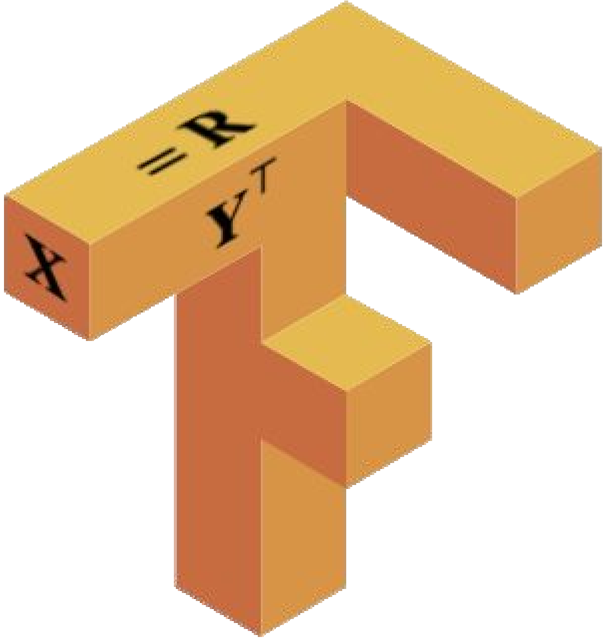
Recommender systems with Python & TensorFlow

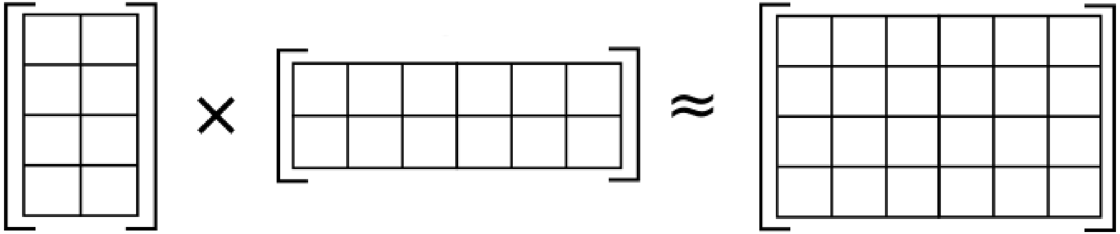
PyData

London 2017

TensorFlow

TensorFlow based Matrix Factorisation

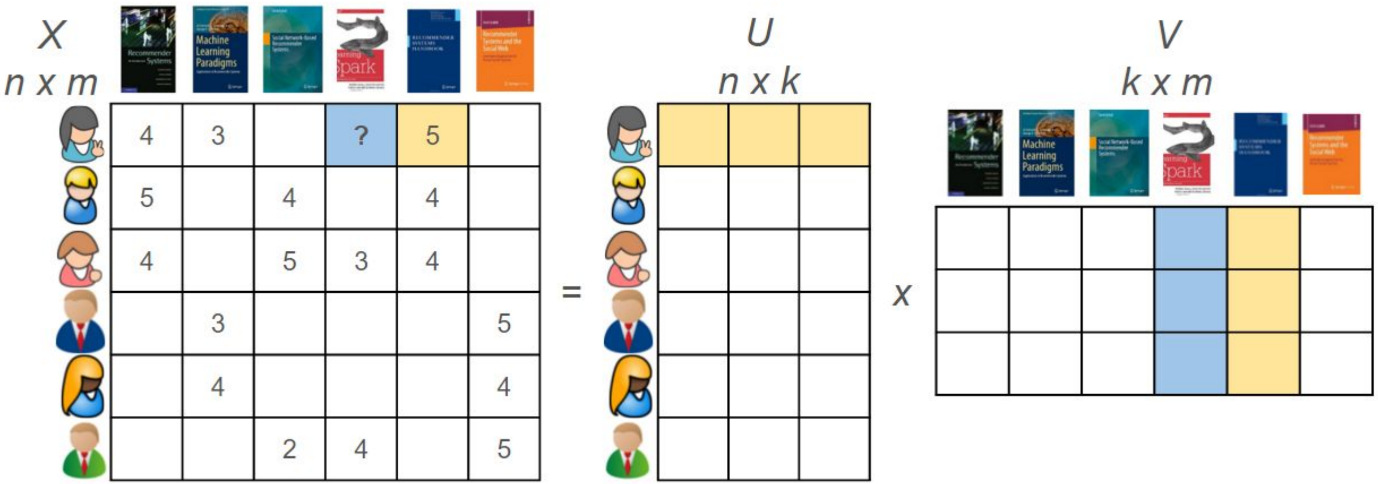




Brief introduction to Recommender systems

From the Netflix prize (2008)

To real world use cases (Amazon, Spotify, Youtube, Criteo, Mendeley, Schibsted)



(From the Mendeley's blog)

Collaborative Filtering for Implicit Feedbacks (2008)

+ Not only explicit ratings, also negative implicit interactions

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

+ Alternating Least Squares is fast and distributed (org.apache.spark.ml)

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

- RMSE only
- Click only, no user or item features => cold start issue!

