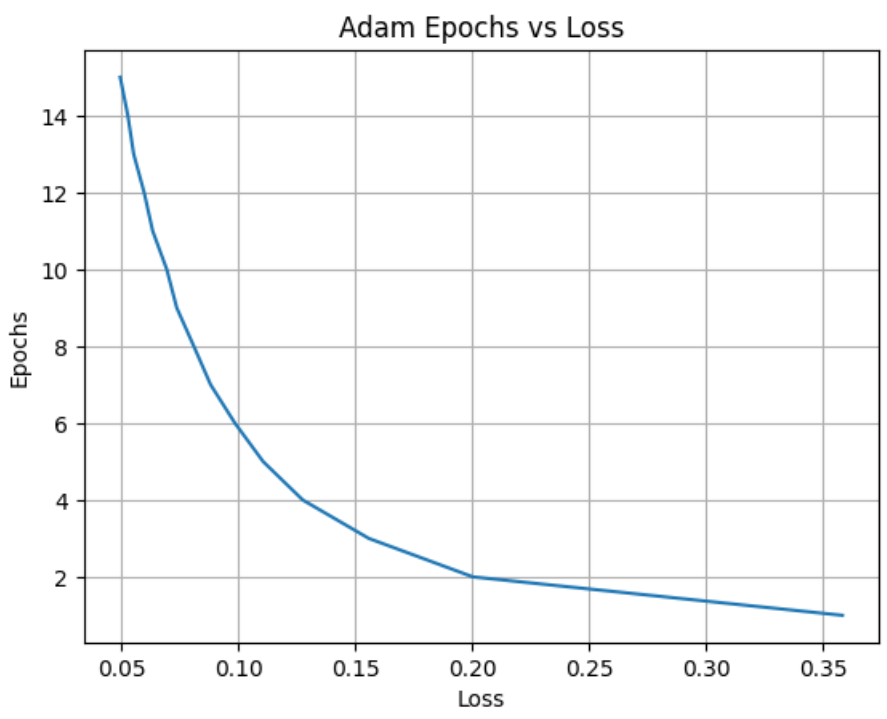
In [ ]:
```
SGD_momentum_model = Sequential([
    Input(shape =(img_size,img_size) ),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax')
])

SGD_momentum_model.compile(optimizer =SGD_momentum, loss =
'sparse_categorical_crossentropy',metrics=['accuracy'])
SGD_momentum_history = SGD_momentum_model.fit(train_images,train_labels, epochs = 15)
```

```
Epoch 1/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 531us/step -
accuracy: 0.8680 - loss: 0.4883
Epoch 2/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 530us/step -
accuracy: 0.9208 - loss: 0.2795
Epoch 3/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 520us/step -
accuracy: 0.9338 - loss: 0.2357
Epoch 4/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 511us/step -
accuracy: 0.9409 - loss: 0.2070
Epoch 5/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 519us/step -
accuracy: 0.9469 - loss: 0.1863
Epoch 6/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 514us/step -
accuracy: 0.9512 - loss: 0.1711
Epoch 7/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 517us/step -
accuracy: 0.9553 - loss: 0.1580
Epoch 8/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 534us/step -
accuracy: 0.9589 - loss: 0.1473
Epoch 9/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 519us/step -
accuracy: 0.9613 - loss: 0.1381
Epoch 10/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 530us/step -
accuracy: 0.9636 - loss: 0.1301
Epoch 11/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 529us/step -
accuracy: 0.9650 - loss: 0.1232
Epoch 12/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 529us/step -
accuracy: 0.9676 - loss: 0.1172
Epoch 13/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m3s[0m 1ms/step - accuracy:
0.9686 - loss: 0.1118
Epoch 14/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 547us/step -
accuracy: 0.9703 - loss: 0.1069
Epoch 15/15
[1m1875/1875[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 552us/step -
accuracy: 0.9712 - loss: 0.1029
```

