

Theoretical Introduction and Applications of Machine and Deep Learning

Introduction to SageMaker
and PyTorch Lightning

Who am I?

Luca Bianchi

AWS Hero, passionate about
serverless and **machine learning**



github.com/aletheia



<https://it.linkedin.com/in/lucabianchipavia>



[@bianchiluca](https://twitter.com/bianchiluca)



<https://speakerdeck.com/aletheia>



www.ai4devs.io

AWS
community
builder

aws
serverless
HERO

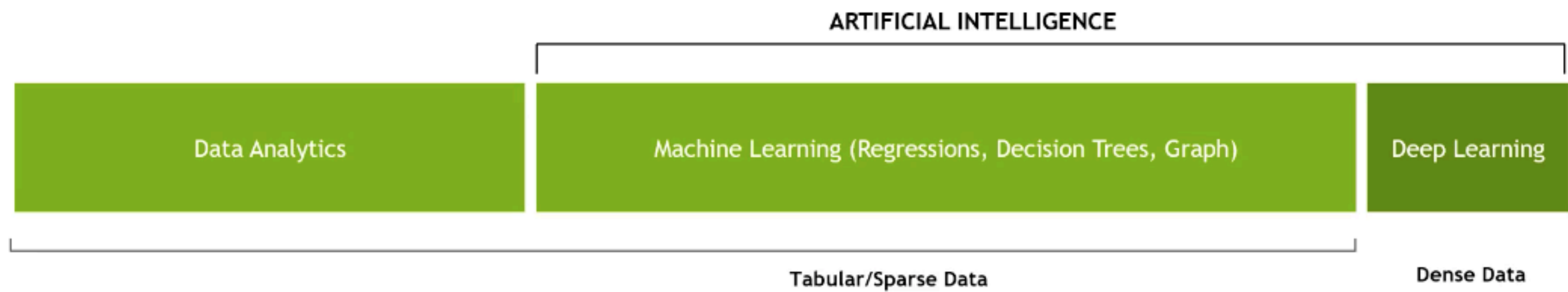


*“deep learning is a great phrase,
it seems so **deep**”*

understanding your problem

Beyond Deep Learning





Structured data doesn't need deep learning, but it could be “just” a machine learning or a big data problem



2.2 exabytes (2.2B GB) of data created daily - McKinsey
\$274B annual revenue by 2022 for big data and business analytics - IDC

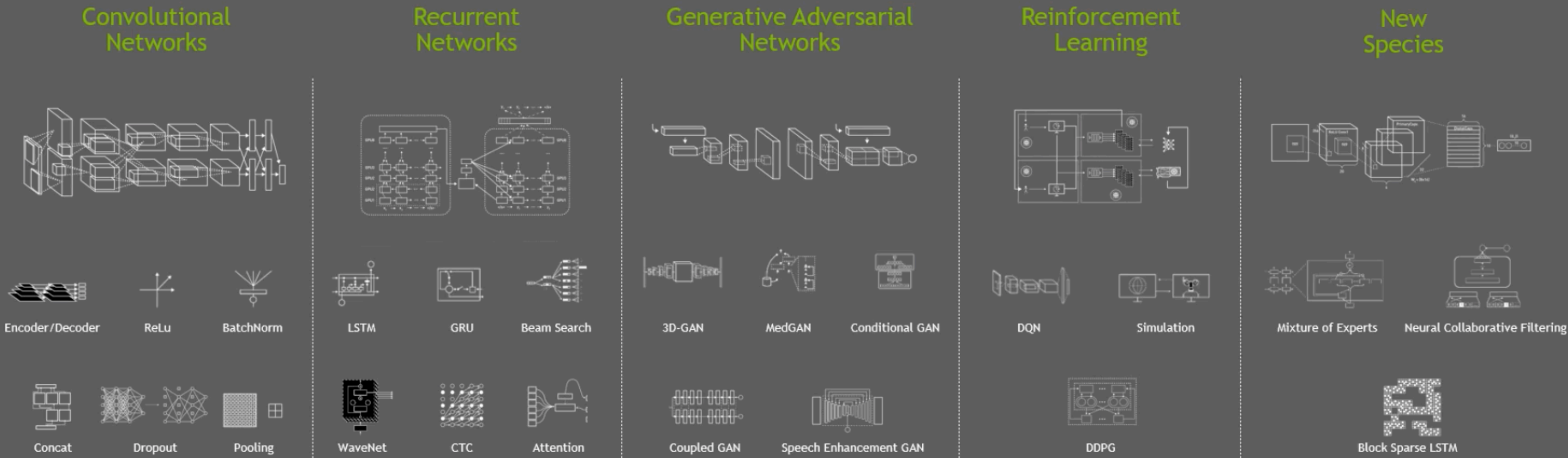
What problem are you solving?

Unstructured data type, deep learning task, and business domain

INPUTS	BUSINESS QUESTIONS	AI / DL TASK	EXAMPLE OUTPUTS		
			HEALTHCARE	RETAIL	MANUFACTURING
<div> Text Data</div> <div> Images</div> <div> Audio</div> <div> Video</div>	Is “it” present or not?	Detection	Cancer Detection	Targeted Ads	Defect Detection
	What type of thing is “it”?	Classification	Transcription / Image Classification	Basket Analysis	Material Sorting
	To what extent is “it” present?	Segmentation	Tumor Size & Shape Analysis	360° Customer Views	Autonomous Navigation
	What is the likely outcome ?	Prediction	Survivability Prediction	Sentiment & Behavior Recognition	Predictive Maintenance
	What will likely satisfy the objective?	Recommendations	Therapy Recommendation	Recommendation Engine	Supply Chain Optimization