Logistic MF for implicit feedback data (2014)

+ Binary click prediction

$$p(l_{ui} \mid x_u, y_i, \beta_i, \beta_j) = \frac{\exp(x_i y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)}$$

```
class LogisticMF():
    ...

    def inference(user_ids, item_ids):

        logits = np.dot(self.user_vectors[user_ids], self.item_vectors[item_ids].T)\
            + self.user_biases[user_ids] + self.item_biases[item_ids].T

        return 1 / (1 + np.exp(-logits))
```

Deriving gradients manually...

$$\frac{\partial}{\partial x_u} = \sum_i \alpha r_{ui} y_i - \frac{y_i(1+\alpha r_{ui})\exp(x_u y_i^T + \beta_u + \beta_i)}{1+\exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda x_u$$

```python
def user_gradients():
    vec_deriv = np.dot(self.counts, self.item_vectors)
    bias_deriv = np.expand_dims(np.sum(self.counts, axis=1), 1)

    A = np.exp(
        np.dot(self.user_vectors, self.item_vectors.T)\
        + self.user_biases + self.item_biases.T))
    A /= (A + self.ones)
    A = (self.counts + self.ones) * A

    vec_deriv -= np.dot(A, self.item_vectors) - self.reg_param * self.user_vectors
    bias_deriv -= np.expand_dims(np.sum(A, axis=1), 1)

    return vec_deriv, bias_deriv
```

And the gradient descent step (ADAGRAD)

$$x_u^t = x_u^{t-1} + \frac{\gamma g_u^{t-1}}{\sqrt{\sum_{t'=1}^{t-1} {g_u^{t'}}^2}}$$

```
def ada_grad_step():
    user_vec_deriv, user_bias_deriv = self.user_deriv()
    user_vec_deriv_sum += np.square(user_vec_deriv)
    user_bias_deriv_sum += np.square(user_bias_deriv)
    vec_step_size = self.gamma / np.sqrt(user_vec_deriv_sum)
    bias_step_size = self.gamma / np.sqrt(user_bias_deriv_sum)
    self.user_vectors += vec_step_size * user_vec_deriv
    self.user_biases += bias_step_size * user_bias_deriv
```

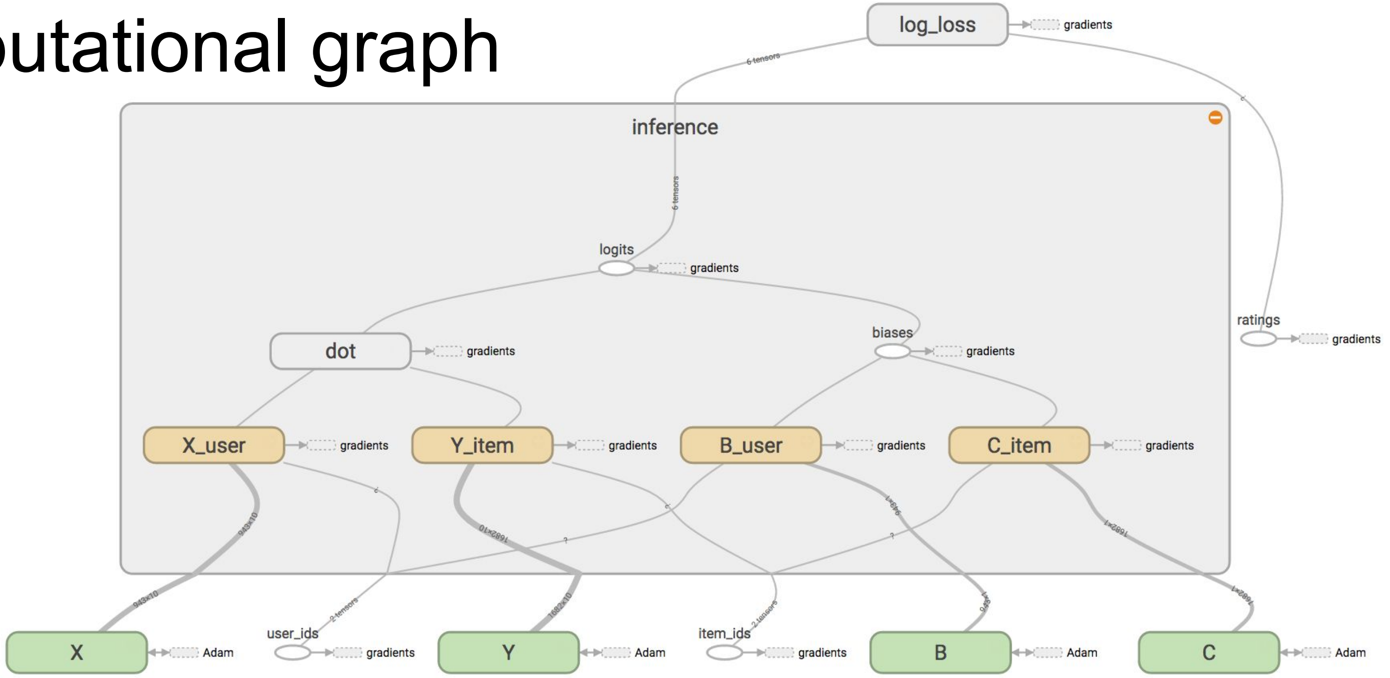
Welcome to symbolic computing and auto-diff!