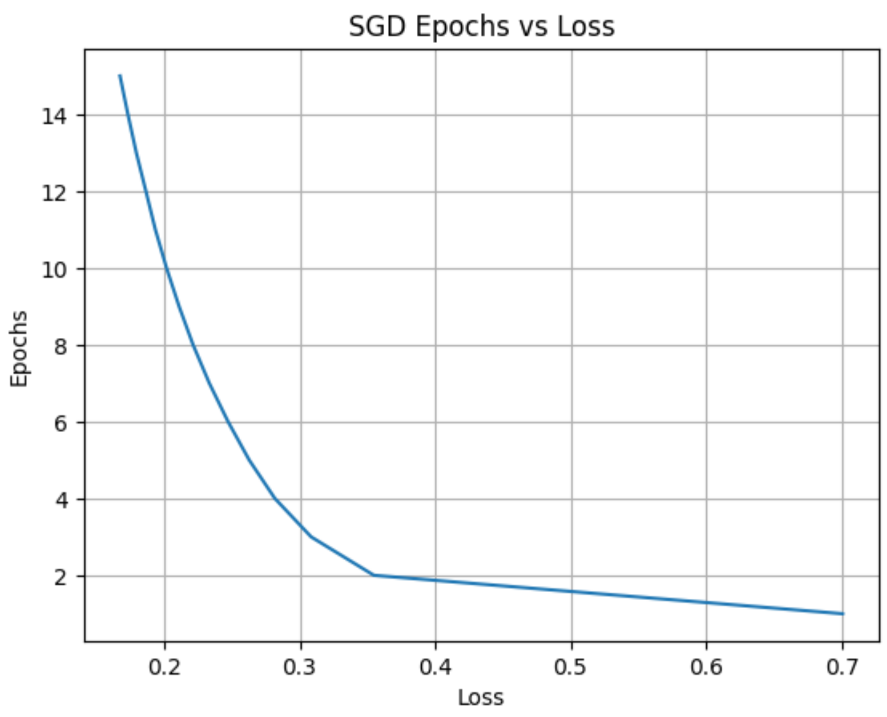
In [ ]:
```
adam_losses = adam_history.history['loss']
SGD_losses = SGD_history.history['loss']
SGD_momentum_losses = SGD_momentum_history.history['loss']
```

In [ ]:
```
x = np.arange(1, 16)
```

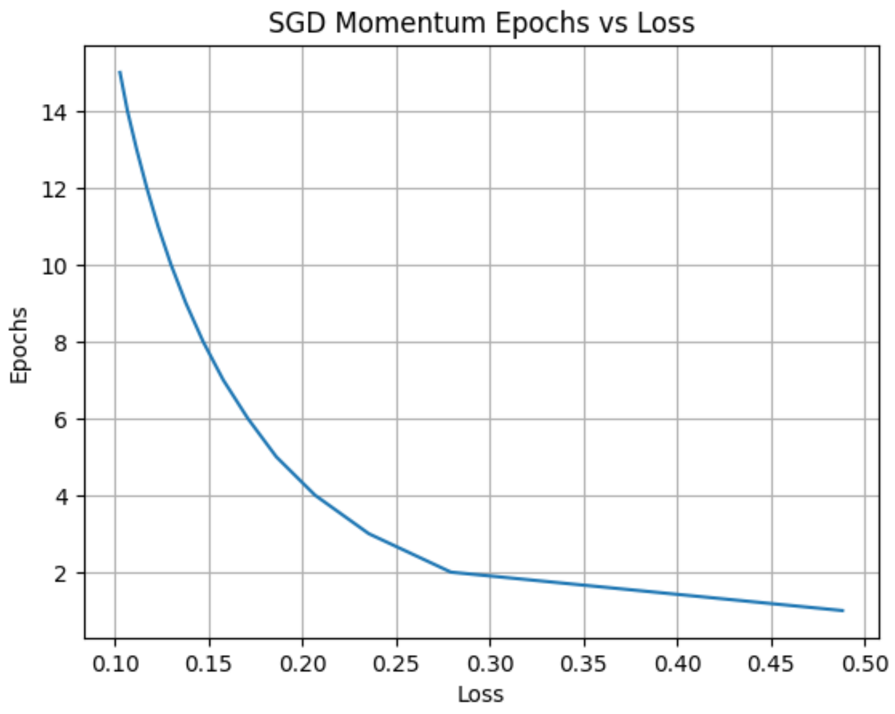
In [ ]:
```
plt.plot(adam_losses, x)
plt.xlabel('Loss')
plt.ylabel('Epochs')
plt.title("Adam Epochs vs Loss")
plt.grid()
```



