S4TF save/export

Not a priority but a nice feature for the edge

Inferencing

Personalization

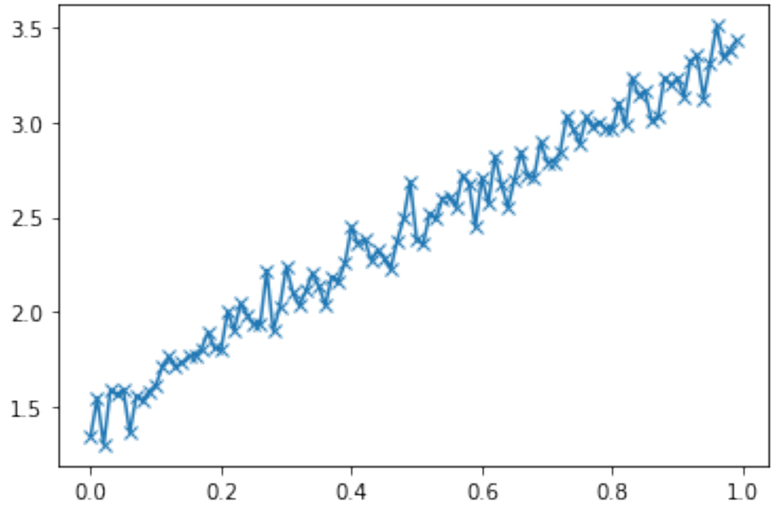
Federated Learning

ProtoBuf / FlatBuf file format

Runtime

S4TF Trivial data, model, hack

```
let SAMPLE_SIZE = 100

let a: Float = 2.0
let b: Float = 1.5
let x = Tensor<Float>(rangeFrom: 0, to: 1, stride: 1.0 / Float(SAMPLE_SIZE))
let noise = (Tensor<Float>(randomNormal: [SAMPLE_SIZE]) - 0.5) * 0.1
let y = (a * x + b) + noise
```




```
struct LinearRegression: Layer {
    var layer1 = Dense<Float>(inputSize: 1, outputSize: 1, activation: identity)

    @differentiable
    func callAsFunction(_ input: Tensor<Float>) -> Tensor<Float> {
        return layer1(input)
    }
}
```

```
var regression = LinearRegression()
let optimizer = SGD(for: regression, learningRate: 0.03)
Context.local.learningPhase = .training

for _ in 0..<100 { //1000
    let ∇model = regression.gradient { r -> Tensor<Float> in
        let ŷ = r(X)
        let loss = meanSquaredError(predicted: ŷ, expected: Y)
        print("Loss: \(loss)")
        return loss
    }
    optimizer.update(&regression, along: ∇model)
}
```

Hack Swift CoreML ProtoBuf format

Apple provide **Python CoreMLTools** package and Swift API for iOS/macOS/.. for model usage but no Swift API for CoreML model creation

1. Install ProtoBuf Compiler - https://github.com/protocolbuffers/protobuf
2. Install Swift ProtoBuf Plugin - https://github.com/apple/swift-protobuf
3. Download CoreML ProtoBuf source file - https://github.com/apple/coremltools/tree/master/mlmodel/format
4. Compile and generate Swift CoreML data structures - protoc --swift_out=[PATH TO FOLDER FOR GENERATED SWIFT FILES] [PATH TO COREMLTOOS FOLDER]/mlmodel/format/*.proto

Swift CoreML ProtoBuf data

protoc --decode_raw < **model.mlmodel**

```
1: 4
2 {
 1 {
  1: "dense_input"
  3 {
   5 {
    1: "\001"
    2: 65600
   }
  }
 }
 10 {
  1: "output"
  3 {
   5 {
    1: "\001"
    2: 65600
   }
  }
 }
 50 {
  1: "dense_input"
  3 {
   5 {
    1: "\001"
    2: 65600
```

Model.proto

```
syntax = “proto3”;
message Model {
    int32 specificationVersion = 1;
    ModelDescription description = 2;
    bool isUpdatable = 10;
    oneof Type {
        ...
        NeuralNetwork neuralNetwork = 500;
        ...
    }
    ...
}

message ModelDescription {
    repeated FeatureDescription input = 1;
    repeated FeatureDescription output = 10;
    string predictedFeatureName = 11;
    string predictedProbabilitiesName = 12;
    repeated FeatureDescription trainingInput = 50;
    Metadata metadata = 100;
}

message FeatureDescription {
    string name = 1;
    string shortDescription = 2;
    FeatureType type = 3;
}

message NeuralNetwork {
    repeated NeuralNetworkLayer layers = 1;
    repeated NeuralNetworkPreprocessing preprocessing = 2;
    NeuralNetworkMultiArrayShapeMapping
arrayInputShapeMapping = 5;
    NeuralNetworkImageShapeMapping imageInputShapeMapping =
6;
    NetworkUpdateParameters updateParams = 10;
}

…
```