Declaring the inference step in TF

```
def user_bias(self, user_ids):
    with tf.name_scope('B_user'):
        return tf.squeeze(tf.nn.embedding_lookup(params=self.user_biases, ids=user_ids), name='B_user')

def item_bias(self, item_ids):
    with tf.name_scope('C_item'):
        return tf.squeeze(tf.nn.embedding_lookup(params=self.item_biases, ids=item_ids), name='C_item')

def user_item_product(self, user_ids, item_ids):
    with tf.name_scope('X_user'):
        batch_user_factors = tf.squeeze(tf.nn.embedding_lookup(self.user_factors, user_ids))
    with tf.name_scope('Y_item'):
        batch_item_factors = tf.squeeze(tf.nn.embedding_lookup(self.item_factors, item_ids))
    with tf.name_scope('dot'):
        factors_prediction = tf.reduce_mean(
            tf.mul(batch_user_factors, batch_item_factors), reduction_indices=1)
    return factors_prediction

def inference(self, user_ids, item_ids):
    with tf.name_scope('inference'):
        return tf.add(
            self.user_item_product(user_ids, item_ids),
            tf.add(self.user_bias(user_ids), self.item_bias(item_ids), name='biases'),
            name='logits')
```

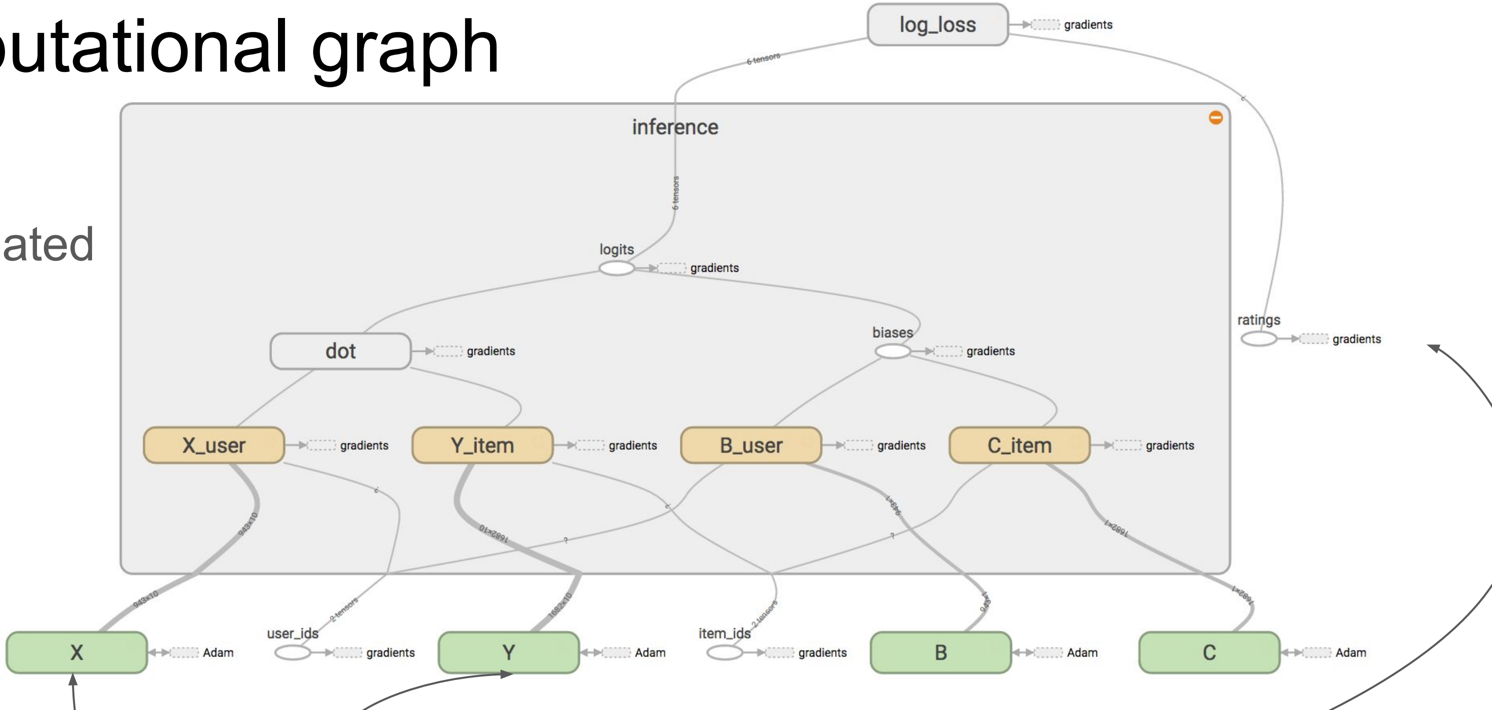
As a computational graph

- Compiled



As a computational graph

- Compiled
- Lazily evaluated by batches

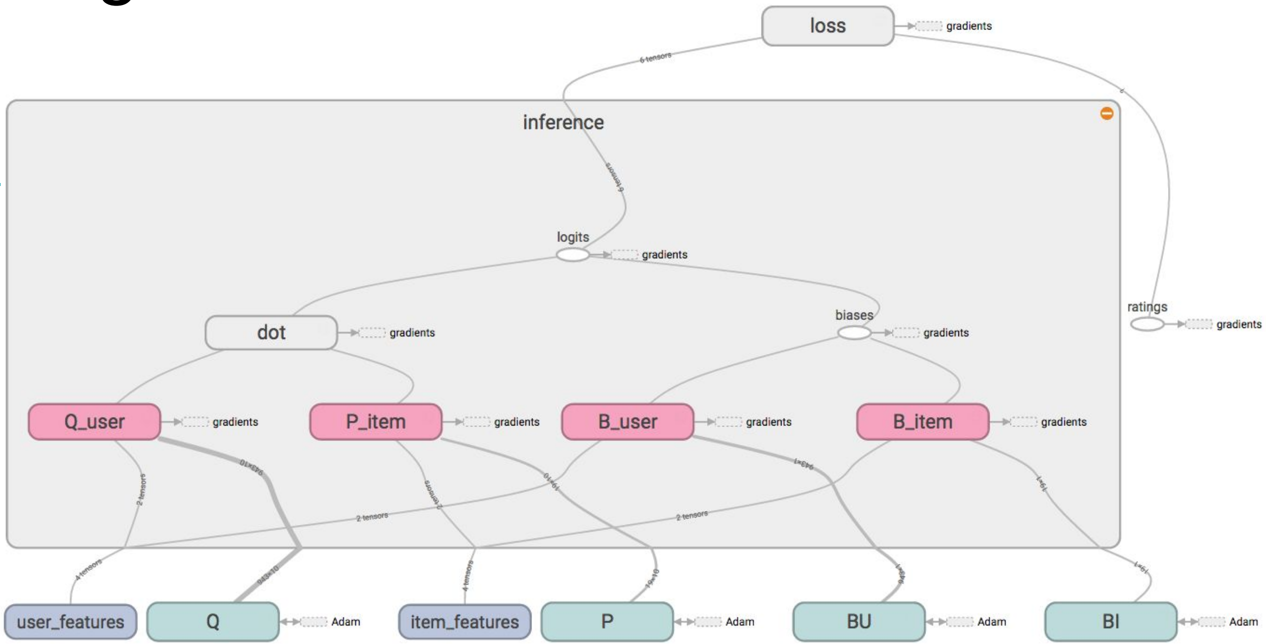


	item	rating	timestamp
user			
1	1193	5	2000-12-31 22:12:40
2	1357	5	2000-12-31 21:38:29
6038	1419	4	2000-04-26 02:06:55

Metadata Embeddings for user and item cold start

+ Incorporate metadata
+ github.com/lyst/lightfm

	item	rating	timestamp
user			
1	1193	5	2000-12-31 22:12:40
2	1357	5	2000-12-31 21:38:29
6038	1419	4	2000-04-26 02:06:55

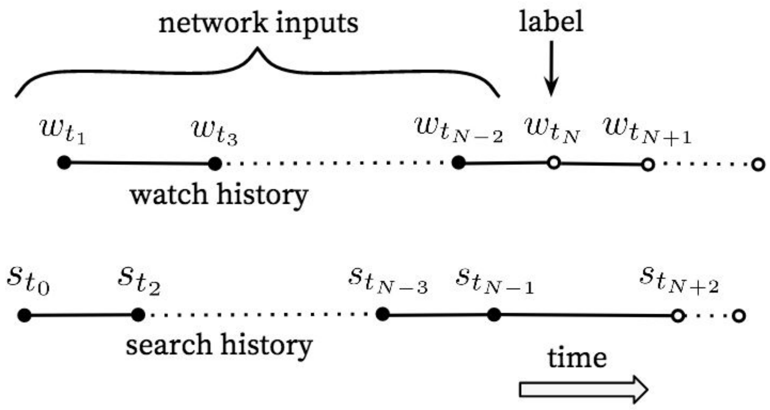


	age=1	age=18	age=25	age=35	age=45	age=50	age=56	gender=F	gender=M
user									
1	1	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	1	0	1
6038	0	0	0	0	0	0	1	1	0

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U \qquad \boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

Deep Neural Networks for YouTube Recs

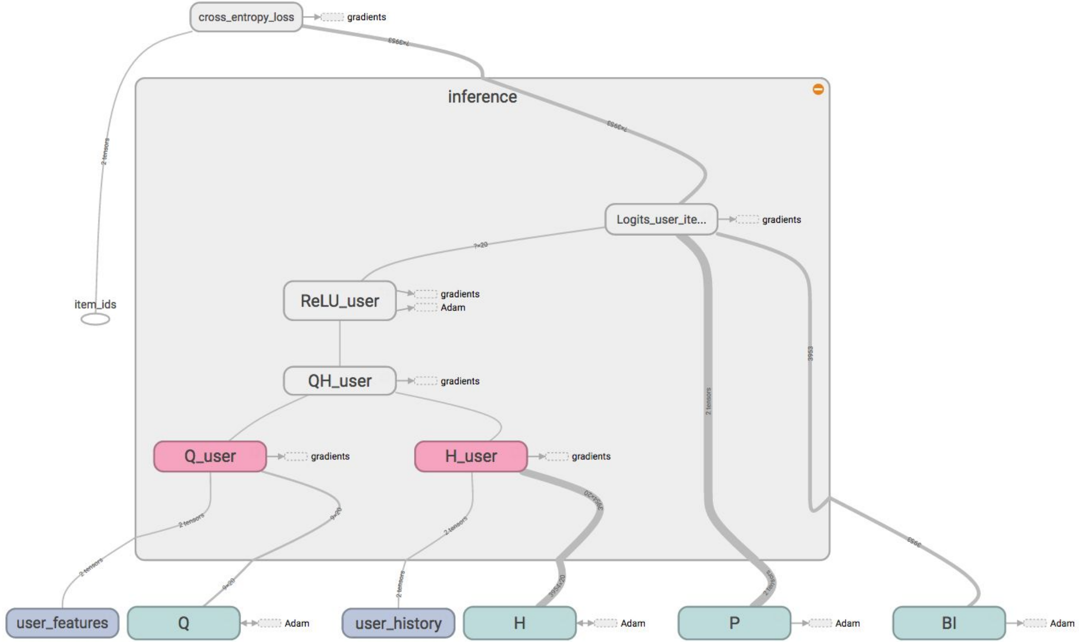
- Predict future interactions from past interactions
- Extreme multi-class classification “à la word2vec”



