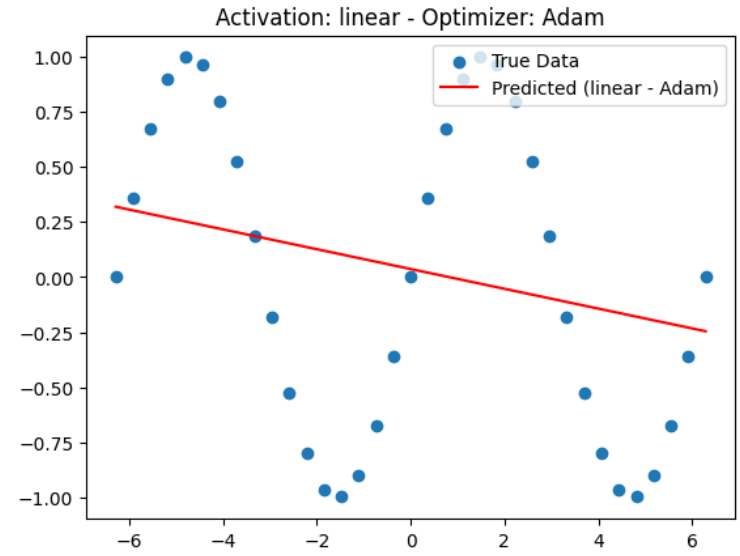
ASSIGNMENT TASK 2

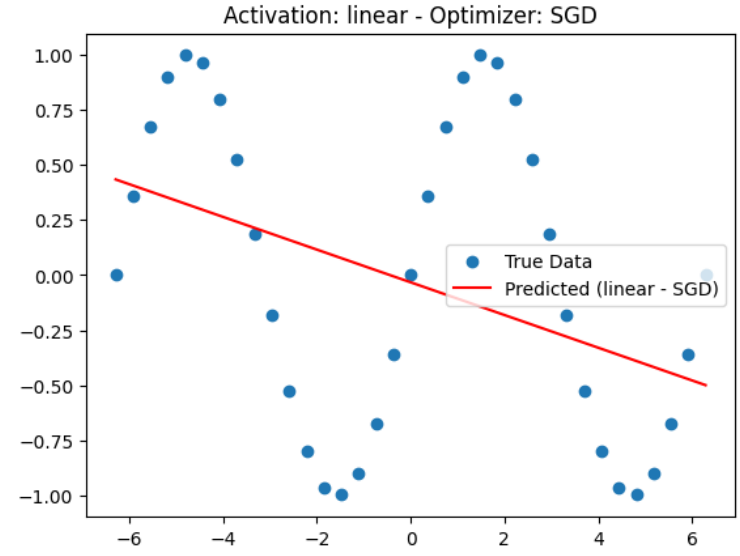
Activation function with epochs 500

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop

# Create sinusoidal dataset
x = np.linspace(-2 * np.pi, 2 * np.pi, 35)
y = np.sin(x)

# Build the neural network model
def build_model(activation_function):
    model = Sequential()
    model.add(Dense(8, input_dim=1, activation=activation_function))  # Hidden Layer 1
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 2
    model.add(Dense(1))  # Output Layer
    return model

# Compile and train the model
def train_model(model, optimizer):
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    model.fit(x, y, epochs=500, verbose=0)
    return model

# Plot the predictions
def plot_predictions(model, activation_function, optimizer_name):
    y_pred = model.predict(x)
    plt.scatter(x, y, label='True Data')
    plt.plot(x, y_pred, label=f'Predicted ({activation_function} - {optimizer_name})', color='red')
    plt.title(f'Activation: {activation_function} - Optimizer: {optimizer_name}')
    plt.legend()
    plt.show()

# Test each activation function
activations = ['linear', 'relu', 'tanh', 'sigmoid']
optimizers = {'Adam': Adam, 'SGD': SGD, 'RMSprop': RMSprop}

