NDev 23

Let's Talk Machine Learning

What are we talking about?

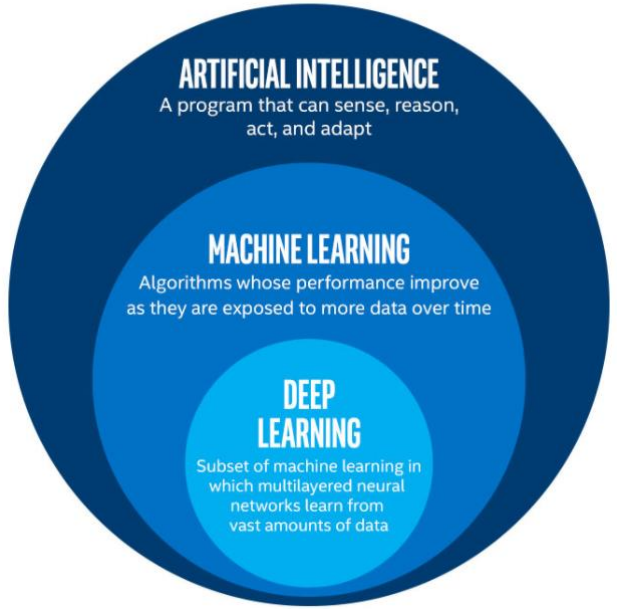
- ML: Foundations
- Explore what we can do using Scikit-learn
- Works is not good enough; make it work better!
- Build an intuition for what is going on under the hood
- Using TensorFlow
- Deep Learning
- Convolutional Neural Networks
- Wrap up

ML: Foundations

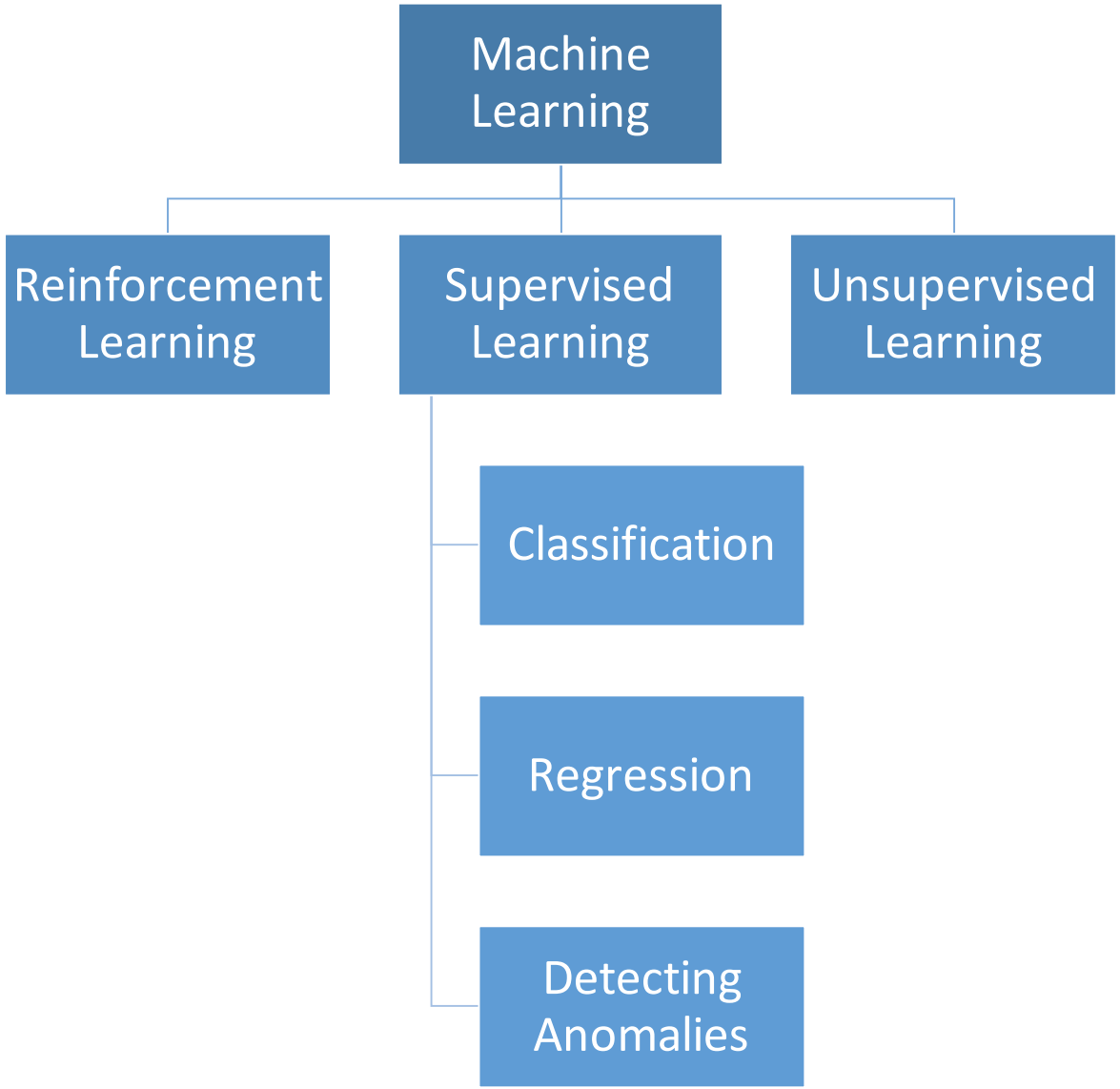
Orange or Apple?

```
def figure_texture()
def get_size()
def get_color()
def compute_probabilities()
```