Model.pb.swift

```
import Foundation
import SwiftProtobuf

struct CoreML_Specification_Model {
 var specificationVersion: Int32 {
   get {return _storage._specificationVersion}
   set {_uniqueStorage()._specificationVersion = newValue}
 }
 var description_p: CoreML_Specification_ModelDescription {
   get {return _storage._description_p ?? CoreML_Specification_ModelDescription()}
   set {_uniqueStorage()._description_p = newValue}
 }
 var hasDescription_p: Bool {return _storage._description_p != nil}
 var isUpdatable: Bool {
   get {return _storage._isUpdatable}
   set {_uniqueStorage()._isUpdatable = newValue}
 }
 var type: OneOf_Type? {
   get {return _storage._type}
   set {_uniqueStorage()._type = newValue}
 }
 ...
 /// generic models start at 500
 var neuralNetwork: CoreML_Specification_NeuralNetwork {
   get {
     if case .neuralNetwork(let v)? = _storage._type {return v}
     return CoreML_Specification_NeuralNetwork()
   }
   set {_uniqueStorage()._type = .neuralNetwork(newValue)}
 }
 ...
}
...
```

Export S4TF to CoreML in Swift

```
struct LinearRegression: Layer {
  var layer1 = Dense<Float>(inputSize: 1, outputSize: 1, activation: identity)
  ...
}
var regression = LinearRegression()
...
let weight = Float(regression.layer1.weight[0][0])!
let bias = Float(regression.layer1.bias[0])!
```

```
let binaryModelData: Data = try coreModel.serializedData()

binaryModelData.write(to: URL(fileURLWithPath: "./s4tf_model_personalization.mlmodel"))
```

