😊 Activation Functions

In deep learning, **activation functions** are mathematical formulas used in neural networks to determine whether a neuron should be "activated" or not. Simply put, they help the neural network decide which information should move forward through the layers of the network and which should not.

Why are Activation Functions Important?

Without activation functions, the neural network would only perform linear transformations on the data. This means it wouldn't be able to solve complex problems like recognizing images or understanding speech. Activation functions introduce non-linearity, allowing the network to learn and solve complex tasks.

- **Non-linearity**: Neural networks need to model complex relationships between inputs and outputs. Activation functions introduce non-linearities into the model, allowing it to learn and approximate any function.
- **Differentiability**: Activation functions must be differentiable so that gradients can be computed during backpropagation, which is essential for learning in neural networks.

Example:

Imagine you're a teacher grading homework. For each question, you either give full credit, partial credit, or no credit based on how correct the student's answer is. The decision-making process—whether to give full, partial, or no credit—is similar to how activation functions work in a neural network. They decide whether to pass the "signal" forward or modify it in some way.

Common Types of Activation Functions

1. **ReLU (Rectified Linear Unit)**
    - **How it works**: If the input is positive, it keeps it the same; if the input is negative, it changes it to zero.
    - **Example**: Imagine a light switch that turns on when the electricity is flowing (positive values) but stays off when there's no power (negative values).
    - **Use case**: ReLU is widely used in hidden layers of neural networks because it helps models learn faster and handle large datasets well.
2. **Sigmoid**
    - **How it works**: It takes any number (positive or negative) and squeezes it into a value between 0 and 1.
    - **Example**: Think of a volume knob that smoothly adjusts the sound from mute (0) to full volume (1), but never goes below 0 or above 1.
    - **Use case**: Sigmoid is often used in the output layer for binary classification (where the result is either 0 or 1), like detecting if an image contains a cat (1) or not (0).

3. **Tanh (Hyperbolic Tangent)**
    - **How it works**: It's similar to the sigmoid function, but instead of outputting values between 0 and 1, it outputs values between -1 and 1.
    - **Example**: Imagine a thermostat that measures temperature. It can indicate very cold (-1), neutral (0), or very hot (1).
    - **Use case**: Tanh is used when you want the outputs to be between negative and positive, like in some types of signal processing or sentiment analysis (positive or negative emotions).
4. **Softmax**
    - **How it works**: Softmax converts a set of raw numbers into probabilities that sum to 1. Each number is transformed into a value between 0 and 1, representing the probability of each class.
    - **Example**: Imagine a pie chart showing the probability of different types of fruit in a basket: 60% apples, 30% oranges, 10% bananas. Softmax would assign each fruit a probability based on the data.
    - **Use case**: Softmax is used in the output layer for multi-class classification, like determining if an image contains a cat, dog, or rabbit.

Summary of Common Activation Functions

Activation Function	Output Range	Common Use Cases
ReLU	0 to ∞	Hidden layers in deep networks
Sigmoid	0 to 1	Binary classification tasks
Tanh	-1 to 1	Sentiment analysis, signal processing
Softmax	0 to 1 (sum to 1)	Multi-class classification

Why Different Activation Functions?

Each activation function behaves differently, and the choice of which to use depends on the problem you're trying to solve:

- **ReLU** is popular for deep networks because it's simple and helps prevent vanishing gradients (where the gradients become too small for the network to learn properly).
- **Sigmoid** and **Tanh** are useful when outputs need to be squeezed into specific ranges, like probabilities or sentiment scores.
- **Softmax** is perfect for problems where you're selecting between multiple classes, like identifying which object is in an image.

In essence, activation functions act as decision-makers in a neural network, helping the model learn complex patterns by introducing non-linearity and controlling how information flows through the network.

Comparison of Activation Functions

Activation Function	Output Range	Use Cases	Advantages	Disadvantages
Sigmoid	(0, 1)	Binary classification tasks	Output interpretable as probability	Vanishing gradient problem
Tanh	(-1, 1)	Hidden layers (zero-centered data)	Stronger gradients than Sigmoid	Vanishing gradient problem
ReLU	[0, ∞)	Hidden layers of deep networks	Simple, computationally efficient	Dying ReLU problem
Leaky ReLU	(-∞, ∞)	Hidden layers to avoid dying ReLU	Solves dying ReLU issue	Still not widely adopted everywhere
Softmax	(0, 1)	Multi-class classification (output)	Produces probabilities (sum = 1)	Not suitable for hidden layers

When to Use Each Activation Function

1. **Sigmoid**: When you need outputs to represent probabilities (usually in binary classification).
2. **Tanh**: When you have zero-centered data and need outputs between -1 and 1, especially in hidden layers.
3. **ReLU**: When you are building deep networks, especially in hidden layers, because of its simplicity and efficiency.
4. **Leaky ReLU**: When you encounter the dying ReLU problem (when many neurons output zero and stop learning).
5. **Softmax**: When you are performing multi-class classification (e.g., determining which category an image belongs to).

