Learning PyTorch by building a recommender system

Mo Patel @mopatel
Neejole Patel @datajolie

San Jose March 2018

Introductions & Networking

Agenda

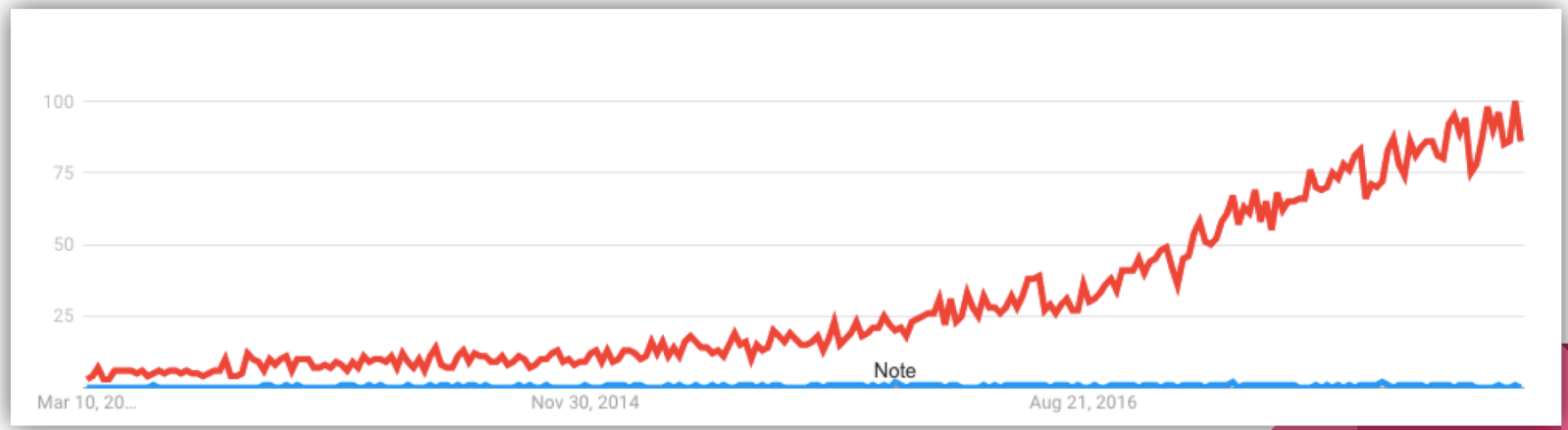
- PyTorch Overview
- Getting started with PyTorch
- Anatomy of a PyTorch model
- Recommender Systems Overview
- PyTorch Data Loading
- Break
- Rec. Sys. Walkthrough
- Rec. Sys. Resources
- PyTorch Deployment
- PyTorch Next

PyTorch Overview

PyTorch Origins

- Torch – Lua
- Chainer
- HIPS Autograd

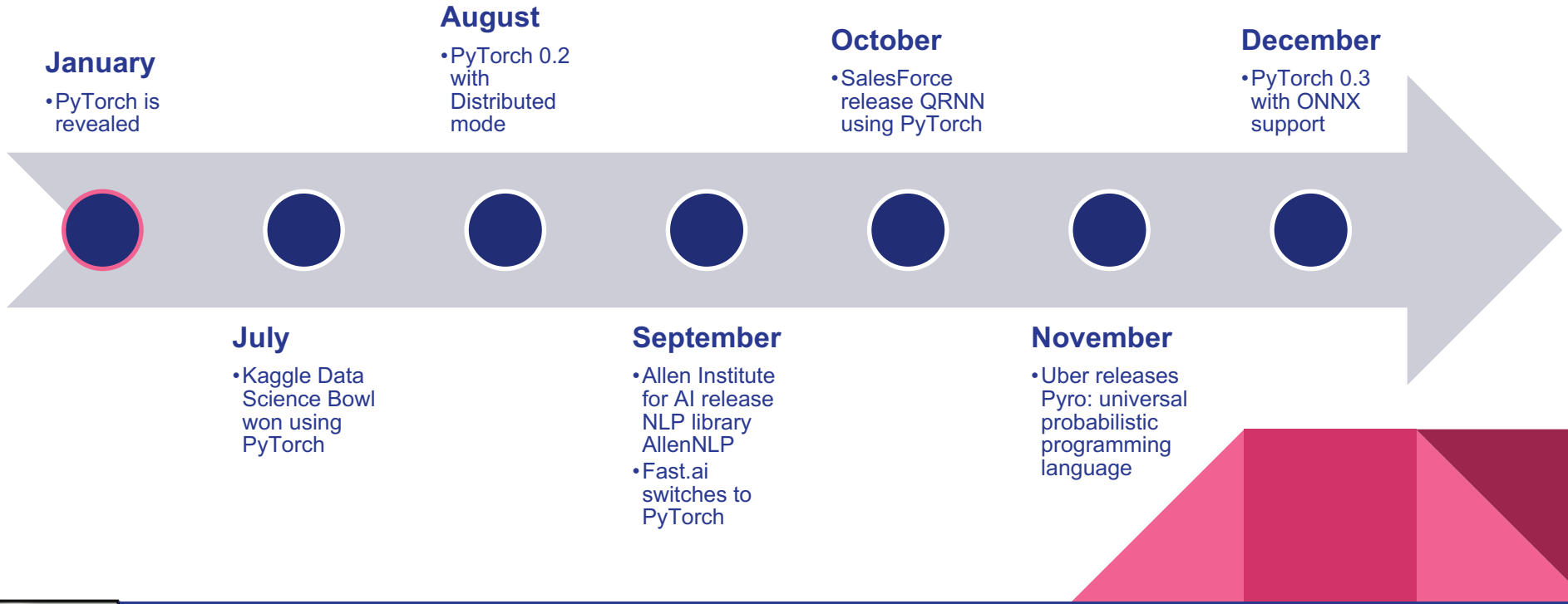
- Need for Dynamic execution library
- Popularity of Python in Machine Learning Community



Google Trends Last Five Years
Terms: Lua Machine Learning Python Machine Learning

http://bit.ly/strata18recsys

PyTorch – Year 2017 in Review



Source: http://pytorch.org/2018/01/19/a-year-in.html

PyTorch – Community Overview

By the numbers

- 87,769 lines of Python code on github that import torch
- 4,959 repositories on Github that mention PyTorch in their name or description
- More than half a million downloads of PyTorch binaries. 651,916 to be precise.
- **5,400 users** wrote **21,500 posts** discussing 5,200 topics on our forums discuss.pytorch.org (http://discuss.pytorch.org/)
- 131 mentions of PyTorch on Reddit's /r/machinelearning since the day of release. In the same period, TensorFlow was mentioned 255 times.

Innovation driver

- In the recent ICLR2018 conference submissions, PyTorch was mentioned in **87 papers**, compared to TensorFlow at 228 papers, Keras at 42 papers, Theano and Matlab at 32 papers.
- Francois Chollet's monthly arxiv.org mentions for frameworks had PyTorch at 72 mentions, with TensorFlow at 273 mentions, Keras at 100 mentions, Caffe at 94 mentions and Theano at 53 mentions.

