Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 09η Συμπληρωματική: Critique of activation functions

A critique of some very popular activation functions

Sigmoid

$$S(x) = \frac{1}{1 + e^{-x}}$$

A smooth, S-shaped curve maps the input to a value [0…1]

- ➢ **Properties**
  - ✓ **Output Range**: The output of the sigmoid function is bounded between 0 and 1. When the input is large and positive, the output approaches 1. Similarly, when the input is large and negative, the output approaches 0
  - ✓ **Non-linearity**: The sigmoid function introduces non-linearity to the network's decision-making process. This non-linear property allows neural networks to model complex relationships between inputs and outputs
  - ✓ **Smoothness**: The sigmoid function is a smooth and differentiable function which facilitates efficient gradient-based optimization algorithms during the training of neural networks.
- ➢ **Limitations**
  - ❖ **Vanishing Gradients**: The gradients of the sigmoid function become very small for large input values, leading to the problem of vanishing gradients. This can hinder the learning process, especially in deep neural networks
  - ❖ **Output Saturation**: The sigmoid function saturates at the extremes (0 and 1), meaning that when the input is very positive or negative, the output becomes close to 0 or 1, respectively. This saturation can slow down learning as the network becomes less sensitive to changes in the input.
- ➢ **Comments**. Due to these limitations, alternative activation functions like ReLU, Leaky ReLU, and variants have gained popularity, especially in deep learning architectures. However, the sigmoid function still finds applications in specific scenarios, such as the output layer of binary classification models

ReLU (1/2)

$$S(x) = max\{0, x\}$$

The Rectified Linear Unit (ReLU) is a widely used activation function particularly for deep learning models. It is known for its simplicity and effectiveness in overcoming the limitations of other activation functions like the sigmoid and tanh functions

- **Properties**
  - ✓ **Non-linearity**: Like other activation functions, ReLU introduces non-linearity to the network, enabling it to learn and model complex relationships in the data. It allows neural networks to approximate any non-linear function.
  - ✓ **Sparsity**: ReLU encourages sparsity in neural networks. Since it outputs 0 for negative inputs, ReLU neurons can be completely inactive for specific inputs. This sparsity can lead to more efficient and concise representations of the data.
  - ✓ **Avoiding Vanishing Gradient**: ReLU helps alleviate the vanishing gradient problem encountered in deep neural networks. The derivative of ReLU is either 0 or 1, which prevents the gradients from vanishing as the network gets deeper. This property facilitates more effective backpropagation and faster training
  - ✓ **Computational Efficiency**: The ReLU function is computationally efficient to evaluate compared to more complex tasks like the sigmoid and tanh. It involves simple thresholding and avoids costly exponential calculations.

ReLU (2/2)

$$S(x) = max\{0, x\}$$

- **Limitations**
  - **Dead Neurons**: ReLU neurons can sometimes become "dead" or non-responsive, where the neuron's output is always 0 for any input. Once a neuron dies, it no longer contributes to the learning process. This issue can be addressed using variants of ReLU, such as Leaky ReLU or Parametric ReLU (PReLU).
  - **Output Saturation**: ReLU saturates at the upper bound, outputting the input value for any positive input. This saturation can cause the neuron to lose sensitivity to large positive values, limiting its ability to learn further from those inputs
- **Comments**. Due to its simplicity and effectiveness, ReLU is widely used in deep learning architectures, particularly in convolutional neural networks (CNNs). It has performed excellently in various computer vision and natural language processing tasks.

tanh (1/2)

$$S(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It is similar to the sigmoid function but is centered around zero and ranges between -1 and 1.

- **Properties**
  - ✓ **Non-linearity**: Like other activation functions, tanh introduces non-linearity to the network, enabling it to learn and model complex relationships in the data. It allows neural networks to approximate any non-linear function.
  - ✓ **Symmetry**: The tanh function is symmetric around the origin (0, 0). It produces negative outputs for negative inputs and positive outputs for positive inputs, resulting in a smooth S-shaped curve.
  - ✓ **Output Range**: The output of the tanh function is bounded between -1 and 1. When the input is large and positive, the output approaches 1. Similarly, when the input is large and negative, the output approaches -1.
  - ✓ **Zero-Centred**: Unlike the sigmoid function, which is centred around 0.5, the tanh function is centred around zero. This can be advantageous in some cases, such as when the input data is zero-centred or when the model benefits from negative and positive activations.

tanh (2/2)

$$S(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Limitations**
  - The tanh function shares some similarities with the sigmoid function, but it has a steeper gradient, which makes it more sensitive to changes in the input.
  - However, like the sigmoid function, it can still suffer from the vanishing gradient problem for very large/small input values.
- **Comments**. The tanh function is used in various neural network architectures, particularly in recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, where its zero-centred property and non-linear characteristics can be beneficial. However, in recent years, the popularity of the ReLU and its variants has increased, mainly due to their simplicity and better performance in deep learning models.