In [ ]:
```
plt.plot(SGD_losses, x)
plt.xlabel('Loss')
plt.ylabel('Epochs')
plt.title("SGD Epochs vs Loss")
plt.grid()
```

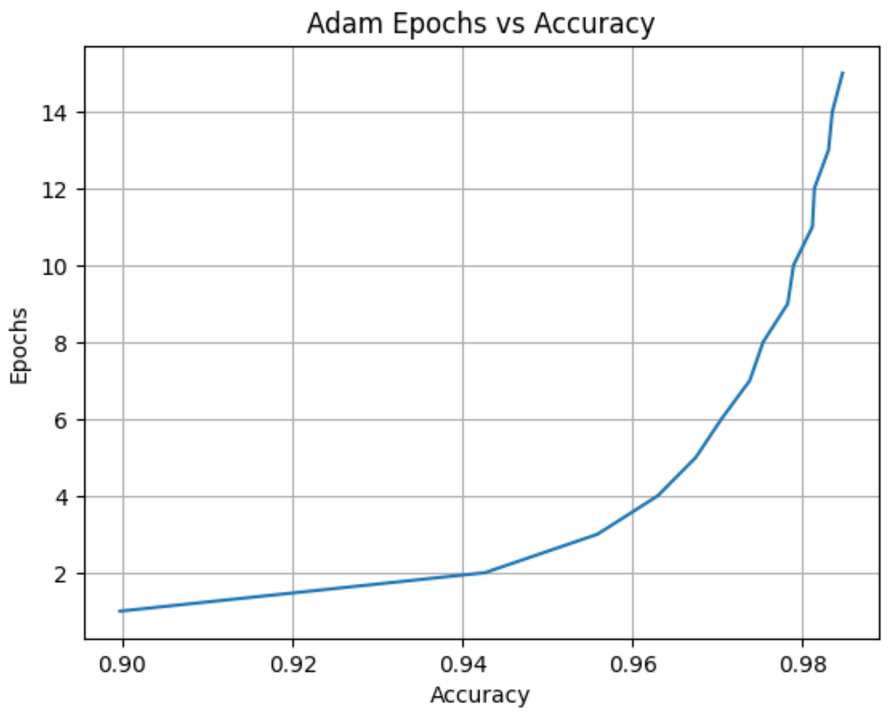


In [ ]:
```
plt.plot(SGD_momentum_losses, x)
plt.xlabel('Loss')
plt.ylabel('Epochs')
plt.title("SGD Momentum Epochs vs Loss")
plt.grid()
```

