Mirai Solutions
smarter analytics - better decisions

TensorFlow Probability - Why we Should Care

Roland A. Schmid

14 June 2019

Mirai Solutions - https://mirai-solutions.ch

Background and Intro

Data scientists in the insurance industry are **adopting AI techniques** rapidly. **Machine learning (ML) is the strongest emerging field** within the actuarial sciences. This can be seen as a paradigm change. We are confronted with:

- Larger datasets
- High-frequency and high-dimensional data
- Applications and models behind them need to provide faster or even real-time responses

Performance gains can be crucial, e.g. when deciding whether to offer / accept a new policy.

Background and Intro

Traditionally, an actuarial model starts with **specifying a stochastic data generator** (i.e. distribution assumptions etc).

Machine learning switches the focus. We can **build and train algorithms that successfully predict responses based on data** (initially losing interpretability and transparency). Would be great if we could again extract information about assumptions and building blocks of a resulting model, thus enabling us to refine it while still combining our expert knowledge into its definition.

The Challenge - (ref only)

This brave new world bears a **triple challenge** for actuarial / modeling teams:

- From classical statistical modeling to ML algorithms (**know-how**).
- New infrastructure and tools (new **technology and hardware**).
- Explaining benefits of ML solutions vs traditional models (new **mindset and focus**: performance, scalability, interoperability, transparency (?), reproducibility (?), interpretability (?), uncertainty (?), sensitivities (?), causal relationships (?)).

Overall, the **goal is to enhance the predictive power of models**, in particular also for problems with small and medium data, while at the same time **reducing the complexity and effort**, allowing more trials with better comparability and shorter time to production.

What is TensorFlow Probability (TFP)?

TensorFlow Probability is an **open source Python library** built using TensorFlow. It **works seamlessly with core TF and Keras**.

Introduced / announced at TF Dev Summit **around April 2018**, still under continuous development. Used in production systems.

Google product. Good integration in Google Cloud Platform and Google Colab, and strong active community behind.

- Provide **comprehensive statistical tools**
- Allows to combine **deep learning** with **probabilistic models**
- On modern hardware:
  - Distributed computing
  - Hardware acceleration (GPUs/TPUs besides CPUs)
  - Batching and vectorization

Goals and Audience of TFP

ML practitioners looking for

- A tool to **capture uncertainty** as part of their **deep learning models**, which can also provide better insight and enhance the interpretability.

Statisticians / data scientists looking for

- A tool with R-like capabilities that is **concise to use, performant and scales well**, by exploiting modern hardware (e.g. runs out-of-the-box on TPUs + GPUs).

–> Should be of **great interest to actuaries and the insurance industry** as a whole!

How Does it Work?

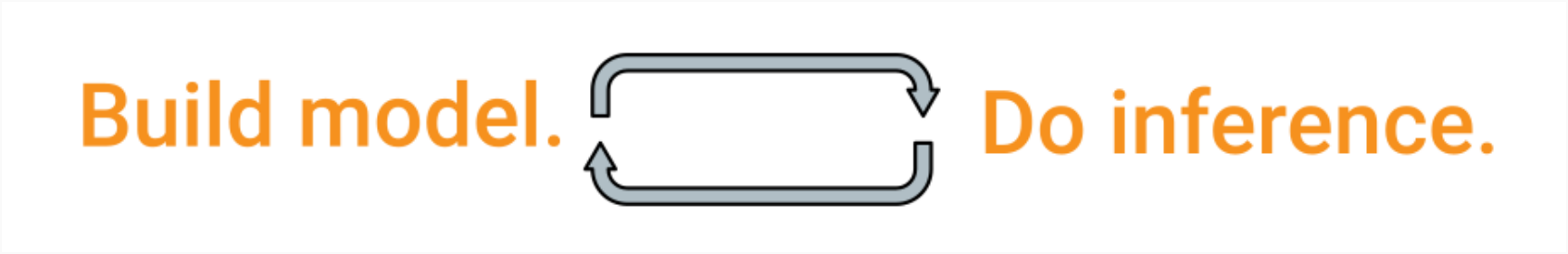


Figure 1: Source: Google

TFP is a **collection** of low-level tools which make it easier for data scientists to express what they already know about their model. It is not going to solve your model issues, but provides better tools to address them (esp. in the context of ML or large computations).

