Neural Network

Li Ju

22/06 - 03/07

Content

Overview

Keras implementation

C implementation

Activation functions/Loss functions ...

Chosen datasets for test

Overview

Classification/regression model:

Input --(Neural Network)--> Output

training

testing

prediction

Advantages:

1. Flexibility
2. Less assumption
3. Generalizability

Disadvantages:

1. Costly computation
2. Lack of interpretability
3. Structure flexibility

Keras implementation

training_x matrix (features)

↓

input_dim = #features, output_dim = #nodes, AF = “relu”

↓

input_dim = #upper_nodes, output_dim = #nodes, AF = “relu”

↓

input_dim = #upper_nodes, output_dim = #categories, AF = “softmax”

↓

output_y matrix ~ target_y: Loss function -> Error

Gradient descending:
Minimize error

Cross validation:
Avoid overfitting

Epoch:
#training times through entire dataset

Batch size:
#samples for each iteration

However…

Training mechanism?

We need mathematical details!

C implementation for a single-hidden-layer network

Scalar-based

Matrix-based

Notation:

$\boldsymbol{X}_{m\times n}$: Input matrix with m samples and n features

$\boldsymbol{Y}_{m\times t}$: Target matrix with m samples and t predicted output

$\boldsymbol{Q}^T$: Transpose of matrix $\boldsymbol{Q}$

L: Output layer

n^a: number of nodes in layer a

$\boldsymbol{W}^a_{n^{a-1}\times n^a}$: Weight matrix between upper layer $a-1$ and lower layer a, the dimension of which is $n^{a-1}\times n^a$

$\boldsymbol{B}^a_{1\times n^a}$: Bias matrix(vector) between upper layer $a-1$ and lower layer$a$, the dimension of which is $1\times n^a$

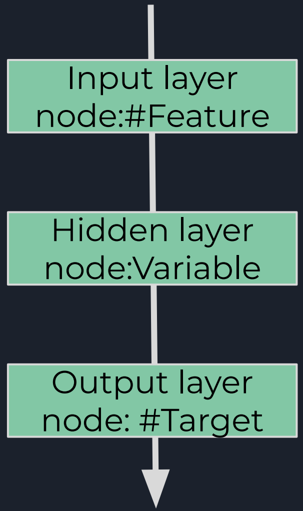
$\boldsymbol{Z}^a_{m\times n^a}$: Weighted input for layer a, the dimension of which is $m\times n^a$

$\boldsymbol{A}^a_{m\times n^a}$: Output matrix of layer $a-1$, or, input matrix of layer a

$\sigma^a(\boldsymbol{Z})$: Activation function of layer a mapping matrix $\boldsymbol{Z}$ element-wise

E: Error calculated by loss function

$\frac{\partial Y}{\partial X}$: Differential of variable Y on variable X



Initialization:

Let all elements of every ***W*** and ***B*** matrices be a random number within -1 and 1;

Forward propagation:

$\boldsymbol{X}_{m\times n}$: Input matrix with m samples and n features

$\boldsymbol{Y}_{m\times t}$: Target matrix with m samples and t predicted output

$\boldsymbol{Q}^T$: Transpose of matrix $\boldsymbol{Q}$

L: Output layer

n^a: number of nodes in layer a

$\boldsymbol{W}^a_{n^{a-1}\times n^a}$: Weight matrix between upper layer $a-1$ and lower layer a, the dimension of which is $n^{a-1}\times n^a$

$\boldsymbol{B}^a_{1\times n^a}$: Bias matrix(vector) between upper layer $a-1$ and lower layer$a$, the dimension of which is $1\times n^a$

$\boldsymbol{Z}^a_{m\times n^a}$: Weighted input for layer a, the dimension of which is $m\times n^a$

$\boldsymbol{A}^a_{m\times n^a}$: Output matrix of layer $a-1$, or, input matrix of layer a

$\sigma^a(\boldsymbol{Z})$: Activation function of layer a mapping matrix $\boldsymbol{Z}$ element-wise

E: Error calculated by loss function

$\frac{\partial Y}{\partial X}$: Differential of variable Y on variable X

$$\boldsymbol{A}^{L+1}_{m\times t} = \sigma^L((\ldots \boxed{\sigma^2(\boxed{\sigma^1(\boldsymbol{X}_{m\times n}\cdot \boldsymbol{W}^1_{n\times n^1}+\boldsymbol{B}^1_{1\times n^1})}\,\boldsymbol{W}^2_{n^1\times n^2}+\boldsymbol{B}^2_{1\times n^2})}\ldots)\cdots \boldsymbol{W}^L_{n^{L-1}\times t}+\boldsymbol{B}^L_{1\times t})$$

Error:

$$E = \frac{\sum (\boldsymbol{A}_{m\times t}^{L+1} - \boldsymbol{Y}_{m\times t}) \odot (\boldsymbol{A}_{m\times t}^{L+1} - \boldsymbol{Y}_{m\times t})}{2m \times t}$$

$\odot$ is Hadamard production and $\sum \boldsymbol{Q}$ stands summing up all elements in $\boldsymbol{Q}$.

