Yale

Machine Learning (and High Energy Physics)

Some of What I learned from MLHEP 2017

Raymond Ehlers

Relativistic Heavy Ion Group
Department of Physics, Yale University

8 Aug 2017

Outline

What Will I Discuss?

- I attended the Machine Learning in High Energy Physics (MLHEP) 2017 summer school during July 2017
  - 6 days of alternating lectures and hands-on seminars.
  - A few additional talks on ML applications in LHCb, Jet Images (DeepJets), industry, etc.
- This talk will cover a brief discussion of some ML algorithms and concepts.
  - This is not remotely meant to be exhaustive.
- However, it is hopefully enough to get everyone oriented.
  - Emphasis on practical information.
- And provide a few interesting ideas along the way.

Machine Learning Introduction

- What is Machine Learning?
  - Using some algorithm to learn about or classify some dataset without explicitly telling it how to do so.
- Usually, this is achieved by optimization of some loss function, such as minimizing the least squares error (LSE)[1]
- The loss function is often minimized via gradient descent.
  - There are many variations and options to improve performance.

[1] There are a number of functions available.

Machine Learning Introduction

- What is Machine Learning?
  - Using some algorithm to learn about or classify some dataset without explicitly telling it how to do so.
- Usually, this is achieved by optimization of some loss function, such as minimizing the least squares error (LSE)[1]
- The loss function is often minimized via gradient descent.
  - There are many variations and options to improve performance.
- It sounds very complicated (and can be!), but many of the ideas are already familiar!
  - As a basic example, fitting to a function is (simple and over-fitted) machine learning problem.

[1] There are a number of functions available.

Outline

Supervised vs Unsupervised Learning[2]

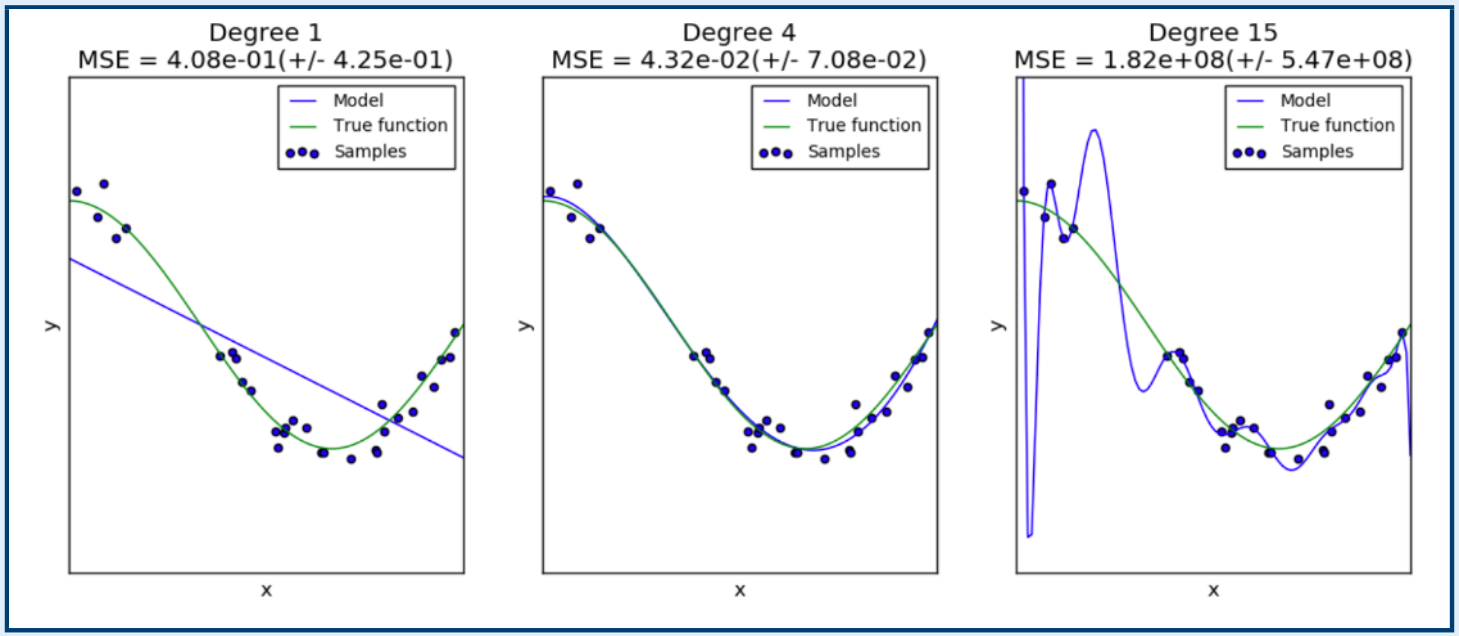
- Supervised
  - Need training data with input and desired output.
  - Predominately used today, and will be the main part of the discussion today.
- Unsupervised
  - The training data does not need truth information.
    - It can learn just from sufficient input data.
  - Would be fantastic, but it is exceedingly difficult
  - Some examples include
    - k-means clustering
    - autoencoders
    - principal component analysis (PCA)
    - singular value decomposition (SVD)

[2] Lecture 1, 7

Training and Test Datasets

- To train any (supervised) method, we need input that has a corresponding true output.
  - The model takes the inputs and then learns how to output the desired output.
- Practically, we want to train the model, and then test how well its done.
  - To test it, we need truth data, so we split the training dataset.
- Split the training dataset into independent parts, train on one part, and then test on another.
  - This also helps to avoid overfitting.
- Some techniques split the training sets further for additional benefits.
  - More on this below.

Overfitting[3]



- Model learns exactly the training data
  - It is useless for predictions!
- Different types of models often require different techniques to avoid.

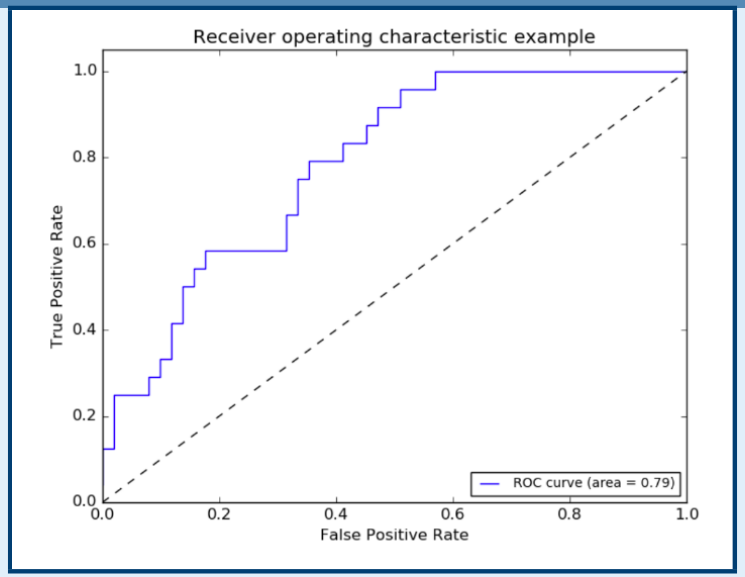
[3] Lecture 1

Ensemble Methods[4]

- Use multiple models to improve the quality of predictions.
- The multiple models could take many different forms.
  - Entirely different models and average the results.
  - Use the same model with different parameters.
  - Same model on different training (sub-)datasets.
- Often an effective technique to take advantage of strengths of different models.