softmax (1/2)

$$S(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The softmax function is an activation function commonly used in the output layer of a neural network for multi-class classification problems. It takes a vector of real numbers as input and normalizes it into a probability distribution, where the sum of the probabilities equals 1

- **Properties**
  - ✓ **Probability Distribution**: The softmax function transforms the input values into a probability distribution, where each value represents the probability of the corresponding class. The output values are positive and sum up to 1, making it suitable for multi-class classification problems.
  - ✓ **Sensitivity to Input Differences**: The softmax function amplifies the differences between the input values, which means that larger input values will correspond to higher probabilities. This property allows the neural network to make more confident predictions for higher-score classes.
  - ✓ **Differentiability**: The softmax function is differentiable, which is crucial for backpropagation and gradient-based optimization algorithms used during the training process of neural networks.

softmax (2/2)

$$S(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- **Limitations**
  - It is important to note that the softmax function is sensitive to outliers and large input values, which can lead to numerical instability. This issue can be mitigated by subtracting the maximum input value from each input vector element, known as "logit shifting" or "logit scaling," before applying the softmax function. This helps prevent numerical overflow or underflow.

- **Comments**. The softmax function is typically used in the final layer of a neural network for multi-class classification tasks, where the goal is to assign an input to one of several possible classes. The class with the highest probability outputted by the softmax function is usually considered the predicted class. Overall, the softmax function is a fundamental tool in multi-class classification tasks, enabling the neural network to provide a probabilistic interpretation of its predictions.

LReLU (1/2)

$$S(x) = max\{ax, x\}$$

The Leaky ReLU is a variant of the ReLU activation function that addresses the “dead ReLU” problem. The Leaky ReLU introduces a small slope for negative inputs, allowing the neurons to have a non-zero output even when the input is negative. This helps mitigate the “dead” or non-responsive neurons in regular ReLU.

Parameter a is a small positive constant (usually a small fraction like 0.01). If x is positive, the function behaves like a regular ReLU, outputting x . However, if x is negative, the function returns ax instead of 0.

LReLU (2/2)

$S(x) = max\{ax, x\}$

- **Properties**
  - ✓ **Non-linearity**: Similar to ReLU, the Leaky ReLU introduces non-linearity to the network, enabling it to learn and model complex relationships in the data.
  - ✓ **Avoiding Dead Neurons**: Introducing a non-zero slope for negative inputs helps mitigate the problem of dead or non-responsive neurons encountered in regular ReLU. With a small positive slope, even neurons that receive negative inputs can still contribute to the learning process.
  - ✓ **Continuous and Piecewise Linear**: The Leaky ReLU function is continuous and piecewise linear, meaning it has a defined derivative for all values. This property facilitates backpropagation and efficient gradient-based optimization algorithms during the training process.
- **Comments**. The choice between ReLU and Leaky ReLU depends on the problem and the characteristics of the data. Leaky ReLU is often preferred over regular ReLU when the risk of dead neurons is high or when having a more diverse range of activations is desirable by allowing negative values. Other variants of ReLU, such as Parametric ReLU (PReLU) which generalizes the concept of Leaky ReLU by allowing the a parameter to be learned during the training process rather than being predefined. This enables the network to determine the slope based on the data adaptively. Leaky ReLU is a popular choice in neural networks, especially in scenarios where regular ReLU may lead to dead neurons or a broader range of activation values is desired.

PReLU (1/2)

$$S(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \end{cases}$$

Parametric ReLU (PReLU) is an activation function that extends the Rectified Linear Unit (ReLU) functionality by introducing a learnable parameter a. In PReLU, the slope a for negative inputs is not fixed but is learned during the training process.

- **Properties**
  - ✓ **Non-linearity**: Similar to ReLU and Leaky ReLU, PReLU introduces non-linearity to the network, enabling it to model complex relationships in the data.
  - ✓ **Adaptive Slope**: The main advantage of PReLU over Leaky ReLU is its ability to learn the optimal slope for each neuron. This adaptability can improve the flexibility and expressiveness of the network.
  - ✓ **Mitigating Dead Neurons**: By allowing negative values to pass through with an adaptive slope, PReLU helps prevent the issue of dead or non-responsive neurons encountered in regular ReLU.

PReLU (2/2)

$$S(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \end{cases}$$

- **Comments**. The choice between ReLU, Leaky ReLU, and PReLU depends on the specific problem and the characteristics of the data. PReLU is often used when there is a concern about dead neurons or when it is desirable to have a learnable slope that can better capture the nuances of the data. It's worth noting that PReLU introduces additional parameters to be learned, which increases the model's complexity and computational requirements. Consequently, PReLU might be more suitable for larger datasets and more complex models. PReLU has been successfully applied in various deep learning architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), and has demonstrated improved performance in specific scenarios compared to other activation functions.