- **Domain knowledge still needed** to define the model
- Let TFP execute it and give back some useful insights, also about your model

What Does it Contain?

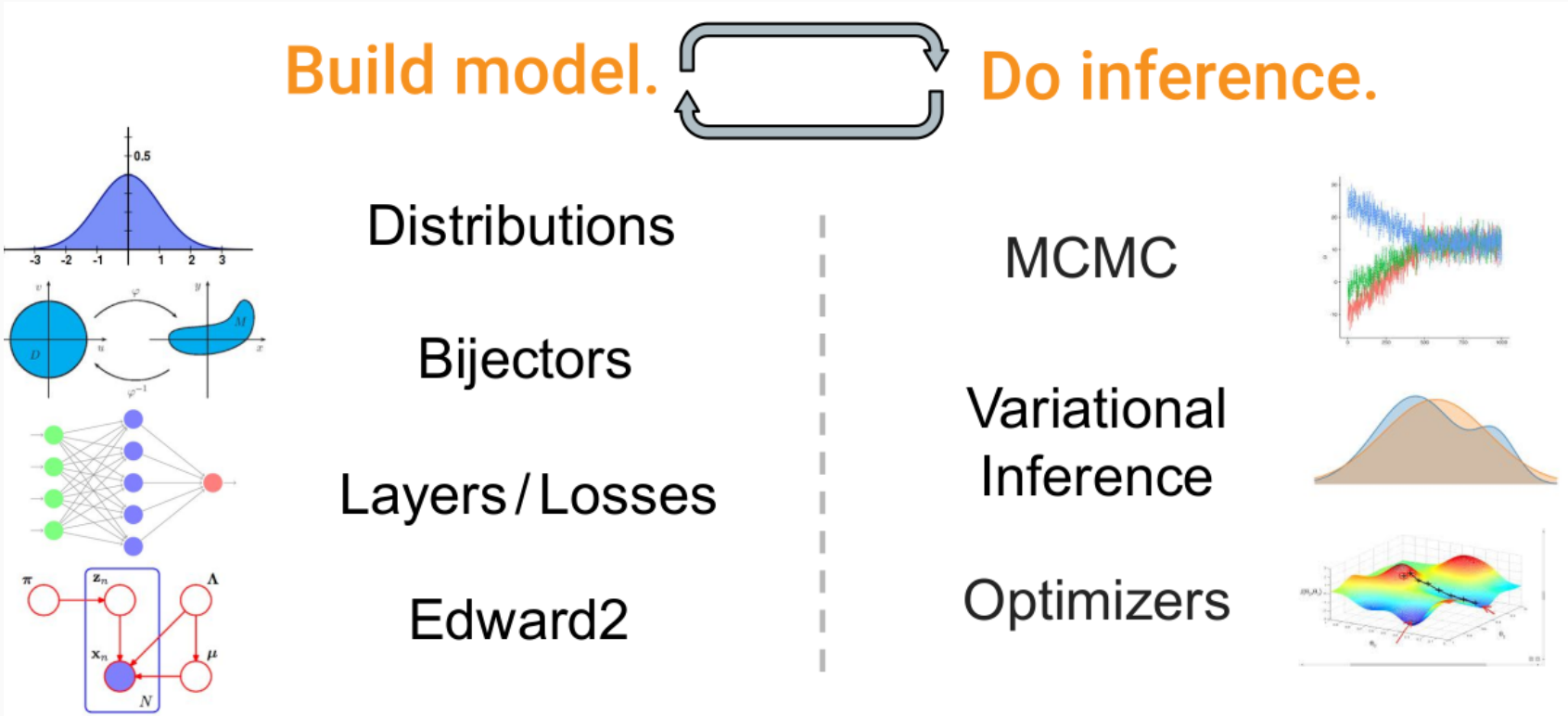


Figure 2: Source: Google

What Does it Contain - (1)

Statistical tools

- Distributions and Stats: A large collection of probability distributions and related statistics.
  - Parameterized distributions which **take advantage of TF vector computation**.
- GLMs: TensorFlow Probability **GLM Python package**.
- Bijectors: **Reversible and composable transformations** of random variables. Bijectors enable a rich class of transformed distributions.
  - Can e.g. be used for **copulas** or **autoregressive flows**.