(b) Predicting future watch

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

Extreme multiclass classification

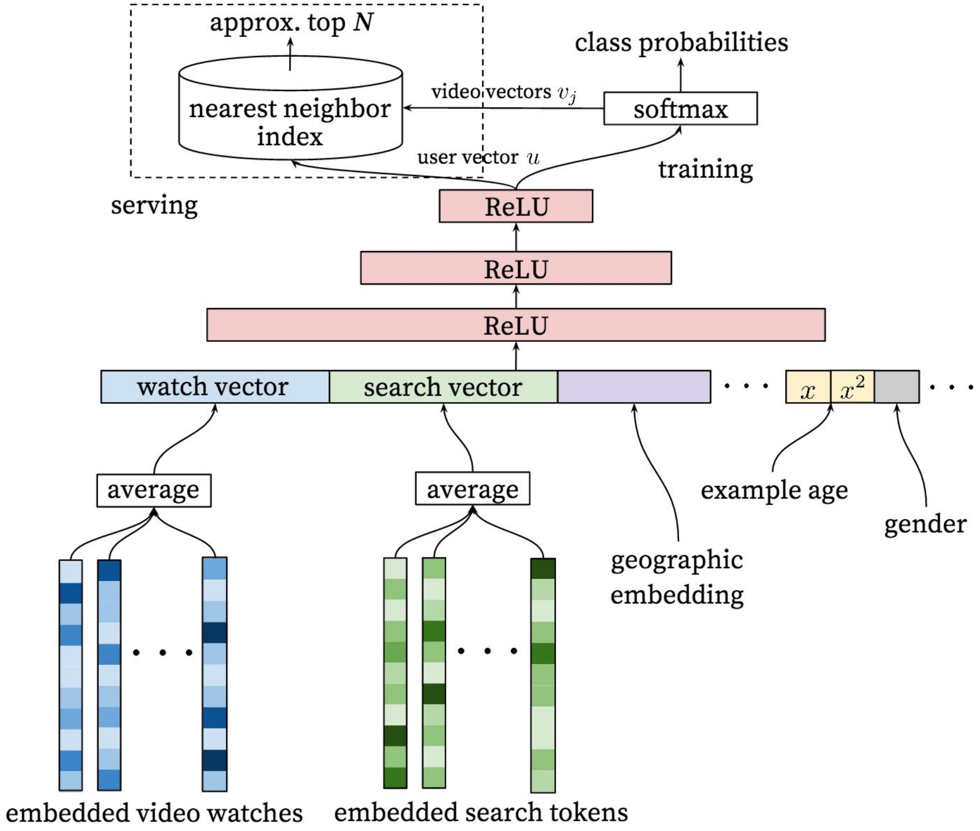


	item	rating	timestamp
user			
1	1193	5	2000-12-31 22:12:40
2	1357	5	2000-12-31 21:38:29
6038	1419	4	2000-04-26 02:06:55

