Dive Deeper in Finance

GTC 2017 – San José – California

Daniel Egloff
Dr. sc. math.
Managing Director QuantAlea
May 7, 2017
Today

- Generative models for financial time series
  - Sequential latent Gaussian Variational Autoencoder
- Implementation in TensorFlow
  - Recurrent variational inference using TF control flow operations
- Applications to FX data
  - 1s to 10s OHLC aggregated data
  - Event based models for tick data is work in progress
Generative Models and GPUs

▪ What I cannot create, I do not understand (Richard Feynman)

▪ Generative models are recent innovation in Deep Learning
  – GANs – Generative adversarial networks
  – VAE – Variational autoencoders

▪ Training is computationally demanding
  – Explorative modelling not possible without GPUs
Deep Learning

- Deep Learning in finance is complementary to existing models and not a replacement

- Deep Learning benefits
  - Richer functional relationship between explanatory and response variables
  - Model complicated interactions
  - Automatic feature discovery
  - Capable to handle large amounts of data
  - Standard training procedures with back propagation and SGD
  - Frameworks and tooling
Latent Variable – Encoding/Decoding

- Latent variable can be thought of a encoded representation of $x$
- Likelihood serves as decoder
- Posterior provides encoder
Intractable Maximum Likelihood

- Maximum likelihood standard model fitting approach

\[ p(x) = \int p(x|z) p(z) dz \rightarrow \text{max} \]

- **Problem**: marginal \( p(x) \) and posterior

\[ p(z|x) = \frac{p(x|z)p(z)}{p(x)} \]

are intractable and their calculation suffers from exponential complexity

- **Solutions**
  - Markov Chain MC, Hamiltonian MC
  - Approximation and variational inference
Variational Autoencoders

- Assume latent space with prior $p(z)$
Variational Autoencoders

- Parameterize likelihood $p(x|z)$ with a deep neural network

$$p(x|z)$$
Variational Autoencoders

- Parameterize likelihood $p(x|z)$ with a deep neural network
- Approximate intractable posterior $p(z|x)$ with a deep neural network
Variational Autoencoders

- Parameterize likelihood $p(x|z)$ with a deep neural network
- Approximate intractable posterior $p(z|x)$ with a deep neural network
- Learn the parameters $\theta$ and $\varphi$ with backpropagation
Variational Inference

- Which loss to optimize?
- Can we choose posterior from a flexible family of distributions $Q$ by minimizing a distance to real posterior?

$$q^*(z|x) = \arg\min_{\theta \in Q} KL(q_\theta(z|x) \| p_\varphi(z|x))$$

- Problem: not computable because it involves marginal $p_\varphi(x)$

$$KL(q_\theta(z|x) \| p_\varphi(z|x)) = E_{q_\theta(z|x)}[\log q_\theta(z|x)] - E_{q_\theta(z|x)}[\log p_\varphi(x, z)] + \log p_\varphi(x) \geq 0$$

Can be made small if $Q$ is flexible enough
Variational Inference

- Which loss to optimize?
- Can we choose posterior from a flexible family of distributions $Q$ by minimizing a distance to real posterior?

$$q^*(z|x) = \arg\min_{\theta \in Q} KL(q_{\theta}(z|x)\|p_{\varphi}(z|x))$$

- Drop left hand side because positive

$$0 \leq E_{q_{\theta}(z|x)}[\log q_{\theta}(z|x)] - E_{q_{\theta}(z|x)}[\log p_{\varphi}(x, z)] + \log p_{\varphi}(x)$$

$$-ELBO(\theta, \varphi)$$
Variational Inference

- Which loss to optimize?
- Can we choose posterior from a flexible family of distributions \( Q \) by minimizing a distance to real posterior?

\[
q^*(z|x) = \arg\min_{\theta \in Q} KL(q_\theta(z|x) \| p_\varphi(z|x))
\]

- Obtain tractable lower bound for marginal

\[
ELBO(\theta, \varphi) \leq \log p_\varphi(x)
\]

- Training criterion: \textbf{maximize evidence lower bound}
Variational Inference

