UCLAAnderson
SCHOOL of MANAGEMENT

MGMTMFE 431:

Data Analytics and Machine Learning

Topic 9:
Neural Networks and Deep Learning

Spring 2025

Professor Lars A. Lochstoer

UCLAAnderson
SCHOOL of MANAGEMENT

Neural Networks

UCLAAnderson
SCHOOL of MANAGEMENT

b. Neural Networks

"Deep learning" a current buzz-word in Machine Learning

- Based on multiple layers of neural networks
- Speech, image processing, and (more slowly) finance

Multi-layer neural nets have shown good performance

- Used in language and image recognition (classification)
- Very 'black box'-y. Typically hard to understand exactly why something works
- Lots of training data could train very complicated, nonlinear models. E.g., language
- In this class, we just cover feedforward networks (as opposed to recurrent neural networks where neurons could also feed back up the layers)

UCLAAnderson
SCHOOL of MANAGEMENT

b. Real vs. Artificial Neuron

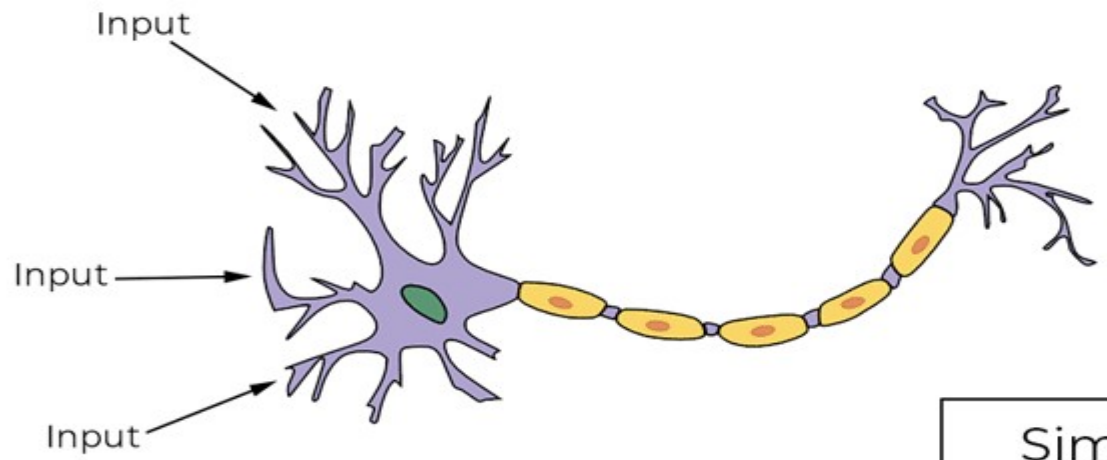
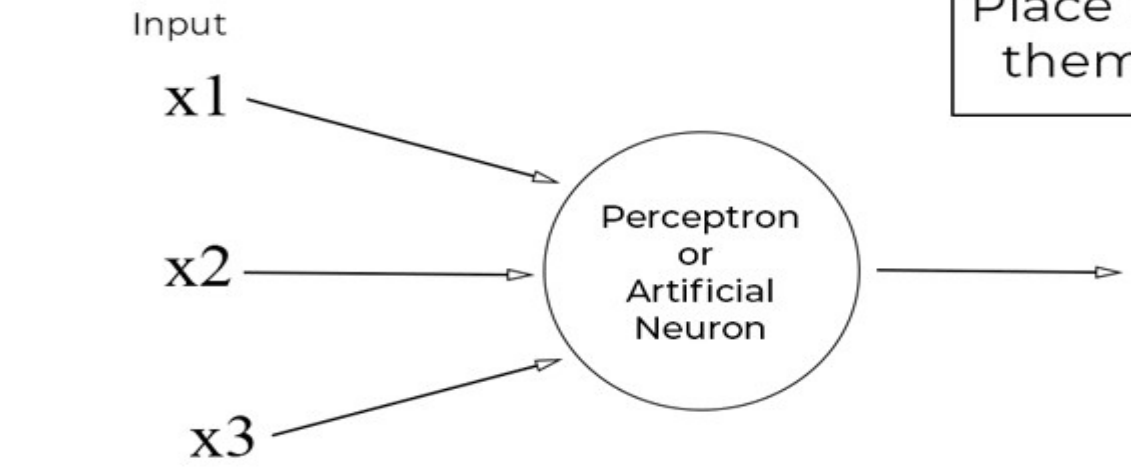


Fig: Biological Neuron

Similarity 1: Input Takes Place in both of them on LHS



UCLAAnderson
SCHOOL of MANAGEMENT

b. Real vs. Artificial Neuron

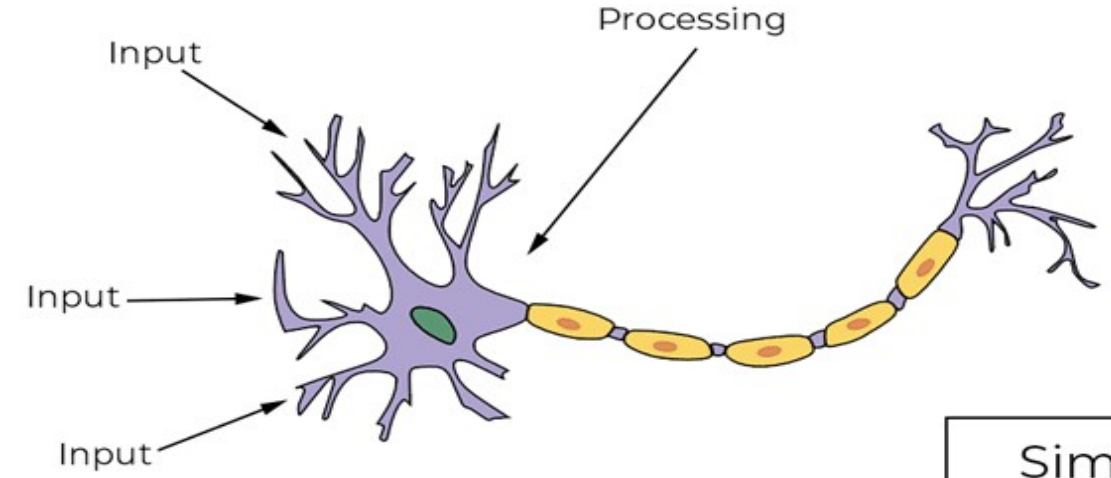
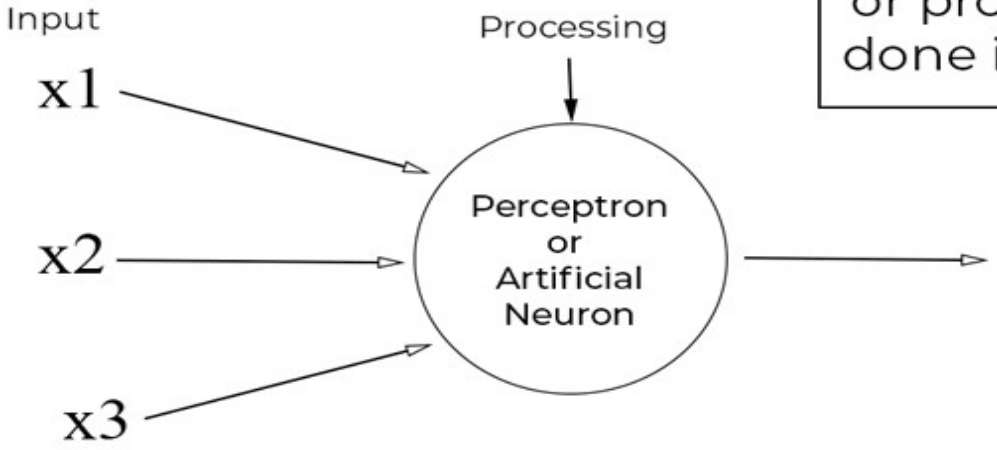


Fig: Biological Neuron

Similarity 2: Logic is applied or procesing is done in middle



UCLAAnderson
SCHOOL of MANAGEMENT

b. Real vs. Artificial Neuron

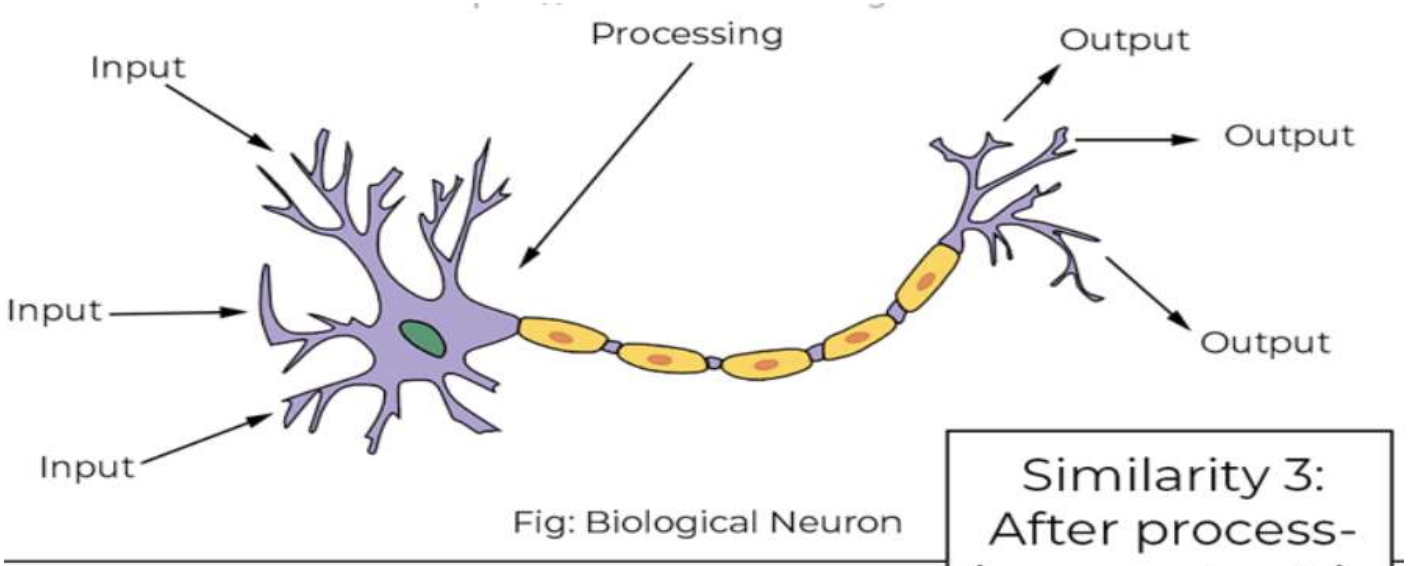
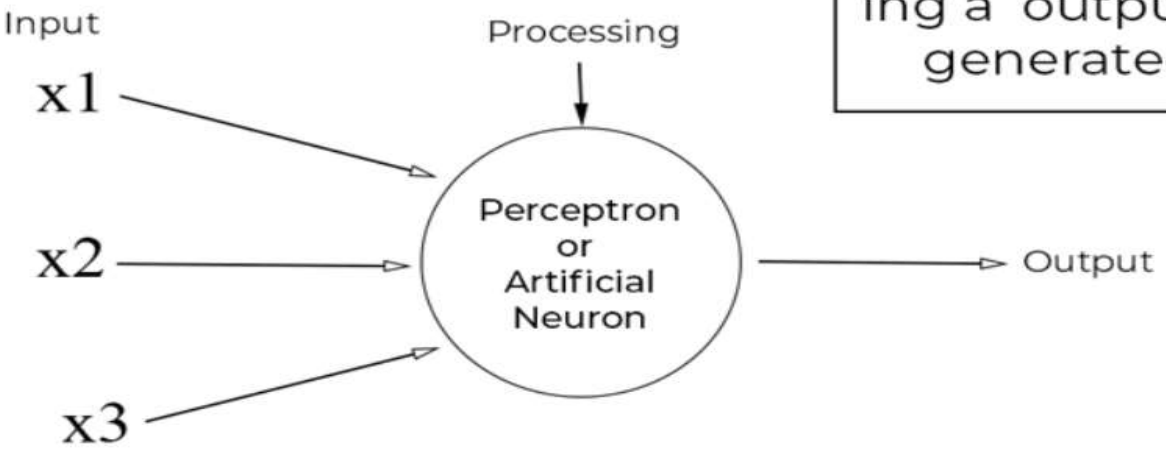


