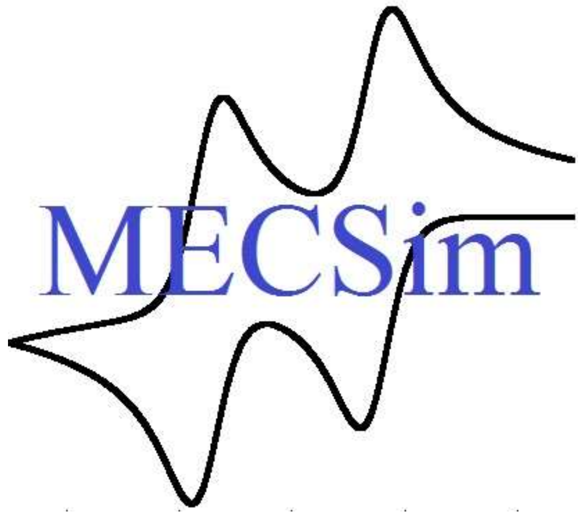
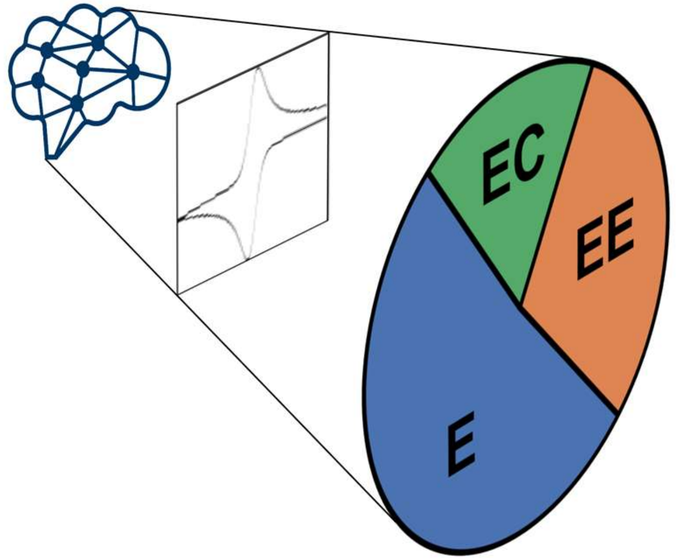
Application of Machine Learning to experimental results from analytical chemistry





Gareth Kennedy, Ph.D.
Data Engineering Manager, Australia Post
Affiliate of the School of Chemistry, Monash

garethkennedy

Overview

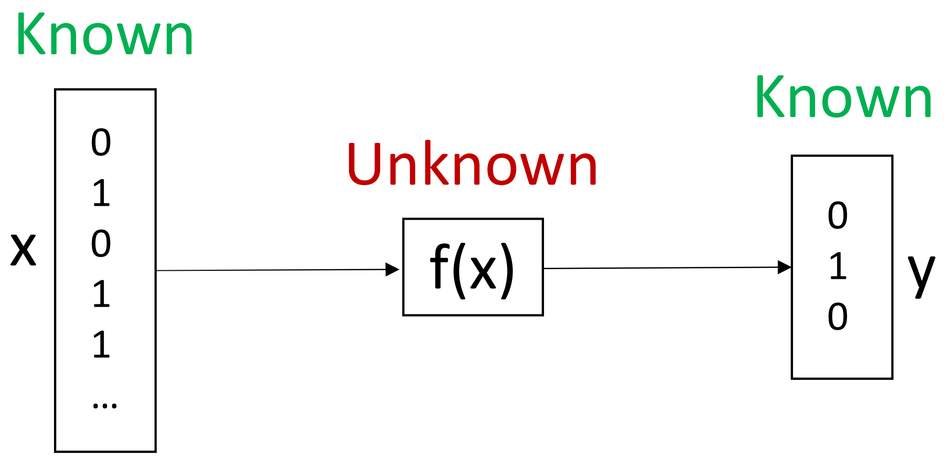
- Introduction to Machine Learning
- Limitations of ML classifiers
    - Overfitting
    - Training data
- Using deep neural networks in electrochemistry
- Need for domain knowledge
    - AI is not intelligent
    - Sanity checks
- Using ML tools to augment role of a scientist

Introduction to Machine Learning

- Basic definition is a model that learns (increases prediction accuracy) from experience (increasing data)
- Three types of ML:
  - **Supervised**: Model learns from labelled data. Aim is to create a general model trained on some labelled data provided by a human expert
  - **Unsupervised**: Model learns by finding patterns (e.g. clusters) present in the data itself. No input required from humans. Often called data mining.
  - **Reinforcement**: Model learns by trial and error with a reinforcing reward given for favourable outcomes (e.g. winning at Go) and a punishment for unfavourable outcomes (e.g. losing a game)
- We'll focus on an example of supervised learning to demonstrate the principals and limitations of machine learning

Supervised Learning Model

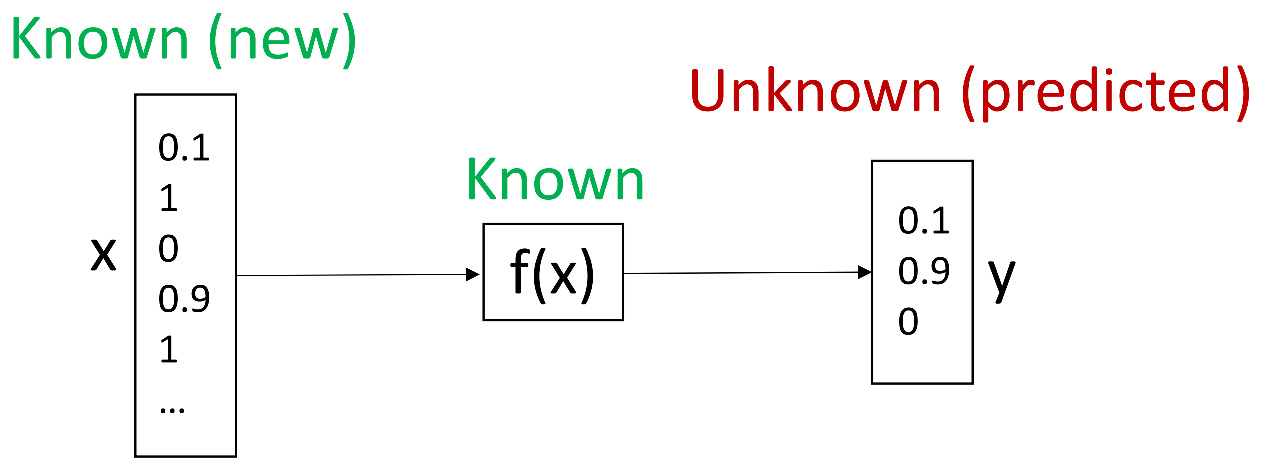
- Can be thought of as a function that maps an input (x) to a label (y)
- Training:



- Data consists of known inputs and labels
- Solve for the model parameters itself using linear algebra and optimization mathematical methods