Source: http://pytorch.org/2018/01/19/a-year-in.html

Use Case Diversity: Research & Applications

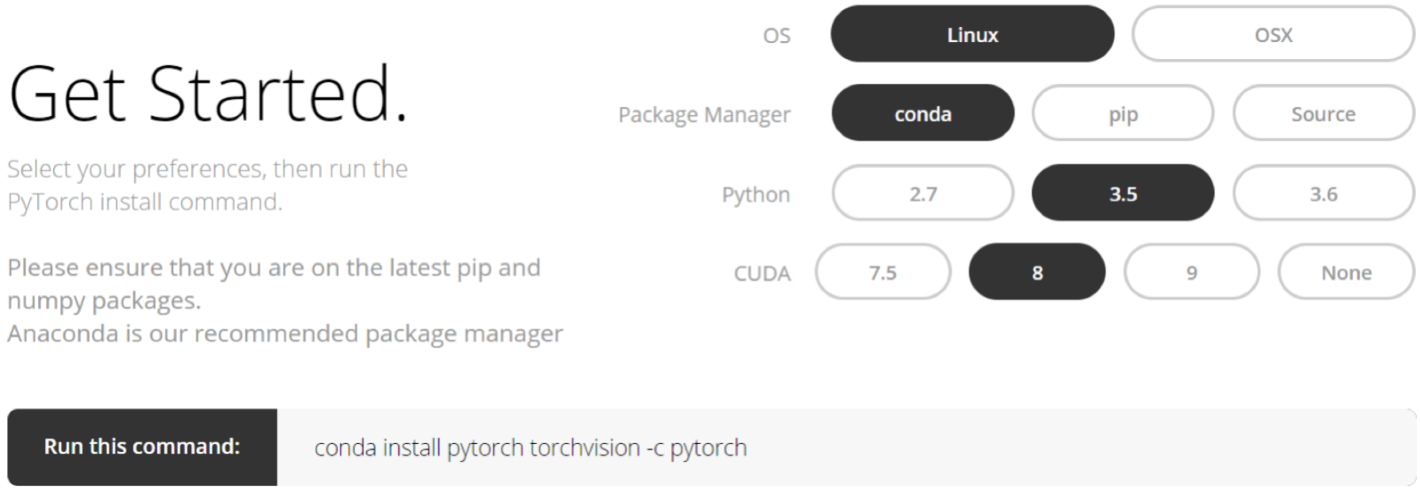
PyTorch Comparison

- PyTorch dynamic computation allows flexibility of input and Python style debugging
- PyTorch best suited for research and prototyping, deployment via ONNX to other frameworks such as Caffe2, Apple CoreML, MXNet, Tensorflow
- Seasoned users of Python data stack (NumPy) can easily transition to PyTorch

Source: https://awni.github.io/pytorch-tensorflow/

Getting started with PyTorch

Installation



Why use PyTorch

- NumPy with power to use GPUs

Why use GPUs?

- Massively parallel computation
- Designed for handling multiple tasks simultaneously
- Greatly decreases training time

Tensors

NumPy ndarrays

- Multidimensional
- Homogenous
- Fixed size items

Tensors

- Same as ndarrays
- Can also be used on GPU

Simple Commands

```
from __future__ import print_function
import torch
```

```
x = torch.Tensor(5,3)
print(x)
```

Output:

```
 0.0000e+00  0.0000e+00 -1.4973e+31
  4.5722e-41 -1.5089e+31  4.5722e-41
-8.9654e+35  4.5722e-41 -8.9653e+35
  4.5722e-41 -1.0677e+36  4.5722e-41
-1.0392e+36  4.5722e-41 -8.9680e+35
[torch.FloatTensor of size 5x3]
```

Simple Commands (cont'd)

```
x = torch.rand(5,3)
print(x)
```

Output:

```
 0.2455  0.1516  0.5319
 0.9866  0.9918  0.0626
 0.0172  0.6471  0.1756
 0.8964  0.7312  0.9922
 0.6264  0.0190  0.0041
[torch.FloatTensor of size 5x3]
```

Simple Commands (cont'd)

```
print(x.size())
```

Output:

```
 torch.Size([5, 3])
```

Simple Commands (cont'd)

```
y = torch.rand(5,3)
print(x+y)
```

Output:

```
 0.2699  0.4096  1.0308
 1.7155  1.0834  0.5545
 0.2245  1.5612  0.8485
 0.9799  1.1686  1.6989
 1.5575  0.4184  0.3967
[torch.FloatTensor of size 5x3]
```

Simple Commands (cont'd)

```
print(torch.add(x,y)
```

Output:

```
 0.2699  0.4096  1.0308
 1.7155  1.0834  0.5545
 0.2245  1.5612  0.8485
 0.9799  1.1686  1.6989
 1.5575  0.4184  0.3967
[torch.FloatTensor of size 5x3]
```

Simple Commands (cont'd)

```
Result = torch.Tensor(5,3)
torch.add(x,y, out=result)
print(result)
```

Output:

```
 0.2699  0.4096  1.0308
 1.7155  1.0834  0.5545
 0.2245  1.5612  0.8485
 0.9799  1.1686  1.6989
 1.5575  0.4184  0.3967
[torch.FloatTensor of size 5x3]
```

Tensor → NumPy Array

```
a = torch.ones(5)
print(a)
b = a.numpy()
print(b)
```

Output a:

```
1
 1
 1
 1
 1
[torch.FloatTensor of size 5]
```

Output b:

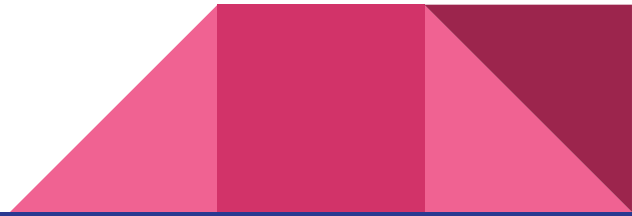
```
[ 1.  1.  1.  1.  1.]
```

NumPy Array → Tensor

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Output:

```
2
 2
 2
 2
 2
[torch.FloatTensor of size 5]

[ 2.  2.  2.  2.  2.]
```



Move Tensors to GPU

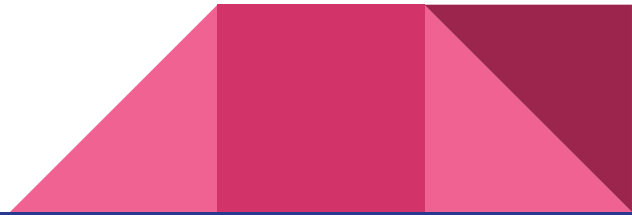
```
if torch.cuda.is_available():
        x = x.cuda()
        y = y.cuda()
        x + y
```

Understating Automatic Differentiation

Automatic Differentiation

Chain rule on steroids

1. Way to determine derivative of function within a function
2. Complex functions can be written as many compositions of simple functions