Fig: Biological Neuron

Similarity 3: After process-ing a output is generated



UCLAAnderson
SCHOOL of MANAGEMENT

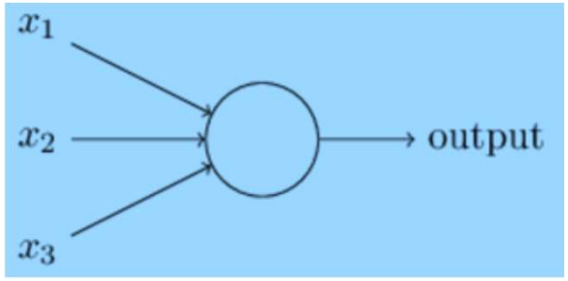
b. Perceptron ("neuron" or nnet unit)

Simplest case: Binary Step

Input: binary data (x)

Output: binary 'decision'

Weights: *w*



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Typical output equation:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

UCLAAnderson
SCHOOL of MANAGEMENT

b. Activation functions

More generally, the perceptrons can have various ***activation functions***

Examples include:

- Binary Step
- Linear (Identity)
- Sigmoid (Logistic, soft-step)
- TanH
- Rectified Linear Unit (ReLU)

…and many more

UCLAAnderson
SCHOOL of MANAGEMENT

b. Activation function: Linear

The neuron output does not have to be binary, but could be ***continuous***

- A baseline example is a linear model

$$output = b + \sum_{j=1}^{J} w_j x_j$$

- ***Thus, with one neuron, this neural net is the same as a linear regression!***
  - But, since we have a linear model, adding more neurons that feeds back or forward can in the end be written as one linear regression
  - Thus, for continuous output, this isn't a very useful case
  - Said differently, the neurons can't "make decisions" on their own

UCLAAnderson
SCHOOL of MANAGEMENT

b. Activation function: Sigmoid

Want activation function (output function) that takes continuous inputs and has small change in prediction for output

- A common choice is the ***logistic function***
- Bounded between zero and one
- Also called Sigmoid or Soft-Step function,

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

- ***Thus, with one neuron, this neural net is the same as a logistic regression!***
  - Note that adding neurons now does have an effect as Sigmoid function is not linear

UCLAAnderson
SCHOOL of MANAGEMENT

b. Activation function: TanH

A close cousin to the Sigmoid function is ***TanH***

- Bounded between -1 and 1

$$output = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}},$$

where

$$z = b + \sum_{j=1}^{J} w_j x_j$$

UCLAAnderson
SCHOOL of MANAGEMENT

b. Activation function: Rectified Linear Unit

The linear function truncated at zero is the ***Rectified Linear Unit (ReLU)***

- Like a Call Option payoff

$$output = \max\left(0, b + \sum_{j=1}^{J} w_j x_j\right)$$

- ***Now, non-linearity is clear, and so is a notion of sparsity***
- Neurons can be turned off for many values of input is the index is negative, but responds linearly if index is positive
- Adding more neurons that feeds back or forward can create complex non-linearities
  - These neurons do "make decisions" on their own

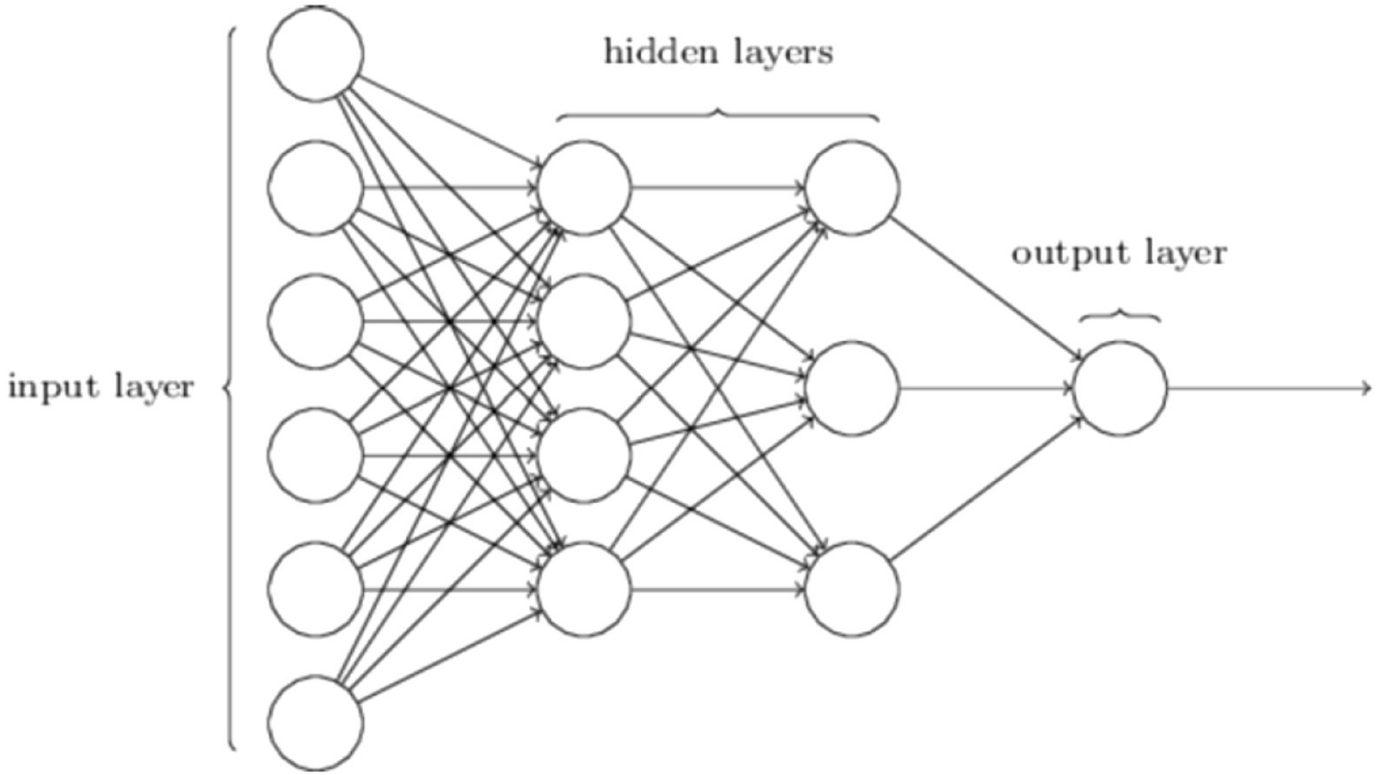
Deep Learning

UCLAAnderson
SCHOOL of MANAGEMENT

b. Multi-layered network

This is the '**deep learning**' part

- Even if used with arbitrary activation function, often called ***multi-layered perceptrons (MLPs)***

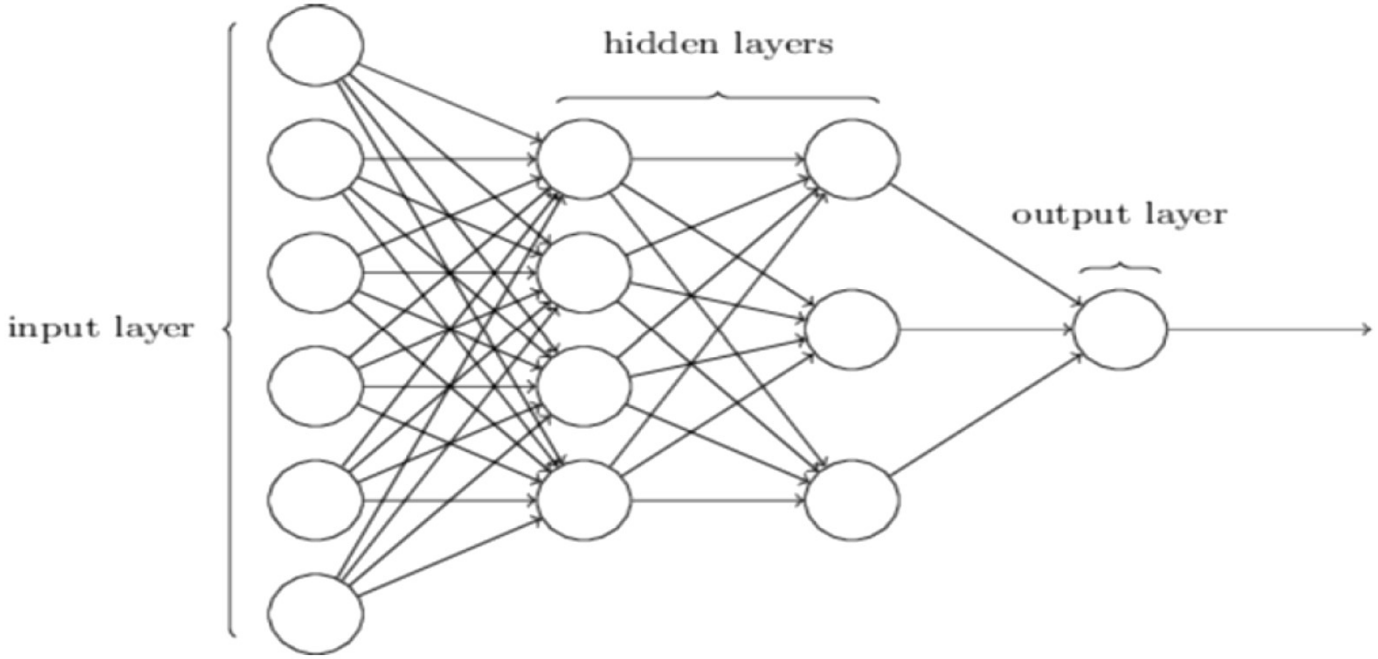


UCLAAnderson
SCHOOL of MANAGEMENT

b. Feed-forward network

The network below is a ***feed-forward*** network

- Hierarchical structure, earlier layers feed into later layers, no looping backwards
- Recurrent Neural Networks also allow backwards feeds and loops. We will not cover these.