Supervised Learning Model

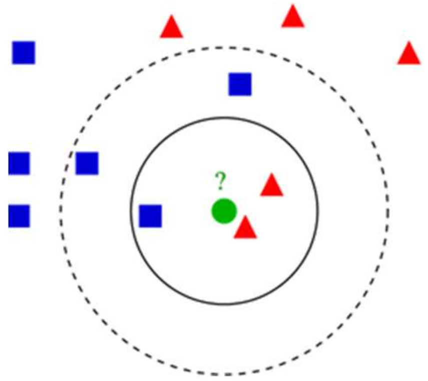
- Can be thought of as a function that maps an input (x) to a label (y)
- Prediction:



- Parameters for the mapping function f(x) are now known from training
- A prediction (aka a classification) is made of the label for a new input x

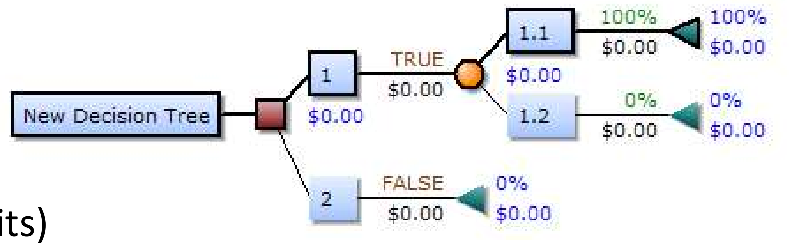
Example Models, f(x)

K-Nearest Neighbours (kNN)



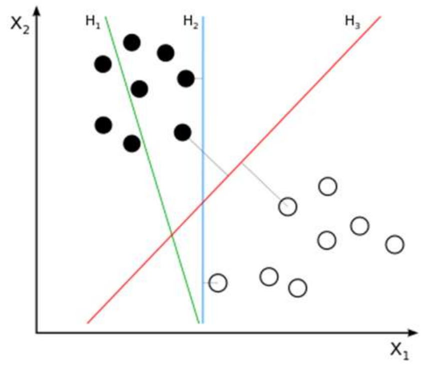
No parameters

Decision Trees (and Random Forest)



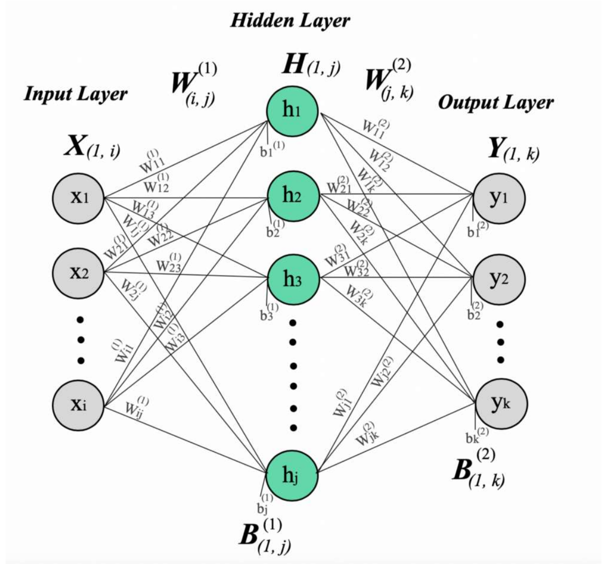
O(splits) parameters

Support Vector Machine (SVM)



O(10) parameters

Artificial Neural Network (ANN)



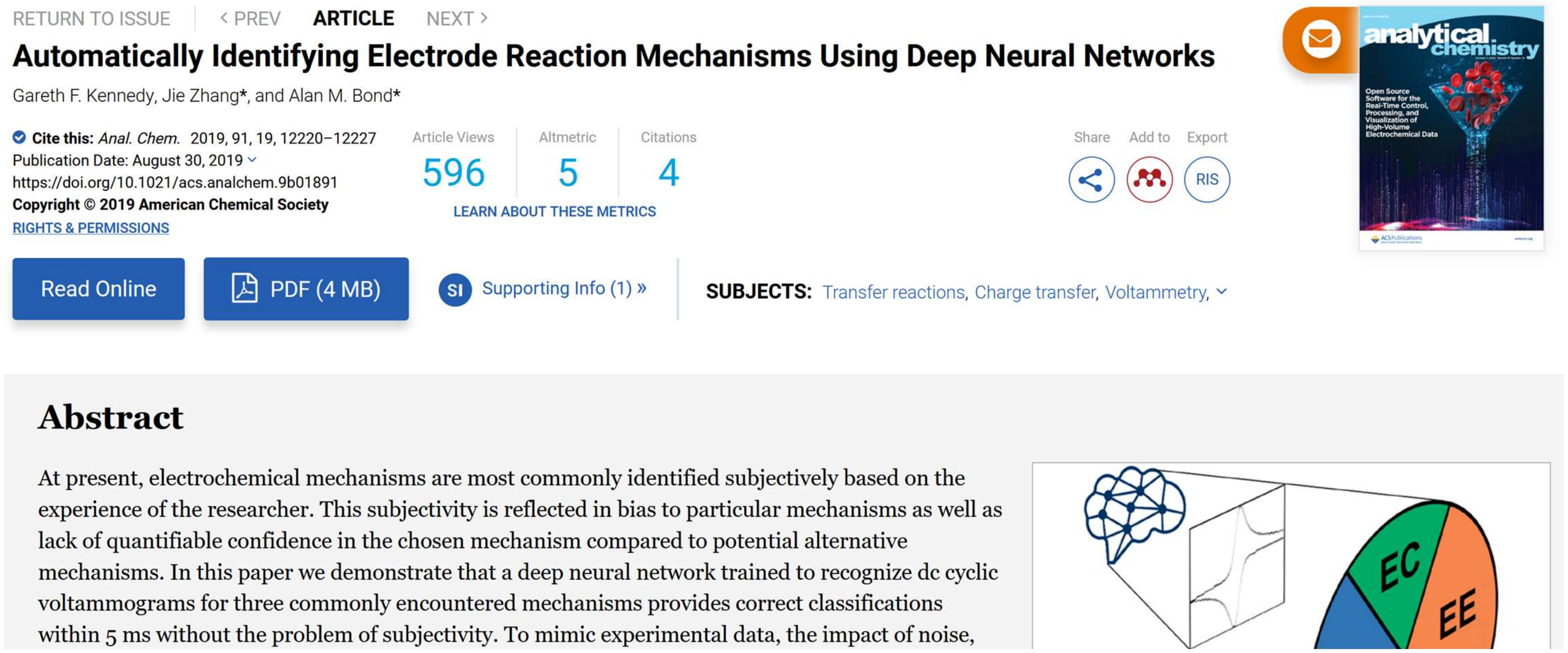
Lots of parameters
Typically > O(10^5)

