Machine Learning

Implementing MLPs III
Regression Problems

CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap

- What is Machine Learning
- How is it useful for Civil Engineers
- Overview of Machine Learning Methods
- Linear Regression
  - Bivariate
  - Regression interpretation
  - Multivariate
- Logistic Regression
  - Maximum likelihood estimation
  - Regularization (introduction)
- Naïve Bayesian Classifier
  - What is it
  - What makes it naïve
  - Bayes theorem
  - Prior, likelihood and posterior
- K-Nearest Neighbor
  - How does the algorithm work
  - Why is it a lazy learner
  - How to do regression and classification
- Introduction to Decision Trees
  - Fundamentals
  - Information Gain, Entropy and Gini Index
  - ID3 algorithm
  - Classification and Regression Trees (CART)
  - Multi-Adaptive Regression Splines (MARS)

- Ensemble learners
  - Introduction
- Their benefits and drawbacks
- Simple (voting) ensemble learners
- Bagging and Pasting
- Generic bagging classifiers
- Random Forest classifiers
- Boosting Classifier
  - Adaboost
- Unsupervised classification
  - K Means Learning
- Introduction to Artificial Neural Networks
- Perceptron
- Overview of MLPs
- MLP – Basic Implementation
  - Keras + Tensorflow
  - Binary Classifier
- MLP – MultiClass Classification
- Optimization methods
  - SGD
  - Adam
- Regularization

Python – Introduction
Python – Functions
Python - Pandas
Python – np, scipy, statsmodels
Python – Scikit learn – linear, metrics
Python – Matplotlib, seaborn
Python – Mixed_Naive_Bayes
Python – scikit learn neighbors module
Python – scikit learn ensemble voting
Python – scikit learn bagging classifier
Python – scikit learn RandomForestClassifer
Python – scikit learn AdaBoostClassifier
Python – scikit learn Kmeans
Python – scikit learn perceptron

R – Classification and Regression Trees using rpart
R – Drawing trees using rpart.plot
R - Multiadaptive Regression Splines (MARS) using Earth Algorithm

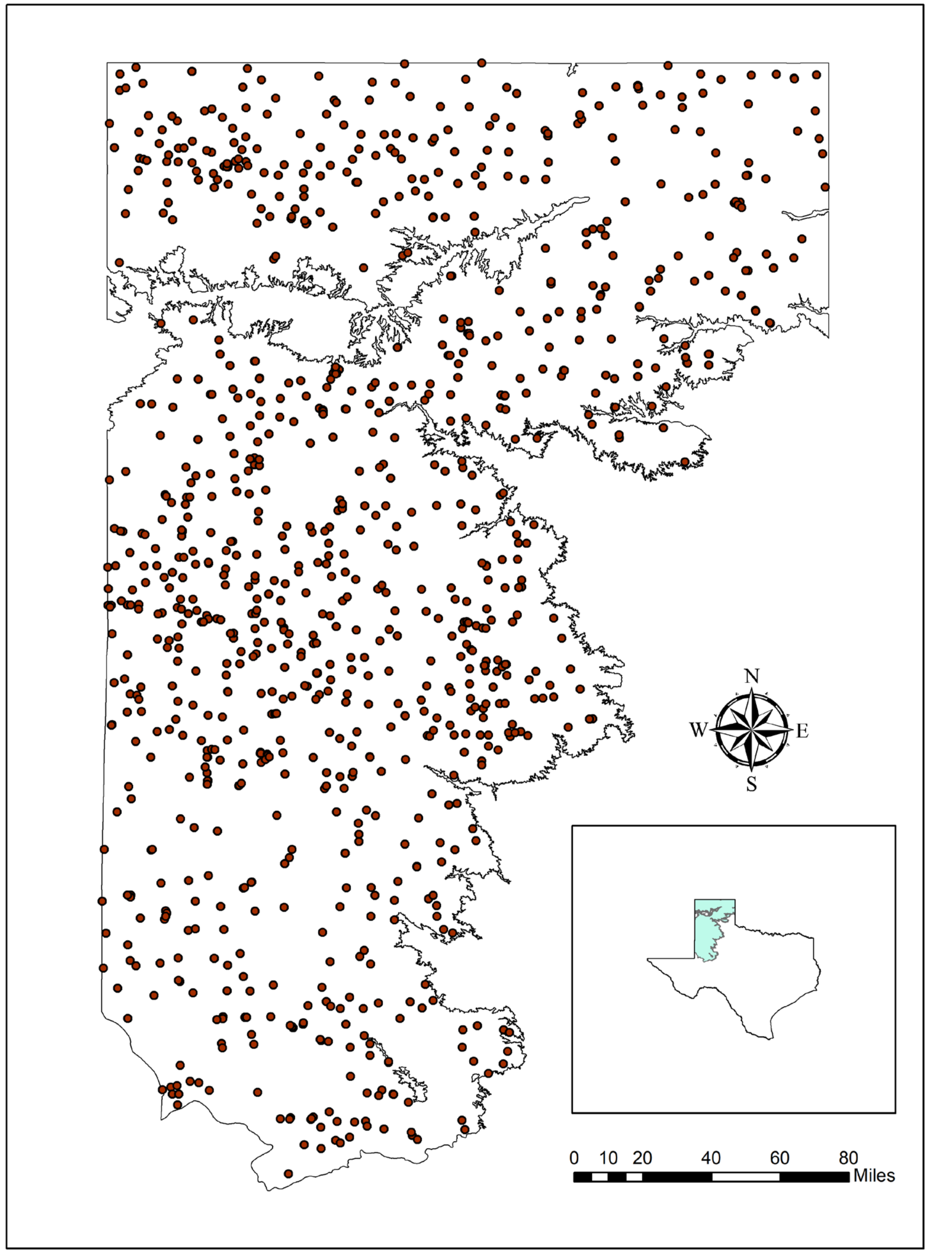
Artificial Neural Networks

Goals

- Use ANN for Regression Problems
- Using ML to refine datasets
- Cross-Validation
- Dropout
- Simple Deep Networks