Deep Learning Species

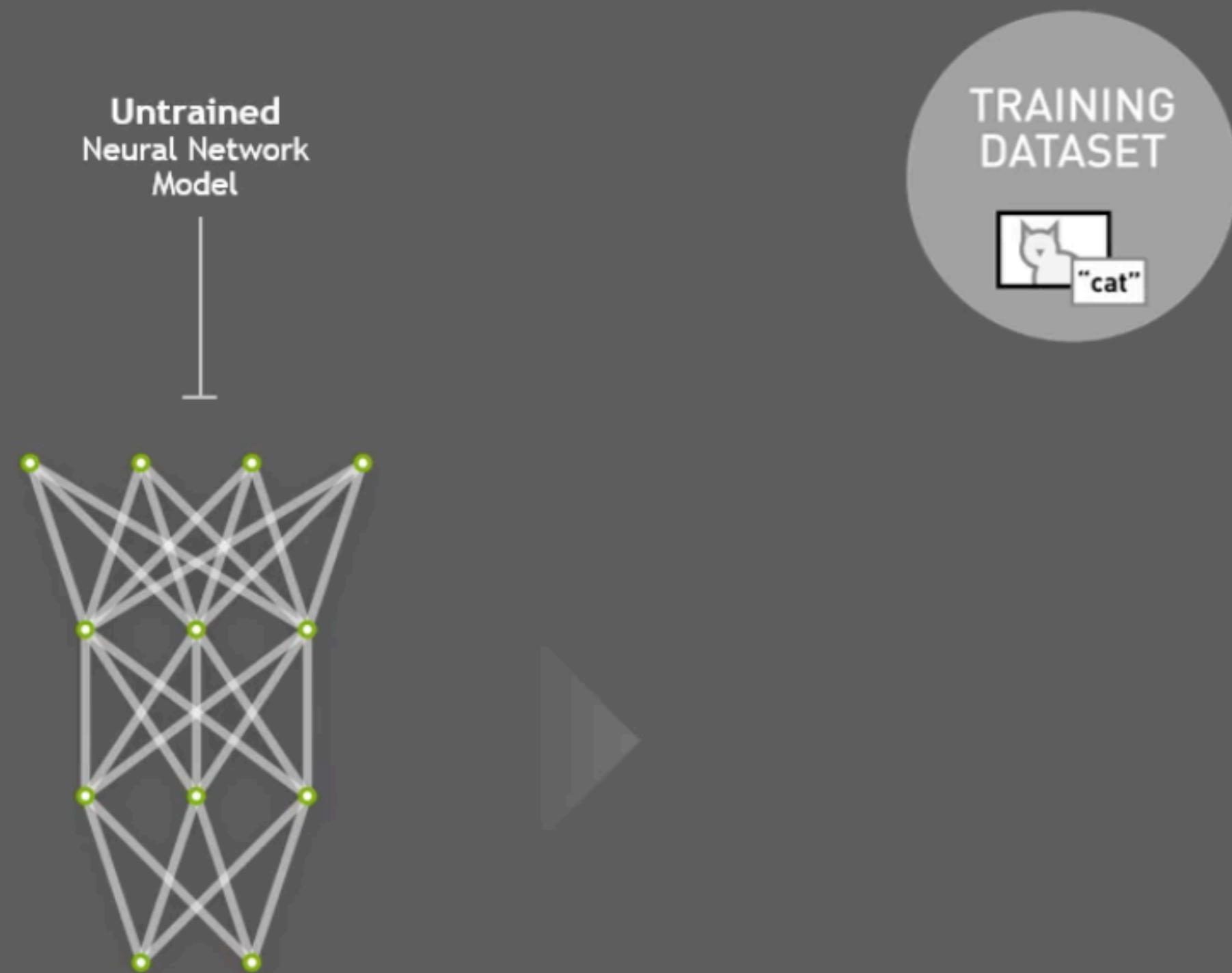
A Cambrian Explosion



DEEP LEARNING APPLICATION DEVELOPMENT



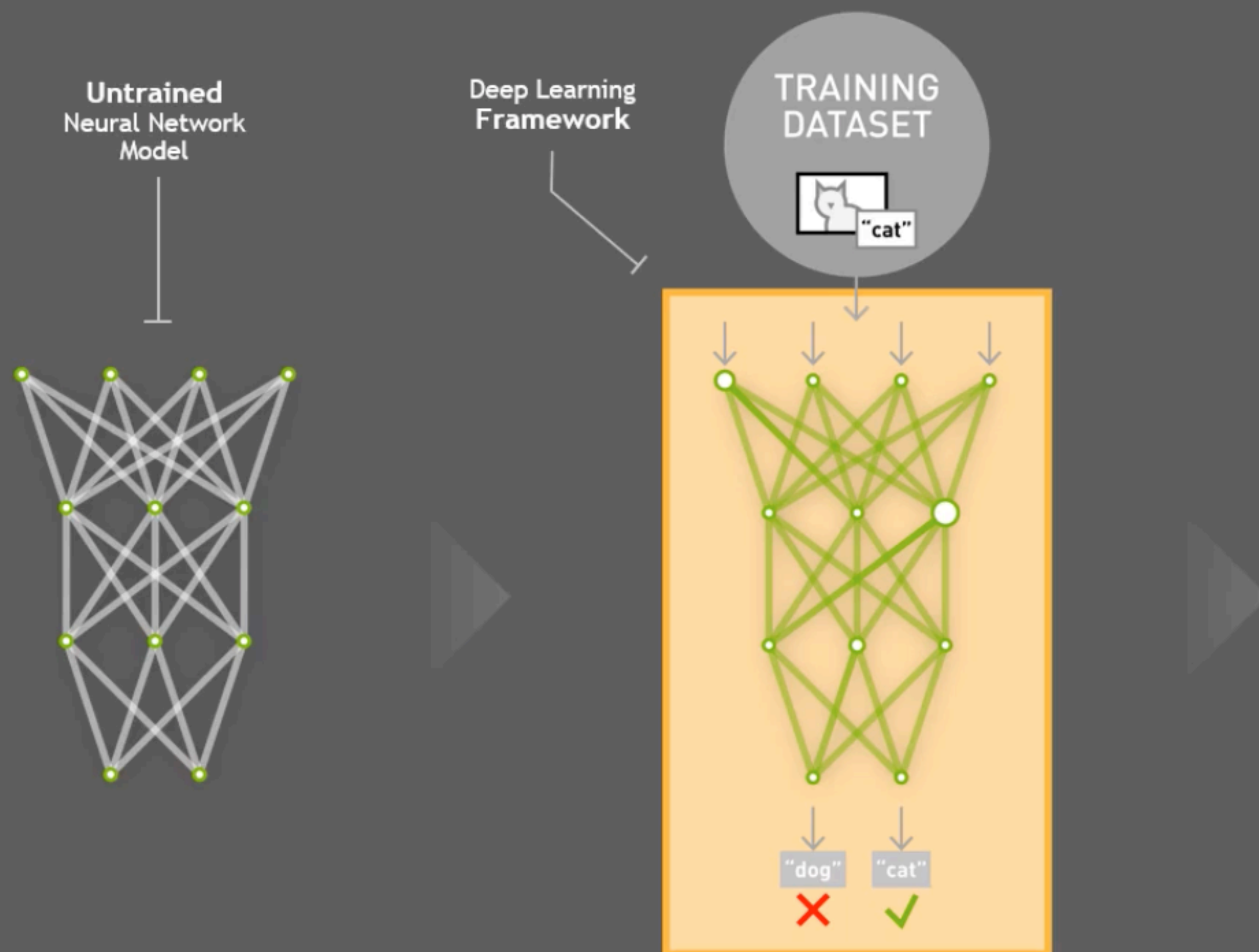
DEEP LEARNING APPLICATION DEVELOPMENT



DEEP LEARNING APPLICATION DEVELOPMENT

TRAINING

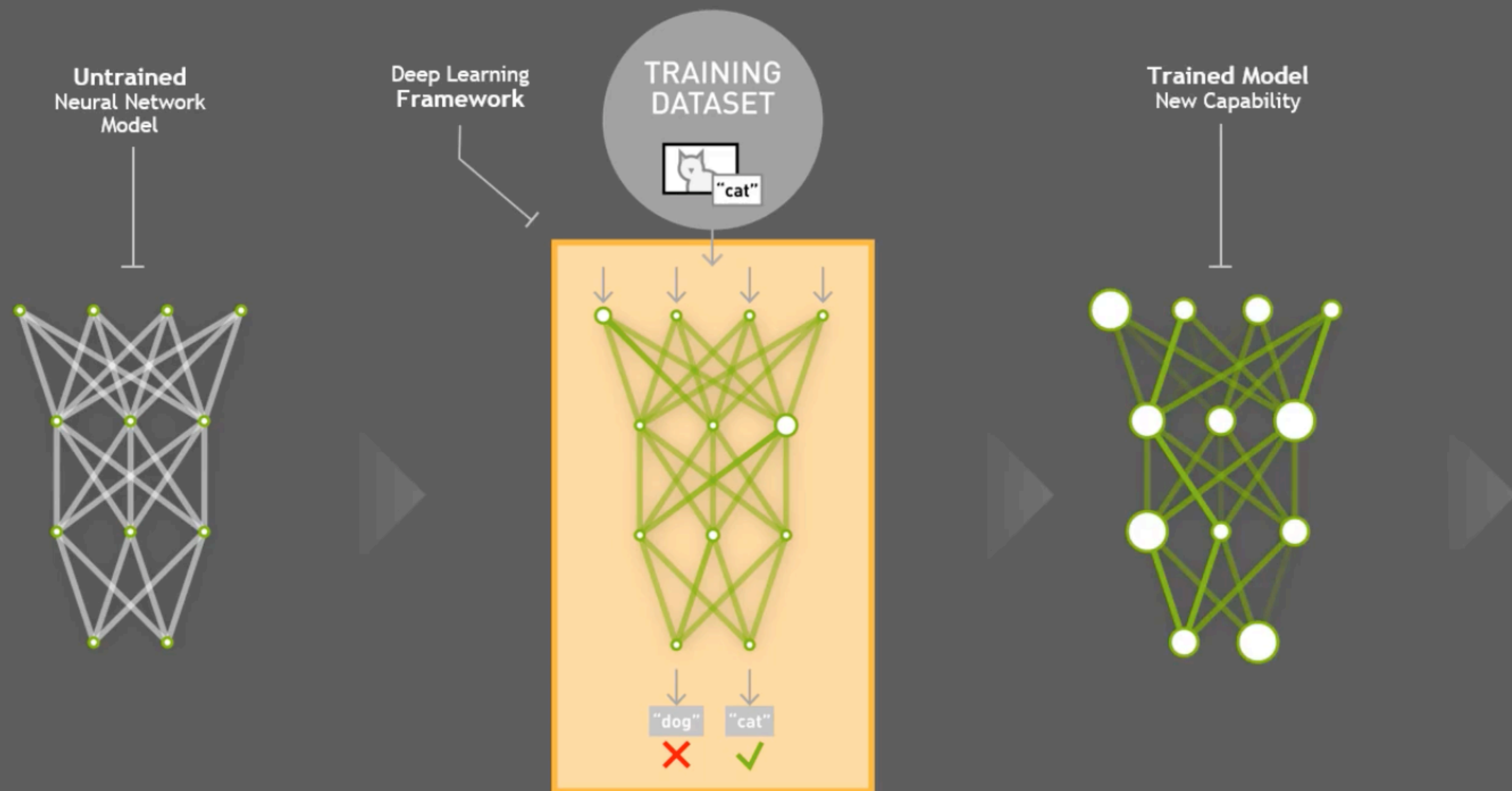
Learning a new capability
from existing data



