



Frankfurt School

Activation Functions

Group 3

Neelesh Bhalla

Nils Marthiensen

Chia-Jung Chang

Kirtesh Pushpakbhai Patel

Rhushikesh Bhosle

Content

- Objective
- Introduction
- Popular Activation Functions
- Deep Learning Problem Statements
- Results
- Further Steps
- References and GitHub

Objective

Activating your understanding:

Exploring the theory, benefits and drawbacks of various activation functions

INTRODUCTION



Introduction: Neural Nets

Refresher:

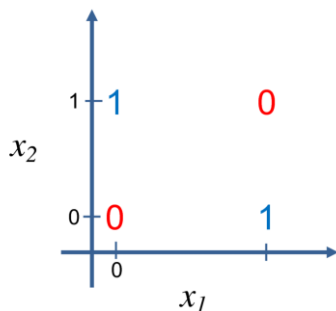
- Neural networks consist of an **input**, an **output** and at least one **hidden layer**
- Each layer consists of several **neurons**
- The output of each neuron is the input **X** times the weights **W** and a bias term **b**:

$$h_{W,b}(X) = \sigma(XW + b)$$

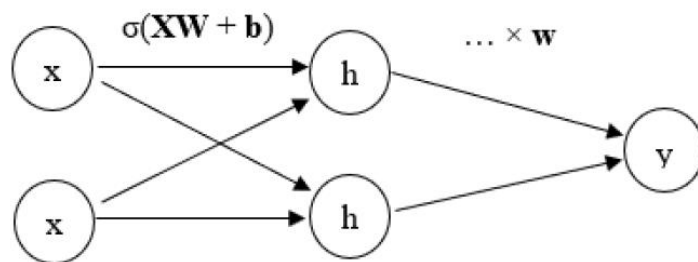
Géron, 2019

XOR function:

- Two binary inputs
- Returns 1 if different
- Returns 0 if similar



Goodfellow, Bengio & Courville, 2016



$$\begin{matrix} \sigma & \mathbf{X} & \mathbf{W} & \mathbf{b} & \mathbf{w} & \mathbf{y} \end{matrix}$$
$$ReLU \left(\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right) \times \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Introduction: Activation Functions

Why are they needed?

- Activation functions **add non-linearity**
- They **regulate** neuron output to prevent numerical **instability** and **overfitting**
- **Different activation functions** are suited to **different types of problems**

How did it start?

- The first activation function was the binary step function:

$$\sigma(u) = \begin{cases} 0 & \text{if } u < 0 \\ 1 & \text{if } u \geq 0 \end{cases}$$

Limitations:

- Not differentiable
- Binary output
- Not robust to noise
- Not suited for multiclass classification or regression tasks

Where are we now?

- Have evolved to address the limitations of earlier functions and to **enhance the capabilities of ANNs**
- Early development was focused on characteristics like **differentiability and continuous output**
- Later problems like **vanishing gradients** or **dead neurons** were tackled
- The introduction of new functions has continued to **improve the accuracy and efficiency** of neural networks

POPULAR ACTIVATION FUNCTIONS



ReLU_N ARiA ShiLU Softsign Hard-ELiSH Mish
GELU SELU Shifted-Softplus Tanhexp
ReGLU Softplus StarReLU NLSIG
SerLU GegLU Siren SReLU Maxout M-ArcSinh
PeLU SiLU GCU ELU PAU ReLU SwigLU CoLU
Tanh Swish Smooth-Step CReLU
ScaledSoftSign Gumbel Sigmoid
NFN E-Swish Hard-Sigmoid modReLU
KAF Rational-Activation-Function
AHAF CosLU Hard-Swish Serf PReLU
Phish DeLU Leaky-ReLU RReLU EvoNorms ASAF

Popular AFs in use (1/2)

The **sigmoid** function

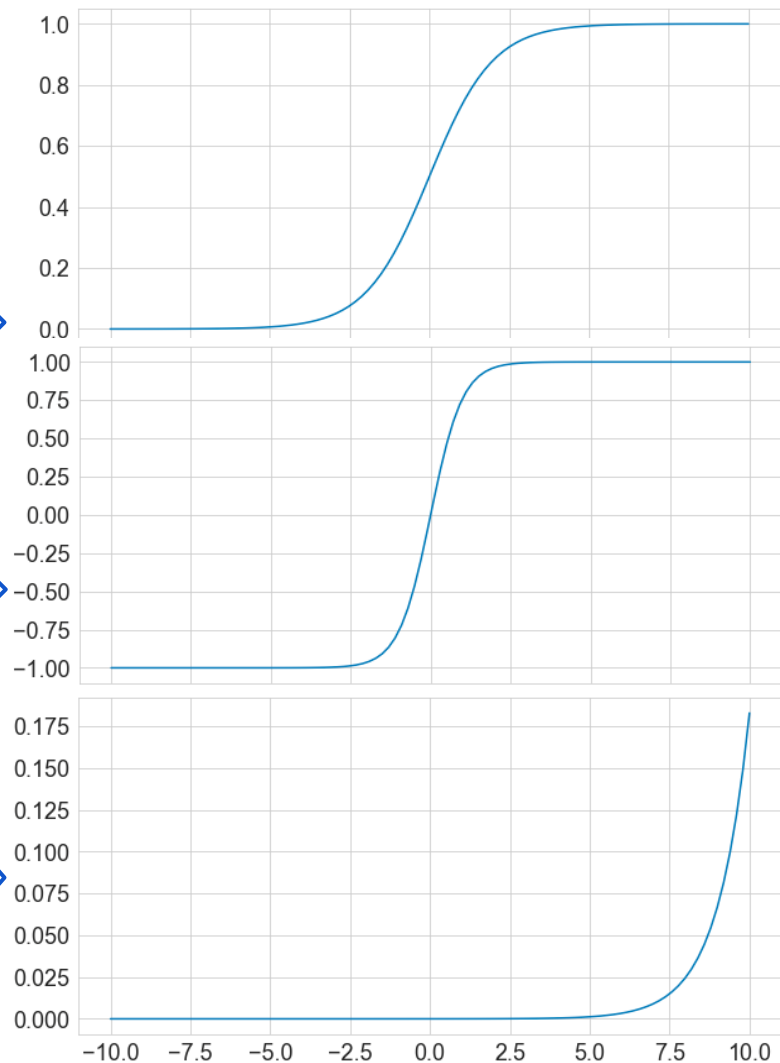
A function that maps any input value to a range between 0 and 1, and is commonly used in binary classification problems.