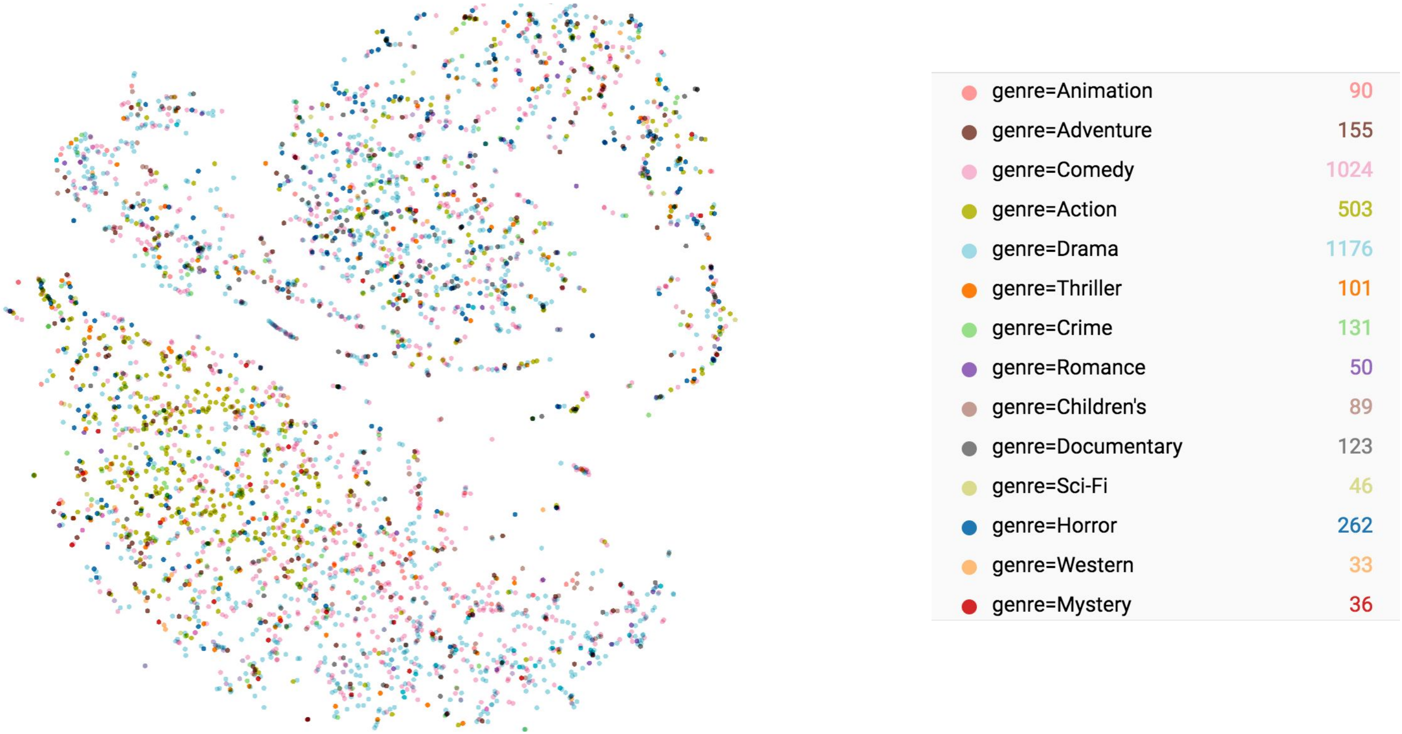
	age=1	age=18	age=25	age=35	age=45	age=50	age=56	gender=F	gender=M
user									
1	1	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	1	0	1
6038	0	0	0	0	0	0	1	1	0

item	0	1	2	3	4	5	6	7	8	9	...	3944	3945	3946	3947	3948	3949	3950	3951	3952
user																				
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
6038	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