DEEP LEARNING APPLICATION DEVELOPMENT

TRAINING

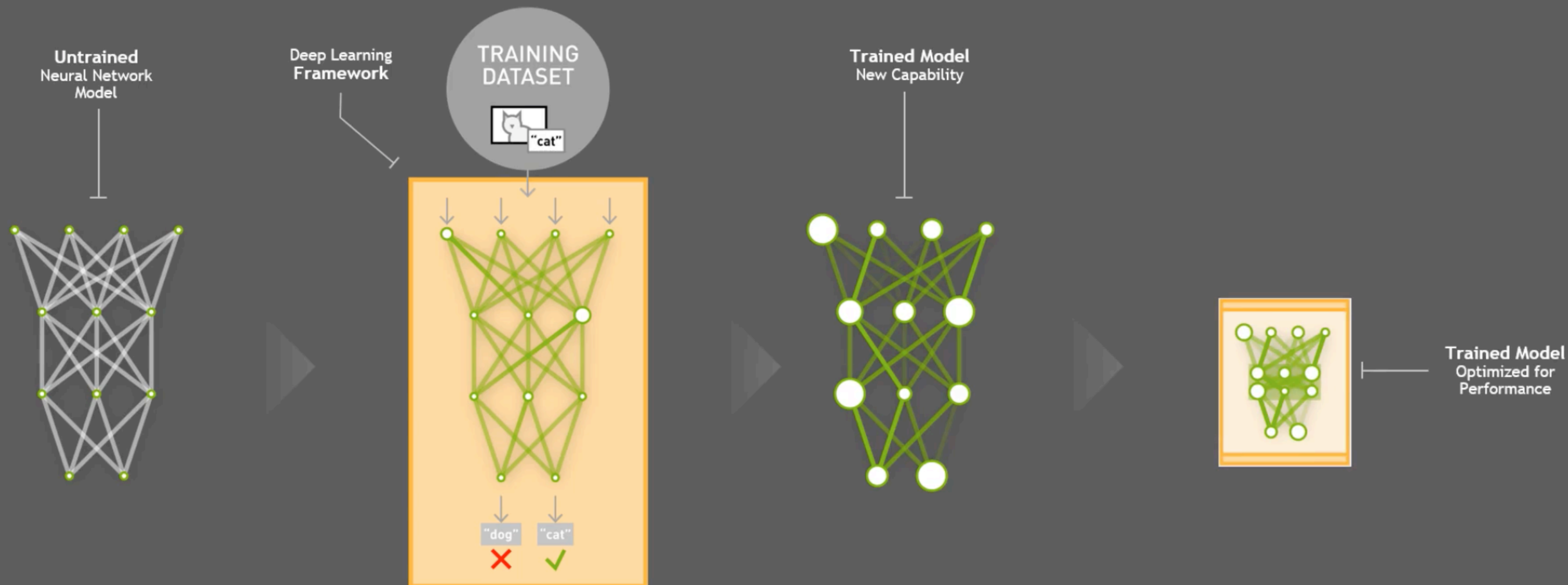
Learning a new capability
from existing data



DEEP LEARNING APPLICATION DEVELOPMENT

TRAINING

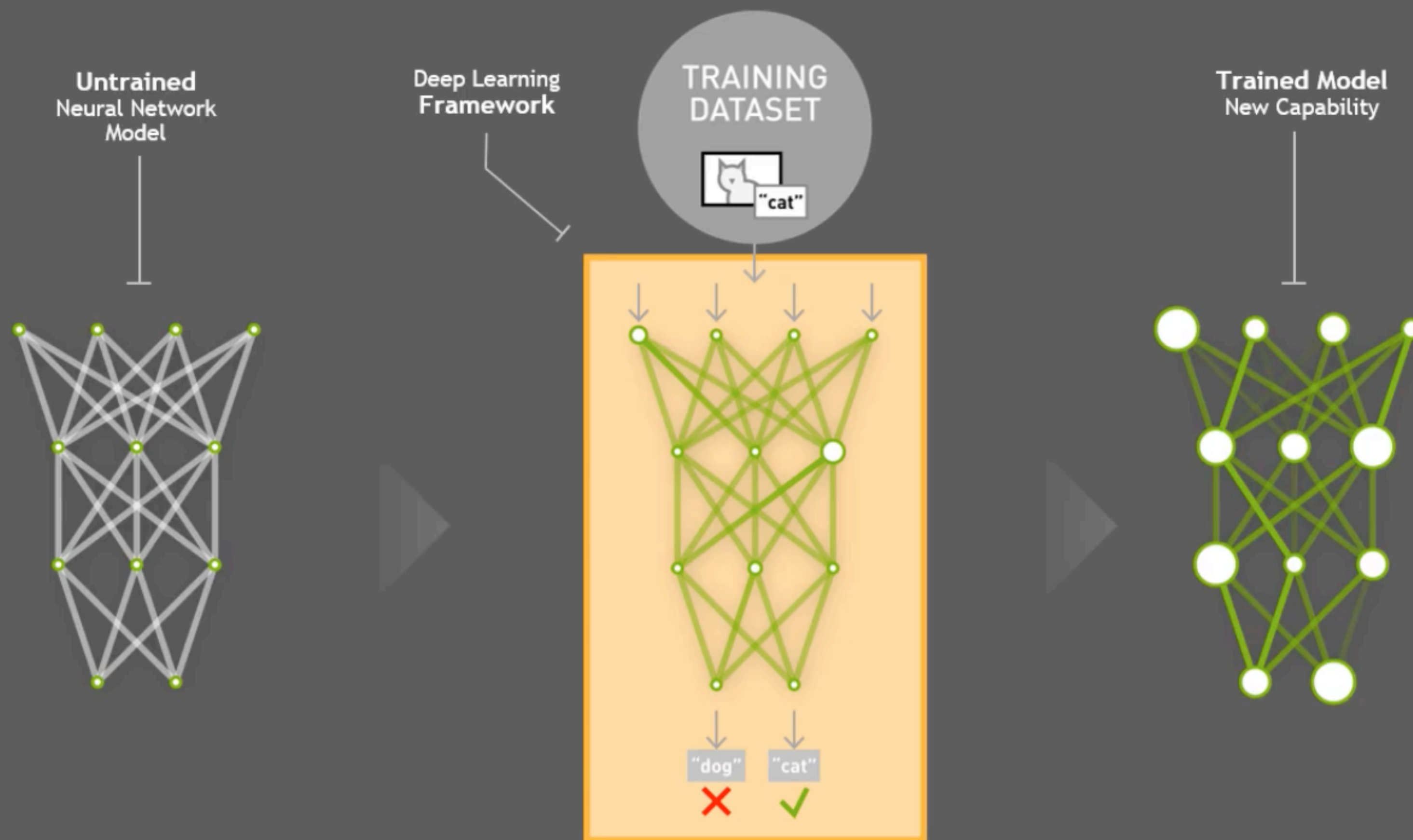
Learning a new capability
from existing data