PyTorch’s autograd Package

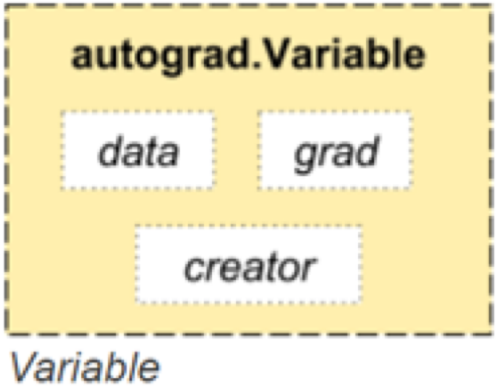
- Provides automatic differentiation on all tensor operations

autograd.Variable

- Central class
  - Wraps a tensor and records the history
  - Supports all tensor operations
- Use .backward
  - Gradients computed automatically

autograd.Variable

- .data
  - Raw tensor
- .grad
  - Gradient
- .grad_fn
  - References a Function
  - User created → grad_fn is None
- .backward()
  - Scalar → no need for argument
  - More elements → specify grad_ouput
    - grad_output = tensor of matching shape



autograd.Variable

```
import torch
from torch.autograd import Variable
```

```
x = Variable(torch.ones(2, 2), requires_grad=True)
print(x)
```

```
Output:
Variable containing:
 1  1
 1  1
[torch.FloatTensor of size 2x2]
```

autograd.Variable

```
y = x + 2
print(y)
```

```
Output:
Variable containing:
 3  3
 3  3
[torch.FloatTensor of size 2x2]
```

autograd.Variable

```
print(y.grad_fn)
```

Output:

```
<AddBackward0 object at 0x7ff91b4f0908>
```

autograd.Variable

```
z = y * y * 3
out = z.mean()
```

```
print(z, out)
```

```
Output:
Variable containing:
 27  27
 27  27
[torch.FloatTensor of size 2x2]
 Variable containing: 27
[torch.FloatTensor of size 1]
```

autograd.Variable

```
out.backward()
```

```
print(x.grad)
```

Output:

```
Variable containing:
 4.5000  4.5000
 4.5000  4.5000
[torch.FloatTensor of size 2x2]
```

autograd.Variable

```
x = torch.randn(3)
x = Variable(x, requires_grad=True)
```

```
y = x * 2
while y.data.norm() < 1000:
    y = y * 2
```

autograd.Variable

```
print(y)
```

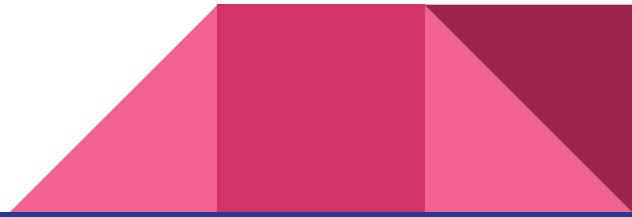
```
Output:
Variable containing:
  164.9539
 -511.5981
-1356.4794
[torch.FloatTensor of size 3]
```

autograd.Variable

```
gradients = torch.FloatTensor([0.1, 1.0, 0.0001])
y.backward(gradients)

print(x.grad)
```

Output:

```
 Variable containing:
   204.8000
  2048.0000
     0.2048
 [torch.FloatTensor of size 3]
```



PyTorch Neural Net & Optimization Package Overview

PyTorch's nn module

- Containers
- Convolution Layers
- Pooling Layers
- Padding Layers
- Non-linear Activations (weighted sum + nonlinearity)
- Non-linear Activations (Other)
- Normalization layers
- Recurrent layers
- Linear layers
- Dropout layers
- Sparse layers
- Distance functions
- Loss functions
- Vision layers
- DataParallel layers (multi-GPU, distributed)
- Utilities

http://pytorch.org/docs/master/nn.html#

PyTorch's optim algorithms

Algorithms		Algorithms	
adadelta	Adaptive Delta	sgd	Stochastic Gradient Descent
adagrad	Adaptive Subgradients	asgd	Averaged Stochastic Gradient Descent
adam	Stochastic Optimization	lbfgs	Limited BFGS
adamax	Adam based on Infinity Norm	rprop	Resilient Backpropagation
sparse_adam	Adam suitable for sparse tensors	rmsprop	Root Mean Squared Propagation

Anatomy of a PyTorch Model

Define the network

```
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


net = Net()
print(net)
```

```
Net(
  (conv1): Conv2d (1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d (6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120)
  (fc2): Linear(in_features=120, out_features=84)
  (fc3): Linear(in_features=84, out_features=10)
)
```

LeNet Image Input

Define the network

```
input = Variable(torch.randn(1, 1, 32, 32))
out = net(input)
print(out)
```

```
Variable containing:
-0.0573 -0.0559 -0.1204  0.1070 -0.0275 -0.0118  0.0264  0.0515 -0.0257  0.0483
[torch.FloatTensor of size 1x10]
```

```
net.zero_grad()
out.backward(torch.randn(1, 10))
```

Always zero out the gradients they are accumulated

The Loss Function and Back Propagation

Example loss calculation

```
output = net(input)
target = Variable(torch.arange(1, 11))
criterion = nn.MSELoss()

loss = criterion(output, target)
print(loss)
```

```
Variable containing:
 38.3962
[torch.FloatTensor of size 1]
```

```
net.zero_grad()     # zeroes the gradient buffers of all parameters

print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

```
conv1.bias.grad before backward
Variable containing:
 0
 0
 0
 0
 0
 0
[torch.FloatTensor of size 6]

conv1.bias.grad after backward
Variable containing:
1.00000e-02 *
 -6.1643
 -2.5201
 -4.5238
  6.8188
 -0.2024
  3.4750
[torch.FloatTensor of size 6]
```

Updating the weights