The **hyperbolic tangent (tanh)** function

Similar to the sigmoid function, but outputs values between -1 and 1, making it better suited for symmetric input data.

The **softmax** function

A function that maps a vector of values to a probability distribution, with each output value representing the probability of the corresponding input value.



Popular AFs in use (1/2)

The **sigmoid** function

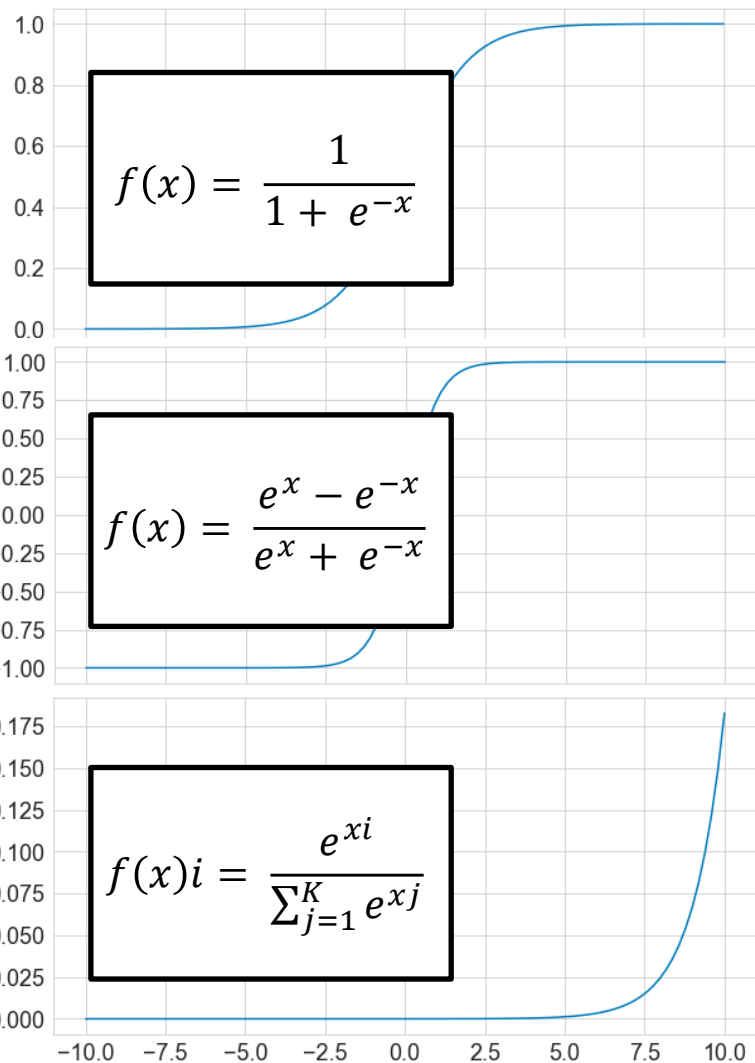
A function that maps any input value to a range between 0 and 1, and is commonly used in binary classification problems.

The **hyperbolic tangent (tanh)** function

Similar to the sigmoid function, but outputs values between -1 and 1, making it better suited for symmetric input data.

The **softmax** function

A function that maps a vector of values to a probability distribution, with each output value representing the probability of the corresponding input value.



Popular AFs in use (2/2)

The rectified linear unit (ReLU) function

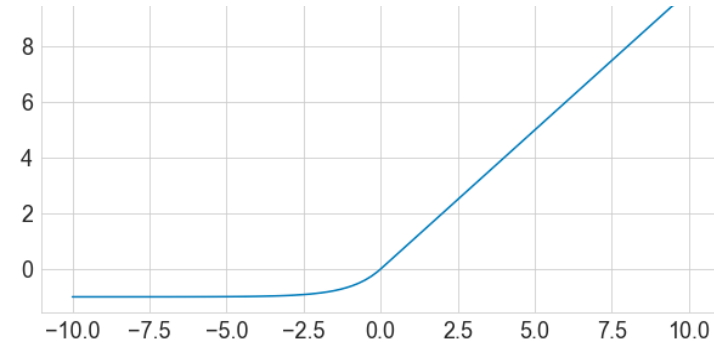
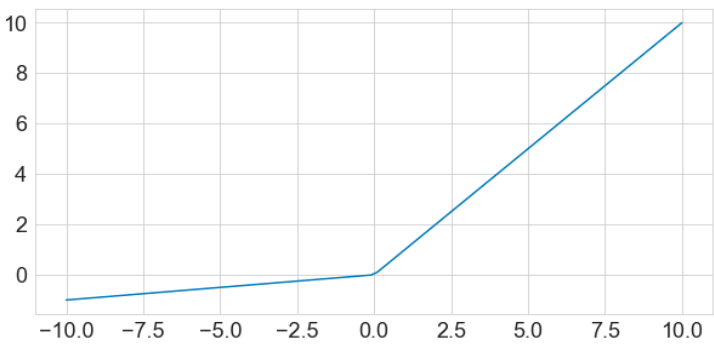
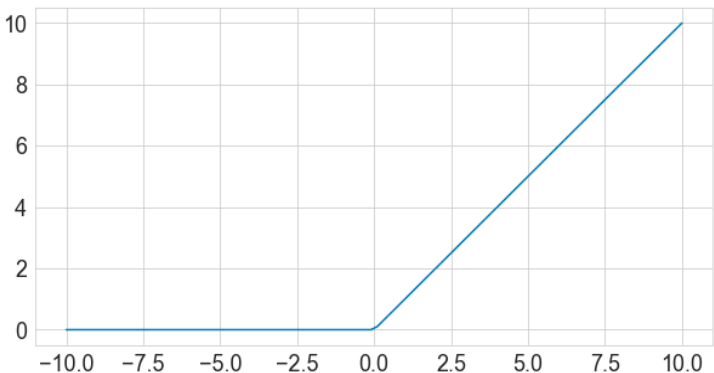
A function that outputs the input value if it is positive, and 0 if it is negative, which is fast to compute and can speed up training of deep neural networks.

The leaky ReLU function

Similar to ReLU, but allows a small positive gradient when the input is negative, addressing the "dying ReLU" problem that can occur when using ReLU.

