Building a Robot Judge: Data Science for Decision-Making

7. Deep Learning Essentials

Recap: Reading Response Essays

- Critical reading is an important skill:
  - useful for writing/reading reports
  - understanding the structure/code behind a paper – why have papers and not textbooks?

Recap: Reading Response Essays

- Critical reading is an important skill:
  - useful for writing/reading reports
  - understanding the structure/code behind a paper – why have papers and not textbooks?
- Some common patterns in the responses:
  - great summaries
  - more mixed on the critique/evaluation

Recap: Reading Response Essays

- Critical reading is an important skill:
  - useful for writing/reading reports
  - understanding the structure/code behind a paper – why have papers and not textbooks?
- Some common patterns in the responses:
  - great summaries
  - more mixed on the critique/evaluation

Another nice guide (on HW Assignments page):
`https://www.icpsr.umich.edu/files/instructors/How_to_Read_a_Journal_Article.pdf`

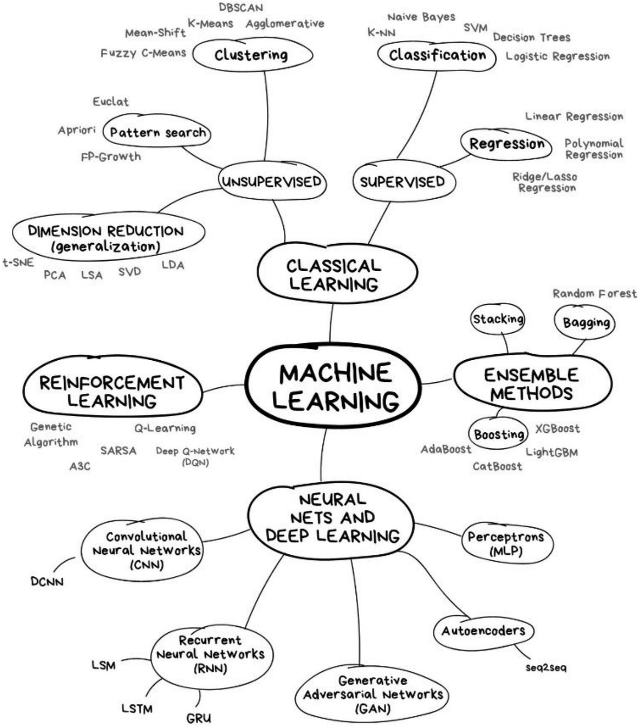
- could be useful for peer feedback on classmates' response essays.

Group Discussion: Real-World Algorithmic Rating System

- partner up into groups of 2-4.
- Flip a coin (or equivalent):
  - heads: `bit.ly/UK-visas` (Visa Algorithm)
  - tails: `bit.ly/UK-exams` (Grading Algorithm)
- Assignment (10 minutes):
  - 2 minutes: one student should summarize/describe the ML decision system described in the article.
  - 6 minutes: brainstorm at least 2 ways the system could be improved.
  - 2 minutes: write down outcomes in the padlet (see instructions in header): `https://padlet.com/eash44/ky22ublyvhr54050`

Learning Objectives

1. **Implement and evaluate machine learning pipelines.**
    - **Evaluate (find problems in) existing machine learning pipelines.**
    - **Design a pipeline to solve a given ML problem.**
    - **Implement some standard pipelines in Python.**
2. Implement and evaluate causal inference designs.
3. Understand how (not) to use data science tools (ML and CI) to support expert decision-making.


MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
SUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Pattern search
Euclat
Apriori
FP-Growth
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq

Objectives in an ML Project

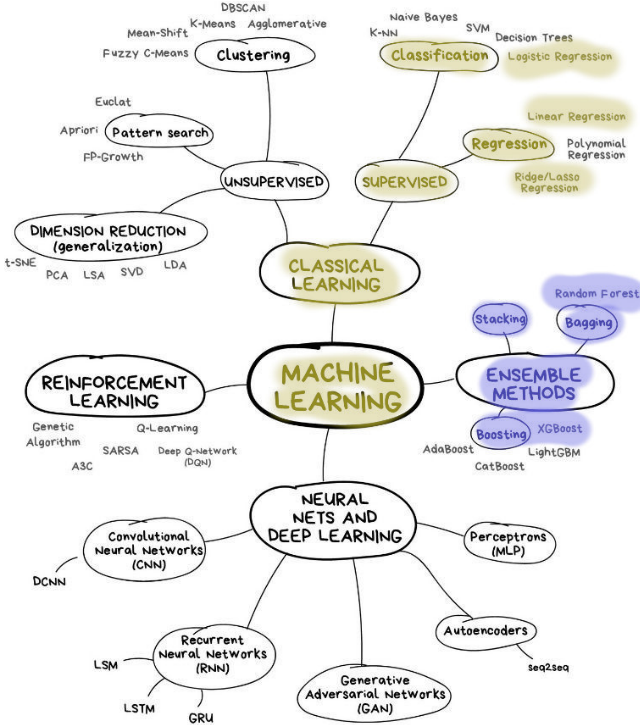
1. **What is the policy problem or research question?**

Objectives in an ML Project

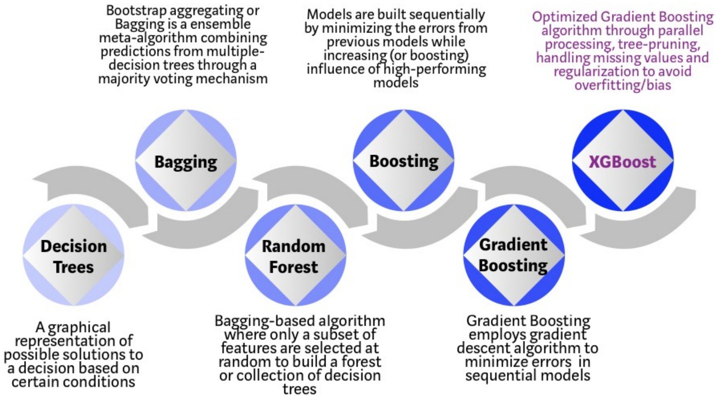
1. **What is the policy problem or research question?**
2. Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata

Objectives in an ML Project

1. **What is the policy problem or research question?**
2. Data:
   - obtain, clean, preprocess, and link.
   - Produce descriptive visuals and statistics on the text and metadata
3. Machine learning:
   - Select a model and train it.
   - Fine-tune hyperparameters for out-of-sample fit.
   - Interpret predictions using model explanation methods.



MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
SUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Pattern search
Euclat
Apriori
FP-Growth
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq




```python
from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(X_train, y_train,
          early_stopping_rounds=10,
          eval_metric="logloss",
          eval_set=[(X_eval, y_eval)]
          )

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

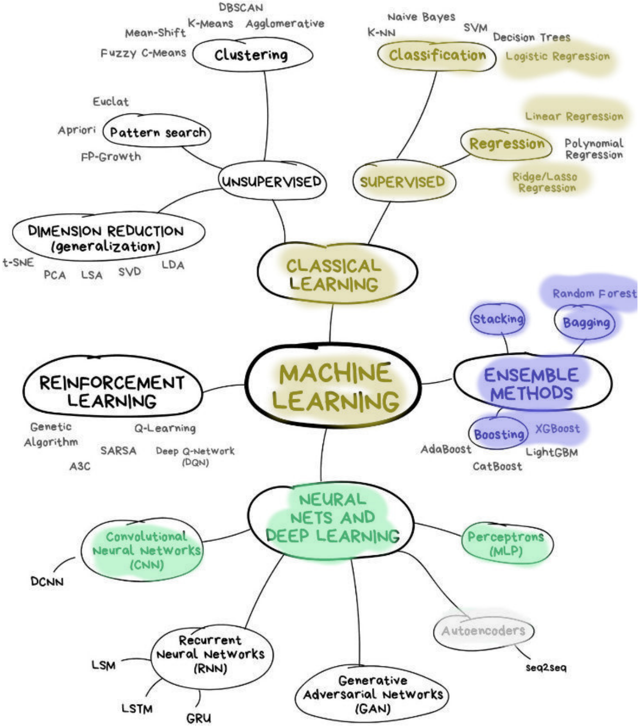
L2-regularized Logistic Regression in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(input_dim=num_features,
                units=2,  # output dimension
                activation='sigmoid',
                kernel_regularizer='l2',
                ))
```

L2-regularized Logistic Regression in Keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(input_dim=num_features,
                units=2,  # output dimension
                activation='sigmoid',
                kernel_regularizer='l2',
                ))
```

