Guideline for Machine Learning Projects

Naveen Kumar
Department of Computer Science
University of Delhi - 110007

Acknowledgement

- Andrew Ng: Machine Learning Yearning: Technical Strategy for AI Engineers in the Era of Deep Learning
- Coursera: Andrew Ng's Deep Learning course
- Tom Mitchell: Machine learning
- Mayor, Adrienne, Gods and Robots, Princeton University Press.

The myths gave expressions to the timeless impulse to create artificial life*

- Beings that were made, not born
- Jason and the Argonauts, the bronze robot Talos, the techno-witch Medea, the genius craftsman Daedalus, the fire-bringer Prometheus, and Pandora, the evil bot created by Hephaestus, the god of invention.
- Greek poets Hesiod and Homer (750-650 BCE): Reinforced the notion that imagination is the spirit that unites myth and science.
- Medieval craftsman: Automatic moving Machines

*Mayor, Adrienne. Gods and Robots, Princeton University Press. Kindle Edition.

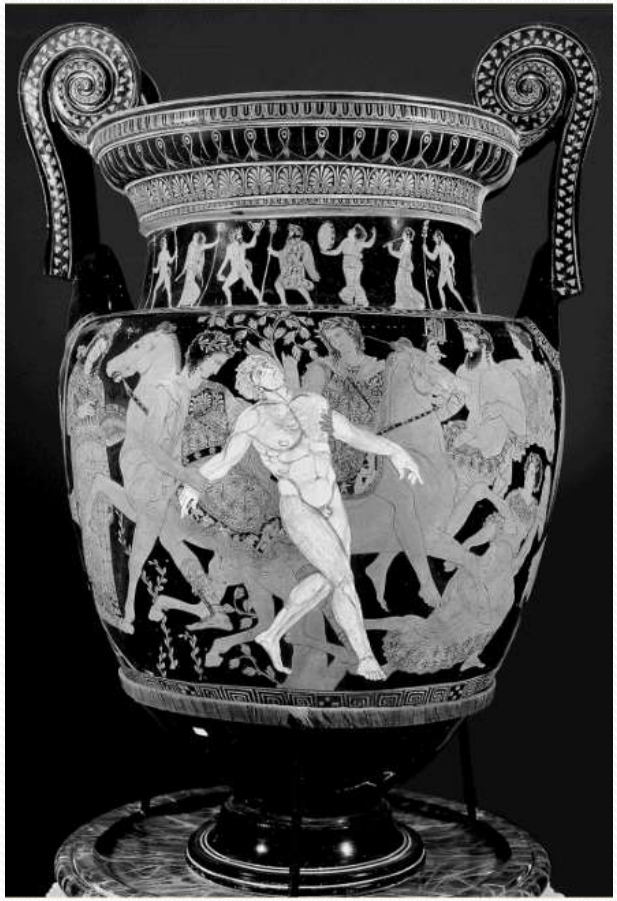


FIG. 1.2. Talos hurling stones on coins of Phaistos, Crete. Left, silver stater, fourth century BC (reverse shows a bull). Theodora Wilbur Fund in memory of Zoe Wilbur, 65.1291. Right, Talos in profile, bronze coin, third century BC (reverse shows the Golden Hound). Gift of Mr. and Mrs. Cornelius C. Vermeule III, 1998.616. Photographs © 2018 Museum of Fine Arts, Boston.

Machine Learning/Deep learning

- 1943, McCulloch: Model for neurons in brain
- 1959, Widrow & Hoff of Stanford: ADALINE & MADALINE
- 1975, Werbos backpropagation algorithm
- 1982, Kohonen: Self Organizing Maps
- 1986, Rumelhnrt, Hinton, & Williams: Learning representations by backpropagation
- 1986, Rumelhart and McClelland parallel distributed processing:
- Maxpooling, Convolution networks, Sequential Networks

Courses in Machine Learning/Deep Learning

- Machine Learning https://www.coursera.org/learn/machine-learning
- Master of Machine Learning & Data Science https://www.coursera.org/degrees/msc-machine-learning-imperial
- Neural Networks and Deep Learning (https://www.coursera.org/learn/neural-networks-deep-learning)
- Deep Learning Specialization (https://www.coursera.org/specializations/deep-learning)
- Deep Learning for Business (https://www.coursera.org/learn/deep-learning-business)
- Deep Learning (https://in.udacity.com/course/deep-learning--ud730)
- Deep Learning A-Z™: Hands-On Artificial Neural Networks (**https://www.udemy.com/deeplearning/**)

Coursera.org

Machine Learning

Stanford University

COURSE

4.9 (149,002) | 3.6M students

Mixed

Machine Learning

University of Washington

SPECIALIZATION

4.6 (13,913) | 380K students

Intermediate

deeplearning.ai

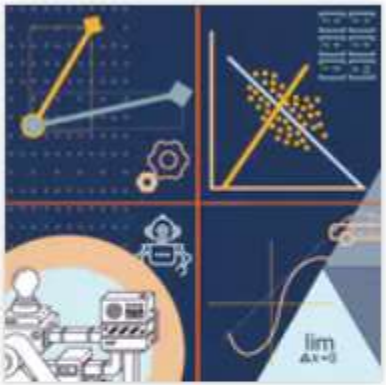
Deep Learning

DeepLearning.AI

SPECIALIZATION

4.8 (110,894) | 920K students

Intermediate



Mathematics for Machine Learning

Imperial College London

SPECIALIZATION

4.6 (9,542) | 230K students

Beginner

Machine Learning for All

University of London

COURSE

4.7 (2,041) | 68K students

Beginner

Ethics in the Age of AI

LearnQuest

SPECIALIZATION

4.7 (11) | 1.8K students

Beginner

Applied Data Science with Python

University of Michigan

SPECIALIZATION

4.5 (26,916) | 660K students

Intermediate

Advanced Machine Learning

National Research University Higher School of Economics

SPECIALIZATION

4.5 (3,476) | 290K students

Advanced

Machine Learning with TensorFlow on Google Cloud Platform

Google Cloud

SPECIALIZATION

4.5 (8,068) | 100K students

Intermediate

Tiny Machine Learning (TinyML)

HarvardX

Professional Certificate (3 courses)

Machine Learning and Finance

NYUx

Professional Certificate (2 courses)

Computer Science for Artificial Intelligence

HarvardX

Professional Certificate (2 courses)

Data Science

HarvardX

Professional Certificate (9 courses)

"machine learning" Courses (117 results)

Show (117)

Data Science: Machine Learning

HarvardX

Course

Machine Learning with Python: A Practical Introduction

IBM

Course

Machine Learning with Python: from Linear Models to Deep Learning

MITx

Course

Machine Learning for Data Science and Analytics

ColumbiaX

Course

Data Science and Machine Learning Capstone Project

IBM

Course

Machine Learning

ColumbiaX

Course

Machine Learning Fundamentals

UCSanDiegoX

Course

Machine Learning (aprendizaje automático) con Python: una introdu...

IBM

Course

Data Science: Machine Learning and Predictions

BerkeleyX

Course

Modelos predictivos con Machine Learning

AnahuacX

Course

Amazon SageMaker: Simplifying Machine Learning Application De...