Illustrative Examples

Regression Problem



Illustrative Example

- Prediction of NO3 (nitrate) Concentrations in the Ogallala Aquifer
- Nitrate is a serious issue in the Ogallala Aquifer
  - Several health effects
    - Blue Baby Syndrome
  - https://www.kcbd.com/story/30601070/kcbd-investigates-whats-in-your-water/
- Nitrate comes from a variety of sources
  - Agriculture
  - Onsite Wastewater
  - Decay of natural organic matter
  - Confined Animal Feed Operations (CAFO)

Per Regulations Nitrate in Drinking Water Cannot exceed 10 mg/L (NO3-N)

Data Compilation

- A comprehensive database was compiled to study nitrate contamination
  - Nitrate concentrations averaged over a period of 1995 – 2018 at over 900 wells
  - Other Groundwater data
    - Major ions, TDS, EC, pH
  - Hydrogeological information
    - Well depth, Depth to Water Table (2018)
  - Soil Properties in the vicinity of the well
    - Soil Organic matter
    - Grain size distribution
    - Cation Exchange Capacity (CEC)
  - Hydrology
    - Annual Precipitation

The file 'Ogallalacleaned.csv' has the data

The database has 31 parameters and 957 records

Input Selection

- Input selection based on theory
  - Variables selected affect NO3 concentrations
- Identify a subset of important variables
  - You can use a model that provides feature selection
    - Random Forests
    - Gradient Boosting Trees
    - Extra Trees Regression
- Remove any variables that are correlated to each other
  - Multicollinearity

ExtraTrees – Extremely Randomized Trees

Usually faster than Random Forests but similar

A random value is selected instead of trying to find a locally optimal solution at each level

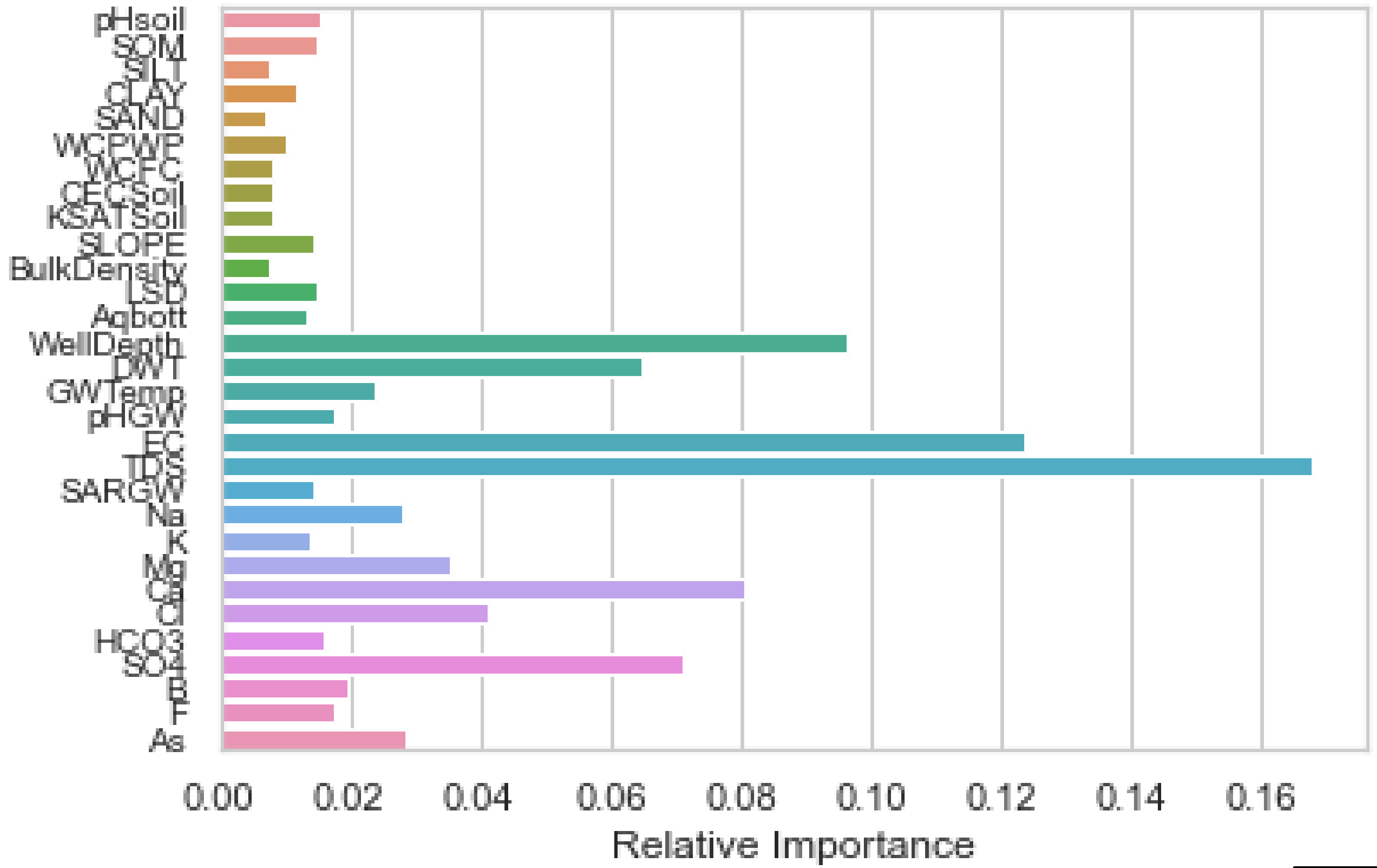
RF and ExTrees give similar performance if the variables are relevant