DEEP LEARNING APPLICATION DEVELOPMENT

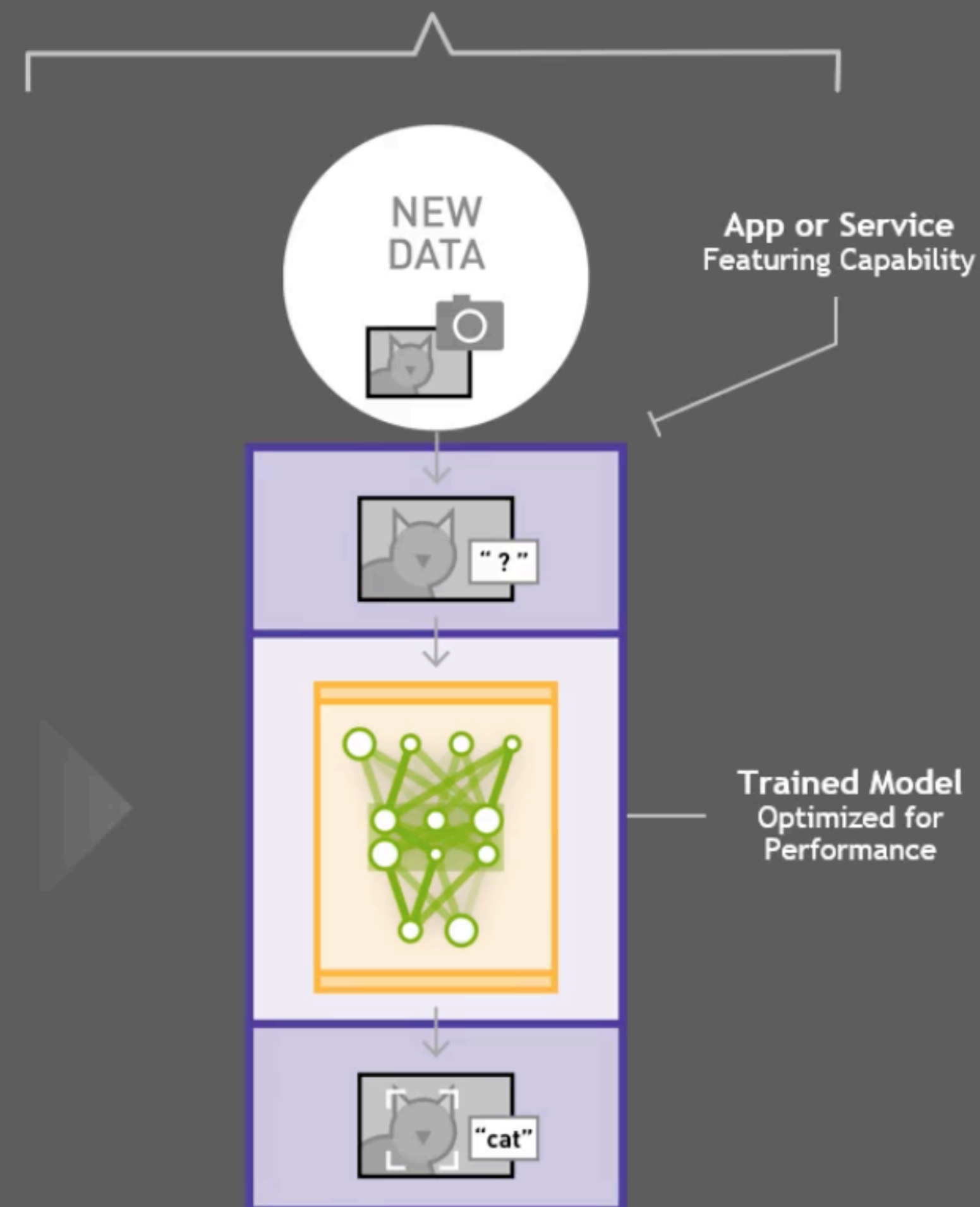
TRAINING

Learning a new capability
from existing data



INFERENCE

Applying this capability
to new data



a real-life scenario

Real-Life Machine Learning Workflow

Business
Problems

Implementing a ML model in real-life

Machine Learning starts before writing a single line of a neural network code

1. Frame and understand your problem
2. Explore data with Analysis tools
3. Engineer features relevant to your use case
4. Partial train small models to build features
5. Explore existing pre-trained models to be adapted (i.e. using transfer learning)
6. Write specific neural network code
7. Train, validate, evaluate ML model

ML problem framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance and error metrics need to be optimized is a key step in ML.

For example, imagine a scenario where a manufacturing company wants to identify which products will maximize profits. Reaching this business goal partially depends on determining the right number of products to produce. In this scenario, you want to predict the future sales of the product, based on past and current sales. Predicting future sales becomes the problem to solve, and using ML is one approach that can be used to solve it.

ML problem framing

- Define criteria for a successful outcome of the project
- Establish an observable and quantifiable performance metric for the project, such as accuracy, prediction latency, or minimizing inventory value
- Formulate the ML question in terms of inputs, desired outputs, and the performance metric to be optimized
- Evaluate whether ML is a feasible and appropriate approach
- Create a data sourcing and data annotation objective, and a strategy to achieve it
- Start with a simple model that is easy to interpret, and which makes debugging more manageable

Data collection

Start exploring our dataset

- Amazon Customer Reviews Dataset
 - <https://s3.amazonaws.com/amazon-reviews-pds/readme.html>
 - <s3://amazon-reviews-pds/tsv/>
 - crawler with name “tsv”
- MSCK REPAIR TABLE tsv

```
# Get Product categories
SELECT DISTINCT product_category
FROM "ai4devs"."amazon_reviews_parquet"

# Get a sample of reviews related to Digital_Video_Games
SELECT *
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'
LIMIT 10

# count all the reviews in Digital_Video_Games category
SELECT count(*) AS "num_reviews"
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'

# Count unique reviews
SELECT count(DISTINCT review_id) AS "unique_reviews"
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'

# Count unique customers
SELECT count(DISTINCT customer_id) AS "customers"
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'

# Find customers with multiple reviews
SELECT customer_id, count(DISTINCT review_id) AS "reviews"
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'
GROUP BY customer_id
ORDER BY reviews DESC

# Get all the reviews in this category
SELECT *
FROM "ai4devs"."amazon_reviews_parquet"
WHERE product_category='Digital_Video_Games'
```

Data collection

Start exploring our dataset

Results												
	marketplace ▾	customer_id ▾	review_id ▾	product_id ▾	product_parent ▾	product_title ▾	star_rating ▾	helpful_votes ▾	total_votes ▾	vine ▾	verified_purchase ▾	review_headline ▾
1	US	891276	R3LGIYKBEHXTYB	B004RMK57U	53534661	Playstation Plus Subscription	5	0	0	N	Y	Fast purchase
2	US	50451909	R14QEUQI9OR0YV	B00GXHISJE	599419294	Fallout 3: Game of The Year Edition	2	0	1	N	Y	Freezes!
3	US	34443239	R40M7NI4IM1Y2	B00K59HKIQ	384246568	Playstation Network Card	5	0	0	N	Y	Super fast !
4	US	4461282	R10DI66H4ZRF9Q	B0087STJLS	296282987	Battlefield 3: Premium Season Pass	5	0	0	N	Y	A lot of really cool maps, enjoy guarantee.
5	US	16183275	R2X6KKO5JTNP27	B00JLK6ULS	216162264	Sid Meier's Civilization: Beyond Earth	2	0	1	N	Y	Very Disappointed.
6	US	2885101	R39DYIKS6SC6JR	B00GGUHS4E	636901019	Call of Duty Black Ops II: Revolution DLC	5	3	3	N	Y	Call of Duty Black Ops II: Revolution DLC has my son's full attention and he's crazy about it.
7	US	20407843	R69QMTIRZS98U	B008NBRXVM	156792383	Bejeweled 3	1	0	0	N	Y	dose not work I cant get it to download but ...
8	US	38895212	R30GL20TL13YXA	B004APAELG	800401220	Rollercoaster Tycoon 2: Triple Thrill Pack	5	0	0	N	Y	can't go wrong
9	US	42320877	R2HGZTM9VHPBHU	B00HDCUTUY	940804794	Travel Pack - 3 in 1 - Hidden Object Game [Download]	3	0	1	N	Y	Three Stars
10	US	3390504	R2SGKXDAY1S1YC	B00DTWEOZ8	869820667	Titanfall [Online Game Code]	4	0	0	N	Y	Utterly Unique, Absolutely Brilliant

Results
review_body ▼
It was a good deal, worked perfectly ty amazon
I have Win 7 and I'm running with 6 gigs of RAM and a GTX 650 Ti card and . The game freezes for some reason. I found a patch for this issue but it still happens.
Super fast !
the maps are great , are full of details this is a very, very good product, people who designed really know how to create amazing environments which make you feel in a war ambient.
I was really excited for this game and picked it up despite the negative reviews. However, I am sad to say that the reviews were right. This is a civilization game that has lost its way. There are way too many options in this game. At times, yo
My Son Loves This Game!!! He's still thanking me for it. Great past time. This is a great investment to make.
dose not work I cant get it to download but what ever I'm not trying for refund because I don't want to mess with people that I dont understand
I loved these games when I was younger, and I love them now. It's very fun, not much of a fan of RCT3. I wish they still made expansions for this game!
ok but not very easy to use
Titanfall offers and delivers one of the most interesting and different FPS experiences available right now. The absolutely seamless transition from powerful ground warrior to 20 foot tall mech is incredible, and something I've never really experienced