[4] Lecture 4

Evaluating Models[5]

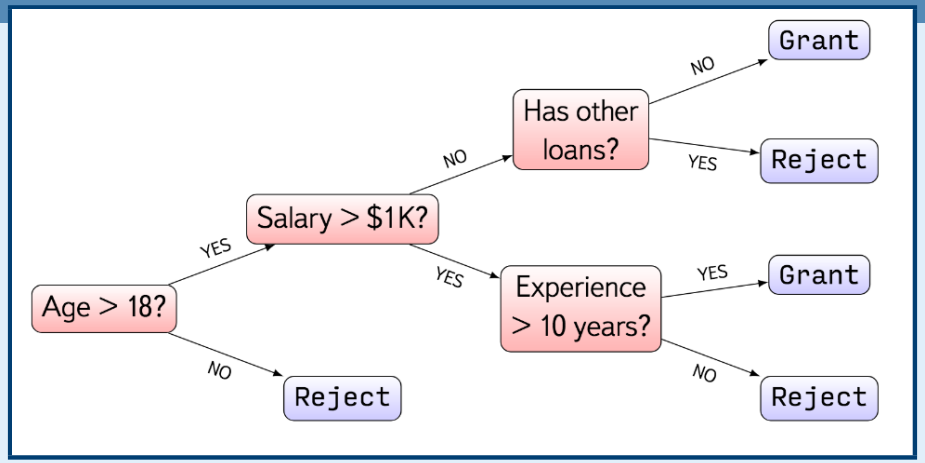


- There are many different ways to evaluate models
- The receiver operating characteristic (ROC) curve is quite common
  - Plots the false positive rate (FPR) vs the true positive rate (TPR)
- Want to maximize TPR while minimizing FPR.
  - There will always be a trade off

[5] Lecture 1

Outline

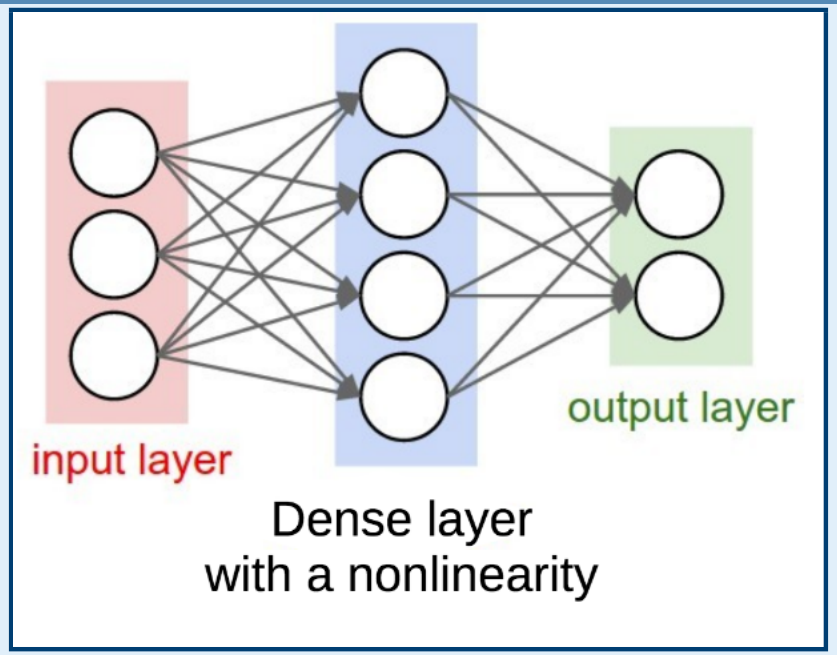
Decision Trees[6]



- A decision is made a each node.
- Wide variety of configuration options:
  - Number of leaves.
    - Too many leaves can lead to overfitting.
  - Depth of tree.
  - Minimum number of entries to split leaves.
  - Stopping conditions
  - Etc.

[6] Lecture 2

Neural Networks(NNs)[7]



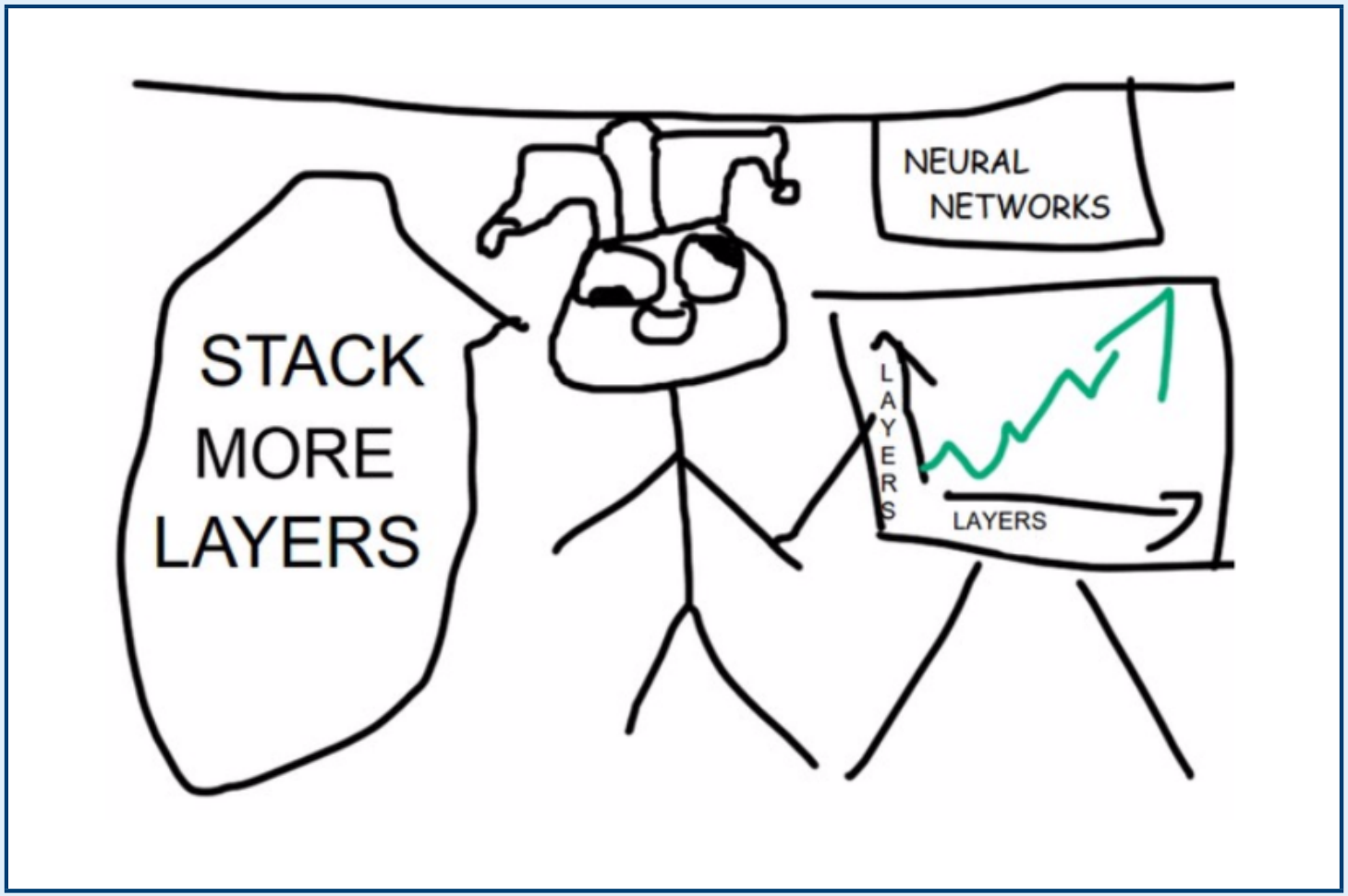
- Nodes arrange in layers are connected together.
- Each node has some sort of non-linear activation needed.
  - ReLU is common, but there are lots of options.
- Difficult to tune because there are so many choices.

[7] Lecture 5

Deep Neural Nets

- Deep just means many layers.
- This can be an incredibly powerful technique, but it also substantially increases the number of parameters.
- Backpropagation is hugely important to iteratively update network weights.
  - It is just the chain rule.
  - (It is important for all NNs, but especially so for deep NNs).
- As you increase the depth, you need better (read: larger) training sets!

