CSC3022F Assignment 3 Design Document

WNGJIA001 14 June 2021

Part 1: The XOR Problem

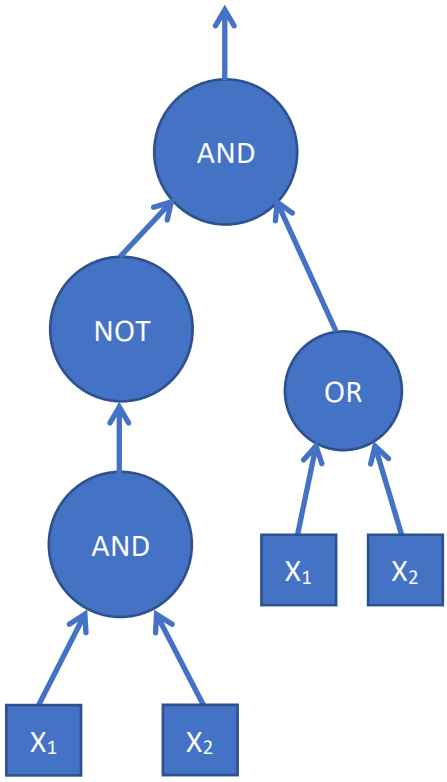
Topology

An XOR logic gate has the formula: $XOR(A,B) = A \cdot \bar{B} + B \cdot \bar{A}$, which can be simplified as:

$$XOR(A,B) = (A+B) \cdot (\overline{A \cdot B})$$

I.e., $XOR(A,B) = AND\left(OR(A,B), NOT\left(AND(A,B)\right)\right)$

The topology of an XOR gate is therefore represented as the figure below:



Training Data

Training data for each gate were generated separately. Each training data-point is made up of a uniformly distributed random floating point in the range $(-0.25 \sim 0.25)$ or $(0.75 \sim 1.25)$. The same data generation method was applied to the validation dataset.
It was assumed that the input data will only be 0 or 1 with a fluctuation of ± 0.25, therefore the abovementioned method was implemented.

Effects of Learning Rate

The learning rate didn't affect the accuracy of the network significantly, but a small learning rate will take longer to converge. The learning rates for all gates in the final network were all 0.2 and the network converges with only a few iterations for most of the cases.

Part 2: Image Classification

Topology

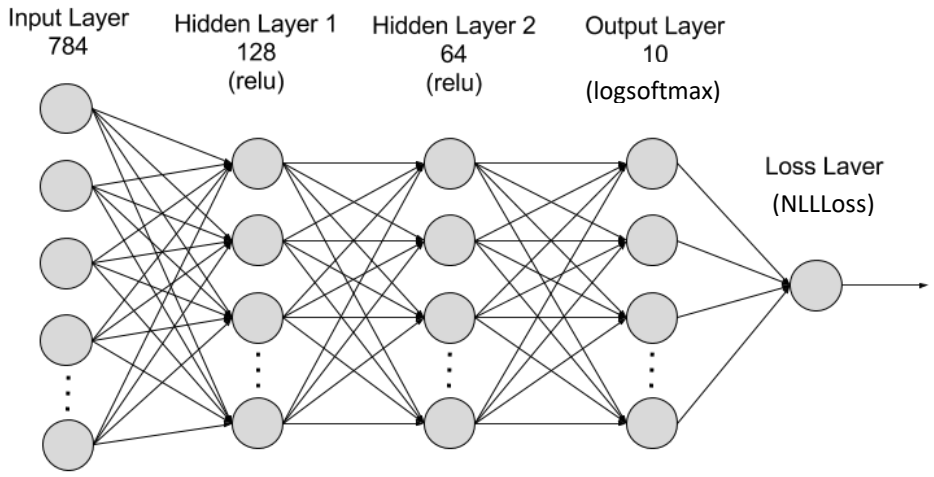
- Layers and Neurons:

The input layer has $28 \times 28 = 784$ node, each accounting for a single pixel (feature) of the handwritten digit image. The output layer will have 10 nodes, each representing a digit from the output range $[0 \sim 9]$.
Since the image classification problem cannot be represented as a linear separable function, hidden layers will be needed. According to Jeff Heaton, 2 hidden layers can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy (Heaton, 2017) and thus there will be 2 hidden layers for this network. There isn't a fixed formula to compute the number of neurons in a hidden layer, but Heaton mentioned that using too few neurons in the hidden layers will result in underfitting and using too many neurons in the hidden layers can result in several problems such as overfitting and increasing the time it takes to train the network. The number of neurons in each hidden layer was then decided to be 128 and 64 respectively, they would be subject to trial-and-error change if after the training the network didn't show satisfactory accuracy.

- Activation functions:

Every layer is linearly connected to the next layer, the neurons in the hidden layers will have an activation of rectified linear unit (ReLU) which is a simple function that allows positive values to pass through, whereas negative values are modified to zero. ReLU was chosen because it behaves close to a linear unit and rejects any number less than 0. Therefore the gradient is large and never saturates in the positive region and converges much faster.
The output layer has an activation function of LogSoftmax. This is simply the logarithm of SoftMax which converts the output from a linear layer into a categorical probability distribution. The LogSoftmax was chosen because it has the effect of heavily penalizing the model when it fails to predict a correct class when used for a classifier case, thus improved performance and gradient optimization.

Therefore, the topology of the neural network can be represented as in figure below:



Preprocessing

- Transform:

Before loading the dataset, a transformation was defined such that when applied, all the images will have the same dimensions and properties.
transforms.ToTensor() converts the image into a Torch Tensor which is a format of numbers that can be understood by the system.
transforms.Normalize() normalizes the tensor with a mean and standard deviation specified in the declaration.

- Dataset:

torch.utils.data.Dataset was used to store the sample images and their corresponding labels for better readability and modularity.

- Dataloader:

Since the dataset is large, torch.utils.data.DataLoader was used to load the dataset for the network. The dataloader will have a batch size which denotes the number of samples contained in each generated batch. At each epoch, dataloader will shuffle the images.

Loss function

If the output activation was using Softmax then CrossEntropyLoss would be used as the loss function since it measures the difference between two probability distributions for a given random variable. Cross-entropy loss increases as the predicted probability diverges from the actual label, therefore a high loss value would result from an incorrect prediction and thus speeds up the weight adjustments. CrossEntropyLoss is a combination of LogSoftmax and NLLLoss in one single class. Since the output activation was actually using LogSoftmax, the negative log likelihood loss (NLLLoss) should be used as the loss function in this network.

Optimizer

The optimizer was used to update the weights based on the computed gradients to reduce model error in each training step.
torch.optim.SGD which implements stochastic gradient descent was used as the optimizer for this network. With SGD, the weights are updated after looping via each training sample.

Training and Validation

Two functions were defined for training and validating the model, namely train_loop() and test_loop().
In train_loop(), the images in training set are divided into batches according to the pre-specified batch size. For each batch, the images are predicted, and the model parameters will be updated at the end of each batch according to the loss function and optimizer.
In test_loop(), each image in validation set was predicted and the output was compared to the label. If the prediction was correct, the correct count will increment otherwise continue to the next image. At the end, the total loss and accuracy of the epoch will be determined and displayed.

Other Miscellaneous Details

SGD is a very basic optimizer and is seldom used now, torch.optim packages contains many other options of more efficient optimizer.

References

Heaton, J., 2017. The Number of Hidden Layers. [online] Heaton Research. Available at: <https://www.heatonresearch.com/2017/06/01/hidden-layers.html> [Accessed 14 June 2021].