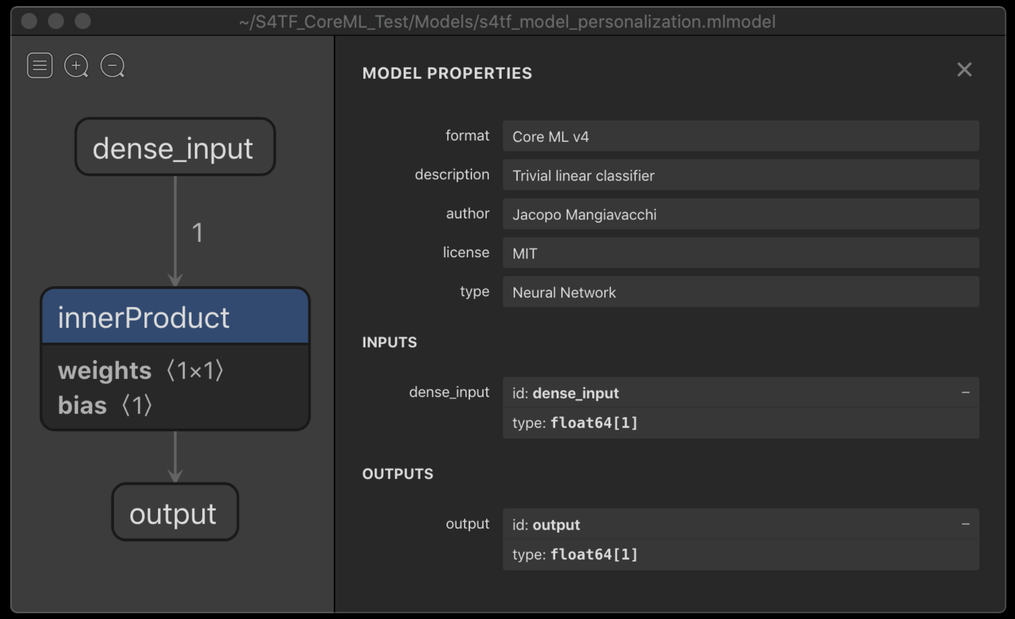
```
let coreModel = CoreML_Specification_Model.with {
    $0.specificationVersion = 4
    $0.description_p = CoreML_Specification_ModelDescription.with {
        $0.input = [CoreML_Specification_FeatureDescription.with {
            $0.name = "dense_input"
            $0.type = CoreML_Specification_FeatureType.with {
                $0.multiArrayType = CoreML_Specification_ArrayFeatureType.with {
                    $0.shape = [1]
                    $0.dataType = CoreML_Specification_ArrayFeatureType.ArrayDataType.double
                }
            }
        }]
        $0.output = [CoreML_Specification_FeatureDescription.with {
            $0.name = "output"
            $0.type = CoreML_Specification_FeatureType.with {
                $0.multiArrayType = CoreML_Specification_ArrayFeatureType.with {
                    $0.shape = [1]
                    $0.dataType = CoreML_Specification_ArrayFeatureType.ArrayDataType.double
                }
            }
        }]
        $0.trainingInput = [CoreML_Specification_FeatureDescription.with {
            $0.name = "dense_input"
            $0.type = CoreML_Specification_FeatureType.with {
                $0.multiArrayType = CoreML_Specification_ArrayFeatureType.with {
                    $0.shape = [1]
                    $0.dataType = CoreML_Specification_ArrayFeatureType.ArrayDataType.double
                }
            }
        }, CoreML_Specification_FeatureDescription.with {
            $0.name = "output_true"
            $0.type = CoreML_Specification_FeatureType.with {
                $0.multiArrayType = CoreML_Specification_ArrayFeatureType.with {
                    $0.shape = [1]
                    $0.dataType = CoreML_Specification_ArrayFeatureType.ArrayDataType.double
                }
            }

        }]
    }
```

```
    $0.isUpdatable = true
    $0.neuralNetwork = CoreML_Specification_NeuralNetwork.with {
        $0.layers = [CoreML_Specification_NeuralNetworkLayer.with {
            $0.name = "dense_1"
            $0.input = ["dense_input"]
            $0.output = ["output"]
            $0.isUpdatable = true
            $0.innerProduct = CoreML_Specification_InnerProductLayerParams.with {
                $0.inputChannels = 1
                $0.outputChannels = 1
                $0.hasBias_p = true
                $0.weights = CoreML_Specification_WeightParams.with {
                    $0.floatValue = [weight]
                    $0.isUpdatable = true
                }
                $0.bias = CoreML_Specification_WeightParams.with {
                    $0.floatValue = [bias]
                    $0.isUpdatable = true
                }
            }
        }]
        $0.updateParams = CoreML_Specification_NetworkUpdateParameters.with {
            $0.lossLayers = [CoreML_Specification_LossLayer.with {
                $0.name = "lossLayer"
                $0.meanSquaredErrorLossLayer = CoreML_Specification_MeanSquaredError
                    $0.input = "output"
                    $0.target = "output_true"
                }
            }]
```

```
        $0.optimizer = CoreML_Specification_Optimizer.with {
            $0.sgdOptimizer = CoreML_Specification_SGDOptimizer.with {
                $0.learningRate = CoreML_Specification_DoubleParameter.with {
                    $0.defaultValue = 0.03
                    $0.range = CoreML_Specification_DoubleRange.with {
                        $0.maxValue = 1.0
                    }
                }
                $0.miniBatchSize = CoreML_Specification_Int64Parameter.with {
                    $0.defaultValue = 1
                    $0.set = CoreML_Specification_Int64Set.with {
                        $0.values = [1]
                    }
                }
                $0.momentum = CoreML_Specification_DoubleParameter.with {
                    $0.defaultValue = 0
                    $0.range = CoreML_Specification_DoubleRange.with {
                        $0.maxValue = 1.0
                    }
                }
            }
        }
        $0.epochs = CoreML_Specification_Int64Parameter.with {
            $0.defaultValue = 100
            $0.set = CoreML_Specification_Int64Set.with {
                $0.values = [100]
            }
        }
        $0.shuffle = CoreML_Specification_BoolParameter.with {
            $0.defaultValue = true
        }
    }
}
```