AWS

Course

PyTorch Basics for Machine Learning

IBM

Course

Dynamic Programming: Applications In Machine Learning and Genomics

UCSanDiegoX

Course

大数据机器学习|Big Data Machine Learning

TsinghuaX

Course

Introduction to Machine Learning

ITMOx

Course

Predictive Analytics using Machine Learning

EdinburghX

Course

Advanced Machine Learning

ITMOx

Course

Classical Machine Learning for Financial Engineering

NYUx

Course

Quantum Machine Learning

University_of_TorontoX

Course

Machine Learning

GTx

Course

Turing Awards

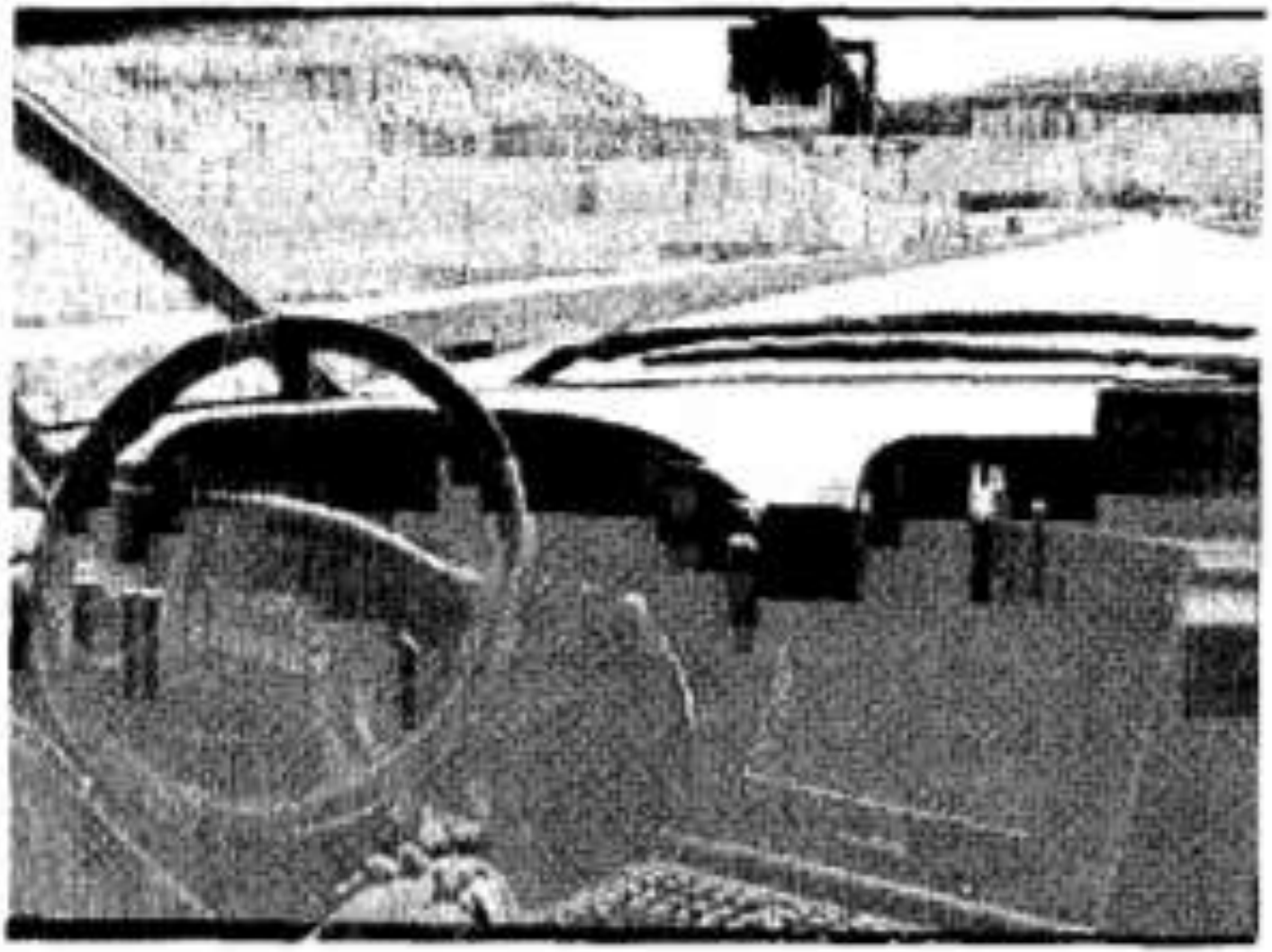
ACM Turing Award Conferred on Pioneers in ML/DL