```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

Training

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.data[0]
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.220
[1,  4000] loss: 1.887
[1,  6000] loss: 1.676
[1,  8000] loss: 1.578
[1, 10000] loss: 1.511
[1, 12000] loss: 1.462
[2,  2000] loss: 1.404
[2,  4000] loss: 1.373
[2,  6000] loss: 1.373
[2,  8000] loss: 1.325
[2, 10000] loss: 1.315
[2, 12000] loss: 1.267
Finished Training
```

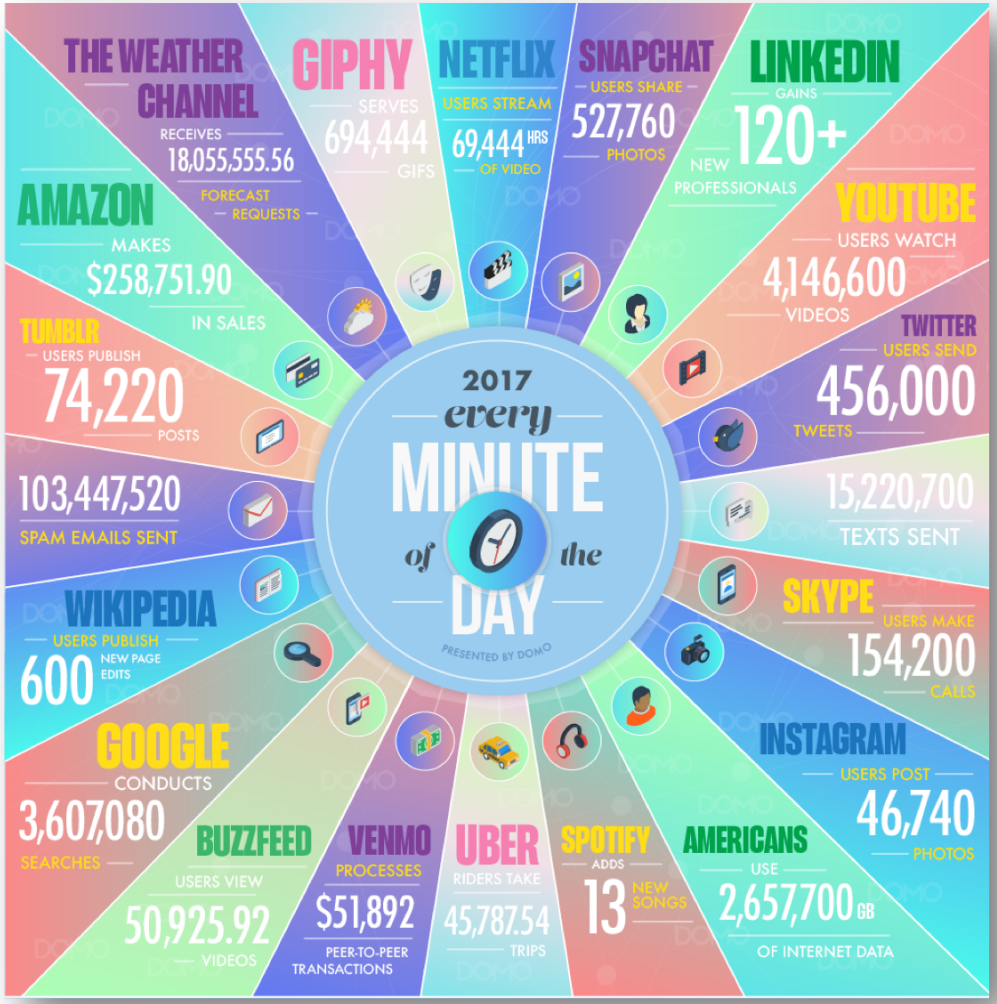
Accuracy

```
correct = 0
total = 0
for data in testloader:
    images, labels = data
    outputs = net(Variable(images))
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 55 %
```

Recommender Systems Overview

http://bit.ly/strata18recsys



Information Explosion

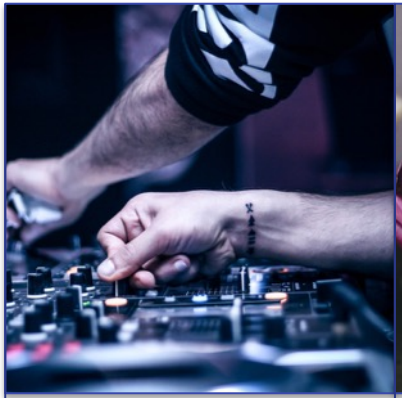
Paradox of Choice

- User Constraints
  - Time/Attention
  - Money
- Which video to show?
- Which song to play?
- Which item to recommend?
- Which job to recommend?
- Which food to recommend?

- Product/Content Developer concern
  - Best User experience
  - Product Stickiness

> “Nobody goes there anymore. It’s too crowded.”
> -Baseball Legend Yogi Berra

Curation is not new



Human

- DJ
- Librarian
- Sommelier
- Concierge

Human + Data

- Artists you may like
- People you may know
- Jobs you may be interested in
- Items you may want to consider

Data + Machines

- Deep & Wide Learning
- Sequence Learning
- Deep Matrix Factorization

Modern day curators

NETFLIX

2/3 of the movies watched are recommended

Google

Recommendations generate 38% more click through on Google News

amazon

70% of Amazon home page is devoted to Recommendations

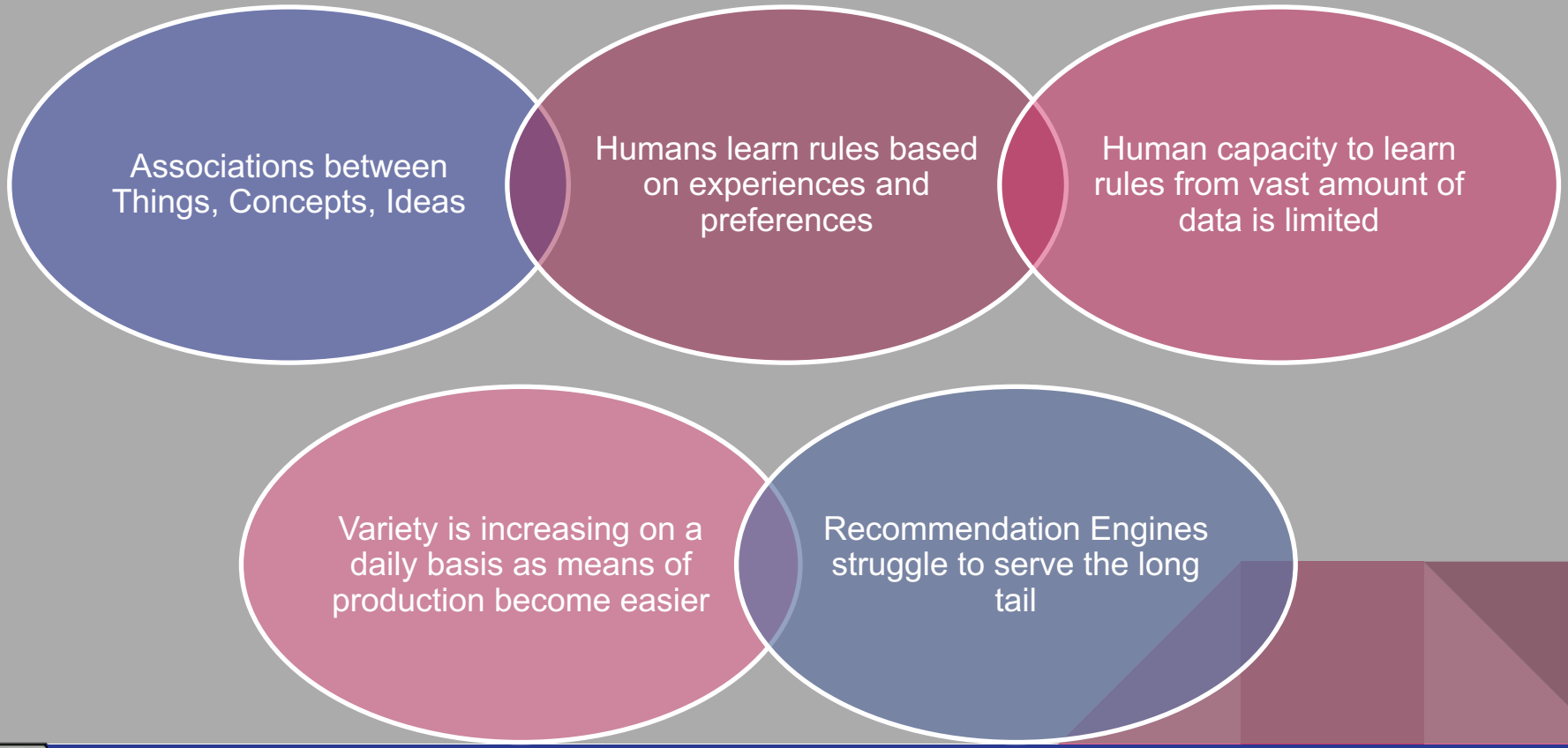
Spotify

“It's scary how well Spotify Disc over Weekly playlists know me. ” @Dave_Horwitz

Recommender Systems (Machine Learning Summer School 2014 @ CMU
https://www.slideshare.net/xamat/recommender-systems-machine-learning-summer-school-2014-cmu
Amazon: https://www.monetate.com/infographic/maximize-online-sales-with-product-recommendations

http://bit.ly/strata18recsys

What are recommender systems?



Recommendation System Challenge

Before

- Few Users, Few Items

Users	Item1	Item2	Item3	Item4	Item5
User1	x		x		x
User2	x	x	x		
User3			x	x	X
User4	x	x			x

Today

- Millions & Billions of Users &

Users	Item1	Item2	Item3	…	m
User1	x		x		x
User2	x	x	x		
…			x	x	X
n	x	x			x

Sparcity

> *A family of methods that enable filtering through large observation and information space in order to provide recommendations in the information space [whether] that user does [or does] not have any observation, where the information space is all of the available items that the user could choose or select and observation space is what user experienced or observed so far.*

Bugra Akyildiz on Recommender Systems

https://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/

Techniques

Collaborative Filtering via Matrix Factorization	Deep & Wide Learning	Deep Matrix Factorization	Sequence Based Models
Predicting future user interaction by studying historical user-item interaction	Wide: Linear model similar to matrix factorization approach Deep: Feed Forward Neural Network to learn from categorical inputs projected into embeddings	Provide trained embeddings of users and items to Deep Neural Network for rating prediction	Use historical interaction data to predict future interaction

PyTorch Data Loading

Implicit and Explicit Data

Implicit

- Behaviors
  - Clicks
  - Purchases
  - Listens
  - Visits
- Challenges
  - Fake Data/Click Farms
  - Shilling Attacks
  - Accidents & Life Events

Explicit

- Actions
  - Ratings
  - Reviews
  - Surveys
  - Likes
  - Hearts
- Not applicable for all problems
- Missing data no random

Movie Lens Dataset from grouplens

- MovieLens 20M Dataset
- 20 million ratings
- 27,000 movies
- 138,000 users
- 465,000 tag applications

- userIds
- movieIds
- Ratings: userId, movieId, rating, timestamp
- Tags: userId, movieId, tag, timestamp
- Movies: movieId, title, genres
- Links: movieId, imdbId, tmdbId

Several Approaches to loading data in PyTorch

- Creating a custom Data Loader
- Fast.ai CollabFilterDataset
  - https://github.com/fastai/fastai/blob/master/fastai/column_data.py
- Spotlight get_movielens_dataset
  - https://github.com/maciejkula/spotlight/blob/master/spotlight/datasets/movielens.py

Custom PyTorch Data Loader using Dataset & DataLoader utilities

- Requires inheriting PyTorch Dataset and overriding `__len__` & `__getitem__` methods