Data preparation

Prepare data to be suitable for ML

```
data['review_body'] = data['review_headline'] + ' ' + data['review_body']

data = data[['review_id',
            'product_id', 'star_rating', 'review_body']]

data['label'] = data.star_rating.map({
    '1': '__label__negative__',
    '2': '__label__negative__',
    '3': '__label__neutral__',
    '4': '__label__positive__',
    '5': '__label__positive__'
})

data['review_body'] =
data['review_body'].apply(nltk.word_tokenize)

data['review_body'] = data.apply(lambda row: "
".join(row['review_body']).lower(), axis=1)
```

```
%%time
s3_athena_file = 's3://lb.athena.queries.result/products_reviews_queries/2021/06/15/07d38fc4-1a3e-401f-9832-7b7529389717.csv'
data = pd.read_csv(s3_athena_file, error_bad_lines=False, dtype='str')

CPU times: user 1.54 s, sys: 238 ms, total: 1.78 s
Wall time: 2.94 s

data.dropna(inplace=True)

print(data.shape)
print(data.columns)

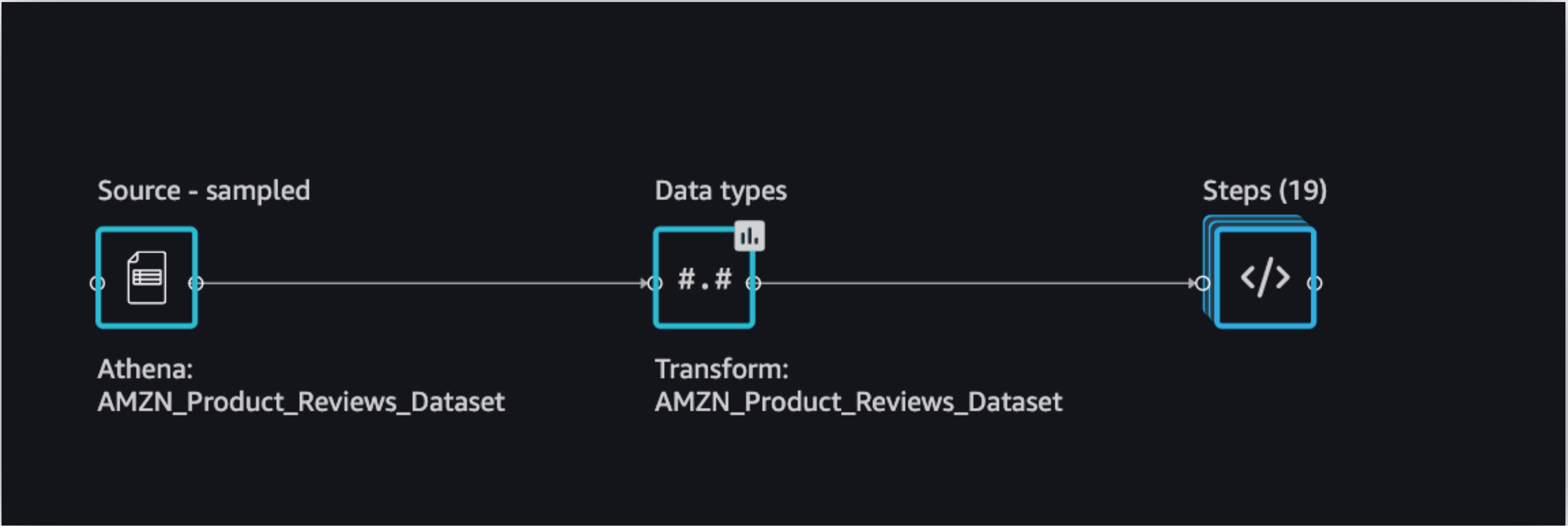
data.head()

(145427, 16)
Index(['marketplace', 'customer_id', 'review_id', 'product_id',
      'product_parent', 'product_title', 'star_rating', 'helpful_votes',
      'total_votes', 'vine', 'verified_purchase', 'review_headline',
      'review_body', 'review_date', 'year', 'product_category'],
      dtype='object')
```

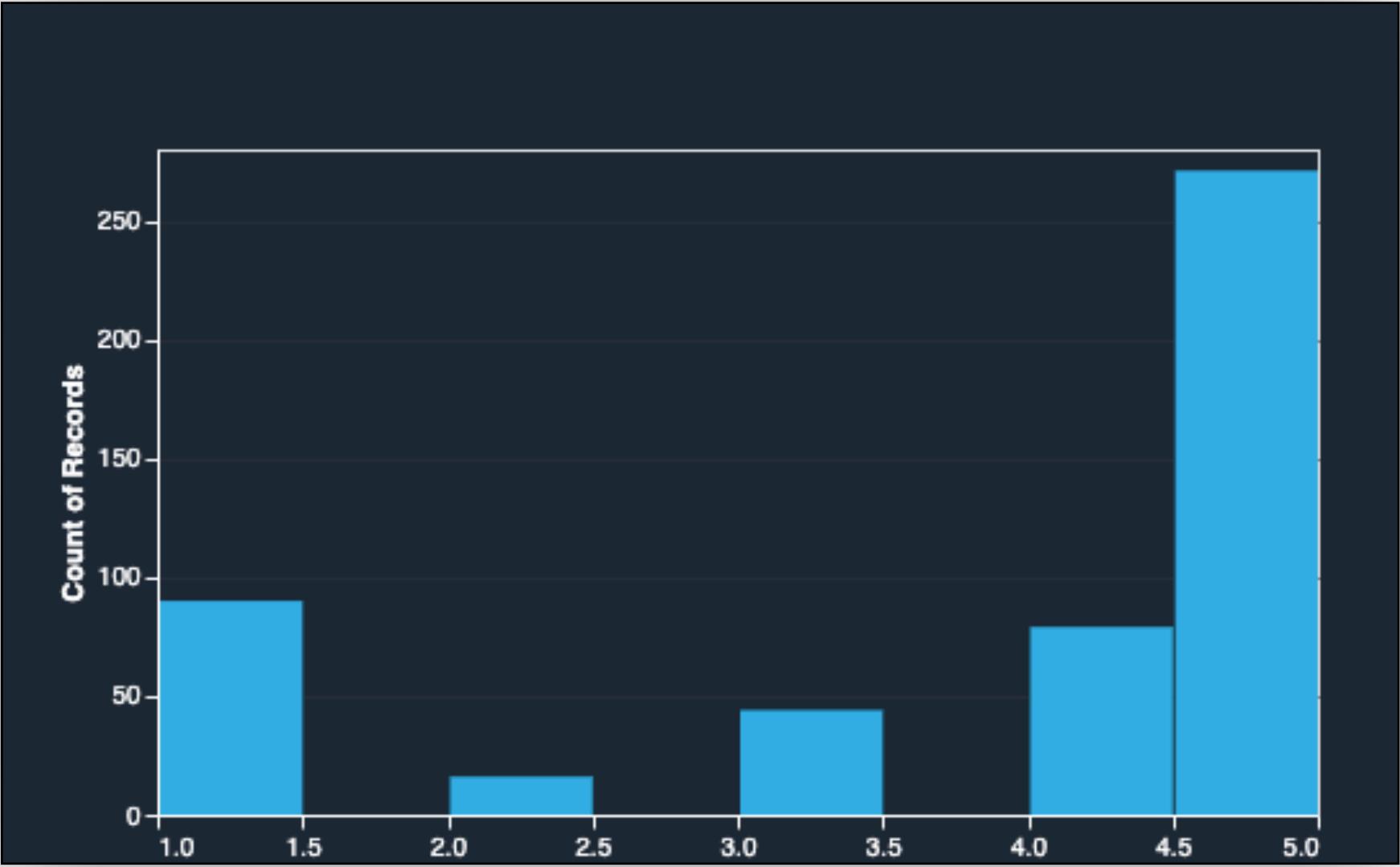
	marketplace	customer_id	review_id	product_id	product_parent	product_title	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline	review_body	review_date	year	prc
0	US	891276	R3LGIYKBEHXTYB	B004RMKS7U	53534661	Playstation Plus Subscription	5	0	0	N	Y	Fast purchase	It was a good deal, worked perfectly ty amazon	16391	2014	Digita
1	US	50451909	R14QEUQI9OR0YV	B00GXHISJE	599419294	Fallout 3: Game of The Year Edition	2	0	1	N	Y	Freezes!	I have Win 7 and I'm running with 6 gigs of RA...	16170	2014	Digita
2	US	34443239	R4OM7NI4IM1Y2	B00K59HKIQ	384246568	Playstation Network Card	5	0	0	N	Y	Super fast !	Super fast !	16391	2014	Digita
3	US	4461282	R10DI66H4ZRF9Q	B0087STJLS	296282987	Battlefield 3: Premium Season Pass	5	0	0	N	Y	A lot of really cool maps, enjoy guarantee.	the maps are great , are full of details this ...	16170	2014	Digita
4	US	16183275	R2X6KKO5JTNP27	B00JLK6ULS	216162264	Sid Meier's Civilization: Beyond Earth	2	0	1	N	Y	Very Disappointed.	I was really excited for this game and picked ...	16391	2014	Digita