The exponential linear unit (ELU) function

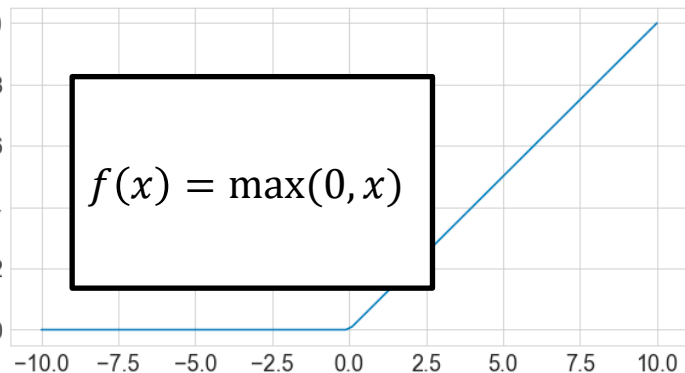
Similar to ReLU but with a smooth transition to negative inputs, leading to improved performance on some tasks.



Popular AFs in use (2/2)

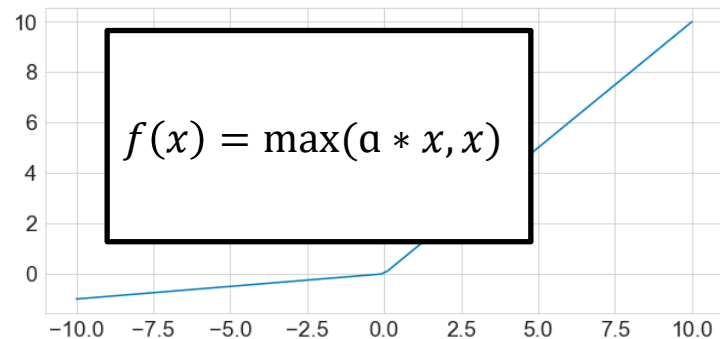
The rectified linear unit (ReLU) function

A function that outputs the input value if it is positive, and 0 if it is negative, which is fast to compute and can speed up training of deep neural networks.



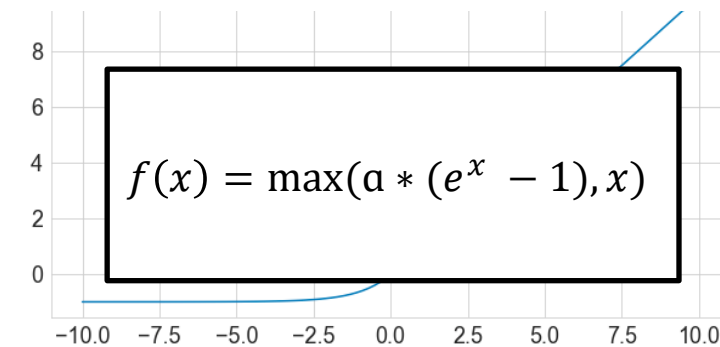
The leaky ReLU function

Similar to ReLU, but allows a small positive gradient when the input is negative, addressing the "dying ReLU" problem that can occur when using ReLU.



The exponential linear unit (ELU) function

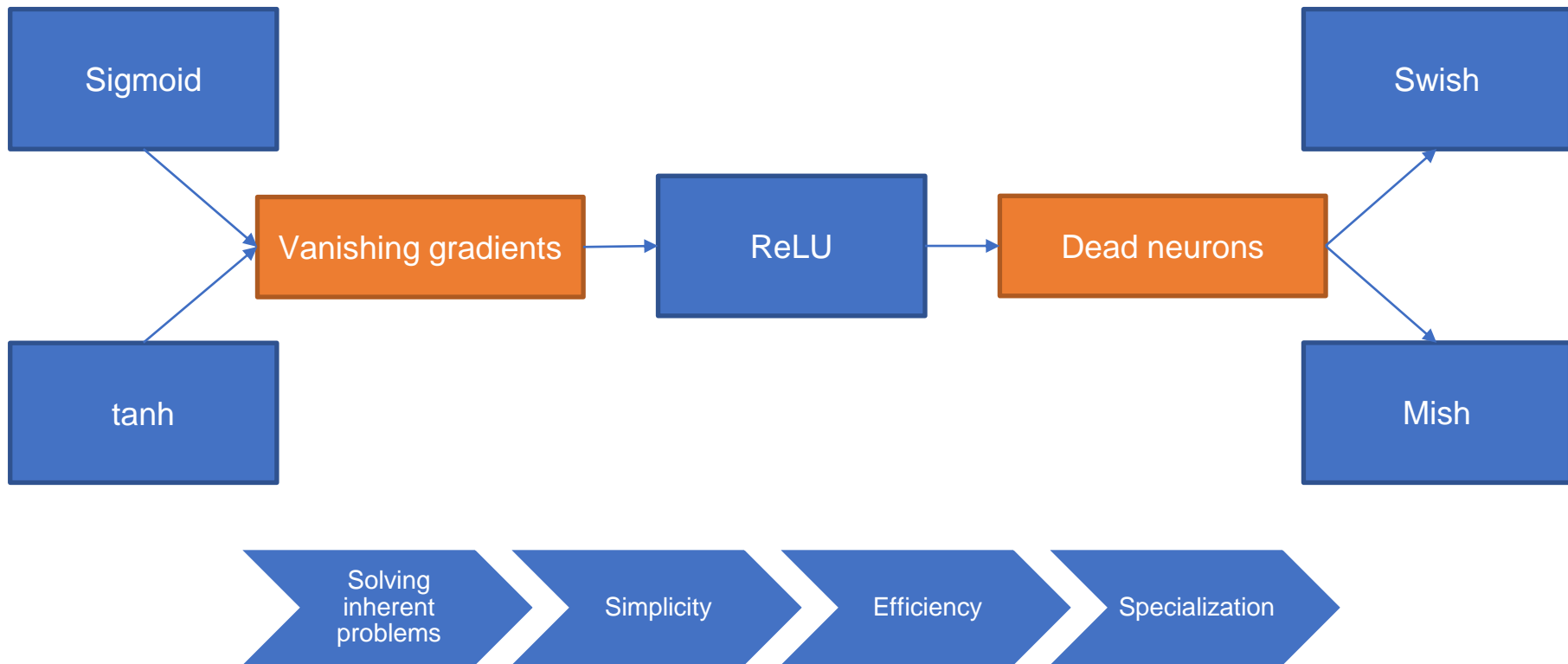
Similar to ReLU but with a smooth transition to negative inputs, leading to improved performance on some tasks.