Deep Neural Nets How To

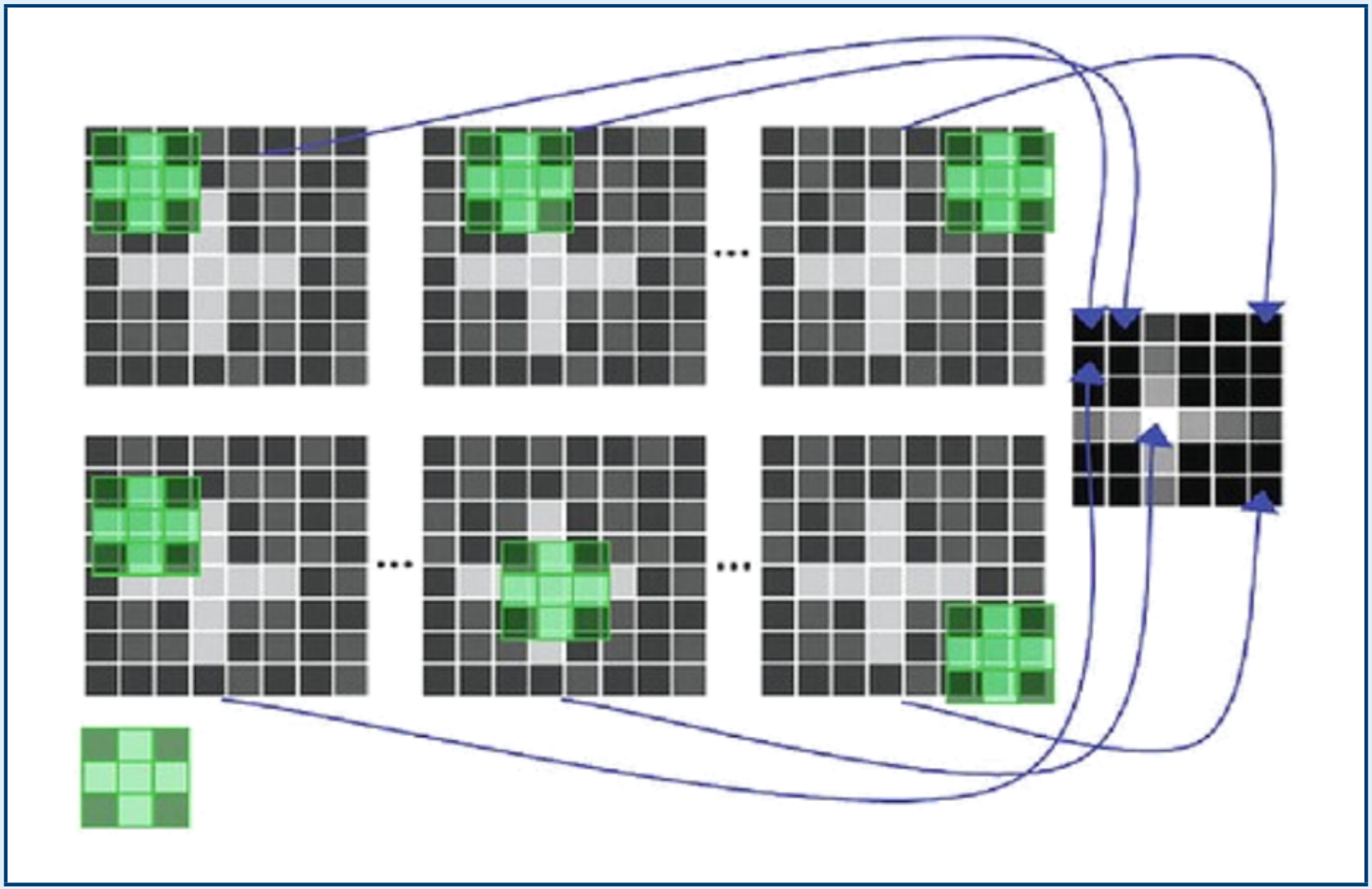


Convolutional NN[8]

- Images are difficult to classify because the same object can shift.
  - Naively, a dog on the left side of the image won't be seen the same as a dog on the right side of an image.
- Focus on extracting features of the input.
  - Search through parts of the image looking for particular features.

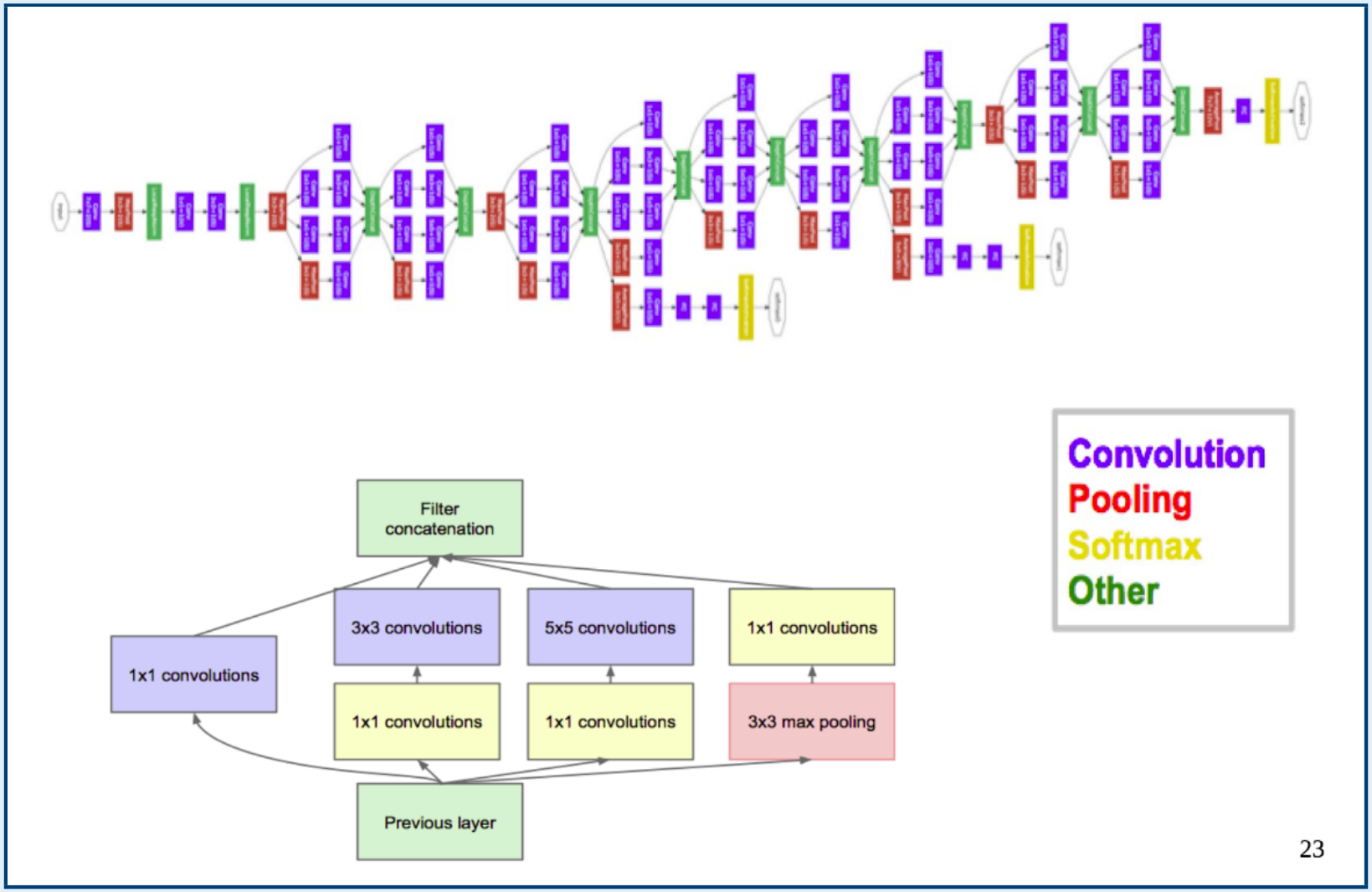
[8] Lecture 6

Convolutional NN[9]



[9] Lecture 6

Convolutional NN[10]



