Лабораторная работа №2. Реализация глубокой нейронной сети

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

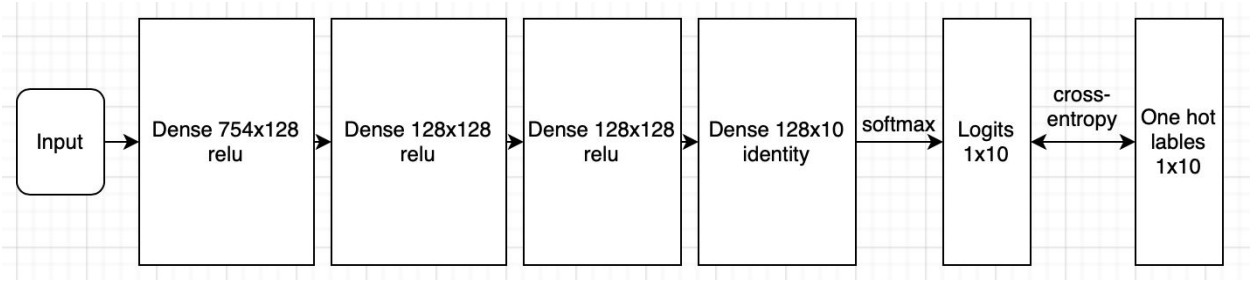
Ход работы:

Для реализации лабораторной работы используется библиотека **Swift for TensorFlow** https://github.com/tensorflow/swift

Исходный код: https://github.com/Stunba/MachineLearning2/tree/master/lab2/ML-lab

Интерактивная версия в коллаб:
https://colab.research.google.com/drive/1pGYTzPjMl0pbSSLpFROm3KDoJCSqqkhM

Структура полносвязной сети



Код модели сети:

```
public struct MLP: Layer {
   public var blocks: [Dense<Float>] = []
   init(dims: [Int], sigmoidLastLayer: Bool = false) {
       for i in 0..<(dims.count-1) {
           if sigmoidLastLayer && i == dims.count - 2 {
                 blocks.append(Dense(inputSize: dims[i], outputSize: dims[i+1],
activation: sigmoid))
           } else {
                 blocks.append(Dense(inputSize: dims[i], outputSize: dims[i+1],
activation: relu))
           }
       }
   }

   @differentiable
   public func callAsFunction(_ input: Tensor<Float>) -> Tensor<Float> {
       let blocksReduced = blocks.differentiableReduce(input) { last, layer in
```

```
            layer(last)
        }
        return blocksReduced
    }
}

extension MLP {
    init(numberOfFeatures: Int,
         numberOfLabels: Int,
         numberOfLayers: Int,
         numberOfUnits: Int,
         sigmoidLastLayer: Bool = true) {

        let insides = [Int](repeating: numberOfUnits, count: numberOfLayers)
            self.init(dims: [numberOfFeatures] + insides + [numberOfLabels] ,
sigmoidLastLayer: sigmoidLastLayer)
    }
}
```

Реализуем обучение модели, модель и оптимизатор идут как параметры:

