HAW Hamburg
Informatik Master

Grundprojekt

Evaluation von TensorFlow Probability
für statistische Analysen

Bearbeiter:
Tom Schöner (2182801)

Betreuung:
Prof. Dr. Olaf Zukunft

17. April 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Abstract

Die auf Tensorflow basierende Bibliothek Tensorflow Probability[1] - fortan mit *TFP* abgekürzt - erweitert das Framework um eine probabilistische Komponente. Mittels einer breiten Masse an bereitgestellten Tools, wie statistischen Verteilungen, Sampling oder verschiedenster probabilistischer Erweiterungen für neronale Netze, können einfache bis hin zu komplexen Modellen erstellt werden. Berechnungen werden, wie man es aus Tensorflow gewohnt ist, durch *Dataflow Graphs*[2] abgebildet. Auf die verschiedenen Funktionsweisen und Schichten von TFP wird in Abschnitt 2 detaillierter eingegangen.

In dieser Evaluation soll die Bibliothek auf ihre Semantik und Pragmatik, Effektivität beim Erstellen von statistischen Modellen und Integration in das Framework Tensorflow untersucht werden. Das maschinelle Lernen mit Hilfe von neuronalen Netzen und deren Abstraktion durch Keras ist hierbei als Schwerpunkt anzusehen.

2 Tensorflow Probability Komponenten

Die Struktur von TFP lässt sich, wie aus der Dokumentation zu entnehmen ist[3], in die folgenden vier Schichten einteilen. Die Schichten bauen hierarchisch aufeinander auf, abstrahieren die unterliegenden Schichten aber nicht zwangsläufig. Möchte man beispielsweise durch *MCMC* in Schicht 3 Parameter seines probabilistischen Modells mittels Sampling ermitteln, sollten *Bijectors* aus Schicht 1 kein Fremdwort sein.

2.1 Schicht 0: Tensorflow

TFP ist nicht als eigenständige Komponente neben Tensorflow anzusehen, sondern als Bestandteil dessen. Die probabilistischen Berechnungen werden

[1] https://www.tensorflow.org/probability
[2] https://www.tensorflow.org/guide/graphs
[3] https://www.tensorflow.org/probability/overview

mit demselben Berechnungsmodel mittels Tensorflow Sessions oder im *Eager*-Modus für sofortige Berechnungen ausgeführt. Tensorflow wird von mehreren Programmiersprachen wie Python, JavaScript oder C++ unterstützt. Die Bibliothek TFP ist aktuell nur für die primär unterstützte Programmiersprache Python implementiert.

2.2 Schicht 1: Statistical Building Blocks

Als Fundament statistischer Modelle sind mehrere, in Python Module aufgeteilte, Klassen und Funktionen gegeben. Diese können in der API Dokumentation der TFP Website eingesehen werden. Ein Beispiel hierfür ist das Modul **fp.stats**. Unter **fp.stats** finden sich unter Anderem Funktionen für die Berechnung für Korrelationen, Quantilen oder Standardabweichungen. Diese Funktionen sind auf die Verwendung von Tensoren ausgelegt, lassen sich im Normalfall aber auch mit normalen n-dimensionalen Python Arrays oder Numpy Arrays aufrufen.

Verschiedenste, für probabilistische Modelle essentielle Verteilungen reihen sich unter dem Modul **tfp.distributions** in dieser Schicht ein: *Normal-, Bernoulli-, Exponential- oder Gammaverteilung*, um einige zu nennen. Generell erben Klassen für Verteilungen (wie etwa die Normalverteilung) von der Klasse **tfp.distributions.Distribution**[4]. Diese gemeinsame Schnittstelle vereinfacht die Benutzung und fordert eine Implementation für hilfreiche Methoden wie für logarithmische Wahrscheinlichkeit oder Sampling. Am Beispiel der Normalverteilung mit der Definition $X \sim \mathcal{N}(\mu, \sigma^2)$ lässt sich die Schnittstelle für Verteilungen verdeutlichen (siehe Listing 1). Visualisiert man die Verteilung auf Basis der Samples ergibt sich Abbildung 1.

```
1 normal_dist = tfd.Normal(name="N", loc=0., scale=1.)
2 # tfp.distributions.Normal("N/", batch_shape=(), event_shape=(),
      dtype=float32)
3
4 sample = normal_dist.sample(sample_shape=normal_sample_size)
```

[4] https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/Distribution

```
# <tf.Tensor: id=188, shape=(2500,), dtype=float32, numpy=array
    ([-0.45733708, -0.19126031, -0.33290815, ..., -1.1285563 ,
    -0.6958163 ,  0.552399  ], dtype=float32)>

normal_dist.mean()
# tf.Tensor(0.0, shape=(), dtype=float32)

normal_dist.prob(1.0)
# tf.Tensor(0.24197072, shape=(), dtype=float32)
```

Listing 1: Verwendung der Klasse tfp.distributions.Normal

Broadcasting, Batching und Shapes sorgen dafür, dass unabhängige Verteilungen als Batch in einer Entität gekapselt werden können. Um beispielsweise ein zweidimensionales Batch für die Normalverteilung aus Listing 1 zu erzeugen, kann als Erwartungswert μ durch *loc=[0., 10.]* übergeben werden. Da TFP hier ebenfalls broadcasting unterstützt müssen die folgenden Aufrufe nicht angepasst werden, nur das Ergebnis wird ebenfalls Zweidimensional sein. Broadcasting verhält sich analog zu dem in Numpy etablierten Konzept[5].

