DEEP LEARNING WITH F#

AN EXPERIENCE REPORT

Faisal Waris
Data Scientist in Telecom
faisalwaris@yahoo.com

OVERVIEW

Why it all matters

Interactive programming experience

Machine Learning with F# experience

Classical Machine Learning

Deep Learning

Key Takeaways

WHY DOES IT MATTER: APP-EMBEDDED AI

Artificial Intelligence (AI/ML) is a practical reality

- (classical) Machine Learning / Deep Learning / Reinforcement Learning / Optimization / ..
- No longer just in the lab

AI/ML solves ***useful problems*** for business, government and other organizations

Deep Learning (DL) performs well on ***'cognitive tasks'*** (e.g. vision and language)

- Performance comparable to humans on some tasks
- Classical methods did not do as well

AI/ML is being ***embedded*** in (regular) applications

- This trend will continue

.Net / F# developers need to take note

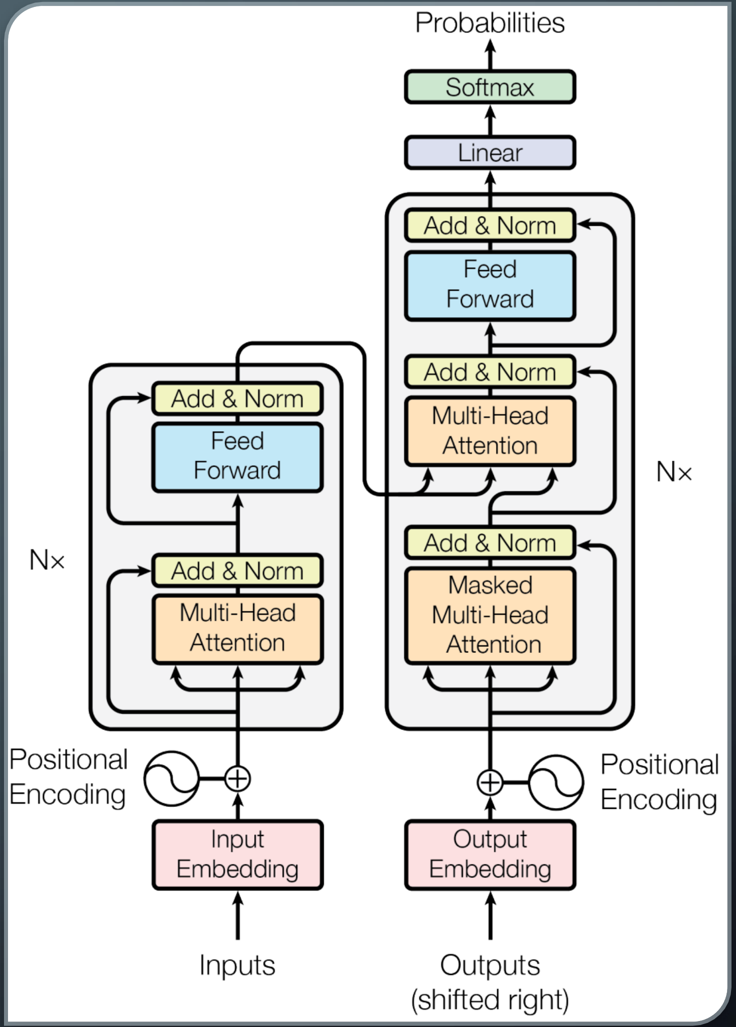
WHY DOES IT MATTER:
DL MODELING LANGUAGE

Computational Graph – Static vs Dynamic

Static: TensorFlow, CNTK

PyTorch, JAX use Dynamic Computation Graphs

Most new research is in PyTorch!