[10] Lecture 6

Training Deep NNs

Some Other Concepts at the School

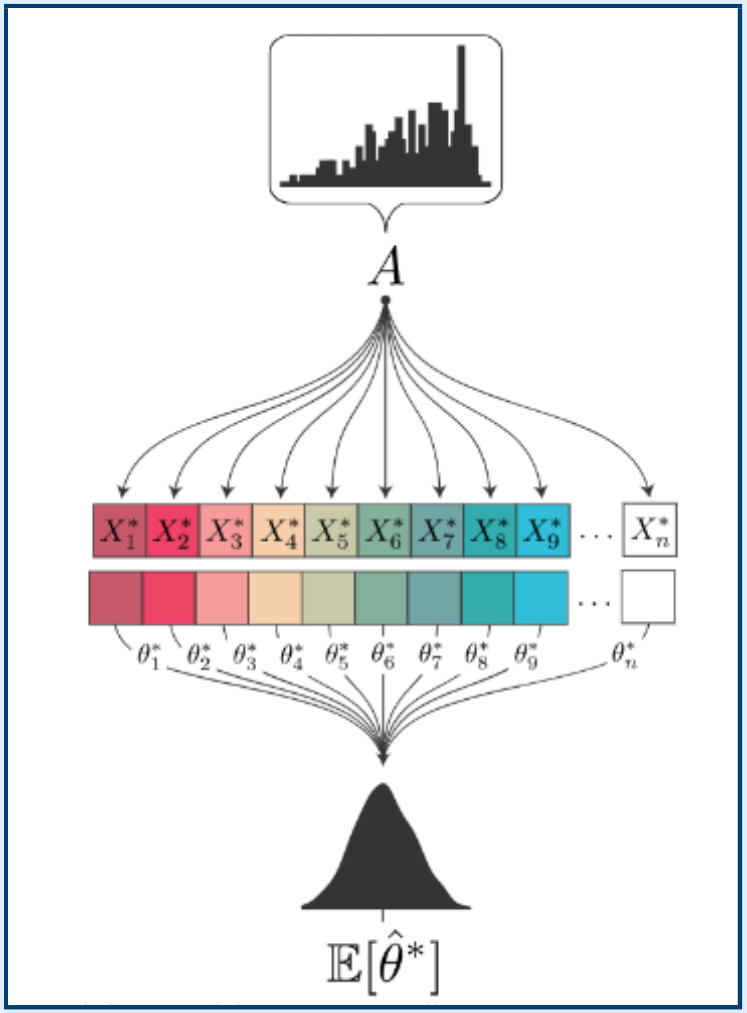
- Hyperparameter (ie. model parameter) optimization.[11]
- Bayesian optimization[12]
- PCA, Autoencoders, k-means clustering, other unsupervised learning techinques[13]

[11] Lecture 4
[12] Lecture 4 + Special lecture on beer!
[13] Lecture 7

Outline

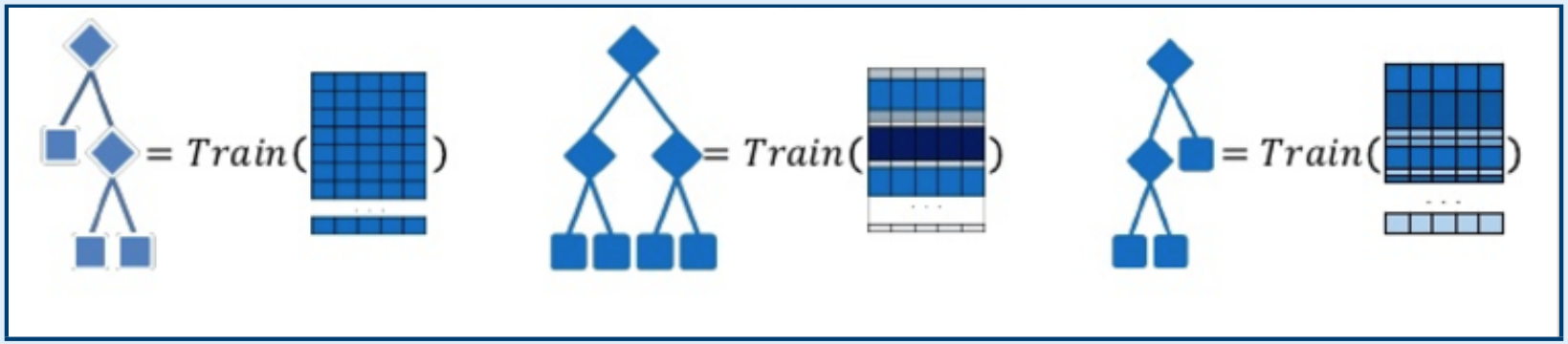
Bagging[14]

- Bagging is ensemble methods that reduces variance and overfitting by using sampled datasets.
  - Also known as bootstrapping
- Given a training dataset, we sample it to create sub-datasets.
- Then train separate models on each sub-datasets.
  - Average the outputs from the different models to determine the prediction.
- Frequently used with decision trees.



[14] Lecture 3

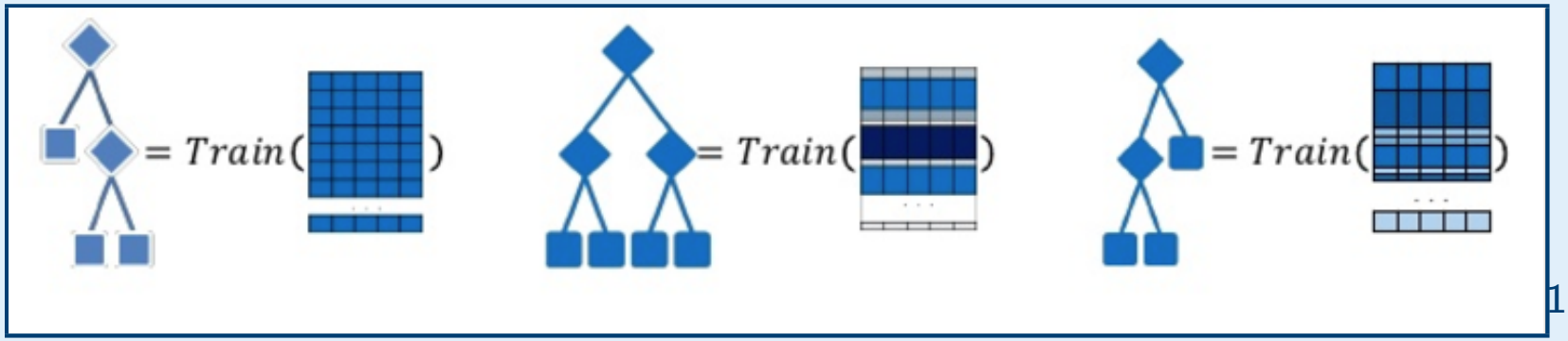
(Gradient) Boosting[15]



- An ensemble method which aims to take a collection of weak learners and combine them together into a strong learner.
- Start by training a weak learner on your dataset.
  - Then compute the accuracy of learner.
- Train another learner with increased weight given to the data where you previous learner(s) were least successful.
- Repeat for N learners and combine their results.