Python Code

```
import os
import statistics
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics
# Step 2: Change working directory
dir = 'D:\\Dropbox\\000CE5333Machine Learning\\Week9ANNS\\Code'
os.chdir(dir)


# Step 3: Read the dataset
a = pd.read_csv('Ogallalacleaned.csv') # read our dataset with multiclass data

# Extract X and Y (nitrate)
X = a[a.columns[4:34]] #skip WellID,lat,lon, NO3
Y = a['NO3']

# Split into Training and Testing
X_train,X_test, y_train,y_test = train_test_split(X,Y, test_size=0.30,
                                random_state=10)
```

```
# Fit and ExtraTrees Regression Model
clf = ExtraTreesRegressor(n_estimators=500,criterion='mse')
clf = clf.fit(X_train, y_train)
imp = clf.feature_importances_
names = list(X_train.columns) # Get names of variables_ # Obtain feature importance
impa = (names,imp) # Make a tuple
impadf = pd.DataFrame(impa) # Write to a dataframe
sns.set(style="whitegrid")
ax = sns.barplot(x=imp, y=names) # Make a barplot
ax.set(xlabel="Relative Importance")

# Make Prediction and Test Accuracy
stddev_Ytest = statistics.stdev(y_test) # Std. Dev
y_pred = clf.predict(X_test)
maetest = print("MAE:",metrics.mean_absolute_error(y_test, y_pred))
varexp = print("VARE:",metrics.explained_variance_score(y_test,y_pred))
r2_test = print("R2:",metrics.r2_score(y_test,y_pred))
rmsetest = print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

Result



Testing Evaluation

R^2 ~ 0.23

Sq.RMSE = 23.03 mg/L

MAE = 9.8 mg/L

Sd(Y_test) = 26.21 mg/L

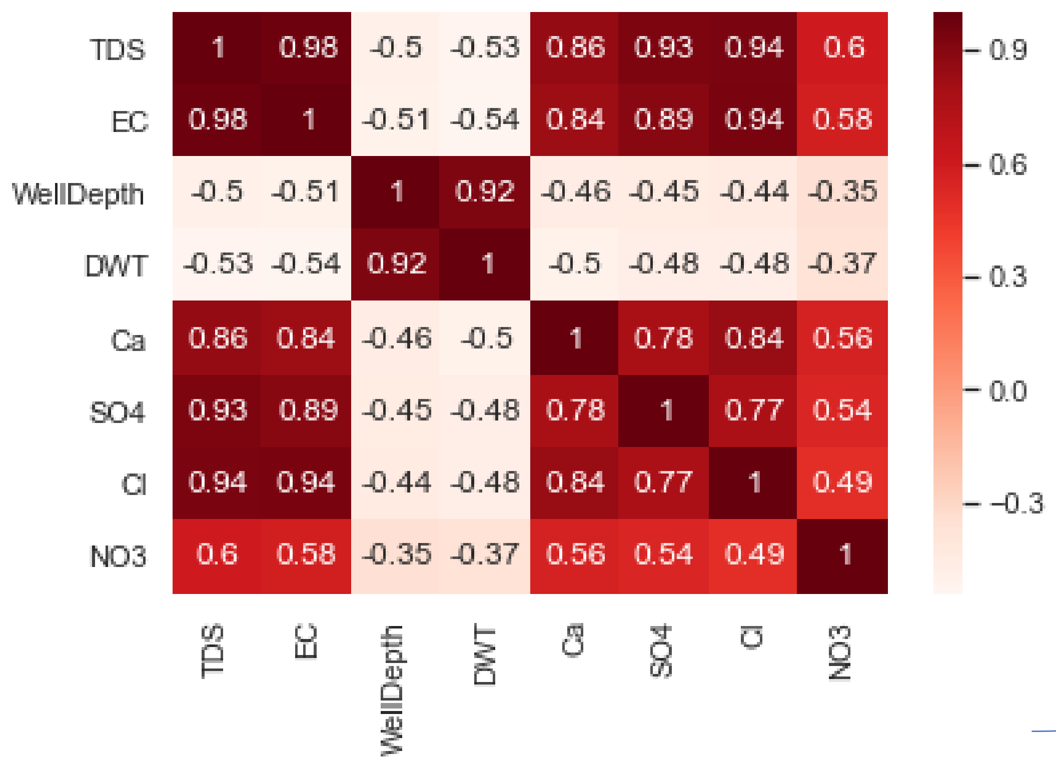
Important Variables

Well Depth
Depth to Water Table
EC
TDS
Ca
SO4
Cl

Remember this is not a unique list - Depends upon the Method Used

Important Variables exhibit correlation

Identifying Correlated Important Variables



	TDS	EC	WellDepth	DWT	Ca	SO4	Cl	NO3
TDS	1	0.98	-0.5	-0.53	0.86	0.93	0.94	0.6
EC	0.98	1	-0.51	-0.54	0.84	0.89	0.94	0.58
WellDepth	-0.5	-0.51	1	0.92	-0.46	-0.45	-0.44	-0.35
DWT	-0.53	-0.54	0.92	1	-0.5	-0.48	-0.48	-0.37
Ca	0.86	0.84	-0.46	-0.5	1	0.78	0.84	0.56
SO4	0.93	0.89	-0.45	-0.48	0.78	1	0.77	0.54
Cl	0.94	0.94	-0.44	-0.48	0.84	0.77	1	0.49
NO3	0.6	0.58	-0.35	-0.37	0.56	0.54	0.49	1