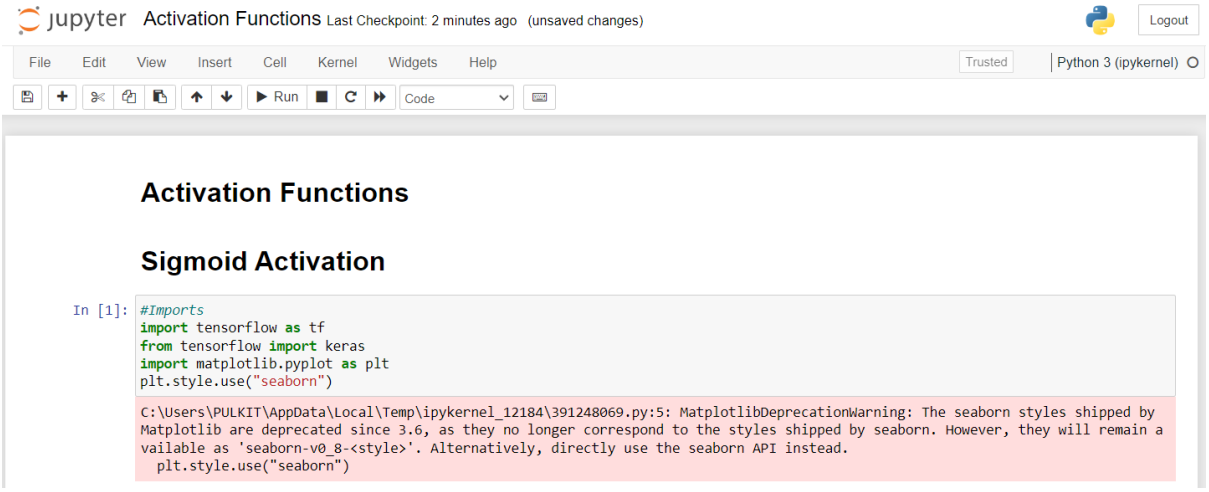
Conclusion

Activation functions play a critical role in enabling neural networks to learn complex patterns from data. They introduce non-linearity into the network, allowing it to solve a wide range of problems. The choice of activation function depends on the specific task and layer of the neural network, and understanding their properties is essential for designing effective deep learning models.

😄 To begin the Lab:

1. In your Jupyter Lab, create a new Python Kernel. In this notebook we are working with the various activation functions.

2. This code begins by importing the TensorFlow library, which is used for building and training machine learning models. It also imports Keras, a high-level API within TensorFlow that simplifies the process of creating neural networks. Finally, it imports Matplotlib, a library for creating visualizations, and applies a specific style to the plots called "seaborn," which enhances their appearance.



3. This code generates a set of 100 evenly spaced data points ranging from -10 to 10. The points are stored in a variable called X, and this range can be used for plotting or training a machine learning model.

```
In [2]: #Generate Data Points
        X = tf.linspace(-10,10,100)
        X
```

```
Out[2]: <tf.Tensor: shape=(100,), dtype=float64, numpy=
        array([-10.        ,  -9.7979798 ,  -9.5959596 ,  -9.39393939,
               -9.19191919,  -8.98989899,  -8.78787879,  -8.58585859,
               -8.38383838,  -8.18181818,  -7.97979798,  -7.77777778,
               -7.57575758,  -7.37373737,  -7.17171717,  -6.96969697,
               -6.76767677,  -6.56565657,  -6.36363636,  -6.16161616,
               -5.95959596,  -5.75757576,  -5.55555556,  -5.35353535,
               -5.15151515,  -4.94949495,  -4.74747475,  -4.54545455,
               -4.34343434,  -4.14141414,  -3.93939394,  -3.73737374,
               -3.53535354,  -3.33333333,  -3.13131313,  -2.92929293,
               -2.72727273,  -2.52525253,  -2.32323232,  -2.12121212,
               -1.91919192,  -1.71717172,  -1.51515152,  -1.31313131,
               -1.11111111,  -0.90909091,  -0.70707071,  -0.50505051,
               -0.3030303 ,  -0.1010101 ,   0.1010101 ,   0.3030303 ,
                0.50505051,   0.70707071,   0.90909091,   1.11111111,
                1.31313131,   1.51515152,   1.71717172,   1.91919192,
                2.12121212,   2.32323232,   2.52525253,   2.72727273,
                2.92929293,   3.13131313,   3.33333333,   3.53535354,
                3.73737374,   3.93939394,   4.14141414,   4.34343434,
                4.54545455,   4.74747475,   4.94949495,   5.15151515,
                5.35353535,   5.55555556,   5.75757576,   5.95959596,
                6.16161616,   6.36363636,   6.56565657,   6.76767677,
                6.96969697,   7.17171717,   7.37373737,   7.57575758,
                7.77777778,   7.97979798,   8.18181818,   8.38383838,
                8.58585859,   8.78787879,   8.98989899,   9.19191919,
                9.39393939,   9.5959596 ,   9.7979798 ,  10.        ])>
```