Limitations

- Overfitting:
  - Huge problem when there are model parameters than train data examples (e.g. early covid-19 DNN hack models)
  - Need for lots of unique examples in training data
- Training data issues:
  - Training data is based on historical data:
    new cases or rare edge cases are not included
  - Unbalanced training data: e.g. fraud detection where the number of anomalies is 1000s of times less frequent than normal transactions
- Aim for a general model that is robust to unexpected inputs, unbiased, safe and becomes more accurate as data is added
- Deep knowledge of the data from a human needed as a sanity check

Using Deep Neural Networks in Electrochemistry

https://pubs.acs.org/doi/abs/10.1021/acs.analchem.9b01891



RETURN TO ISSUE | < PREV **ARTICLE** NEXT >

Automatically Identifying Electrode Reaction Mechanisms Using Deep Neural Networks

Gareth F. Kennedy, Jie Zhang*, and Alan M. Bond*

Cite this: *Anal. Chem.* 2019, 91, 19, 12220–12227
Publication Date: August 30, 2019
https://doi.org/10.1021/acs.analchem.9b01891
Copyright © 2019 American Chemical Society
RIGHTS & PERMISSIONS

Article Views 596 | Altmetric 5 | Citations 4
LEARN ABOUT THESE METRICS

Share | Add to | Export RIS

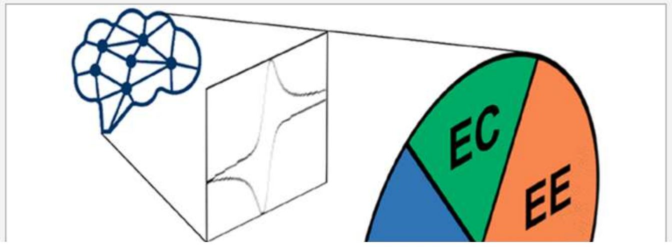
Read Online | PDF (4 MB) | SI Supporting Info (1) »

SUBJECTS: Transfer reactions, Charge transfer, Voltammetry,

analytical chemistry
Open Source Software for the Real-Time Control, Processing, and Visualization of High-Volume Electrochemical Data
ACS Publications

Abstract

At present, electrochemical mechanisms are most commonly identified subjectively based on the experience of the researcher. This subjectivity is reflected in bias to particular mechanisms as well as lack of quantifiable confidence in the chosen mechanism compared to potential alternative mechanisms. In this paper we demonstrate that a deep neural network trained to recognize dc cyclic voltammograms for three commonly encountered mechanisms provides correct classifications within 5 ms without the problem of subjectivity. To mimic experimental data, the impact of noise,



Automatic electrode reaction identification

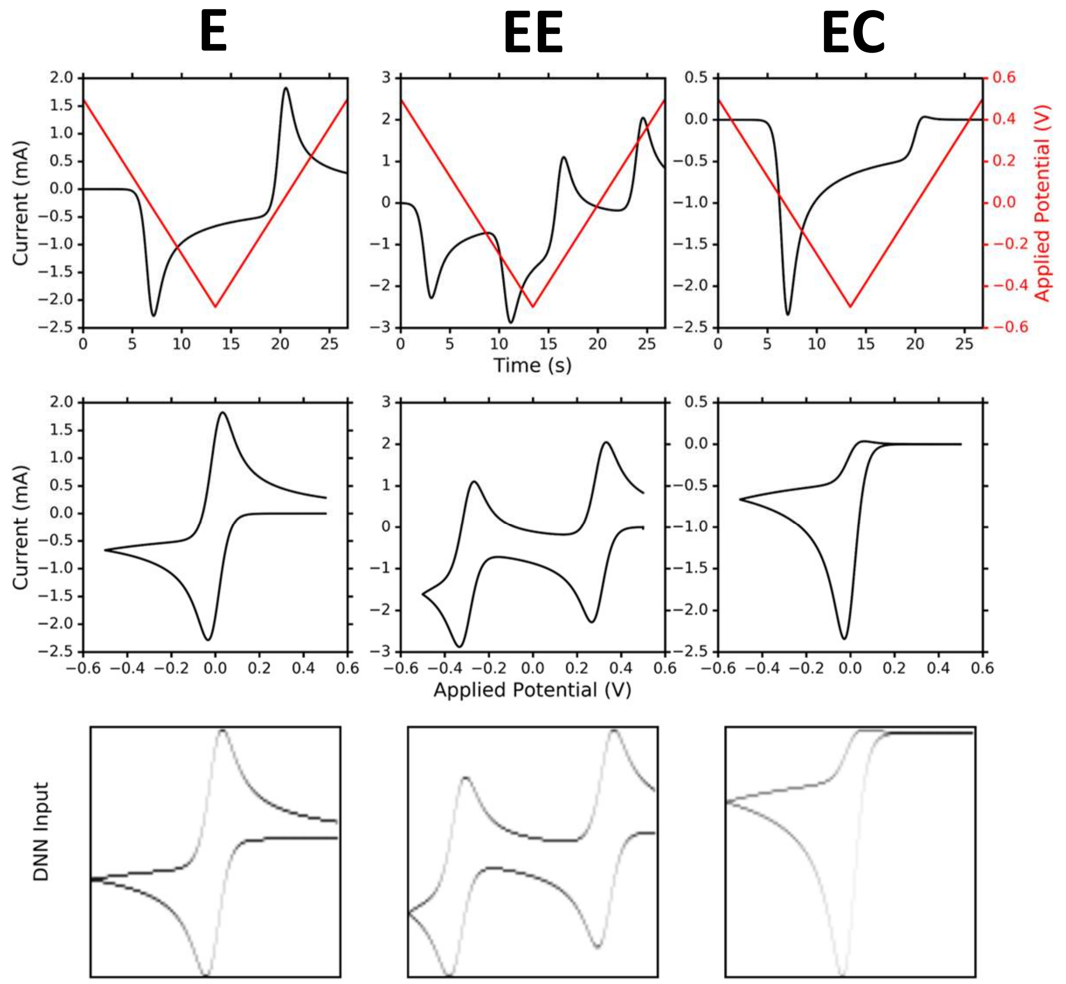
- Currently, classifying an electrochemical mechanism takes an expert with years of experience
- Electrode reactions occur in:
  - Biosensors (e.g. diabetes and other automatic health monitoring devices)
  - Substance sensors (e.g. explosive residual detection in airports)
  - Fuel cell and battery technology (e.g. electrode efficiencies)
  - Energy efficient reduction of CO_2 (e.g. climate change offset)
- Parameter fitting is required but many potential mechanisms
- Anything that helps to automatically narrow the possible mechanisms is a huge time saver for scientists

Physical system

- A voltage is applied to the working electrode and the current response depends on the chemical reactions at the electrode surface
- Examined three commonly encountered mechanisms:
    - **E**: single electron transfer; A + e = B
    - **EE**: two subsequent electron transfers: A + e = B, B + e = C
    - **EC**: electron transfer followed by chemical reaction: A + e = B, B = C
- Have software that can generate all the (labelled) data we need
    - MECSim is public via: http://garethkennedy.net/MECSimDownload.html
- Convert the dc cyclic voltammograms that a researcher would examine (current vs voltage) into a machine learning friendly image