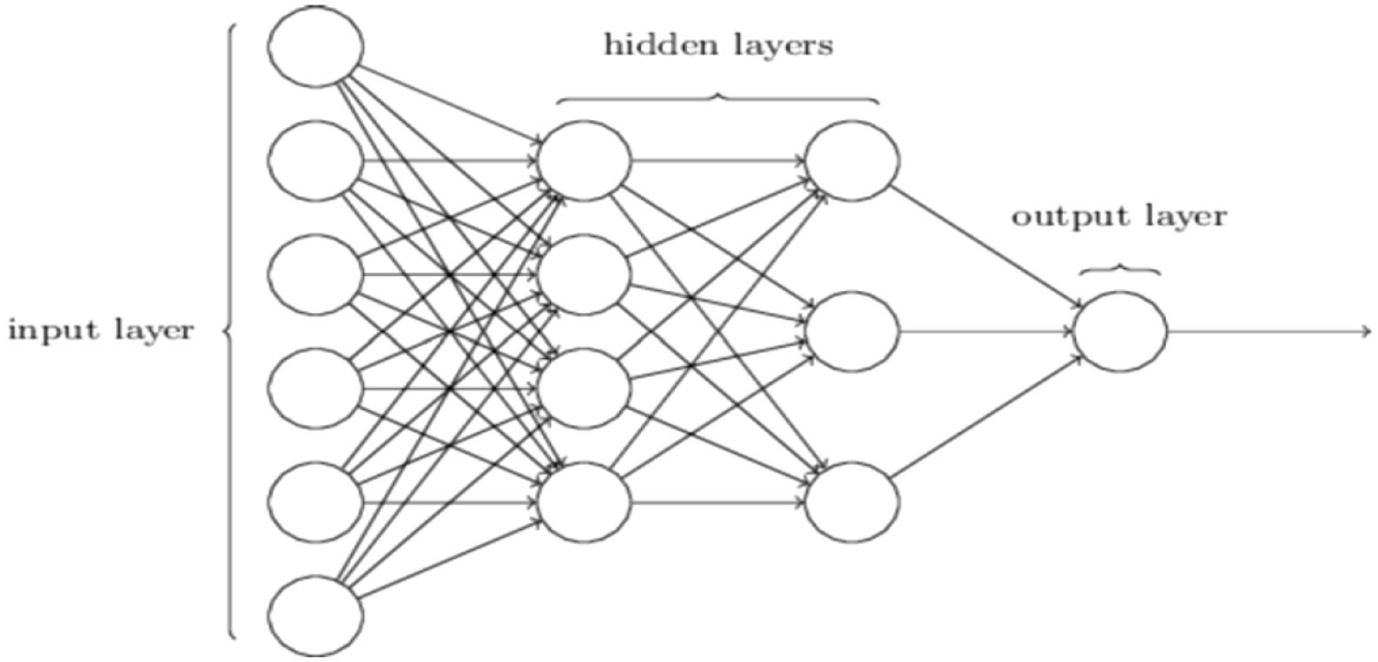


UCLAAnderson
SCHOOL of MANAGEMENT

b. Parameter proliferation

Quickly many parameters

- ***Each node has J weights and an intercept***
- Often the layers are decreasing in the number of nodes as one moves down the network so J depends on the layer
- Some regularization, cross-validation needed to tune network

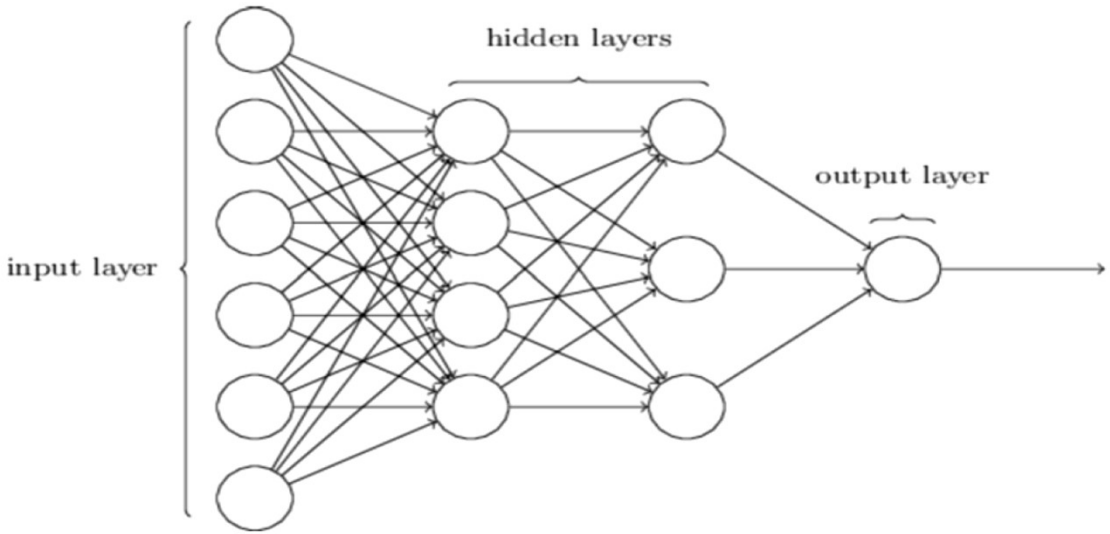


UCLAAnderson
SCHOOL of MANAGEMENT

b. Estimation complicated

While it in principle is possible to find the of globally optimal weights, it is typically too time-consuming and, sometimes, plain infeasible

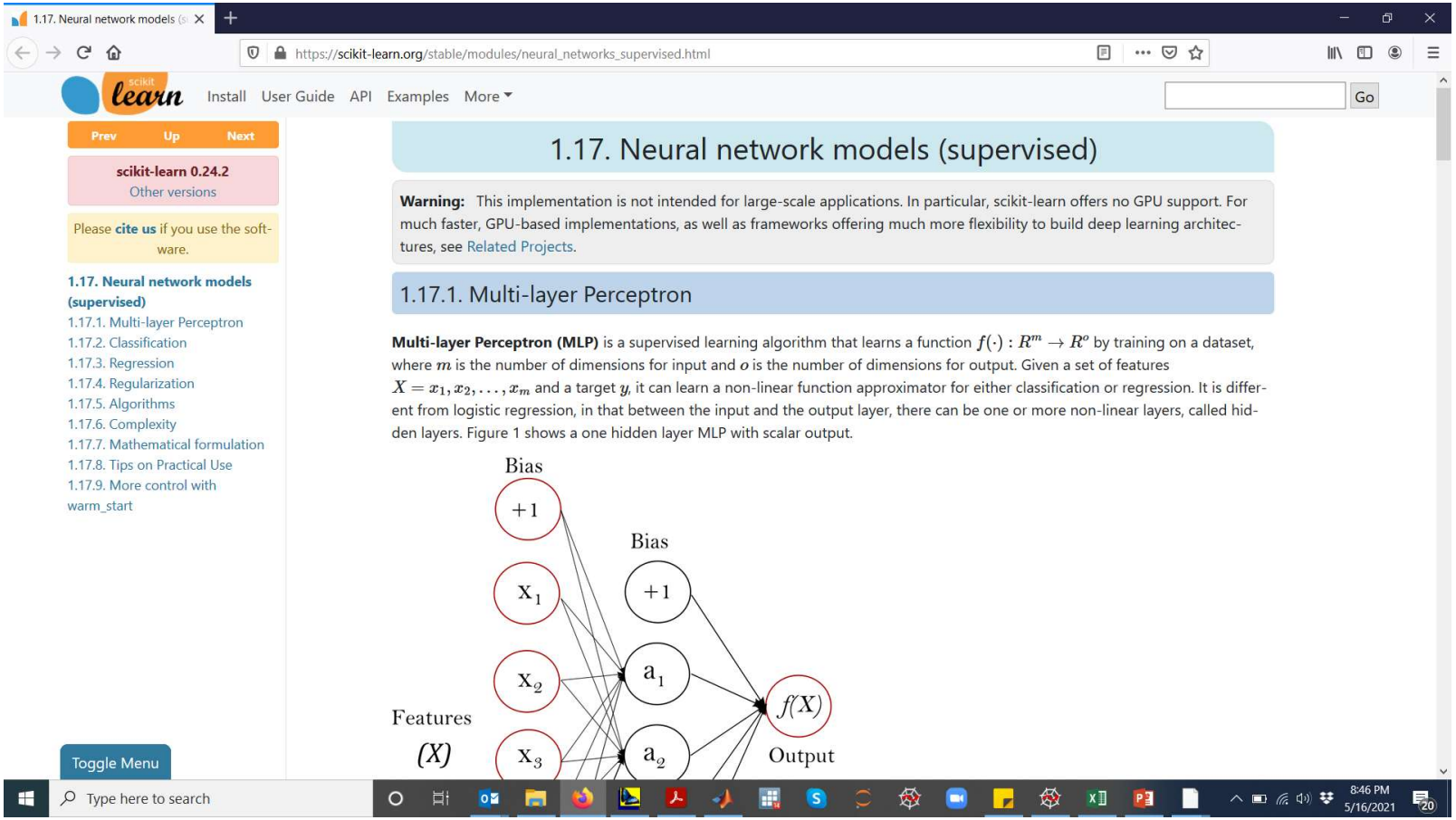
- Many different optimizers used
- A popular one: ***Stochastic Gradient Descent (SGD)***
- Simply at random choose a subset of observations in data-set to find next step in gradient descent



UCLAAnderson
SCHOOL of MANAGEMENT

b. Deep learning package

https://scikit-learn.org/stable/modules/neural_networks_supervised.html



UCLAAnderson
SCHOOL of MANAGEMENT

b. Bias nodes

In the previous slide, each layer has a "***bias node***"

- Consider the case where all inputs *x* are zero
- Regardless of weights *w*, *w'x* is zero in this case
- But, the data we are trying to predict may require a different value than zero when all *x*'s are zero