What Does it Contain - (2)

Model building tools

- Trainable distributions: Probability distributions **parameterized by a single tensor**, making it easy to build neural nets that output probability distributions.
- **Probabilistic layers**, also through the Keras-like TFP Layers interface.
- Edward2: A **(deep) probabilistic programming language / system** for higher-level specification of flexible probabilistic models as programs.

What Does it Contain - (3)

Probabilistic (Bayesian) inference

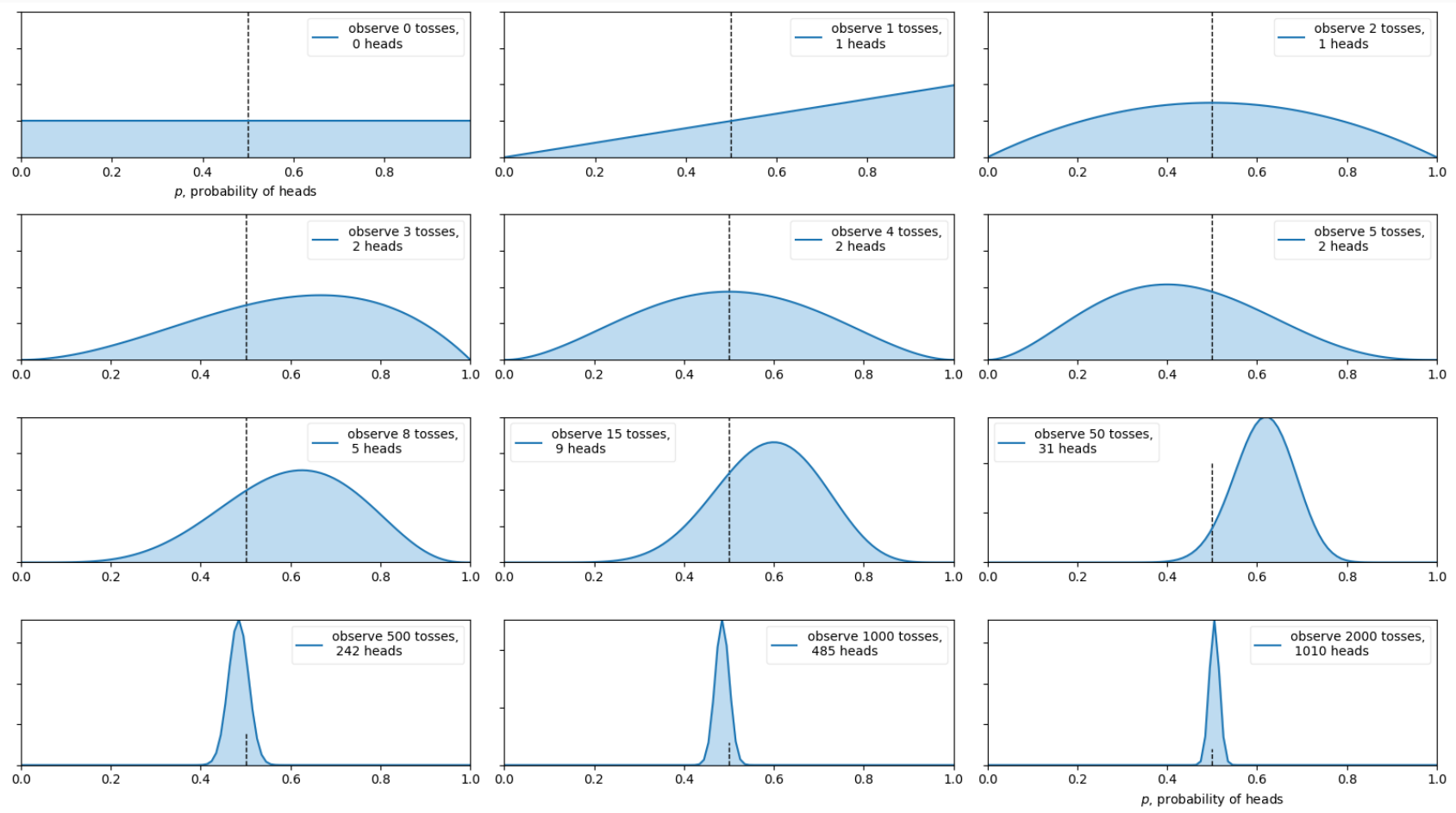
- Markov chain Monte Carlo (MCMC): **Algorithms for approximating integrals via sampling**. Includes Hamiltonian Monte Carlo and the possibility to build custom transition kernels.
- **Variational Inference**: Algorithms for approximating integrals through optimization.
- Optimizers: **Stochastic optimization methods**, extending TensorFlow Optimizers. Includes Stochastic Gradient Langevin Dynamics besides Nelder-Mead and BFGS.