Yoshua Bengio: University of Montreal

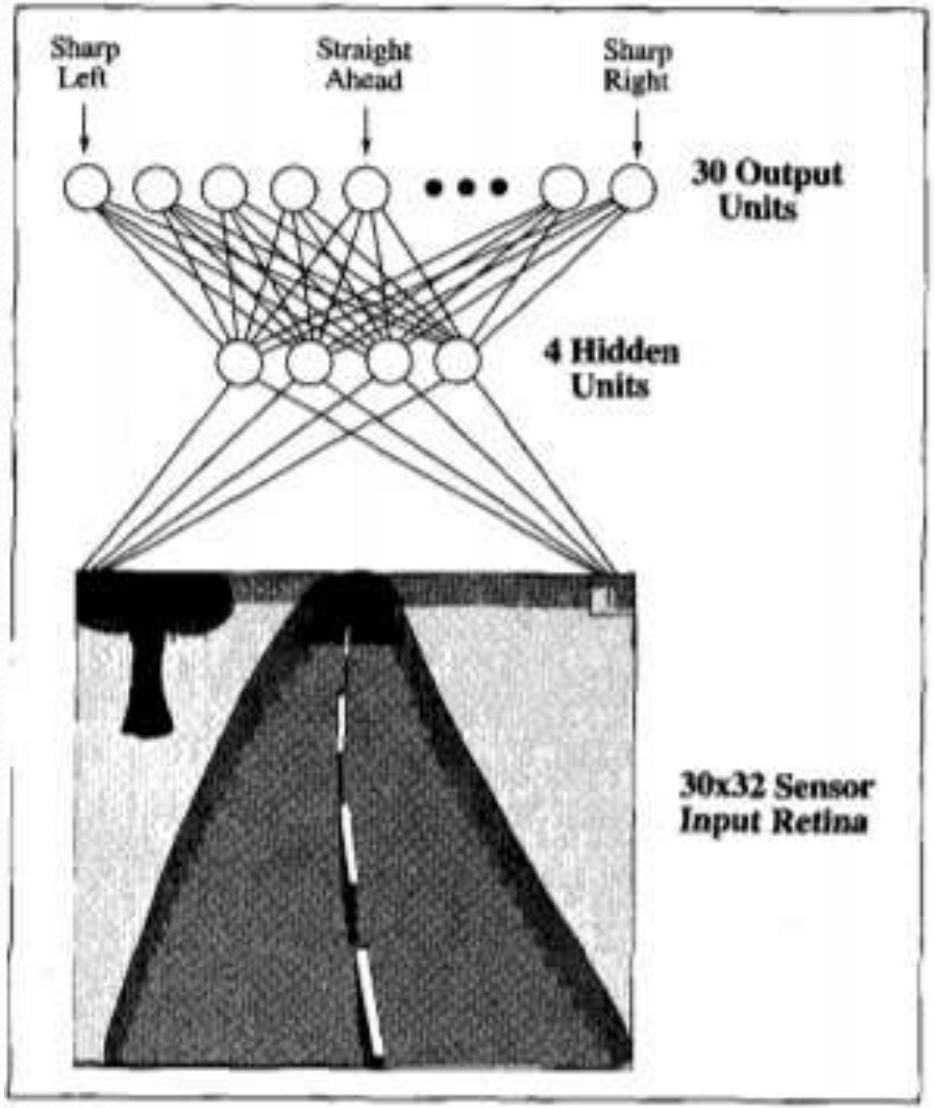
Geoffrey Hinton: Univ. of Toronto

Yann LeCun: CIMS

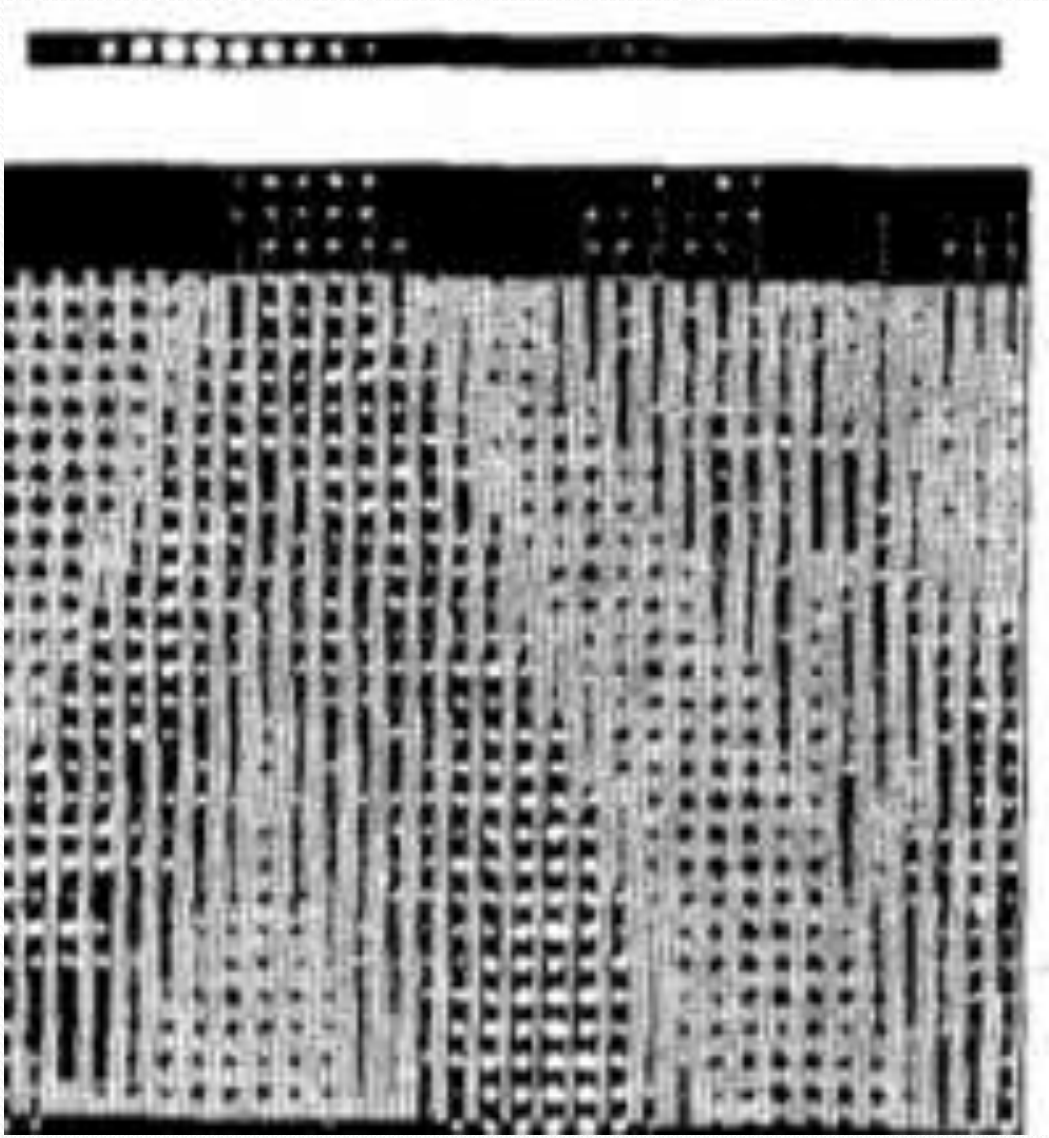
ALVINN



ALVINN (Pomerleau: 1989)

