jczic / MicroMLP Público

Código Asuntos 4 Solicitudes de extracción Comportamiento Seguridad Perspectivas

Maestro

MicroMLP / LÉAME.md

jczic Solucionar problemas de funciones derivadas

1 colaborador

MicroMLP es un perceptrón multicapa de red neuronal micro artificial (principalmente utilizado en módulos ESP32 y Pycom)

Muy fácil de integrar y muy ligero con un solo archivo:

- `"microMLP.py"`

Características de MicroMLP:

- Estructura modificable multicapa y conexiones
- Sesgo integrado en las neuronas
- Plasticidad de las conexiones incluidas
- Funciones de activación por capa
- Parameters Alpha, Eta and Gain
- Managing set of examples and learning
- QLearning functions to use reinforcement learning

- Save and load all structure to/from json file
- Various activation functions :
  - Heaviside binary step
  - Logistic (sigmoid or soft step)
  - Hyperbolic tangent
  - SoftPlus rectifier
  - ReLU (rectified linear unit)
  - Gaussian function

Use deep learning for :

- Signal processing (speech processing, identification, filtering)
- Image processing (compression, recognition, patterns)
- Control (diagnosis, quality control, robotics)
- Optimization (planning, traffic regulation, finance)
- Simulation (black box simulation)
- Classification (DNA analysis)
- Approximation (unknown function, complex function)

Using *MicroMLP* static functions :

Name	Function
Create	`mlp = MicroMLP.Create(neuronsByLayers, activationFuncName, layersAutoConnectFunction=None, useBiasValue=1.0)`
LoadFromFile	`mlp = MicroMLP.LoadFromFile(filename)`

Using *MicroMLP* speedly creation of a neural network :

```
from microMLP import MicroMLP
mlp = MicroMLP.Create([3, 10, 2], "Sigmoid", MicroMLP.LayersFullConnect)
```

Using *MicroMLP* main class :

Name	Function
Constructor	`mlp = MicroMLP()`
GetLayer	`layer = mlp.GetLayer(layerIndex)`
GetLayerIndex	`idx = mlp.GetLayerIndex(layer)`
RemoveLayer	`mlp.RemoveLayer(layer)`
GetInputLayer	`inputLayer = mlp.GetInputLayer()`
GetOutputLayer	`outputLayer = mlp.GetOutputLayer()`

Learn	`ok = mlp.Learn(inputVectorNNValues, targetVectorNNValues)`
Test	`ok = mlp.Test(inputVectorNNValues, targetVectorNNValues)`
Predict	`outputVectorNNValues = mlp.Predict(inputVecto`
QLearningLearnForChosenAction	`ok = mlp.QLearningLearnForChosenAction(stateVectorN rewardNNValue, pastStateVectorNNValues, chosenActionIndex, terminalState=True, discountFactorNNValue=None)`
QLearningPredictBestActionIndex	`bestActionIndex = mlp.QLearningPredictBestActionIndex(stateVecto`
SaveToFile	`ok = mlp.SaveToFile(filename)`
AddExample	`ok = mlp.AddExample(inputVectorNNValues, targetVectorNNValues)`
ClearExamples	`mlp.ClearExamples()`
LearnExamples	`learnCount = mlp.LearnExamples(maxSeconds=30, maxCount=None, stopWhenLearned=True, printMAEAverage=True)`

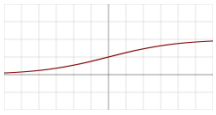
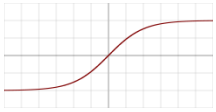
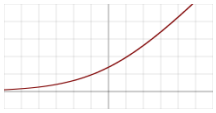
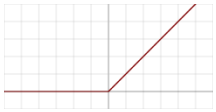
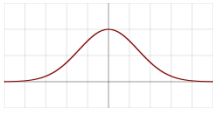
Property	Example	Read/Write
Layers	`mlp.Layers`	get
LayersCount	`mlp.LayersCount`	get
IsNetworkComplete	`mlp.IsNetworkComplete`	get
MSE	`mlp.MSE`	get
MAE	`mlp.MAE`	get
MSEPercent	`mlp.MSEPercent`	get
MAEPercent	`mlp.MAEPercent`	get
ExamplesCount	`mlp.ExamplesCount`	get