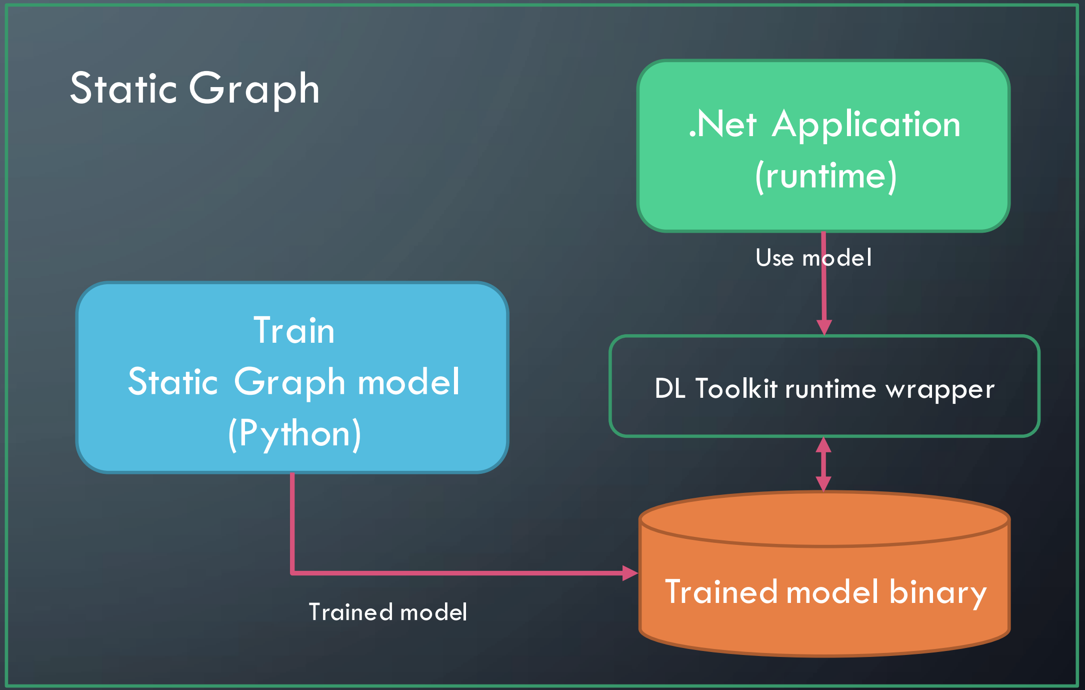


Computation Graph: Transformer Module

Source: “Attention is all you need”, Vaswani et al.

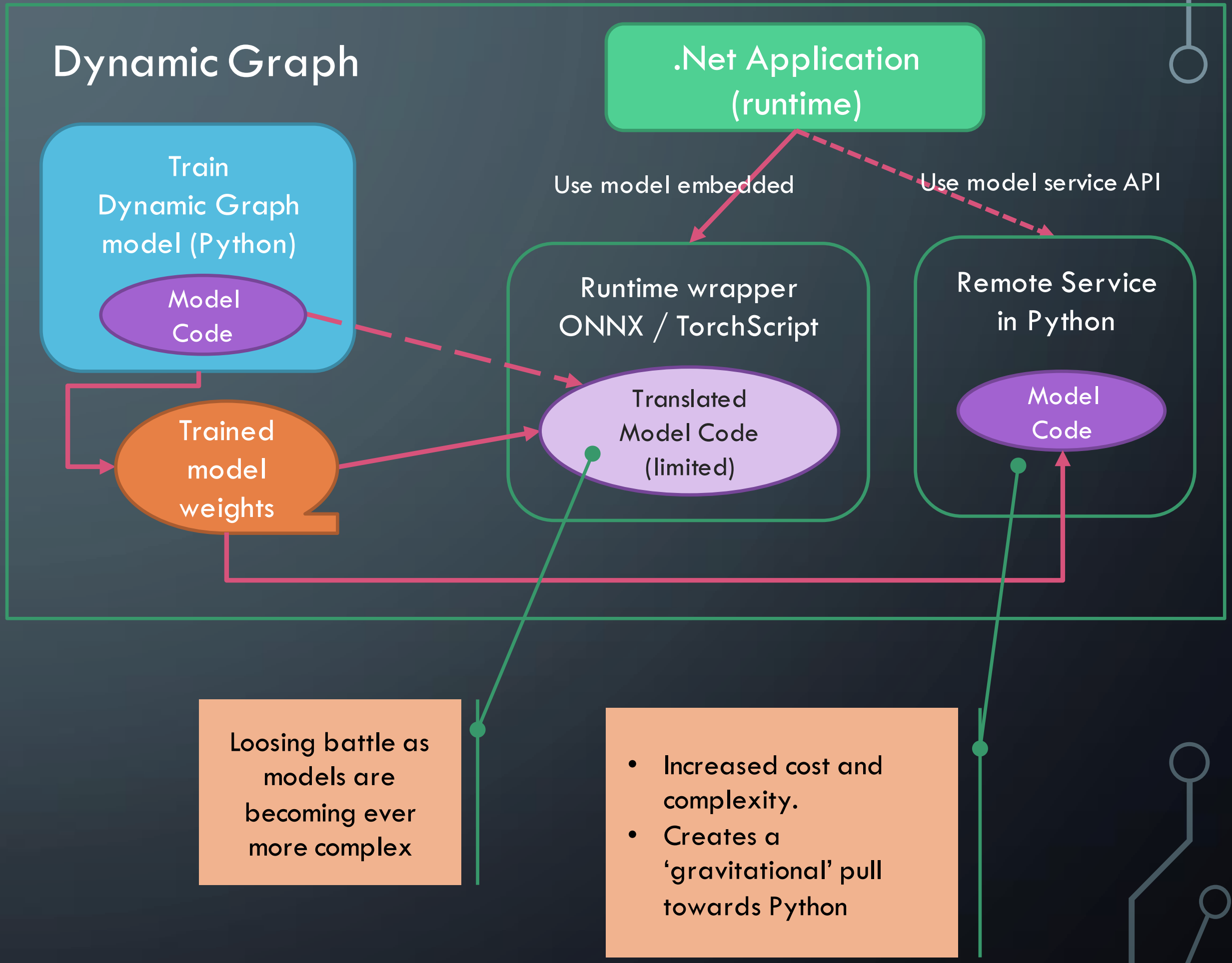
WHY DOES IT MATTER: DL MODELING LANGUAGE