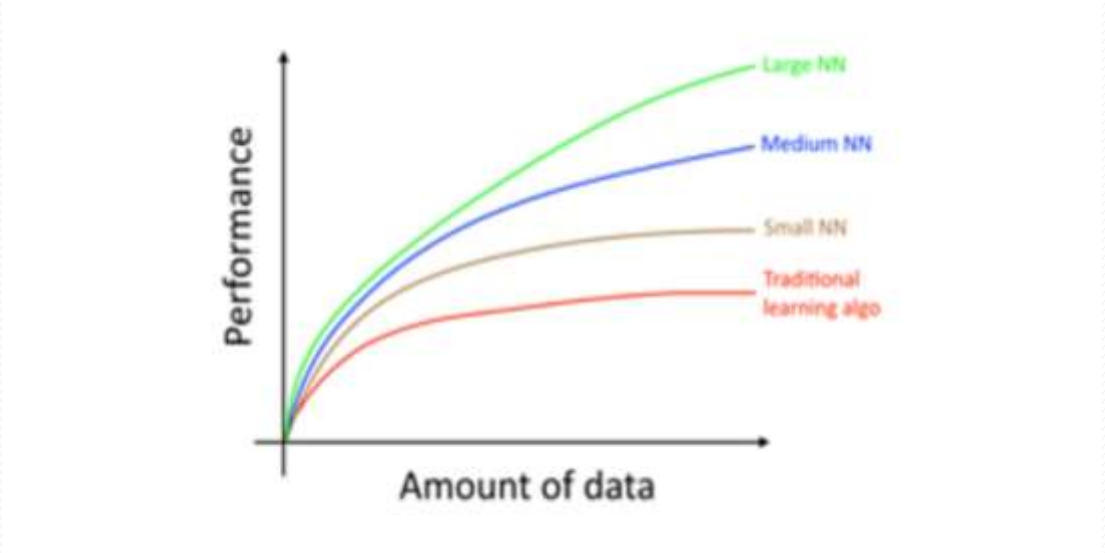


ALVINN

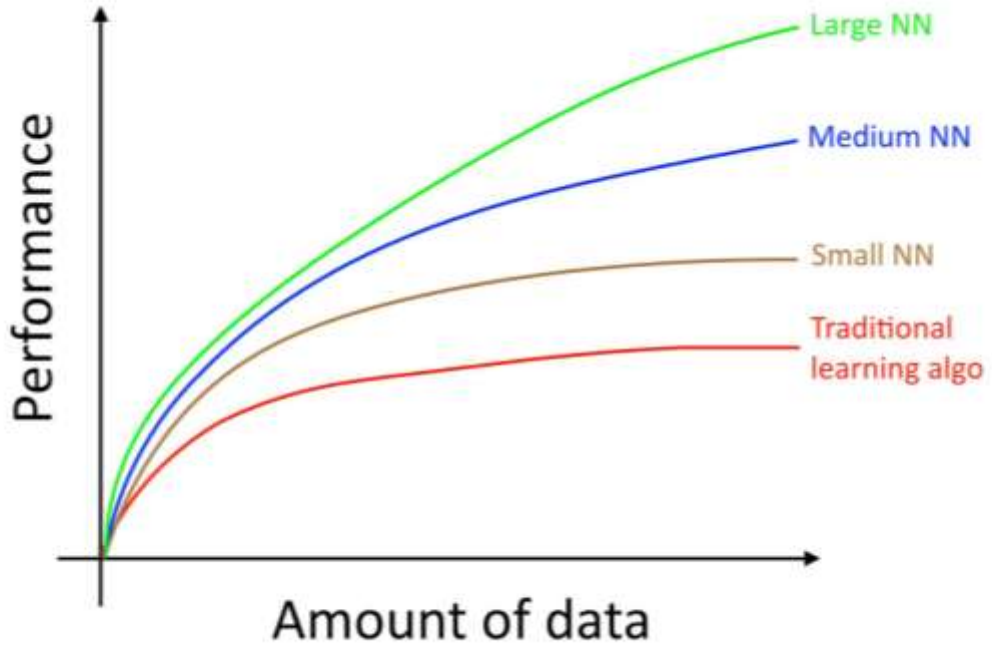


Scale Drives ML/DL

- Driving Factors:
  - Scale of Data –Lot of data with complex problem to solve
  - Size of Network – A lot of hidden layers with a lot of parameters and a lot of connections



Scale Impacts Learning



Setting up Data Sets

- Training set : Initial learning
- Development (dev) set/ hold out cross-validation set
  - Used for Parameter tuning
- Test set: Evaluate performance
- Data Distribution
  - Dev and Test datasets must come from the same distributions
  - Choose dev and test sets to reflect data you expect to get in the future and want to do well on

Setting up Data Sets: Mobile App Cat vs Non-cat

- Not yet launched your mobile app
- Do not have any users yet
- Unable to get data that accurately reflects what you have to do well on in the future

Setting up Data Sets: Think Hard

- Try to approximate real-life scenario
- Ask your friends to take mobile phone pictures of cats and send them to you.
- Once your app is launched, update your dev/test sets using actual user data

Classifier Accuracy

- Classifier A: 90% accuracy
- Classifier B: 90.1% accuracy
- Do we need to distinguish between the two classifiers

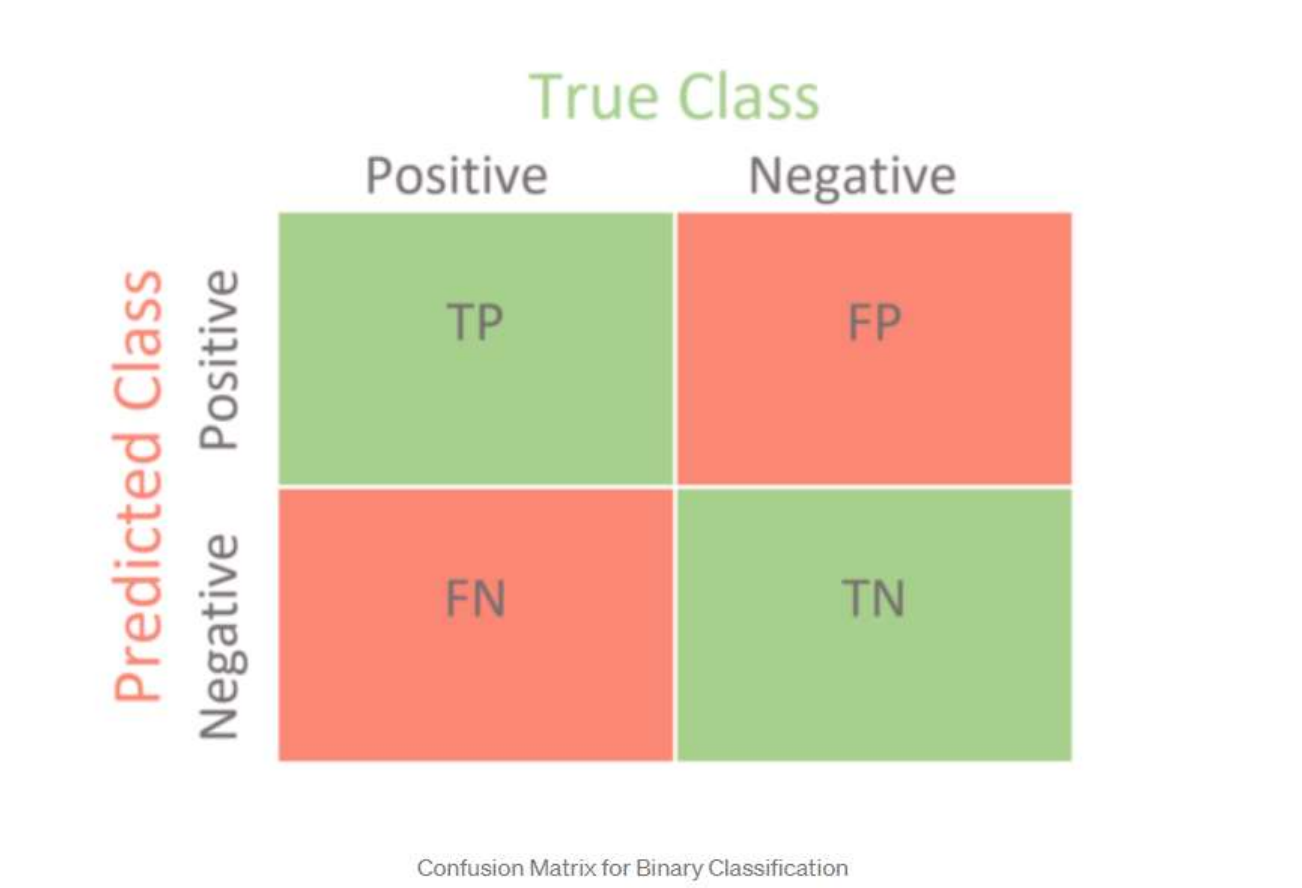
Size of dev/test sets

- Classifier A: 90% accuracy
- Classifier B: 90.1% accuracy
- Do we need to distinguish between the two classifiers
  - Depends on application
  - Web search, recommendation system: 0.01% improvement matters
- If yes, what what should be Dev set: 100? 1,000? 10,000? More?
- How should training/dev/test datasets be distributed?