Intuitively, bias nodes serve ***the same purpose as an intercept*** in a linear regression

- Allows for a non-zero prediction even if all *x*'s are zero
- MLC function adds these nodes automatically

MLC sensitive to scale of features, so use *standard_scaler* first

UCLAAnderson
SCHOOL of MANAGEMENT

b. MLP Classifier

able/modules/generated/sklearn.neural_network.MLPClassifier.html 80%

Examples More

sklearn.neural_network.MLPClassifier

class sklearn.neural_network.**MLPClassifier**(*hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

[source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

UCLAAnderson
SCHOOL of MANAGEMENT

b. Predicting defaults

Neural network with two hidden layers

- 5 nodes in first layer, 2 in second

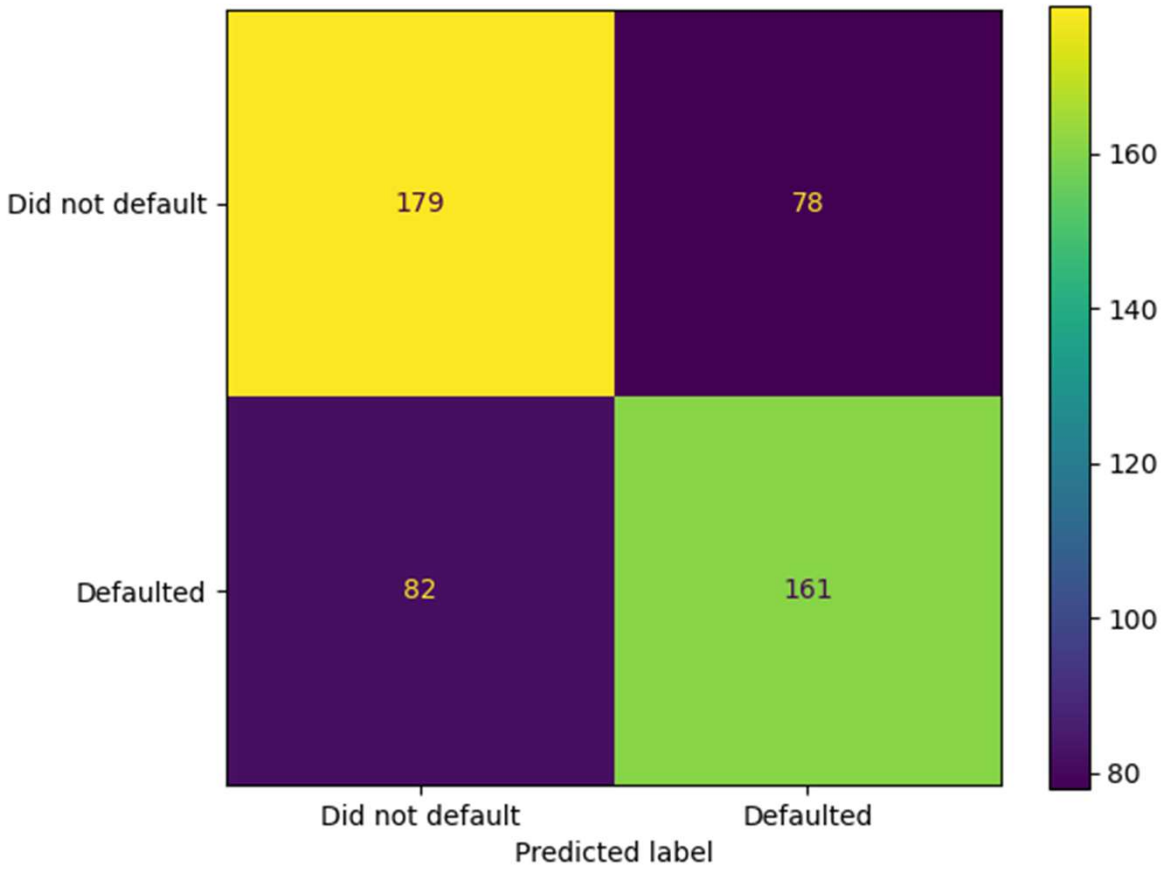
```
# Define networks
clf_nn = MLPClassifier(solver='lbfgs', alpha=1e-5,
                       hidden_layer_sizes=(5, 2), max_iter = 1000, random_state=1)

# Let's compare SVM with Neural nets with the same task
# Train the model
clf_nn.fit(X_train_scaled, y_train)

# Make predictions
plot_confusion_matrix(clf_nn,
                      X_test_scaled,
                      y_test,
                      values_format='d',
                      display_labels=["Did not default", "Defaulted"])
```

UCLAAnderson
SCHOOL of MANAGEMENT

b. Neural Net performance

The neural net does worse than SVM in this case

- But, we haven't tried tuning through cross-validation exercise



UCLAAnderson
SCHOOL of MANAGEMENT

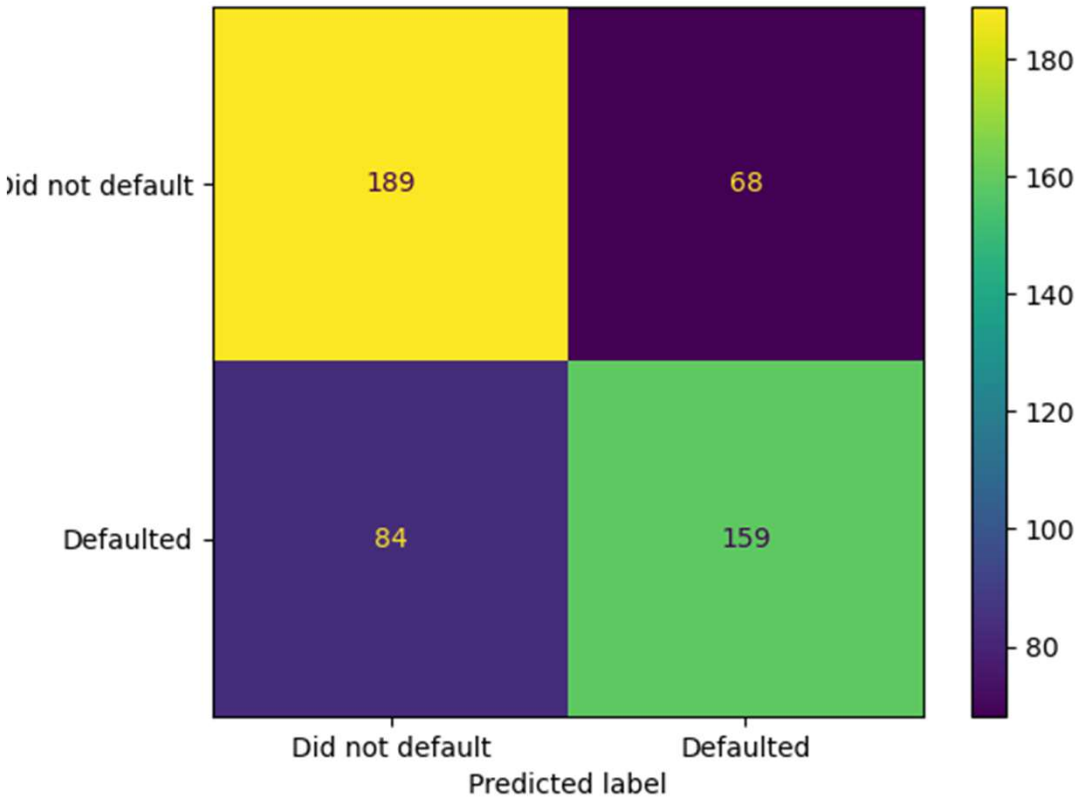
b. Alternative set of layers

```
# Let's try adding more hidden layers

clf_nn = MLPClassifier(solver='lbfgs', alpha=1e-5,
                       hidden_layer_sizes=(5,3,2,1), max_iter = 10000,
random_state=1)

clf_nn.fit(X_train_scaled, y_train)

plot_confusion_matrix(clf_nn,
                      X_test_scaled,
                      y_test,
                      values_format='d',
                      display_labels=["Did not default", "Defaulted"])
```

UCLAAnderson
SCHOOL of MANAGEMENT

b. Neural Net performance

Better for did not default, worse for default



UCLAAnderson
SCHOOL of MANAGEMENT

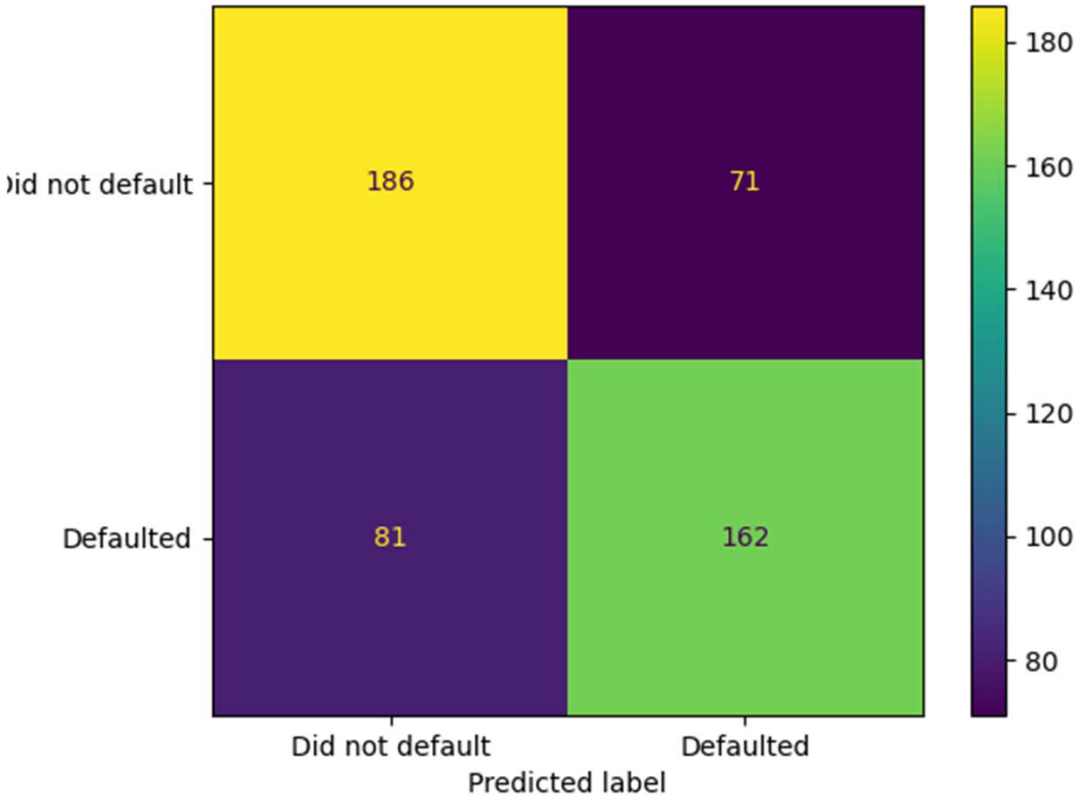
b. More regularization

```
# alpha governs degree of regularization when estimating weights (L2-penalty)

clf_nn = MLPClassifier(solver='lbfgs', alpha=1e-1,
                       hidden_layer_sizes=(5,3,2,1), max_iter = 10000,
random_state=1)

clf_nn.fit(X_train_scaled, y_train)

plot_confusion_matrix(clf_nn,
                      X_test_scaled,
                      y_test,
                      values_format='d',
                      display_labels=["Did not default", "Defaulted"])
```

UCLAAnderson
SCHOOL of MANAGEMENT

b. Neural Net performance

Now better for both cases, relative to first attempt



UCLAAnderson
SCHOOL of MANAGEMENT

b. MLP Regression

sklearn.neural_network.MLPRegressor

class `sklearn.neural_network.MLPRegressor`(*hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

[source]

Multi-layer Perceptron regressor.