```
model.compile(optimizer='sgd', # stochastic gradient descent
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=100,
          validation_data=(x_val, y_val))
```


MACHINE LEARNING
CLASSICAL LEARNING
UNSUPERVISED
SUPERVISED
Clustering
DBSCAN
K-Means
Agglomerative
Mean-Shift
Fuzzy C-Means
Pattern search
Euclat
Apriori
FP-Growth
DIMENSION REDUCTION (generalization)
t-SNE
PCA
LSA
SVD
LDA
Classification
Naive Bayes
K-NN
SVM
Decision Trees
Logistic Regression
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
ENSEMBLE METHODS
Stacking
Bagging
Random Forest
Boosting
XGBoost
AdaBoost
LightGBM
CatBoost
REINFORCEMENT LEARNING
Genetic Algorithm
Q-Learning
SARSA
Deep Q-Network (DQN)
A3C
NEURAL NETS AND DEEP LEARNING
Convolutional Neural Networks (CNN)
DCNN
Perceptrons (MLP)
Autoencoders
seq2seq
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)

Activity: True/False Quiz

1. Mean absolute error is less sensitive to large regression errors than mean squared error.
2. L2 or ridge penalties output a sparse model where weak predictors go to zero.
3. If labels are balanced, accuracy is preferred to F1 as a classification metric.
4. To make sure I miss no important documents, I should maximize precision.
5. xgboost is an optimized random forest model.
6. Double ML solves the problem of unobserved confounders
7. A convex cost function is necessary for machine learning algorithms to work.

Outline

Outline

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, for example image recognition / text generation

- Neural networks $\leftrightarrow$ deep learning models
  - solve machine learning problems, just like logistic regression or gradient boosted machines
  - use tensorflow/keras or torch, rather than sklearn or xgboost.
- **why use neural nets?**
  - sometimes outperform standard ML techniques on standard problems
  - greatly outperform standard ML techniques on some problems, for example image recognition / text generation
- **why not use neural nets?**
  - usually worse than standard ML on standard problems, and harder to implement.
  - Computational constraints: Recent models like OpenAI's GPT-3 would take ETH Deep Learning Cluster 18 months to train.

“Neural Networks”, “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.

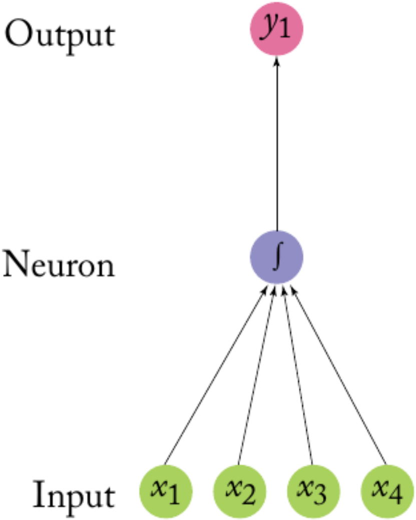
“Neural Networks”, “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.
- “**Networks**”:
  - NNs are not “networks” as that is understood in mathematical network theory or social science.

“Neural Networks”, “Deep Learning”

- “**Neural**”:
  - NN’s do not work like the brain – such metaphors are misleading.
- “**Networks**”:
  - NNs are not “networks” as that is understood in mathematical network theory or social science.
- “**Deep**” Learning:
  - does not speak to profundity or effectiveness.
  - a banal origin, and a source of hype.

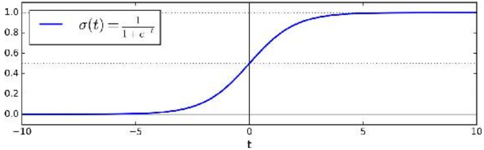
A "Neuron"