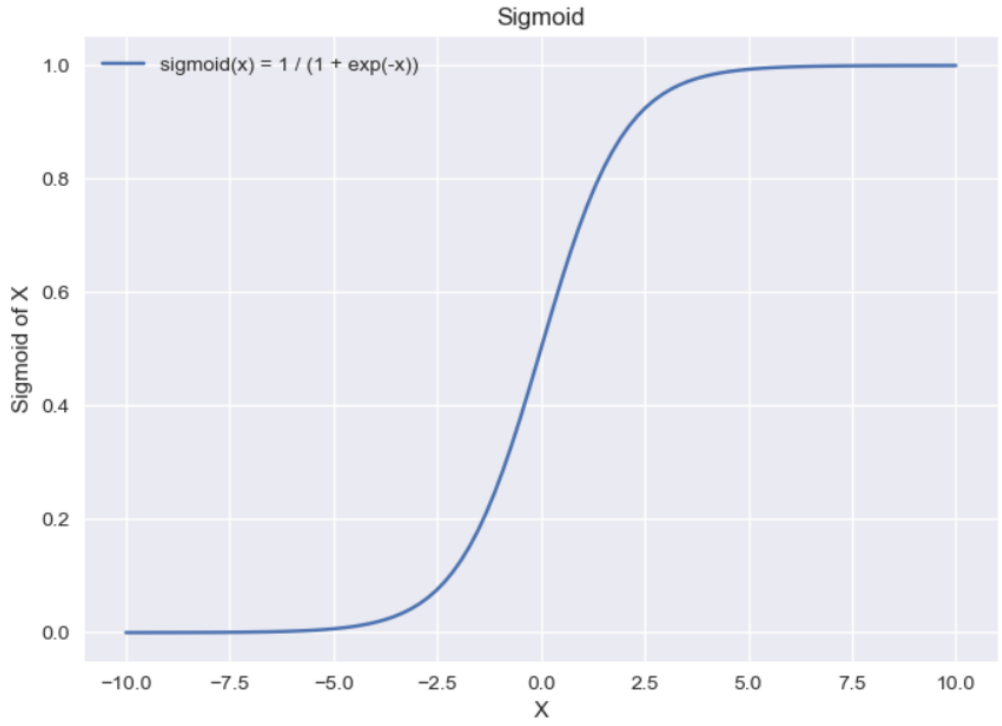
4. First, the code checks the shape of the array X to see how many data points it contains. Then, it applies the sigmoid activation function to each point in X, calculating the output y. Finally, it visualizes the relationship between X and y by plotting the sigmoid curve. The graph is titled "Sigmoid," with labelled axes and a legend explaining the function. The plot is then displayed on the screen.

```
In [3]: X.shape
```

```
Out[3]: TensorShape([100])
```

In [4]:
```
#Activations
y = tf.keras.activations.sigmoid(X)

#Visualize Graph
plt.plot(X,y,label = "sigmoid(x) = 1 / (1 + exp(-x))" )
plt.title("Sigmoid")
plt.xlabel("X")
plt.ylabel("Sigmoid of X")
plt.legend()
plt.show()
```