[15] Lecture 3

(Gradient) Boosting[17]



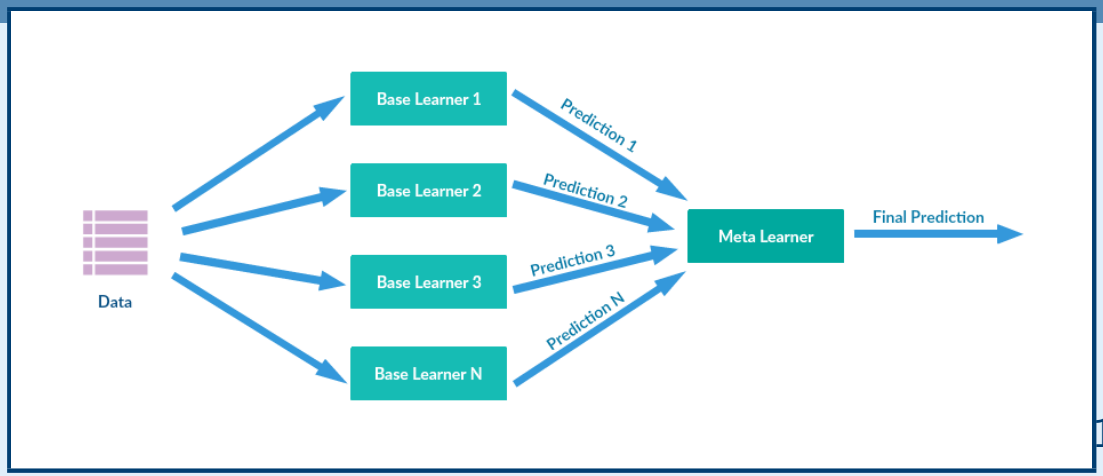
16

- Many variations, such as gradient boosting, which allows the use of gradient descent to optimize the weights for the new learners.
- Practically, some variation of gradient boosting seems to be a good way to go.

[16] Image from: https://www.slideshare.net/hustwj/scaling-up-machine-learning-the-tutorial-kdd-2011-part-iia-tree-ensembles

[17] Lecture 3

Stacking[19]



[18]

- Another ensemble method, where we train a new learner based on the output from other learners.
- Start by training a set of learners on the data.
- Feed the output of those learners into the input of another learner to provide a prediction.
- Can also feed the input data into the final learner.

[18] Image from: https://supunsetunga.blogspot.com/2016/06/stacking-in-machine-learning.html

[19] Lecture 2

Outline

Decision Trees[20]

- Some concepts and tools to be familiar with:
  - Bagging
  - Adaboost = A type of boosted decision trees
  - Extreme Gradient Boosting (XGBoost) = Boosting with the weights according to gradient descent
  - Random Forest = Bagging many decision trees which were learned with a random subset of features
- Particular boosting implementations have different choices for various parameters.
  - Generically referred to as BDTs.
- BDTs are frequently used for tagging and triggering

[20] Lecture 2

Neural Nets[21]

- Many of the ensemble methods can also be applied to NNs.
  - It is often more difficult due to the training time
- Frequently train over the dataset multiple times to converge on values for the weights.
  - Each iteration is known as an epoch.
- To do anything interesting, you need GPUs!
- CNNs are frequently used for image recognition.

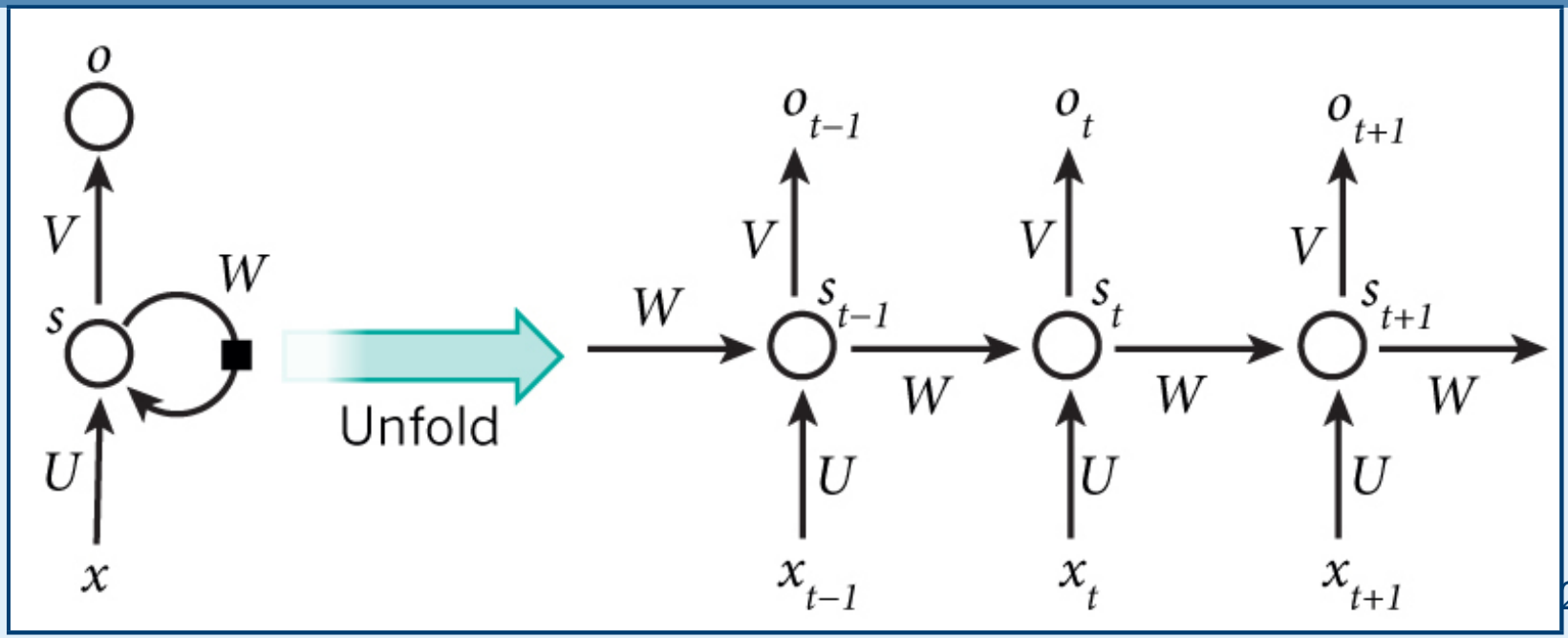
[21] Lecture 5

Regressors vs Classifier

- You will see both regressors and classifiers when looking at available implementations, etc.
- Regressors yield continuous values.
  - You are getting some estimate or prediction.
- Classifiers yield discrete values.
  - You are getting some class membership or categorical answer.
- Why do we care?
  - Both will understand attempt to characterize your data, but often you want to understand what your dataset looks like, so you'll often want the classifier.

Outline

Recurrent NN(RNN)[23]



[22]

- Assumes some relationship between each node in a network.
- Can be used to predict time series, natural language processing and prediction, etc

[22] Image from: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[23] Lecture 9

Recurrent NN(RNN)[25]