Size of dev/test sets

- Classifier A: 90% accuracy
- Classifier B: 90.1% accuracy
- Do we need to distinguish between the two classifiers
  - Depends on application
  - Web search, recommendation system: 0.01% improvement matters
- If yes, what should be Dev set: 100? 1,000? 10,000? More?
- How should training/dev/test datasets be distributed?
  - Historically (Small data sets) : 70%, 20%, 10%
  - Currently (Many applications have M size data sets): 98%, 1%, 1% or 99% 0.5% 0.5%

Evaluating Classifiers



Confusion Matrix for Binary Classification

Source: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

Evaluating Classifiers

- Recall (Sensitivity): TP/(TP+FN)
  - True Positives/All Positives
- Specificity: TN/ (TN+FP)
  - True Negatives/All Negatives
- Precision: TP/(TP + FP)
  - True Positives/Reported Positive
- Accuracy: (TP+ TN)/(TP+FP+TN+FN)

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

Single-number evaluation metric

F1 Score	Precision	Recall	F1 Score
A	95%	90%	92.4%
B	98%	85%	91%

Use some kind of averaging operator

Single-number evaluation metric

- Suppose you are separately tracking the accuracy of your cat classifier in four key markets:
  - (i) US, (ii) China, (iii) India, and (iv) Other.
- This gives four metrics.
- Consider: an average or weighted average

Optimizing and Satisficing Metrics

- Accuracy Vs Running time

Classifier	Accuracy	Running time
A	95%	80ms
B	97%	90ms
C	99%	2,000ms

Mobile App: Cat vs Non-Cat

- Observation: 90% accuracy, app mistakes dogs for cats
- What to do?
  - Get a sample of 100 misclassified dev examples
  - Count what fraction of these are dogs
- Finding: 5% of misclassified images are dogs
- Q: Do we incorporate a tool that works well on dogs
- Best expected?
  - All misclassified dogs marked Non-cat
  - On a sample of 100, there are 10 errors
  - Out of all errors, 5% are dogs classified as cats
  - Improvement: 5% of 10 i.e. 0.5, Accuracy moves up 90% to 90.5%

Cleaning up Mislabelled Dev/Test Set Examples

- Overall accuracy: 90% (10% overall errors)
- Mislabelled data: 6%
- Should we correct the labels in the dev set?

Cleaning up Mislabelled Dev/Test Set Examples

- Overall accuracy: 90% (10% overall errors)
- Mislabelled data: 6%
- Should we correct the labels in the dev set?
- Errors due to mislabelled examples: 0.6 (6% of 10)
- Errors due to other causes: 9.4 (94% of dev set errors)

Cleaning up Mislabelled Dev/Test Set Examples

Overall accuracy: 98% (2% overall errors)

Errors due to mislabelled examples:

0.6% out of overall 2%

% of dev set errors caused by mislabelled examples:

(0.6/2)*100 = 30%

Errors due to other causes: 1.4% (70% of dev set errors)

Q: Should we correct the mislabelled examples?

Cleaning up Mislabelled Dev/Test Set Examples

Overall accuracy: 98% (2% overall errors)

Errors due to mislabelled examples:

0.6% out of overall 2%

% of dev set errors caused by mislabelled examples:

(0.6/2)*100) = 30%)

Errors due to other causes: 1.4% (70% of dev set errors)

Q: Should we correct the mislabelled examples?

Cleaning up Mislabelled Dev/Test Set Examples

Overall accuracy: 98% (2% overall errors)

Errors due to mislabelled examples:

0.6% out of overall 2%

% of dev set errors caused by mislabelled examples:

(0.6/2)*100 = 30%

Errors due to other causes: 1.4% (70% of dev set errors)

Q: Should we correct the mislabelled examples?

Whatever process you apply to fixing dev set labels, remember to apply it to the test set labels too so that dev and test sets continue to be drawn from the same distribution

Cleaning up Mislabelled Dev/Test Set Examples

- What about labels of examples that system correctly classified?
- Do we need to need to examine these?

Cleaning up Mislabelled Dev/Test Set Examples

- What about labels of examples that system correctly classified?
- Do we need to need to examine these?
- Both the original label and the one generated by the learning algorithm could be wrong on an example.
- If you fix only the labels of examples that your system had misclassified, you might introduce bias into your evaluation.

Eyeball Dev set vs Blackbox Dev Set

- Dev set: 5,000 examples
- Error rate: 20% (1,000 examples)
- Size of eyeball dev set

Eyeball Dev set vs Blackbox Dev Set

- Dev set: 5,000 examples
- Error rate: 20% (1,000 examples)
- Size of eyeball dev set
  - ~50 errors: good sense of error sources
  - ~100 errors: very good sense of error sources
- Manually examine: 100 errors (10% of errors)

Eyeball Dev set vs Blackbox Dev Set

- Dev set: 5,000 examples
- Error rate: 20% (1,000 examples)
- Size of eyeball dev set
  - ~50 errors: good sense of error sources
  - ~100 errors: very good sense of error sources
- Manually examine: 100 errors (10% of errors)
- Split Dev Set
  - Eyeball dev set: 500 examples (10% of 5,000)
  - Blackbox dev set: remaining 4,500 examples

How big should the Eyeball and Blackbox dev sets be?

- Error rate: 5%
- Looking for ~100 errors

How big should the Eyeball and Blackbox dev sets be?

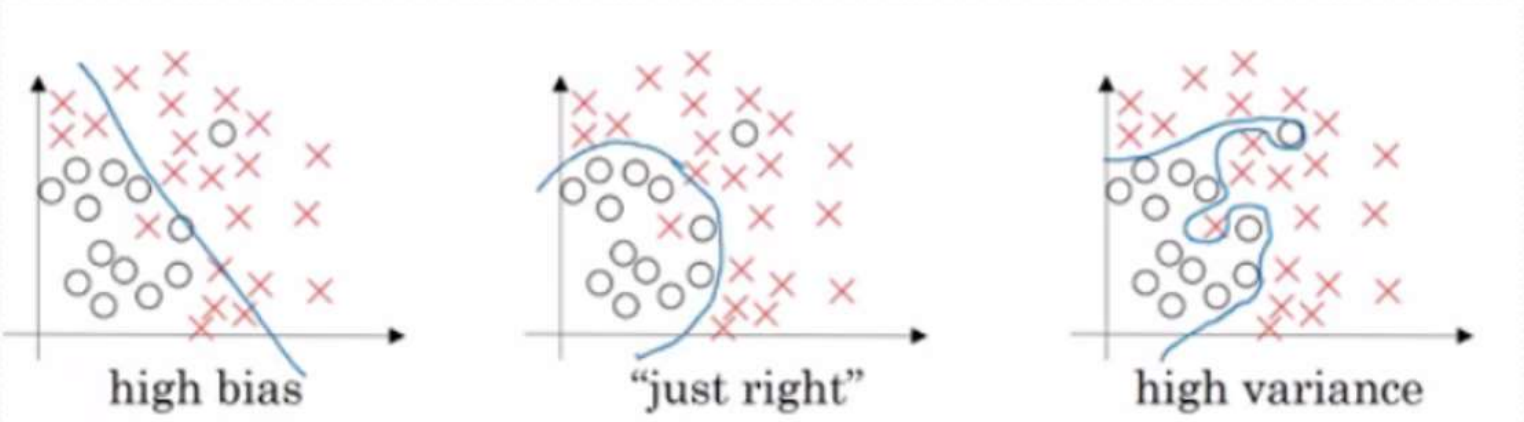
- Error rate: 5%
- Looking for ~100 errors
- N*5/100 = 100
- N = 2,000

How big should the Eyeball and Blackbox dev sets be?

- Error rate: 5%
- Looking for ~100 errors
- N*5/100 = 100
- N = 2,000
- Conclusion:

  The lower your classifier's error rate, the larger your Eyeball dev set needs to be in order to get a large enough set of errors to analyze.

Bias/Variance



Bias and Variance

- Objective: Cat recognizer with 95% accuracy
- Training set error: 15%
- Dev set error: 16%
- Bias: 15%
- Variance: 1%
- What to do?