$\boldsymbol{X}_{m\times n}$: Input matrix with m samples and n features

$\boldsymbol{Y}_{m\times t}$: Target matrix with m samples and t predicted output

$\boldsymbol{Q}^T$: Transpose of matrix $\boldsymbol{Q}$

L: Output layer

n^a: number of nodes in layer a

$\boldsymbol{W}^a_{n^{a-1}\times n^a}$: Weight matrix between upper layer $a-1$ and lower layer a, the dimension of which is $n^{a-1} \times n^a$

$\boldsymbol{B}^a_{1\times n^a}$: Bias matrix(vector) between upper layer $a-1$ and lower layer$a$, the dimension of which is $1 \times n^a$

$\boldsymbol{Z}^a_{m\times n^a}$: Weighted input for layer a, the dimension of which is $m \times n^a$

$\boldsymbol{A}^a_{m\times n^a}$: Output matrix of layer $a-1$, or, input matrix of layer a

$\sigma^a(\boldsymbol{Z})$: Activation function of layer a mapping matrix $\boldsymbol{Z}$ element-wise

E: Error calculated by loss function

$\frac{\partial Y}{\partial X}$: Differential of variable Y on variable X

How to learn?
-- Find ***W*** and ***B*** that minimize error E

Ideally, find ***W*** that makes dE/d***W*** = 0 and ***B*** that makes dE/d***B*** = 0 --- However,

Almost impossible to do --- 1. Multilayer; 2. Not scalar...

Therefore, gradient descending is applied...

Gradient descending:

for each iteration:

W = ***W*** - learning_rate * dE/d***W***,

B = ***B*** - learning_rate * dE/d***B***

$$\boldsymbol{A}^{L+1}_{m\times t} = \sigma^{L}((\ldots\sigma^{2}(\sigma^{1}(\boldsymbol{X}_{m\times n}\cdot \boldsymbol{W}^{1}_{n\times n^{1}}+\boldsymbol{B}^{1}_{1\times n^{1}})\cdot \boldsymbol{W}^{2}_{n^{1}\times n^{2}}+\boldsymbol{B}^{2}_{1\times n^{2}})\ldots)\cdots \boldsymbol{W}^{L}_{n^{L-1}\times t}+\boldsymbol{B}^{L}_{1\times t})$$

$$E = \frac{\sum(\boldsymbol{A}^{L+1}_{m\times t} - \boldsymbol{Y}_{m\times t})\odot(\boldsymbol{A}^{L+1}_{m\times t} - \boldsymbol{Y}_{m\times t})}{2m\times t}$$

$\odot$ is Hadamard production and $\sum \boldsymbol{Q}$ stands summing up all elements in $\boldsymbol{Q}$.

$$E = f(\boldsymbol{A}^{L+1}) = f(g(\boldsymbol{Z}^{L})) = f(g(h(\boldsymbol{W}^{L})))$$

Derivative with Chain rule:

$$dW^L = \frac{\partial E}{\partial W} = \frac{\partial E}{\partial A^{L+1}} \times \frac{\partial A^{L+1}}{\partial Z^L} \times \frac{\partial Z^L}{\partial W^L} = (A^{L-1}_{m \times n^L})^T \times \frac{A^{L+1}_{m \times t} - Y_{m \times t}}{m \times t} \odot \sigma'^L(Z^L)$$

$$dB^L = \frac{\partial E}{\partial B} = \frac{\partial E}{\partial A^{L+1}} \times \frac{\partial A^{L+1}}{\partial Z^L} \times \frac{\partial Z^L}{\partial B^L} = \frac{A^{L+1}_{m \times t} - Y_{m \times t}}{m \times t} \odot \sigma'^L(Z^L) \quad \text{x1}$$