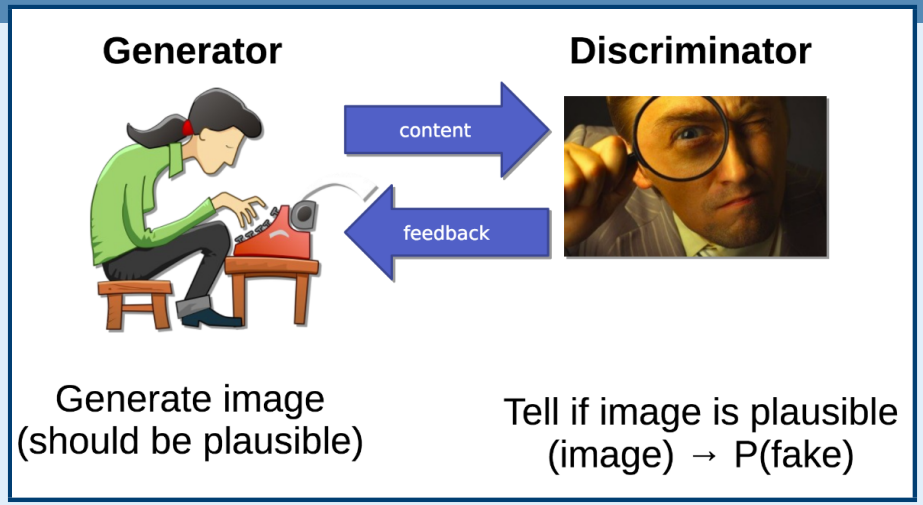
```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

[24]

- Trained on the Linux kernel and generates code!

[24] Image from: https://karpathy.github.io/2015/05/21/rnn-effectiveness/
[25] Lecture 9

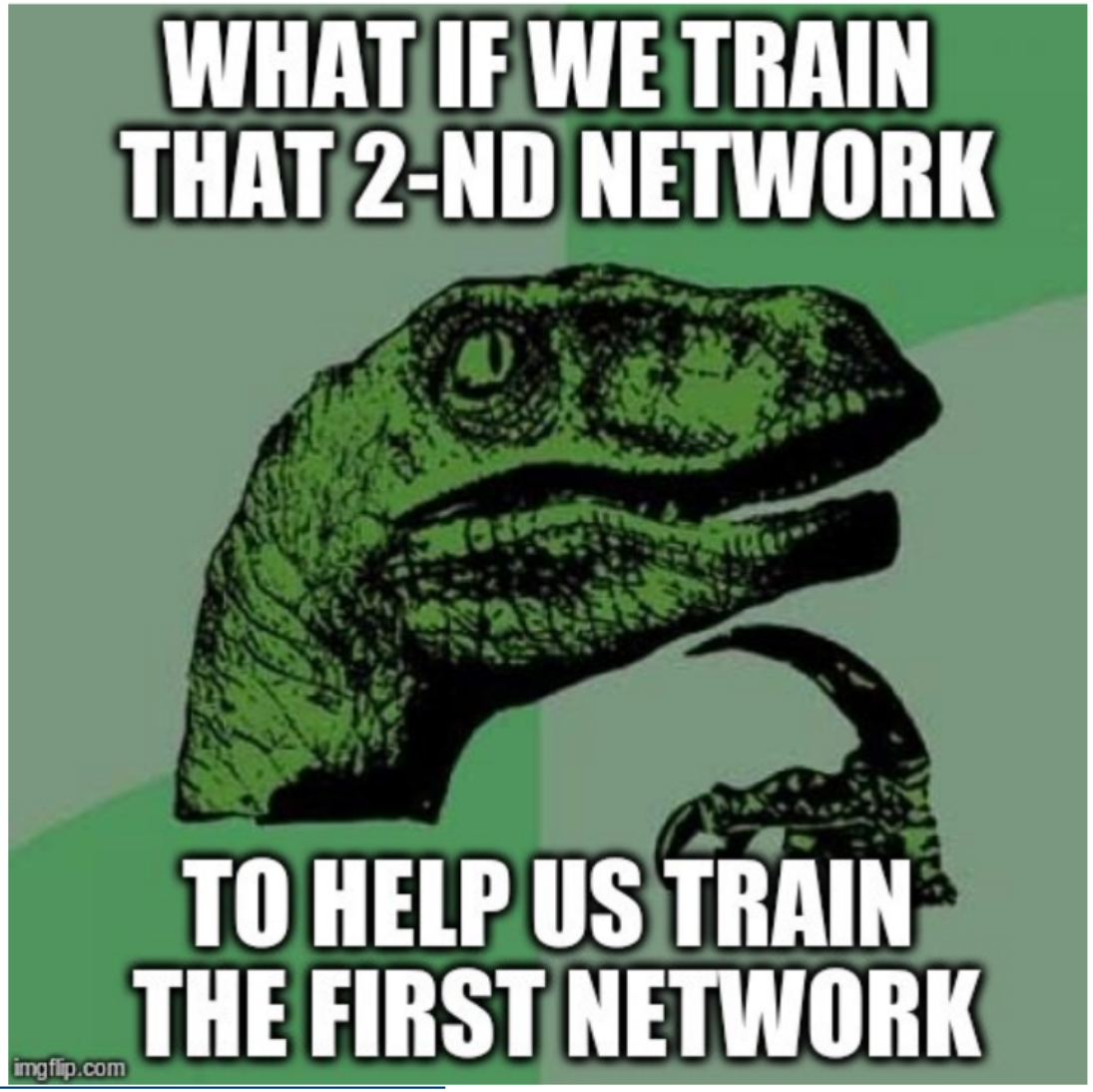
Generative Adversarial NN[26]



- The idea is to use one neural network to train another.
- Start by training a neural network on a dataset.
- Input noise into another network, and have the accuracy of that network measured by the first network.
  - The result is a network that takes noise and gives out realistic looking data
- Perhaps could be used for MC?

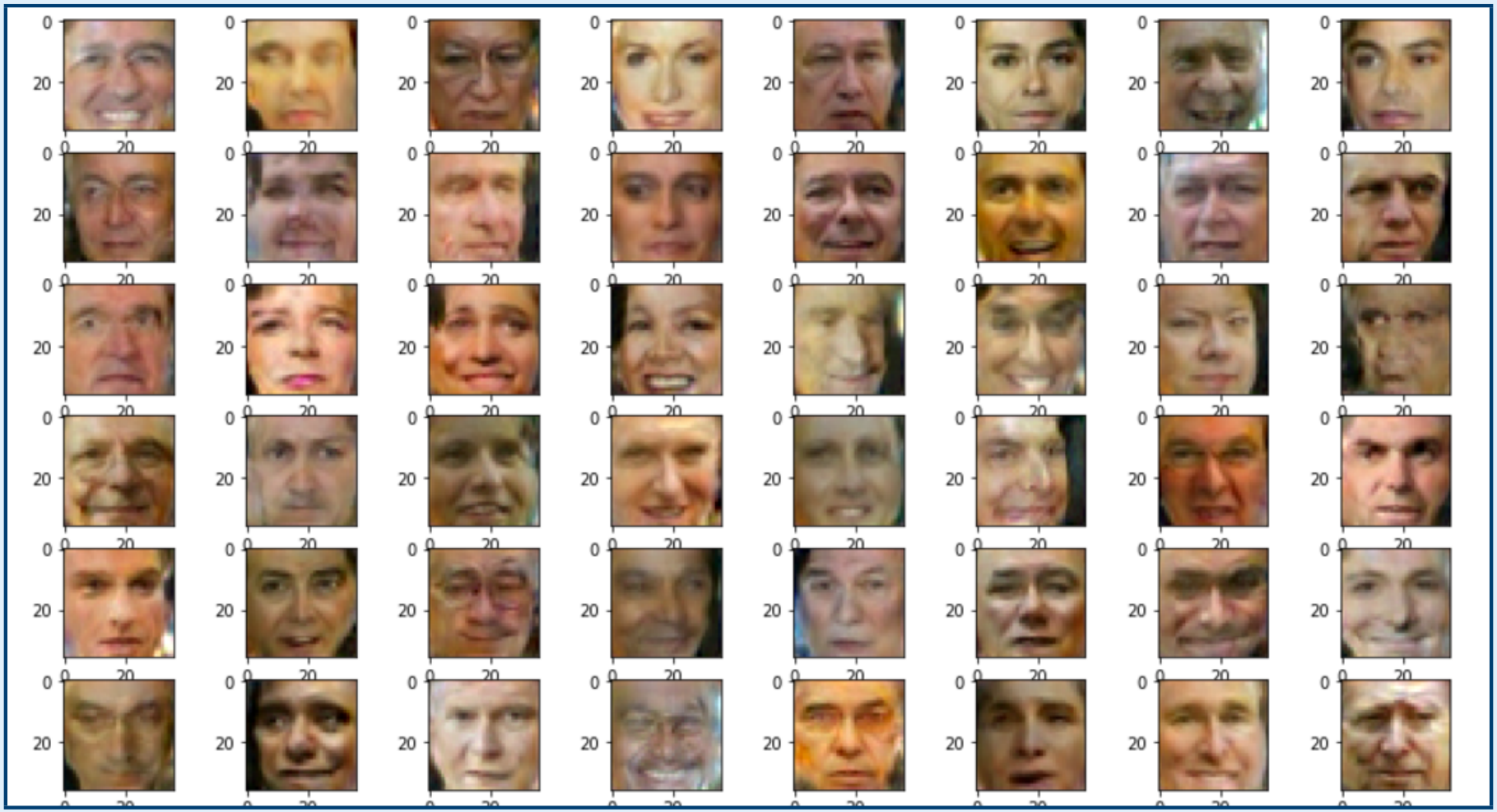
[26] Lecture 8

Generative Adversarial NN[27]



[27] Lecture 8

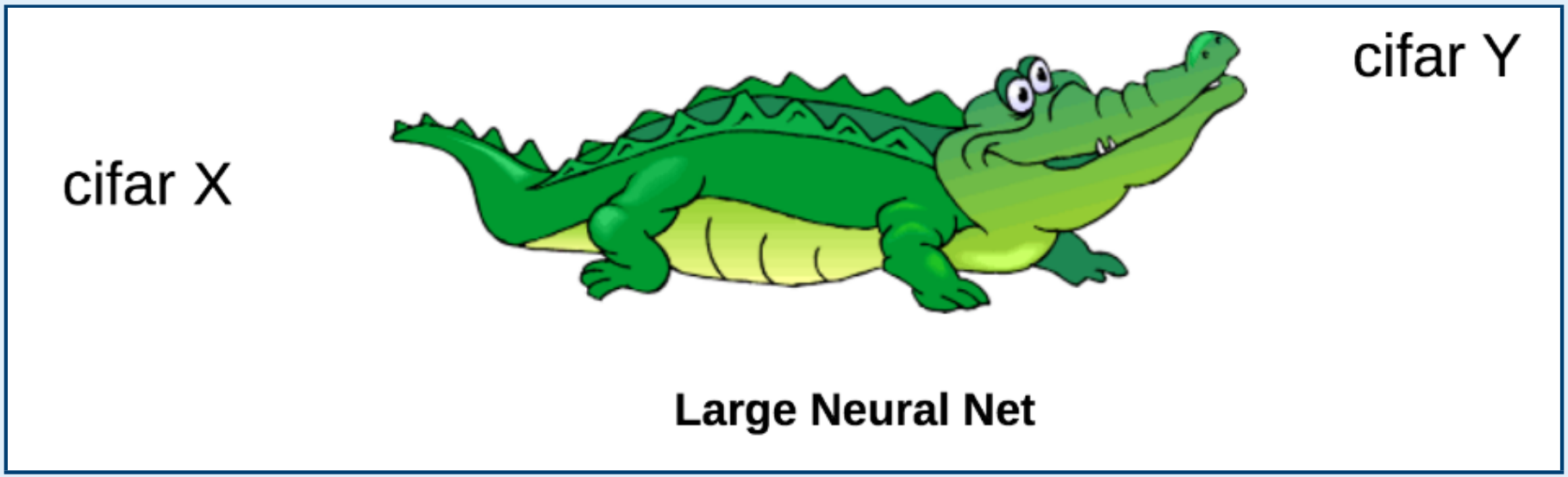
Generative Adversarial NN[28]



[28] Lecture 8

Transfer learning, pre-training, and fine tuning [29]

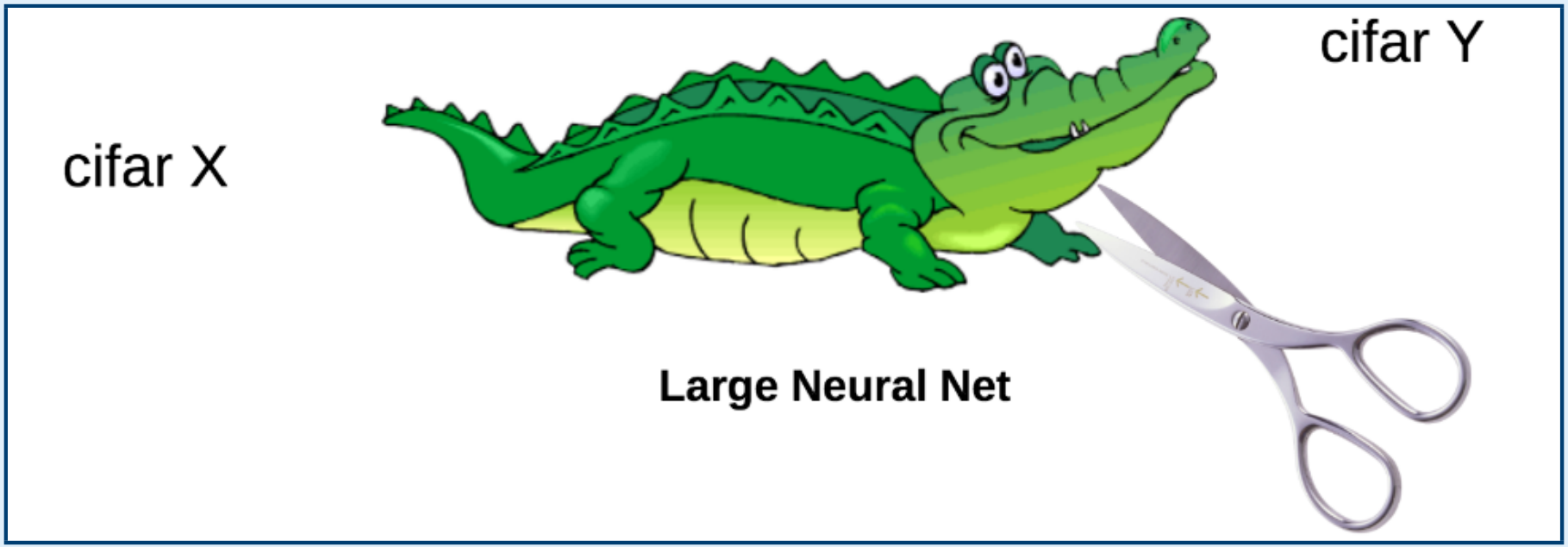
- We can also use existing trained models for our own purposes.
- To explain, we'll use this alligator (model):
  - First, we train our alligator



[29] Lecture "7"

Transfer learning, pre-training, and fine tuning [30]

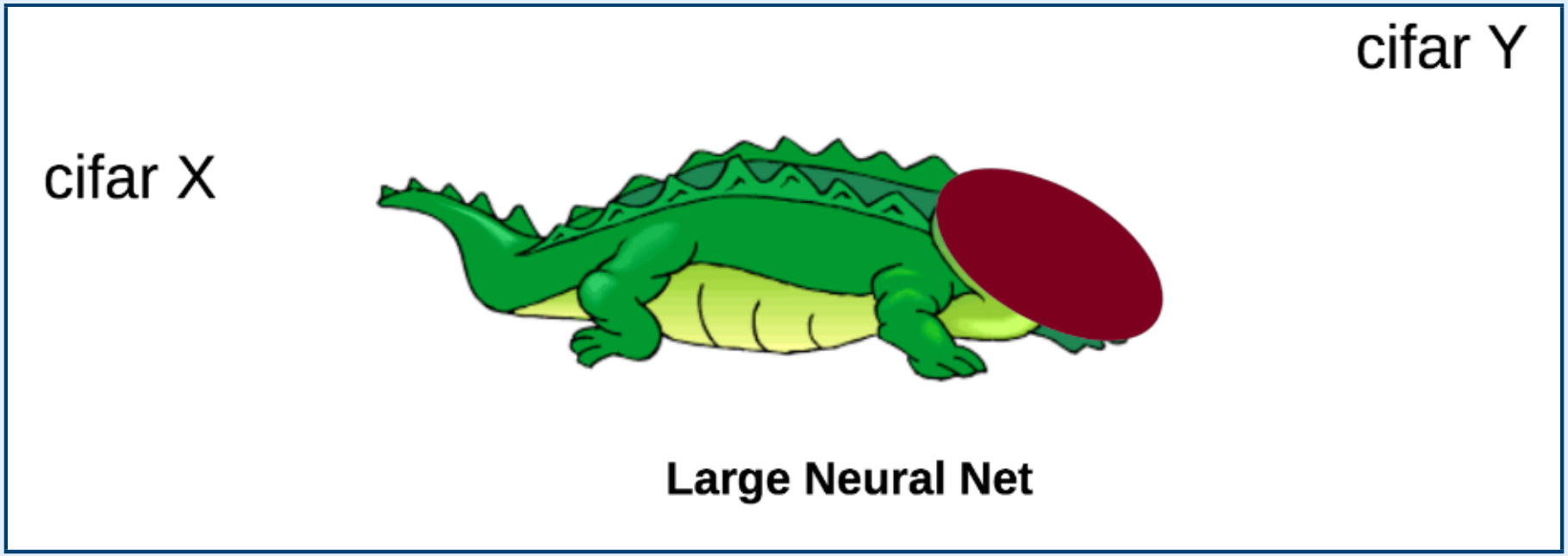
- We want to use that NN up to some intermediate layer
  - Clip off the rest



[30] Lecture "7"