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" to the sum
  - (e.g., the $\int$ shape indicates a sigmoid transformation)
- passes the output.

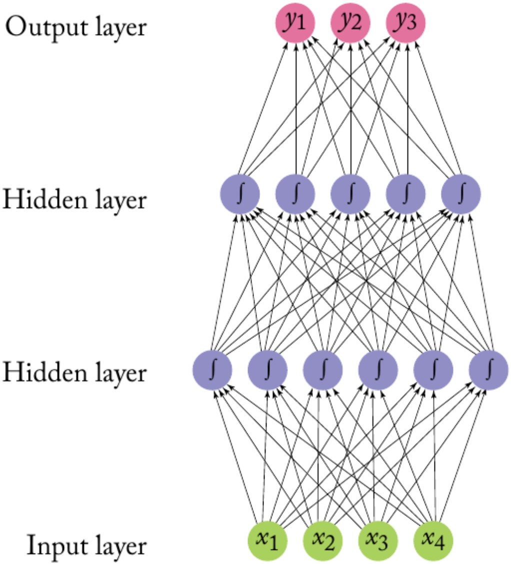
"Neuron" = Logistic Regression

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- applies dot product to vector of numerical inputs:
  - multiplies each input by a learned weight (parameter or coefficient)
  - sums these products
- applies a non-linear "activation function" (sigmoid) to the sum
- passes the output.

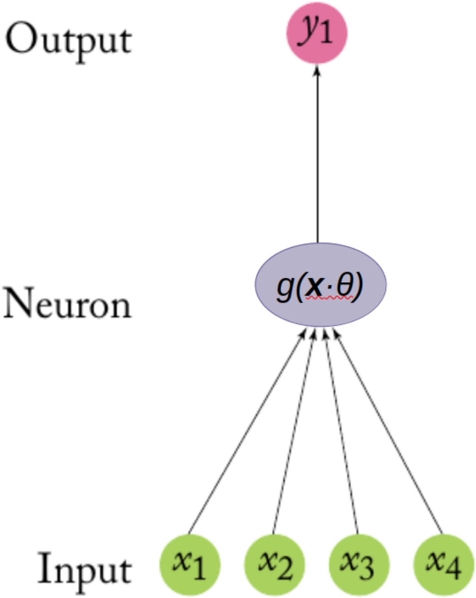
Feed-Forward Neural Network (FFN)



- A feed-forward network (also called a multi-layer perceptron or sequential model) stacks neurons horizontally and vertically.
- alternatively, think of it as a stacked ensemble of logistic regression models.
- this vertical stacking is the “deep” in “deep learning”!

- FFN's are composed of "Dense" layers – means that all neurons are connected.
- FFN with a single hidden layer, with sigmoid activation, can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$, and any mapping from one finite discrete space to another finite discrete space (Hornik et al 1989, Cybenko 1989).
  - But NN would have to be exponentially large in some cases (Telgarsky 2016) .

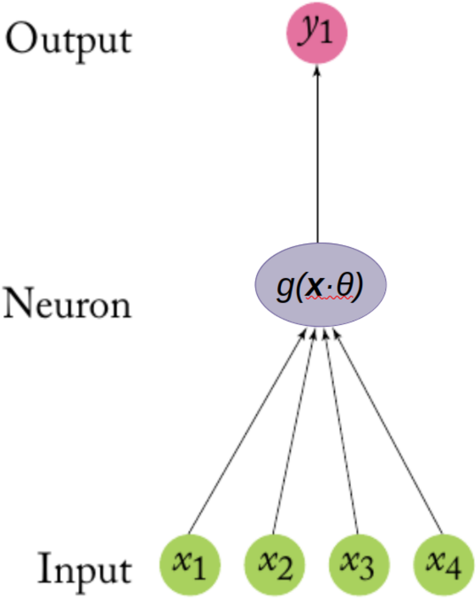
Activation functions $g(\boldsymbol{x} \cdot \theta)$



Previously we had

$$g(\boldsymbol{x} \cdot \theta)=\text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1+\exp(-\boldsymbol{x} \cdot \theta)}$$

Activation functions $g(\boldsymbol{x} \cdot \theta)$

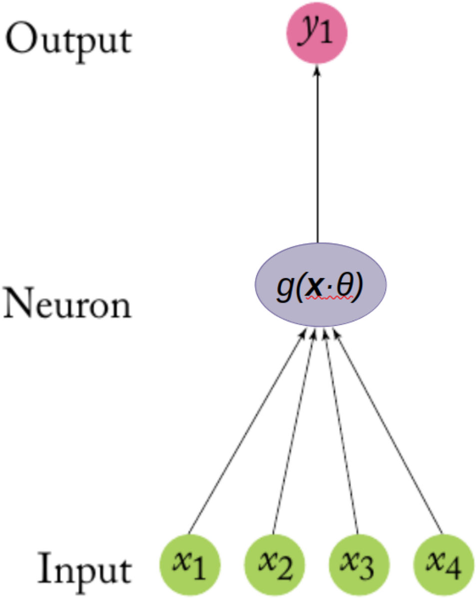


Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

Activation functions $g(\boldsymbol{x} \cdot \theta)$



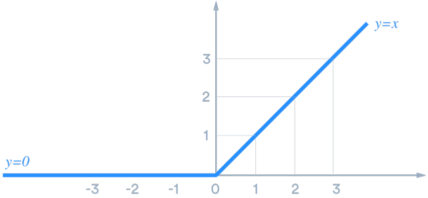
Previously we had

$$g(\boldsymbol{x} \cdot \theta) = \mathrm{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$

It turns out that sigmoid does not work well in hidden layers, mainly because gradient is flat except around zero.

ReLU (rectified linear unit) function:

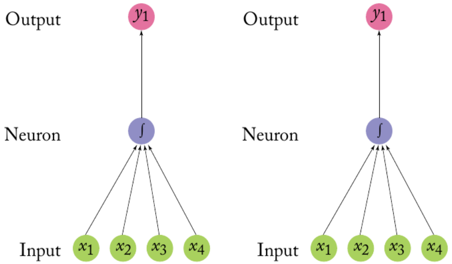
$$g(\boldsymbol{x} \cdot \theta) = \mathrm{ReLU}(\boldsymbol{x} \cdot \theta) = \max\{0, \boldsymbol{x} \cdot \theta\}$$



L2-regularized Logistic Regression in Keras

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(input_dim=num_features,
                units=2,  # output dimension
                activation='sigmoid',
                kernel_regularizer='l2',
                ))
```

```
print(model.summary())
 Model: "sequential_3"
 _________________________________________________________________
 Layer (type)                 Output Shape              Param #
 =================================================================
 dense_3 (Dense)              (None, 2)                 10
 =================================================================
 Total params: 10
 Trainable params: 10
 Non-trainable params: 0
```