```python
from torch.utils.data import Dataset
import torch


class Hdf5Dataset(Dataset):
    """Custom Dataset for loading entries from HDF5 databases"""

    def __init__(self, h5_path, transform=None):

        self.h5f = h5py.File(h5_path, 'r')
        self.num_entries = self.h5f['labels'].shape[0]
        self.transform = transform

    def __getitem__(self, index):

        features = self.h5f['features'][index]
        label = self.h5f['labels'][index]
        if self.transform is not None:
            features = self.transform(features)
        return features, label

    def __len__(self):
        return self.num_entries
```

```python
from torch.utils.data import DataLoader


train_dataset = Hdf5Dataset(h5_path='iris.h5',
                            transform=None)

train_loader = DataLoader(dataset=train_dataset,
                          batch_size=50,
                          shuffle=True,
                          num_workers=4)
```


https://github.com/rasbt/deep-learning-book/blob/master/code/model_zoo/pytorch_ipynb/custom-data-loader-csv.ipynb

Recommender Systems Walkthrough

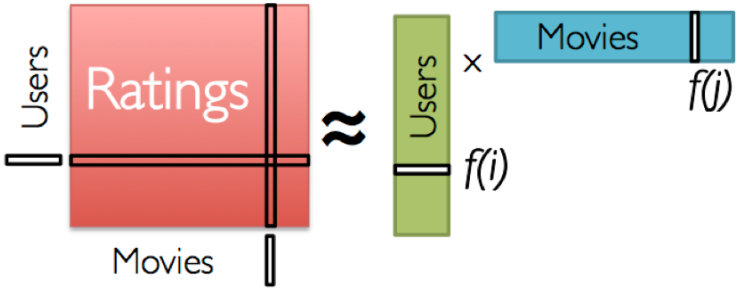
Matrix Factorization

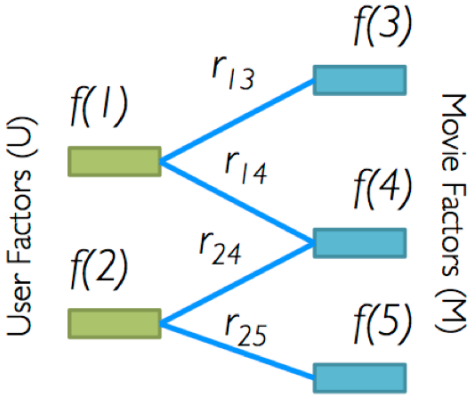
Overview of Collaborative Filtering

- Given items learn rules based on past interactions
  - Wine & Cheese
  - Rain & Umbrella
- Multiple Approaches: Memory Based (Cosine Distance), Model Based (Matrix Factorization)
  - Memory based approach limited to dense datasets
  - Model based approach allows scalability as data becomes sparse with Increase in interactions from more users and more items

Model based CF via Matrix Factorization

Low-Rank Matrix Factorization:





Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Matrix Factorization in PyTorch

1. Setup

```
import numpy as np
from scipy.sparse import rand as sprand
import torch
from torch.autograd import Variable


# Make up some random explicit feedback ratings
# and convert to a numpy array
n_users = 1000
n_items = 1000
ratings = sprand(n_users, n_items,
                 density=0.01, format='csr')
ratings.data = (np.random.randint(1, 5,
                                  size=ratings.nnz)
                            .astype(np.float64))
ratings = ratings.toarray()
```

2. Model

```
class MatrixFactorization(torch.nn.Module):

    def __init__(self, n_users, n_items, n_factors=20):
        super().__init__()
        self.user_factors = torch.nn.Embedding(n_users,
                                               n_factors,
                                               sparse=True)
        self.item_factors = torch.nn.Embedding(n_items,
                                               n_factors,
                                               sparse=True)

    def forward(self, user, item):
        return (self.user_factors(user) *
self.item_factors(item)).sum(1)
```

3. Model Settings

```
model = MatrixFactorization(n_users, n_items, n_factors=20)
loss_func = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(),lr=1e-6) # learning rate
```

Matrix Factorization in PyTorch

4. Training

```
# Sort our data
rows, cols = ratings.nonzero()
p = np.random.permutation(len(rows))
rows, cols = rows[p], cols[p]


for row, col in zip(*(rows, cols)):
    # Turn data into variables
    rating = Variable(torch.FloatTensor([ratings[row,
col]]))
    row = Variable(torch.LongTensor([np.long(row)]))
    col = Variable(torch.LongTensor([np.long(col)]))