```
# Plot correlation among important Variables
# This is done based on earlier steps
impv = a[['TDS','EC','WellDepth','DWT','Ca','SO4','Cl','NO3']]
cor = impv.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```

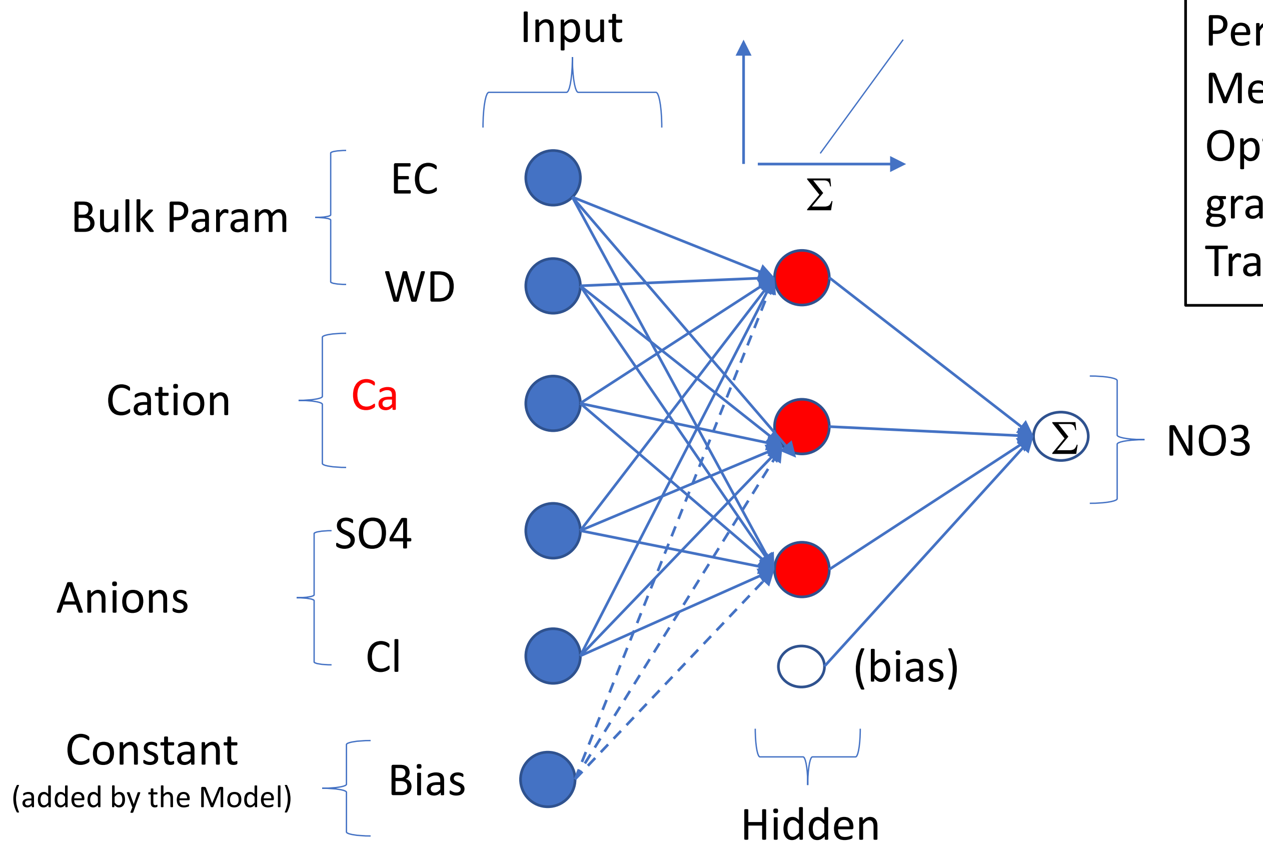
Pick either TDS or EC
(Will pick EC because it is easy to measure)

WellDepth and DWT strongly correlated

Pick Well Depth as it implies weather the aquifer is shallow or deep

Final Inputs – EC, WellDepth, Ca, SO4, Cl

Conceptual Model



Inputs – Same as Before
Hidden Nodes – Requires some trial and error
Activation Functions:

- Hidden layer (s) – RELU
- Output layer – NO3
  - No activation Layer
  - Only summation

Performance Measure (loss) – Cross-Entropy
Metrics – Accuracy
Optimization method – Adam (version of stochastic gradient descent)
Training – 5 epochs; batch size 32; validation split 0.2

Based on Physical Considerations it might be good to add 'Soil Organic Matter' as well