To interpret lower bound, write it as

\[ \log p_\phi(x) \geq ELOB(\theta, \varphi) \]

\[ = E_{q_\theta(z|x)}[\log p_\phi(x|z)] - KL(q_\theta(z|x)\|p(z)) \]

- **Reconstruction score**
- **Penalty of deviation from prior**

The smaller \( KL(q_\theta(z|x)\|p_\phi(z|x)) \) the tighter the lower bound
Applications to Time Series

- Sequence structure for observable and latent factor
- Model setup
  - Gaussian distributions with parameters calculated from deep recurrent neural network
  - Prior standard Gaussian
  - Model training with variational inference
Inference and Training

\[
\begin{align*}
\mu_{t-1} &\quad \sigma_{t-1} \\
h_{t-1} &\quad \mu_t &\quad \sigma_t &\quad h_t &\quad \mu_{t+1} &\quad \sigma_{t+1} \\
x_{t-1} &\quad \mu_{t-1} &\quad \sigma_{t-1} &\quad x_t &\quad \mu_t &\quad \sigma_t &\quad x_{t+1} \\
z_{t-1} &\quad \mu_{t+1} &\quad \sigma_{t+1} &\quad z_t &\quad q_\theta(z|x) &\quad z_{t+1} \\
h_{t-1} &\quad h_t &\quad h_t &\quad h_{t+1} \\
\end{align*}
\]
Implied Factorization

- Probability distributions factorize

\[ p_{\phi}(x_{\leq T}|z_{\leq T}) = \prod_{t=1}^{T} p_{\phi}(x_t|x_{<t}, z_{\leq t}) = \prod_{t=1}^{T} N(x_t|\mu_{\phi}(x_{<t}, z_{\leq t}), \sigma_{\phi}(x_{<t}, z_{\leq t})) \]

\[ q_{\theta}(z_{\leq T}|x_{\leq T}) = \prod_{t=1}^{T} q_{\theta}(z_t|x_{<t}, z_{<t}) = \prod_{t=1}^{T} N(z_t|\mu_{\theta}(x_{<t}, z_{<t}), \sigma_{\theta}(x_{<t}, z_{<t})) \]

- Loss calculation
  - Distributions can be easily simulated to calculate expectation term
  - Kullback Leibler term can be calculated analytically
Calculating ELBO

- Loss calculation
  - Kullback Leibler term can be calculated analytically
  - For fixed \( t \) the quantities \( \mu_\varphi, \mu_\theta, \sigma_\varphi, \sigma_\theta \) depend on 
    \[ z_t \sim N(z_t | \mu_\theta(x_{<t}, z_{<t}), \sigma_\theta(x_{<t}, z_{<t})) \]
  - Simulate from this distribution to estimate expectation with a sample mean

\[
ELBO(\theta, \varphi) = -E_q \left[ \sum_t \left\{ (x_t - \mu_\varphi)^T \sigma_\varphi^{-1} (x_t - \mu_\varphi) + \log \det \sigma_\varphi + 
\mu_\theta^T \mu_\theta + tr\sigma_\theta - \log \det \sigma_\theta \right\} \right]
\]

Approximate with Monte Carlo sampling from \( q_\theta(z_{\leq T} | x_{\leq T}) \)
Generation
Time Series Embedding

- Single historical value not predictive enough
- Embedding
  - Use lag of ~20 historical observations at every time step
Implementation

- Implementation in TensorFlow
- Running on P100 GPUs for model training
- Long time series and large batch sizes require substantial GPU memory
TensorFlow Dynamic RNN

- Unrolling `rnn` with `tf.nn.dynamic_rnn`
  - Simple to use
  - Can handle variable sequence length
- Not flexible enough for generative networks

```
B = 3
D = 4
T = 5
PKEEP = 0.9

tf.reset_default_graph()

x = tf.placeholder(shape=[T, B, D], dtype=tf.float32)

with tf.variable_scope("RNN"):
    cell = tf.contrib.rnn.GRUCell(num_units = D)
    cell = tf.contrib.rnn.DropoutWrapper(cell, output_keep_prob = PKEEP)
    cells = tf.contrib.rnn.MultiRNNCell([cell])

    h, states = tf.nn.dynamic_rnn(cells, inputs = x, time_major=True, dtype=tf.float32)
```
TensorFlow Control Structures