Ein weiterer Bestandteil von Schicht 1 sind *Bijectors*. Bijectors bilden eine Zahl aus $\mathbb{R}^n$ auf $\mathbb{R}^m$ ab oder jeweils einer Submenge dieser. Aufgrund der im Bijector hinterlegten bijektiven Funktion ist dieser Schritt umkehrbar, daher: $x = f^{-1}(f(x))$. Dies ist besonders bei der Transformierung von Stichproben aus Verteilungen nützlich und wird folglich für das Erstellen von Modellen (Abschnitt 2.3) und Berechnungen probabilistischer Interferenzen (Abschnitt 2.4) eingesetzt. TFP bietet bereits einige vorgefertigte Bijectors unter **tfp.bijectors**[6] an.

2.3 Schicht 2: Model Building

Edward2 / Probabilistic Layers with Keras / Trainable Distributions

In Abschnitt 6 wird die Verwendung und probabilistischen Layers mit Keras anhand eines Beispiels verdeutlicht.

[5] https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html
[6] https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors

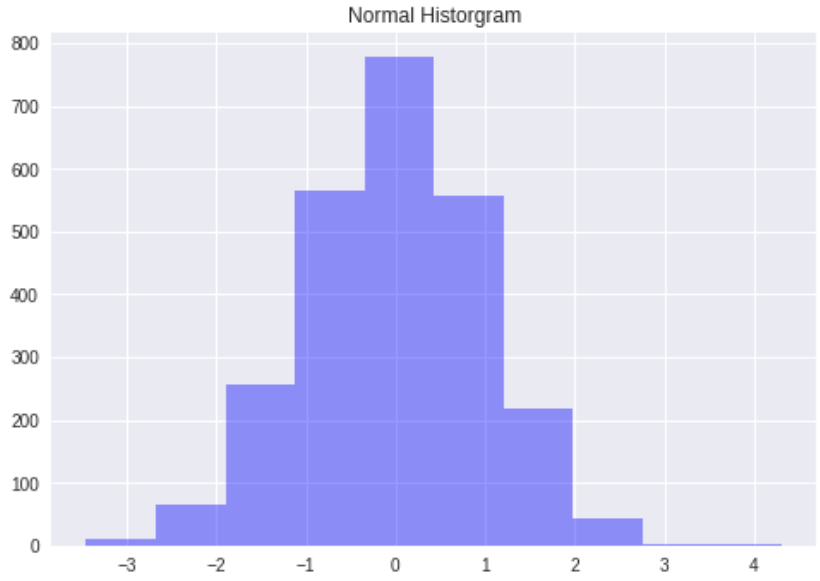

Abbildung 1: $X \sim \mathcal{N}(0, 1)$ *mit 2500 Samples*

2.4 Schicht 3: Probabilistic Inference

Als letzte Schicht enthält TFP Werkzeuge für probabilistische Inferenz. Dazu gehören *Marcov chain Monte Carlo (MCMC)* Algorithmen, *Variational Inference* Algorithmen und stochastische Optimierungsverfahren. Durch Sampling ermöglicht MCMC das Berechnen statistischer Modelle mit einer hohen Anzahl an Parametern. Eine analytische Herangehensweise ist bereits ab wenigen Parametern nicht mehr sinnvoll, da hierzu das Berechnen eines multidimensionalen Integrals erforderlich wäre. TFP bietet mehrere MCMC Algorithmen unter **tfp.mcmc** an.

Listing 2 zeigt einen Ausschnitt der Verwendung des Hamiltonian Monte Carlo Algorithmus. Die hier nicht gezeigte Auswertung würde in einer approximierten Verteilung des Parameters *tau* für das gegebene Modell *_data_model* resultieren. Ein praxisbezogenes und ausführlicheres Beispiel ist unter https://github.com/tom-schoener/ml-probability/blob/master/tfp-evaluation/notebooks/mcmc.ipynb einsehbar.

```
1 # Set the chain's start state.
```

```
initial_chain_state = [ 0.5 * tf.ones([], dtype=tf.float32, name
    ="init_tau") ]

def joint_log_prob(_data_model, tau):
  rv_tau = tfd.Uniform()
  rv_observation = tfd.Poisson(rate=rv_tau)

  return rv_tau.log_prob(tau) + tf.reduce_sum(rv_observation.
    log_prob(_data_model))

# setup the chain
[ tau ], kernel_results = tfp.mcmc.sample_chain(
  num_results=1000,
  num_burnin_steps=500,
  current_state=initial_chain_state,
  kernel=tfp.mcmc.TransformedTransitionKernel(
    inner_kernel=tfp.mcmc.HamiltonianMonteCarlo(
      target_log_prob_fn=lambda tau: joint_log_prob(_data_model,
    tau)),
    bijector=[ tfp.bijectors.Sigmoid() ] # Maps [0,1] to R
  )
)
```

Listing 2: Verwendung des Hamiltonian Monte Carlo Algorithmus (gekürzt)

Die in der API verwendeten Bezeichnungen decken sich hier größtenteils mit der in der Literatur verwendeten Sprache, wodurch die Übertragung von Konzepten aus unterschiedlichen Quellen deutlich vereinfacht wird.

3 Pragmatik und Semantik

4 Integration in Tensorflow

5 Anwendungsgebiete

6 Beispiel: Korrelation von Luftverschmutzung und Temperatur

Das Jupyter Notebook für das folgende Beispiel ist unter https://github.com/tom-schoener/ml-probability/blob/master/tfp-evaluation/notebooks/air_quality.ipynb einsehbar.

7 Fazit