2_DL_ActivationFunction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

Training & Evaluating using STEP Activation Function

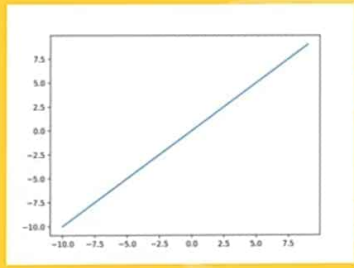
```
activationFn = lambda x: 0 if x < 0 else 1 #step activation function (if i/p is negative o/p is 0 else 1)
for j in range(n):
    x, expected = choice(dataset)
    result = dot(weights, x)
    err = expected-activationFn(result)
    weights += r * err * x

for x, _ in dataset:
    result = dot(x, weights)
    print("ResultBAFn: {} ResultAFn: {}".format(round(result,3), activationFn(result)))
```

```
ResultBAFn: -0.048 ResultAFn: 0
ResultBAFn: 0.468 ResultAFn: 1
ResultBAFn: 0.305 ResultAFn: 1
ResultBAFn: 0.82 ResultAFn: 1
```

0s completed at 19:14

LINEAR ACTIVATION FUNCTION

Linear functions are pretty simple. It returns what it gets as input.



2_DL_ActivationFunction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
ResultBAFn: 0.82 ResultAFn: 1
```

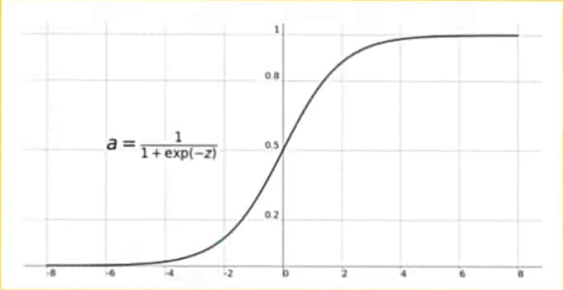
Training & Evaluating using LINEAR Activation Function

```
activationFn = lambda x: x

for j in range(n):
    x, expected = choice(dataset)
    result = dot(weights, x)
    err = expected-activationFn(result)
    weights += r * err * x

for x, _ in dataset:
    result = dot(x, weights)
    print("ResultBAFn: {} ResultAFn: {}".format(round(result,3), activationFn(result)))
```

0s completed at 19:14

SIGMOID ACTIVATION FUNCTION

- Sigmoid function returns the value between 0.0 and 1.0
- Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0
- It has a characteristic S-shaped curve



1_DL_Perceptron.ipynb - Colabor | 2_DL_ActivationFunction.ipynb -

colab.research.google.com/drive/1D1iRtJQ90W6h_xQBzpkMksXLvq0c2QAd#scrollTo=xV6bz_6-3x77

Apps | Road Trip Journey R... | (2) SLAM for the ro... | Sensors - ROS Wiki | (2) [ROS Q&A] 031... | The Secret of Nikol... | (2) How to Set Up T... | (8) Setting Up IR re... | jetson.pdf | How to recover the... | Reading list

2_DL_ActivationFunction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

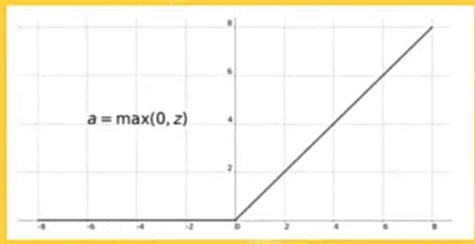
Training & Evaluating using SIGMOID Activation Function

```
activationFn = lambda x: 1/(1+np.exp(-x))
err = []
for j in range(n):
    x, expected = choice(dataset)
    result = dot(weights, x)
    err = expected-activationFn(result)
    error.append(err)
    weights += r * err * x

for x, _ in dataset:
    result = dot(x, weights)
    print("ResultBAFn: {} ResultAFn: {}".format(round(result,3), activationFn(result)))
```

0s completed at 19:14

RELU ACTIVATION FUNCTION

- RELU is less computational expensive than the other non linear activation functions
- RELU returns 0 if the x (input) is less than 0
- RELU returns x if the x (input) is greater than 0



2_DL_ActivationFunction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

Training & Evaluating using RELU Activation Function

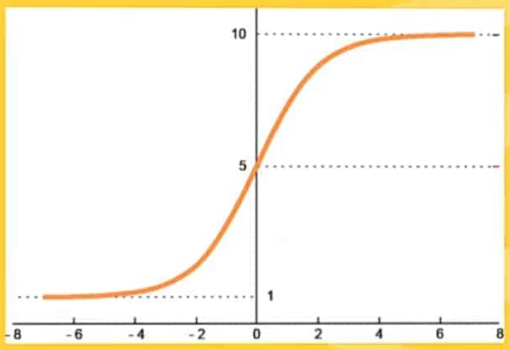
```
activationFn = lambda x: 0 if x < 0 else x
for j in range(n):
    x, expected = choice(dataset)
    result = dot(weights, x)
    err = expected-activationFn(result)
    weights += r * err * x

for x, _ in dataset:
    result = dot(x, weights)
    print("ResultBAFn: {} ResultAFn: {}".format(round(result,3), activationFn(result)))
```

Training & Evaluating using SOFTMAX Activation Function

0s completed at 19:20

SOFTMAX ACTIVATION FUNCTION

- Softmax turns logits, the numeric output of the last linear layer of a multi-class classification neural network into probabilities
- The function is great for classification problems, especially if you're dealing with multi-class classification problems, as it will report back the "confidence score" for each class



2_DL_ActivationFunction.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

Training & Evaluating using SOFTMAX Activation Function

```
activationFn = lambda x: np.exp(x) / np.sum(np.exp(x), axis=0)
err = []
for j in range(n):
    x, expected = choice(dataset)
    result = dot(weights, x)
    err = expected-activationFn(result)
    error.append(err)
    weights += r * err * x

for x, _ in dataset:
    result = dot(x, weights)
    print("ResultBAFn: {} ResultAFn: {}".format(round(result,3), activationFn(result)))
```

```
ResultBAFn: -6.485 ResultAFn: 1.0
ResultBAFn: -4.395 ResultAFn: 1.0
```

0s completed at 19:40

Deep Learning Terminology - 1

Batch & Batch size

- The set of examples used in one iteration (that is, one gradient update) of model training.
- **Batch Size:** The number of examples in a batch

Deep Learning Terminology - 2

Normalization

- The process of converting an actual range of values into a standard range of values, typically -1 to +1 or 0 to 1.
- For example, suppose the natural range of a certain feature is 800 to 6,000. Through subtraction and division, you can normalize those values into the range -1 to +1.

Deep Learning Terminology - 3

Batch normalization

- Normalizing the input or output of the activation functions in a hidden layer.

Benefits

- Make neural networks more stable
- Enable higher learning rates
- Reduce overfitting