for activation in activations:
    for optimizer_name, optimizer_class in optimizers.items():
        # Build a new model for each combination of activation and optimizer
        model = build_model(activation)
        # Create a new instance of the optimizer for each model
        optimizer = optimizer_class()
        trained_model = train_model(model, optimizer)
        plot_predictions(trained_model, activation, optimizer_name)
```

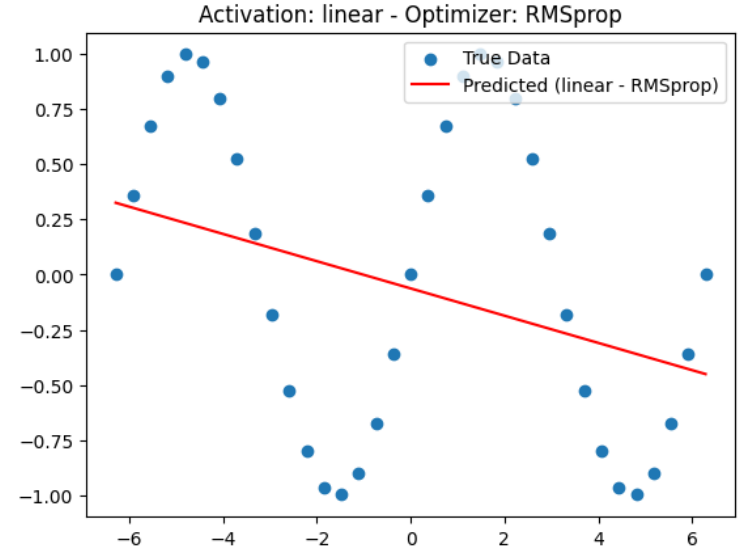
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```



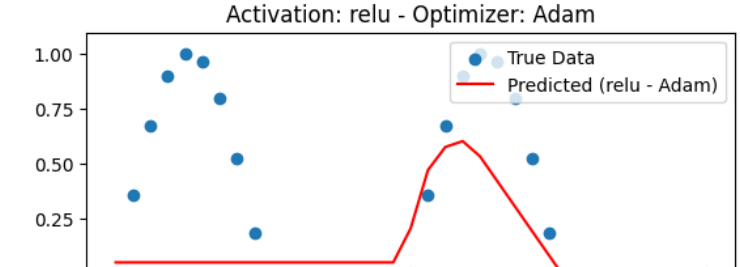

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

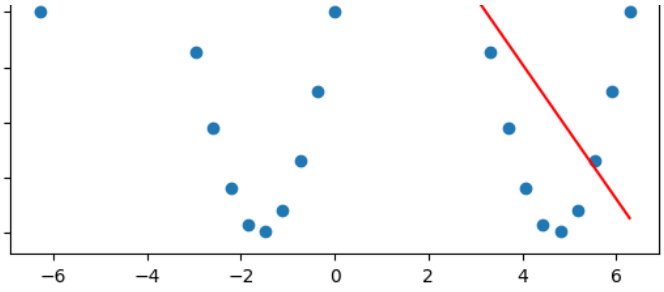



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```



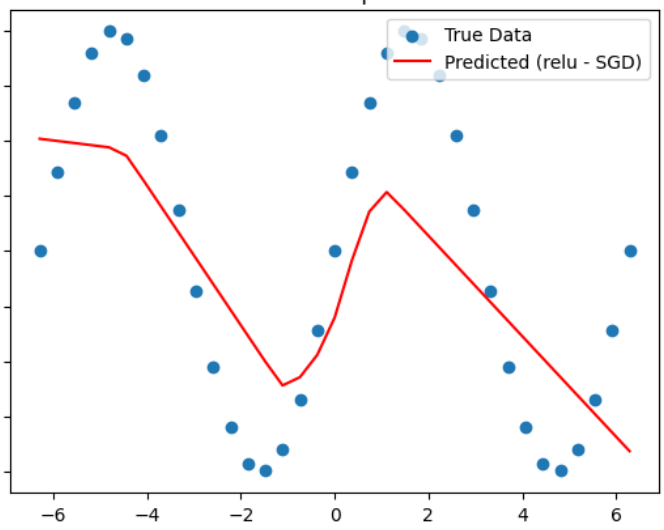

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```





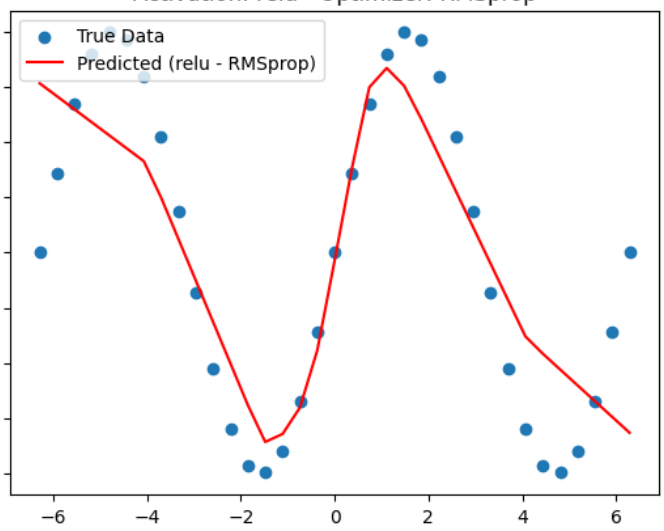
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```

Activation: relu - Optimizer: SGD



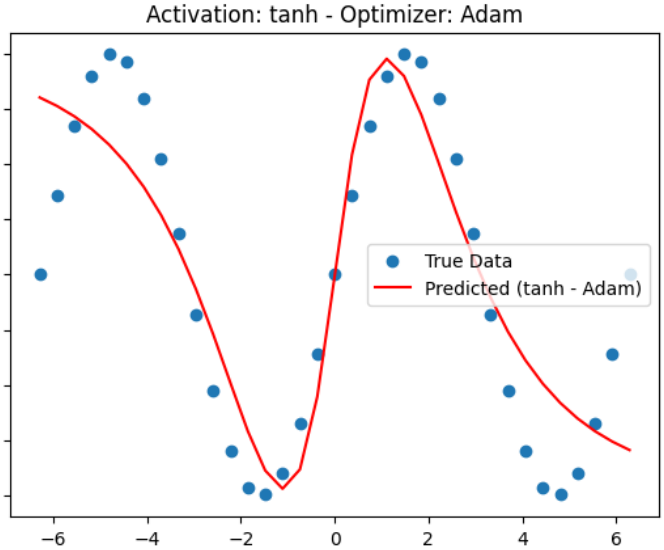

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```

Activation: relu - Optimizer: RMSprop

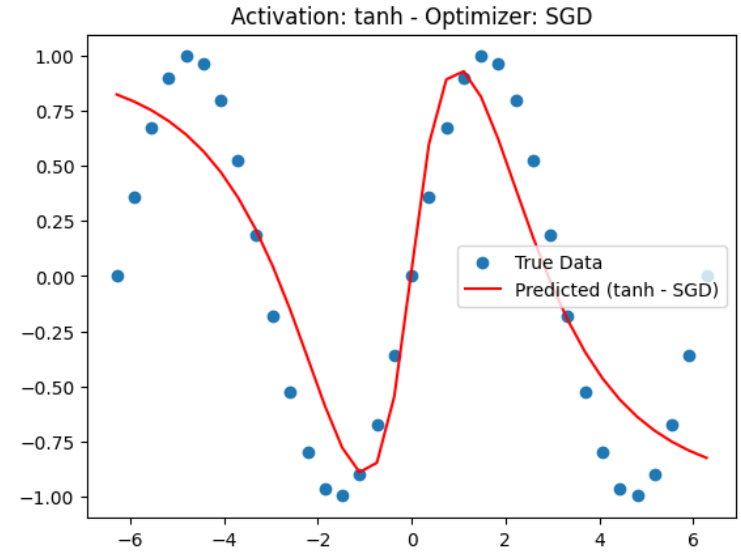



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```

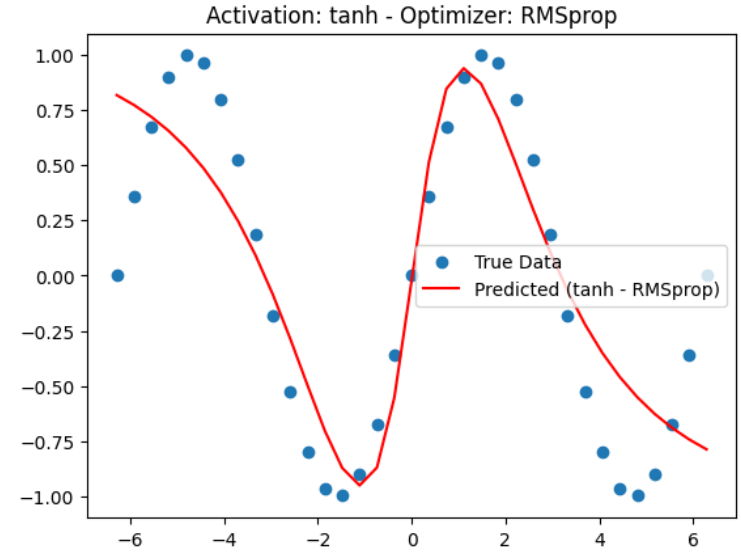
Activation: tanh - Optimizer: Adam



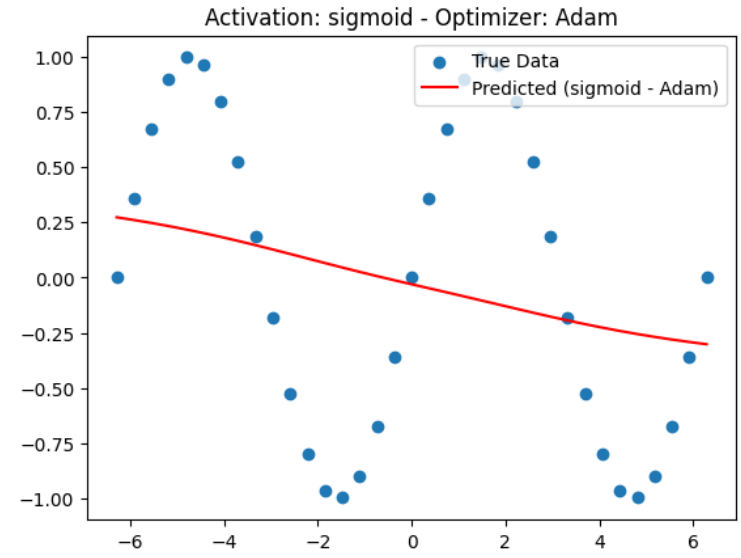

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
```