SageMaker Data Wrangler

A workflow management tool for data analysis and preparation



Summary: Summary						
summary	marketplace	customer_id	review_id	product_id	product_parent	product_title
count	50000	50000	50000	50000	50000	50000
mean	None	2.683339001638E7	None	None	4.5833153185316E8	None
stddev	None	1.5084834320756255E7	None	None	2.775176088500891E8	None
min	US	10775	R1001BHT0TIGVT	B001AU6TQ8	243525	007 Legends [Do
max	US	53094755	RZZVIQNT2YEK3	B0149HT55K	999641217	rain - PS3 [Digital



SageMaker Processing Platform

Offload SageMaker tasks to external workers



```
from sagemaker.sklearn.processing import SKLearnProcessor

sklearn_processor = SKLearnProcessor(
    framework_version='0.23-1',
    role=role,
    instance_type='ml.m5.2xlarge',
    instance_count=1)
```



```
from sagemaker.processing import ProcessingInput, ProcessingOutput

sklearn_processor.run(
    code='preprocessing.py',

    inputs=[
        ProcessingInput(
            source=input_data,
            destination='/opt/ml/processing/input'
        )
    ],

    outputs=[
        ProcessingOutput(
            output_name='bt_data',
            source='/opt/ml/processing/output/bt'
        ),
        ProcessingOutput(
            output_name='fs_data',
            source='/opt/ml/processing/output/fs'
        )
    ],

    arguments=[
        '--filename', filename
    ]

)
```

SageMaker Feature Store

- A single feature corresponds to a column in your dataset. A feature group is a predefined schema for a collection of features - each feature in the feature group has a specified data type and name. A single record in a feature group corresponds to a row in your dataframe. A feature store is a collection of feature groups.
- Record identifier name is the name of the feature defined in the feature group's feature definitions whose value uniquely identifies a Record defined in the feature group's feature definitions.
- Event time feature name is the name of the EventTime feature of a Record in FeatureGroup. An EventTime is a timestamp that represents the point in time when a new event occurs that corresponds to the creation or update of a Record in the FeatureGroup. All Records in the FeatureGroup must have a corresponding EventTime.

Model Evaluation

How to know we arrived there?

- After the model has been trained, evaluate it to determine if its performance and accuracy will enable you to achieve your business goals. You might want to generate multiple models using different methods and evaluate the effectiveness of each model. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You also might evaluate whether your model needs to be more sensitive than specific, or more specific than sensitive. For multiclass models, evaluate error rates for each class separately.
- You can evaluate your model using historical data (offline evaluation) or live data (online evaluation). In offline evaluation, the trained model is evaluated with a portion of the dataset that has been set aside as a holdout set. This holdout data is never used for model training or validation—it's only used to evaluate errors in the final model. The holdout data annotations need to have high accuracy for the evaluation to make sense. Allocate additional resources to verify the accuracy of the holdout data.
- AWS services that are used for model training also have a role in this phase. Model validation can be performed using Amazon SageMaker, AWS Deep Learning AMI, or Amazon EMR.
- Based on the evaluation results, you might fine-tune the data, the algorithm, or both. When you fine-tune the data, you apply the concepts of data cleansing, preparation, and feature engineering.

Model Evaluation

How to know we arrived there?

- Have a clear understanding of how you measure success
- Evaluate the model metrics against the business expectations for the project
- Plan and execute Production Deployment (Model Deployment and Model Inference)

Apply these best practices:

- Monitor model performance in production and compare to business expectations
- Monitor differences between model performance during training and in production
- When changes in model performance are detected, retrain the model. For example, sales expectations and subsequent predictions may change due to new competition
- Use batch transform as an alternative to hosting services if you want to get inferences on entire datasets
- Take advantage of production variants to test variations of a new model with A/B testing

AWS ML Stack

The AWS machine learning stack

Broadest and most complete set of Machine Learning capabilities

AI SERVICES

VISION Amazon Rekognition	SPEECH Amazon Polly	 Amazon Transcribe <small>+Medical NEW</small>	TEXT Amazon Comprehend <small>+Medical</small>	 Amazon Translate	 Amazon Textract	<small>NEW!</small> SEARCH Amazon Kendra	CHATBOTS Amazon Lex	PERSONALIZATION Amazon Personalize	FORECASTING Amazon Forecast	<small>NEW!</small> FRAUD Amazon Fraud Detector	<small>NEW!</small> DEVELOPMENT Amazon CodeGuru	<small>NEW!</small> CONTACT CENTERS Contact Lens <small>For Amazon Connect</small>
---	-----------------------------------	--	---	----------------------	---------------------	---	-----------------------------------	--	---	--	--	--

ML SERVICES

 Amazon SageMaker	Ground Truth	Augmented AI	ML Marketplace	SageMaker Studio IDE <small>NEW!</small>								Neo
				Built-in algorithms	<small>NEW!</small> Notebooks	<small>NEW!</small> Experiments	Model training & tuning	<small>NEW!</small> Debugger	<small>NEW!</small> Autopilot	Model hosting	<small>NEW!</small> Model Monitor	

ML FRAMEWORKS & INFRASTRUCTURE

TensorFlow	mxnet	GLUON	Keras	Deep Learning AMIs & Containers	GPUs & CPUs	Elastic Inference	Inferentia (Inf1 instance)	FPGA
PYTORCH								

PyTorch Lightning

Amazon SageMaker

A Machine Learning platform

- Amazon SageMaker is a platform to run training and inference from your laptop, directly in cloud.
- SageMaker training jobs allow setting up and tearing down cloud infrastructure
- Can run training jobs locally on bare metal or SageMaker containers

```
pytorch_estimator = PyTorch('pytorch-train.py',
                             instance_type='ml.p3.2xlarge',
                             instance_count=1,
                             framework_version='1.5.0',
                             py_version='py3',
                             hyperparameters = {
                                 'epochs': 20,
                                 'batch-size': 64,
                                 'learning-rate': 0.1
                             })

pytorch_estimator.fit({
    'train': 's3://my-data-bucket/path/to/my/training/data',
    'test': 's3://my-data-bucket/path/to/my/test/data'
})
```

PyTorch

A deep learning platform

- is pythonic (its n-dimensional tensor is similar to numpy) with a quite easy learning curve
- built-in support for data parallelism
- support for dynamic computational graphs
- Imperative programming model



PyTorch on SageMaker

Running training on Amazon SageMaker

Initializes SageMaker session which holds context data

The bucket containing our input data

The IAM Role which SageMaker will impersonate to run the estimator

Remember you cannot use `sagemaker.get_execution_role()` if you're not in a SageMaker notebook, an EC2 or a Lambda (i.e. running from your local PC)

name of the runnable script containing `__main__` function (entrypoint)

path of the folder containing training code. It could also contain a `requirements.txt` file with all the dependencies that needs to be installed before running

these hyperparameters are passed to the main script as arguments and can be overridden when fine tuning the algorithm

Call fit method on estimator, which trains our model, passing training and testing datasets as environment variables. Data is copied from S3 before initializing the container

```
import json
import boto3
import sagemaker
from sagemaker.pytorch import PyTorch

sagemaker_session = sagemaker.Session()

bucket = 's3://dataset.mnist'

role = 'arn:aws:iam::XXXXXXXX:role/SageMakerRole_MNIST'

estimator = PyTorch(

    entry_point='train.py',
    source_dir='code',
    role=role,
    framework_version='1.4.0',
    train_instance_count=1,
    train_instance_type='ml.p2.xlarge',
    hyperparameters={
        'epochs': 6,
        'batch-size': 128,
    })

estimator.fit({
    'train': bucket+'/training',
    'test': bucket+'/testing'
})
```

Amazon SageMaker

- A PyTorch implementation of MNIST neural network is given.
- The network is built at forward pass.
- Each batch of data of each epoch within train method
 - loads data
 - resets optimizer
 - computes output
 - computes loss
 - optimizes weights

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output

def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{}] ({:.0f}%) \t Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break
```

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%) \n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

def main():
    # arguments handling
    # [...]

    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])
    dataset1 = datasets.MNIST('../data', train=True, download=True,
                               transform=transform)
    dataset2 = datasets.MNIST('../data', train=False,
                               transform=transform)
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    if args.save_model:
        torch.save(model.state_dict(), "mnist_cnn.pt")
```

PyTorch Lightning

With Lightning, PyTorch gets both simplified AND on steroids

Published in 2019, it is a framework to structure a PyTorch project, gain support for less boilerplate and improved code reading.