CoreML Compile and Inference

Xcode has fantastic drag&drop integration of CoreML model into project with Swift wrapper code generation but models can also be loaded, compiled and used dynamically




```swift
func compileCoreML(path: String) -> (MLModel, URL) {
    let modelUrl = URL(fileURLWithPath: path)

    let compiledUrl = try! MLModel.compileModel(at: modelUrl)

    return try! (MLModel(contentsOf: compiledUrl), compiledUrl)
}


func inferenceCoreML(model: MLModel, x: Float) -> Float {
    let multiArr = try! MLMultiArray(shape: [1], dataType: .double)
    multiArr[0] = NSNumber(value: x)

    let inputValue = MLFeatureValue(multiArray: multiArr)

    let dataPointFeatures: [String: MLFeatureValue] = [inputName: "dense_input"]

    let provider = try! MLDictionaryFeatureProvider(dictionary: dataPointFeatures)

    let prediction = try! model.prediction(from: provider)

    return Float(prediction.featureValue(for: "output")!.multiArrayValue![0].doubleValue)
}


let (coreModel, compiledModelUrl) = compileCoreML(path: coreMLFilePath)
let prediction = inferenceCoreML(model: coreModel, x: 1.0)
```

CoreML Personalization / Training

Prepare Batch Data

```swift
func generateData(sampleSize: Int = 100) -> ([Float], [Float]) {
    let a: Float = 2.0
    let b: Float = 1.5
    var X = [Float]()
    var Y = [Float]()
    for i in 0..<sampleSize {
        let x: Float = Float(i) / Float(sampleSize)
        let noise: Float = (Float.random(in: 0..<1) - 0.5) * 0.1
        let y: Float = (a * x + b) + noise
        X.append(x)
        Y.append(y)
    }
    return (X, Y)
}

func prepareTrainingBatch() -> MLBatchProvider {
    var featureProviders = [MLFeatureProvider]()
    let inputName = "dense_input"
    let outputName = "output_true"
    let (X, Y) = generateData()
    for (x,y) in zip(X, Y) {
        let multiArr = try! MLMultiArray(shape: [1], dataType: .double)
        multiArr[0] = NSNumber(value: x)
        let inputValue = MLFeatureValue(multiArray: multiArr)
        multiArr[0] = NSNumber(value: y)
        let outputValue = MLFeatureValue(multiArray: multiArr)
        let dataPointFeatures: [String: MLFeatureValue] = [inputName: inputValue,
                                                           outputName: outputValue]
        if let provider = try? MLDictionaryFeatureProvider(dictionary: dataPointFeatures) {
            featureProviders.append(provider)
        }
    }
    return MLArrayBatchProvider(array: featureProviders)
}
```

Training

```swift
func train(url: URL) {
    let configuration = MLModelConfiguration()
    configuration.computeUnits = .all
    configuration.parameters = [.epochs : 100]
    let progressHandler = { (context: MLUpdateContext) in
        switch context.event {
        case .trainingBegin: ...
        case .miniBatchEnd: ...
        case .epochEnd: ...
       }
    }
    let completionHandler = { (context: MLUpdateContext) in
        guard context.task.state == .completed else { return }
        let trainLoss = context.metrics[.lossValue] as! Double
        let updatedModel = context.model
        let updatedModelURL = URL(fileURLWithPath: retrainedCoreMLFilePath)
        try! updatedModel.write(to: updatedModelURL)
    }
    let handlers = MLUpdateProgressHandlers(
                        forEvents: [.trainingBegin, .miniBatchEnd, .epochEnd],
                        progressHandler: progressHandler,
                        completionHandler: completionHandler)

    let updateTask = try! MLUpdateTask(forModelAt: url,
                                       trainingData: prepareTrainingBatch(),
                                       configuration: configuration,
                                       progressHandlers: handlers)
    updateTask.resume()
}
```

```swift
train(url: compiledModelUrl)

// Wait for completition of the asyncronous training task

let retrainedModel = try! MLModel(contentsOf: URL(fileURLWithPath: retrainedCoreMLFilePath))
let prediction = inferenceCoreML(model: retrainedModel, x: 1.0)
```

S4TF How Automate Model Export ?

- Extend Layer, Sequencial/sequenced(), DSL function builder ???
- What about Training parameters (Cost Functions, Optimizations, …) ??

+ Model Optimizations:
  - Other then Quantization and Pruning
  - i.e. Microsoft ONNX BERT condensed layers (17x inference acceleration)
  - i.e. CoreML Custom Layers and/or Custom Activation Functions

- https://github.com/JacopoMangiavacchi/S4TF_CoreML_Test
- https://medium.com/@JMangia/swift-loves-tensorflow-and-coreml-2a11da25d44