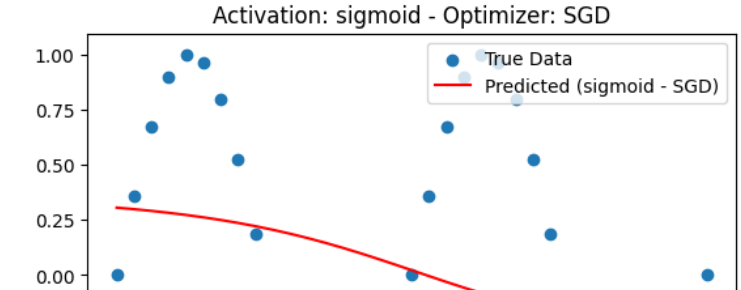

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```

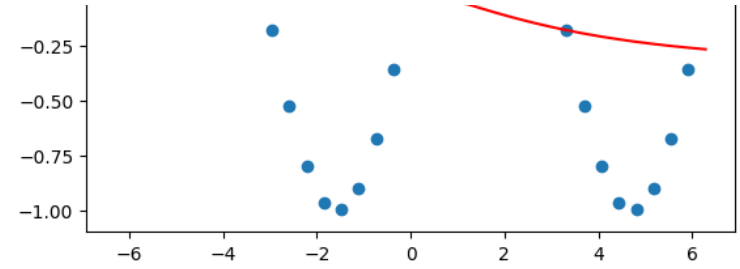



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```

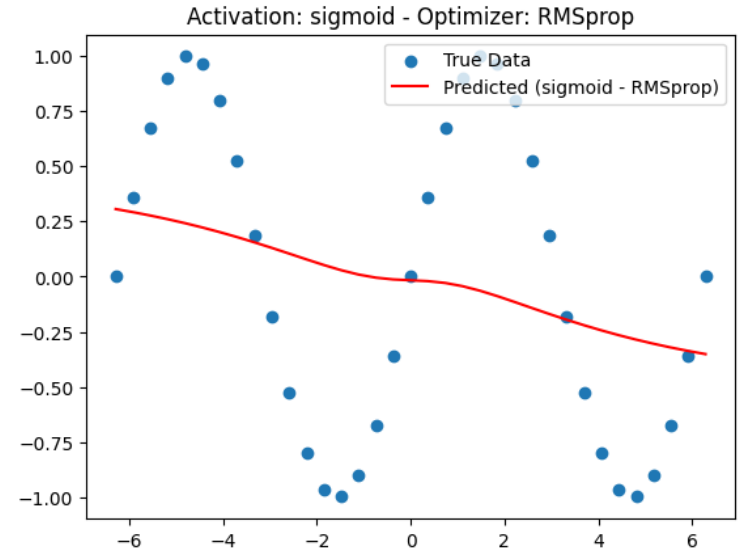



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
```





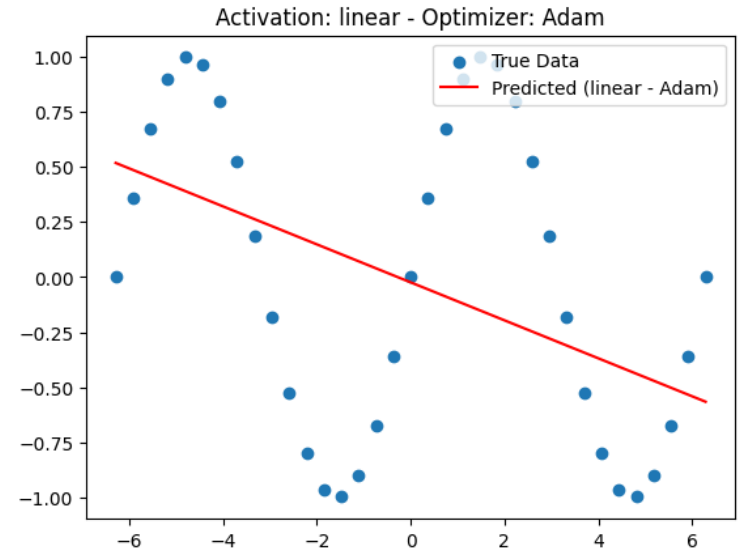

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```



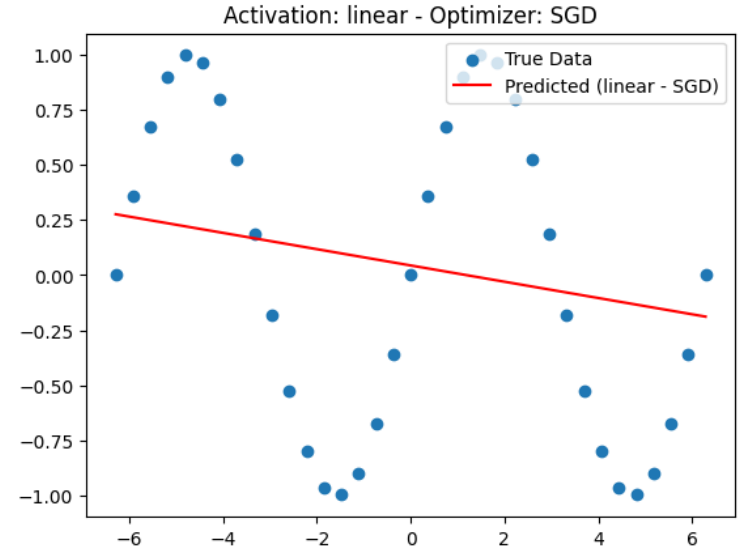
Activation functions with epochs 1000

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop

# Create sinusoidal dataset
x = np.linspace(-2 * np.pi, 2 * np.pi, 35)
y = np.sin(x)

# Build the neural network model
def build_model(activation_function):
    model = Sequential()
    model.add(Dense(8, input_dim=1, activation=activation_function))  # Hidden Layer 1
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 2
    model.add(Dense(1))  # Output Layer
    return model

# Compile and train the model
def train_model(model, optimizer):
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    model.fit(x, y, epochs=1000, verbose=0)
    return model

# Plot the predictions
def plot_predictions(model, activation_function, optimizer_name):
    y_pred = model.predict(x)
    plt.scatter(x, y, label='True Data')
    plt.plot(x, y_pred, label=f'Predicted ({activation_function} - {optimizer_name})', color='red')
    plt.title(f'Activation: {activation_function} - Optimizer: {optimizer_name}')
    plt.legend()
    plt.show()

# Test each activation function
activations = ['linear', 'relu', 'tanh', 'sigmoid']
optimizers = {'Adam': Adam, 'SGD': SGD, 'RMSprop': RMSprop}

for activation in activations:
    for optimizer_name, optimizer_class in optimizers.items():
        # Build a new model for each combination of activation and optimizer
        model = build_model(activation)
        # Create a new instance of the optimizer for each model
        optimizer = optimizer_class()
        trained_model = train_model(model, optimizer)
        plot_predictions(trained_model, activation, optimizer_name)
```

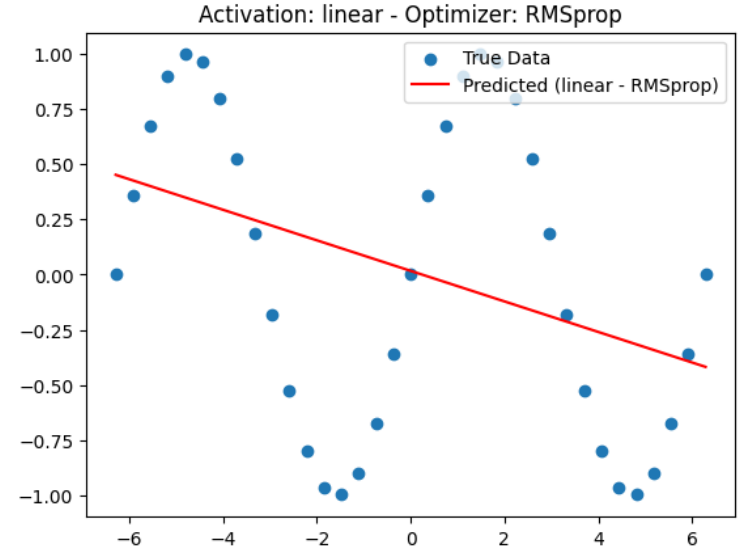
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
```



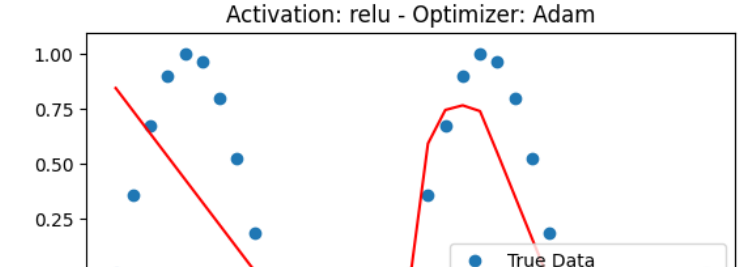

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
```