- Using `tf.while_loop`
  - More to program, need to understand control structures in more detail
  - Much more flexible

```python
x = tf.placeholder(shape=[T, B], dtype=tf.float32)
output_ta = tf.TensorArray(size=T, dtype=tf.float32)
input_ta = tf.TensorArray(size=T, dtype=tf.float32)
input_ta = input_ta.unstack(x)

def body(time, output_ta):
    xt = input_ta.read(time)
    output_ta = output_ta.write(time, tf.reduce_sum(xt**2))
    return (time+1, output_ta)

time_final, output_ta_final = tf.while_loop(
    cond=lambda time, *_: time < T,
    body=body,
    loop_vars=(time, output_ta))

output_final = output_ta_final.stack()
```
Implementation

- Notations

# B  batch size
# T  number of time steps
# XDIM  time series embedding dimension, e.g. number of lags
# ZDIM  dimension of latent variable
# ODIM  number of outputs
# NZ   number of latent variable samples
Implementation

- Variable and Weight Setup

```python
def create_rnn(dim):
    cells = []
    for i in range(0, 2):
        c = tf.contrib.rnn.GRUCell(num_units = dim)
        c = tf.contrib.rnn.DropoutWrapper(c, output_keep_prob = PKEEP)
        cells.append(c)
    return tf.contrib.rnn.MultiRNNCell(cells)

x = tf.placeholder(shape=[T, B, XDIM], dtype=tf.float32)
eps_q = tf.placeholder(shape=[T, B, NZ, ZDIM], dtype=tf.float32)

with tf.variable_scope("inference") as inf_scope:
    rnn_inf = create_rnn(XDIM)
    w_inf = weight([XDIM, 2*ZDIM])
    b_inf = bias([2*ZDIM])

with tf.variable_scope("generator") as gen_scope:
    rnn_gen = create_rnn(XDIM+ZDIM)
    w_gen = weight([XDIM+ZDIM, 2*ODIM])
    b_gen = bias([2*ODIM])
```

Recurrent neural network definition
Implementation

- Allocate TensorArray objects
- Fill input TensorArray objects with data

```python
# prepare tensor arrays
z_ta = tf.TensorArray(size=T, dtype=tf.float32)
x_ta = tf.TensorArray(size=T, dtype=tf.float32)
eps_q_ta = tf.TensorArray(size=T, dtype=tf.float32)
mu_x_ta = tf.TensorArray(size=T, dtype=tf.float32)
sigma_x_ta = tf.TensorArray(size=T, dtype=tf.float32)
mu_x_mean_ta = tf.TensorArray(size=T, dtype=tf.float32)
sigma_x_mean_ta = tf.TensorArray(size=T, dtype=tf.float32)

x_ta = x_ta.unstack(x)
eps_q_ta = eps_q_ta.unstack(eps_q)
```
Implementation

- While loop body inference part

```python
def body(time, z_ta, mu_x_ta, sigma_x_ta, mu_x_mean_ta, sigma_x_mean_ta, state_inf, state_gen):
    xt = x_ta.read(time)  # [B, XDIM]

    with tf.variable_scope(inf_scope):
        output_inf, state_inf = rnn_inf(xt, state_inf)  # [B, XDIM]
        qt_inf_params = tf.matmul(output_inf, w_inf) + b_inf  # [B, 2*ZDIM]

        # parameters for q(z_{t+1} / x_{<=t}, z_{<=t})
        mu_qt, log_var_qt = tf.split(axis=1, num_or_size_splits=2, value=qt_inf_params)  # [B, ZDIM], [B, ZDIM]
        sigma_qt = tf.sqrt(tf.exp(log_var_qt) + EPS)

    zt_inf = tf.expand_dims(mu_qt, 1) + tf.expand_dims(sigma_qt, 1)*eps_q_ta.read(time)  # [B, NZ, ZDIM]

    xt_stacked = tf.stack([xt]*NZ, axis=1)  # [B, NZ, XDIM]
    xt_zt_inf = tf.concat(values=[xt_stacked, zt_inf], axis=2)  # [B, NZ, XDIM+ZDIM]
    xt_zt_inf_2d = tf.reshape(xt_zt_inf, [-1, XDIM+ZDIM])  # [B*NZ, XDIM+ZDIM]
```