- Static vs. Dynamic Implications
- Static Graphs
  - Trained static graphs can be compiled into stand alone modules
  - These can be called at run-time from any language
  - DL Model language (e.g., Python) is not required at runtime


Static Graph
.Net Application (runtime)
Use model
Train Static Graph model (Python)
DL Toolkit runtime wrapper
Trained model binary
Trained model

WHY DOES IT MATTER: DL MODELING LANGUAGE

- Static vs. Dynamic Implications
- Dynamic graphs
  - Interplay between DL toolkit (C++) and DL Model language
  - GPU processing interspersed with DL Model language code invocation
  - Affords great flexibility in model design; model can be dynamic
  - Lock-in effect – DL Model language (or equivalent) required at runtime


Dynamic Graph
.Net Application (runtime)
Train Dynamic Graph model (Python)
Model Code
Use model embedded
Use model service API
Runtime wrapper ONNX / TorchScript
Remote Service in Python
Translated Model Code (limited)
Model Code
Trained model weights

Loosing battle as models are becoming ever more complex

- Increased cost and complexity.
- Creates a ‘gravitational’ pull towards Python

WHY DOES IT MATTER: KEY TAKEAWAYS

Integration of AI/ML with regular apps

- F# is a great app-dev language!
- Embedding AI / ML functionality should be par for the course

DL model language lock-in

- Consumption of DL models is becoming harder from ‘other’ languages / ecosystem
- .Net ecosystem needs effective DL modeling capability to stay relevant over time

INTERACTIVE PROGRAMMING [IP]