How many Hidden Nodes - Regularization

- Identifying hidden nodes is both science and art
  - Will require some trial-and-error experimentation
- We can add regularization to decay some weights
  - Small weights (decayed out to zero do not add much to the network)
- Regularization essentially adds an additional variable to the loss function (objective function being minimized)
- Keras allows regularization to be added for each layer
- Regularization can be applied on
  - Kernel Regularizer
    - Reduce all weights including bias terms
  - Activity Regularizer – Regularizes layer output
    - used for optimizing hidden nodes
  - Bias terms

Original Loss Function

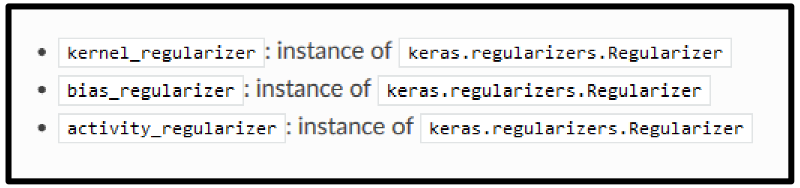
$$L(x, y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

where $h_\theta x_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$

$$L(x, y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 \boxed{+ \lambda \sum_{i=1}^{n} \theta_i^2}$$

L2 Regularizer

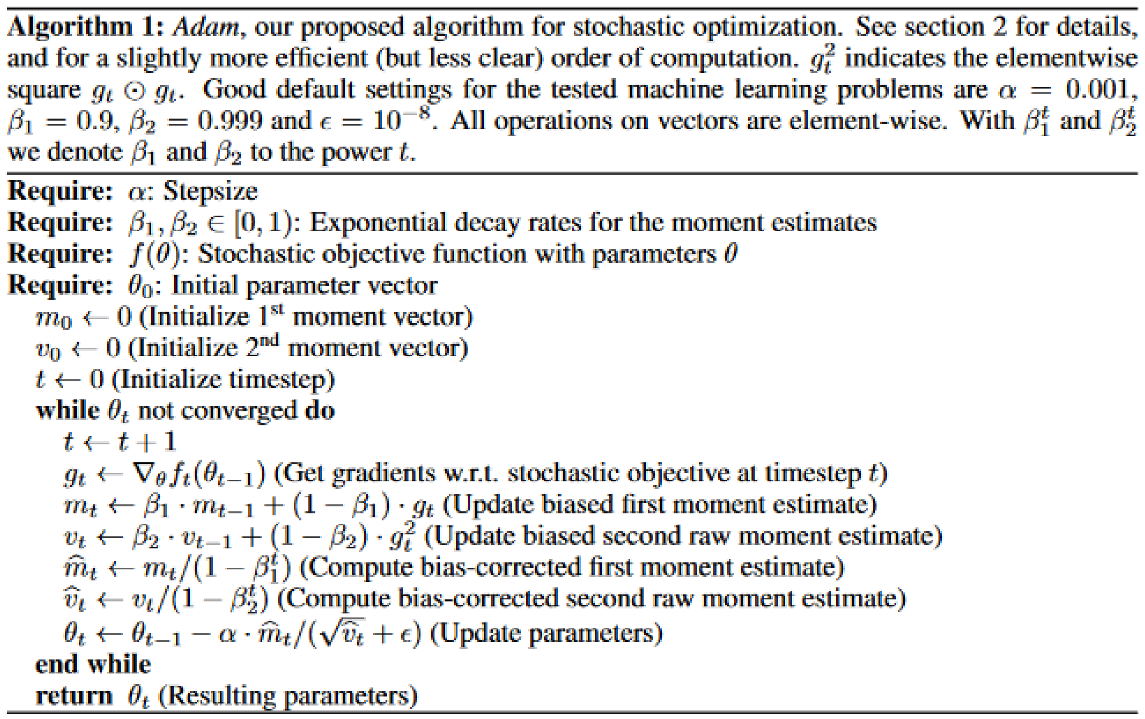
Here λ is a weighting factor (typically a small number 0.001)



- `kernel_regularizer` : instance of `keras.regularizers.Regularizer`
- `bias_regularizer` : instance of `keras.regularizers.Regularizer`
- `activity_regularizer` : instance of `keras.regularizers.Regularizer`

Optimization Method - Adam

- Adam is a variant of stochastic gradient descent
  - Closely related to other variants
    - RMSProp
    - AdaMax
- Has three (four) hyperparameters
  - Initial step size (α)
  - Exponential Decay rates (β1,β2) for momentum
  - Error tolerance (ε)



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

ADAM Method

- Each parameter can have its own learning rate
- Gradients from previous steps are used to update weights (momentum)
  - In SGD only weights from previous steps are used
  - Use of gradients from previous steps tends to dampen oscillations
  - Older gradients are weighted less than recent gradients
- Squares of gradients are also used
  - Adds second moment (variance)
- The weights are second-order unbiased
  - The expected mean and (uncentered) variance of the weight must equal the unbiased mean and variance
- The initialization of weights (as zeros) adds bias
  - This needs to be corrected (which the algorithm does)
- Exponential smoothing is used to average previous and current gradient terms
- Initial learning rate is adapted over iterations
- Hyper-parameters for exponential decay are seldom changed from the recommendations
  - β1 = 0.9 and β2 = 0.999

$For\ each\ Parameter\ w^j$

$(j\ subscript\ dropped\ for\ clarity)$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$