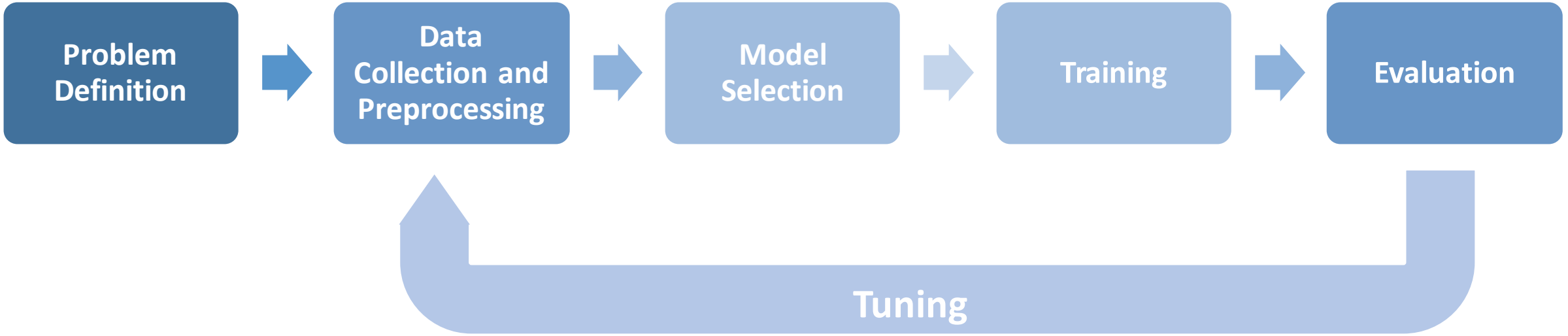
- AI ≠ML
- Machine Learning: Let the algorithm figure out the rules and parameters for itself, instead of hardcoding the rules
- Rules can be too complex, too many, hard to generalize etc.



ML: Foundations



ML: Foundations



Explore what we can do using Scikit-learn

A Standard Practical Example:

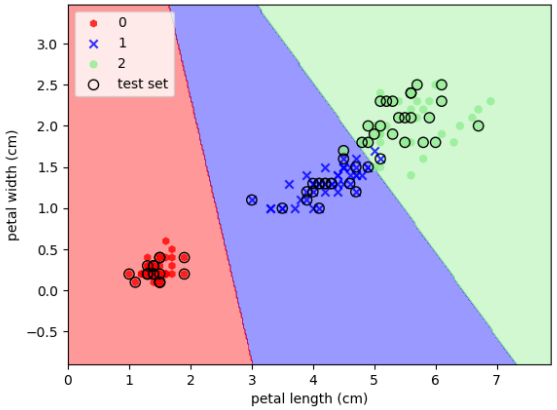
- The *Iris* flower dataset
  - Considers 3 species of the genus *Iris*: *Virginica, Setosa* and *Versicolor*
  - Given sepal length, sepal width, petal length and petal width (all in cm), classify an *Iris* flower as one of the three species

Let's give it a shot...