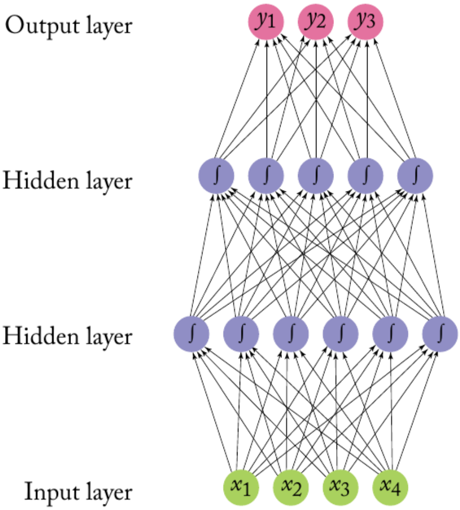
In this example, keras learns 10 parameters:

- coefficients on four predictors, plus a constant
- for each of two outcome classes

FFN in Keras

```
model = Sequential(name='FFN1')

# first hidden layer
model.add(Dense(6,  # neurons in layer
                input_dim=4,
                activation='relu'))

# second hidden layer
model.add(Dense(5,  # neurons in layer
                activation='relu'))

# output layer
model.add(Dense(3,  # output classes
                activation='softmax'))

print(model.summary())
```

```
Model: "FFN1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 6)                 30
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 35
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 18
=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
```



Outline

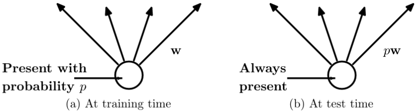
Early stopping

```python
from tensorflow.keras.callbacks import EarlyStopping
earlystop = EarlyStopping(monitor='loss', patience=3)
model.fit(x_train, y_train,
          epochs=100,
          validation_data=(x_val, y_val)),
          callbacks=[earlystop])
```

As done with xgboost, a standard regularization approach for NNs is **early stopping:**

- Split data into three sets: training, validation, and test.
- stop training when validation-set loss stops improving
- evaluate model in test set.

Dropout



Source: Srivastava et al, JMLR 2014

- add after dense layers:

```
from tensorflow.keras.layers import Dropout
model.add(Dropout(.5))
```

An elegant regularization technique:

- at every training step, every neuron has some probability (typically $p = 0.5$) of being temporarily dropped out, so that it will be ignored at this step.
- at test time, neurons dont get dropped anymore but coefficients are down-weighted by p.