Interactive programming capability is crucial for Data Science

It is not clear how to solve a problem up front

Need to explore - data and processing approaches

A combinatorial explosion of methods

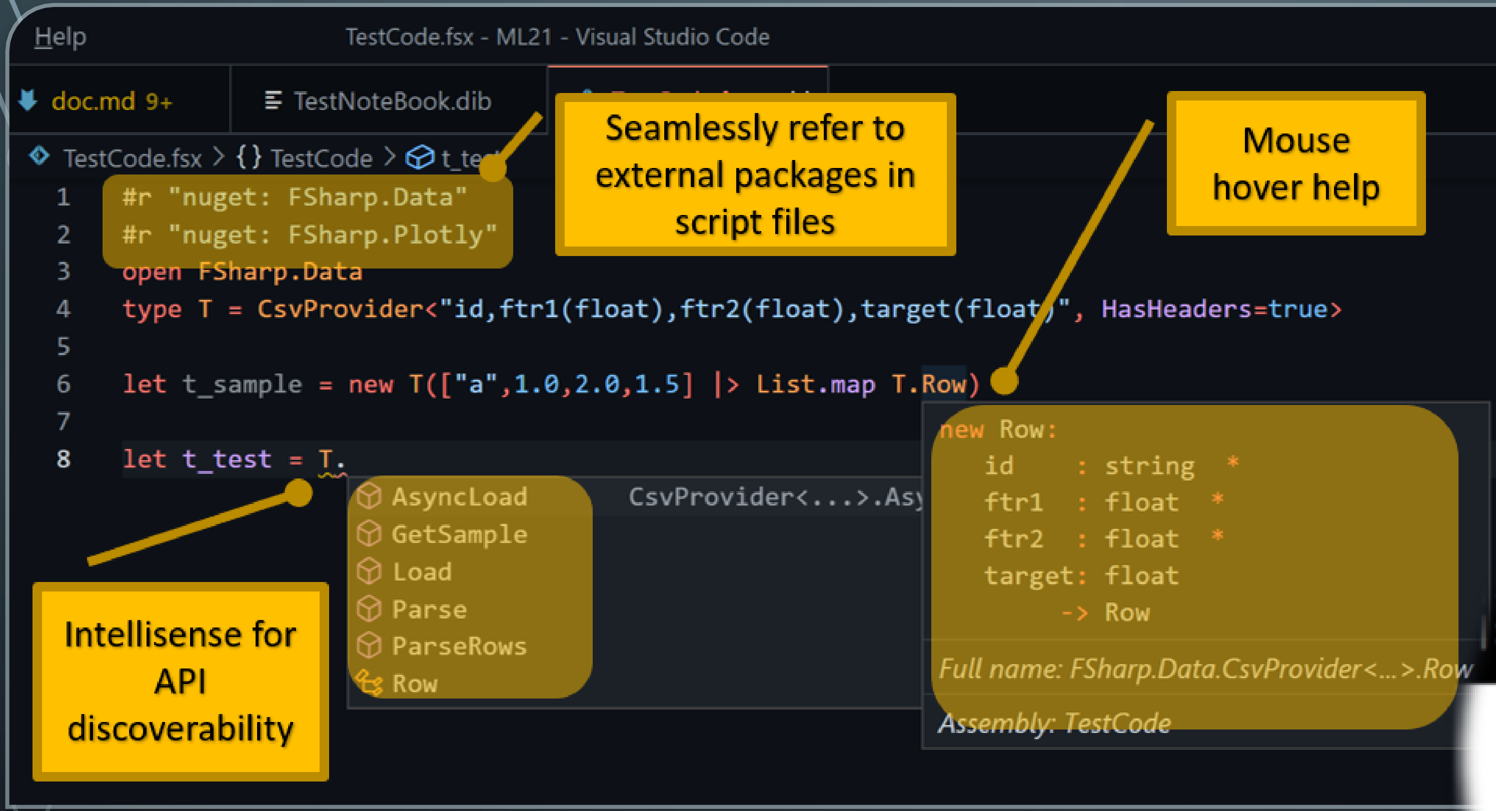
- N: Data pre-processing
- P: Feature engineering
- Q: Model structure / methodology
- N * P * Q possibilities

Need to conduct a myriad small experiments

Tedious and frustrating work

[IP] F# 'REPLS' OFFER RICH CAPABILITIES

- F# Script Support
  - Visual Studio
  - VS Code
  - Others (not used)
- Since 2010!


Help
TestCode.fsx - ML21 - Visual Studio Code
doc.md 9+
TestNoteBook.dib
TestCode.fsx > {} TestCode > t_te

```
#r "nuget: FSharp.Data"
#r "nuget: FSharp.Plotly"
open FSharp.Data
type T = CsvProvider<"id,ftr1(float),ftr2(float),target(float)", HasHeaders=true>

let t_sample = new T(["a",1.0,2.0,1.5] |> List.map T.Row)

let t_test = T.
```

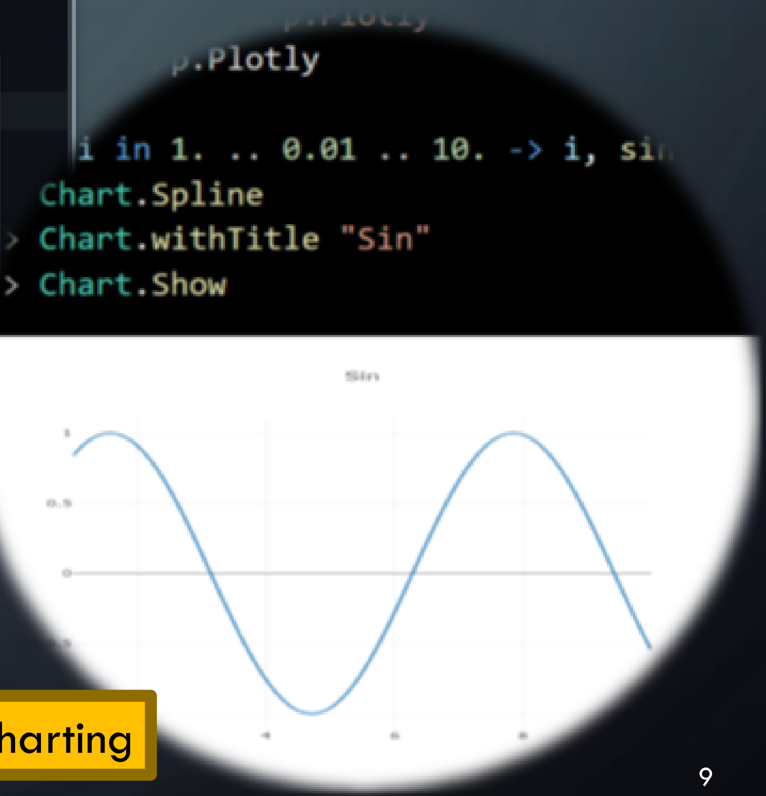
```
AsyncLoad       CsvProvider<...>.Asy
GetSample
Load
Parse
ParseRows
Row
```

```
new Row:
    id    : string  *
    ftr1  : float   *
    ftr2  : float   *
    target: float
         -> Row
Full name: FSharp.Data.CsvProvider<...>.Row
Assembly: TestCode
```