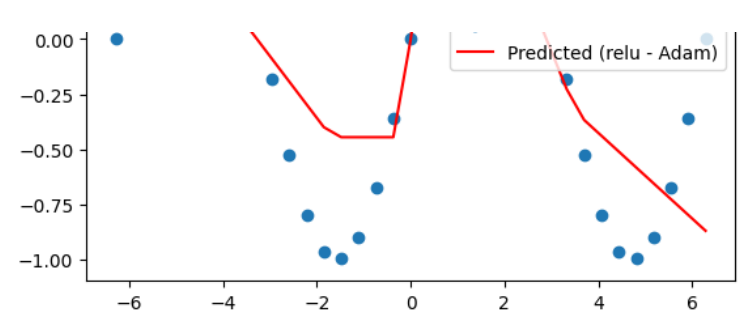



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
```



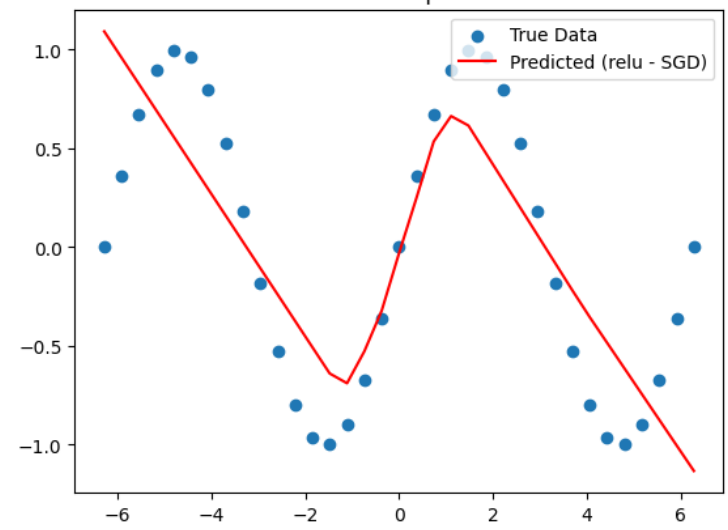

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
```





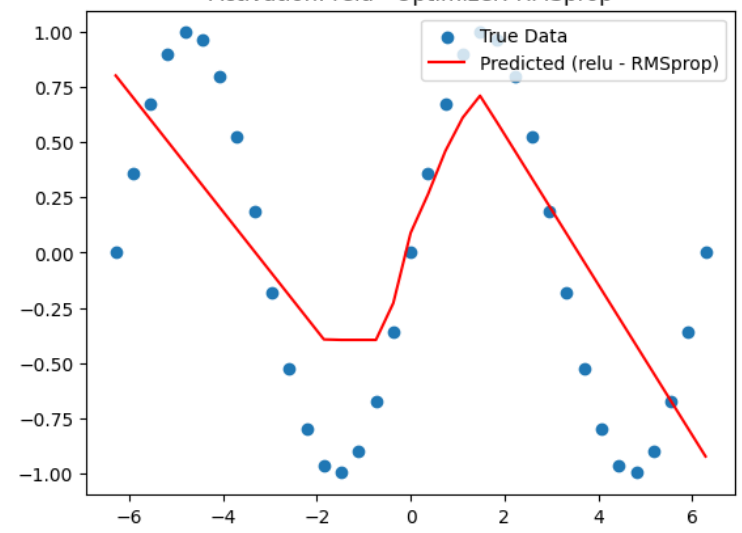

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
```

Activation: relu - Optimizer: SGD



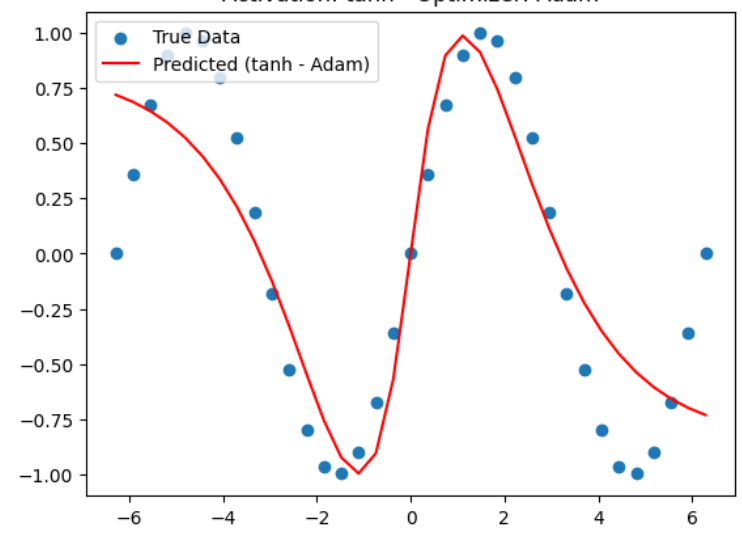

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```

Activation: relu - Optimizer: RMSprop

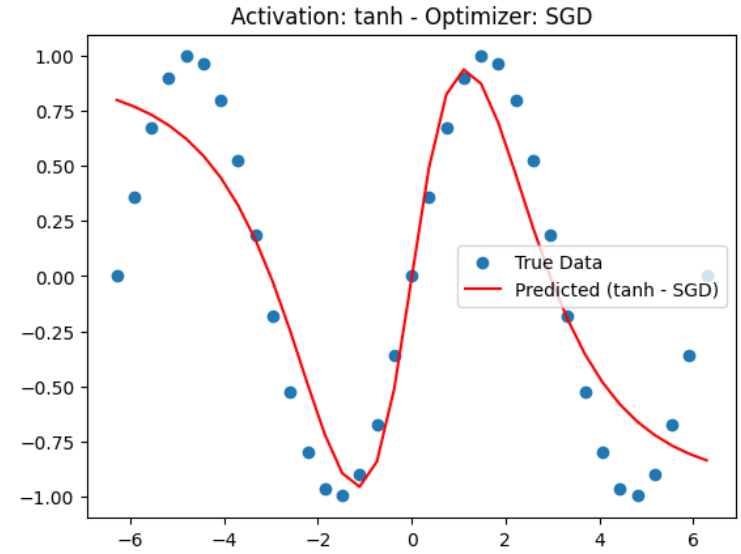



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```

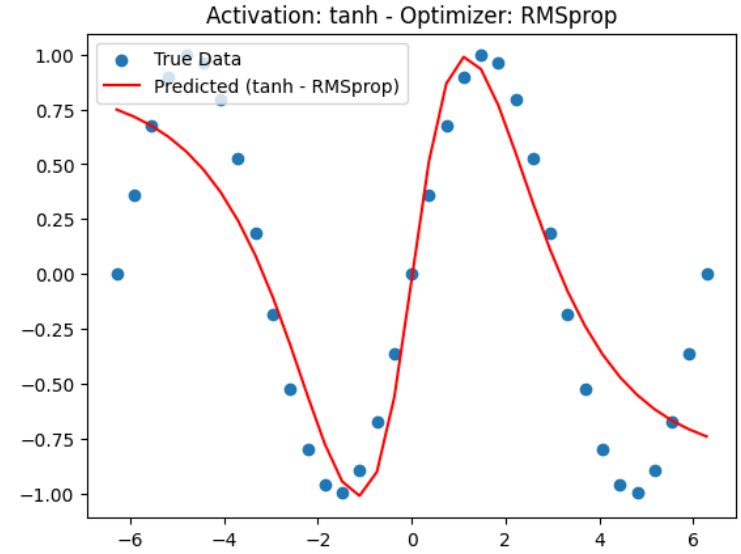
Activation: tanh - Optimizer: Adam



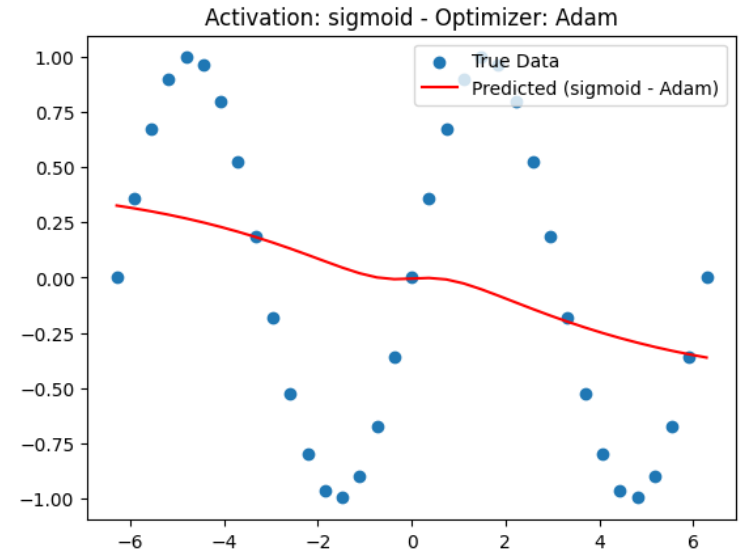

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```



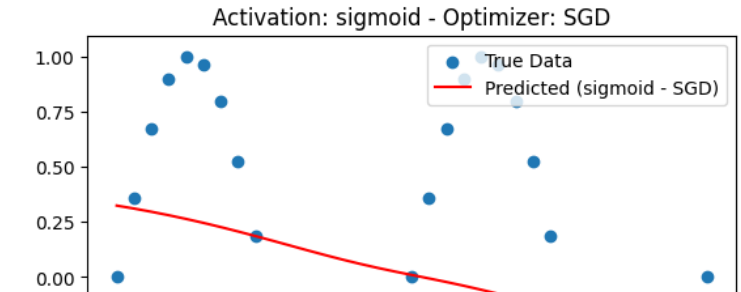

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
```