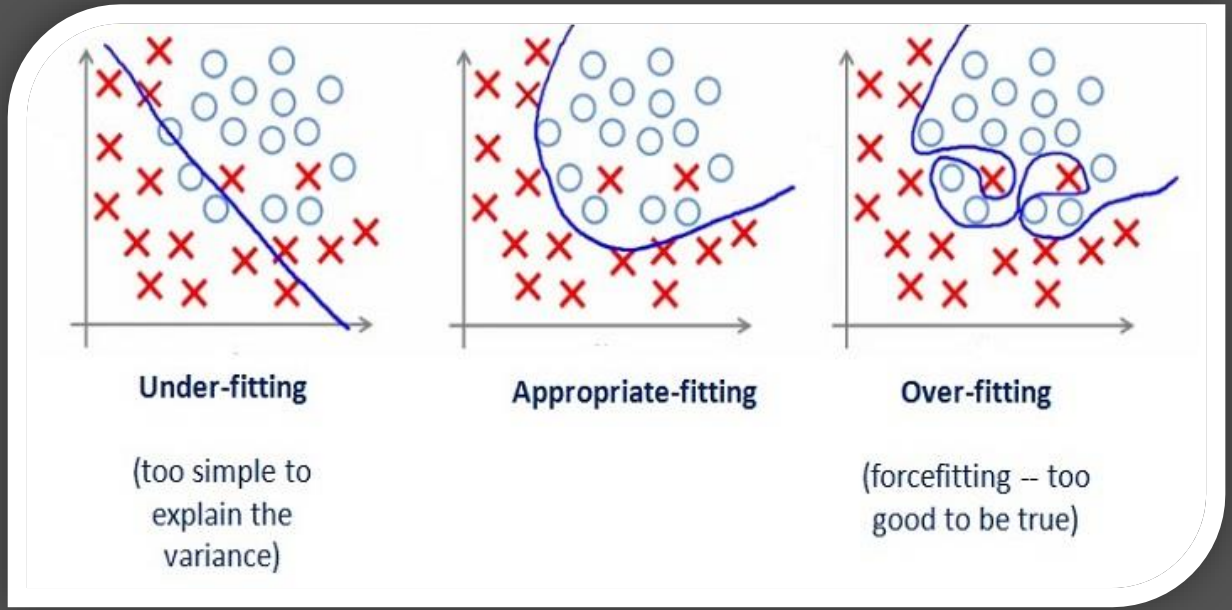
Building an intuition for what is going on under the hood

That was a lot of magic happening; after all, Scikit-learn is a high level API for ML.

But to move forward, we need to understand some core concepts



OVERFITTING AND REGULARIZATION



- Overfitting: Decision boundary fits training data very well, but fails to generalise to unforeseen data.
- Regularization: Often referred to by λ, it is essentially $\lambda = \frac{1}{C}$ that we were just working with. Regularization is a penalty term to discourage weights from growing in magnitude, and thus producing complex boundaries.

Decision Trees

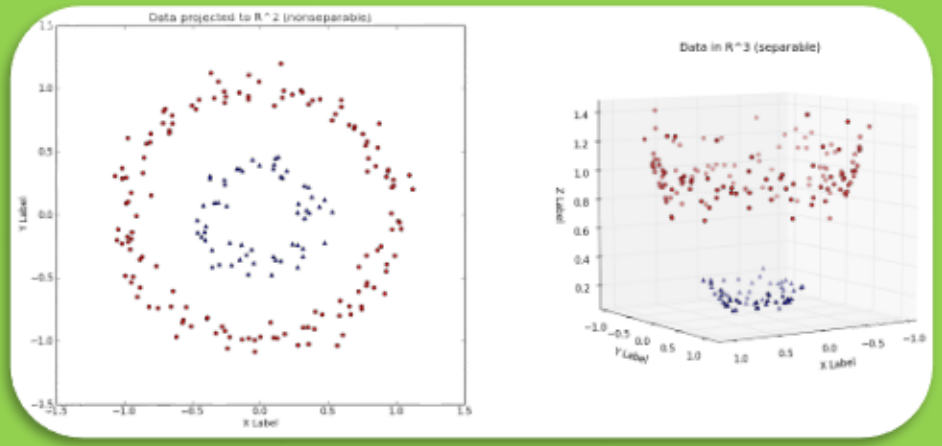
- Ask the right questions: finds a query condition that results in the greatest *Information Gain* (i.e. purifies a collection of observations with mixed classes into collections containing one class).
- On its own, prone to overfitting.

Random Forests

- Make multiple decision trees, each taking a random subset of features (with replacement)
- Take consensus among all trees
- Quite accurate, and can handle datasets with a lot of features.
- However, relatively take time to train

SUPPORT VECTOR MACHINES (SVMs)

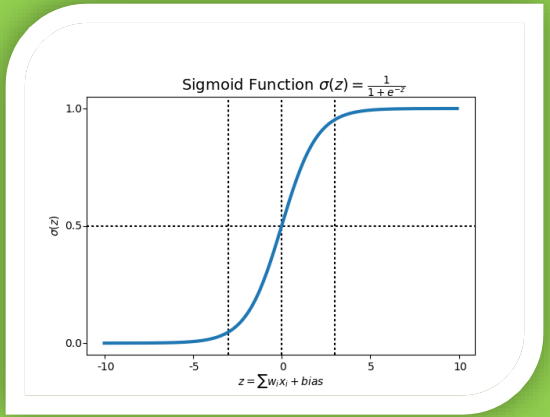
- Attempt to achieve a separating decision boundary with the greatest margin possible from nearby observations, in order to avoid overfitting.
- Can fit non-linear decision boundaries using the Radial Basis Function kernel trick: move the data to a higher dimension where they become linearly separable by a hyperplane