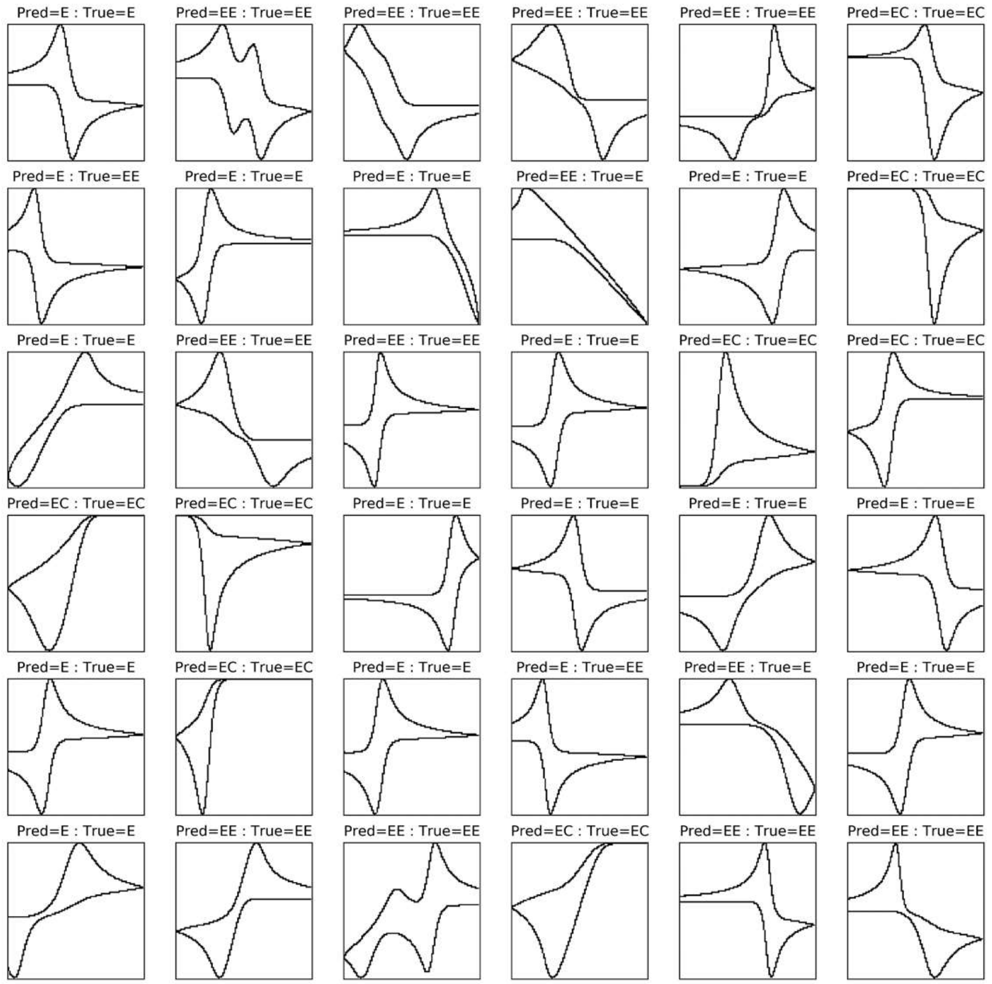
Data

- Applied voltage in red on the top panels
- Simulated current in black
- DC voltammograms in second row
- Human would identify the mechanism based on the shapes (# peaks etc)
- ML friendly 100x100 representation at bottom



Training data

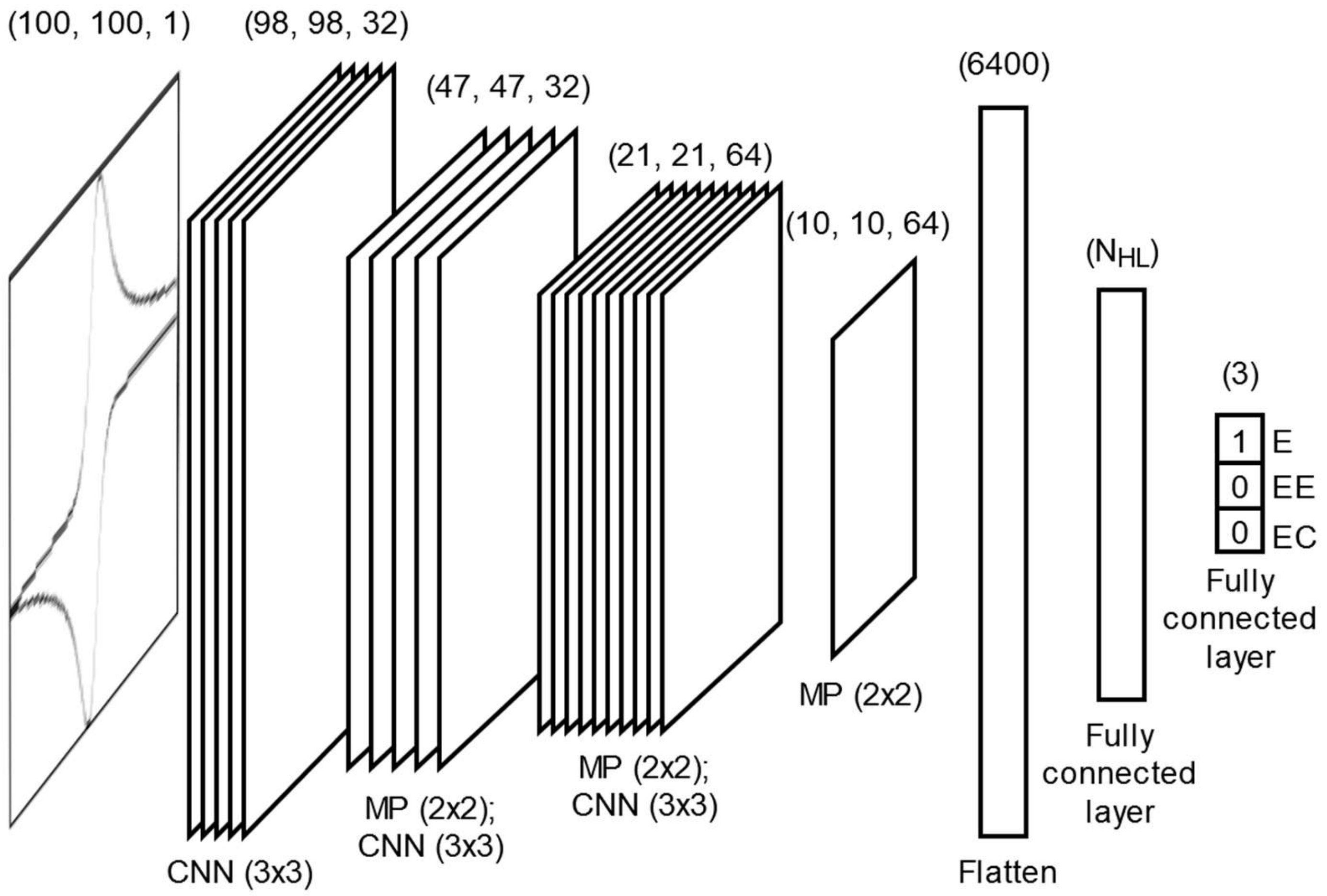
- Want the data to cover as **diverse** range as possible (here a variety of shapes)
- Want to **minimize bias** in the data (here some parameter combinations will make one mechanism indistinguishable from another)
- For this example we could carefully chose the range of parameters used to generate our training data set – generally we are not so lucky