$$\Delta^L_{m \times t} = dZ^L = \frac{\partial E}{\partial Z^L} = \frac{\partial E}{\partial A^{L+1}} \times \frac{\partial A^{L+1}}{\partial Z^L} \times = \frac{A^{L+1}_{m \times t} - Y_{m \times t}}{m \times t} \odot \sigma'^L(Z^L)$$

$$\Delta^l_{m \times n^l} = dZ^l = \frac{\partial E}{\partial Z^l} = \Delta^{l+1} \times \frac{\partial Z^{l+1}}{\partial A^{L+1}} \times \frac{\partial A^{L+1}}{\partial Z^L} = \Delta^{i+1} \cdot (W^{i+1}_{n^i \times n^{i+1}})^T \odot \sigma'^l(Z^l)$$

Derivation of E on weighted input

Error on Layer l

4 important equations in back propagation:

$$\Delta^{L}_{m\times t} = \frac{\boldsymbol{A}^{L+1}_{m\times t} - \boldsymbol{Y}_{m\times t}}{m \times t} \odot \sigma'^{L}(\boldsymbol{Z}^{L})$$

$$\Delta^{l}_{m\times n^{l}} = \Delta^{i+1} \cdot (\boldsymbol{W}^{i+1}_{n^{i}\times n^{i+1}})^{T} \odot \sigma'^{l}(\boldsymbol{Z}^{l})$$

$$d\boldsymbol{W}^{l} = (\boldsymbol{A}^{l-1})^{T} \times \Delta^{l}$$

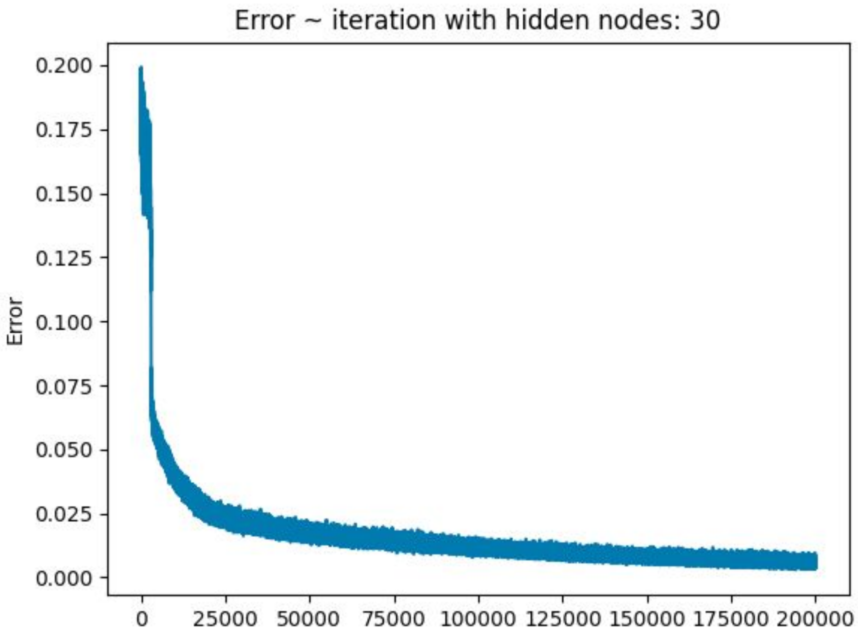
$$d\boldsymbol{B}^{l} = \Delta^{l}$$

Why back propagation?

Propagate error to each layer and update ***W*** and ***B*** layer by layer backward.

With implemented program,
y = (sin(x_1+x_2)+1)/2
was simulated with 30 hidden nodes and 10000 samples.

Algorithm can converge with low error (less than 0.01)!



https://github.com/Li-Ju666/FedML/tree/master/w1/C_implementation/Matrix-based_implementation

What to improve?

Activation function, loss function, structure, training techniques...

Activation functions:

Sigmoid:
range 0 ~ 1;
smooth

Relu:
range 0 ~ 1;
Computation efficient

tanh:
range -1 ~ 1;
smooth

Softmax:
range 0 ~ 1;
multi-class

Loss functions:

Mean square error:
For real number output

Binary crossentropy:
For bianry classification

Categorical Crossentropy:
For multiclass classification

Sparse Categorical Crossentropy:
Same as Category Crossentropy, but the output formats differ

Chosen datasets:

MNIST -- Image analysis (numbers classification)

Fashion MNIST -- Image analysis (Clothes classification)

CIFAR-10 -- Image analysis (Object detection)

Daily Minimum Temperatures in Melbourne --Time series

Thank you for your attention!