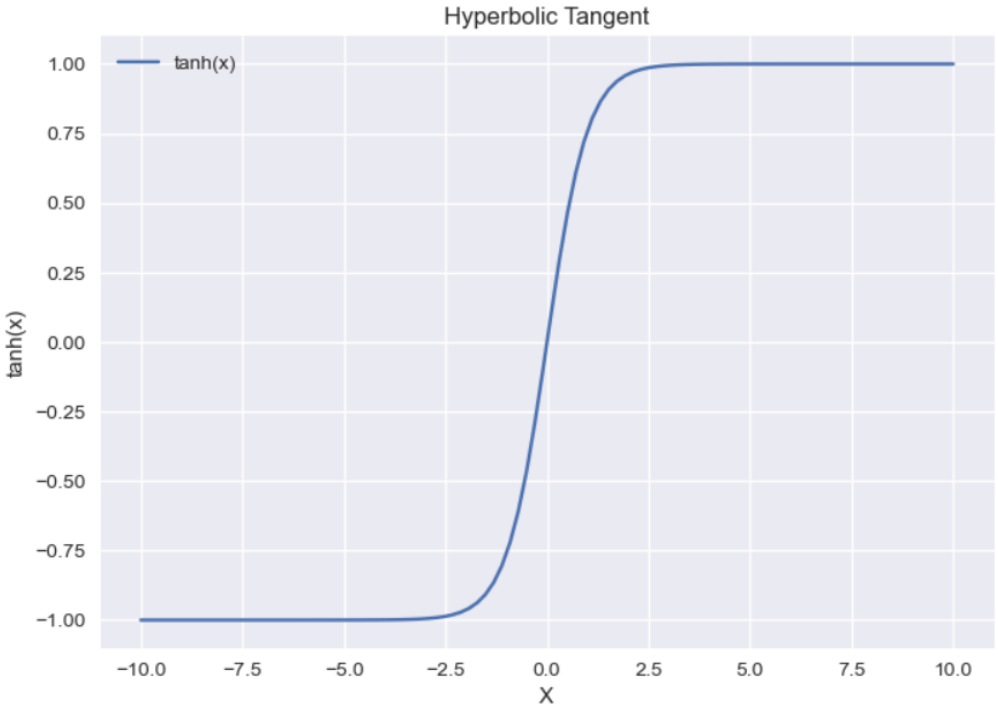


5. In this code, the hyperbolic tangent (tanh) activation function is applied to each data point in X, resulting in a new output y. Then, it visualizes the tanh curve by plotting X against y. The graph is titled "Hyperbolic Tangent," with labelled axes and a legend indicating the function. Finally, the plot is displayed on the screen.

Tanh - Hyperbolic Tangent Function

In [5]:
```
#Activations
y = tf.keras.activations.tanh(X)

#Visualize Graph
plt.plot(X,y,label = "tanh(x)" )
plt.title("Hyperbolic Tangent")
plt.xlabel("X")
plt.ylabel("tanh(x)")
plt.legend()
plt.show()
```



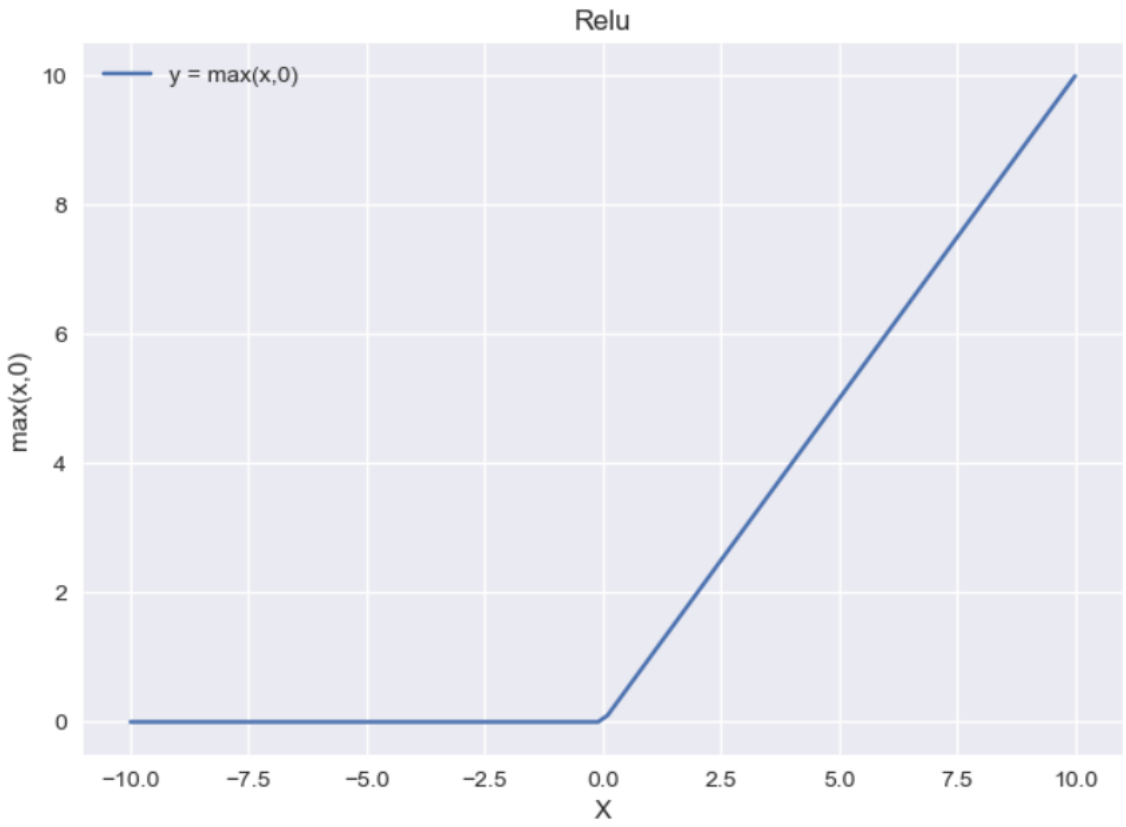
6. In this code, the Rectified Linear Unit (ReLU) activation function is applied to each point in X, resulting in an output y where all negative values are set to zero and positive values remain unchanged. The code then visualizes this ReLU behaviour by plotting X against y. The graph is titled "Relu," with labelled axes and a legend that explains the function as y = max (x,0). Finally, the plot is displayed on the screen.