Classification model setup

- **Deep neural network (DNN) was setup using open-source software**
- Jupyter notebook running python using public libraries
- TensorFlow (now including Keras) was developed by Google Brain in 2015 with first stable version in 2017
- Consists of very efficient differentiable programming functions and matrix solvers that are well suited to building neural networks
- Keras layer provides a good starting point for building most networks
- GPU support means that training takes a few minutes on a laptop

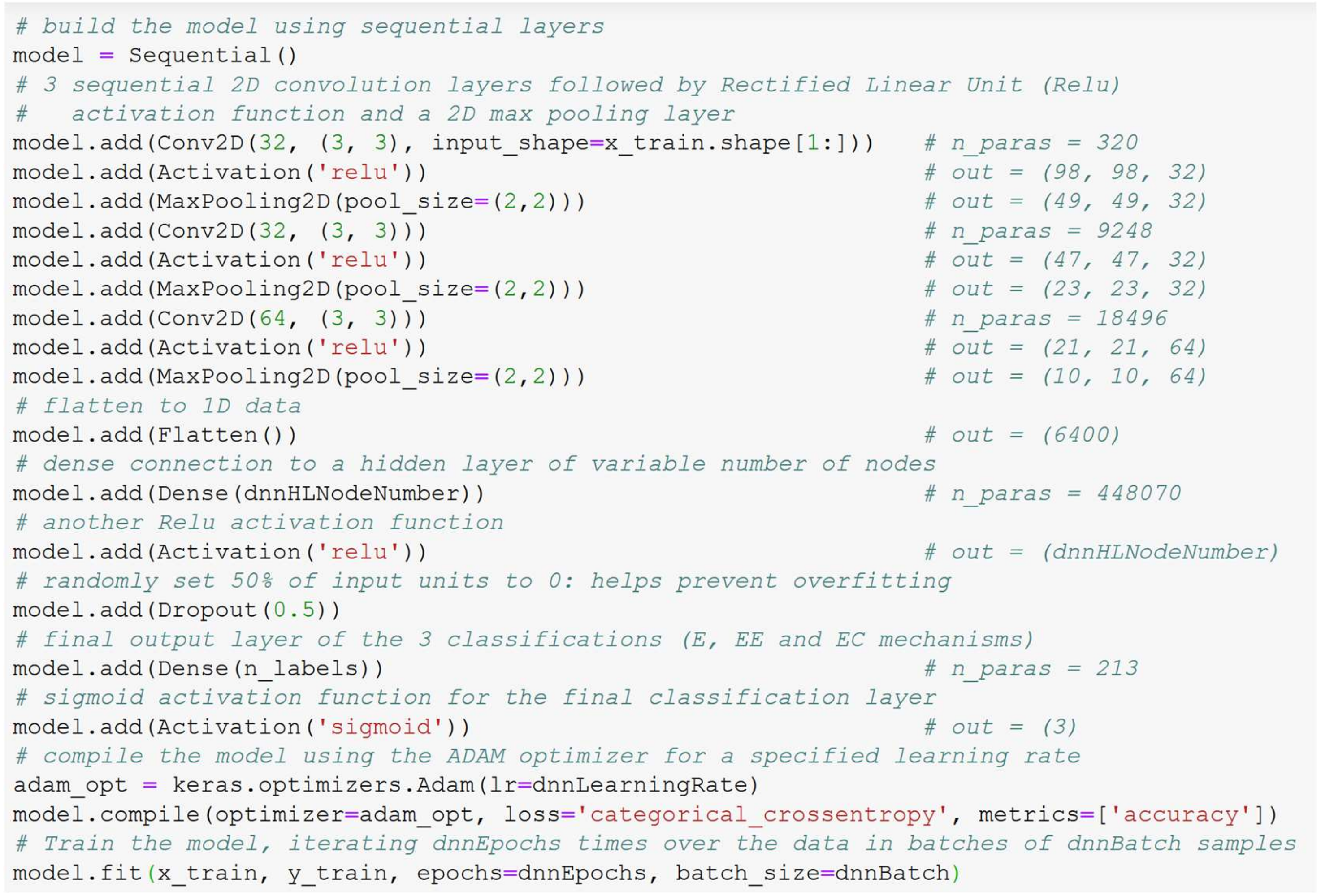
DNN architecture



Hyper parameter optimization

- Randomly split the data into training (10000) and test (5000) images
- Vary the parameters for training the DNN, e.g.
  - Number of nodes in the hidden layer
  - Learning rate, batch size and epoch size
  - Amount of training data used: more is typically better, but good to see what the least amount of data that is required for an accurate model
- Best model is the one with the best accuracy
  - Accuracy defined as the fraction of correct predictions on the test data
  - For other use cases compute time, storage and/or memory could be included
  - Ideally models are robust to inputs not in the original training data
  - Lower number of parameters often makes a model more robust