The simple interface gives **professional production** teams and newcomers access to the latest state of the art techniques developed by the PyTorch and PyTorch Lightning community.

- 96 contributors
- 8 research scientists
- rigorously tested

Principle 1

Enable maximal flexibility.

Principle 2

Abstract away unnecessary boilerplate, but make it accessible when needed.

Principle 3

Systems should be self-contained (ie: optimizers, computation code, etc).

Principle 4

Deep learning code should be organized into 4 distinct categories.

- Research code (the LightningModule).
- Engineering code (handled by the Trainer).
- Non-essential research code (in Callbacks).
- Data (PyTorch Dataloaders).

Getting Started

Step 0: imports

Import PyTorch standard packages such as **nn** and **Functional** and **DataLoader**

Import **Transforms** from torchvision (when needed)

Import **pytorch_lightning** core class

```
import os
import torch
from torch import nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
from torchvision import transforms
import pytorch_lightning as pl
```

Getting Started

Step 1: Lightning module

Build a class extending **pl.LightningModule** and implement utility methods which will be called by trainer during the training loop

dataset preparation and loading

neural network definition

loss computation

optimizers definition

validation computation and stacking

```
class myClass(pl.LightningModule):  
  
    def __init__(self):  
        super().__init__()  
  
    def prepare_data(self):  
  
    def train_dataloader(self):  
  
    def val_dataloader(self):  
  
    def test_dataloader(self):  
  
  
    def forward(self, x):  
  
    def training_step(self, batch, batch_idx):  
  
    def configure_optimizers(self):  
  
  
    def validation_step(self, batch, batch_idx):  
  
    def validation_epoch_end(self, outputs):  
  
    def validation_end(self, outputs):
```

Getting Started

Step 2: Trainer

Lightning Trainer class controls flow execution, multi-GPU parallelization and intermediary data saving to **default_root_dir**

Our defined model class is instantiated passing all the required hyperparams, then **fit** method is called on trainer, passing params as an argument

Training on multiple GPUs is easy as setting an argument

```
import pytorch_lightning as pl

from pytorch_lightning import Trainer

myTrainer=pl.Trainer(
    gpus=args.gpus,
    max_epochs=args.epochs,
    default_root_dir=args.output_data_dir
)

model = myClass(
    # params
)

myTrainer.fit(model)
```

MNIST

MNIST is the new Hello World

It's a well known problem, that can be used as a reference

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.



709.png



712.png



733.png



743.png



745.png



236.png



239.png



244.png



259.png



266.png

SageMaker job script

Can be run from a Notebook or any Python environment

- Configure SageMaker Session
- Setup an Estimator, configuring instance count, PyTorch container version and instance type
- Pass training and testing datasets paths from S3. Data is copied from S3 before initializing the container and mapped to local folders
- After training containers get dismissed and instances destroyed

```
# MNIST on SageMaker with PyTorch Lightning
import json
import boto3
import sagemaker
from sagemaker.pytorch import PyTorch

# Initializes SageMaker session which holds context data
sagemaker_session = sagemaker.Session()

# The bucket containig our input data
bucket = 's3://dataset.mnist'

# The IAM Role which SageMaker will impersonate to run the estimator
# Remember you cannot use sagemaker.get_execution_role()
# if you're not in a SageMaker notebook, an EC2 or a Lambda
# (i.e. running from your local PC)

role = 'arn:aws:iam::XXXXXXX:role/SageMakerRole_MNIST'

# Create a new PyTorch Estimator with params
estimator = PyTorch(
    # name of the runnable script containing __main__ function (entrypoint)
    entry_point='train.py',
    # path of the folder containing training code. It could also contain a
    # requirements.txt file with all the dependencies that needs
    # to be installed before running
    source_dir='code',
    role=role,
    framework_version='1.4.0',
    train_instance_count=1,
    train_instance_type='ml.p2.xlarge',
    # these hyperparameters are passed to the main script as arguments and
    # can be overridden when fine tuning the algorithm
    hyperparameters={
        'epochs': 6,
        'batch-size': 128,
    })

# Call fit method on estimator, wich trains our model, passing training
# and testing datasets as environment variables. Data is copied from S3
# before initializing the container
estimator.fit({
    'train': bucket+'/training',
    'test': bucket+'/testing'
})
```

Training class

Use PyTorch Lightning Trainer class

- Receives arguments from SageMaker (as arg variables)
- Instantiates a Trainer class
- Instantiates a classifier passing training parameters
- calls **.fit** method on trainer, passing the model
- saves trained model to local **model_dir** which is mirrored to S3 by SageMaker when container is dismissed

```
import argparse
import os

# default pytorch import
import torch

# import lightning library
import pytorch_lightning as pl

# import trainer class, which orchestrates our model training
from pytorch_lightning import Trainer

# import our model class, to be trained
from MNISTClassifier import MNISTClassifier

# This is the main method, to be run when train.py is invoked
if __name__ == '__main__':

    parser = argparse.ArgumentParser()

    # hyperparameters sent by the client are passed as command-line arguments to the script.
    parser.add_argument('--epochs', type=int, default=50)
    parser.add_argument('--batch-size', type=int, default=64)
    parser.add_argument('--gpus', type=int, default=1) # used to support multi-GPU or CPU training

    # Data, model, and output directories. Passed by sagemaker with default to os env variables
    parser.add_argument('-o', '--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
    parser.add_argument('-m', '--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
    parser.add_argument('-tr', '--train', type=str, default=os.environ['SM_CHANNEL_TRAIN'])
    parser.add_argument('-te', '--test', type=str, default=os.environ['SM_CHANNEL_TEST'])

    args, _ = parser.parse_known_args()
    print(args)

    # Now we have all parameters and hyperparameters available and we need to match them with sagemaker
    # structure. default_root_dir is set to out_put_data_dir to retrieve from training instances all the
    # checkpoint and intermediary data produced by lightning
    mnistTrainer=pl.Trainer(gpus=args.gpus, max_epochs=args.epochs, default_root_dir=args.output_data_dir)

    # Set up our classifier class, passing params to the constructor
    model = MNISTClassifier(
        batch_size=args.batch_size,
        train_data_dir=args.train,
        test_data_dir=args.test
    )

    # Runs model training
    mnistTrainer.fit(model)

    # After model has been trained, save its state into model_dir which is then copied to back S3
    with open(os.path.join(args.model_dir, 'model.pth'), 'wb') as f:
        torch.save(model.state_dict(), f)
```

MNISTClassifier

```
import os
import math
import random as rn
import numpy as np

import torch
import torch.nn as nn
from torch.nn import functional as F
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler
from torchvision import transforms as T, datasets
import pytorch_lightning as pl

class MNISTClassifier(pl.LightningModule):
```

```
def __init__(self, train_data_dir=batch_size=128, test_data_dir=None, num_workers=4):
    '''Constructor method
    Parameters:
    train_data_dir (string): path of training dataset to be used either for training and validation
    batch_size (int): number of images per batch. Defaults to 128.
    test_data_dir (string): path of testing dataset to be used after training. Optional.
    num_workers (int): number of processes used by data loader. Defaults to 4.
    '''

    # Invoke constructor
    super(MNISTClassifier, self).__init__()

    # Set up class attributes
    self.batch_size = batch_size
    self.train_data_dir = train_data_dir
    self.test_data_dir = test_data_dir
    self.num_workers = num_workers

    # Define network layers as class attributes to be used
    self.conv_layer_1 = torch.nn.Sequential(
        # The first block is made of a convolutional layer (3 channels, 28x28 images and a kernel mask of 5),
        torch.nn.Conv2d(3, 28, kernel_size=5),
        # a non linear activation function
        torch.nn.ReLU(),
        # a maximization layer, with mask of size 2
        torch.nn.MaxPool2d(kernel_size=2))

    # A second block is equal to the first, except for input size which is different
    self.conv_layer_2 = torch.nn.Sequential(
        torch.nn.Conv2d(28, 10, kernel_size=2),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2))

    # A dropout layer, useful to reduce network overfitting
    self.dropout1=torch.nn.Dropout(0.25)

    # A fully connected layer to reduce dimensionality
    self.fully_connected_1=torch.nn.Linear(250,18)

    # Another fine tuning dropout layer to make network fine tune
    self.dropout2=torch.nn.Dropout(0.08)

    # The final fully connected layer wich output maps to the number of desired classes
    self.fully_connected_2=torch.nn.Linear(18,10)
```

