■ MLP Decision Boundaries

Comparing Activation Functions on make_moons

Objective: Understand how ReLU, Sigmoid, and Tanh create different decision boundaries

- How do different activations shape boundaries?
- Which activation works best?
- Why does the choice matter?

■ The Challenge

Scenario: Classify points in the 'two moons' pattern

The make_moons dataset creates two interleaving half-moon shapes.

Challenge: A straight line CANNOT separate these!

Solution: Use neural networks with different activation functions

■ Real-World Applications

Domain	Example
Medical	Classify tumors as benign/malignant
Email	Spam vs legitimate email
Finance	Fraud vs normal transactions
Vision	Cat vs dog in images

Key Insight: Real data is rarely linearly separable!

■ The make_moons Dataset

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=300, noise=0.2, random_state=42)
```

Parameter	Value	Purpose
n_samples	300	Total data points
noise	0.2	Adds realism
random_state	42	Reproducibility

■ Key Concepts

1. Neural Network (MLP)

- Learns complex patterns through layers of neurons
- Architecture: Input → Hidden → Output

2. Activation Functions

- ReLU, Sigmoid, Tanh
- Transform neuron outputs non-linearly

3. Decision Boundary

- Where prediction changes from one class to another
- Visualizes how model 'sees' the data

■ Activation Functions Breakdown

Activation	Formula	Range	Boundary Shape
ReLU	max(0, x)	[0, ∞)	Angular
Logistic	1/(1+e^-x)	(0, 1)	Smooth
Tanh	(e^x-e^-x)/(e^x+e^-x)	(-1, 1)	Smooth

Analogies:

- ReLU = One-way valve (positive flows, negative blocked)
- Sigmoid = Dimmer switch (smoothly scales 0 to 1)
- Tanh = Centered dimmer (-1 to 1)

■ Solution Flow

Step 1: Generate make_moons dataset (300 samples)

↓

Step 2: Create 3 MLPClassifier models (ReLU, Logistic, Tanh)

↓

Step 3: Train all models on the same data

↓

Step 4: Create meshgrid and predict on all points

↓

Step 5: Visualize decision boundaries with contourf

↓

Step 6: Compare accuracies and analyze results

■ Code Logic Summary

```
# 1. Data Generation
X, y = make_moons(n_samples=300, noise=0.2, random_state=42)

# 2. Model Creation
model = MLPClassifier(hidden_layer_sizes=(8,), activation='relu', random_state=42)

# 3. Training
model.fit(X, y)

# 4. Visualization
Z = model.predict(meshgrid_points)
plt.contourf(xx, yy, Z, alpha=0.8)
```

■■ Important Parameters

Parameter	Value	Effect
hidden_layer_sizes	(8,)	1 layer, 8 neurons
activation	varies	Boundary shape
solver	adam	Optimization
max_iter	1000	Training cycles
random_state	42	Fair comparison

■ Results

Activation	Accuracy	Rank
ReLU	88.33%	■ 1st
Tanh	86.33%	■ 2nd
Logistic	85.67%	■ 3rd

Boundary Shapes:

- ReLU creates angular, piecewise-linear edges
- Sigmoid/Tanh create smooth, curved boundaries

■ Observations & Insights

1. **ReLU creates ANGULAR boundaries** - Due to piecewise-linear nature

2. **Sigmoid/Tanh create SMOOTH boundaries** - Due to continuous curves

3. **All accuracies similar (~85-88%)** - Dataset is 'easy' for 8 neurons

4. **ReLU wins by small margin** - Advantages more visible in deep networks

■■ Advantages & Limitations

ReLU

- ■ No vanishing gradient, fast computation, modern default
- ■ Dead neurons possible, not zero-centered

Sigmoid/Tanh

- ■ Bounded output, probability interpretation, smooth gradients
- ■ Vanishing gradient, slower computation

■ Interview Key Takeaways

1. ReLU = max(0, x) → Default for hidden layers
2. Sigmoid = 1/(1+e^-x) → Binary output layers
3. Tanh → RNNs, zero-centered needed
4. Vanishing gradient → Sigmoid/Tanh problem, not ReLU
5. hidden_layer_sizes=(8,) → Tuple notation with comma!
6. random_state → For reproducibility

■ Conclusion

What We Learned:

- Different activations create different boundary shapes
- ReLU: Angular, Sigmoid/Tanh: Smooth
- For simple data, differences are small

Scenario	Use
Hidden layers	ReLU
Binary output	Sigmoid
RNNs	Tanh
Deep networks	ReLU (avoid vanishing gradient)

"Activation choice matters more in deep networks than shallow ones."