Bias and Variance

- Objective: Cat recognizer with 95% accuracy
- Training set error: 15%
- Dev set error: 16%
- Bias: 15%
- Variance: 1%
- What to do?
  - Fine-tune hyperparameters?
  - Increase model complexity?
  - Modify input features?
  - Reduce regularization?
  - Increase size of dataset?
  - Do nothing?

Comparing to Optimal Error Rate

- Ideal error rate: Optimal classifier (Human Error)
- Cat classification problem: 0%
- Speech recognition: May be 14% clips too noisy
- Optimal error rate (unavoidable bias or Bayes rate): 14%
- Avoidable bias: 1%
- Bias = unavoidable bias + avoidable bias
- Human performance not good on all tasks

Surpassing Human-level Performance

- Predicting movie ratings
- how long it will take to drive somewhere
- Whether to approve a loan application

Bias and Variance

- Bias: Error rate on your training set
- Variance: Gap between performance on the test set vis a vis training set

	A	B	C	D
Training Error %	1	15	15	0.5
Dev Error%	11	16	30	1

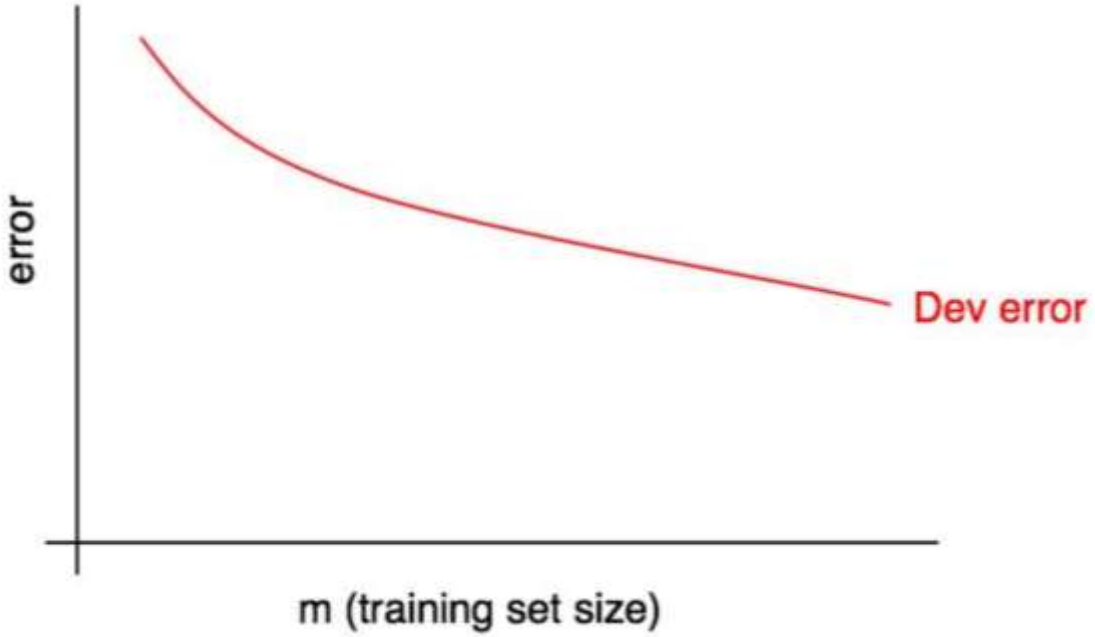
Addressing Bias/Variance

- High avoidable bias
  - Increase the size of your model
  - Try different architecture
  - Modify input features
  - Reduce regularization
  - Typically doesn't work: Add more training data

Addressing Bias/Variance

- High variance
  - Add data to your training set
  - Apply regularization (L2, L1, drop out)
  - Early stopping
  - Feature selection to decrease number of input features significantly
  - Decrease the model size
  - Modify input features based on insights from error analysis
  - Modify model architecture (Prefer simpler architecture)

Learning curves: Diagnosing bias and variance

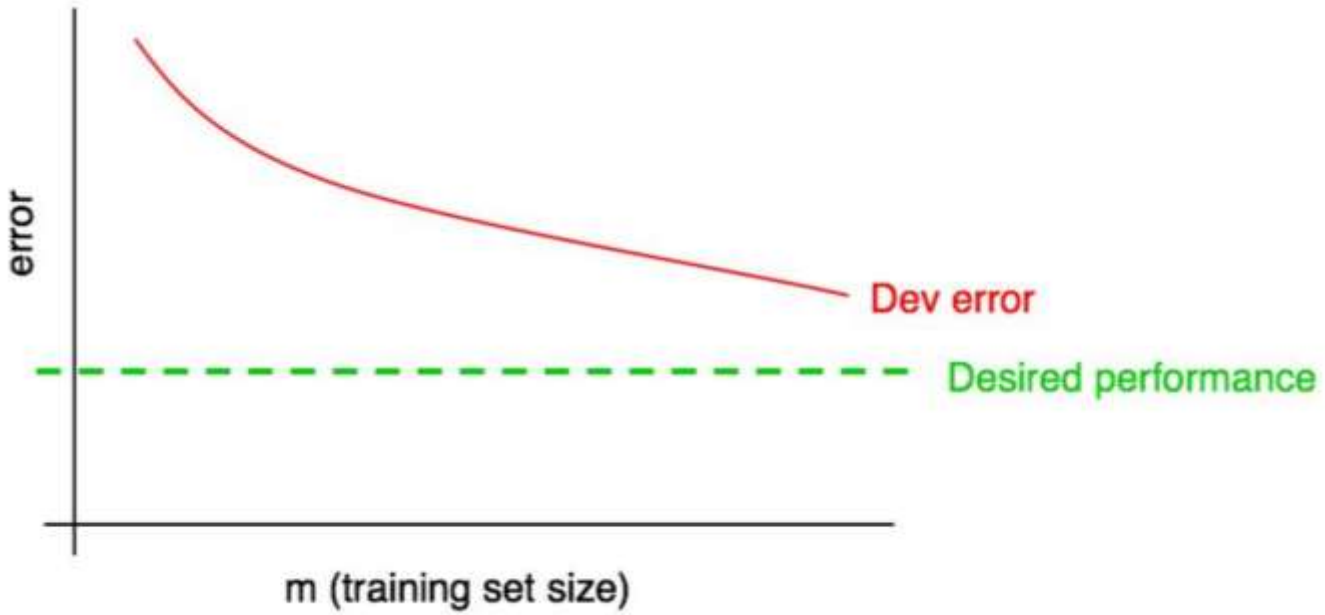


Desired Error rate:

Human Level Performance

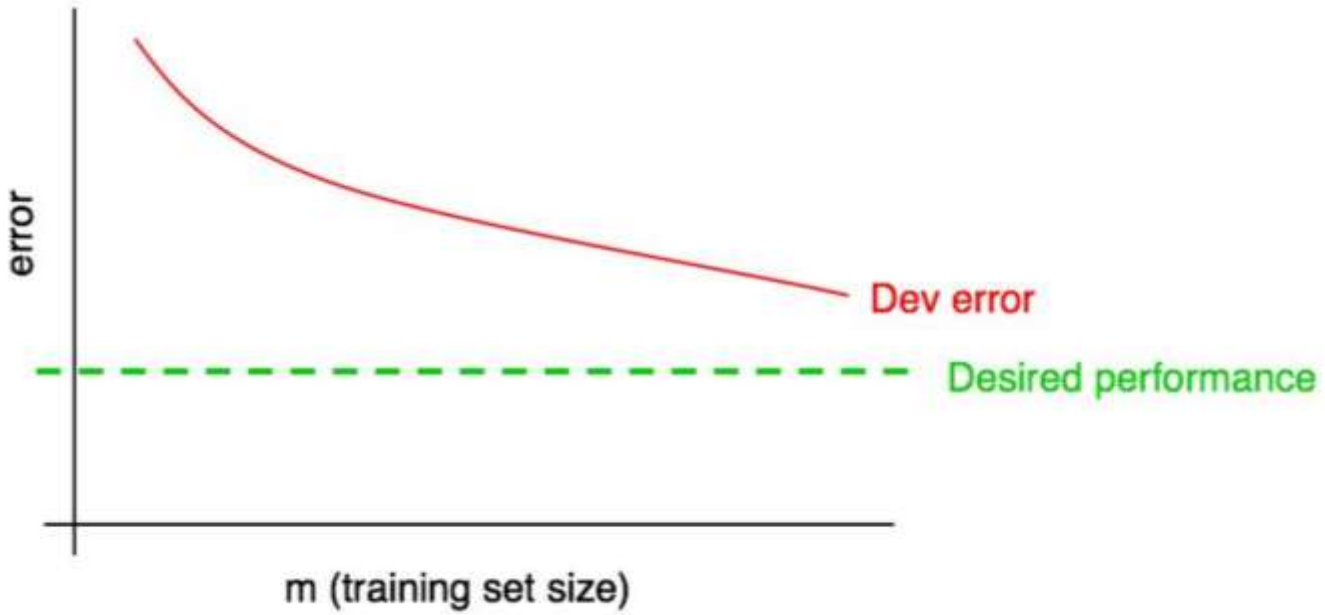
Acceptability to user

Desired Performance on Dev set



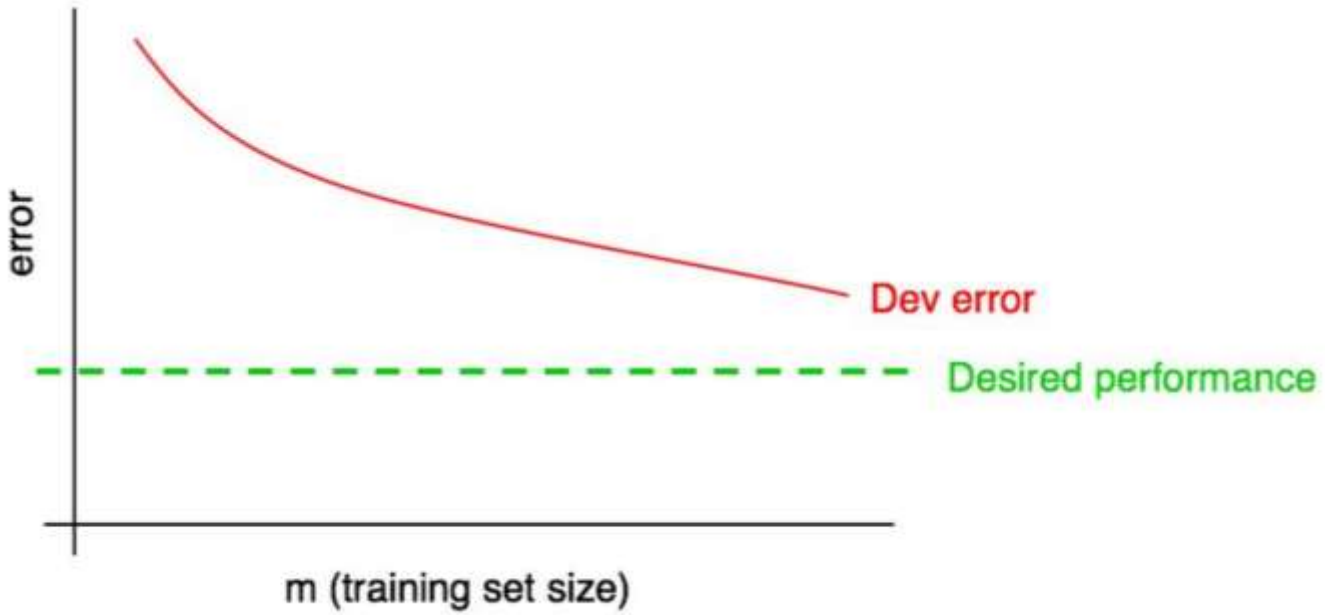
Will increasing the training set size help?

Desired Performance on Dev set



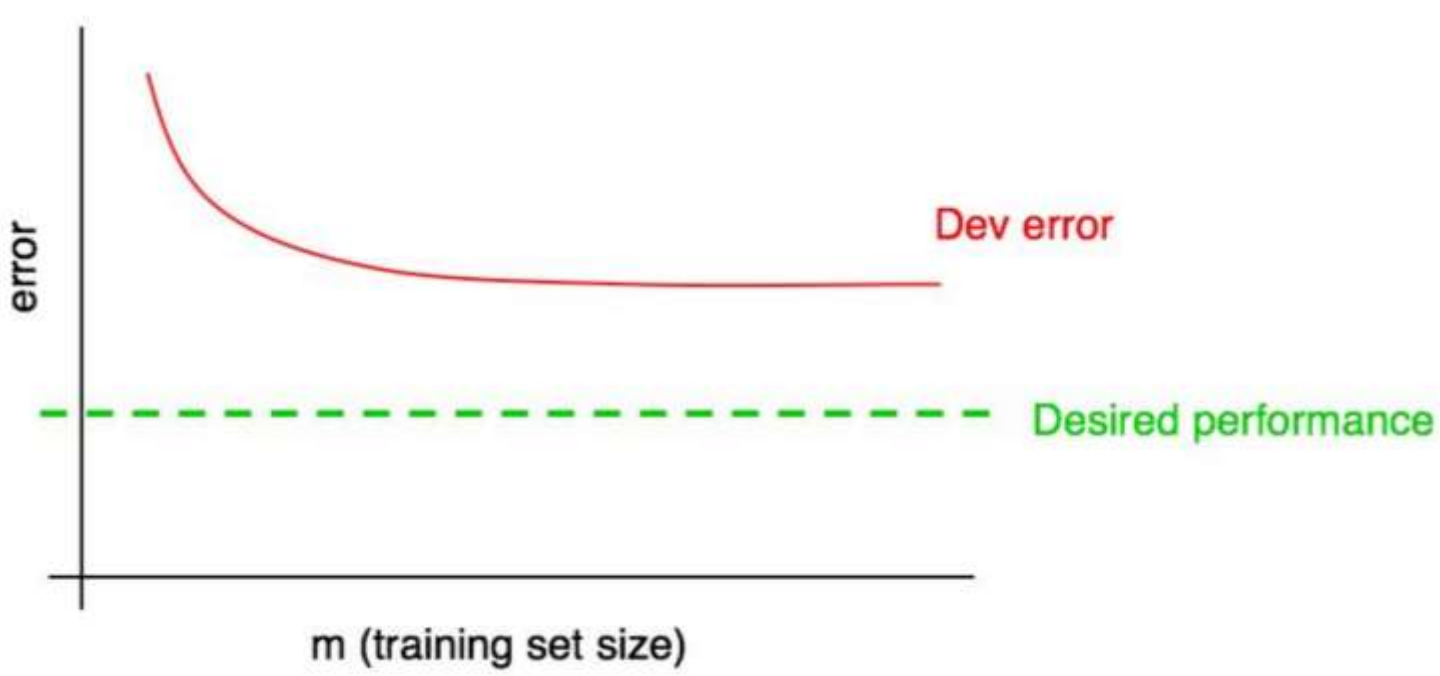
If yes, how much data should we add?