Relu Activation

Relu Activation : y = max(x,0)

```
In [6]: #Activations
        y = tf.keras.activations.relu(X)

        #Visualize Graph
        plt.plot(X,y,label = "y = max(x,0)" )
        plt.title("Relu")
        plt.xlabel("X")
        plt.ylabel("max(x,0)")
        plt.legend()
        plt.show()
```



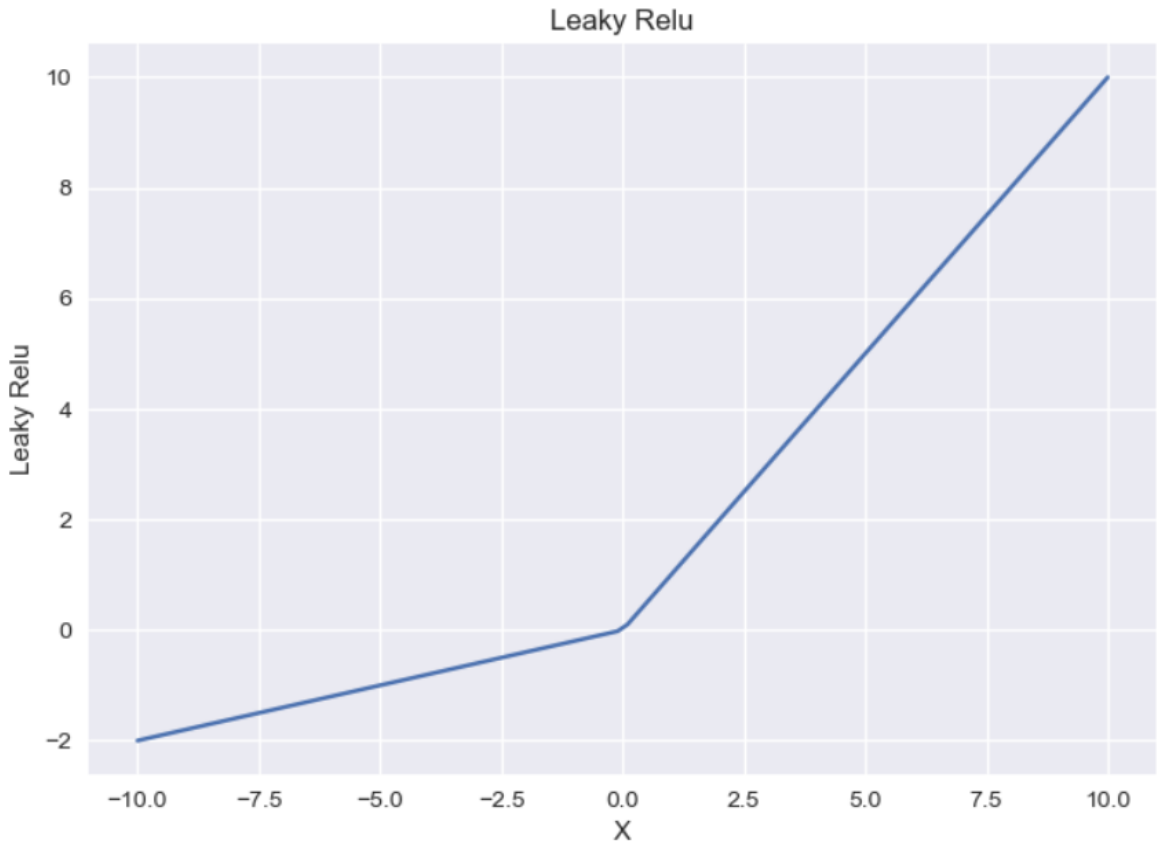
7. In this code, the Leaky Rectified Linear Unit (Leaky ReLU) activation function is applied to each point in X. This function allows a small, non-zero gradient for negative values, instead of setting them to zero. The resulting output y is then visualized by plotting XXX against y. The graph is titled "Leaky Relu," with labelled axes. Finally, the plot is displayed on the screen.