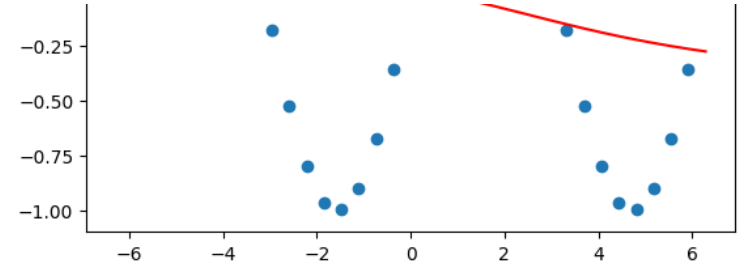



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
```

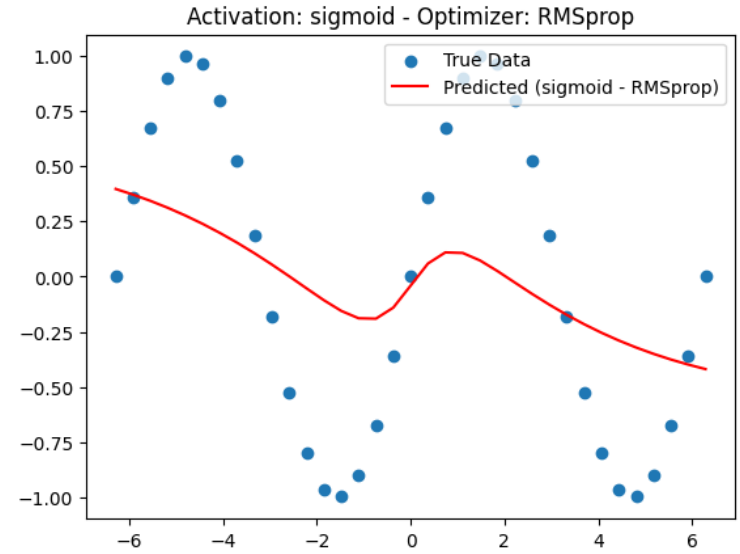



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
```





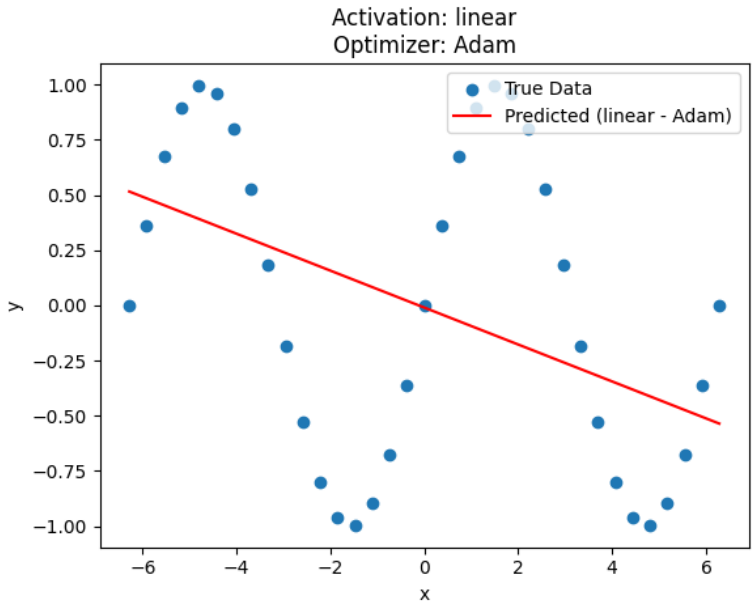

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```



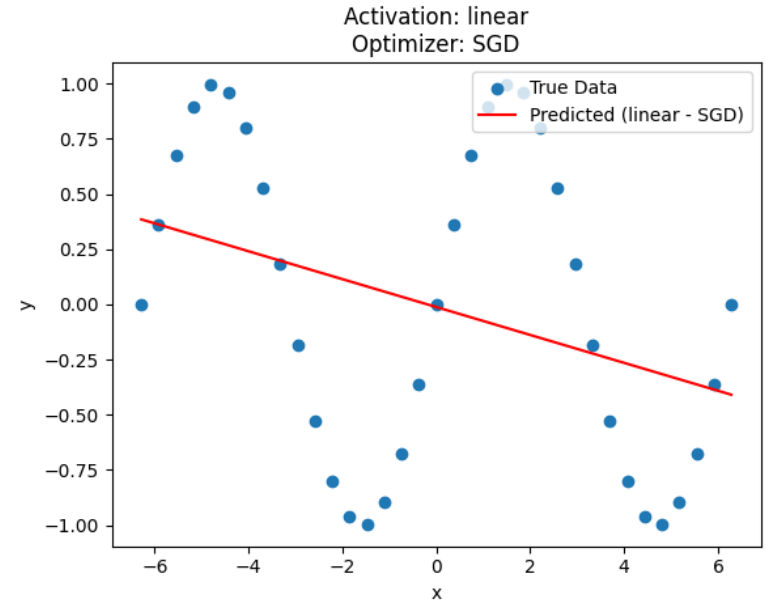
Activation function with added layer.