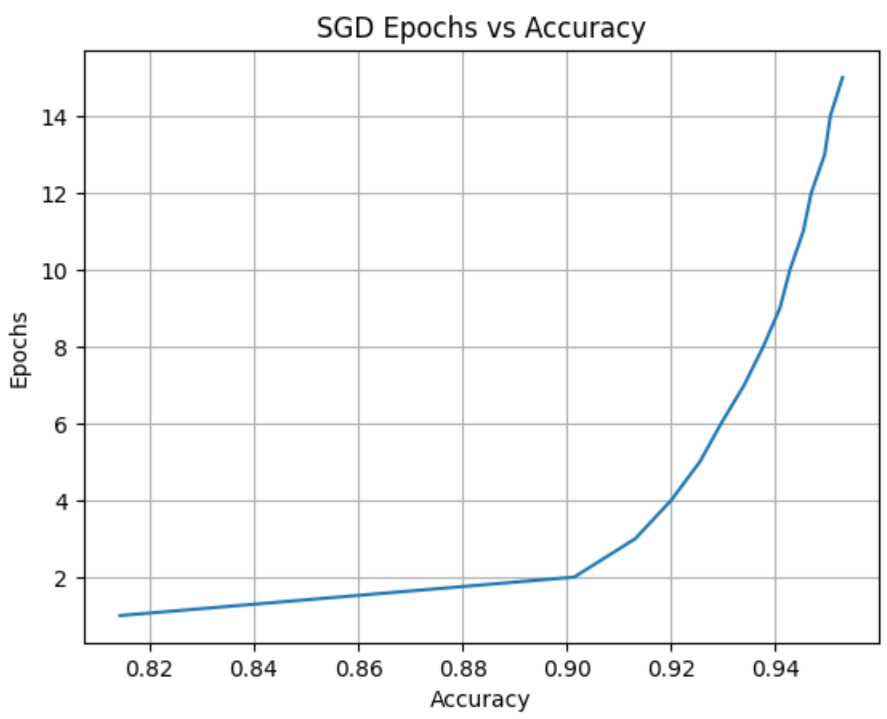


In [ ]:
```
adam_acc = adam_history.history['accuracy']
SGD_acc = SGD_history.history['accuracy']
SGD_momentum_acc = SGD_momentum_history.history['accuracy']
```

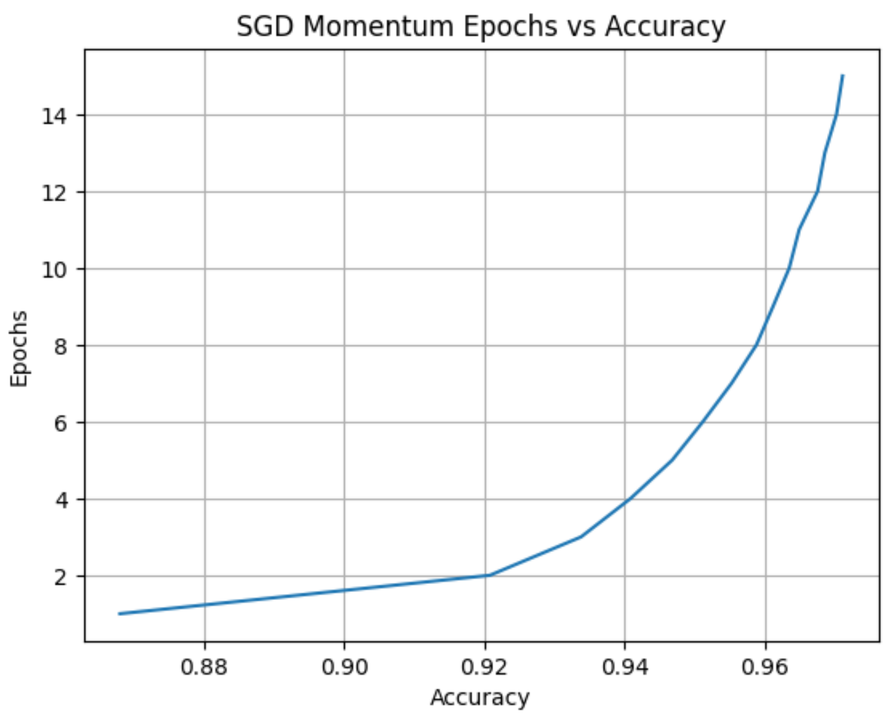
In [ ]:
```
plt.plot(adam_acc, x)
plt.xlabel('Accuracy')
plt.ylabel('Epochs')
plt.title("Adam Epochs vs Accuracy")
plt.grid()
```



In [ ]:
```
plt.plot(SGD_acc, x)
plt.xlabel('Accuracy')
plt.ylabel('Epochs')
plt.title("SGD Epochs vs Accuracy")
plt.grid()
```



In [ ]:
```
plt.plot(SGD_momentum_acc, x)
plt.xlabel('Accuracy')
plt.ylabel('Epochs')
plt.title("SGD Momentum Epochs vs Accuracy")
plt.grid()
```