Leaky Relu

In [7]:
```
#Activations
y = tf.nn.leaky_relu(X)

#Visualize Graph
plt.plot(X,y )
plt.title("Leaky Relu")
plt.xlabel("X")
plt.ylabel("Leaky Relu")
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
() is called with no argument.
```



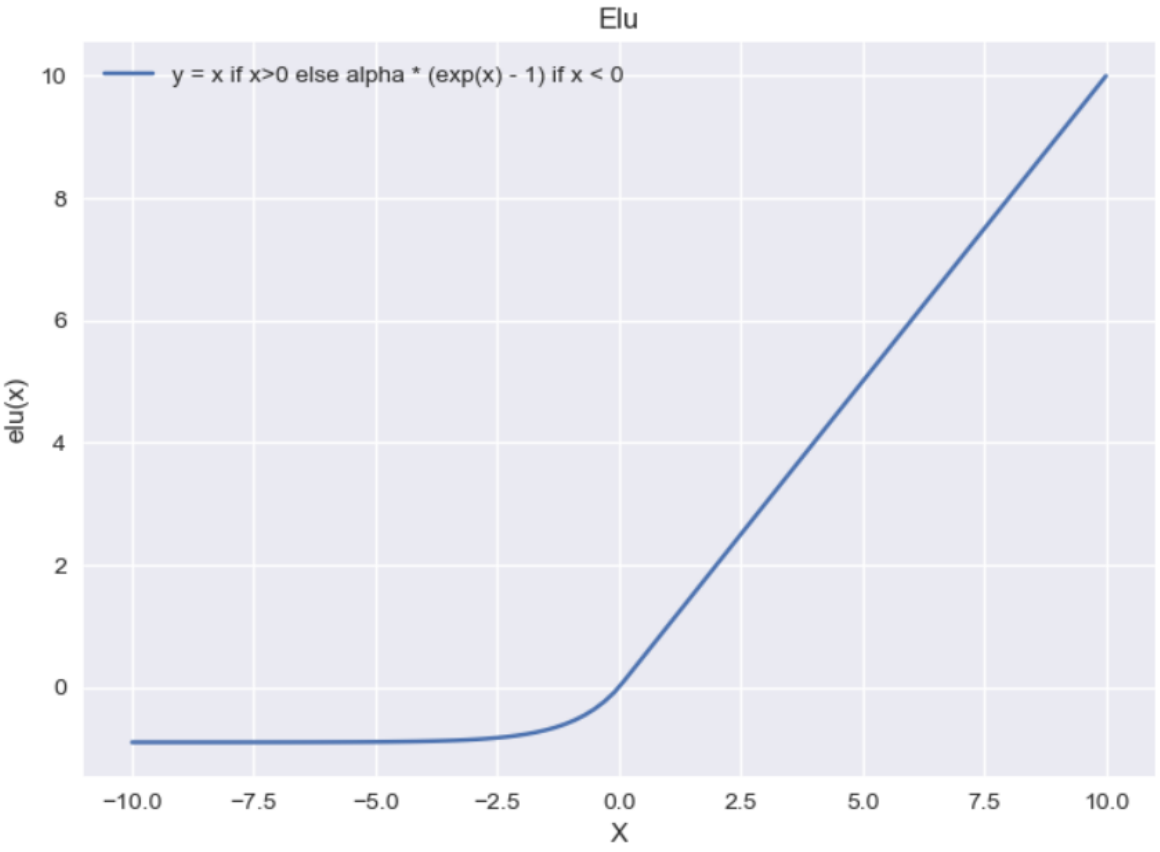
8. In this code, the Exponential Linear Unit (ELU) activation function is applied to each point in X, using an alpha value of 0.9. The ELU function outputs xxx for positive values and a modified exponential function for negative values. The resulting output y is then visualized by plotting X against y. The graph is titled "Elu," with labelled axes and a legend explaining the function. Finally, the plot is displayed on the screen.