This model optimizes the squared-loss using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

UCLAAnderson
SCHOOL of MANAGEMENT

b. Let's go back to return prediction

Same data on characteristics and returns as used in beginning of class

```
# Let's try predicting stock returns using the MLP Regressor procedure
# For brevity, use scaler across all data to normalize, which induces
# look-ahead bias. But, you know how to code this up in a loop where
# everything is tradeable, so I will leave that implementation to you

# Load data
StockRetAcct_DF = pd.read_csv("StockRetAcct_DT.csv")

# Drop missing observations
StockRetAcct_DF = StockRetAcct_DF.dropna()

# Excess return, add a constant column for use in regressions
StockRetAcct_DF['ExRet'] = np.exp(StockRetAcct_DF.lnAnnRet)- \
                           np.exp(StockRetAcct_DF.lnRf)

# Split data into dependent and independent variables
X = StockRetAcct_DF[['lnBM','lnProf','lnIssue','lnMom','lnME','lnInv','rv','lnROE']].copy()
y = StockRetAcct_DF['ExRet'].copy()

# Split data randomly into train/test
# since proportion not specified, defaults to 25% in test sample
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# scale X variables to mean zero and unit variance
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

UCLAAnderson
SCHOOL of MANAGEMENT

b. Train MLPRegressor + Score (R2)

We fit the MLP Regression model, score is basically R2

```
# Define networks
stock_nn = MLPRegressor(solver='lbfgs', alpha=1,activation='tanh',
                    hidden_layer_sizes=(5,2), max_iter = 10000, random_state=1)

# Train the model
stock_nn.fit(X_train_scaled, y_train)

# R2 of neural net fit, in sample
stock_nn.score(X_train_scaled, y_train, sample_weight=None)
Out[76]: 0.037023149542497524

# R2 of neural net fit, out of sample
stock_nn.score(X_test_scaled, y_test, sample_weight=None)
Out[77]: 0.02636141749041343
```

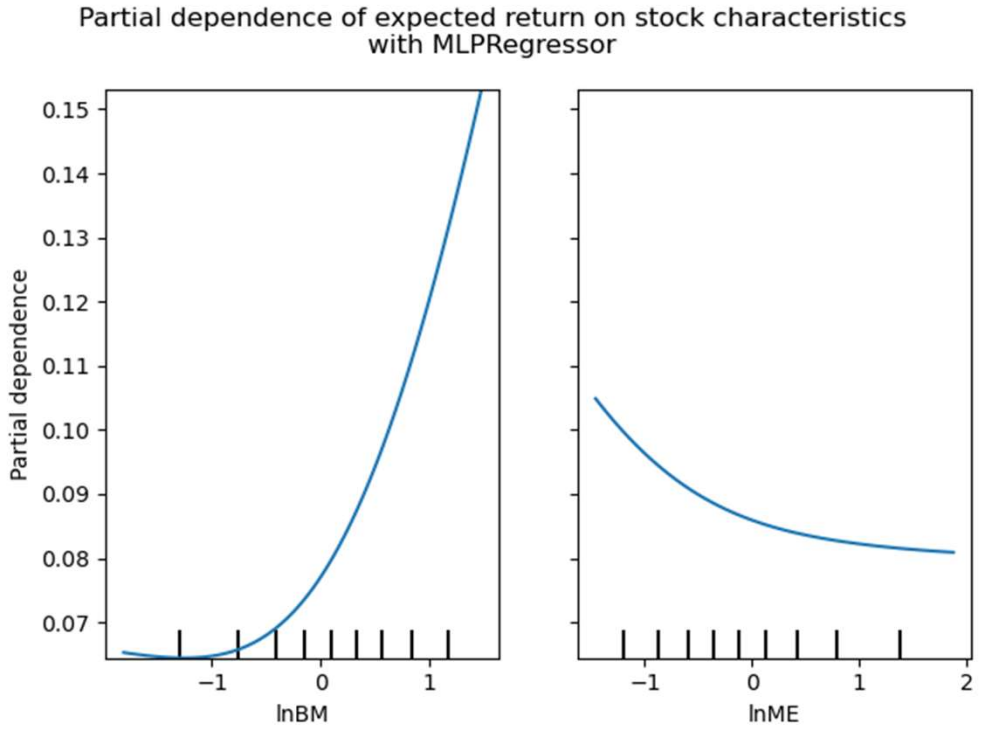
UCLAAnderson
SCHOOL of MANAGEMENT

b. Partial Dependence Plot

Plot partial dependence for lnBM and lnME

- Individual effect of each feature, marginalizing out all others

```
features = ['lnBM', 'lnME']
display = plot_partial_dependence(
       stock_nn, X_train_scaled, features, kind="average")
display.figure_.suptitle(
    'Partial dependence of expected return on stock characteristics\n'
    'with MLPRegressor'
)
display.figure_.subplots_adjust(hspace=0.3)
```



UCLAAnderson
SCHOOL of MANAGEMENT

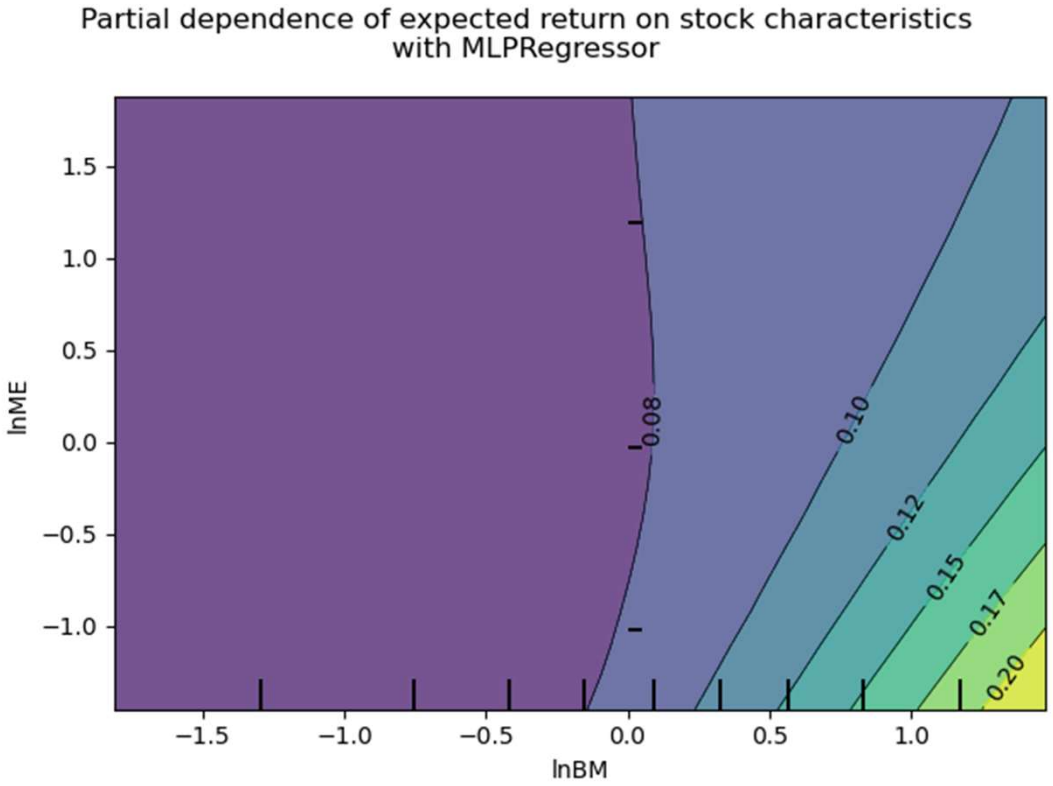
b. Interaction effects

Plot partial dependence interaction of lnBM and lnME