$g_t : Gradient\ at\ time\ t\ along\ \omega^j$

$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$

$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$

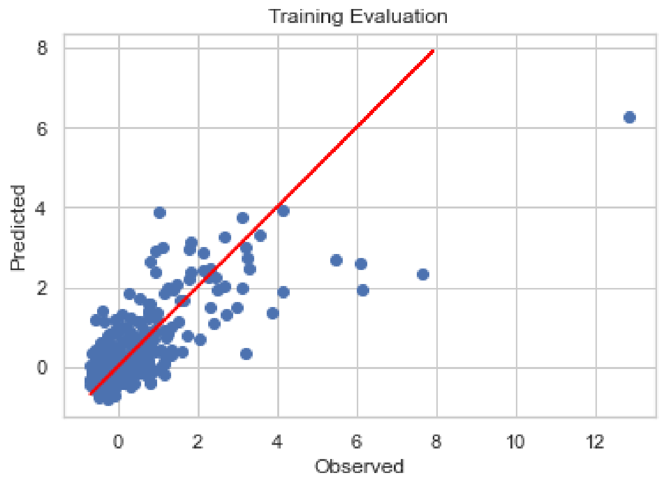
$\beta_1, \beta_2 : Hyperparameters$

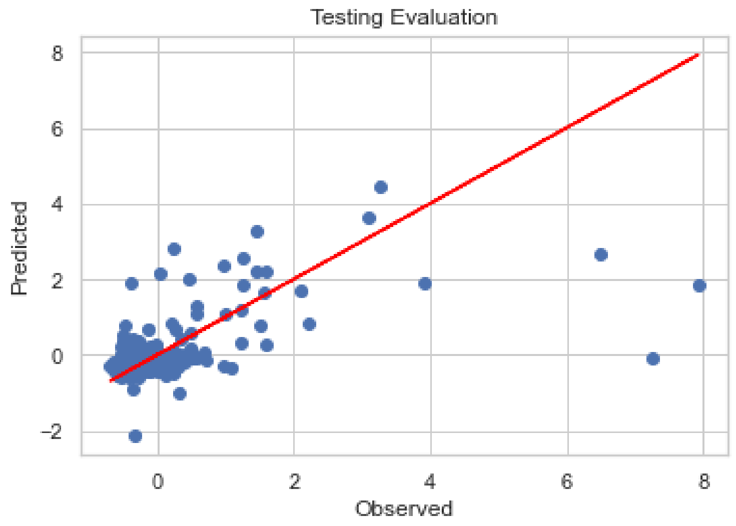
ADAM is called Adaptive Moment method

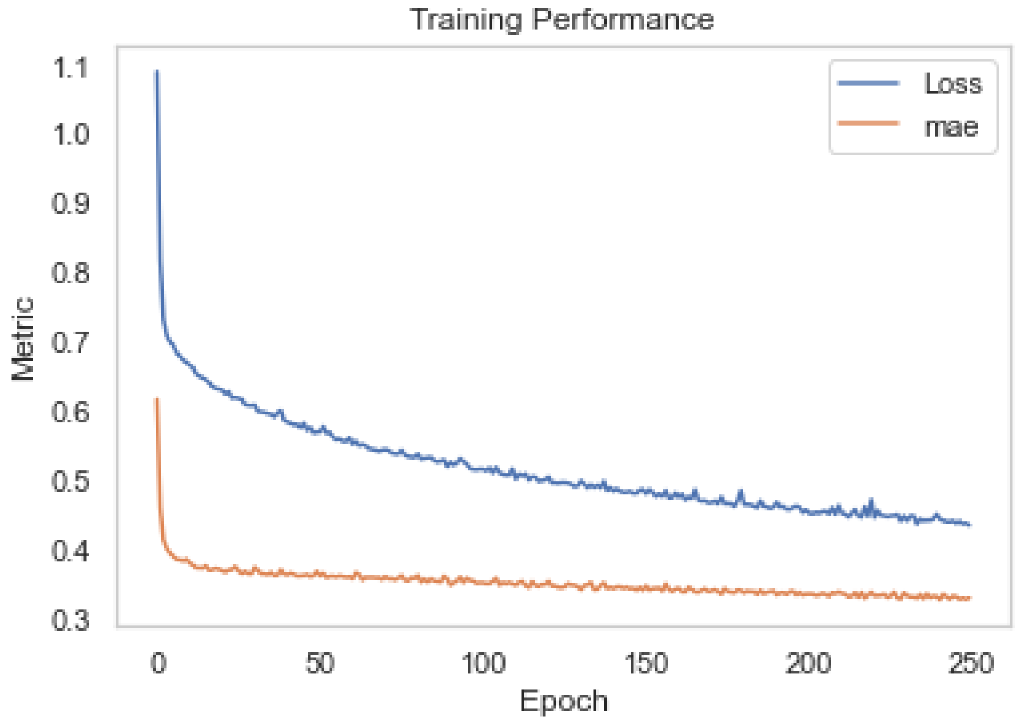
Adaptive because the learning rate (step size) of weights change