Elu Activation Function - Exponential Linear Unit

y = x if x > 0 and alpha * (exp(x) - 1) if x < 0.

In [8]:
```
#Activations
y = tf.keras.activations.elu(X,alpha = 0.9)

#Visualize Graph
plt.plot(X,y,label = "y = x if x>0 else alpha * (exp(x) - 1) if x < 0 " )
plt.title("Elu")
plt.xlabel("X")
plt.ylabel("elu(x)")
plt.legend()
plt.show()
```



9. In this code, the Scaled Exponential Linear Unit (SELU) activation function is applied to each point in X. The SELU function adjusts the output for both positive and negative values, scaling them in a specific way to help with training neural networks. The resulting output y is then visualized by plotting X against y. The graph is titled "SELU," with labelled axes and a legend indicating the function. Finally, the plot is displayed on the screen.

SELU Activation Function - Scaled Exponential Linear Unit

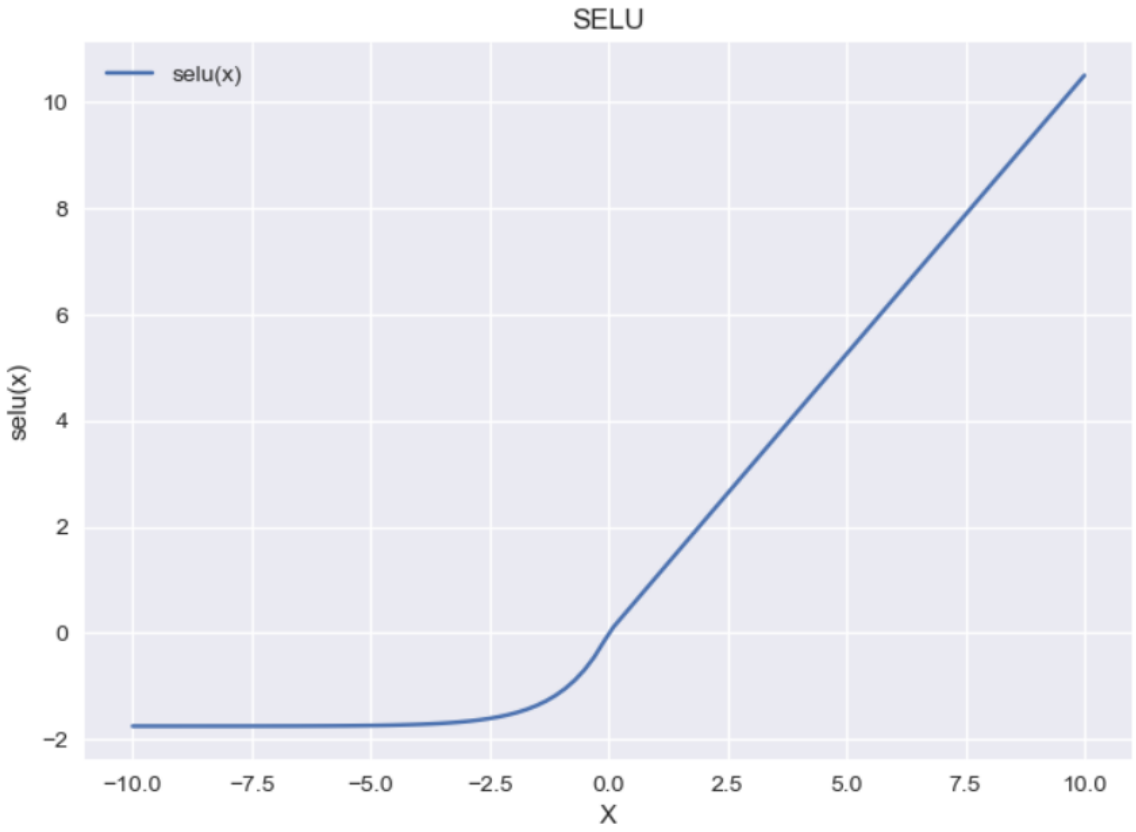
if x > 0: return scale * x

if x < 0: return scale * alpha * (exp(x) - 1)

where alpha and scale are pre-defined constants (alpha=1.67326324 and scale=1.05070098)

In [9]:
```
#Activations
y = tf.keras.activations.selu(X)

#Visualize Graph
plt.plot(X,y,label = "selu(x)" )
plt.title("SELU")
plt.xlabel("X")
plt.ylabel("selu(x)")
plt.legend()
plt.show()
```