Seamlessly refer to external packages in script files

Mouse hover help

Intellisense for API discoverability

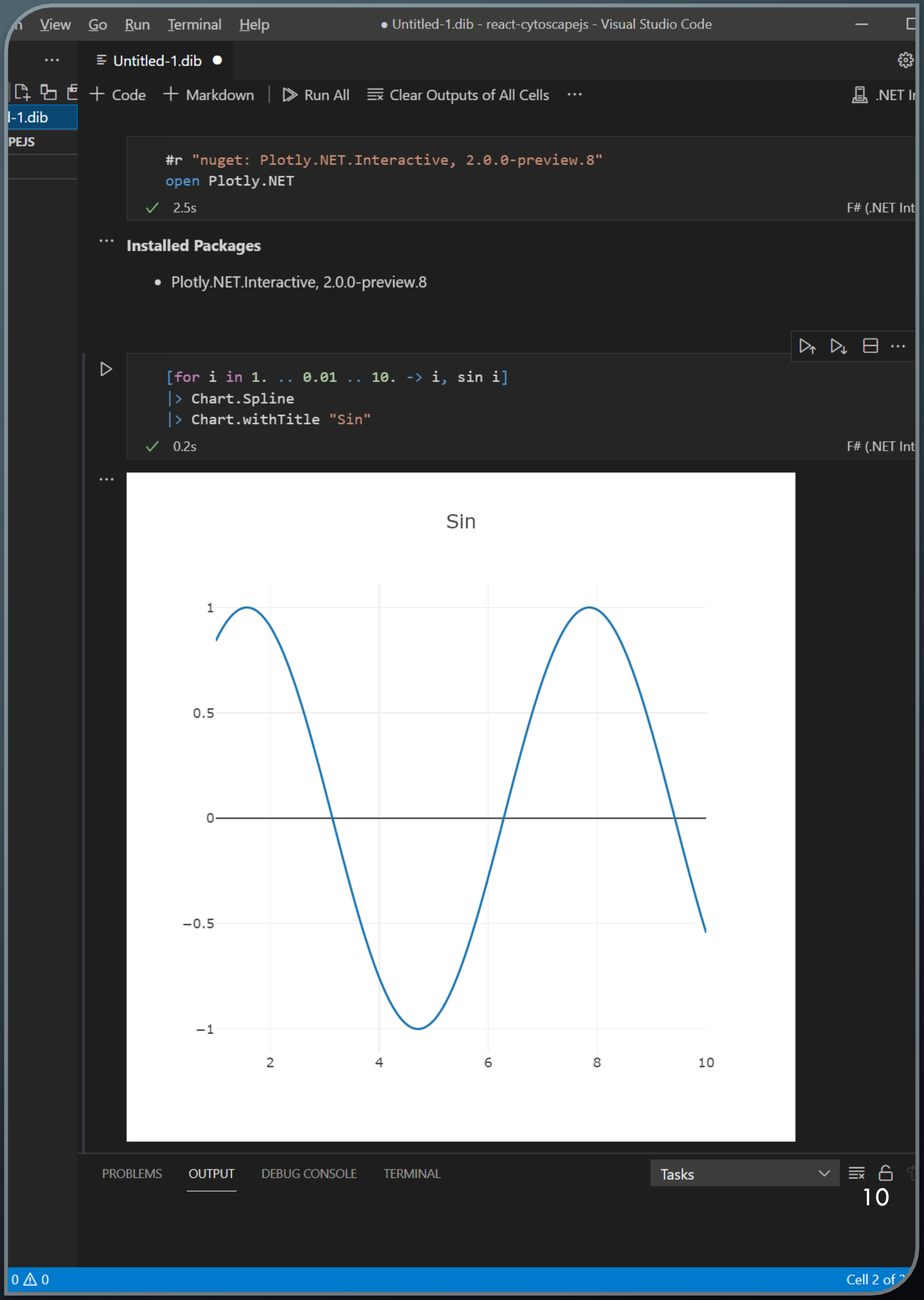


```
p.Plotly
i in 1. .. 0.01 .. 10. -> i, si
Chart.Spline
|> Chart.withTitle "Sin"
|> Chart.Show
```