TFP Basic Example - Bayesian Statistics

Coin-flip example - after McAteer/Seybold/Google

```
import tensorflow as tf
import tensorflow_probability as tfp

rv_coin_flip_prior = tfp.distributions.Bernoulli(probs=0.5, dtype=tf.int32)
# set of increasing numbers of trials
num_trials = tf.constant([0, 1, 2, 3, 4, 5, 8, 15, 50, 500, 1000, 2000])

gdummy = tf.set_random_seed(11)  # fix the seed for the next "eager" operation
coin_flip_data = rv_coin_flip_prior.sample(num_trials[-1])
# prepend a 0 onto tally of heads and tails, for zeroth flip
coin_flip_data = tf.pad(coin_flip_data, tf.constant([[1, 0]]), "CONSTANT")

# compute cumulative headcounts, then grab them at each of num_trials' intervals
cumulative_headcounts = tf.gather(tf.cumsum(coin_flip_data), num_trials)

rv_observed_heads = tfp.distributions.Beta(
    concentration1=tf.cast(1 + cumulative_headcounts, tf.float32),
    concentration0=tf.cast(1 + num_trials - cumulative_headcounts, tf.float32))

probs_of_heads = tf.linspace(start=0., stop=1., num=100, name="linspace")
observed_probs_heads = tf.transpose(rv_observed_heads.prob(probs_of_heads[:, tf.newaxis]))
```