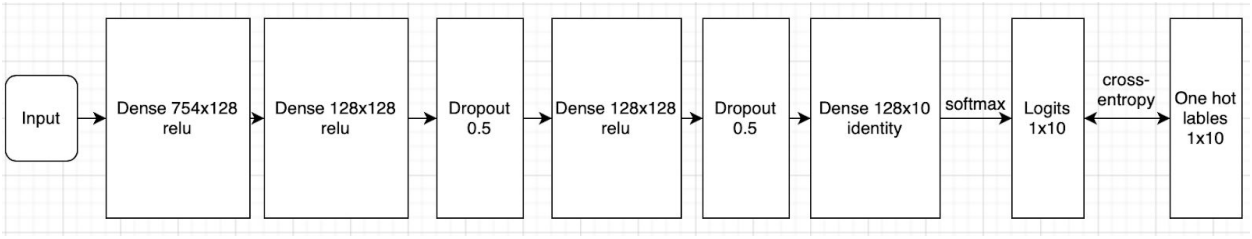
```
struct NNClassifier<Model: Layer, Optimizer: TensorFlow.Optimizer>
where Optimizer.Model == Model,
Model.Input == Tensor<Float>,
Model.Output == Tensor<Float> {
    struct Statistics {
        var correctGuessCount: Int = 0
        var totalGuessCount: Int = 0
        var totalLoss: Float = 0
        var batches: Int = 0
    }

    let epochCount: Int
    var model: Model
    var optimizer: Optimizer

    init(modelCreator: () -> Model,
         optimazerCreator: (Model) -> Optimizer,
         epochCount: Int = 100) {

        self.epochCount = epochCount
        self.model = modelCreator()
        self.optimizer = optimazerCreator(self.model)
    }
```

```
    mutating func fit(dataset: NotMNISTDataset) {
        for epoch in 1...epochCount {
            var trainStats = Statistics()
            var testStats = Statistics()

            Context.local.learningPhase = .training
            for batch in dataset.training.sequenced() {
                // Compute the gradient with respect to the model.
                    let ∇model = TensorFlow.gradient(at: model) { classifier ->
Tensor<Float> in
                    let ŷ = classifier(batch.features)
                         let correctPredictions = ŷ.argmax(squeezingAxis: 1) .==
batch.labels
                    trainStats.correctGuessCount += Int(
                        Tensor<Int32>(correctPredictions).sum().scalarized()
                    )
                    trainStats.totalGuessCount += batch.features.shape[0]
                              let loss = softmaxCrossEntropy(logits: ŷ, labels:
batch.labels)
                    trainStats.totalLoss += loss.scalarized()
                    trainStats.batches += 1
                    return loss
                }
                      // Update the model's differentiable variables along the
gradient vector.
                optimizer.update(&model, along: ∇model)
            }

            Context.local.learningPhase = .inference
            for batch in dataset.test.sequenced() {
                // Compute loss on test set
                let ŷ = model(batch.features)
                        let correctPredictions = ŷ.argmax(squeezingAxis: 1) .==
batch.labels
                                                    testStats.correctGuessCount   +=
Int(Tensor<Int32>(correctPredictions).sum().scalarized())
                testStats.totalGuessCount += batch.features.shape[0]
                let loss = softmaxCrossEntropy(logits: ŷ, labels: batch.labels)
                testStats.totalLoss += loss.scalarized()
                testStats.batches += 1
            }
```

```
                    let trainAccuracy = Float(trainStats.correctGuessCount) /
Float(trainStats.totalGuessCount)
                    let testAccuracy = Float(testStats.correctGuessCount) /
Float(testStats.totalGuessCount)
            print(
                """
                [Epoch \(epoch)] \
                                        Training Loss: \(trainStats.totalLoss /
Float(trainStats.batches)), \
                                                            Training Accuracy:
\(trainStats.correctGuessCount)/\(trainStats.totalGuessCount) \
                (\(trainAccuracy)), \
                Test Loss: \(testStats.totalLoss / Float(testStats.batches)), \
                                                                Test Accuracy:
\(testStats.correctGuessCount)/\(testStats.totalGuessCount) \
                (\(testAccuracy))
                """)
        }
    }

    func predict(features: Tensor<Float>) -> Tensor<Int32> {
        model(features).argmax(squeezingAxis: 1)
    }

    func accuracy(features: Tensor<Float>, truths: Tensor<Int32>) -> Float {
                                Tensor<Float>(predict(features: features) .==
truths).mean().scalarized()
    }
}

var classifier =
NNClassifier<MLP, SGD<MLP>>.defaultClassifier(
numberOfFeatures: dataset.numberOfFeatures, numberOfLabels: 10)

classifier.fit(dataset: dataset)
```

Используя SGD получили точность классификации 93.4%. По сравнению с логистической регрессией точность увеличилась на 10%.

Структура полносвязной сети с регуляризацией методом сброса нейронов



Код модели:

```
struct DropoutModel: Layer {
   var dense1: Dense<Float>
   var dense2: Dense<Float>
   var dropout1 = Dropout<Float>(probability: 0.5)
   var dense3: Dense<Float>
   var dropout2 = Dropout<Float>(probability: 0.5)
   var dense4: Dense<Float>

   init(numberOfFeatures: Int,
        numberOfLabels: Int,
        numberOfUnits: Int) {

           dense1 = Dense(inputSize: numberOfFeatures, outputSize:
numberOfUnits, activation: relu)
            dense2 = Dense(inputSize: numberOfUnits, outputSize:
numberOfUnits, activation: relu)
            dense3 = Dense(inputSize: numberOfUnits, outputSize:
numberOfUnits, activation: relu)
            dense4 = Dense(inputSize: numberOfUnits, outputSize:
numberOfLabels, activation: softmax)
   }

   @differentiable
        public func callAsFunction(_ input: Tensor<Float>) ->
Tensor<Float> {
         input.sequenced(through: dense1, dense2, dropout1, dense3,
dropout2, dense4)
   }
}
```

```
var dropoutClf =
NNClassifier.defaultDropout(numberOfFeatures:
dataset.numberOfFeatures, numberOfLabels: 10)

dropoutClf.fit(dataset: dataset)
```

Используя регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением и SGD получили точность 94.5%.

```
var adaptiveClf = NNClassifier.dropoutWithAda(numberOfFeatures:
dataset.numberOfFeatures, numberOfLabels: 10)
adaptiveClf.fit(dataset: dataset)
```

Добавим динамически изменяемую скоростью обучения (learning rate) к предыдущей модели получили точность 95.7%.

Вывод:

В данной лабораторной работе был построена и обучена глубокая нейроная сеть используя набор данных notMNIST использую метод сброса нейронов для регуляризации и адаптивная скорость обучения.