Interactive Charting

[IP] NEW F# (AND C#) INTERACTIVE NOTEBOOKS

- Different interactive experience than F# REPL
- Similar to Jupyter Notebooks
- Charts and visualizations are in-lined
- Great for collaboration
- New and constantly improving



```
#r "nuget: Plotly.NET.Interactive, 2.0.0-preview.8"
open Plotly.NET
```

✓ 2.5s F# (.NET Int

Installed Packages

- Plotly.NET.Interactive, 2.0.0-preview.8

```
[for i in 1. .. 0.01 .. 10. -> i, sin i]
|> Chart.Spline
|> Chart.withTitle "Sin"
```

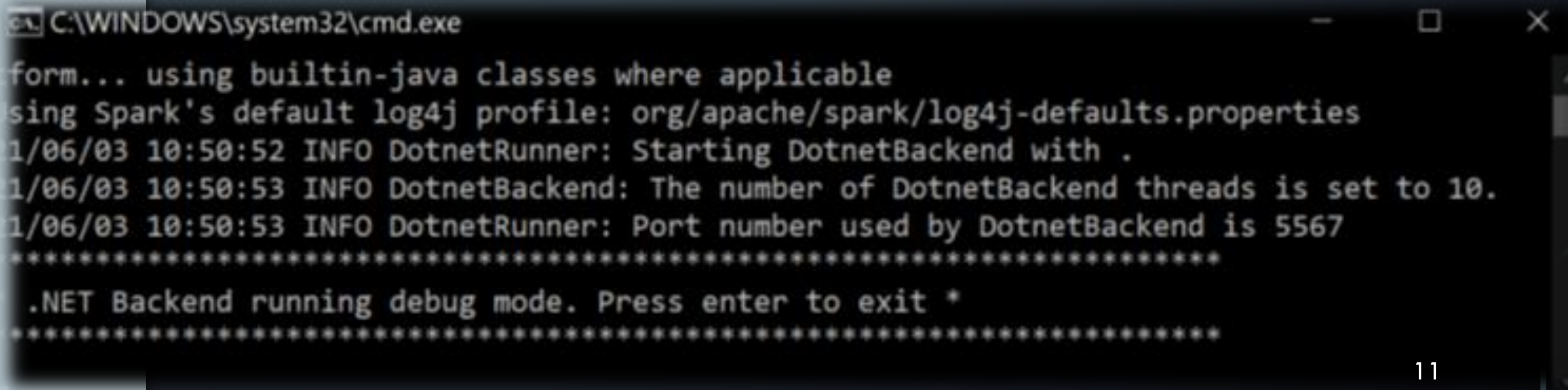
✓ 0.2s F# (.NET Int

[IP] INTERACTIVELY USE SPARK FROM F#

- Uses .Net for Spark
  - JVM JAR + .Net Packages
  - Spark <-> F# integration
- Process 100GB datasets on high-end laptop
- Spark runs as a separate process

- Works from F# REPL and Notebook
- User-defined functions work in Notebook only, for now
- Very large data:
  - Develop logic interactively then
  - Run on Spark cluster for scalability

```
open Microsoft.Spark.Sql
open type Microsoft.Spark.Sql.Functions
open Microsoft.Spark.Sql.Types
//create spark context
let spark =
SparkSession.Builder()
.AppName("test_spark")
.GetOrCreate()
//read text data
let df =
spark.Read()
.Option("header","true")
.Option("inferSchema","true")
.Option("delimiter","\t")
.Csv("...<data folder>...")
```



```
form... using builtin-java classes where applicable
sing Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
1/06/03 10:50:52 INFO DotnetRunner: Starting DotnetBackend with .
1/06/03 10:50:53 INFO DotnetBackend: The number of DotnetBackend threads is set to 10.
1/06/03 10:50:53 INFO DotnetRunner: Port number used by DotnetBackend is 5567
***********************************************************************
 .NET Backend running debug mode. Press enter to exit *
***********************************************************************
```

[IP] KEY TAKEAWAYS

F# well suited for DS tasks

- Top class F# REPLs
- Rich set of tools and capabilities

Notebooks

- Great for collaboration
- However, I prefer F# REPL when:
  - Code > 200 lines
  - Code in multiple files