Transfer learning, pre-training, and fine tuning [31]

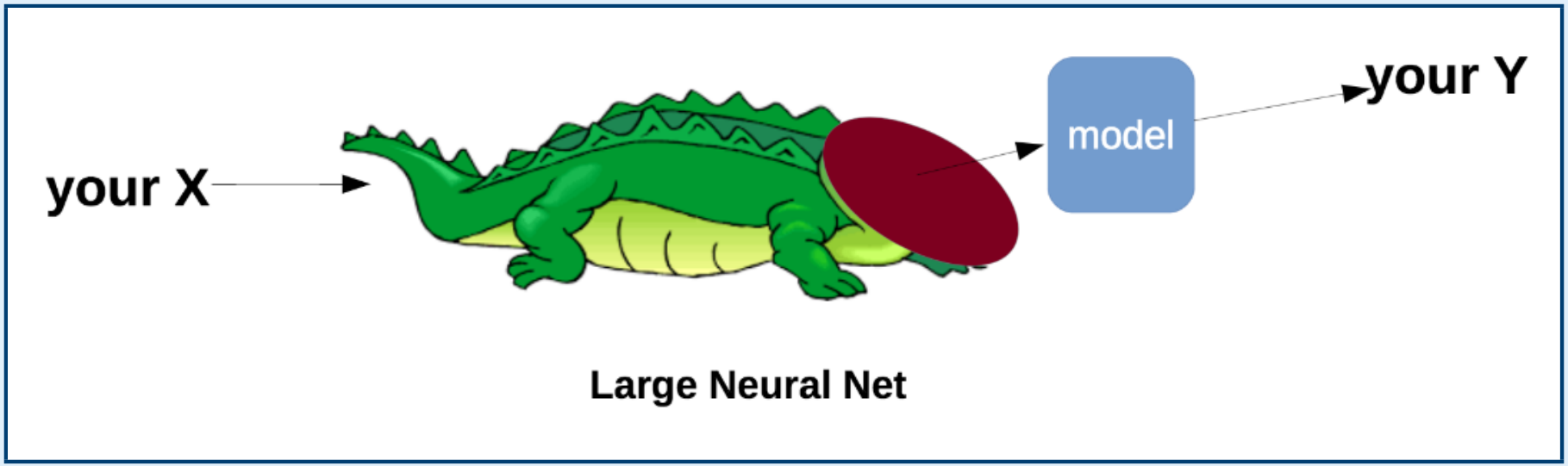
- The features already learned by the NN are fairly universal.
  - We keep their weights frozen



[31] Lecture "7"

Transfer learning, pre-training, and fine tuning [32]

- We use the output from the model as an input into our own model.
  - Our model can then be trained, even if the data is entirely different.
- Can also unfreeze the weights in the previous network to further optimize.



[32] Lecture "7"

Transfer learning, pre-training, and fine tuning [33]

- Useful to leverage other's hard work and training.
  - Can also be useful if you don't have a lot of training data.
- Lecture on day 5 about the "model zoo" is generally recommended.
  - Direct link available here.
- Nice explanation of some available models and how to take advantage of them.
- A more in-depth explanation with a good discussion is available here.

[33] Lecture "7"

Outline

Scikit-learn

- General purpose python machine learning package
  - Implements nearly everything in ML, although not always the best option for every situation.
- **Fantastic API**
  - Same API for many very different algorithms.
  - Other packages frequently support this API, even if it's not their main interface!
- Install with `pip install scikit-learn`

Setup