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop

# Create sinusoidal dataset
x = np.linspace(-2 * np.pi, 2 * np.pi, 35)
y = np.sin(x)

# Build the neural network model with an additional hidden layer
def build_model(activation_function):
    model = Sequential()
    model.add(Dense(8, input_dim=1, activation=activation_function))  # Hidden Layer 1
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 2
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 3 (New Layer)
    model.add(Dense(1))  # Output Layer
    return model

# Compile and train the model
def train_model(model, optimizer):
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    model.fit(x, y, epochs=200, verbose=0)
    return model

# Plot the predictions
def plot_predictions(model, activation_function, optimizer_name):
    y_pred = model.predict(x)
    plt.scatter(x, y, label='True Data')
    plt.plot(x, y_pred, label=f'Predicted ({activation_function} - {optimizer_name})', color='red')
    plt.title(f'Activation: {activation_function}\nOptimizer: {optimizer_name}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.show()

# Test each activation function with different optimizers
activations = ['linear', 'relu', 'tanh', 'sigmoid']
optimizers = {'Adam': Adam, 'SGD': SGD, 'RMSprop': RMSprop}

for activation in activations:
    for optimizer_name, optimizer_class in optimizers.items():
        # Build a new model for each combination of activation and optimizer
        model = build_model(activation)
        # Create a new instance of the optimizer for each model
        optimizer = optimizer_class()
        trained_model = train_model(model, optimizer)
        plot_predictions(trained_model, activation, optimizer_name)
```

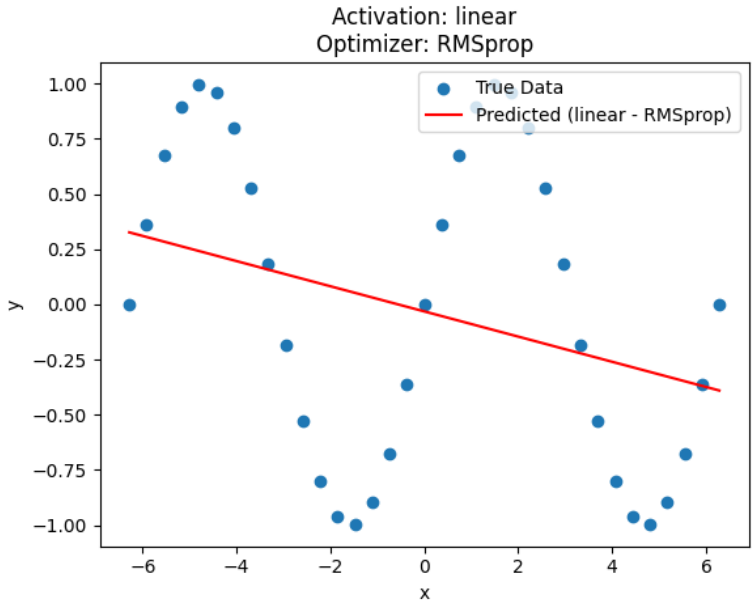
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```



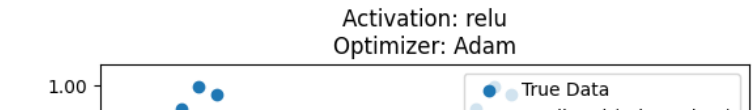

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
```

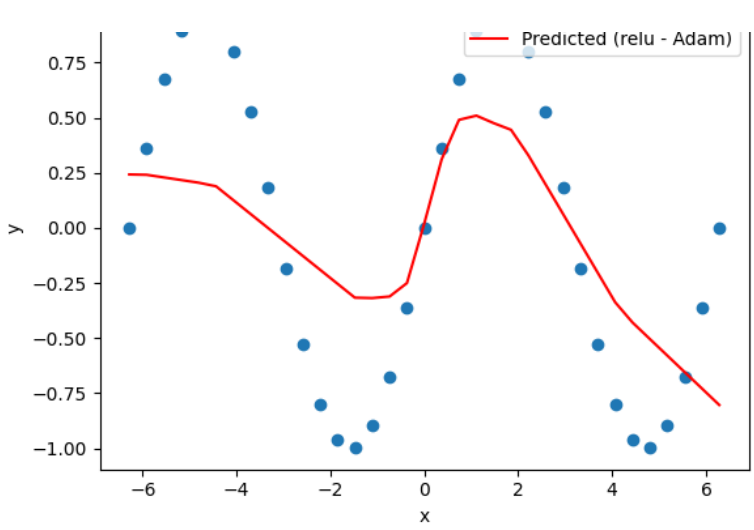



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
```



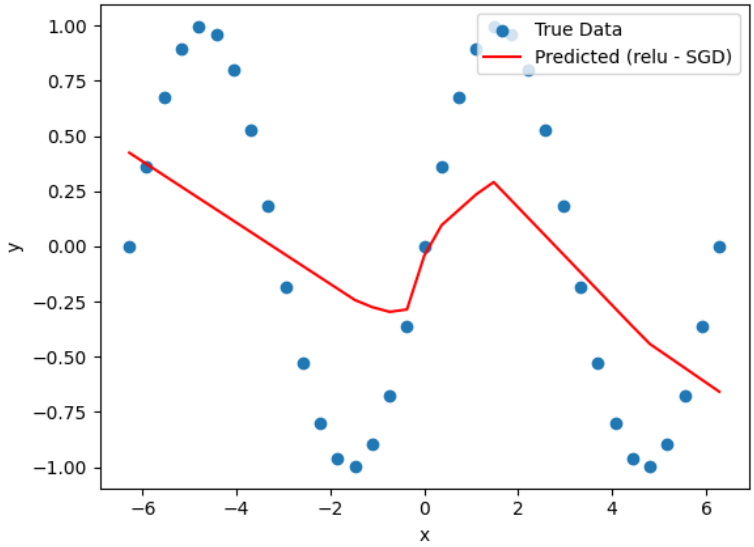

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
```





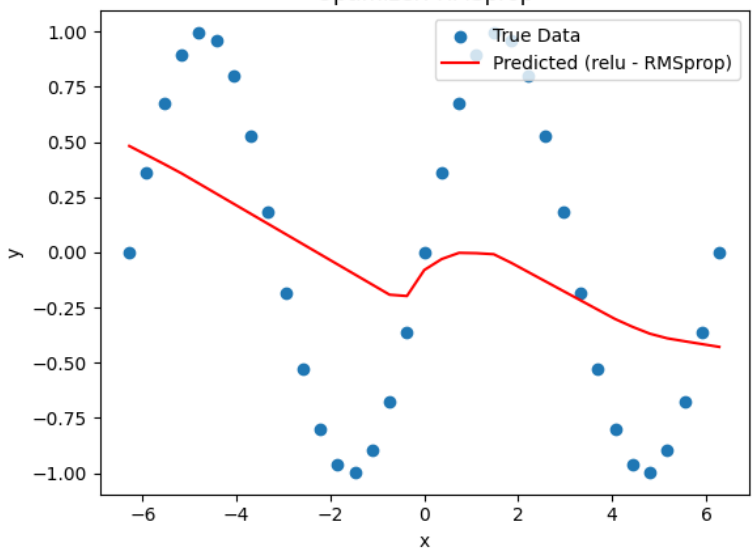

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
```

Activation: relu
Optimizer: SGD



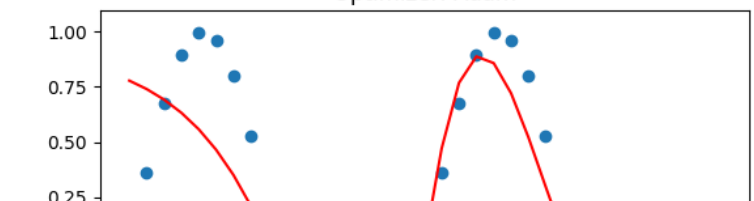

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
```