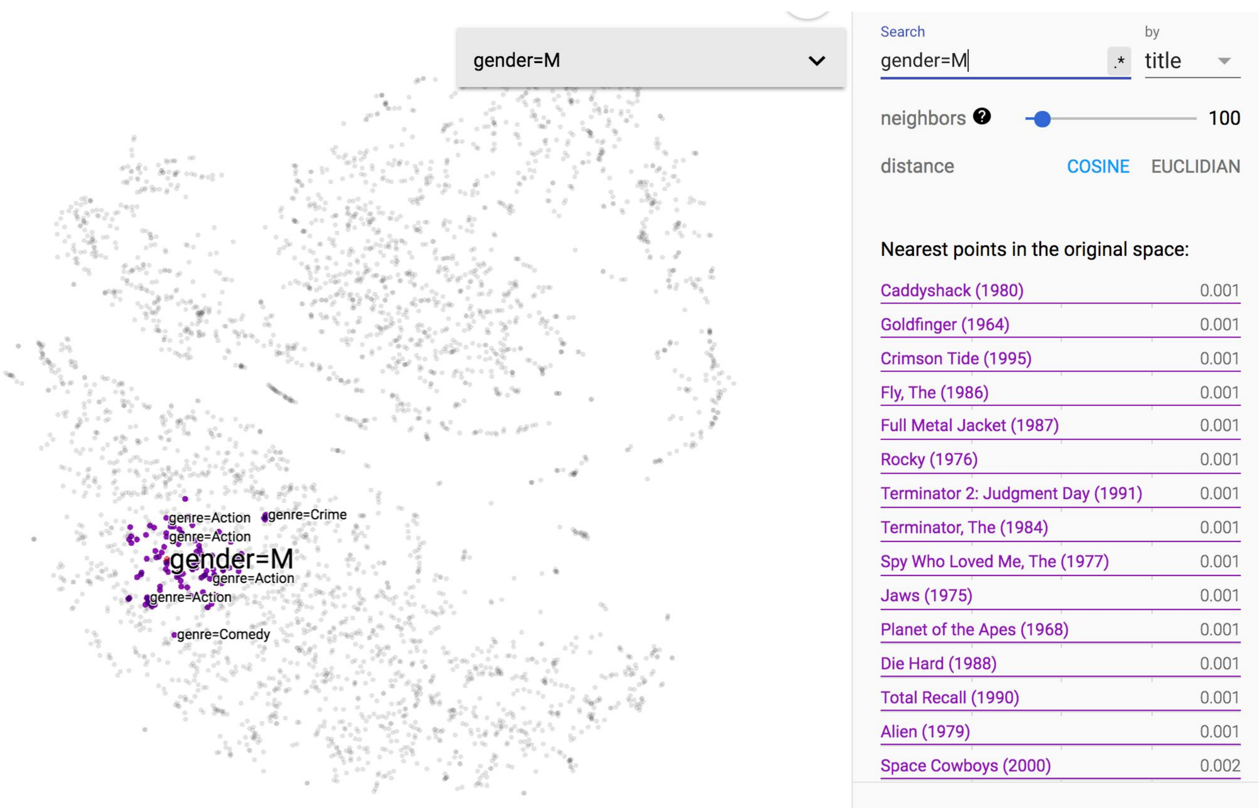
Deep Neural Networks for YouTube Recs



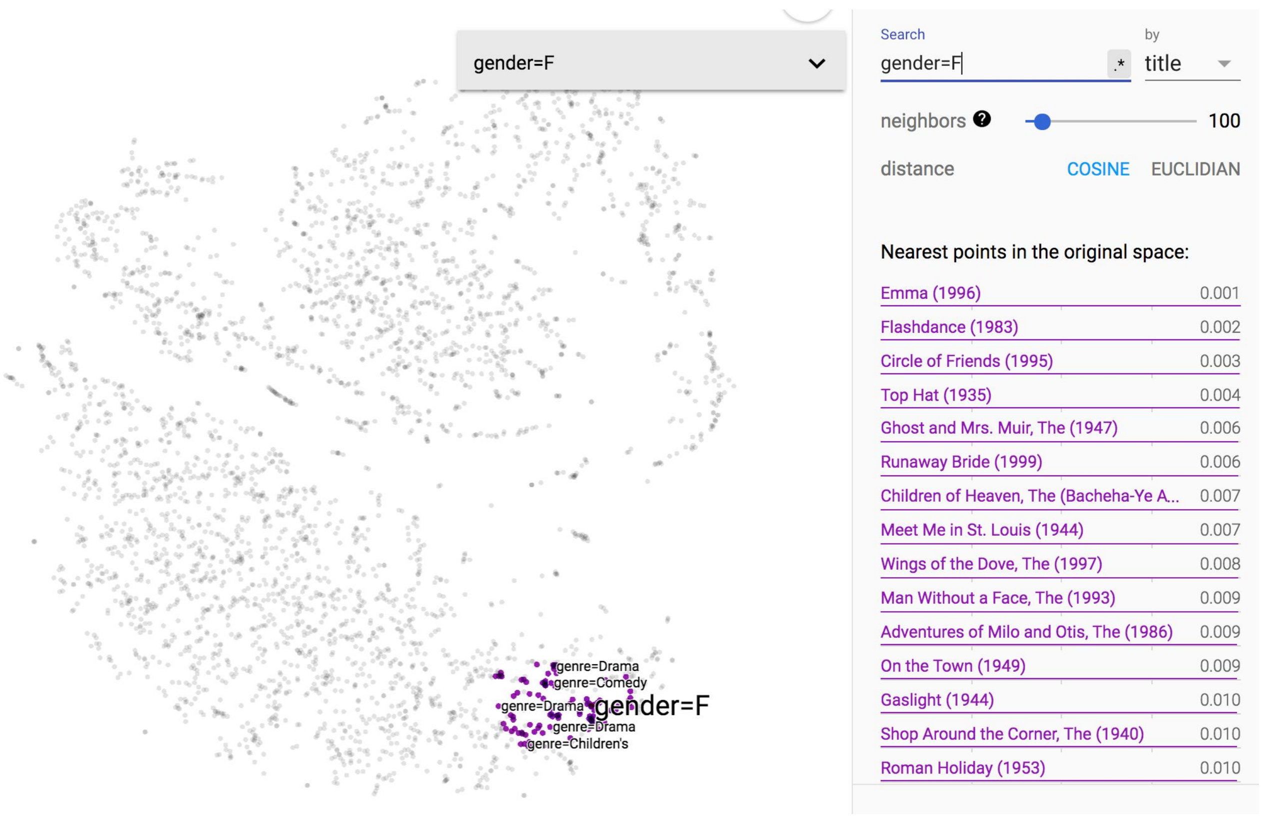
Embeddings visualisation with Tensorboard



