Layer\_dense:

Weights are set randomly (inputs, nurons avoids having to use transosition at every forward pass)

Biasas are all set to zero

The data passed to the next layer of the network is the dot product of inputs and weights, biases dosnt matter since they where set to zero

BACKWARD UPDATES PREVIOUS LAYER BASED ON CALCULATED OUTPUT ERROR

###########################################

Activation\_ReLU:

Forward and backwards methods

###########################################

Activation\_Softmax:

forward - same as chapter 4 with inputs as collums normalizing input and having the gradient as a row vector

Backward - creats an empty array with the same shape as the gradients that we’re receiving to apply the chain rule.

then iterate

sample-wise over pairs of the outputs and gradients, calculating the partial derivatives and using the chain rule to caluclate the final product.

The result is then stored as a row in the dinput array.

###########################################

LOSS:

returns mean value of all sample losses

###########################################

Loss\_CategoricalCrossentropy(Loss):

inherits the loss class

Forward - preforms error calculations returning the liklyhood of negative logrithem of correct\_confidences

Backward - changes numerical lables into one-hot vectors. and normalizes the gradient

###########################################

Activation\_Softmax\_Loss\_CategoricalCrossentropy

Forward - calculates loss

backward - takes one-hot values and returns descrete values, calculates thae gradient then normalizes it

###########################################