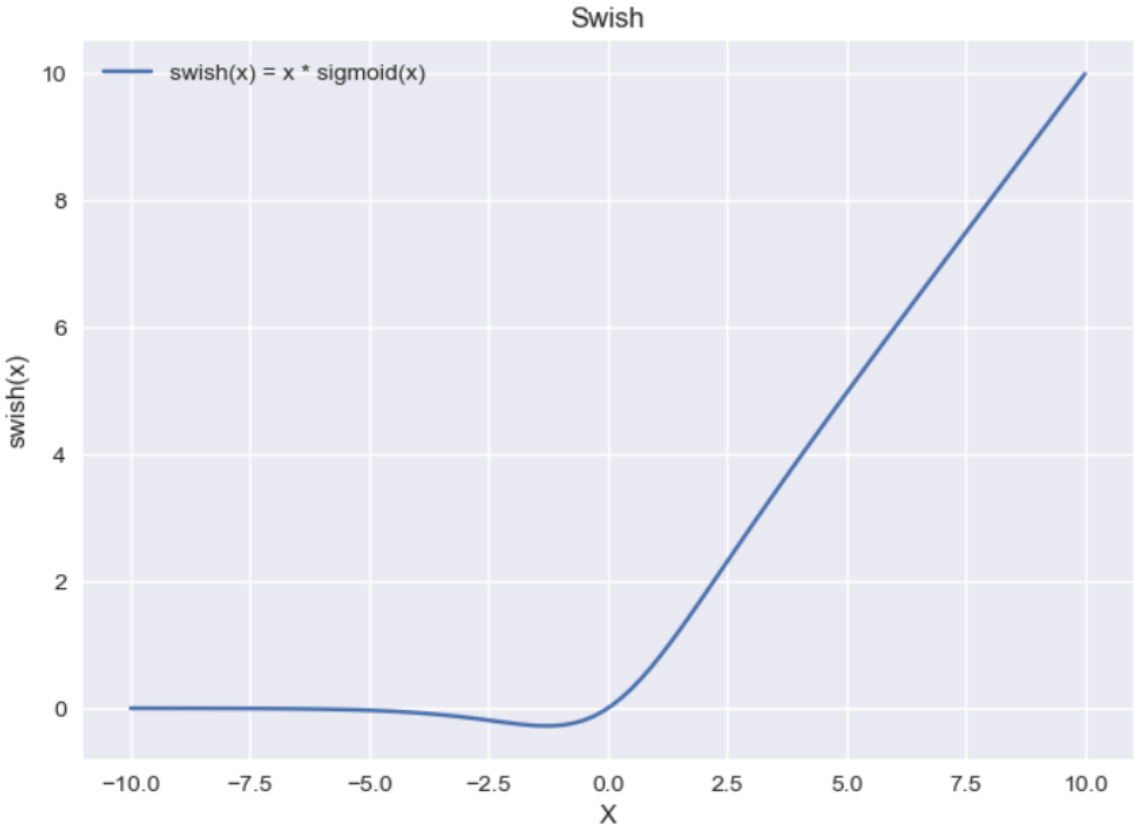
10. In this code, the Swish activation function is applied to each point in X. The Swish function combines the input xxx with the sigmoid of xxx, resulting in a smooth and non-monotonic output. The resulting output y is then visualized by plotting X against y. The graph is titled "Swish," with labelled axes and a legend explaining the function as swish **swish(x) = x × sigmoid(x)**. Finally, the plot is displayed on the screen.

Swish Activation

swish(x) = x * sigmoid(x)

In [10]:
```
#Activations
y = tf.keras.activations.swish(X)

#Visualize Graph
plt.plot(X,y,label = "swish(x) = x * sigmoid(x)" )
plt.title("Swish")
plt.xlabel("X")
plt.ylabel("swish(x)")
plt.legend()
plt.show()
```



11. In this code, a TensorFlow constant xxx is created with three values: 150, 50, and 10, and it's defined as a float type. The **tf.expand_dims** function adds an extra dimension to xxx, making it a 2D tensor (specifically, a single row with three columns). The shape of this tensor and its contents are printed.

12. Next, the SoftMax activation function is applied to x, which transforms the values into probabilities that sum to 1. The resulting output y, representing these probabilities, is then printed.

Softmax

converts a vector of values to a probability distribution

Formula

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector

In [11]:
```
x = tf.constant([150,50,10],dtype=tf.float32)
x = tf.expand_dims(x,axis=0)
print(x.shape,x)
y = tf.keras.activations.softmax(x)
print(y)
```

```
(1, 3) tf.Tensor([[150.  50.  10.]], shape=(1, 3), dtype=float32)
tf.Tensor([[1. 0. 0.]], shape=(1, 3), dtype=float32)
```