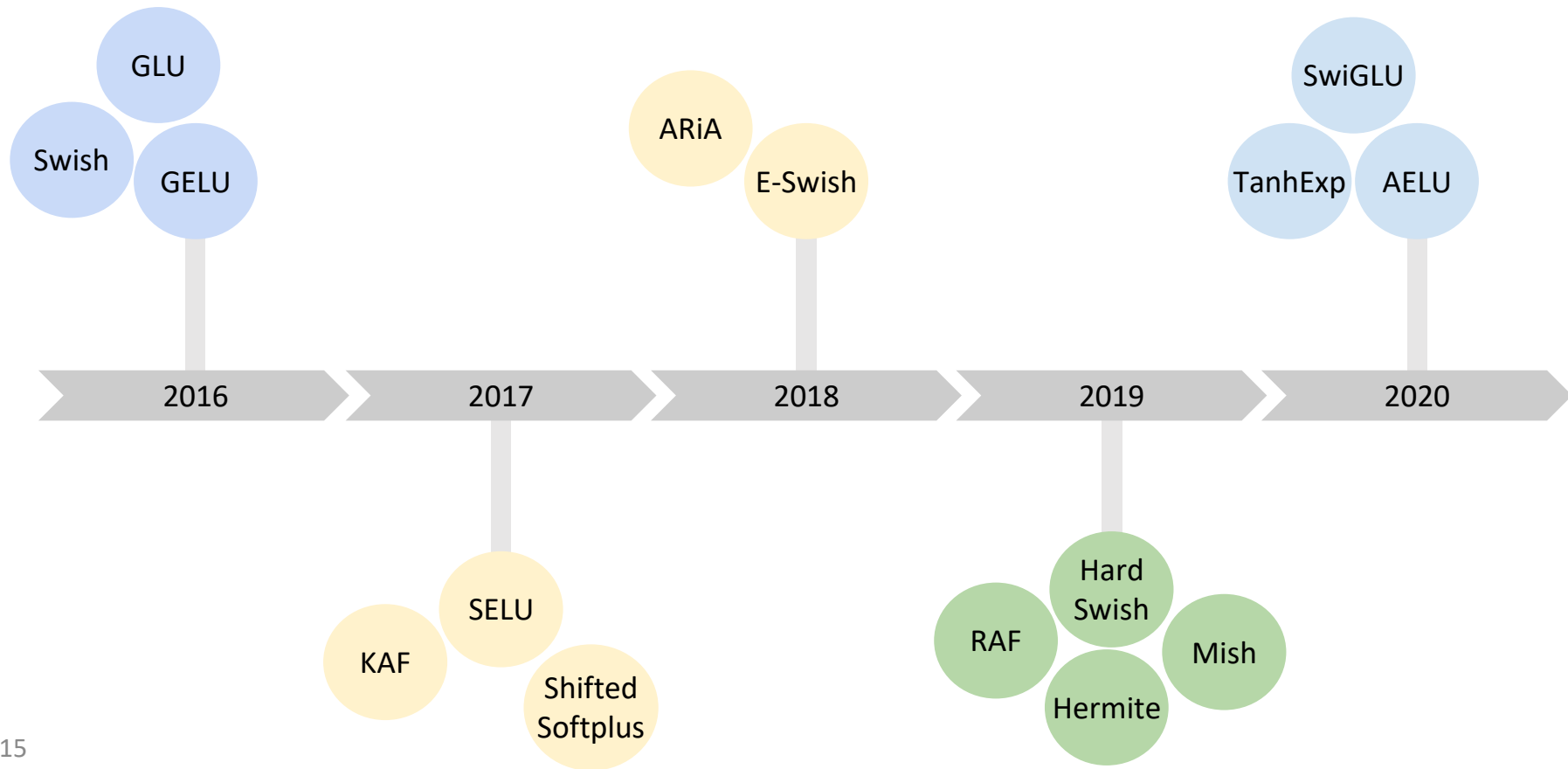
Objective Comparison: Popular AFs

	Pros	Cons	Use case
Sigmoid	Easy to calculate gradient, good choice for probability or decision boundary	Sensitive to outlier, may suffer from vanishing gradients	Binary classification, output layer of neural networks
ReLU	Fast computation, good for deep neural networks	Can suffer from dying ReLU problem	Deep neural networks, computational efficiency, sparse and high-dimension data
Tanh	Centered around 0, good for symmetric data	Sensitive to outlier, may suffer from vanishing gradients	Symmetric data, image classification
Softmax	Produces probabilities for each class, useful for decision-making	Sensitive to outlier, may suffer from vanishing gradients	Multi-class classification, output layer of neural networks
Leaky ReLU	Allows some negative input compared to ReLU	May introduce noise and still suffer from dying ReLU problem	Similar to ReLU but alleviate some dying ReLU problem
ELU	Reduces the likelihood of overfitting	Computationally expensive for negative inputs	Deeper networks, smoother transition to negative values

Why continue development?



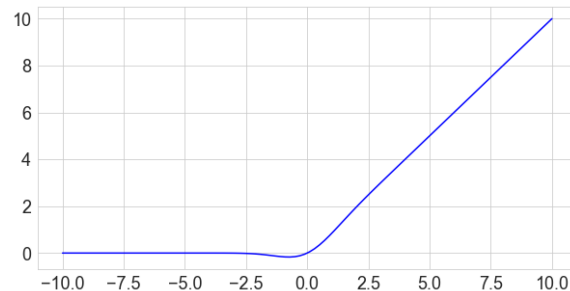
Modern Developments & Timeline



State-of-the-art AFs (1/3)

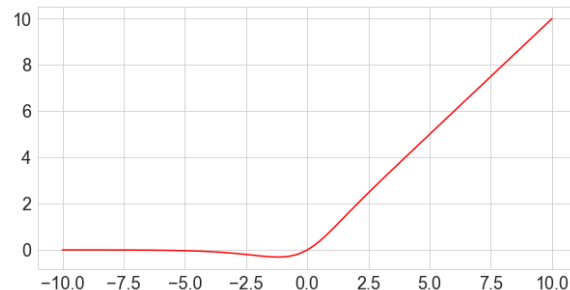
GELU (Gaussian Error Linear Function)

Ability to approximate the identity function in its linear region, which can help preserve information during training and improve the overall performance of the network. Used in Computer vision and NLP.