Using *MicroMLP* to learn the XOr problem (with hyperbolic tangent) :

```
from microMLP import MicroMLP

mlp = MicroMLP.Create( neuronsByLayers           = [2, 2, 1],
                       activationFuncName        = MicroMLP.ACTFUNC_TANH,
                       layersAutoConnectFunction = MicroMLP.LayersFullConnect )

nnFalse  = MicroMLP.NNValue.FromBool(False)
nnTrue   = MicroMLP.NNValue.FromBool(True)

mlp.AddExample( [nnFalse, nnFalse], [nnFalse] )
mlp.AddExample( [nnFalse, nnTrue ], [nnTrue ] )
mlp.AddExample( [nnTrue , nnTrue ], [nnFalse] )
mlp.AddExample( [nnTrue , nnFalse], [nnTrue ] )

learnCount = mlp.LearnExamples()

print( "LEARNED :" )
print( "  - False xor False = %s" % mlp.Predict([nnFalse, nnFalse])[0].AsBool )
print( "  - False xor True  = %s" % mlp.Predict([nnFalse, nnTrue] )[0].AsBool )
print( "  - True  xor True  = %s" % mlp.Predict([nnTrue , nnTrue] )[0].AsBool )
print( "  - True  xor False = %s" % mlp.Predict([nnTrue , nnFalse])[0].AsBool )

if mlp.SaveToFile("mlp.json") :
        print( "MicroMLP structure saved!" )
```

Variable	Description	Default
`mlp.Eta`	Weighting of the error correction	0.30

Variable	Description	Default
`mlp.Alpha`	Strength of connections plasticity	0.75
`mlp.Gain`	Network learning gain	0.99
`mlp.CorrectLearnedMAE`	Threshold of self-learning error	0.02

Graphe	Activation function name	Const	Detail
	`"Heaviside"`	MicroMLP.ACTFUNC_HEAVISIDE	Heaviside binary step
	`"Sigmoid"`	MicroMLP.ACTFUNC_SIGMOID	Logistic (sigmoid or soft step)
	`"TanH"`	MicroMLP.ACTFUNC_TANH	Hyperbolic tangent
	`"SoftPlus"`	MicroMLP.ACTFUNC_SOFTPLUS	SoftPlus rectifier
	`"ReLU"`	MicroMLP.ACTFUNC_RELU	Rectified linear unit
	`"Gaussian"`	MicroMLP.ACTFUNC_GAUSSIAN	Gaussian function

Layers auto-connect function	Detail
`MicroMLP.LayersFullConnect`	Network fully connected

Using *MicroMLP.Layer* class :

Name	Function
Constructor	`layer = MicroMLP.Layer(parentMicroMLP, activationFuncName=None, neuronsCount=0)`
GetLayerIndex	`idx = layer.GetLayerIndex()`

Name	Function
GetNeuron	`neuron = layer.GetNeuron(neuronIndex)`
GetNeuronIndex	`idx = layer.GetNeuronIndex(neuron)`
AddNeuron	`layer.AddNeuron(neuron)`
RemoveNeuron	`layer.RemoveNeuron(neuron)`
GetMeanSquareError	`mse = layer.GetMeanSquareError()`
GetMeanAbsoluteError	`mae = layer.GetMeanAbsoluteError()`
GetMeanSquareErrorAsPercent	`mseP = layer.GetMeanSquareErrorAsPercent()`
GetMeanAbsoluteErrorAsPercent	`maeP = layer.GetMeanAbsoluteErrorAsPercent()`
Remove	`layer.Remove()`

Property	Example	Read/Write
ParentMicroMLP	`layer.ParentMicroMLP`	get
ActivationFuncName	`layer.ActivationFuncName`	get
Neurons	`layer.Neurons`	get
NeuronsCount	`layer.NeuronsCount`	get

Using *MicroMLP.InputLayer(Layer)* class :

Name	Function
Constructor	`inputLayer = MicroMLP.InputLayer(parentMicroMLP, neuronsCount=0)`
SetInputVectorNNValues	`ok = inputLayer.SetInputVectorNNValues(inputVectorNNValues)`

Using *MicroMLP.OutputLayer(Layer)* class :