MNISTClassifier

```
def load_split_train_test(self, valid_size = .2):
    '''Loads data and builds training/validation dataset with provided split size

    Parameters:
    valid_size (float): the percentage of data reserved to validation
    Returns:
    (torch.utils.data.DataLoader): Training data loader
    (torch.utils.data.DataLoader): Validation data loader
    (torch.utils.data.DataLoader): Test data loader

    ...

    num_workers = self.num_workers

    train_transforms = T.Compose(
        [T.RandomHorizontalFlip(),
         T.ToTensor(),
         T.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])

    train_data = datasets.ImageFolder(self.train_data_dir, transform=train_transforms)

    num_train = len(train_data)
    indices = list(range(num_train))
    split = int(np.floor(valid_size * num_train))
    np.random.shuffle(indices)

    # extracts indexes for train and validation, then builds a random sampler
    train_idx, val_idx = indices[split:], indices[:split]
    train_sampler = SubsetRandomSampler(train_idx)
    val_sampler = SubsetRandomSampler(val_idx)
    # which is passed to data loader to perform image sampling when loading data
    train_loader = torch.utils.data.DataLoader(
        train_data, sampler=train_sampler,
        batch_size=self.batch_size, num_workers=num_workers)
    val_loader = torch.utils.data.DataLoader(
        train_data, sampler=val_sampler,
        batch_size=self.batch_size, num_workers=num_workers)

    # if testing dataset is defined, we build its data loader as well
    test_loader = None
    if self.test_data_dir is not None:
        test_transforms = T.Compose(
            [T.ToTensor(),T.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])
        test_data = datasets.ImageFolder(self.test_data_dir, transform=test_transforms)
        test_loader = torch.utils.data.DataLoader(
            train_data,batch_size=self.batch_size,
            num_workers=num_workers)
    return train_loader, val_loader, test_loader
```

MNISTClassifier

```
def prepare_data(self):
    '''Prepares datasets. Called once per training execution
    '''
    self.train_loader, self.val_loader, self.test_loader = self.load_split_train_test()

def train_dataloader(self):
    '''
    Returns:
    (torch.utils.data.DataLoader): Training set data loader
    '''
    return self.train_loader

def val_dataloader(self):
    '''
    Returns:
    (torch.utils.data.DataLoader): Validation set data loader
    '''
    return self.val_loader

def test_dataloader(self):
    '''
    Returns:
    (torch.utils.data.DataLoader): Testing set data loader
    '''
    return DataLoader(MNIST(
        os.getcwd(), train=False,
        download=False, transform=transform.ToTensor()), batch_size=128)
```

```
def forward(self, x):
    '''Forward pass, it is equal to PyTorch forward method.
    Here network computational graph is built
    Parameters:
    x (Tensor): A Tensor containing the input batch of the network
    Returns:
    An one dimensional Tensor with probability array for each input image
    '''
    x=self.conv_layer_1(x)
    x=self.conv_layer_2(x)
    x=self.dropout1(x)
    x=torch.relu(self.fully_connected_1(x.view(x.size(0),-1)))
    x=F.leaky_relu(self.dropout2(x))
    return F.softmax(self.fully_connected_2(x), dim=1)

def configure_optimizers(self):
    '''
    Returns:
    (Optimizer): Adam optimizer tuned wit model parameters
    '''
    return torch.optim.Adam(self.parameters())
```

PyTorch Lightning Bolts

Useful resources

PyTorch

<https://pytorch.org/>

PyTorch Lightning

<https://github.com/PyTorchLightning/pytorch-lightning>

PyTorch Lightning Bolts

<https://github.com/PyTorchLightning/pytorch-lightning-bolts>

AWS re:Invent getting started video

<https://www.youtube.com/watch?v=6lhl7hPFpX8>

Getting started with PL and Sagemaker

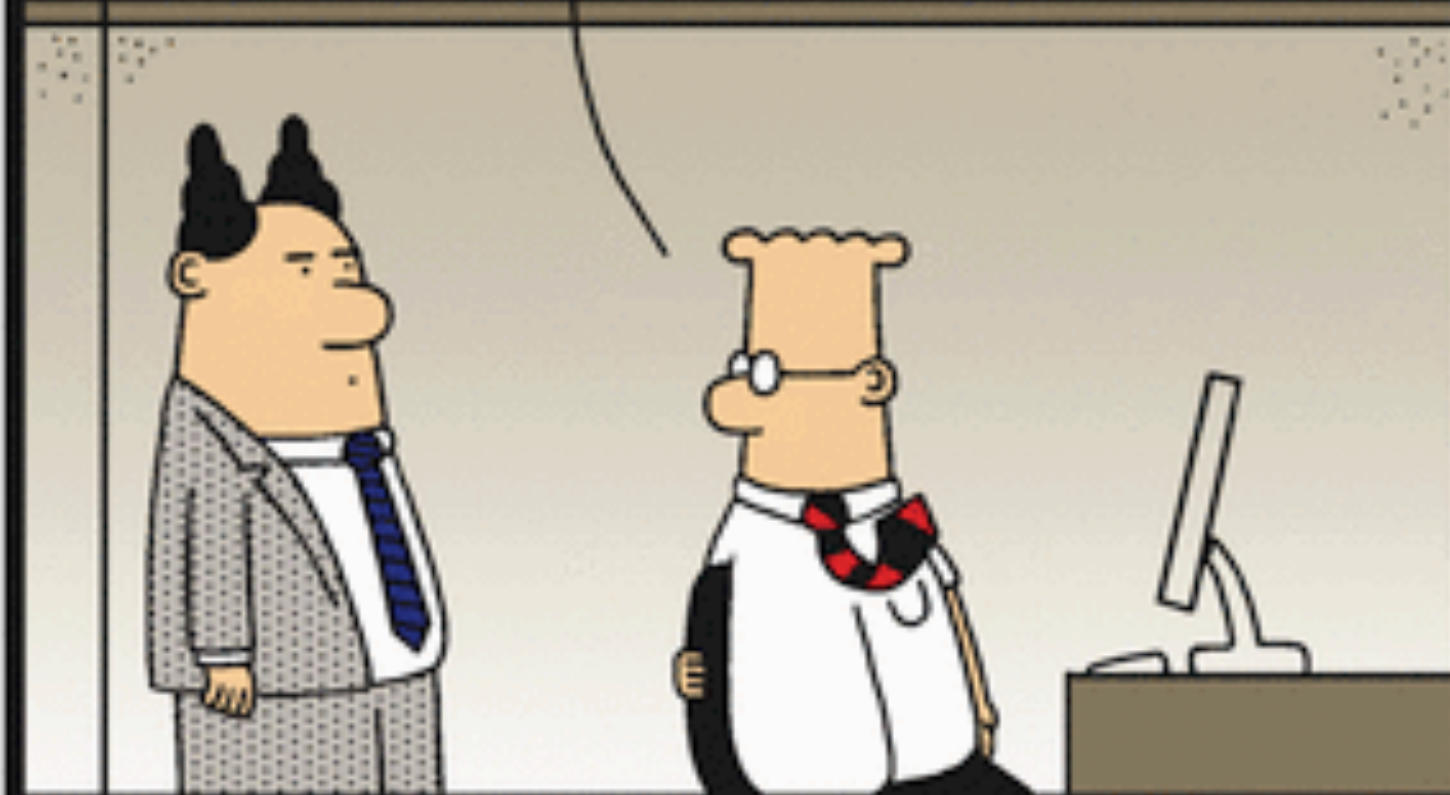
<https://towardsdatascience.com/building-a-neural-network-on-amazon-sagemaker-with-pytorch-lightning-63730ec740ea>

DO WE HAVE ANY
ACTIONABLE ANALYTICS
FROM OUR BIG DATA
IN THE CLOUD?



Dilbert.com DilbertCartoonist@gmail.com

YES, THE DATA SHOWS
THAT MY PRODUCTIVITY
PLUNGES WHENEVER YOU
LEARN NEW JARGON.



1-9-13 ©2013 Scott Adams, Inc./Dist. by Universal Uclick

MAYBE IN-MEMORY
COMPUTING WILL ACCEL-
ERATE YOUR APPLICA-
TIONS.



thank you.