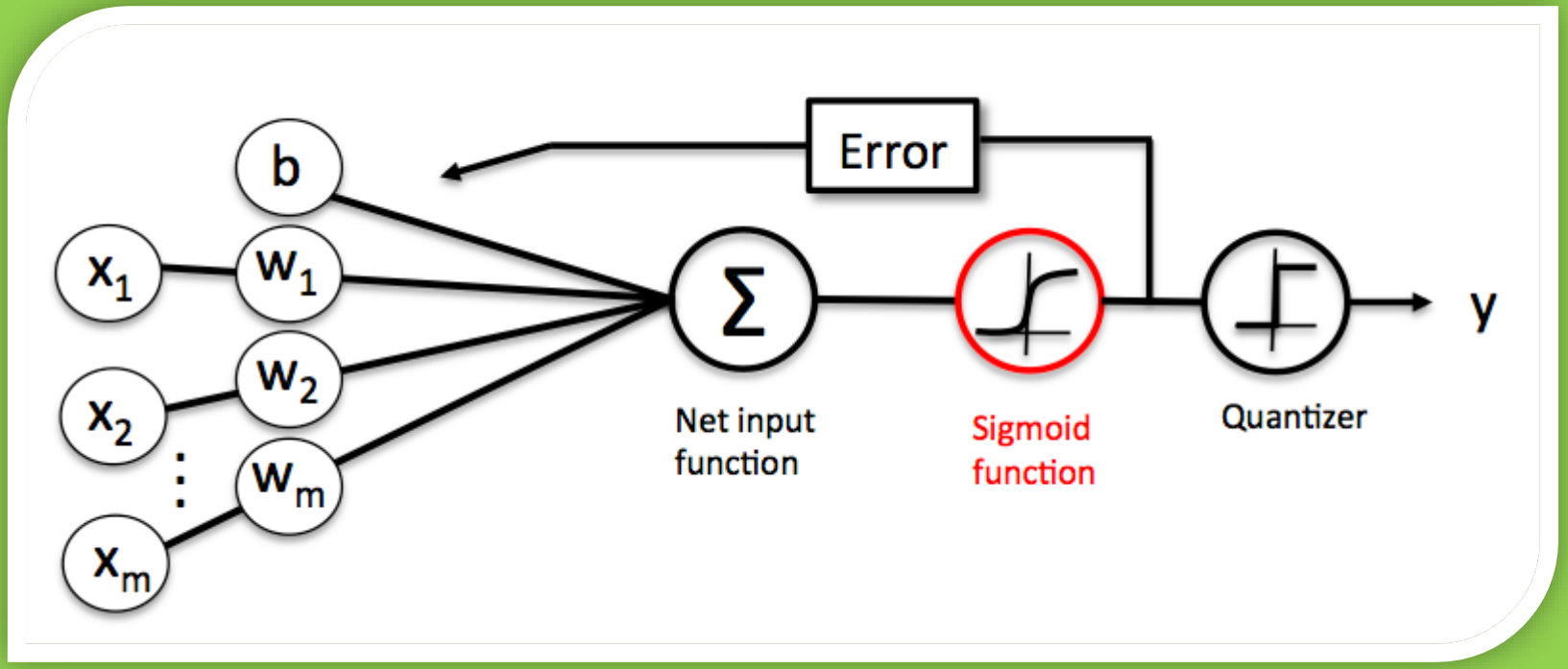
- Training dataset should be standardized.
- Scales well for large datasets with a lot of observations and features.

LOGISTIC REGRESSION – BINARY CLASSIFICATION

- Encode class labels numerically: {positive => 1, negative => 0}
- The algorithm has its internal state: its *weights*. Weights are simply integers representing how pronounced each feature is in the positive class. Store the weights as a column vector w
- Data is in the form of a matrix, X, whose rows represent *observations* and columns represent *features*
- *Compute a "net input" (logit) as a weighted sum of the features and add biases:* $z = X.w + \mathrm{b}$
- Pass z to "activation function", $\phi(z) = \frac{1}{1+e^{-z}}$. This specific activation function is called the *sigmoid* function, and it represents the probability of belonging to the positive class
- Quantize $\phi(z^{(i)})$ to make prediction:
  - Return 1 if $\phi(z^{(i)}) \geq 0.5$ else 0