```
features = [('lnBM','lnME')] # note interaction term using ()
display = plot_partial_dependence(
       stock_nn, X_train_scaled, features, kind="average", grid_resolution=20)
display.figure_.suptitle(
    'Partial dependence of expected return on stock characteristics\n'
    'with MLPRegressor'
)
```

Notice: MLP regression captures stylized fact from lecture 1:

Value effect strongest in small stocks

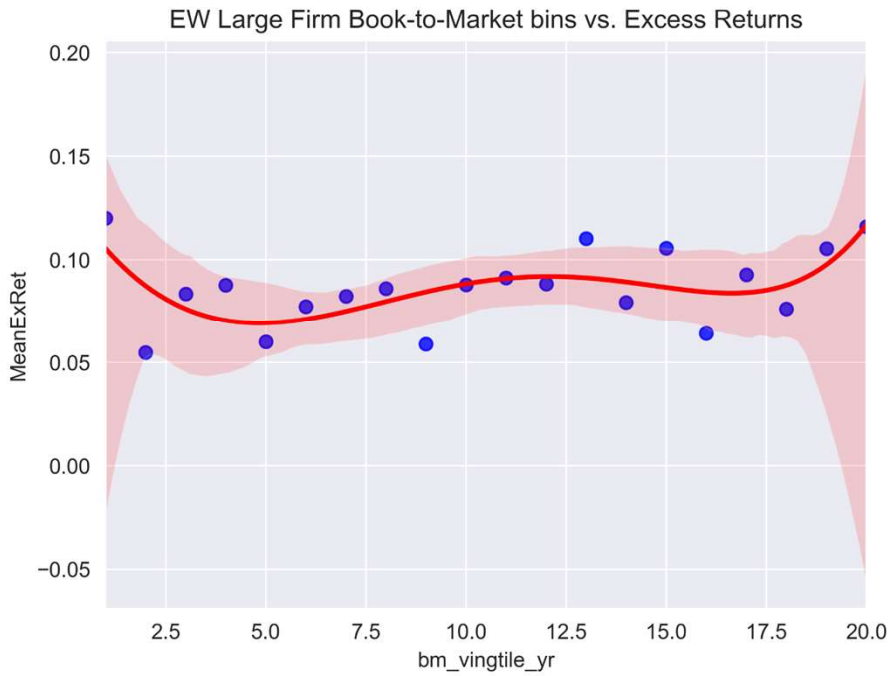


UCLAAnderson
SCHOOL of MANAGEMENT

b. Interaction effects

Below plots are reproduced from lecture 1: same effect!

- Thus: deep learning "automatically" uncover important data nonlinearities



c. Machine Learning in Finance

Empirical Asset Pricing via Machine Learning*

Shihao Gu
Booth School of Business, University of Chicago

Bryan Kelly
Yale University, AQR Capital Management, and NBER

Dacheng Xiu
Booth School of Business, University of Chicago

We perform a comparative analysis of machine learning methods for the canonical problem of empirical asset pricing: measuring asset risk premiums. We demonstrate large economic gains to investors using machine learning forecasts, in some cases doubling the performance of leading regression-based strategies from the literature. We identify the best-performing methods (trees and neural networks) and trace their predictive gains to allowing nonlinear predictor interactions missed by other methods. All methods agree on the same set of dominant predictive signals, a set that includes variations on momentum, liquidity, and volatility. (*JEL* C52, C55, C58, G0, G1, G17)

UCLAAnderson
SCHOOL of MANAGEMENT

c. Paper overview

Evaluate machine learning tools for standard expected return regressions

- Argue one can improve on Sharpe ratio of market via market timing (from 0.51 to 0.71)

- Argue one can, again in out of sample exercise, get Sharpe ratio of 1.35 from long-short strategy on U.S. stocks, more than double that from standard factor models

- Find that decision trees (random forest, xgboost) and deep neural networks do the best, though I'm not sure about tradeability of all positions (transaction costs, price impact of trade)

- Machine learning both an input to stock picking for AQR and their marketing material

UCLAAnderson
SCHOOL of MANAGEMENT

c. Parameter tuning and model comparison

Authors split available sample data into 3 sub-samples:

1. Training sample (used to estimate model, conditional on tuning parameters)
2. Validation sample (used to find tuning parameters)
3. "Truly out-of-sample"-sample, for model comparison (not used in estimation)

- While each machine learning technique has different model parameters, tuning parameters, and optimization methods (e.g., stochastic gradient descent or greedy), the three sub-samples above remain the same

UCLAAnderson
SCHOOL of MANAGEMENT

c. ML methods evaluated in the paper

The problem of finding expected returns (forecasting monthly returns) is a regression and not a classification problem

- Therefore limit study to common methods for such problems

1. Linear panel regression
2. Linear regularized (penalized) regression (elastic net)
3. Generalized linear model (adding non-linear transformation of variables)
4. Linear with dimensionality reduction (PCA)
5. Decision trees (boosted, random forest)
6. Hidden layer neural networks (deep learning)

UCLAAnderson
SCHOOL of MANAGEMENT

c. Overview of their neural net approach

The paper goes into detail in all their different methods and how they implement. We will just go over the neural net approach here, but I recommend you read the paper for the others.

Consider models with up to 5 hidden layers.

Simplest model has 1 hidden layer with 32 nodes; second has two layers with 32 and 16 nodes, respectively; third has three layers with 32, 16, and 8 nodes, respectively; etc.

Minimize MSE in validation sample. Use Stochastic Gradient Descent with Early Stopping, adaptive learning rate shrinkage, batch normalization, average (ensemble) across nnets estimated with different random seeds.

UCLAAnderson
SCHOOL of MANAGEMENT

c. The model in more detail

The authors choose Rectified Linear Unit as their activation function:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise,} \end{cases}$$

Our neural network model has the following general formula. Let $K^{(l)}$ denote the number of neurons in each layer $l = 1, \ldots, L$. Define the output of neuron k in layer l as $x_k^{(l)}$. Next, define the vector of outputs for this layer (augmented to include a constant, $x_0^{(l)}$) as $x^{(l)} = (1, x_1^{(l)}, \ldots, x_{K^{(l)}}^{(l)})'$. To initialize the network, similarly define the input layer using the raw predictors, $x^{(0)} = (1, z_1, \ldots, z_N)'$. The recursive output formula for the neural network at each neuron in layer $l > 0$ is then