Spark for large data handling

- Use interactively from F#
- Scale on clusters
- However, complex transforms need Scala
  - Worthwhile learning enough Scala to use Spark well

MACHINE / DEEP LEARNING WITH F# [ML]

Rich ecosystem of toolkits and packages to service most Data Science needs

TorchSharp

- .Net PyTorch binding
- Now under .Net Foundation

ML.Net

- classical machine learning

MathNet.Numerics

- General numerical routines

Infer.Net

- Bayesian modeling framework
- Now part of ML.Net

TensorFlow.Net

- TensorFlow binding
- TensorFlow Lite – Mobile device

OpenCV

- classical machine learning / computer vision
- mobile support

Plotly

- Visualization

Algomera

- Hierarchical clustering

Others: FSharp.Data; FSharp.Collections.ParallelSeq; KdTree (nearest neighbor); .Net for Spark; QuikGraph; FSharp.Control.AsyncSeq (stream processing); FsPickler; SimMetrics.Net; SQLProvider; Fable / SAFE Stack (UI); Z3 (constraint programming)

[ML] DEEP LEARNING WITH F# - PERSONAL HISTORY

Started Deep Learning with F# in 2015 with CNTK!

- FsCNTK – a functional wrapper over CNTK .Net API
- CNTK heavily used till 2019 (along with TensorFlow Python)
- CNTK deprecated in 2019

2019 Switched to TorchSharp / PyTorch

- Part of .Net Foundation (contributor)
- Wraps libtorch (C++)
- TorchSharp.Fun – created experimental functional F# wrapper over TorchSharp