Coin-flip Plots

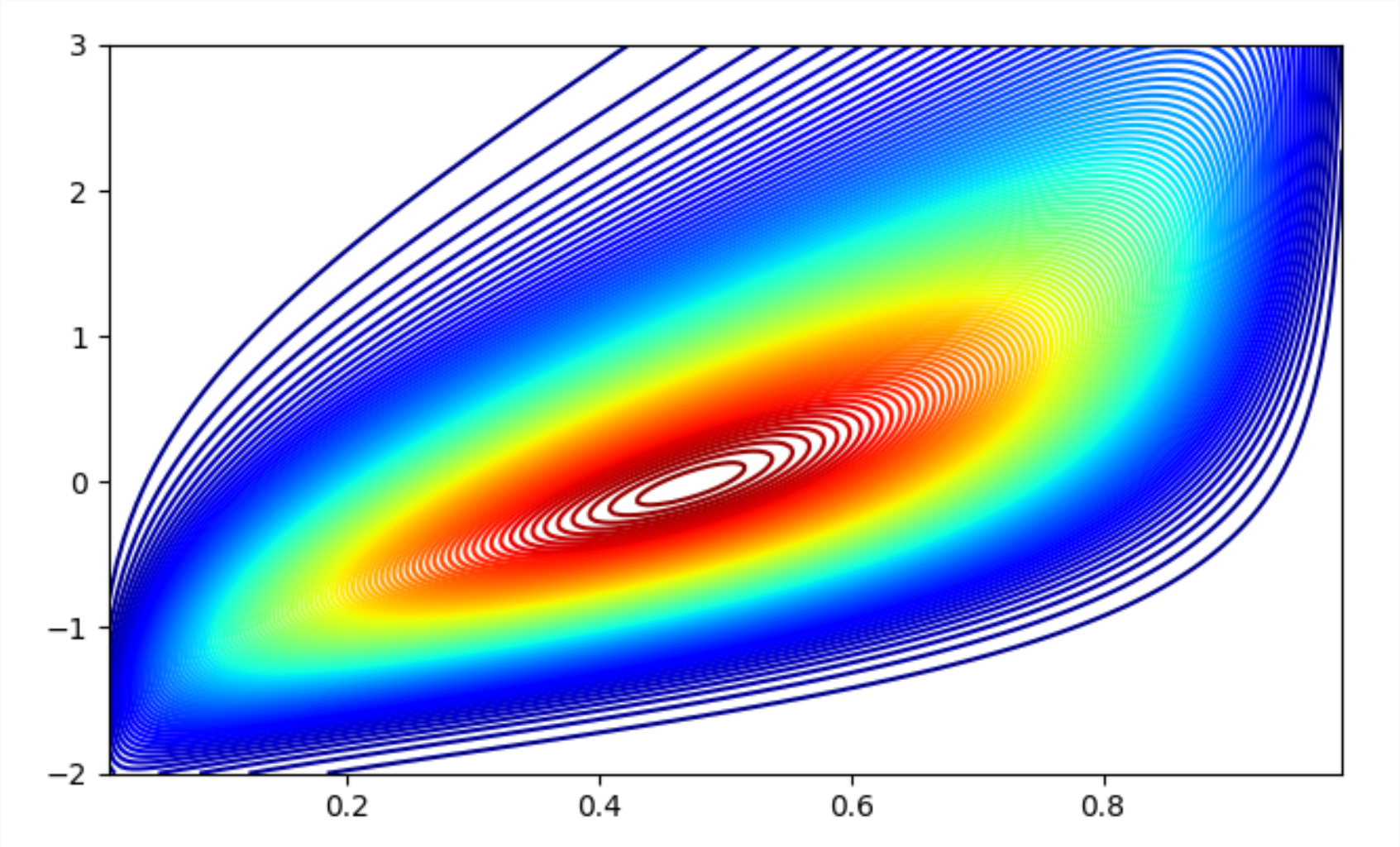


Bijectors Example - Gaussian Copula

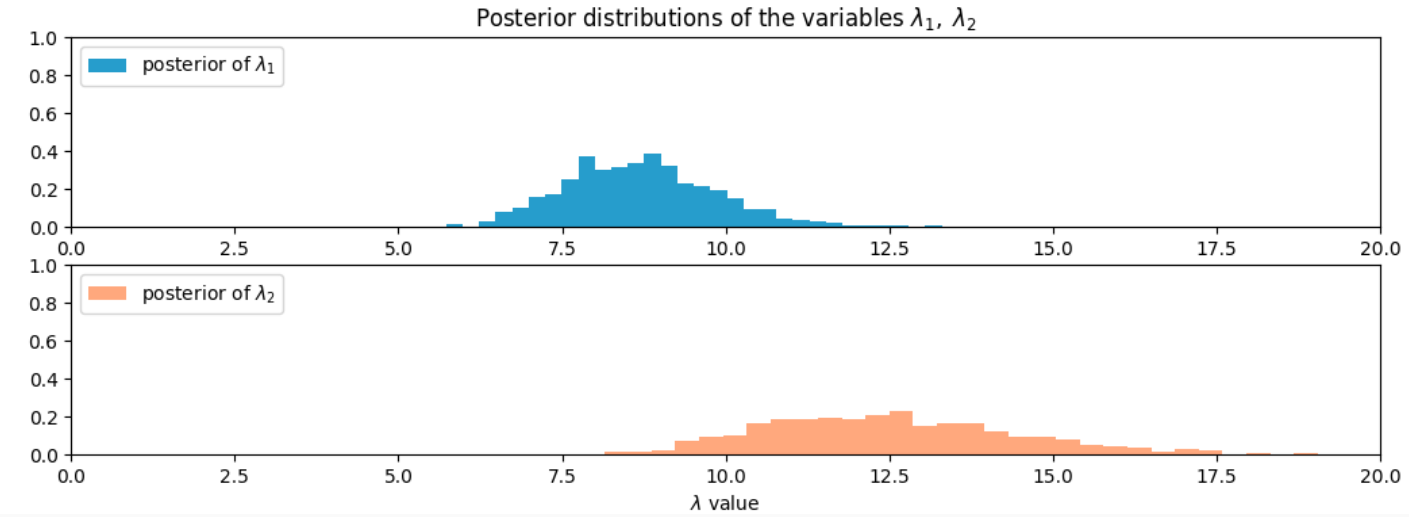
Based on Google / TF example on GitHub

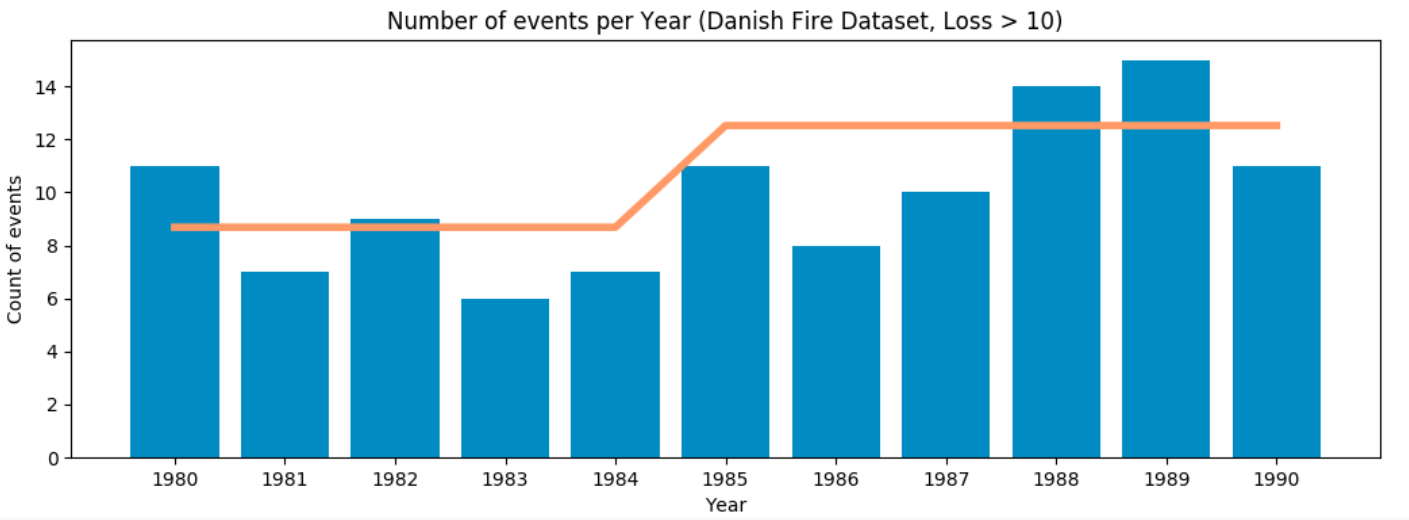
```
tfd = tfp.distributions; tfb = tfp.bijectors

class MyGaussianCopula(tfd.TransformedDistribution):
    """This implements an application of a Gaussian Copula, such that the
    resulting multivariate distribution has the specified target marginals.
    The marginals are specified by `marginal_bijectors`.
    """
    def __init__(self, loc, scale_tril, marginal_bijectors):
        super(MyGaussianCopula, self).__init__(
            distribution=tfd.MultivariateNormalTriL(loc=loc, scale_tril=scale_tril),
            bijector=tfb.Chain(bijectors=[Concat(marginal_bijectors), tfb.NormalCDF()]),
            validate_args=False,
            name="GaussianCopula")

copula = MyGaussianCopula(
    loc=[0., 0.],
    scale_tril=[[1., 0.], [0.7, tf.sqrt(1. - 0.7 ** 2)]],
    # Below encodes the marginals we want. X_0 has Kumaraswamy, and X_1 Gumbel.
    marginal_bijectors=[
        tfb.Kumaraswamy(2., 2.),
        tfb.Invert(tfb.Gumbel(loc=0., scale=1.))])

copula.sample(100)
copula.prob(coordinates)
```