$$x_k^{(l)} = \text{ReLU}\left(x^{(l-1)'} \theta_k^{(l-1)}\right), \tag{17}$$

with final output

$$g(z;\theta) = x^{(L-1)'} \theta^{(L-1)}. \tag{18}$$

The number of weight parameters in each hidden layer l is $K^{(l)}(1 + K^{(l-1)})$, plus another $1 + K^{(L-1)}$ weights for the output layer.

UCLAAnderson
SCHOOL of MANAGEMENT

c. The expected return equation

Throughout our analysis, we define the baseline set of stock-level covariates $z_{i,t}$ as

$$z_{i,t} = x_t \otimes c_{i,t}, \tag{21}$$

where $c_{i,t}$ is a $P_c \times 1$ matrix of characteristics for each stock i, and x_t is a $P_x \times 1$ vector of macroeconomic predictors (and are thus common to all stocks, including a constant). Thus, $z_{i,t}$ is a $P \times 1$ vector of features for predicting individual stock returns (with $P = P_c P_x$) and includes interactions between stock-level characteristics and macroeconomic state variables. The total number of covariates is $94 \times (8+1)+74 = 920$.

$$\mathrm{E}_t(r_{i,t+1}) = g^\star(z_{i,t}).$$

c. The data sample

2.1 Data and the overarching model

We obtain monthly total individual equity returns from CRSP for all firms listed in the NYSE, AMEX, and NASDAQ. Our sample begins in March 1957 (the start date of the S&P 500) and ends in December 2016, totaling 60 years. The number of stocks in our sample is almost 30,000, with the average number of stocks per month exceeding 6,200.[28] We also obtain the Treasury-bill rate to proxy for the risk-free rate from which we calculate individual excess returns.

In addition, we build a large collection of stock-level predictive characteristics based on the cross-section of stock returns literature. These include 94 characteristics[29] (61 of which are updated annually, 13 are updated quarterly, and 20 are updated monthly). In addition, we include 74 industry dummies corresponding to the first two digits of Standard Industrial Classification (SIC) codes. Table A.6 in the Internet Appendix provides the details of these characteristics.[30]

We also construct eight macroeconomic predictors following the variable definitions detailed in Welch and Goyal (2008), including dividend-price ratio (dp), earnings-price ratio (ep), book-to-market ratio (bm), net equity expansion (ntis), Treasury-bill rate (tbl), term spread (tms), default spread (dfy), and stock variance (svar).[31]

UCLAAnderson
SCHOOL of MANAGEMENT

c. The characteristics

- Stock level characteristics:

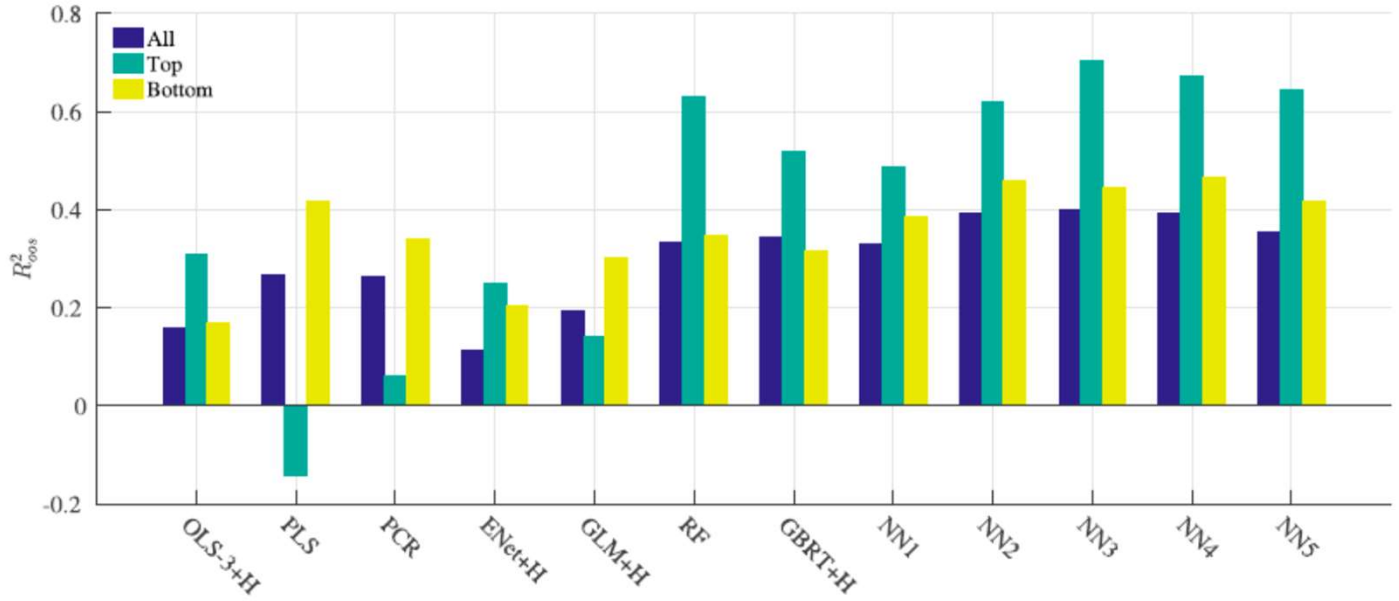
four categories. The first are based on recent price trends, including 5 of the top-7 variables in Figure 5: short-term reversal (mom1m), stock momentum (mom12m), momentum change (chmom), industry momentum (indmom), recent maximum return (maxret), and long-term reversal (mom36m). Next are liquidity variables, including turnover and turnover volatility (turn, SD_turn), log market equity (mvel1), dollar volume (dolvol), Amihud illiquidity (ill), number of zero trading days (zerotrade), and bid-ask spread (baspread). Risk measures constitute the third influential group, including total and idiosyncratic return volatility (retvol, idiovol), market beta (beta), and beta squared (betasq). The last group includes valuation ratios and fundamental signals, such as earnings-to-price (ep), sales-to-price (sp), asset growth (agr), and number of recent earnings increases (nincr).

UCLAAnderson
SCHOOL of MANAGEMENT

c. OOS R2 in return prediction

Monthly out-of-sample stock-level prediction performance (percentage R^2_{oos})

	OLS +H	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
All	−3.46	0.16	0.27	0.26	0.11	0.19	0.33	0.34	0.33	0.39	0.40	0.39	0.36
Top 1,000	−11.28	0.31	−0.14	0.06	0.25	0.14	0.63	0.52	0.49	0.62	0.70	0.67	0.64
Bottom 1,000	−1.30	0.17	0.42	0.34	0.20	0.30	0.35	0.32	0.38	0.46	0.45	0.47	0.42



In this table, we report monthly R^2_{oos} for the entire panel of stocks using OLS with all variables (OLS), OLS using only size, book-to-market, and momentum (OLS-3), PLS, PCR, elastic net (ENet), generalize linear model (GLM), random forest (RF), gradient boosted regression trees (GBRT), and neural networks with 1 to 5 layers (NN1–NN5). "+H" indicates the use of Huber loss instead of the l_2 loss. We also report these R^2_{oos} within subsamples that include only the top-1,000 stocks or bottom-1,000 stocks by market value. The lower panel provides a visual comparison of the R^2_{oos} statistics in the table (omitting OLS because of its large negative values).

UCLAAnderson
SCHOOL of MANAGEMENT

c. Variable Importance

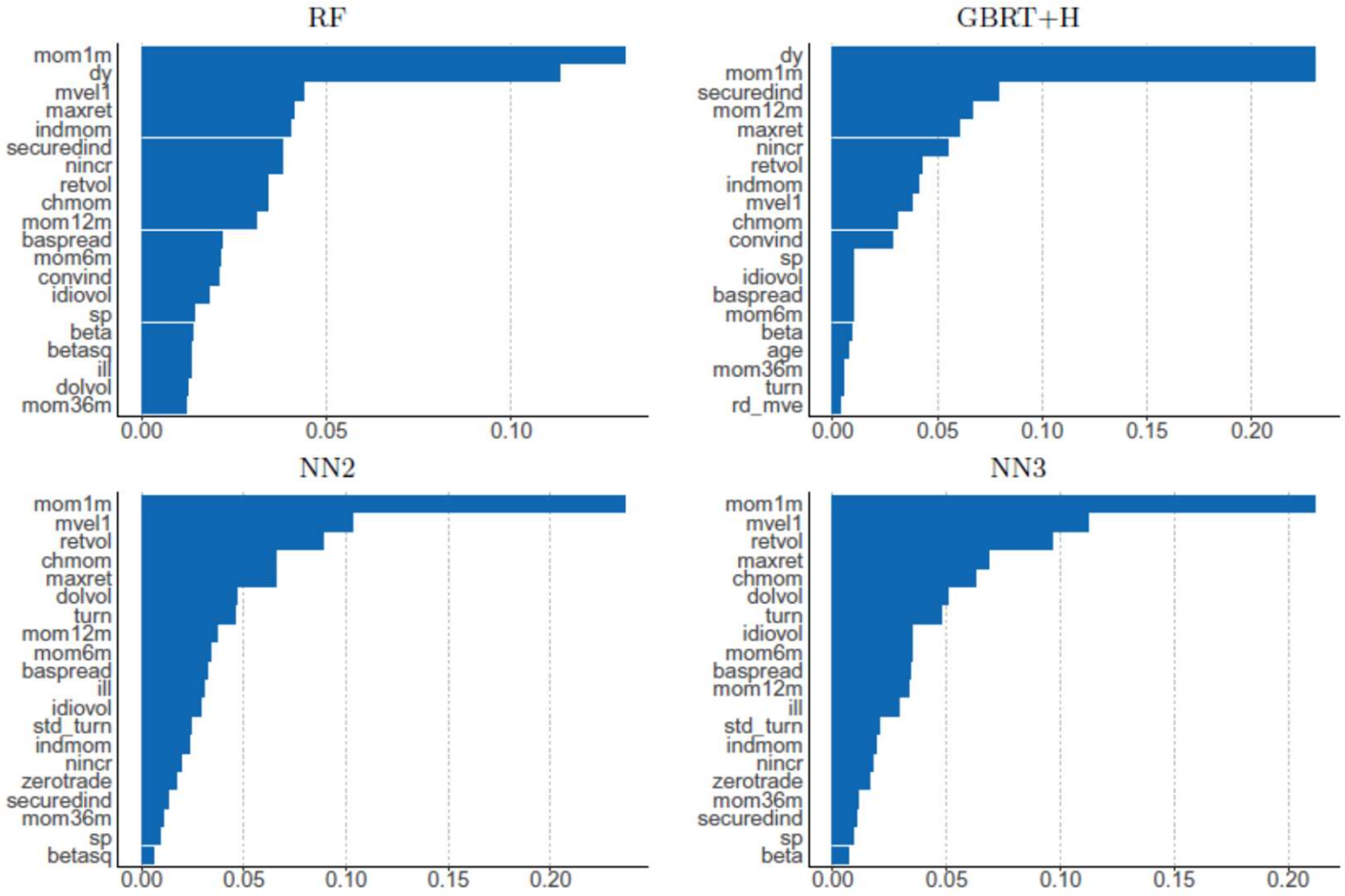


Figure 4
Variable importance by model