LOGISTIC REGRESSION – BINARY CLASSIFICATION

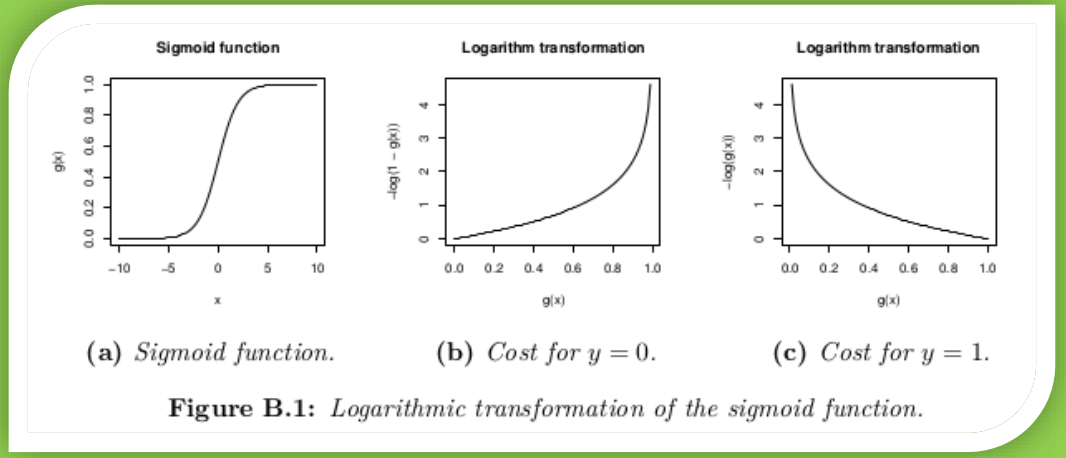


LOGISTIC REGRESSION – TRAINING

- Define a cost function J:
  - Decays exponentially as the algorithm's certainty in outcome matches the true classification
  - Grows exponentially as the algorithm's certainty in outcome opposes the true classification
- Goal is to minimise J as much as possible ↔ get accurate predictions! [*Insert a good bit of vector calculus here*]

MULTICLASS CLASSIFICATION:

- One-vs-Rest: Train individually for each available class – predict a yes or no answer for each class
- Or use a different activation function (more on this later)



(a) *Sigmoid function.* (b) *Cost for* $y = 0$. (c) *Cost for* $y = 1$.

Figure B.1: *Logarithmic transformation of the sigmoid function.*

Using TensorFlow

- Tensors: Multi-dimensional matrices
- The “flow” part is essentially computations linking the tensors

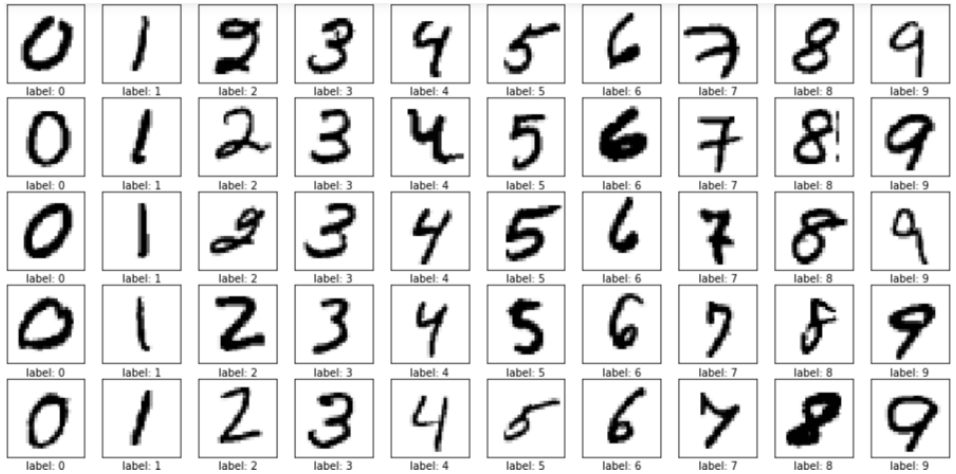
- A computation graph is created and stored into memory
- Need to run a session to perform computations

Using TensorFlow

A standard example: MNIST dataset

- Given a set of 28x28 images of hand-written digits, predict what digit is represented by a 28x28 image.

We will start off with a simple implementation, and then dig into more advanced concepts…

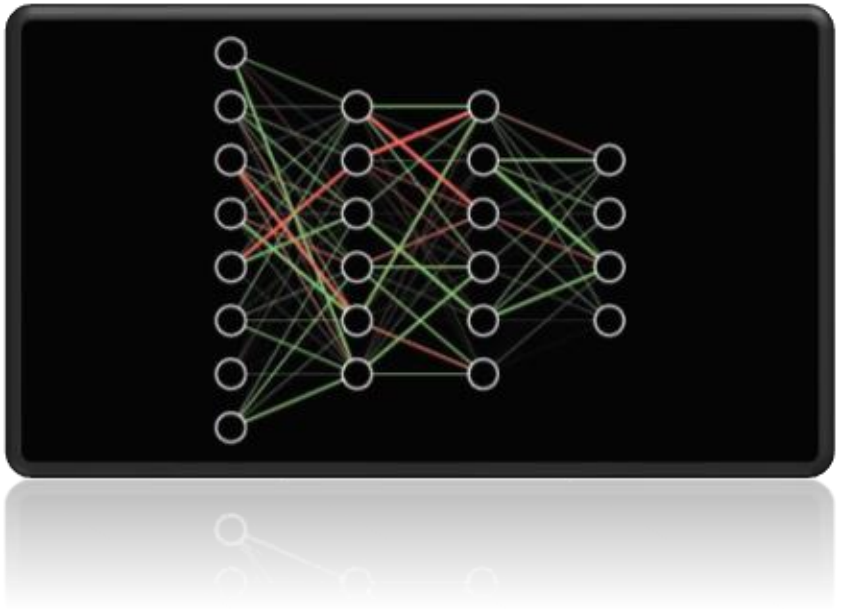


Deep Learning

- ANNs that are multiple layers deep (output of one layer is the input of the next one)
- Can extract features and perform classification
- For classification: very accurate, but take a lot of time to fully train.