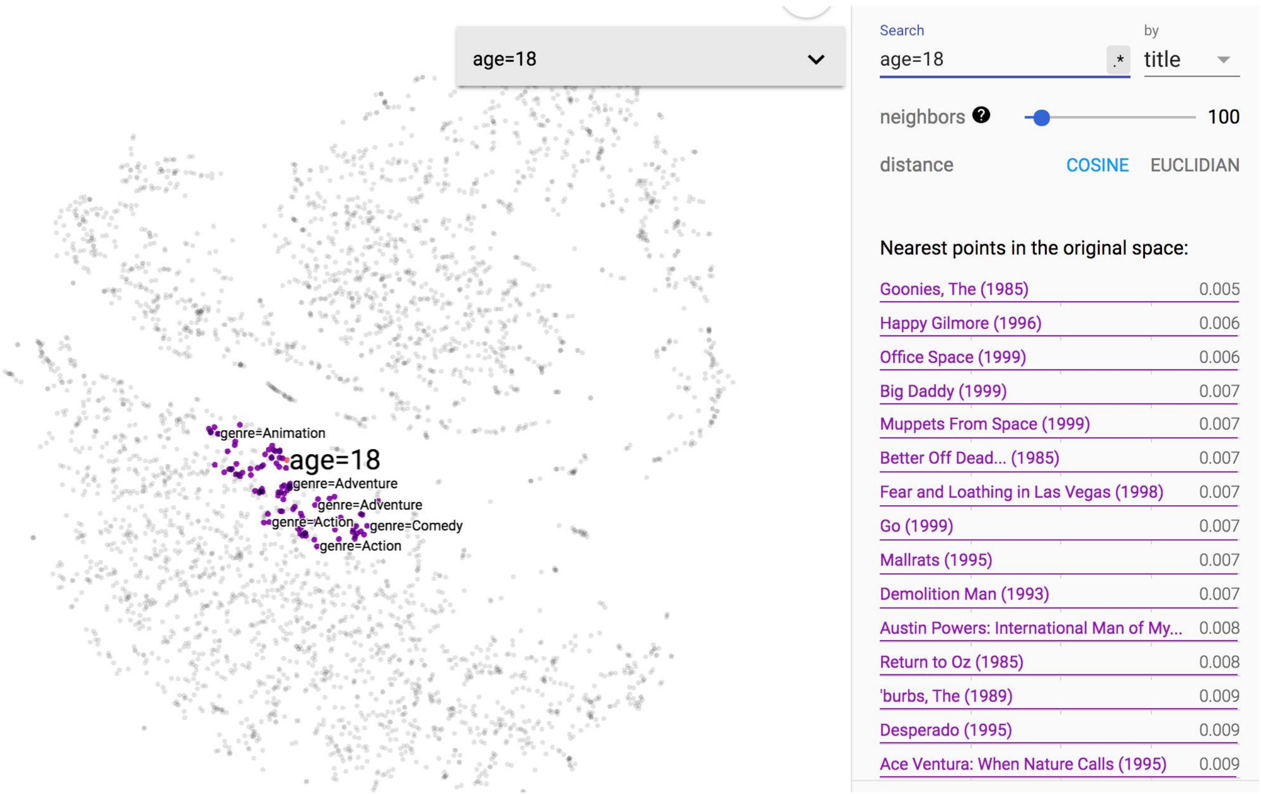
Recommendations as a nearest neighbour search



Recommendations as a nearest neighbour search



Recommendations as a nearest neighbour search



Conclusions and openings

Python ecosystem with TF strikes the right balance

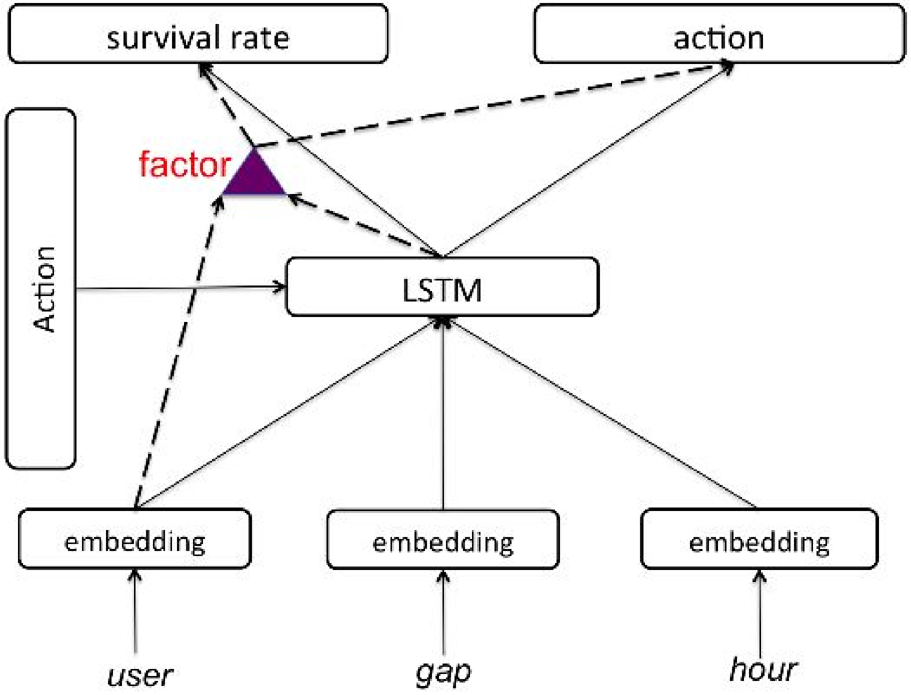
- Possible to implement state of the art of recommender systems
- With guarantees wrt scaling & deployment

Lots more to cover!

- Learning to rank: Neural net approach with a pairwise (BPR) ranking loss
- Wide and deep recommenders, Factorisation machines with Tensorflow…
- Sequence modelling and time-dependencies

Neural Survival Recommender (2017)

- RNNs to model sequential actions
- Jointly predicting **When** and **What** will be the user's next action



More bibliography

- Fast single core ALS http://www.benfrederickson.com/matrix-factorization/
- Large scale ALS harnessing GPUs
- AutoRec: Autoencoders Meet Collaborative Filtering
- Collaborative Deep Learning for Recommender Systems
- Generating Recommendations at Amazon Scale with Apache Spark and Amazon DSSTNE

Links to my notebooks:

- Ml1m-history2multiclass
- Movielens-binary and movielens-fm using lightfm

Collaborative Denoising Auto-Encoders