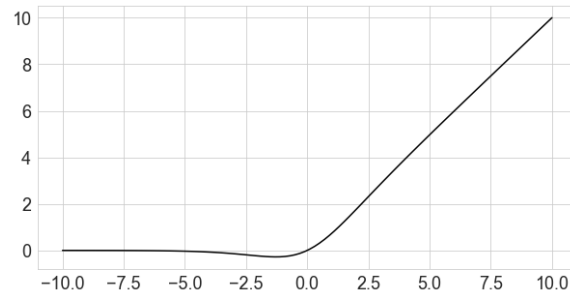
The Mish Activation function

Key features of Mish is its non-monotonicity, which allows it to capture complex patterns in data and improve the performance of the network. Used in deep learning applications that require high performance.



The Swish Activation function

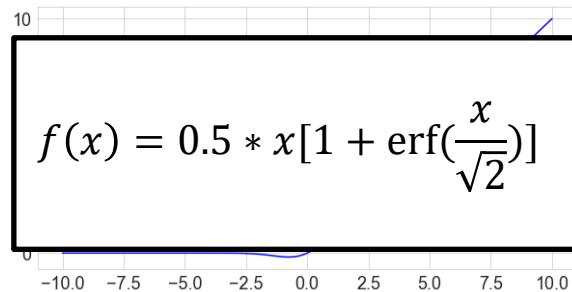
It has the best computational efficiency and easy implementation, which makes it suitable for deep learning applications with large datasets and complex architectures. A promising alternative to existing activation functions in applications where data security is critical.



State-of-the-art AFs (1/3)

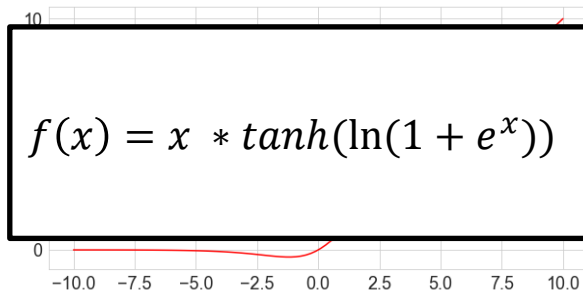
GELU (Gaussian Error Linear Function)

Ability to approximate the identity function in its linear region, which can help preserve information during training and improve the overall performance of the network. Used in Computer vision and NLP.



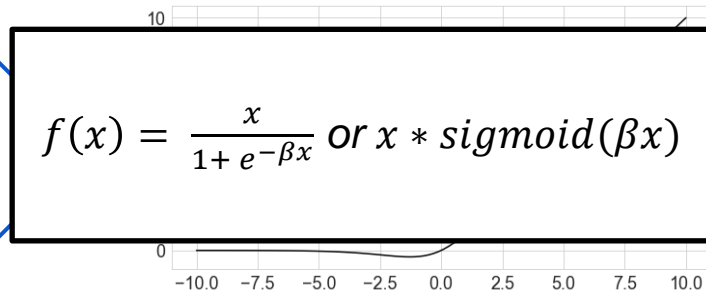
The Mish Activation function

Key features of Mish is its non-monotonicity, which allows it to capture complex patterns in data and improve the performance of the network. Used in deep learning applications that require high performance.



The Swish Activation function

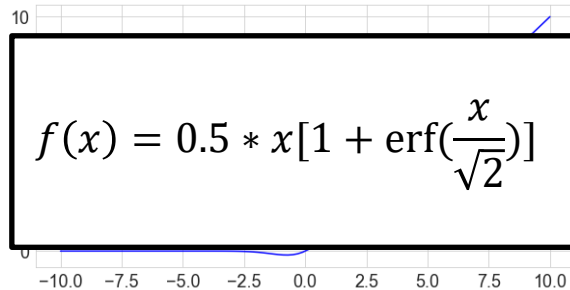
It has the best computational efficiency and easy implementation, which makes it suitable for deep learning applications with large datasets and complex architectures. A promising alternative to existing activation functions in applications where data security is critical.



State-of-the-art AFs (1/3)

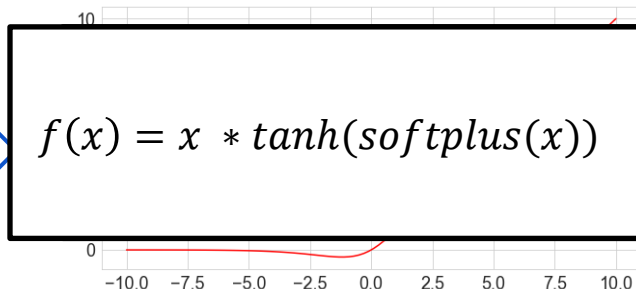
GELU (Gaussian Error Linear Function)

Ability to approximate the identity function in its linear region, which can help preserve information during training and improve the overall performance of the network. Used in Computer vision and NLP.



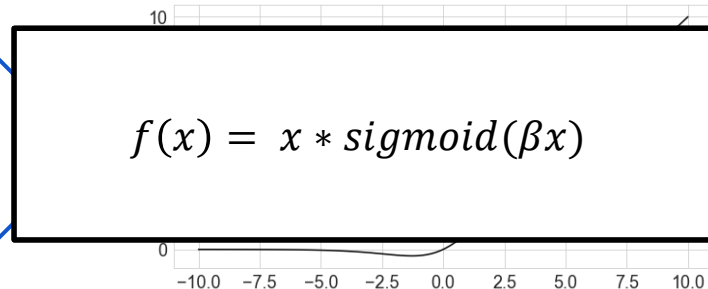
The Mish Activation function

Key features of Mish is its non-monotonicity, which allows it to capture complex patterns in data and improve the performance of the network. Used in deep learning applications that require high performance.



The Swish Activation function

It has the best computational efficiency and easy implementation, which makes it suitable for deep learning applications with large datasets and complex architectures. A promising alternative to existing activation functions in applications where data security is critical.