Moments because first and second moments are used

Results



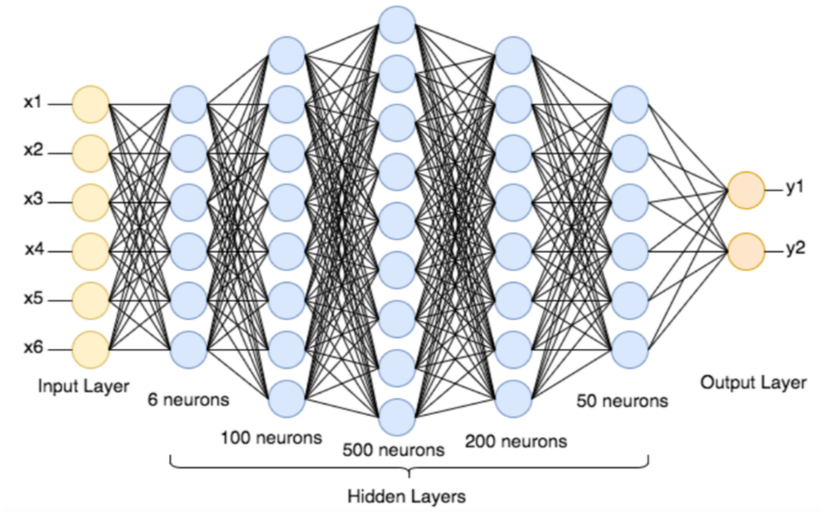




Perhaps there is more training to be done
The model result is being affected by some outliers
There does not appear to be much overfitting

Adding More Hidden Layers

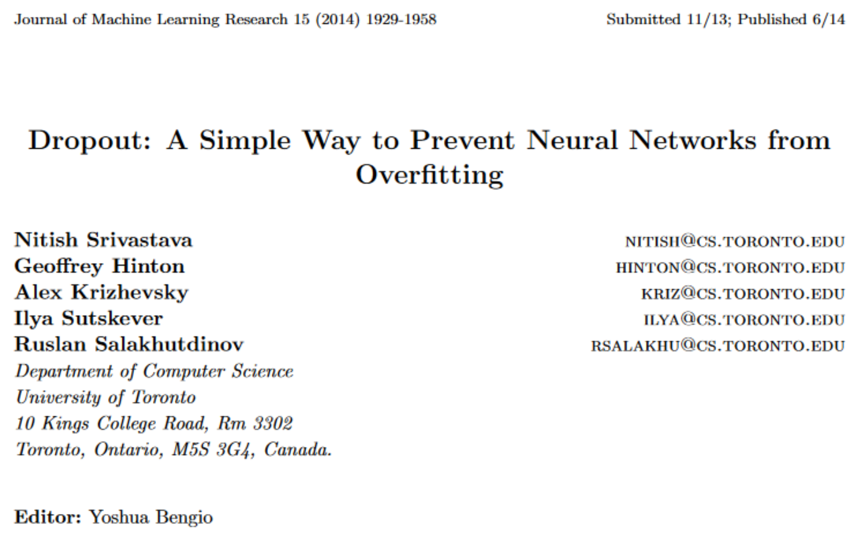
- ANNs with more than one Hidden Layers are referred to as Deep Neural Networks
  - Have a few layers may not be very deep network
  - Some ANNs have hundreds of Hidden Layers
    - Makes optimization difficult
- Adding an additional hidden layer in a sequential model (MLP) is easy in Keras
  - Define a dense layer(s) below the first hidden layer
  - Need to define the number of nodes
  - Define the type of activation function
  - Define if any regularization is to be performed



Dropout Regularization

- Dropout is a form of regularization that can be applied
  - Applied to the input layer
  - Usually applied to one or more hidden layers
- Dropout is specified using a dropout fraction
- Randomly selected neurons are ignored during training
  - Dropout is randomly applied to each epoch

```
model = Sequential()
model.add(Dropout((0.2), input_shape=(6,))) # 20% of inputs are dropped
model.add(Dense(30, input_dim=6, activation='relu'))  # Hidden layer)) # Add
regularizer
model.add(Dropout(0.3))  # Dropout 30% of hidden node during each epoch
model.add(Dense(10,activation='relu',kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1)) # Output Layer
```



Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Dropout Regularization

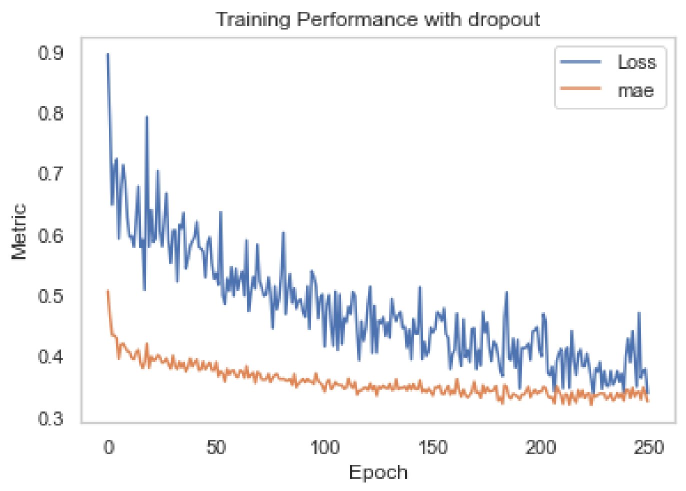
- When nodes are dropped other variables have to pick up the slack
  - The weights of the dropout nodes are readjusted to existing nodes
- Dropout breaks the structure of the ANNs
- In Networks with fixed structures – The nodes within a layer develop reliance on neighboring nodes
  - This is removed by dropout
- The benefits of dropout are only proven experimentally
  - Theory is still not fully developed
- Dropout may not provide intended results
  - Start with small dropout (~ 20% - 30%)
  - Try to do both for input and hidden layers
  - Always have a base model to check results