Name	Function

Name	Function
Constructor	`outputLayer = MicroMLP.OutputLayer(parentMicroMLP, activationFuncName, neuronsCount=0)`
GetOutputVectorNNValues	`outputVectorNNValues = outputLayer.GetOutputVectorNNValues()`
ComputeTargetLayerError	`ok = outputLayer.ComputeTargetLayerError(targetVectorNNVa`

Using *MicroMLP.Neuron* class :

Name	Function
Constructor	`neuron = MicroMLP.Neuron(parentLayer)`
GetNeuronIndex	`idx = neuron.GetNeuronIndex()`
GetInputConnections	`connections = neuron.GetInputConnections()`
GetOutputConnections	`connections = neuron.GetOutputConnections()`
AddInputConnection	`neuron.AddInputConnection(connection)`
AddOutputConnection	`neuron.AddOutputConnection(connection)`
RemoveInputConnection	`neuron.RemoveInputConnection(connection)`
RemoveOutputConnection	`neuron.RemoveOutputConnection(connection)`
Establecer sesgo	`neuron.SetBias(bias)`
Obtener sesgo	`neuron.GetBias()`
EstablecerSalidaNNValor	`neuron.SetOutputNNValue(nnvalue)`
Calcular valor	`neuron.ComputeValue()`
ComputeError	`neuron.ComputeError(targetNNValue=None)`
Eliminar	`neuron.Remove()`

Propiedad	Ejemplo	Leer escribir
ParentLayer	`neuron.ParentLayer`	obtener
Salida calculada	`neuron.ComputedOutput`	obtener
CalculatedDeltaError	`neuron.ComputedDeltaError`	obtener

Propiedad	Ejemplo	Leer escribir
Error de señal calculada	`neuron.ComputedSignalError`	obtener

Usando la clase *MicroMLP.Connection* :

Nombre	Función
Constructor	`connection = MicroMLP.Connection(neuronSrc, neuronDst, weight=None)`
Eliminar	`connection.Remove()`

254 líneas (208 turnos) | 9.96KB

Propiedad	Ejemplo	Leer escribir
NeuronSrc	`connection.NeuronSrc`	obtener
NeuronDst	`connection.NeuronDst`	obtener
Peso	`connection.Weight`	obtener

Usando la clase *MicroMLP.Bias* :

Nombre	Función
Constructor	`bias = MicroMLP.Bias(neuronDst, value=1.0, weight=None)`
ActualizarPeso	`bias.UpdateWeight(eta, alpha)`
Eliminar	`bias.Remove()`

Propiedad	Ejemplo	Leer escribir
NeuronDst	`bias.NeuronDst`	obtener
Valor	`bias.Value`	obtener
Peso	`bias.Weight`	obtener

Usando funciones estáticas *MicroMLP.NNValue :*

Nombre	Función

Nombre	Función
DePorcentaje	`nnvalue = MicroMLP.NNValue.FromPercent(value)`
nuevoporcentaje	`nnvalue = MicroMLP.NNValue.NewPercent()`
DeByte	`nnvalue = MicroMLP.NNValue.FromByte(value)`
NuevoByte	`nnvalue = MicroMLP.NNValue.NewByte()`
DesdeBool	`nnvalue = MicroMLP.NNValue.FromBool(value)`
NuevoBool	`nnvalue = MicroMLP.NNValue.NewBool()`
de señal analógica	`nnvalue = MicroMLP.NNValue.FromAnalogSignal(value)`
Nueva señal analógica	`nnvalue = MicroMLP.NNValue.NewAnalogSignal()`

Usando la clase *MicroMLP.NNValue* :

Nombre	Función
Constructor	`nnvalue = MicroMLP.NNValue(minValue, maxValue, value)`

Propiedad	Ejemplo	Leer escribir
como flotar	`nnvalue.AsFloat = 639.513`	obtener / establecer
AsInt	`nnvalue.AsInt = 12345`	obtener / establecer
como porcentaje	`nnvalue.AsPercent = 65`	obtener / establecer
como byte	`nnvalue.AsByte = b'\x75'`	obtener / establecer
comoBool	`nnvalue.AsBool = True`	obtener / establecer
como señal analógica	`nnvalue.AsAnalogSignal = 0.39472`	obtener / establecer

Por JC`zic para HC² ;')

Mantenlo simple, estúpido 👍