```
# List of our models
models = []
# Split true data X and y into half training and half test
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(
                                        X, y, test_size=0.5)

# For scoring predictions
# Area under ROC curve
from sklearn.metrics import roc_auc_score
```

Define Some Models

Boosted Decision Tree:

```
from sklearn.ensemble import AdaboostClassifier
models.append(AdaboostClassifier(n_estimators = 10))
```

Ensemble of Decision Trees:

```
from sklearn.ensemble import RandomForestClassifier
models.appned(RandomForest(n_estimators = 10))
```

XGBoost:

```
from xgboost import XGBoost # Not a part of sklearn!
models.append(XGBosot(n_estimators = 10))
```

Even TensorFlow

Perhaps not the most practical way to do so, but...

TensorFlow:

```
from skflow import TensorFlowDNNClassifier
models.append(TensorFlowDNNClassifier(
                          hidden_units=[10, 20, 10]))
```

Train and Test Your Models

```
for model in models:
    # Fit the model
    model.fit(X_train, y_train)
    # Predict the probability of the test set
    prediction = model.predict_proba(X_test)

    # Evaluate the score using area under the ROC curve
    score = roc_auc_score(Y_test, prediction[:,1])
    print("Model: {0}, Score: {1}".format(model, score))
```

Outline

Software and Tools

- Python is the way to go here.
  - Perhaps R is another option. (But probably not for HEP)
- numpy and pandas are required for handling data.
  - They play a similar role in the python ecosystem that ROOT does for HEP.
- However, in comparison to ROOT, numpy and pandas are:
  - Better supported! (by the entire python data analysis community)
  - Better documented!
  - Better designed!
  - They are fairly straightforward to learn and they operate very well on large datasets.
    - They are compiled c underneath, so generally rather fast (but perhaps not as fast as only c/c++)
  - There are even plugins for ROOT data.

Software and Tools

- Scikit-learn is extremely useful for most ML problems!
  - Nearly everything discussed is available!
    - If not in sklearn, at least via the same API.
  - TMVA is not as up to date and cannot use other packages as easily.
- NNs are one major exception to the sklearn ecosystem (although skflow does exist...)
  - Keras is a high level package for defining and training NNs
    - Often backed by TensorFlow or Theano (can switch between them with little effort)
- TensorFlow (TF)
  - Developed by Google.
  - Can be used directly, but generally more difficult!
- Best NN package is less settled, but TF is good.

Selecting the Best Algorithm

- Although there are some guidelines for selecting algorithms, you generally can't make recommendations
  - There are frequently many factors that determine which algorithm is best
- Fortunately, sklearn makes testing many algorithms very easy!
  - Just test many of them as in the example and it should get you going in the right direction.
- Even once you select an algorithm, what is the best design?
  - Again, the answer is not obvious a priori, but sklearn can help.

Tips and Tricks[35]

- Some ways to potentially avoid overfitting
  - Ensemble methods.
  - Early stopping of model training.
  - Monitor test vs train error.
- Parameter optimization[34]
  - Use tools like hyperopt.
- For CNNs, use a 3x3 filter unless you have a good reason for another value.

[34] Lecture 4
[35] Lecture 10

Final Thoughts[36]

- "Machine learning is about using prior knowledge about the problem wisely."

[36] Lecture 10

Resources

- Everything from the summer school is available online.
  - Includes presentations and hands-on seminar content.
  - Hands-on content is in the form of jupyter notebooks.
    - Can be viewed nicely online in GitHub, as well as downloaded, modified, and executed interactively.
- Main repository is here.
  - There is a tremendous amount of useful information (although not all of it will necessarily be useful in a vacuum)!
  - The MLHEP Indico page (available here) is your friend to figuring out the lectures and ordering!
    - Note that the titles and content start to diverge near the end when modifications were made to the schedule.

Resources

- My personal fork is here.
  - Contains my notes own notes, work and modifications from the seminars, and some information on the various challenges and solutions.
- Additional Kaggle competition if you want a challenge!
  - Available here, and closes on September 17.