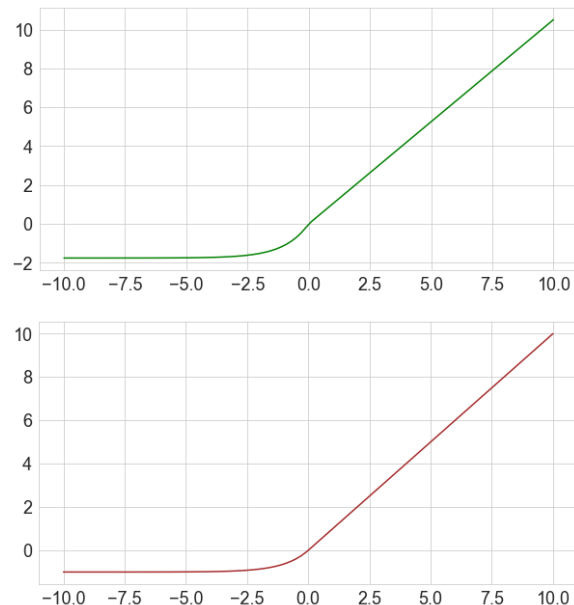
State-of-the-art AFs (2/3)

SELU (Scaled Exponential Linear Unit)

An activation function that introduces self-normalizing properties to the Neural Network. It is very stable, but it can be computationally expensive. It is often used for speech recognition.

AELU (Adaptive Exponential Linear Unit)

This function is exponential before $x = 0$ and goes linear after $x = 0$. The parameter alpha is usually set to 1.5 and can be adjustable, and the same allows function to adapt different type of data and achieve better performance.



State-of-the-art AFs (2/3)

SELU (Scaled Exponential Linear Unit)

An activation function that introduces self-normalizing properties to the Neural Network. It is very stable, but it can be computationally expensive. It is often used for speech recognition.

$$\begin{aligned} f(x) &= \lambda x && \text{if } x > 0 \\ f(x) &= \lambda \alpha (e^x - 1) && \text{if } x \leq 0 \end{aligned}$$



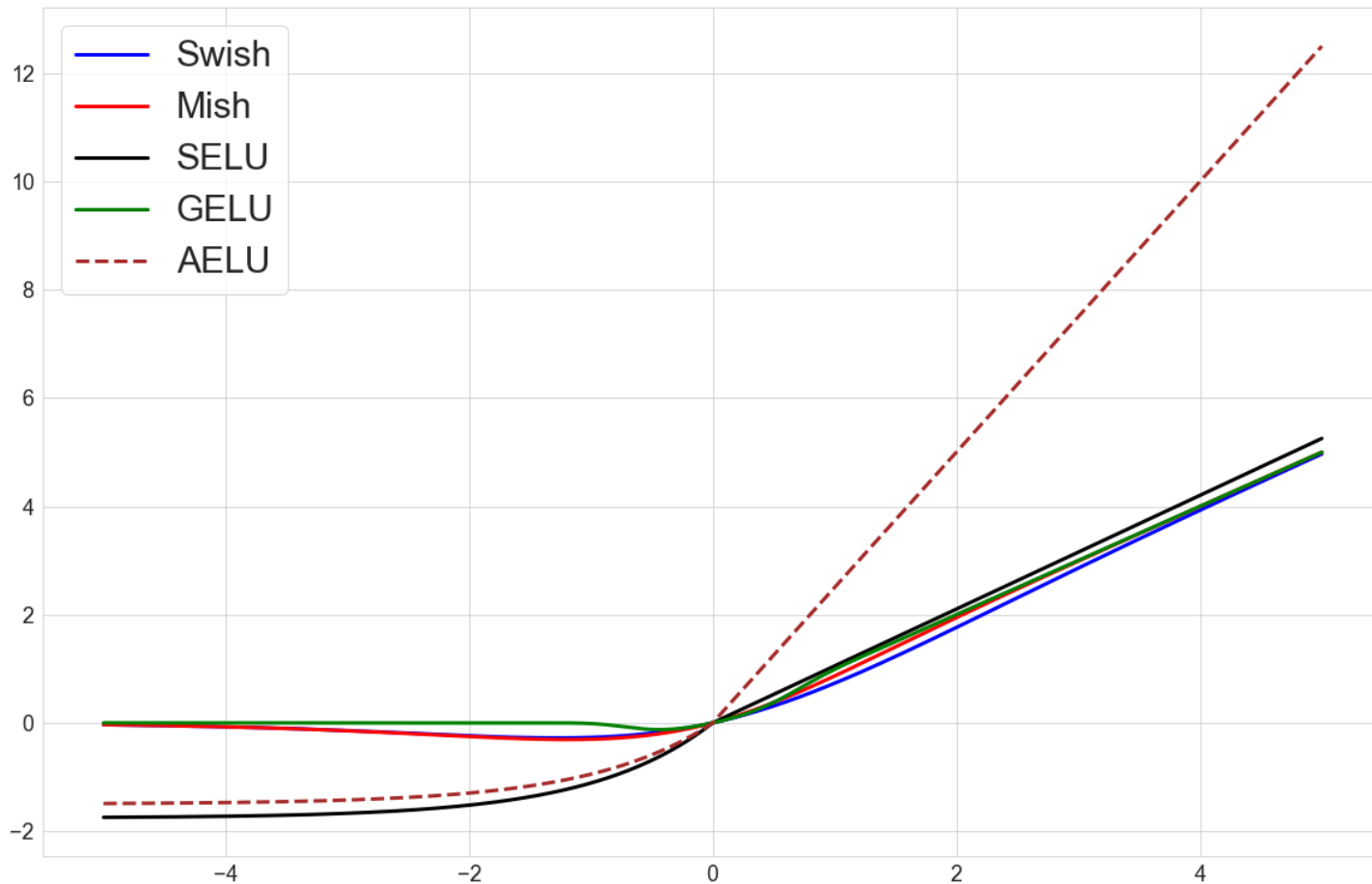
AELU (Adaptive Exponential Linear Unit)

This function is exponential before $x = 0$ and goes linear after $x = 0$. The parameter alpha is usually set to 1.5 and can be adjustable, and the same allows function to adapt different type of data and achieve better performance.

$$\begin{aligned} f(x) &= x && \text{if } x \geq 0 \\ f(x) &= \alpha (e^x - 1) && \text{if } x < 0 \end{aligned}$$



State-of-the-art AFs (3/3)



PROBLEM STATEMENT

Datasets (1/2)

Magic Gamma Telescope Dataset: Classifying high energy gamma and hadron particles based on measured physical quantities using an atmospheric Cherenkov gamma telescope.

Data Set Characteristics:	Multivariate	Number of Instances:	19020	Area:	Physical
Attribute Characteristics:	Real	Number of Attributes:	11	Date Donated	2007-05-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	174533

<https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope>

Covertypes Dataset: Predicting forest cover type from cartographic variables only using binary and quantitative attributes for wilderness areas in northern Colorado.