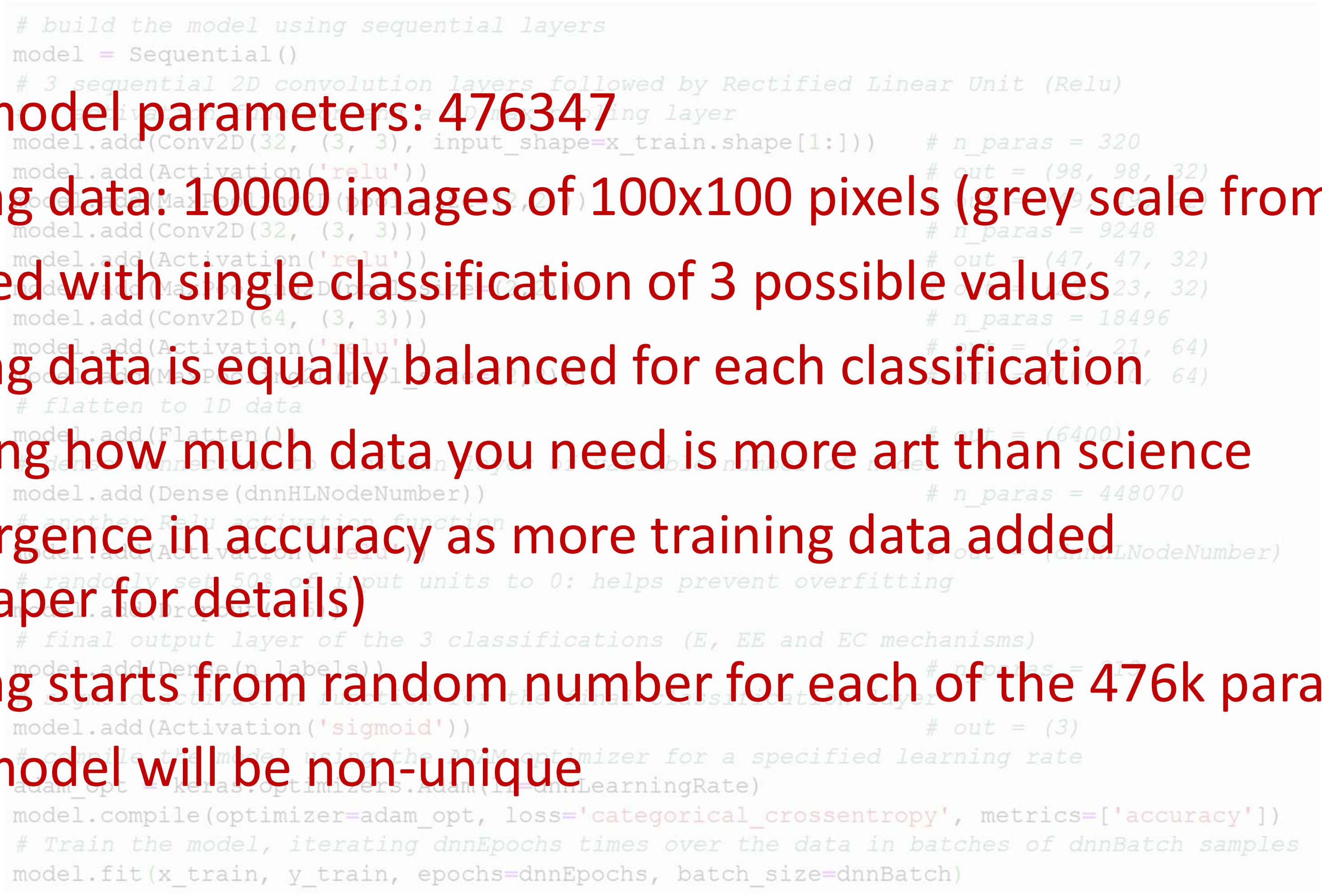
DNN architecture: code



```python
# build the model using sequential layers
model = Sequential()
# 3 sequential 2D convolution layers followed by Rectified Linear Unit (Relu)
#   activation function and a 2D max pooling layer
model.add(Conv2D(32, (3, 3), input_shape=x_train.shape[1:]))    # n_paras = 320
model.add(Activation('relu'))                                   # out = (98, 98, 32)
model.add(MaxPooling2D(pool_size=(2,2)))                        # out = (49, 49, 32)
model.add(Conv2D(32, (3, 3)))                                   # n_paras = 9248
model.add(Activation('relu'))                                   # out = (47, 47, 32)
model.add(MaxPooling2D(pool_size=(2,2)))                        # out = (23, 23, 32)
model.add(Conv2D(64, (3, 3)))                                   # n_paras = 18496
model.add(Activation('relu'))                                   # out = (21, 21, 64)
model.add(MaxPooling2D(pool_size=(2,2)))                        # out = (10, 10, 64)
# flatten to 1D data
model.add(Flatten())                                            # out = (6400)
# dense connection to a hidden layer of variable number of nodes
model.add(Dense(dnnHLNodeNumber))                               # n_paras = 448070
# another Relu activation function
model.add(Activation('relu'))                                   # out = (dnnHLNodeNumber)
# randomly set 50% of input units to 0: helps prevent overfitting
model.add(Dropout(0.5))
# final output layer of the 3 classifications (E, EE and EC mechanisms)
model.add(Dense(n_labels))                                      # n_paras = 213
# sigmoid activation function for the final classification layer
model.add(Activation('sigmoid'))                                # out = (3)
# compile the model using the ADAM optimizer for a specified learning rate
adam_opt = keras.optimizers.Adam(lr=dnnLearningRate)
model.compile(optimizer=adam_opt, loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model, iterating dnnEpochs times over the data in batches of dnnBatch samples
model.fit(x_train, y_train, epochs=dnnEpochs, batch_size=dnnBatch)
```

DNN architecture: parameters

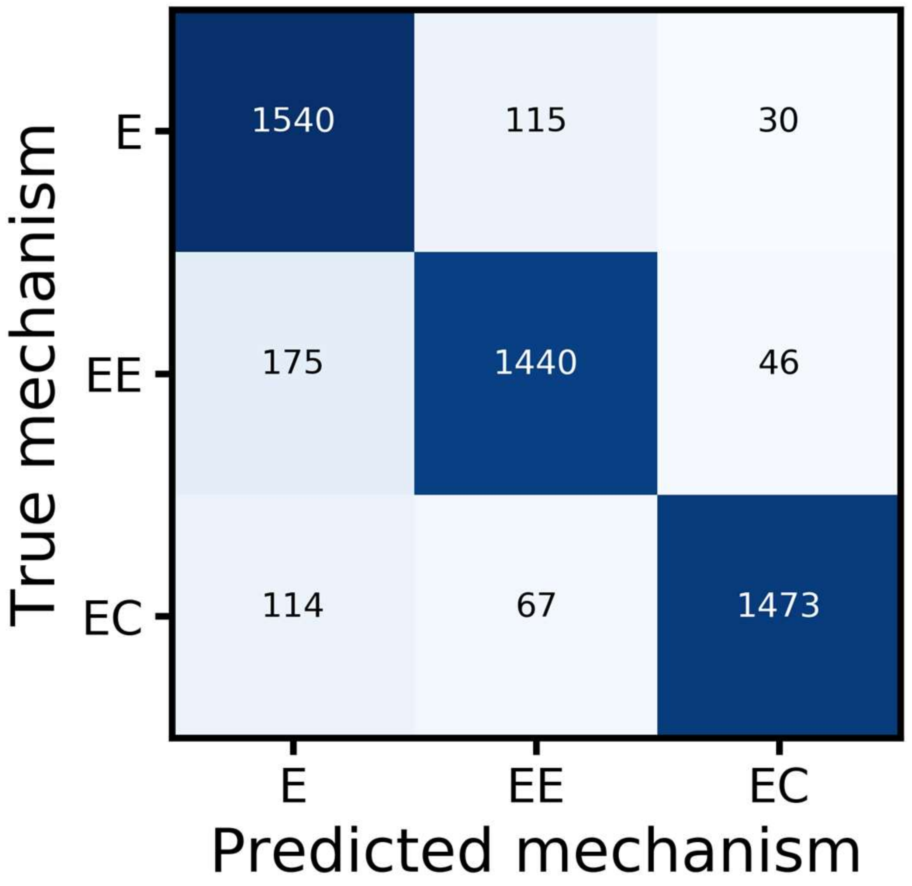
- DNN model parameters: 476347
- Training data: 10000 images of 100x100 pixels (grey scale from 0 to 1)
- Labelled with single classification of 3 possible values
- Training data is equally balanced for each classification
- Knowing how much data you need is more art than science
- Convergence in accuracy as more training data added (see paper for details)
- Training starts from random number for each of the 476k parameters
- Final model will be non-unique



```
# build the model using sequential layers
model = Sequential()
# 3 sequential 2D convolution layers followed by Rectified Linear Unit (Relu)
# [illegible] 2D [illegible]ing layer
model.add(Conv2D(32, (3, 3), input_shape=x_train.shape[1:]))    # n_paras = 320
model.add(Activation('relu'))                                     # out = (98, 98, 32)
model.add(MaxPooling2D(pool_size=(2, 2)))                         # out = (49, 49, 32)
model.add(Conv2D(32, (3, 3)))                                     # n_paras = 9248
model.add(Activation('relu'))                                     # out = (47, 47, 32)
model.add(MaxPooling2D(pool_size=(2, 2)))                         # out = (23, 23, 32)
model.add(Conv2D(64, (3, 3)))                                     # n_paras = 18496
model.add(Activation('relu'))                                     # out = (21, 21, 64)
model.add(MaxPooling2D(pool_size=(2, 2)))                         # out = (10, 10, 64)
# flatten to 1D data
model.add(Flatten())                                              # out = (6400)
# [illegible] connected [illegible] layer [illegible] number of nodes
model.add(Dense(dnnHLNodeNumber))                                 # n_paras = 448070
# another Relu activation function
model.add(Activation('relu'))                                     # out = (dnnHLNodeNumber)
# randomly set 50% of input units to 0: helps prevent overfitting
model.add(Dropout(0.5))
# final output layer of the 3 classifications (E, EE and EC mechanisms)
model.add(Dense(n_labels))                                        # n_paras = [illegible]
# sigmoid activation function for the final classification layer
model.add(Activation('sigmoid'))                                  # out = (3)
# compile the model using the ADAM optimizer for a specified learning rate
adam_opt = keras.optimizers.Adam(lr=dnnLearningRate)
model.compile(optimizer=adam_opt, loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model, iterating dnnEpochs times over the data in batches of dnnBatch samples
model.fit(x_train, y_train, epochs=dnnEpochs, batch_size=dnnBatch)
```