Update inference rnn state
def body(time, z_ta, mu_x_ta, sigma_x_ta, mu_x_mean_ta, sigma_x_mean_ta, state_inf, state_gen):
    # ..... 

    with tf.variable_scope(gen_scope):
        output_gen, state_gen = rnn_gen(xt_zt_inf_2d, state_gen): # [B*NZ, XDIM+ZDIM]
            xt_gen_params = tf.matmul(output_gen, w_gen) + b_gen # [B*NZ, 2*ODIM]

        xt_gen_params = tf.reshape(xt_gen_params, [B, NZ, 2*ODIM])
        mu_xt, log_var_xt = tf.split(axis=2, num_or_size_splits=2, value=xt_gen_params)
        sigma_xt = tf.sqrt(tf.exp(log_var_xt) + EPS)

        z_ta = z_ta.write(time, zt_inf)
        mu_x_ta = mu_x_ta.write(time, mu_xt)
        sigma_x_ta = sigma_x_ta.write(time, sigma_xt)
        mu_x_mean_ta = mu_x_mean_ta.write(time, tf.reduce_mean(mu_xt, 1))
        sigma_x_mean_ta = sigma_x_mean_ta.write(time, tf.reduce_mean(sigma_xt, 1))

    return (time+1, z_ta, mu_x_ta, sigma_x_ta, mu_x_mean_ta, sigma_x_mean_ta, state_inf, state_gen)
### Implementation

- Call while loop
- Stacking TensorArray objects

```python
# Initialize variables

# Call while loop

# Stack the resulting arrays
```
Implementation

- Loss Calculation

```python
def drop_first(tensor):
    return tensor[1::,::,::]

def drop_last(tensor):
    return tensor[:T-1,::,::]

def get_last_embedded(tensor):
    return tf.expand_dims(tensor[:,::,XDIM-ODIM:], 2)

# the last output section of the embedded time series is the target
# we forecast the next point so start at T = 1
target = drop_first(get_last_embedded(x))

# loss to minimize, which is -ELBO, as we would have to maximize ELBO
loss1 = 0.5*(tf.reduce_mean(mu_x**2) + tf.reduce_mean(sigma_x**2) - tf.reduce_mean(tf.log(sigma_x**2)))
loss2 = 0.5*tf.reduce_mean(tf.log(drop_last(sigma_x)**2))
loss3 = 0.5*tf.reduce_mean(((target - drop_last(mu_x))**2/drop_last(sigma_x))
weight_penalty = tf.reduce_mean(w_inf**2) + tf.reduce_mean(w_gen**2)
loss = loss1 + loss2 + loss3 + weight_penalty
```
FX Market

- FX market is largest and most liquid market in the world
- Decentralized over the counter market
  - Not necessary to go through a centralized exchange
  - No single price for a currency at a given point in time
- Fierce competition between market participants
- 24 hours, 5 ½ days per week
  - As one major forex market closes, another one opens
FX Data

- Collect tick data from major liquidity provider e.g. LMAX
- Aggregation to OHLC bars (1s, 10s, ...)
- Focus on US trading session

8am - 5pm EST

3am - 12am EST

7pm - 4am EST (Tokyo)

5pm - 2am EST (Sidney)

US session

London session

Asian session
Single Day
One Hour
At high frequency FX prices fluctuate in range of deci-pips.

Larger jumps in the order of multiple pips and more.

1/10 pips = 1 deci-pip.
Setup

- Normalize data with std deviation $\hat{\sigma}$ over training interval
- 260 trading days in 2016, one model per day
- 60 dim embedding, 2 dim latent space
Results

Training
Out of Sample
Volatility of Prediction
Latent Variables
Pricing in E-Commerce

- Attend our talk on our latest work on AI and GPU accelerated genetic algorithms with Jet.com

Prices Drop as you Shop

1. Intro
2. E-Commerce Market Potential
3. Problem
4. Brute Force Approach with GPUs
5. A Smarter GPU Solution
Contact details

Daniel Egloff
Dr. sc. math.
Phone: +41 79 430 03 61
daniel.egloff@quantalea.net