In [ ]:

```
# Plot ROC AUC Curves for Multi-class Classification (One-vs-Rest)
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import numpy as np
import matplotlib.pyplot as plt

# Get probability predictions for all three models
adam_preds = adam_model.predict(test_images)
sgd_preds = SGD_model.predict(test_images)
sgd_momentum_preds = SGD_momentum_model.predict(test_images)

n_classes = 10

# Binarize the test labels for multi-class ROC AUC
y_bin = label_binarize(test_labels, classes=list(range(n_classes)))

# Function to compute and plot ROC curves
def plot_roc_curves(y_pred_proba, y_bin, model_name, n_classes=10):
    fpr = dict()
    tpr = dict()
    roc_auc_dict = dict()

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_pred_proba[:, i])
        roc_auc_dict[i] = auc(fpr[i], tpr[i])

    # Calculate macro-average AUC
    macro_auc = np.mean(list(roc_auc_dict.values()))

    # Plot ROC curves for each class (showing first 5 classes for clarity)
    plt.figure(figsize=(12, 8))
    colors = plt.cm.tab10(np.linspace(0, 1, n_classes))

    for i in range(min(n_classes, 10)):
        plt.plot(fpr[i], tpr[i], color=colors[i], lw=1.5, alpha=0.7,
                 label=f'Class {i} (AUC = {roc_auc_dict[i]:.3f})')

    plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Classifier')

    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate', fontsize=12)
    plt.ylabel('True Positive Rate', fontsize=12)
    plt.title(f'ROC Curves - MNIST with {model_name}\n(Macro-average AUC =
{macro_auc:.4f})',
              fontsize=14, fontweight='bold')
    plt.legend(loc="lower right", fontsize=9)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    return macro_auc, roc_auc_dict

print("\n" + "="*70)
print("ROC AUC ANALYSIS FOR MNIST CLASSIFICATION")
print("="*70)

# Plot ROC for Adam
print("\n1. ADAM OPTIMIZER")
adam_macro_auc, adam_auc_dict = plot_roc_curves(adam_preds, y_bin, 'Adam')
print(f"Macro-average AUC (Adam): {adam_macro_auc:.4f}")

# Plot ROC for SGD
print("\n2. SGD WITHOUT MOMENTUM")
sgd_macro_auc, sgd_auc_dict = plot_roc_curves(sgd_preds, y_bin, 'SGD (No Momentum)')
print(f"Macro-average AUC (SGD): {sgd_macro_auc:.4f}")

# Plot ROC for SGD + Momentum
print("\n3. SGD WITH MOMENTUM")
sgd_momentum_macro_auc, sgd_momentum_auc_dict = plot_roc_curves(sgd_momentum_preds,
y_bin, 'SGD + Momentum')
print(f"Macro-average AUC (SGD + Momentum): {sgd_momentum_macro_auc:.4f}")

# Comparison plot
print("\n" + "="*70)
print("OPTIMIZER COMPARISON - MACRO-AVERAGE AUC")
print("="*70)
plt.figure(figsize=(10, 6))
optimizers = ['Adam', 'SGD\n(No Momentum)', 'SGD +\nMomentum']
auc_scores = [adam_macro_auc, sgd_macro_auc, sgd_momentum_macro_auc]
colors_bar = ['#FF6B6B', '#4ECDC4', '#45B7D1']

bars = plt.bar(optimizers, auc_scores, color=colors_bar, alpha=0.8,
edgecolor='black', linewidth=2)
plt.ylabel('Macro-average AUC', fontsize=12)
plt.title('MNIST Optimizer Comparison - ROC AUC Score', fontsize=14,
fontweight='bold')
plt.ylim([0, 1.0])
plt.grid(True, alpha=0.3, axis='y')

# Add value labels on bars
for bar, score in zip(bars, auc_scores):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height,
            f'{score:.4f}',
            ha='center', va='bottom', fontsize=12, fontweight='bold')

plt.tight_layout()
plt.show()

print(f"\nBest Optimizer (by AUC): {optimizers[np.argmax(auc_scores)]} with AUC =
{max(auc_scores):.4f}")
```