ELU (1/2)

$$S(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{if } x < 0 \end{cases}$$

The Exponential Linear Unit (ELU) aims to overcome some of the limitations of the ReLU, such as the dying ReLU problem and the saturation of negative values. ELU introduces a differentiable function that smoothly saturates negative inputs and gives negative values a non-zero output. $a \geq 0$

- **Properties**
  - ✓ **Non-linearity**: Like other activation functions, ELU introduces non-linearity to the network, enabling it to model complex relationships in the data.
  - ✓ **Smooth Saturation**: The ELU function smoothly saturates negative inputs, avoiding the abrupt saturation of ReLU. This helps to alleviate the issue of dead or non-responsive neurons encountered in ReLU.
  - ✓ **Continuity and Differentiability**: ELU is a continuous and differentiable function, allowing for efficient backpropagation and gradient-based optimization algorithms during training.
  - ✓ **Negative Output**: ELU allows negative values to have a non-zero output, which can be helpful in cases where capturing negative activations is essential for the task.
  - ✓ **Exponential Decay**: The negative values in the ELU function decay exponentially as x approaches negative infinity, providing a more robust and well-behaved response to extreme negative inputs.

ELU (2/2)

$$S(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{if } x < 0 \end{cases}$$

- **Comments**. It’s important to note that ELU introduces additional computational complexity compared to ReLU and other more straightforward activation functions due to the exponential function. However, the improved performance and mitigated limitations make it a popular choice, especially in deep learning architectures. ELU has been used in various applications and has shown promising results in reducing overfitting, improving learning efficiency, and achieving better generalization compared to ReLU, particularly in deep neural networks. When choosing an activation function, it is vital to consider the specific problem and the characteristics of the data and experiment with different options to find the most suitable activation function for optimal performance.

$$S(x) = 0.5 \times x \times (1 + tanh(\sqrt{2/\pi} \times (x + 0.044715 \times x^3)))$$

GELU (1/2)

The Gaussian Error Linear Unit (GELU) is an activation function that aims to combine the desirable properties of the Gaussian distribution and the rectifier function. It provides a smooth approximation to the rectifier while preserving the desirable properties of both functions.

- **Properties**
  - ✓ **Non-linearity**: GELU introduces non-linearity to the network, allowing it to model complex relationships in the data.
  - ✓ **Smoothness**: GELU is a smooth function that transitions from negative to positive inputs. It is differentiable everywhere, facilitating backpropagation and gradient-based optimization.
  - ✓ **Gaussian Approximation**: GELU approximates a Gaussian cumulative distribution function (CDF). This approximation allows GELU to exhibit similar behaviour to the rectifier function while providing smoother gradients.
  - ✓ **Saturation**: GELU saturates at the upper and lower bounds, preventing significant activations. This can be beneficial for the stability and convergence of neural networks.

$$S(x) = 0.5 \times x \times (1 + tanh(\sqrt{2/\pi} \times (x + 0.044715 \times x^3)))$$

GELU (2/2)

- **Comments**. GELU has gained popularity in deep learning models, particularly in natural language processing tasks and transformer architectures. It has shown improved convergence speed and generalization performance compared to other activation functions, such as ReLU. However, it is worth noting that GELU introduces additional computational complexity due to trigonometric and exponential operations. Therefore, it might have a slight impact on the overall computational efficiency of the model. When choosing an activation function, it is essential to consider the specific requirements of the problem and experiment with different options to find the most suitable activation function for optimal performance.

Pure linear (1/2)

$$S(x) = x$$

The Linear activation function, also known as the identity function, is one of the simplest activation functions used in neural networks. It applies a linear transformation to the input, meaning the output equals the input without any non-linear mapping.

- **Properties**
  - ✓ **Linearity**: As the name suggests, the Linear activation function introduces linearity to the network. It performs a simple scaling of the input without introducing any non-linear transformations.
  - ✓ **No Activation**: Unlike other activation functions that introduce non-linearities to capture complex relationships, the Linear activation function does not alter the input values. It is essentially a pass-through function.
  - ✓ **Limited Representation Power**: Since the Linear activation function does not introduce non-linearity, it has limited representation power. Neural networks with only linear activation functions can only learn linear relationships between the input and output.
  - ✓ **Gradient Stability**: The gradient of the Linear activation function is constant and does not depend on the input. This can be advantageous in some cases, as it ensures stable gradients during backpropagation.

Pure linear (2/2)

$$S(x) = x$$

- **Comments**. The Linear activation function is often used in the output layer of regression problems, where the goal is to predict continuous values. It allows the neural network to directly output values without any transformation. However, non-linear activation functions are typically used for most other tasks, such as classification or complex pattern recognition. Non-linear activation functions enable neural networks to learn and represent more complex relationships in the data. It's important to note that even though individual layers of a neural network may use linear activation functions, stacking multiple linear layers does not increase the model's representative capacity beyond that of a single linear layer. To model complex relationships, non-linear activation functions are essential in intermediate layers of the network.