Data Set Characteristics:	Multivariate	Number of Instances:	581012	Area:	Life
Attribute Characteristics:	Categorical, Integer	Number of Attributes:	54	Date Donated	1998-08-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	446157

<https://archive.ics.uci.edu/ml/datasets/Covertypes>

Datasets (2/2)

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	g
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	g
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	g
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	g
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	g

<https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope>

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_3pm
0	2596	51	3	258	0	510	221	
1	2590	56	2	212	-6	390	220	
2	2804	139	9	268	65	3180	234	
3	2785	155	18	242	118	3090	238	
4	2595	45	2	153	-1	391	220	

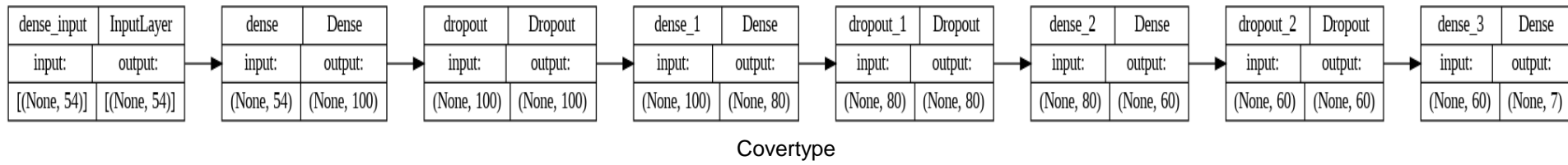
<https://archive.ics.uci.edu/ml/datasets/Covertype>

Python code

```
def buildSequentialModel(hidden_layers_activation, first_layer_node_count, dropout_fraction, nb_classes, weight_class, wt_mean, wt_std, bias_mean, bias_std):  
    # Build a Sequential Model.  
    model = Sequential()  
    # model.add(Flatten(input_shape=(28, 28)))  
  
    model.add(Dense(first_layer_node_count, kernel_initializer=weight_class.normal_weight(wt_mean, wt_std), activation=hidden_layers_activation,  
                    bias_initializer=weight_class.normal_weight(bias_mean, bias_std)))  
    model.add(Dropout(dropout_fraction))  
    model.add(Dense(first_layer_node_count*0.8, kernel_initializer=weight_class.normal_weight(wt_mean, wt_std), activation=hidden_layers_activation,  
                    bias_initializer=weight_class.normal_weight(bias_mean, bias_std)))  
    model.add(Dropout(dropout_fraction))  
    model.add(Dense(first_layer_node_count*0.6, kernel_initializer=weight_class.normal_weight(wt_mean, wt_std), activation=hidden_layers_activation,  
                    bias_initializer=weight_class.normal_weight(bias_mean, bias_std)))  
    model.add(Dropout(dropout_fraction))  
  
    # Output Layer  
    model.add(Dense(nb_classes, activation=hidden_layers_activation, kernel_initializer=weight_class.normal_weight(wt_mean, wt_std),  
                    bias_initializer=weight_class.normal_weight(bias_mean, bias_std)))  
  
    return model
```

```
class CustomWeight(tf.keras.initializers.Initializer):  
    def __init__(self):  
        print('gg')  
    def constant_weight(self, num):  
        return tf.keras.initializers.Constant(num)  
  
    def normal_weight(self, mean, stddev):  
        return tf.keras.initializers.RandomNormal(mean=0., stddev=1.)
```

Neural Network Architecture



RESULTS



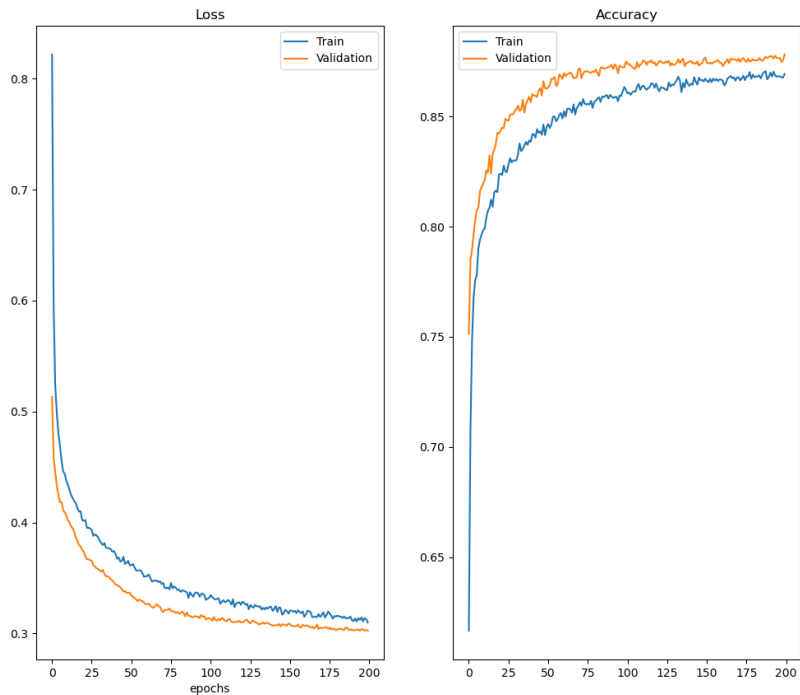
Results (1/3)

	data_Dimension	Classes	Hidden_AF	TrainTest_Split	First_Hidden_Layer_Tensor_Count	Dropouts	Robust_Train_Accuracy	Robust_Test_Accuracy
0	(19020, 11)	2	sigmoid	0.25	100	0.3	0.8880	0.8752
1	(19020, 11)	2	softmax	0.25	100	0.3	0.8915	0.8745
2	(19020, 11)	2	gelu	0.25	100	0.3	0.8045	0.8069
3	(19020, 11)	2	tanh	0.25	100	0.3	0.6480	0.6481
4	(19020, 11)	2	mish	0.25	100	0.3	0.6480	0.6480
5	(19020, 11)	2	relu	0.25	100	0.3	0.6480	0.6480
6	(19020, 11)	2	selu	0.25	100	0.3	0.6480	0.6480