Why Dropout Works

- Approximates an ensemble of N models (where N is the number of neurons).

Why Dropout Works

- Approximates an ensemble of N models (where N is the number of neurons).
- Neurons cannot co-adapt with neighboring neurons and must be independently useful.
- Layers cannot rely excessively on just a few input neurons; they have to pay attention to all input neurons.
  - Makes the model less sensitive to slight changes in the inputs.

How to choose among so many options?

- the # of layers, # of neurons, regularization, dropout, etc are all tunable hyperparameters.
  - can pick these with cross-validation as we did previously.
- neural nets have many many dimensions for tuning.
  - this is a serious downside of neural nets, compared to the standard scikit-learn models.
- see the Geron book for advice on this point.
  - in general, make a big model (too many layers, too many neurons) and regularize with dropout/early stopping.

Outline

Predicting Mortgage Default with FFNs (Sirignano, Sadhwani, & Giesecke 2018)

- Analyze mortgage risk using data from over 120 million loans for U.S. borrowers, 1995-2014
- Estimate deep learning model to predict loan status changes:
  - current; late; foreclosure
- Predictors:
  - loan variables at origination
  - loan performance variables over time
  - local economic variables

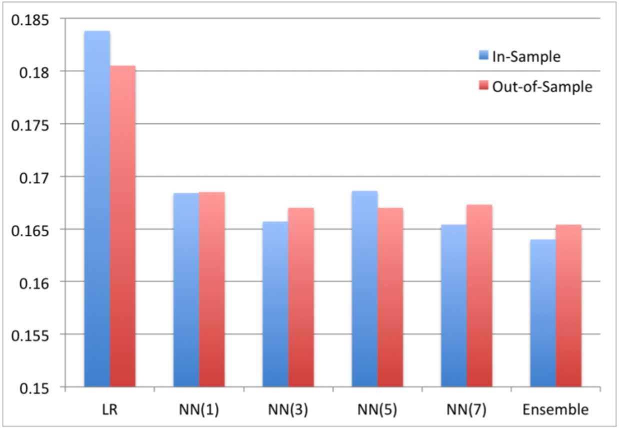
Monthly Transition Matrix (Outcome)

	Current	30	60	90+	Foreclosure
Current	97	1.4	0	0	.001
30 days	34.6	44.6	19	0	.004
60 days	12	16.8	34.5	34	1.6
90+ days	4.1	1.4	2.6	80.2	10
Foreclosure	1.9	.3	.1	6.8	87

Modeling

- Dataset is 350 billion loan-month transitions.
  - 294 predictors.
- Feed-forward network:
  - cross-validation picks 5 layers, $\sim$200 neurons each, ReLU activation.
  - compare to logistic regression baseline

In- and out-of-sample errors vs. network depth



Global variable ranking by "leave-one-out"

Variable	**Test Loss**
State unemployment rate	1.160
Current outstanding balance	.303
Original interest rate	.233
FICO score	.204
Number of times 60dd in last 12 months	.179
Number of times current in last 12 months	.175
Original loan balance	.175
Total days delinquent $\geq$ 160	.171
Lien type $=$ first lien	.171
Original interest rate - national mortgage rate	.170
LTV ratio	.169
Time since origination	.168
Debt-to-income ratio	.168
$\vdots$	$\vdots$

How to avoid machine learning pitfalls (rest of class)

- Summarize and discuss a section from "How to avoid machine learning pitfalls: a guide for academic researchers" (https://arxiv.org/abs/2108.02497)
- Revisit with your group from earlier.
- Pick one of sections 2-6 from the paper
  - pick the section that was newest or most interesting to the group
  - or pick a section randomly
- Instructions:
  - create a google slides presentation, template: https://docs.google.com/presentation/d/1mU8TeuDiKblKt9VdjkyL132sgpxbQTGg3zsasfAID0s/edit?usp=sharing
  - slide 1: summarize the section
  - slide 2: what was new to any in your group?
  - slide 3: for the topic in this section, are there any special considerations for deep learning, relative to classical machine learning?
  - slide 4: what are open questions / issues that could be addressed?
- Post link here: https://padlet.com/eash44/71qkwn5zezyo9ata