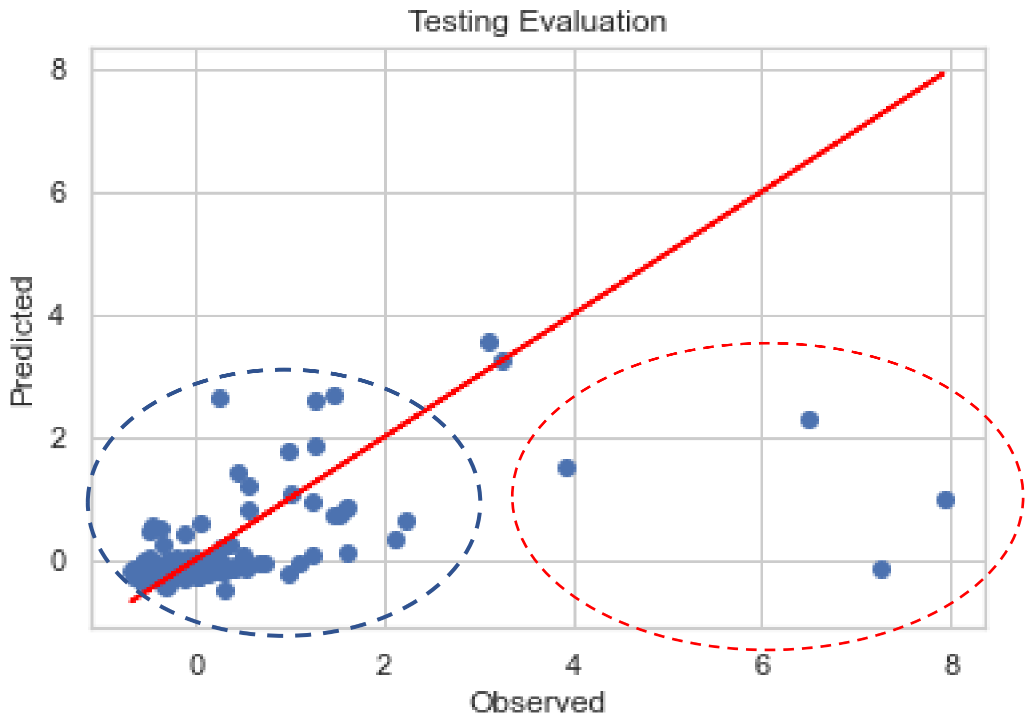
Dropout can be regarded as an ensemble learning

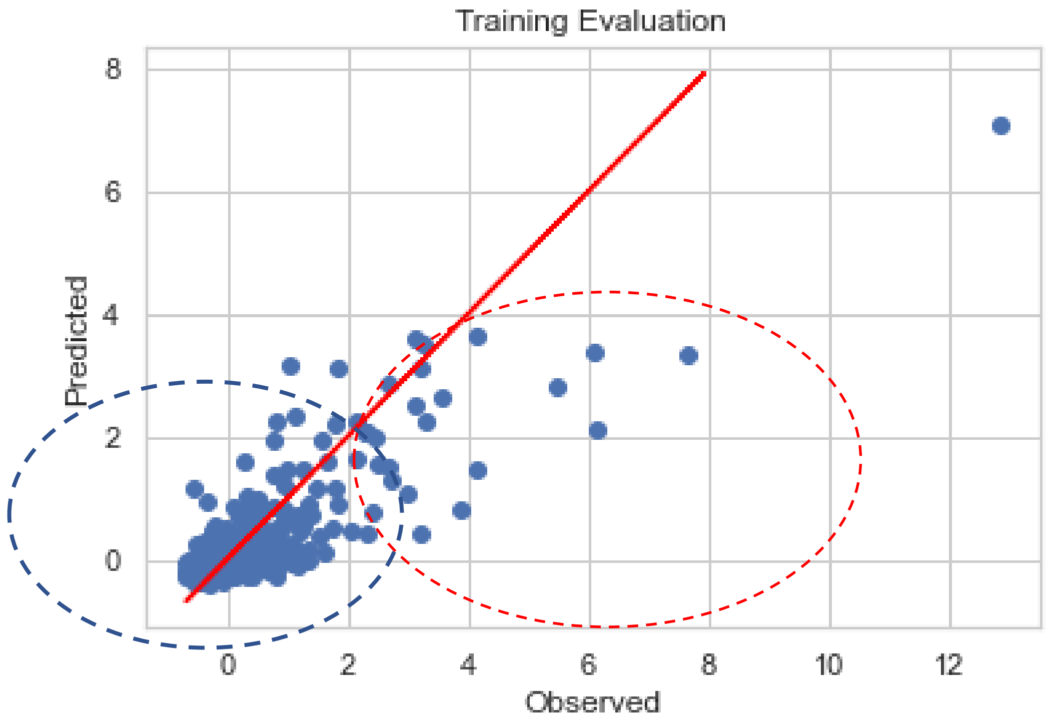
$$\boxed{E_R = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_i w_i I_i\right)^2 + \sum_{i=1}^{n} p_i(1-p_i) w_i^2 I_i^2}$$

Model 2

- Add 2 Hidden Layers
  - 30 neurons
  - 10 neurons
- Use 30% dropout in the first hidden layer







You Should Know

- What are deep Neural Networks
- ADAM Method
- What is dropout regularization
- How to implement regression problems in MLP ANNs
    - Treatment of the output layer