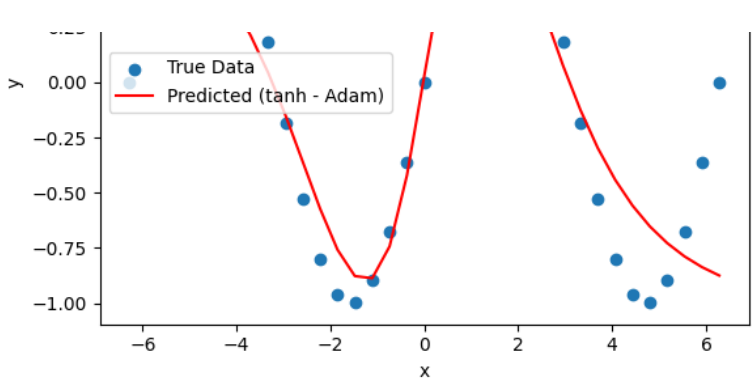
Activation: relu
Optimizer: RMSprop




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
```

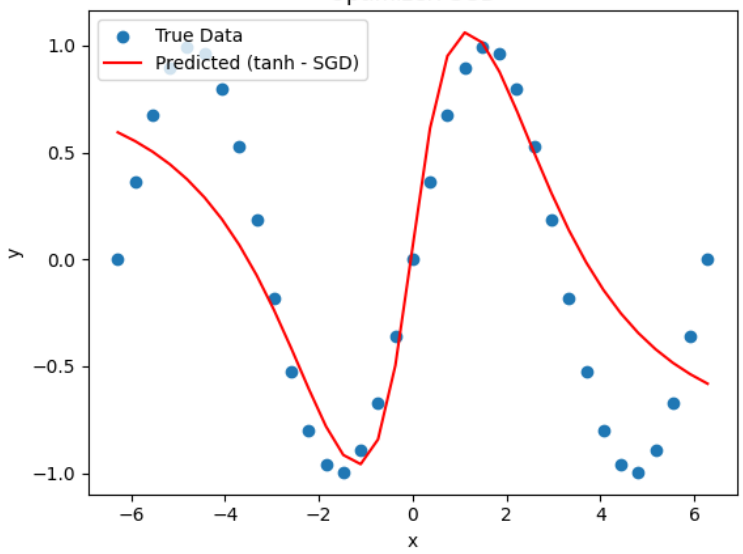
Activation: tanh
Optimizer: Adam





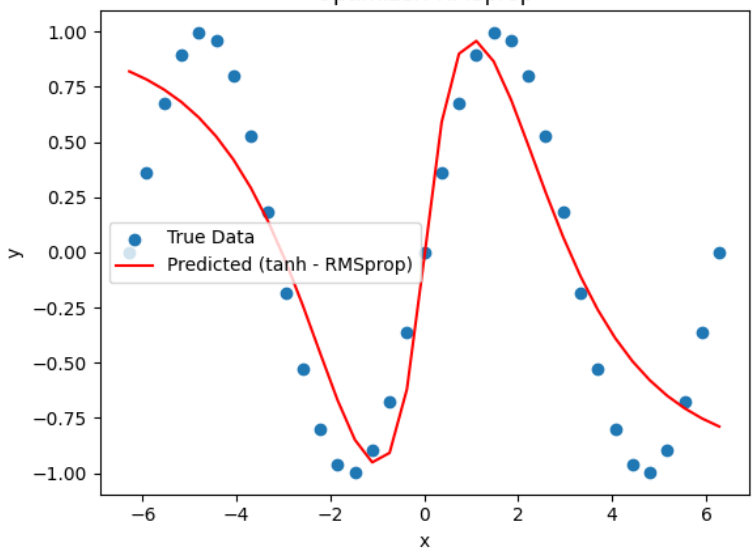

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
```

Activation: tanh
Optimizer: SGD



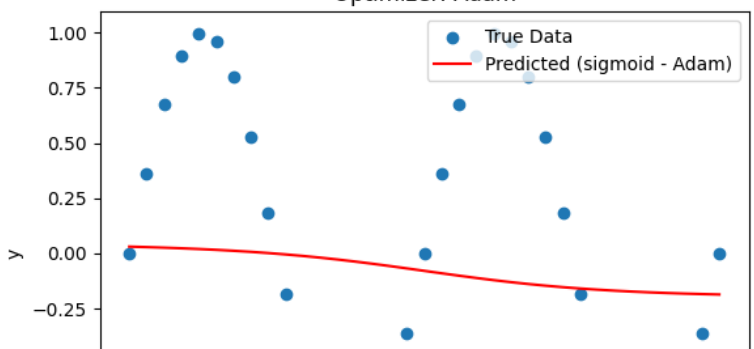

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

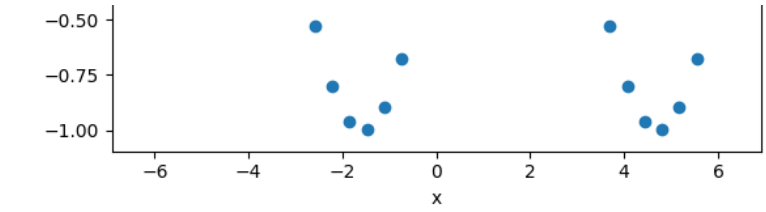
Activation: tanh
Optimizer: RMSprop



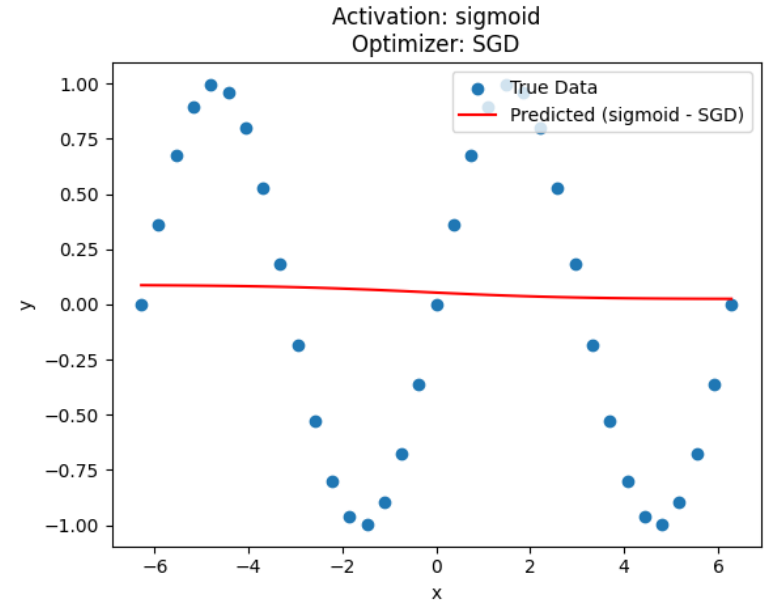

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
```

Activation: sigmoid
Optimizer: Adam

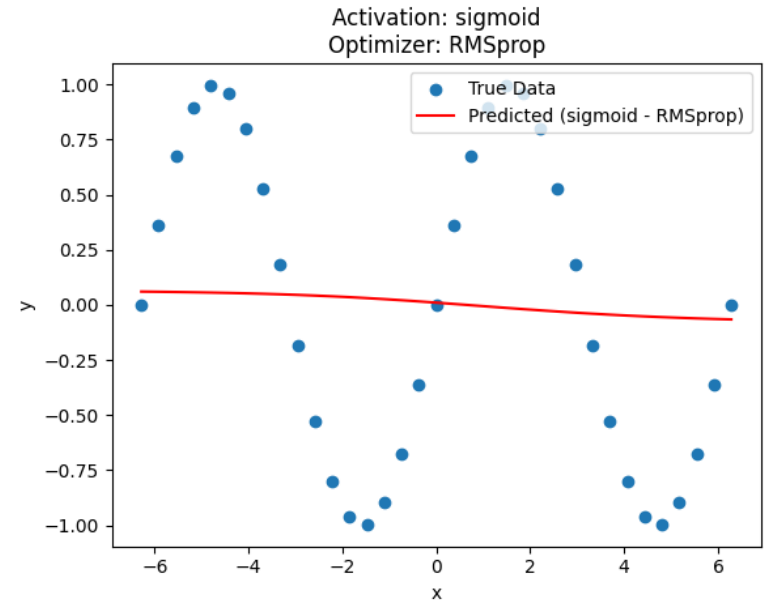