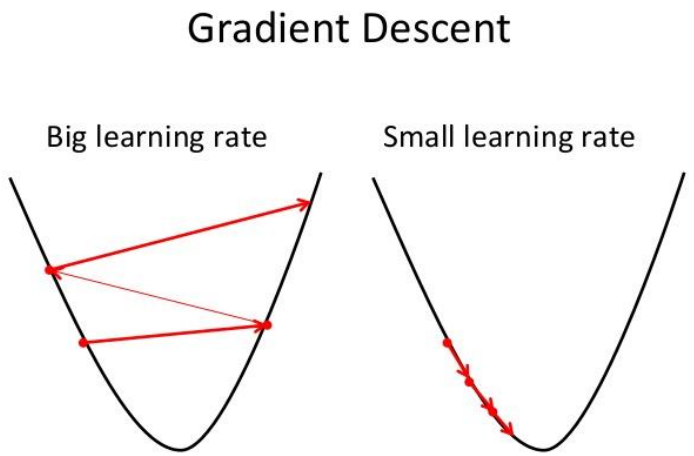
Intuition:

- Compare output to prediction, and penalise differences according to cost function
- Propagate updates back through the layers

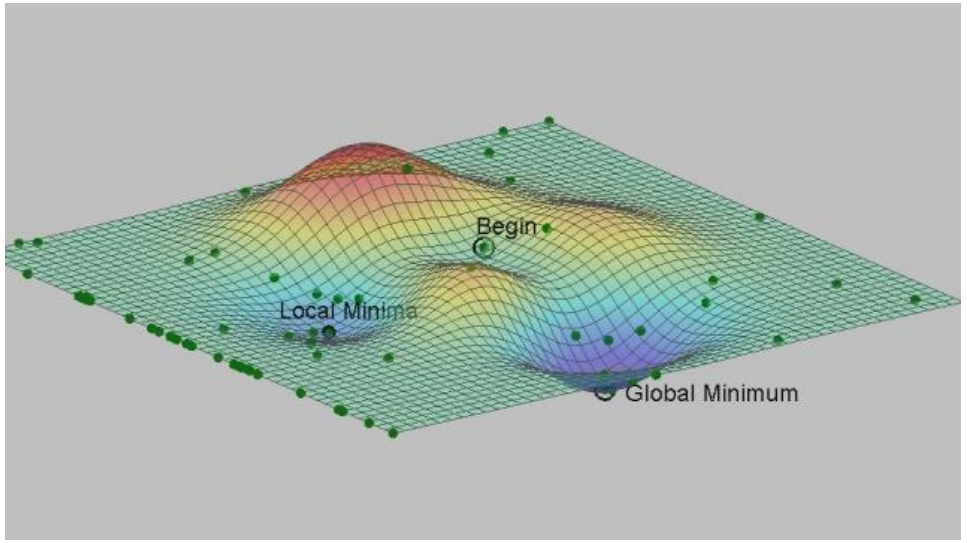


Optimization:

- Minimization of cost function by adjustment of degrees of freedom (weights and biases)
- Gradient Descent: Take steps in the direction of greatest decrease of cost function
- Size of steps determined by a parameter called the *learning rate*



In DNNs, there are often a lot of saddle points in the cost. Thus, we converge to local minima that can be far off from the global minimum.



The *Adam* optimizer is a more efficient and resilient optimization algorithm for DNNs

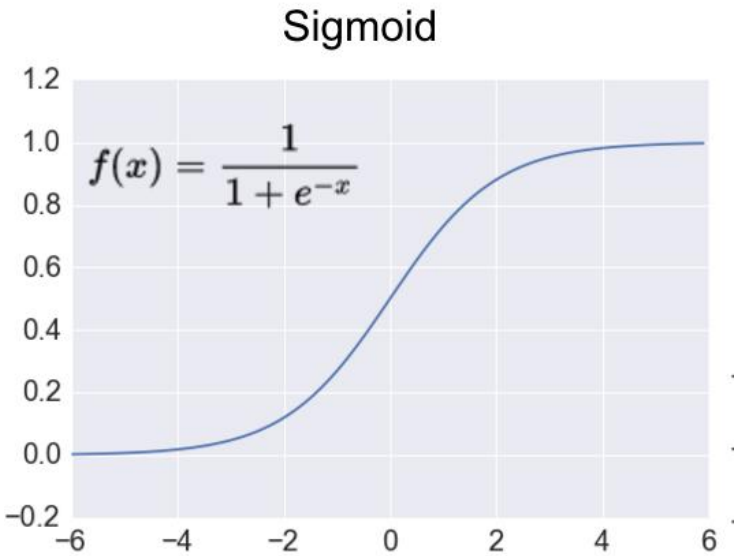
More on Activation Functions:

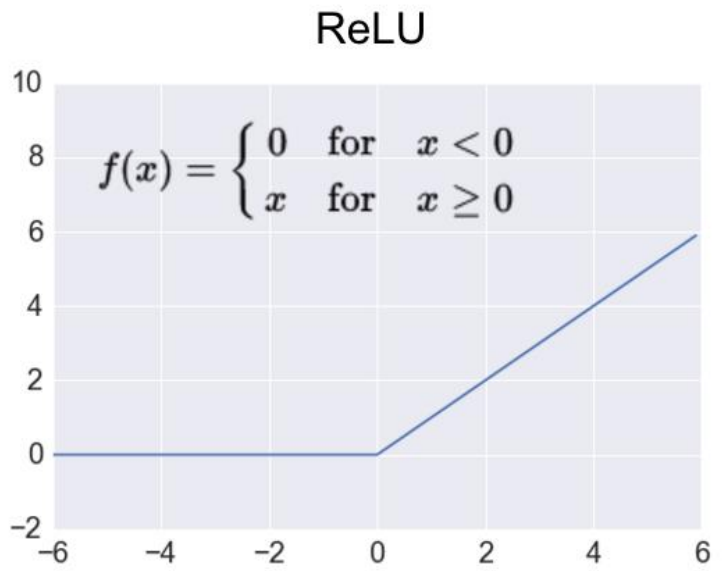
- **Softmax: Similar to the sigmoid function, but gives multi-class probabilities, rather than binary classification probability**

$$\text{softmax}\left(z^{(i)}\right) = \frac{e^{z^{(i)}}}{\|e^{z}\|}$$

- **ReLU (Rectified Linear Unit): Avoids vanishing gradient / gradient saturation**