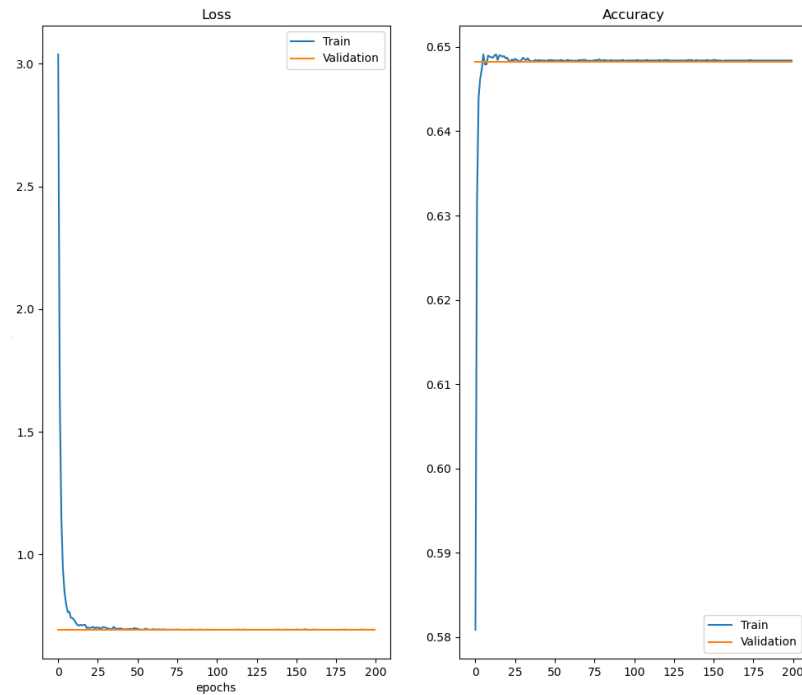
	data_Dimension	Classes	Hidden_AF	TrainTest_Split	First_Hidden_Layer_Tensor_Count	Dropouts	Robust_Train_Accuracy	Robust_Test_Accuracy
0	(581012, 55)	7	sigmoid	0.25	100	0.3	0.854	0.850
1	(581012, 55)	7	softmax	0.25	100	0.3	0.806	0.805
2	(581012, 55)	7	selu	0.25	100	0.3	0.367	0.367
3	(581012, 55)	7	mish	0.25	100	0.3	0.365	0.365
4	(581012, 55)	7	relu	0.25	100	0.3	0.365	0.365
5	(581012, 55)	7	gelu	0.25	100	0.3	0.365	0.365
6	(581012, 55)	7	tanh	0.25	100	0.3	0.062	0.056

Results (2/3)

sigmoid

















ReLU

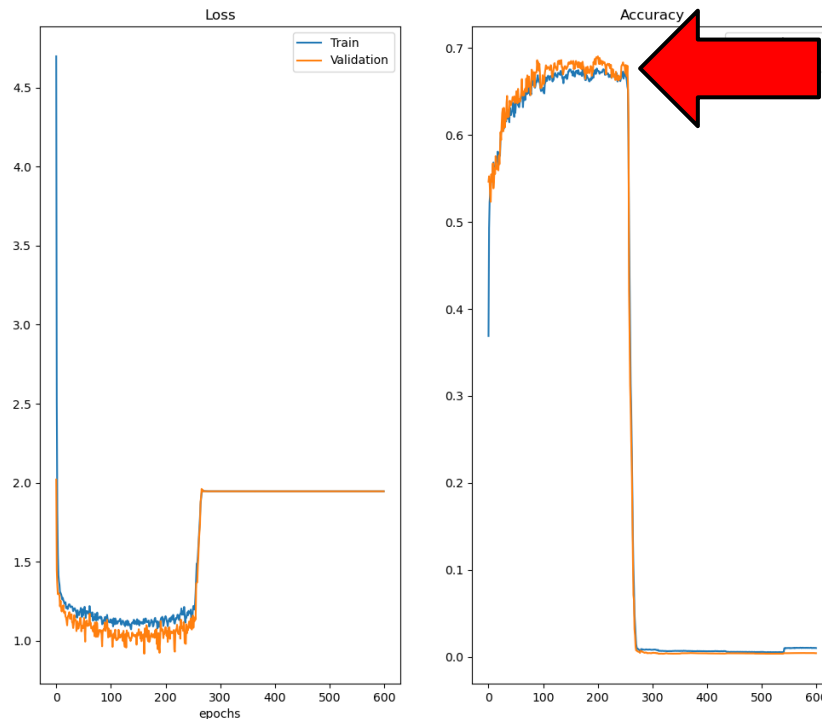


Convergence of 'loss' and 'accuracy' for best and worst performing AFs (Magic Gamma Telescope Dataset)

Results (3/3)

	binary	multi-class
Sigmoid		
Softmax		
Tanh		
Mish		
ReLU		
GELU		
SELU		

→ Example of the vanishing gradient problem:
tanh



Notes on our findings

- The ranking is specific to our chosen neural-network architecture and the specific datasets used
- The performance of individual activation functions are largely dependent on the other model parameters and the data
- There are some general benefits and drawbacks to the individual activation functions however
- Knowing those can give a good intuition about which ones to test in a specific context
- This can significantly speed up the process of hyperparameter tuning

FURTHER STEPS



What's next?

Generally

- Oscillating activation functions
- Closer to the biological functioning of neurons
- Solving classification with fewer neurons
- Reduced training times

→ Non-Monotonic Cubic Unit (NCU)

$$f(z) = z - z^3$$

→ Shifted Quadratic Unit (SQU)

$$f(z) = z^2 + z$$

→ Decaying Sine Unit (DSU)

$$f(z) = \frac{\pi}{2}(\text{sinc}(z - \pi) - \text{sinc}(z + \pi))$$

→ Shifted Sinc Unit (SSU)

$$f(z) = \pi \text{sinc}(z - \pi)$$

Our project

- Use “standard” models like ResNet-20 and “standard” datasets like CIFAR10
- Check how popular activation functions and various state-of-the-art options perform
- Make some modifications to the known functions and see if we can come up with a viable new option
- Test oscillating activation functions and see if our results are similar to current research

References and GitHub

- Géron, A. (2019). *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und Tensorflow* (2. Auflage). O'Reilly Media, 121-130, 281-376, 449-499.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press, 326-366. Retrieved from <http://www.deeplearningbook.org>
- Gustineli, M. (2022). *A survey on recently proposed activation functions for Deep Learning*. arXiv:2204.02921v2.
- Mohammad, M. (2023). *Unification of popular artificial neural network activation functions*. <https://arxiv.org/abs/2302.11007>
- Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton Project Para. *Cornell Aeronautical Laboratory*.



<https://github.com/nilsmart96/comparing-activation-functions>

STAY GOLDEN