```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
```



Discussion of the Best Activation Function

For this particular problem, Tanh is the best activation function.

Why Tanh Provided the Best Fit:

Non-linear nature: The Tanh activation function is non-linear, which allows the model to handle the complexity of the sinusoidal data. Since the sine wave has both positive and negative values, Tanh, with its output range between -1 and 1, fits the oscillating pattern well.

Smoothness: The smooth gradient of Tanh helps avoid sudden changes in the predicted outputs, unlike ReLU, which might zero out negative values and struggle with fitting negative portions of the sine wave.

Better convergence: In combination with the Adam optimizer, Tanh provided faster and more accurate convergence compared to other activation functions, as Adam dynamically adapts the learning rate and stabilizes the updates.

Additional Observations and Insights

Impact of Changing the Number of Epochs:

Accuracy vs. Time Complexity: Increasing the number of epochs leads to more accurate results as the model has more time to learn the underlying pattern. However, this comes at the cost of increased time complexity. For instance, when increasing epochs significantly, the model performance improves, but the training time also grows substantially.

Impact of Changing the Number of Layers:

Deeper networks provide better accuracy: Increasing the number of hidden layers in the neural network also improved the model's ability to fit the data. The additional layers allow the network to learn more complex patterns, resulting in better accuracy. However, deeper networks can also be prone to overfitting, so proper regularization techniques might be necessary.

Added layer with increased epoch

ADAM optimizer worked well than other optimizers:

Adaptive Learning Rates: Adam dynamically adjusts the learning rate for each parameter, making it more efficient at handling the non-linear nature of the sinusoidal data compared to fixed-rate optimizers like SGD.

Momentum: Incorporating momentum allows Adam to smooth out updates, preventing oscillations and accelerating convergence, especially helpful for the fluctuating peaks and valleys of sine wave data.

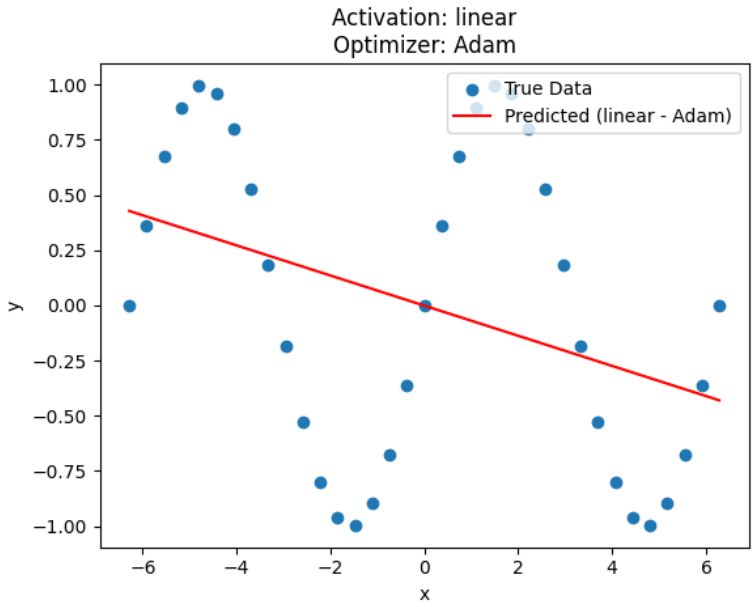
Bias-Correction: Adam's bias-correction ensures stable and accurate updates in the early stages of training, preventing issues with noisy or sparse gradients.

Combination of RMSprop and Momentum: Adam combines RMSprop's learning rate adaptation with momentum, making it faster and more stable than other optimizers.

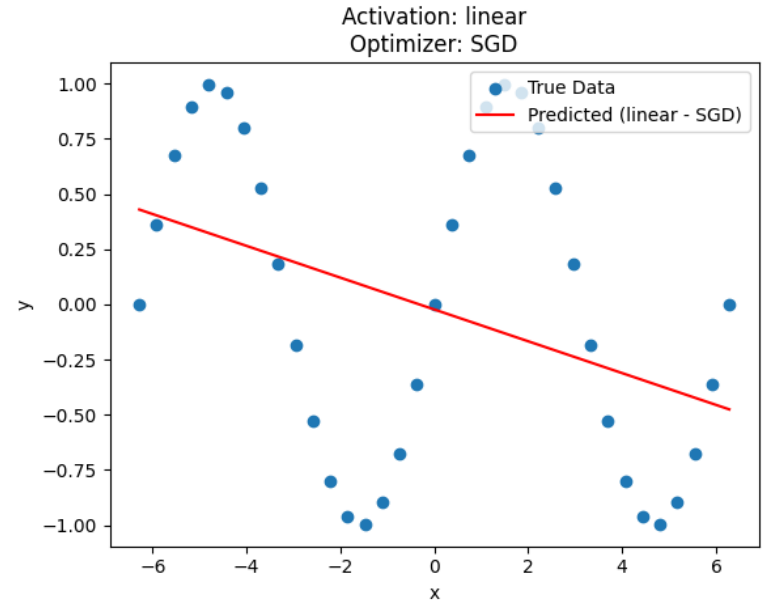
Handling Sparse Gradients: Adam excels at managing sparse gradients, allowing it to update parameters more flexibly, resulting in better fitting to complex data like the sinusoidal pattern.

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop

# Create sinusoidal dataset
x = np.linspace(-2 * np.pi, 2 * np.pi, 35)
y = np.sin(x)

# Build the neural network model with an additional hidden layer
def build_model(activation_function):
    model = Sequential()
    model.add(Dense(8, input_dim=1, activation=activation_function))  # Hidden Layer 1
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 2
    model.add(Dense(8, activation=activation_function))  # Hidden Layer 3 (New Layer)
    model.add(Dense(1))  # Output Layer
    return model

# Compile and train the model
def train_model(model, optimizer):
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    model.fit(x, y, epochs=500, verbose=0)
    return model

# Plot the predictions
def plot_predictions(model, activation_function, optimizer_name):
    y_pred = model.predict(x)
    plt.scatter(x, y, label='True Data')
    plt.plot(x, y_pred, label=f'Predicted ({activation_function} - {optimizer_name})', color='red')
    plt.title(f'Activation: {activation_function}\nOptimizer: {optimizer_name}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.show()

# Test each activation function with different optimizers
activations = ['linear', 'relu', 'tanh', 'sigmoid']
optimizers = {'Adam': Adam, 'SGD': SGD, 'RMSprop': RMSprop}

for activation in activations:
    for optimizer_name, optimizer_class in optimizers.items():
        # Build a new model for each combination of activation and optimizer
        model = build_model(activation)
        # Create a new instance of the optimizer for each model
        optimizer = optimizer_class()
        trained_model = train_model(model, optimizer)
        plot_predictions(trained_model, activation, optimizer_name)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```




```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```