$$\text{relu}\left(z^{(i)}\right) = \max\left(0, z^{(i)}\right)$$

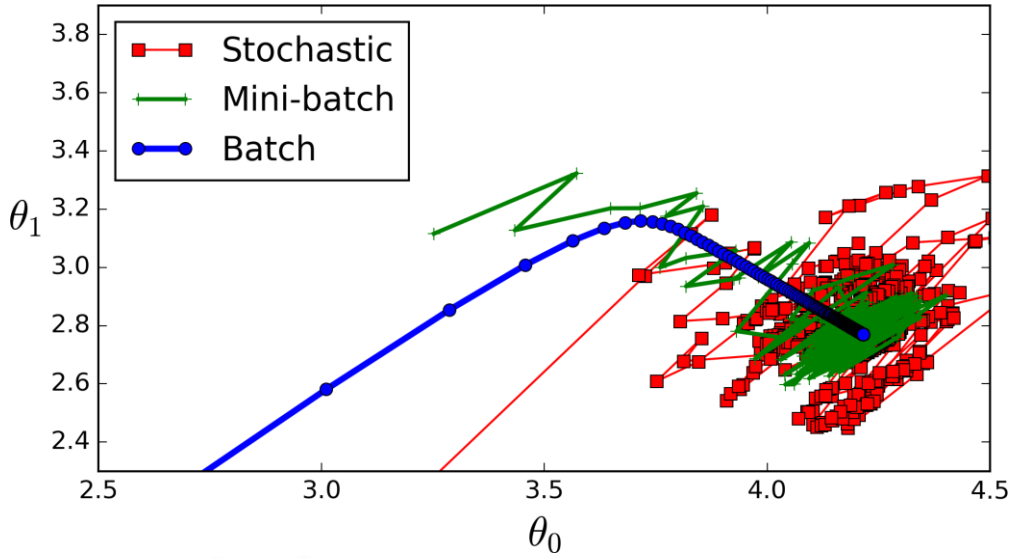




Deep Learning

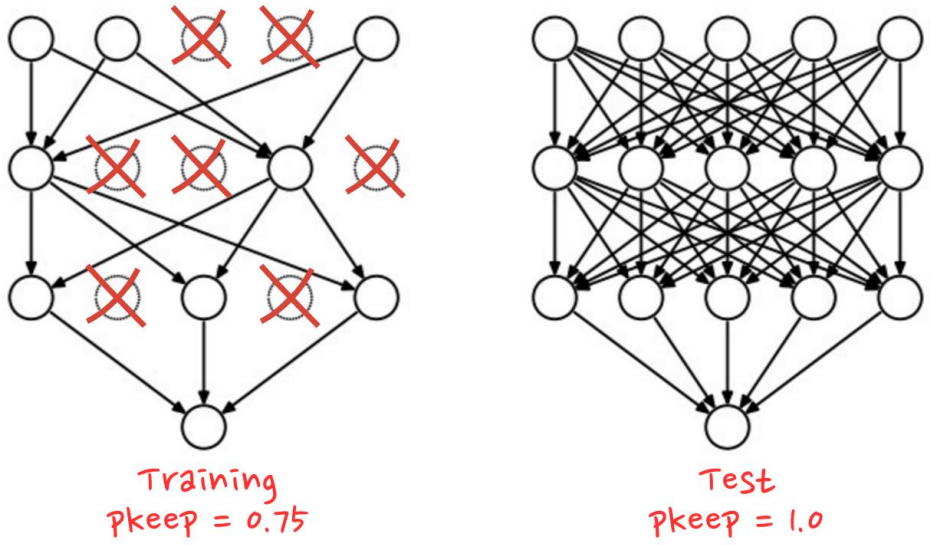
Mini-batch learning:

- Split the training data into small batches, and train the network on each batch, one at a time
- Compromise between training time and step accuracy



Avoiding Overfitting in DNNs:

- Dropout: Disable a proportion of neurons, at random, during the training phase of the network.

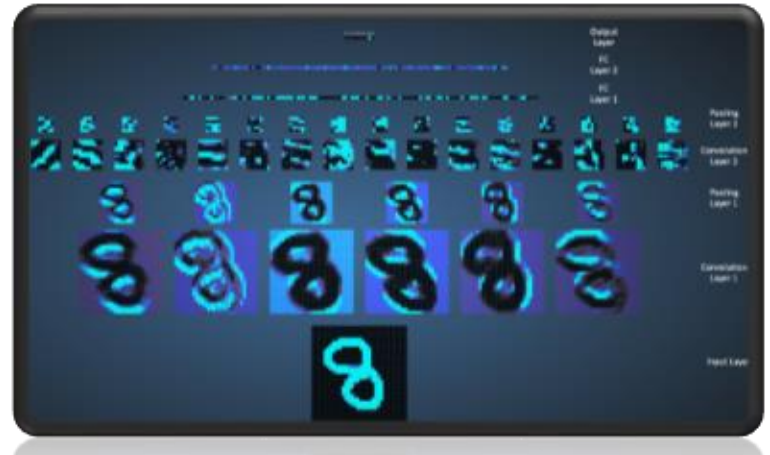


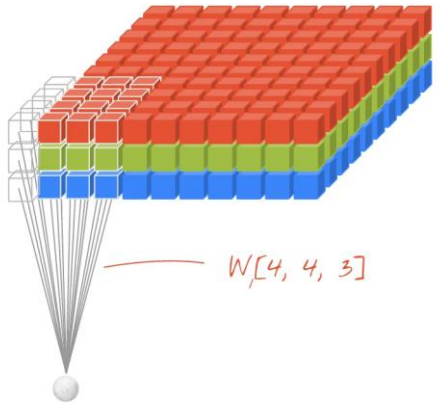
Convolutional Neural Networks (CNNs)

Recall that our DNN's input had the image pixels flattened into a 1D array: that's an issue, because we can not extract spatial meaning from the input!

CNNs:

- Take a filter and apply it over the image, generating a filtered version of the input (the filter aims to recognise patterns in the input, and the filtered version is the pattern matching result)
- Can stack convolutional layers to pinpoint more details
- Use a fully connected layer to take votes among filters





Convolutional Neural Networks (CNNs)

We can have multiple filters per convolutional layer, each responsible for spotting a given pattern.

As we go down the layers, we need to decrease the size of the layers (after all, we need to map all that data to 10 softmax neurons!)

Once done with convolutional neural networks, use a fully connected layer (like what we have seen before) to take votes among the filters, then pass output to softmax layer.

Let's see how this works...

Wrap Up

We covered a lot of tools, and each has a lot of knobs to turn!
Here is an overview to bring everything together:

- What metrics are important to you, the type of problem at hand, the amount of data and the required output, dictate which model(s) are fit for the job.
- Is your data linearly separable? You probably don't need the fancy neural networks, and SVMs, LR, or RFs will do just fine.
- Models come with their hyperparameters – experience helps in tuning them for best performance.
- Not all features are as important as each other
- Some tools work better with each other, and some have their own prerequisites to perform well
- DNNs are helpful for non-linear and more complex datasets. CNNs can help even further where proximity of features is relevant (if moving the features around does not matter, don't use a CNN)

Wrap Up

Want to learn more about ML?

- Google Codelabs: https://codelabs.developers.google.com/
- Inspiration from Martin Görner –
  Twitter: https://twitter.com/martin_gorner
  Tensorflow and deep learning – without a PhD:
  https://www.youtube.com/results?search_query=martin+gorner+Tensorflow+and+deep+learning

- Siraj Raval: https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A
- Check MUN Computer Science Society tutorials page:
  https://muncompsci.ca/resources/tutorials/

Thank you all for coming!