    # Predict and calculate loss
    prediction = model(row, col)
    loss = loss_func(prediction, rating)

    # Backpropagate
    loss.backward()

    # Update the parameters
    optimizer.step()
```

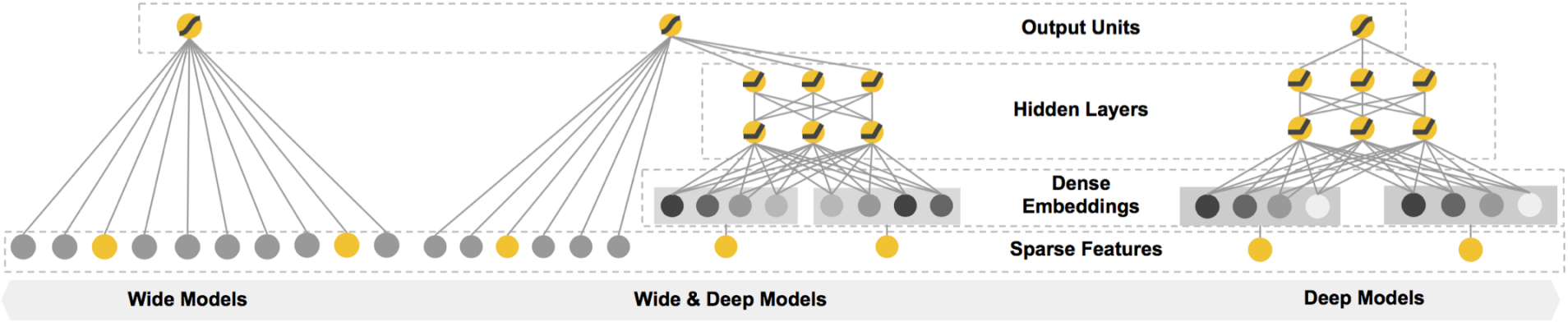
Collaborative Filtering Matrix Factorization Resources

- Fast.ai Collaborative Filtering Tutorial: http://course.fast.ai/lessons/lesson5.html
- Explicit Factorization Models using Spotlight built on PyTorch: https://maciejkula.github.io/spotlight/factorization/explicit.html

Recommender Systems Technique

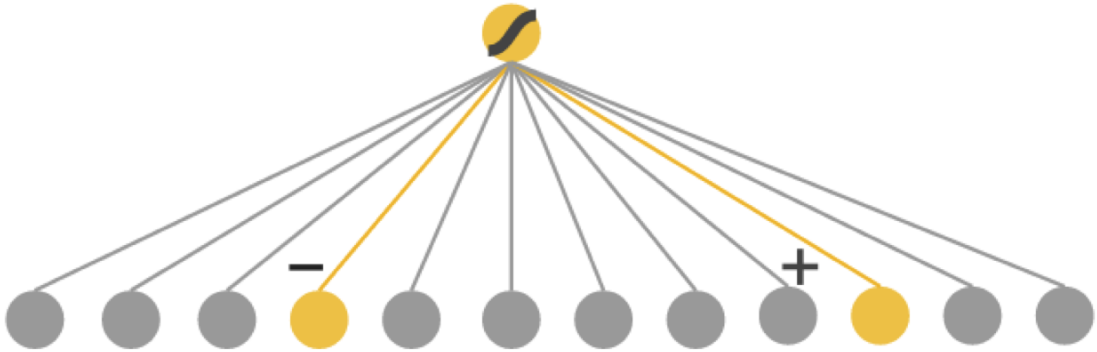
Deep & Wide Learning

Wide & Deep Learning