Gaussian Copula Plot



Loss Frequency (Bayesian Approach)





Risk Quantification (Code Excerpt)

```
# Initialize the step_size. (It will be automatically adapted.)
with tf.variable_scope(tf.get_variable_scope(), reuse=tf.AUTO_REUSE):
    step_size = tf.get_variable(
        name='step_size',
        initializer=tf.constant(0.05, dtype=tf.float32),
        trainable=False,
        use_resource=True
    )

# Sample from the chain.
[lambda_1_samples,
    lambda_2_samples,
    posterior_tau], kernel_results = tfp.mcmc.sample_chain(
    num_results=1000,
    num_burnin_steps=100,
    current_state=initial_chain_state,
    kernel=tfp.mcmc.TransformedTransitionKernel(
        inner_kernel=tfp.mcmc.HamiltonianMonteCarlo(
            target_log_prob_fn=unnormalized_log_posterior,
            num_leapfrog_steps=2,
            step_size=step_size,
            step_size_update_fn=tfp.mcmc.make_simple_step_size_update_policy(num_adaptation_steps=int(burnin * 0.8)),
            state_gradients_are_stopped=True),
        bijector=unconstraining_bijectors))

tau_samples = tf.floor(posterior_tau * tf.to_float(tf.size(events_tf)))
```

Conclusions and Take-away Messages

TFP is a fairly new Python library based on TF which helps to **combine deep learning with probabilistic models**.

- So far, few examples on TFP project, but **little practical expertise** in insurance / finance.
- TFP community is working to **close the gap** between IT experts and practitioners.
- No applied examples for insurance.

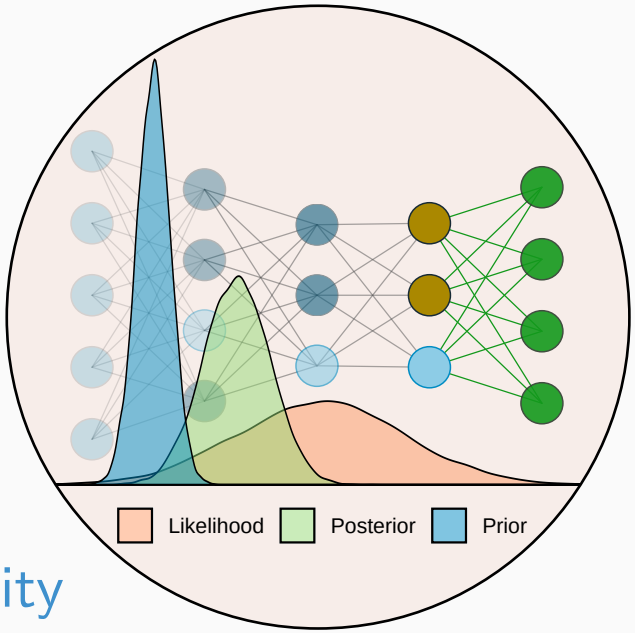
Open-source, free, community-driven, rather easy & accessible (Python). R packages with interface to Python available too.

–> further contributions / collaborations towards more **applied examples highly desired!**

Conclusions and Take-away Questions

- When could TFP pay off compared to a traditional implementation of e.g. copula, Bayesian modeling or a regression? What are expected benefits?
- What are the pre-requisites for a successful integration of a TFP solution (e.g. human capital, hardware / infrastructure, etc)?
- What might be the promising applications (of TFP) in insurance? Why exactly is it promising again?

Learn More



Where to get started / further reading:

- https://www.tensorflow.org/probability
- https://github.com/tensorflow/probability
- https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers
- https://github.com/rstudio/tfprobability
- https://blogs.rstudio.com/tensorflow/