Variable importance for the top-20 most influential variables in each model. Variable importance is an average over all training samples. Variable importance within each model is normalized to sum to one.

UCLAAnderson
SCHOOL of MANAGEMENT

c. Nonlinear effects in data

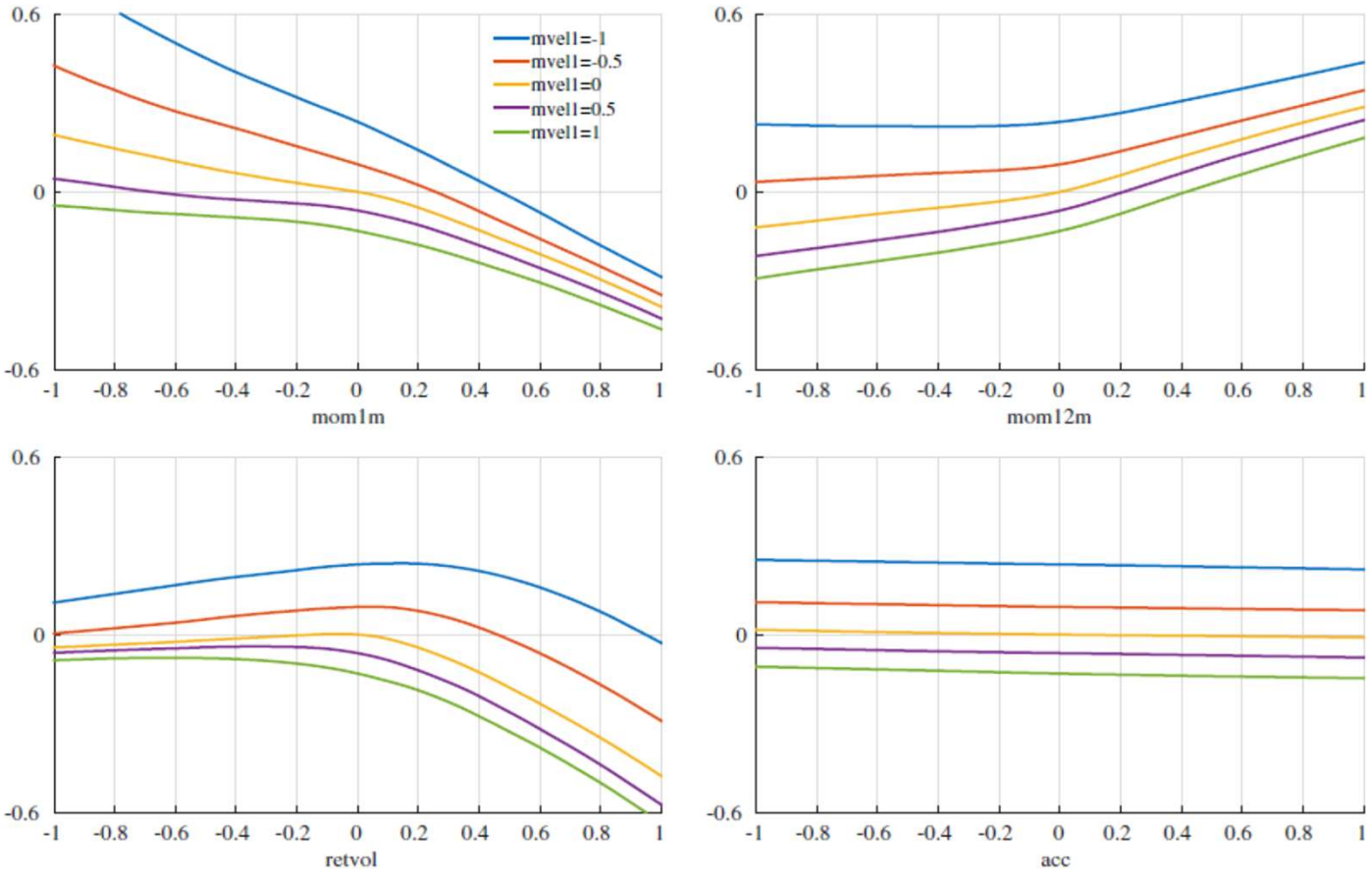


Figure 7
Expected returns and characteristic interactions (NN3)
The panels should the sensitivity of the expected monthly percentage returns (vertical axis) to the interactions effects for mvel1 with mom1m, mom12m, retvol, and acc in model NN3 (holding all other covariates fixed at their median values).

UCLAAnderson
SCHOOL of MANAGEMENT

c. Nonlinear effects in data

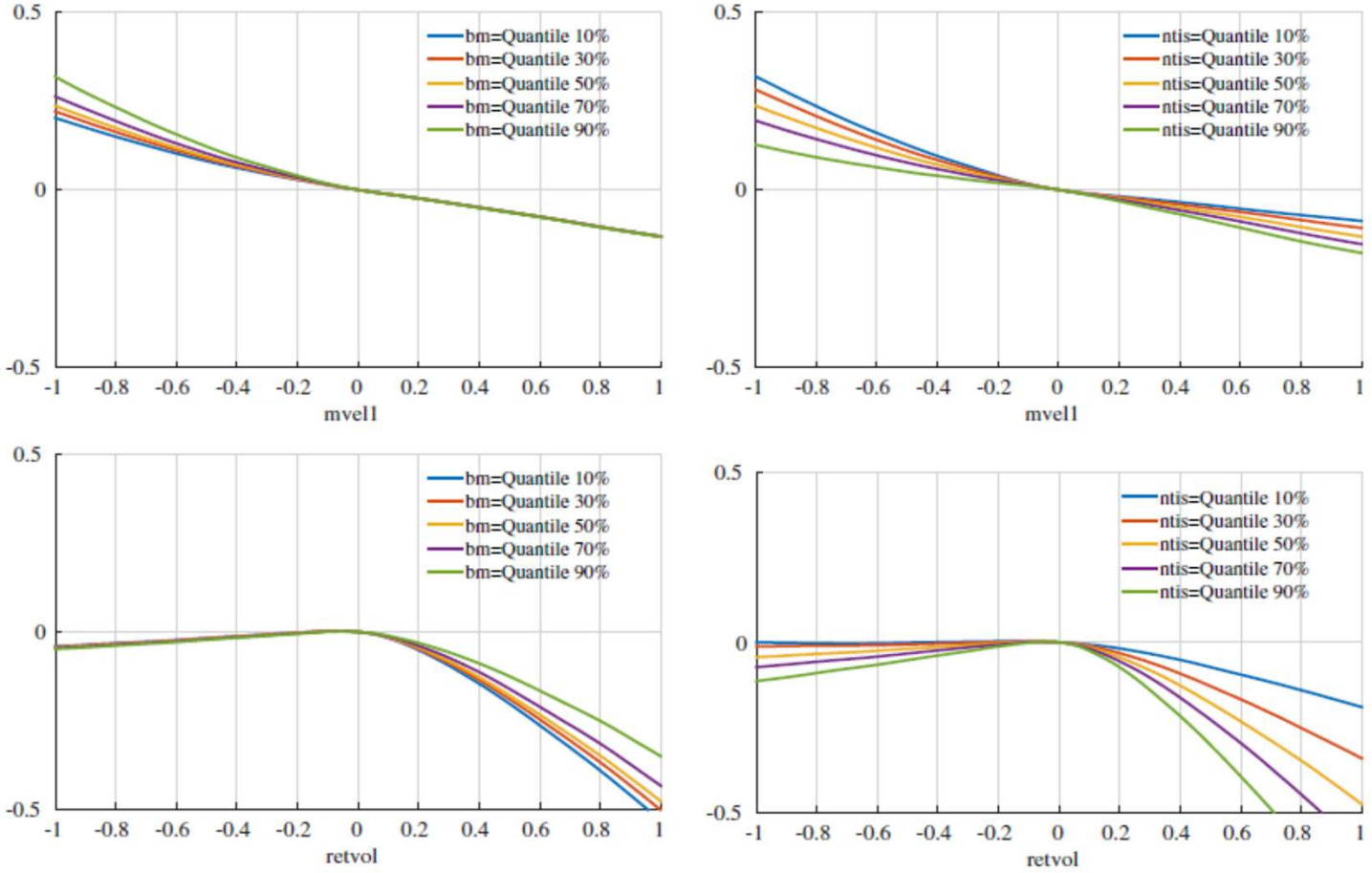


Figure 8
Expected returns and characteristic/macroeconomic variable interactions (NN3)
The panels show the sensitivity of expected monthly percentage returns (vertical axis) to interactions effects for mvel1 and retvol with bm and ntis in model NN3 (holding all other covariates fixed at their median values).

UCLAAnderson
SCHOOL of MANAGEMENT

c. Portfolio sorts based on E[r]

	NN3				NN4				NN5			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low(L)	−0.03	−0.43	7.73	−0.19	−0.12	−0.52	7.69	−0.23	−0.23	−0.51	7.69	−0.23
2	0.34	0.30	6.38	0.16	0.30	0.33	6.16	0.19	0.23	0.31	6.10	0.17
3	0.51	0.57	5.27	0.37	0.50	0.42	5.18	0.28	0.45	0.54	5.02	0.37
4	0.63	0.66	4.69	0.49	0.62	0.60	4.51	0.46	0.60	0.67	4.47	0.52
5	0.71	0.69	4.41	0.55	0.72	0.69	4.26	0.56	0.73	0.77	4.32	0.62
6	0.79	0.76	4.46	0.59	0.81	0.84	4.46	0.65	0.85	0.86	4.35	0.68
7	0.88	0.99	4.77	0.72	0.90	0.93	4.56	0.70	0.96	0.88	4.76	0.64
8	1.00	1.09	5.47	0.69	1.03	1.08	5.13	0.73	1.11	0.94	5.17	0.63
9	1.21	1.25	5.94	0.73	1.23	1.26	5.93	0.74	1.34	1.02	6.02	0.58
High(H)	1.83	1.69	7.29	0.80	1.89	1.75	7.51	0.81	1.99	1.46	7.40	0.68
H-L	1.86	2.12	6.13	1.20	2.01	2.26	5.80	1.35	2.22	1.97	5.93	1.15

In this table, we report the performance of prediction-sorted portfolios over the 30-year out-of-sample testing period. All stocks are sorted into deciles based on their predicted returns for the next month. Columns "Pred," "Avg," "SD," and "SR" provide the predicted monthly returns for each decile, the average realized monthly returns, their standard deviations, and Sharpe ratios, respectively. All portfolios are value weighted.

UCLAAnderson
SCHOOL of MANAGEMENT

c. Alpha regressions

Drawdowns, turnover, and risk-adjusted performance of machine learning portfolios

	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1	NN2	NN3	NN4	NN5
Drawdowns and turnover (value weighted)												
Max DD(%)	69.60	41.13	42.17	60.71	37.09	52.27	48.75	61.60	55.29	30.84	51.78	57.52
Max 1M loss(%)	24.72	27.40	18.38	27.40	15.61	26.21	21.83	18.59	37.02	30.84	33.03	38.95
Turnover(%)	58.20	110.87	125.86	151.59	145.26	133.87	143.53	121.02	122.46	123.50	126.81	125.37
Drawdowns and turnover (equally weighted)												
Max DD(%)	84.74	32.35	31.39	33.70	21.01	46.42	37.19	18.25	25.81	17.34	14.72	21.78
Max 1M loss(%)	37.94	32.35	22.33	32.35	15.74	34.63	22.34	12.79	25.81	12.50	9.01	21.78
Turnover(%)	57.24	104.47	118.07	142.78	137.97	120.29	134.24	112.35	112.43	113.76	114.17	114.34
Risk-adjusted performance (value weighted)												
Mean ret.	0.94	1.02	1.22	0.60	1.06	1.62	0.99	1.81	1.92	2.12	2.26	1.97
FF5+Mom α	0.39	0.24	0.62	−0.23	0.38	1.20	0.66	1.20	1.33	1.52	1.76	1.43
$t(\alpha)$	2.76	1.09	2.89	−0.89	1.68	3.95	3.11	4.68	4.74	4.92	6.00	4.71
R^2	78.60	34.95	39.11	28.04	30.78	13.43	20.68	27.67	25.81	20.84	20.47	18.23
IR	0.54	0.21	0.57	−0.17	0.33	0.77	0.61	0.92	0.93	0.96	1.18	0.92

UCLAAnderson
SCHOOL of MANAGEMENT

c. Cumulative return performance

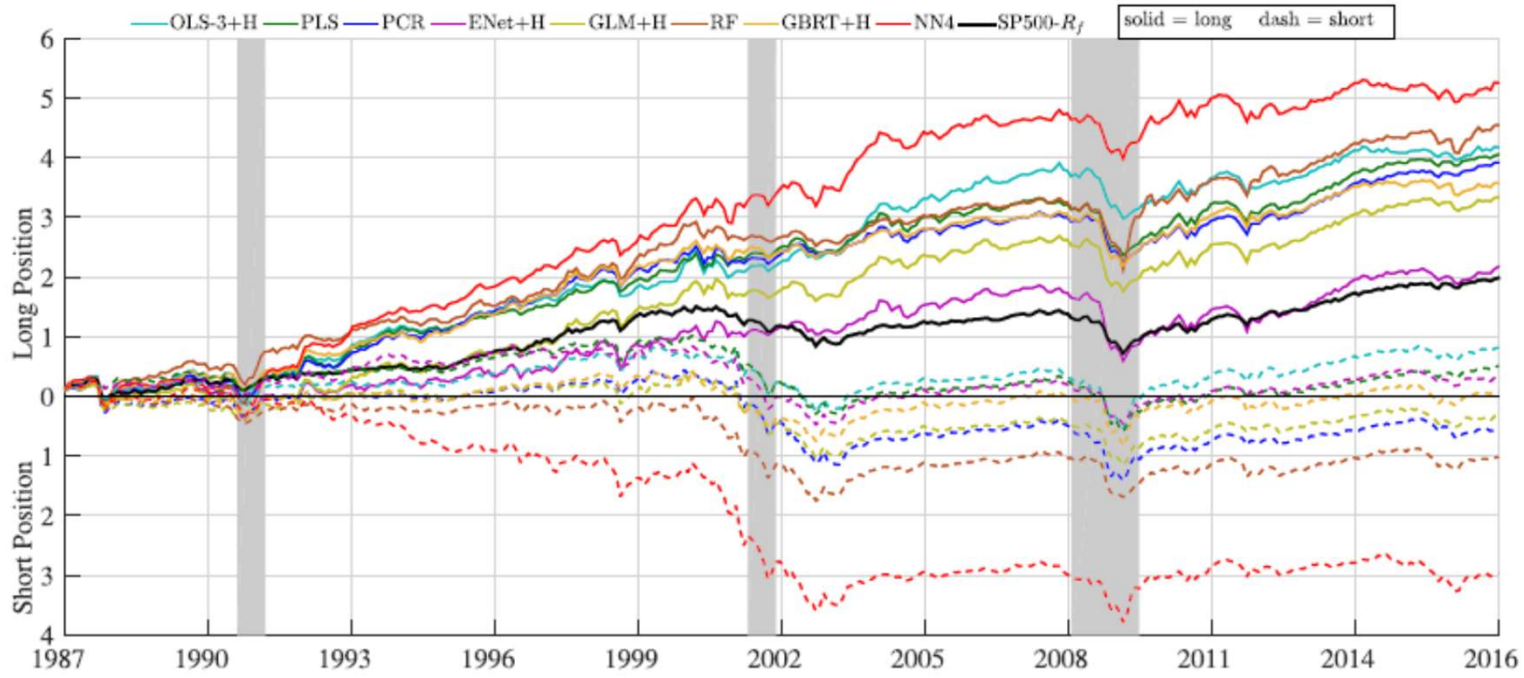


Figure 9
Cumulative return of machine learning portfolios
The figure shows the cumulative log returns of portfolios sorted on out-of-sample machine learning return forecasts. The solid and dashed lines represent long (top decile) and short (bottom decile) positions, respectively. The shaded periods show NBER recession dates. All portfolios are value weighted.

UCLAAnderson
SCHOOL of MANAGEMENT

c. Conclusions from paper

- The authors find that ***"shallow" learning outperforms "deep" learning***, which differs from the typical conclusion in other fields, such as computer vision or bioinformatics, and is likely ***due to*** the comparative dearth of data and ***low signal-to-noise ratio*** in asset pricing problems.

- ***Machine learning methods are most valuable for forecasting larger and more liquid stock returns and portfolios.***

- Lastly, they find that ***all* methods agree on a fairly small set of dominant predictive signals**, the most powerful predictors being associated with ***price trends including return reversal and momentum***. The next most powerful predictors are measures of ***stock liquidity, stock volatility, and valuation ratios***.