https://research.googleblog.com/2016/06/wide-deep-learning-better-together-with.html

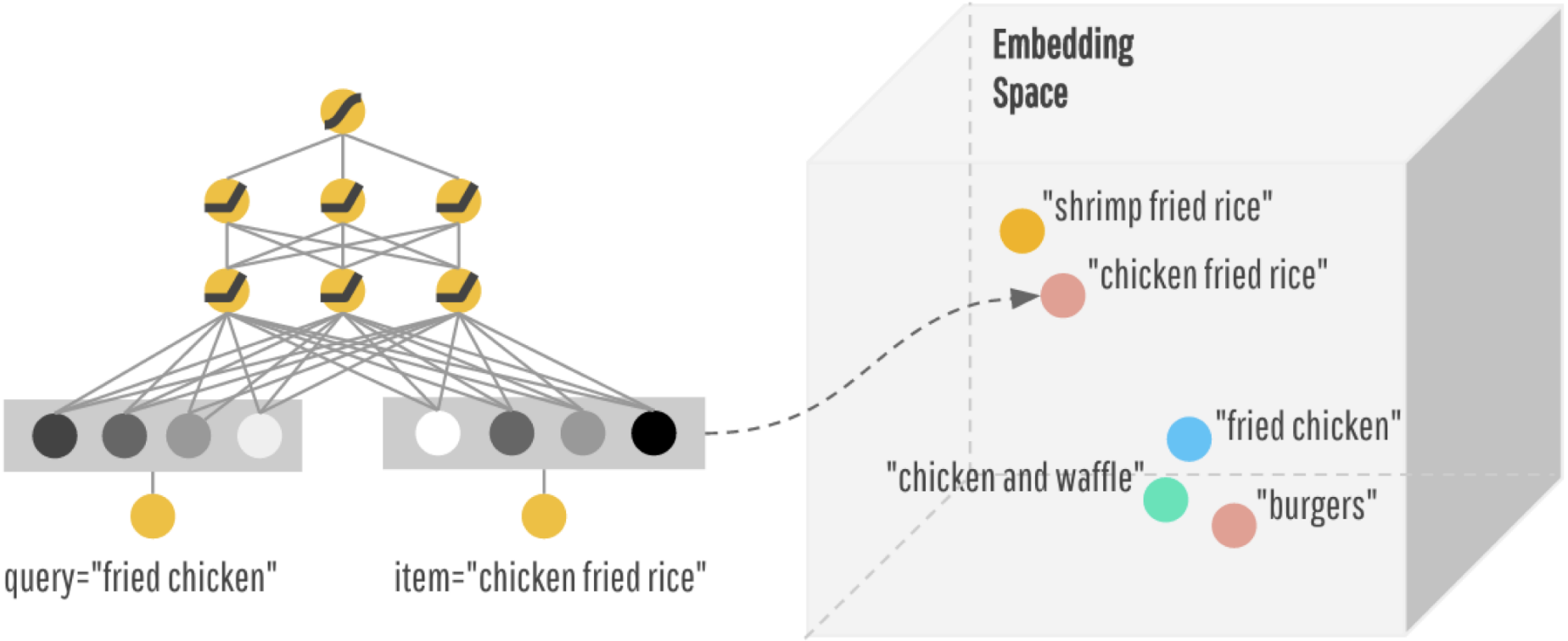
Wide Model



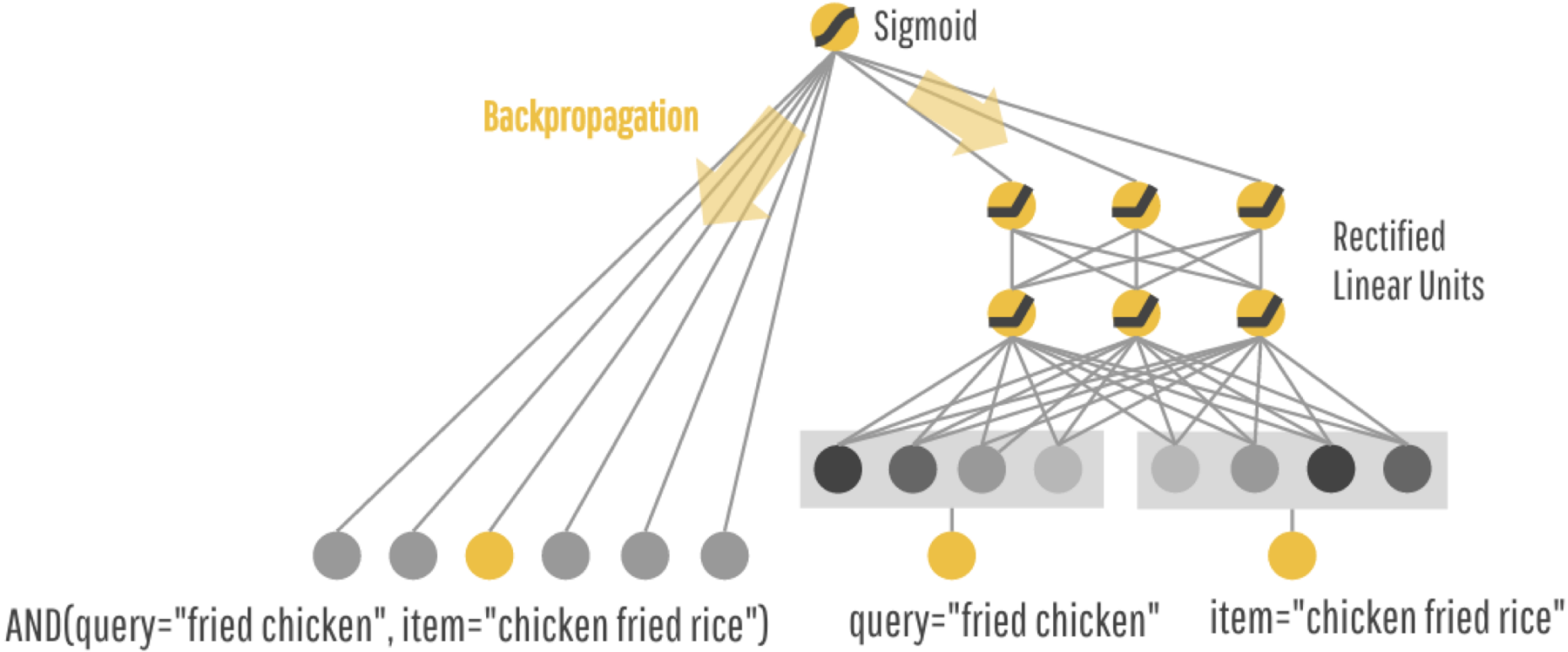
AND(query="fried chicken", item="chicken fried rice")

AND(query="fried chicken", item="chicken and waffle")

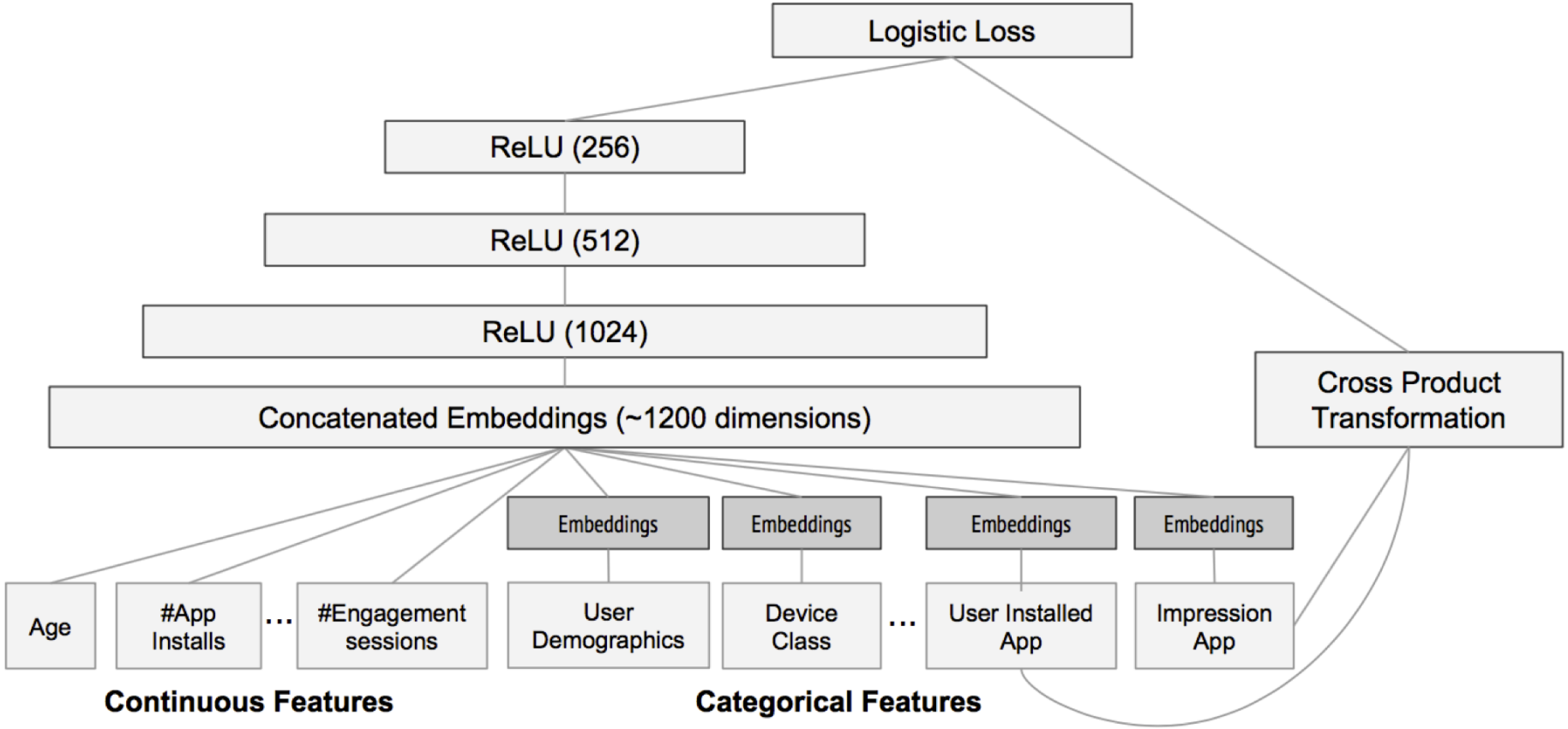
Deep Model



Wide & Deep Model



Wide & Deep Model for Google Play Store



PyTorch Wide & Deep Learning

- https://github.com/jrzaurin/Wide-and-Deep-PyTorch/blob/master/demo2_building_blocks.ipynb

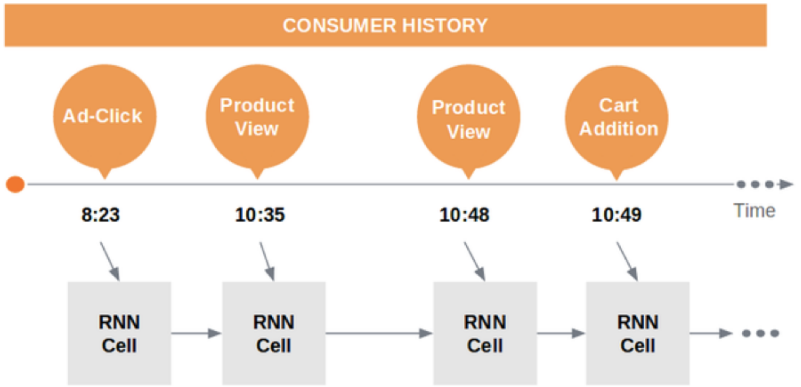
Recommender Systems Technique

Sequence Modeling

Spotlight Recommender System based Sequence Modeling