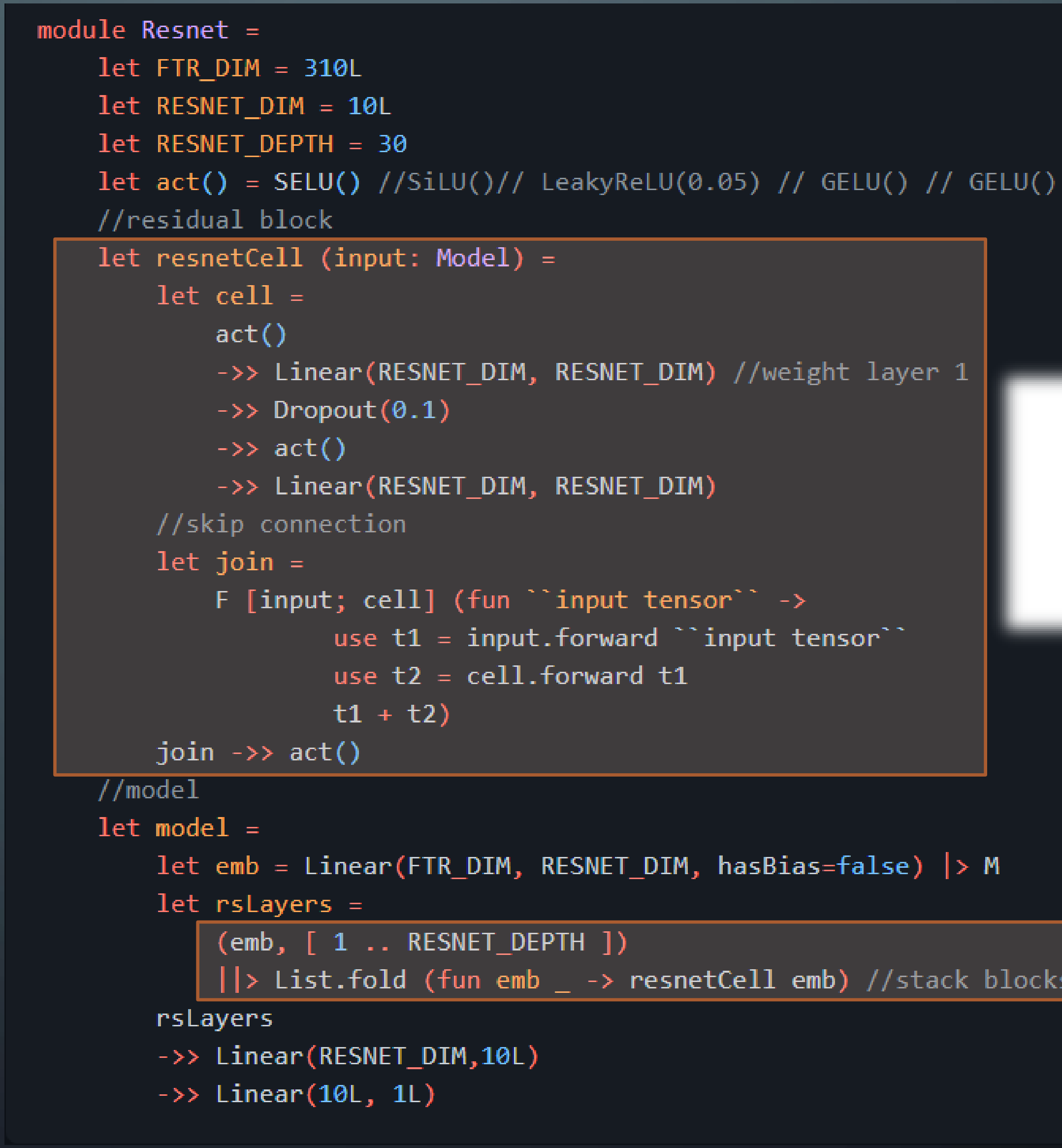
DiffSharp brief stint

- F# modeling library with advanced Automatic Differentiation
- Inspired TorchSharp.Fun

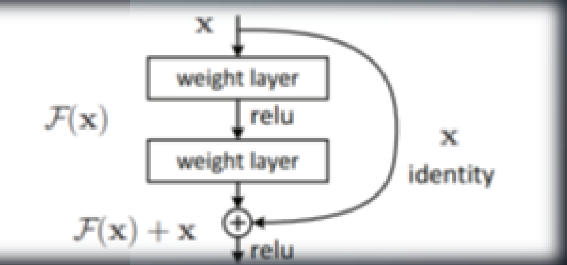
[ML] TorchSharp.Fun EXAMPLES

```
let model =
Linear(10L,5L)
->> Dropout(0.5)
->> Linear(5L,1L)
->> RelU()
```

- TorchSharp exposes an object-oriented API – similar to PyTorch Python API
- *TorchSharp.Fun* enables a functional, compositional style of modeling
  - ***Model structure is more apparent!***
- *TorchSharp.Fun* is experimental and not a stable API yet
- TorchSharp may expose a functional API in future



```
module Resnet =
    let FTR_DIM = 310L
    let RESNET_DIM = 10L
    let RESNET_DEPTH = 30
    let act() = SELU() //SiLU()// LeakyReLU(0.05) // GELU() // GELU()
    //residual block
    let resnetCell (input: Model) =
        let cell =
            act()
            ->> Linear(RESNET_DIM, RESNET_DIM) //weight layer 1
            ->> Dropout(0.1)
            ->> act()
            ->> Linear(RESNET_DIM, RESNET_DIM)
        //skip connection
        let join =
            F [input; cell] (fun ``input tensor`` ->
                    use t1 = input.forward ``input tensor``
                    use t2 = cell.forward t1
                    t1 + t2)
        join ->> act()
    //model
    let model =
        let emb = Linear(FTR_DIM, RESNET_DIM, hasBias=false) |> M
        let rsLayers =
            (emb, [ 1 .. RESNET_DEPTH ])
            ||> List.fold (fun emb _ -> resnetCell emb) //stack blocks
        rsLayers
        ->> Linear(RESNET_DIM,10L)
        ->> Linear(10L, 1L)
```



[ML] F# AND PYTHON

Most new models are developed and released in Python / PyTorch

- Translation to F# / TorchSharp required
- Translation sometimes not trivial
- Great way to deeply understand the model!

F# performance better on non-model code; same on model code

- F# pre-processing 7x faster than Python for Graph Convolutional Network model
- Pipeline (data pre-processing) code often much larger than model code
- F# type-safety a blessing for large code bases

[ML] KEY TAKEAWAYS

Sufficient set of libraries

- Deep Learning
- Classical ML
- Meet 95% of DS needs
- However, also need to use Python or R for capabilities not available in .Net

F# Great for Modeling

- Functional, compositional style ideal for DL models
- Type safety; performance
- However, most new research is in Python/PyTorch so new models will have to be translated into F# / .Net

F# FOR DEEP LEARNING EXPERIENCE: KEY TAKEAWAYS

- F# ecosystem offers:
  - Excellent tooling
  - Large of AI/ML libraries and packages
  - To meet most DS needs
- Use F# 90% of the time for my day Data Science job
  - Large F# DS application currently in production
  - More on the way
- F# is excellent DL modeling language
  - F# expressed models more succinct than OO models
  - Model structure more apparent
- However, need better adoption of DL with F# to keep the tooling ecosystem current and well-lubricated

THANK YOU

Q&A