Accuracy: confusion matrix

- Here there are a relatively small number of classes
- Useful to see the confusion matrix
- Note there is some confusion where EE or EC is predicted to be E
- Can then follow up on mis-classifications in the model



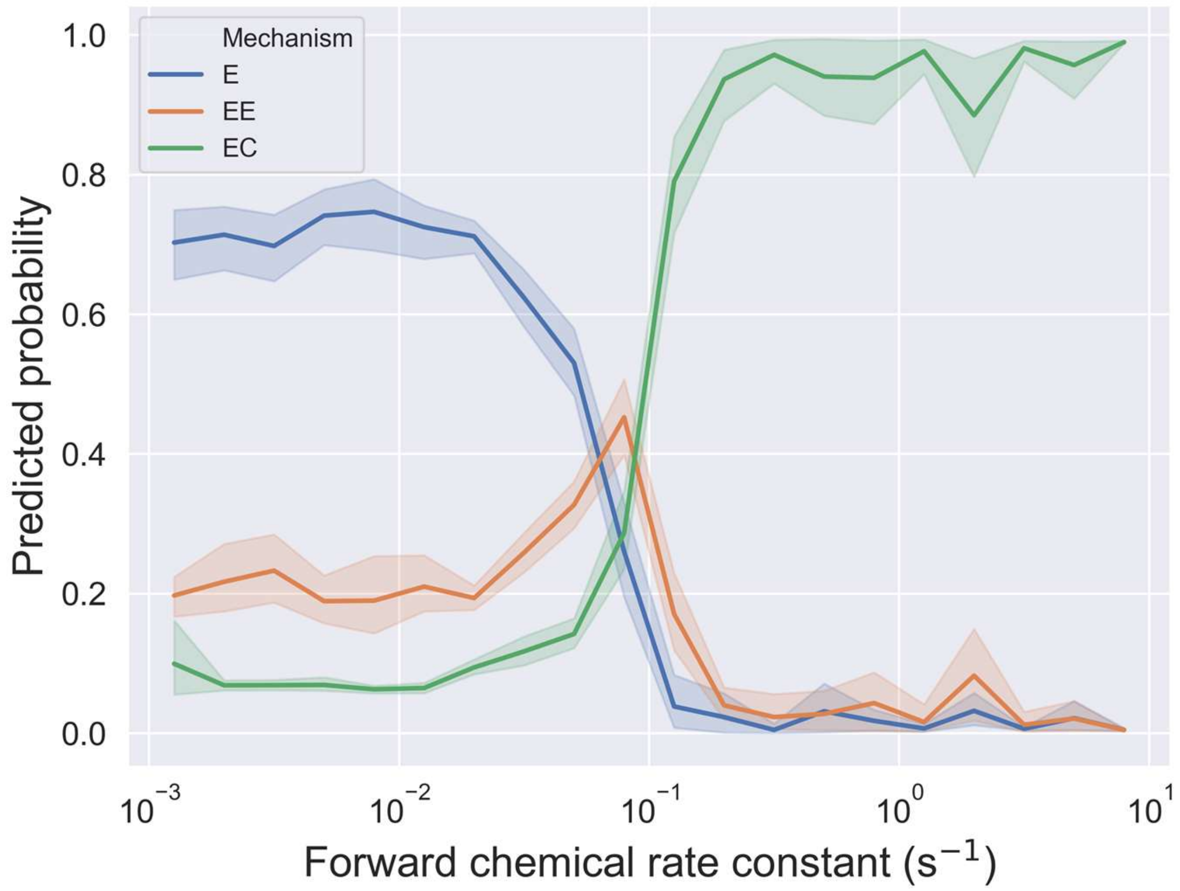
	E	EE	EC
E	1540	115	30
EE	175	1440	46
EC	114	67	1473

Need for domain knowledge

- A model will ALWAYS give an answer, but it will never know if it was right. It lacks understanding
- At it’s best a ML model is a black box that becomes more accurate with more data while not suffering from bias
- An expert human can always wonder “that doesn’t seem right…”
- How can we investigate further?
    - Sanity checks on edge cases (looks for bias)
    - Stress tests to find where it will break (check robustness)