```
from spotlight.cross_validation import user_based_train_test_split
from spotlight.datasets.synthetic import generate_sequential
from spotlight.evaluation import sequence_mrr_score
from spotlight.sequence.implicit import ImplicitSequenceModel

dataset = generate_sequential(num_users=100,
                              num_items=1000,
                              num_interactions=10000,
                              concentration_parameter=0.01,
                              order=3)

train, test = user_based_train_test_split(dataset)

train = train.to_sequence()
test = test.to_sequence()

model = ImplicitSequenceModel(n_iter=3,
                              representation='cnn',
                              loss='bpr')
model.fit(train)

mrr = sequence_mrr_score(model, test)
```

https://github.com/maciejkula/spotlight/tree/master/examples/movielens_sequence



Consumer History graphic from Zalando: https://jobs.zalando.com/tech/blog/deep-learning-in-production-for-predicting-consumer-behavior/

Deploying PyTorch Models

Deploying PyTorch Models: ONNX

- ONNX: Open Neural Network eXchange
  - Framework fragmentation: Project goal to allow researchers and developers to port models from one framework to another allowing interoperability
  - Currently supports some of Caffe2, PyTorch, Microsoft Cognitive Toolkit, Apache MXNet
- Currently PyTorch only supports model export not import
- Resources:
  - ONNX: https://onnx.ai/ & https://github.com/onnx/tutorials
  - PyTorch: http://pytorch.org/docs/master/onnx.html

PyTorch AlexNet model to ONNX

```
from torch.autograd import Variable
import torch.onnx
import torchvision

dummy_input = Variable(torch.randn(10, 3, 224, 224)).cuda()
model = torchvision.models.alexnet(pretrained=True).cuda()

# providing these is optional, but makes working with the
# converted model nicer.
input_names = [ "learned_%d" % i for i in range(16) ] + [ "actual_input_1" ]
output_names = [ "output1" ]

torch.onnx.export(model, dummy_input,
          "alexnet.proto", verbose=True,
          input_names=input_names, output_names=output_names)
```

Readable Model

```
    # All parameters are encoded explicitly as inputs.  By convention,
    # learned parameters (ala nn.Module.state_dict) are first, and the
    # actual inputs are last.
    graph(%learned_0 : Float(10, 3, 224, 224)
          %learned_1 : Float(64, 3, 11, 11)
          # The definition sites of all variables are annotated with type
          # information, specifying the type and size of tensors.
          # For example, %learned_2 is a 192 x 64 x 5 x 5 tensor of floats.
          %learned_2 : Float(64)
          %learned_3 : Float(192, 64, 5, 5)
          # ---- omitted for brevity ----
          %learned_14 : Float(4096)
          %learned_15 : Float(1000, 4096)
          %actual_input_1 : Float(1000)) {
      # Every statement consists of some output tensors (and their types),
      # the operator to be run (with its attributes, e.g., kernels, strides,
      # etc.), its input tensors (%learned_0, %learned_1, %learned_2)
      %17 : Float(10, 64, 55, 55) = Conv[dilations=[1, 1], group=1, kernel_shape=[11, 11], pads=[2, 2, 2, 2], strides=[4, 4]](%learned_0, %learned_1, %learned_2), scope:
AlexNet/Sequential[features]/Conv2d[0]
      %18 : Float(10, 64, 55, 55) = Relu(%17), scope: AlexNet/Sequential[features]/ReLU[1]
      %19 : Float(10, 64, 27, 27) = MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%18), scope: AlexNet/Sequential[features]/MaxPool2d[2]
      # ---- omitted for brevity ----
      %29 : Float(10, 256, 6, 6) = MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%28), scope: AlexNet/Sequential[features]/MaxPool2d[12]
      %30 : Float(10, 9216) = Flatten[axis=1](%29), scope: AlexNet
      # UNKNOWN_TYPE: sometimes type information is not known.  We hope to eliminate
      # all such cases in a later release.
      %31 : Float(10, 9216), %32 : UNKNOWN_TYPE = Dropout[is_test=1, ratio=0.5](%30), scope: AlexNet/Sequential[classifier]/Dropout[0]
      %33 : Float(10, 4096) = Gemm[alpha=1, beta=1, broadcast=1, transB=1](%31, %learned_11, %learned_12), scope: AlexNet/Sequential[classifier]/Linear[1]
      # ---- omitted for brevity ----
      %output1 : Float(10, 1000) = Gemm[alpha=1, beta=1, broadcast=1, transB=1](%38, %learned_15, %actual_input_1), scope: AlexNet/Sequential[classifier]/Linear[6]
      # Finally, a network returns some tensors
      return (%output1);
    }
```

PyTorch AlexNet model to ONNX

```
import onnx
# Load the ONNX model

model = onnx.load("alexnet.proto")
# Check that the IR is well formed
onnx.checker.check_model(model)

# Print a human readable representation of the graph
onnx.helper.printable_graph(model.graph)

import caffe2.python.onnx.backend as backend
import numpy as np

rep = backend.prepare(model, device="CUDA:0") # or "CPU"
# For the Caffe2 backend:
#     rep.predict_net is the Caffe2 protobuf for the network
#     rep.workspace is the Caffe2 workspace for the network
#       (see the class caffe2.python.onnx.backend.Workspace)
outputs = rep.run(np.random.randn(10, 3, 224, 224).astype(np.float32))
# To run networks with more than one input, pass a tuple
# rather than a single numpy ndarray.

print(outputs[0])
```

PyTorch Next

What is next for PyTorch?

- Tensor Comprehensions now available in PyTorch speed up innovation in Deep Learning by allowing researchers to scale new ideas to large models

```
def avgpool(float(B, C, H, W) input) -> (output) {{
    output(b, c, h, w) += input(b, c, h * {sH} + kh, w * {sW} + kw)
                          where kh in 0:{kH}, kw in 0:{kW}
}}
```

Tensor Comprehension for 2D Average Pooling

Learning PyTorch
by building a recommender system

Discussion