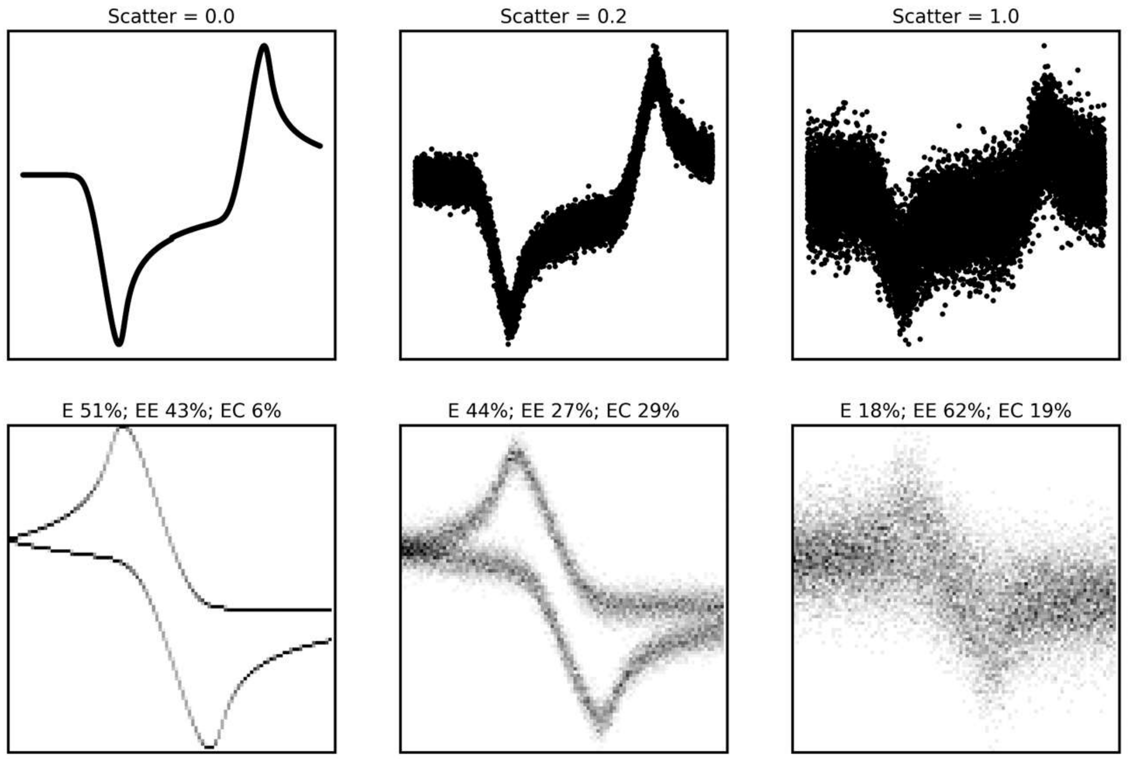
“Sanity checks”

- **Examine** misclassifications (e.g. EC to E)
- Here there is a physical reason in that a low chemical rate constant reduces the EC mechanism (A + e = B; B = C) to a single electron transfer reaction (A + e = B)



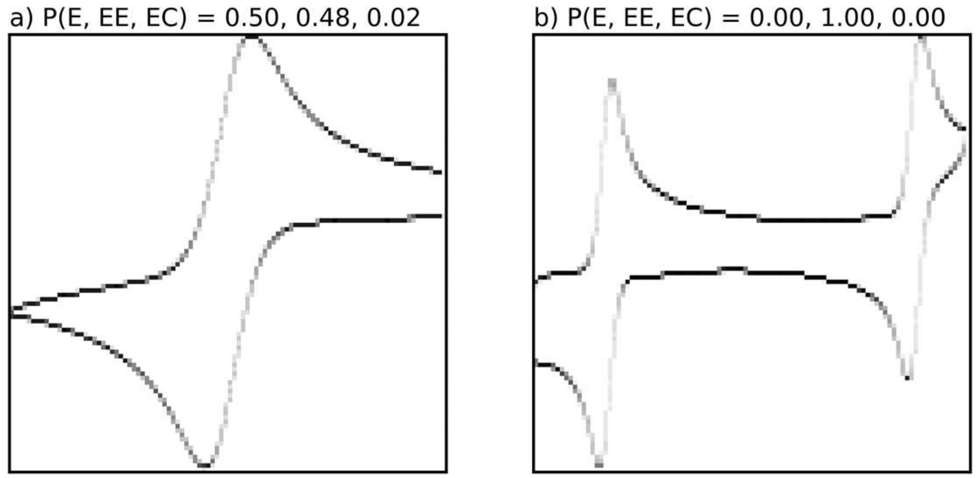
“Stress test”

- Every model is invalid somewhere
- **Find where the model breaks**
- Is that within tolerable bounds?
- Fine here as the noise must be more than 25% the magnitude of the signal to mis-classify



Productionize model

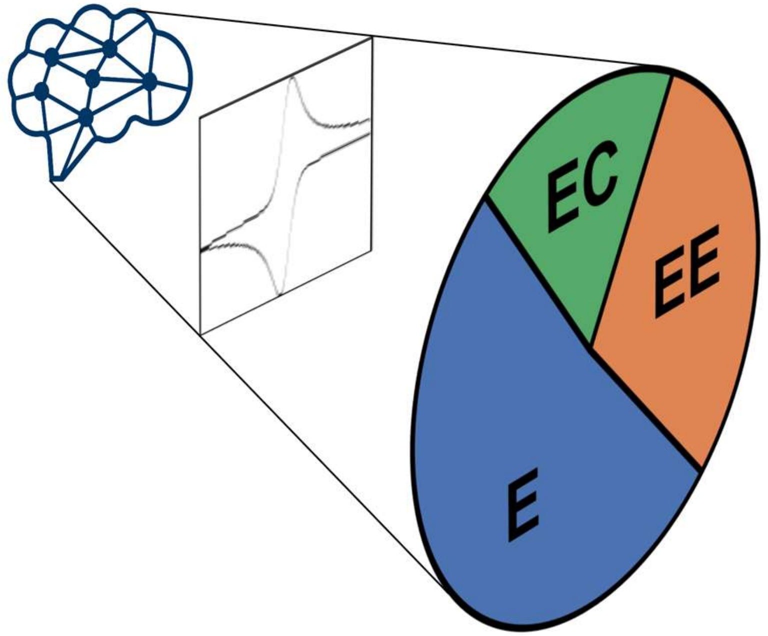
- **Aim of the model is to classify new experimental data without human input**
- New data is converted to ML friendly image (right)
- DNN classifies image and gives the probabilities of each
- Parameters for the most likely mechanism (e.g. rate constants) can now be automatically optimized
- ML models often get stuck as “proof of concepts” – must be productionized for it to be useful



Classification probabilities determined by DNN using experimental cyclic voltammetric data sets derived from (a) E ($Fc^{0/+}$) and (b) EE ($[SVW_{11}O_{40}]^{3-/4-}$ and $[SVW_{11}O_{40}]^{4-/5-}$) mechanisms.

Summary of a DNN results

- New experimental data automatically classified (within 5 ms) into 3 common electrode reaction types
- Next version will be developed by Luke Gundry (School of Chemistry, Monash) to use FTAC data with more reaction types
- First step of an automated data flow consisting of classification, optimization (with group at Oxford) and a "next experiment" recommendation engine
- **Online version will allow massive time savings for research and industry chemists from around the world**



Using ML tools to augment role of a scientist

- ML models have a black box problem
- Any tool must be as robust and as clear of bias as possible
  - sanity and stress tests needed to know their limitations
- Aim is to use ML tools to save researcher’s time
- ML will not replace researchers as deep understanding will always be needed to interpret the results

- Note that the same is true in industry