Desired Performance on Dev set



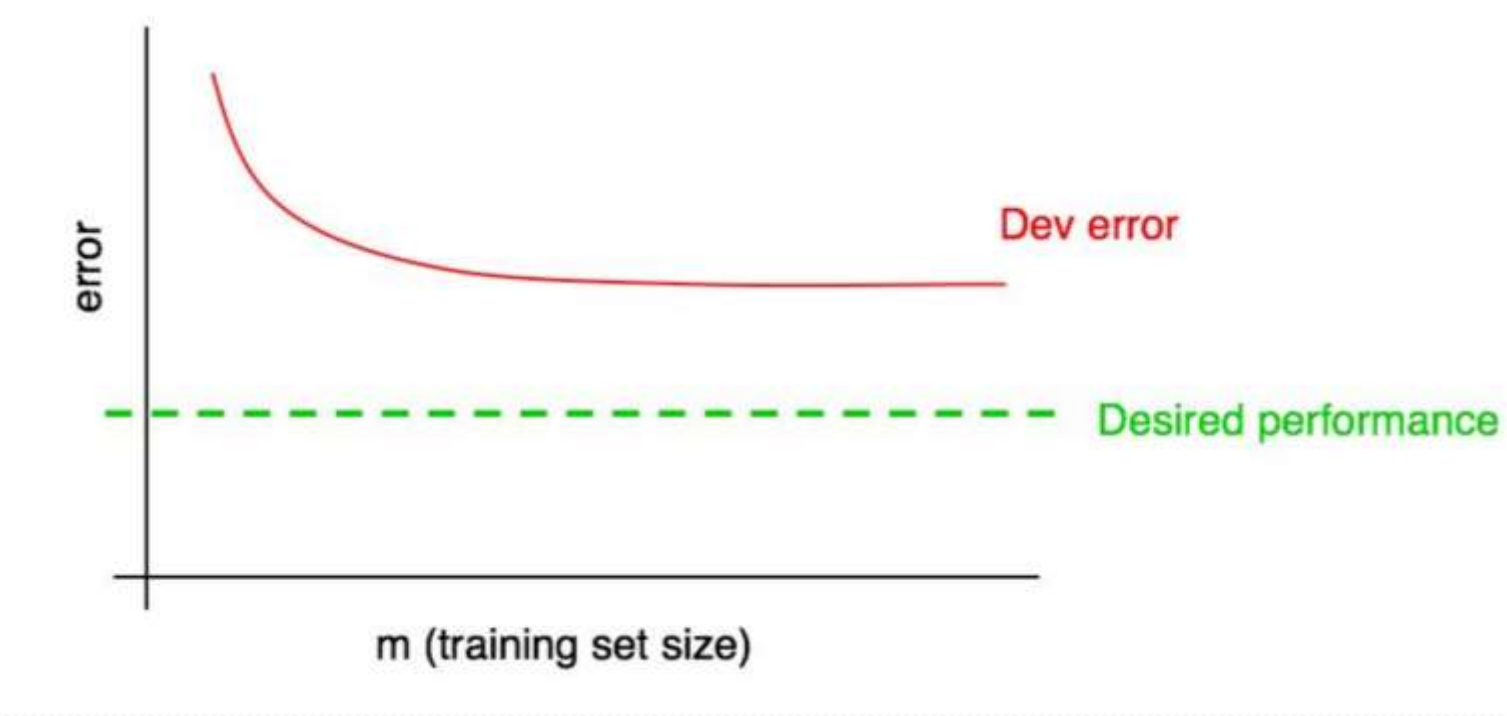
Extrapolate the curve to the desired level.
Doubling the data should do the needful.

Flattened Dev error Curve



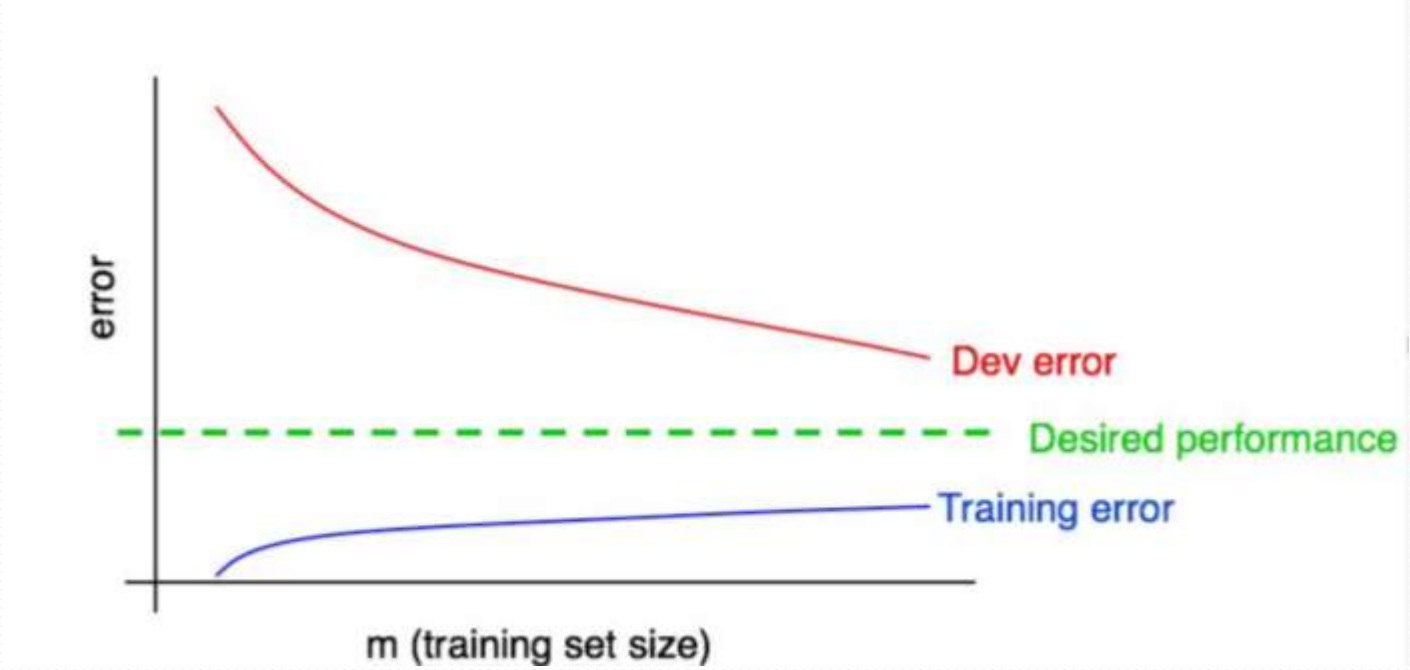
Will increasing the training set size help?

Flattened Dev error Curve



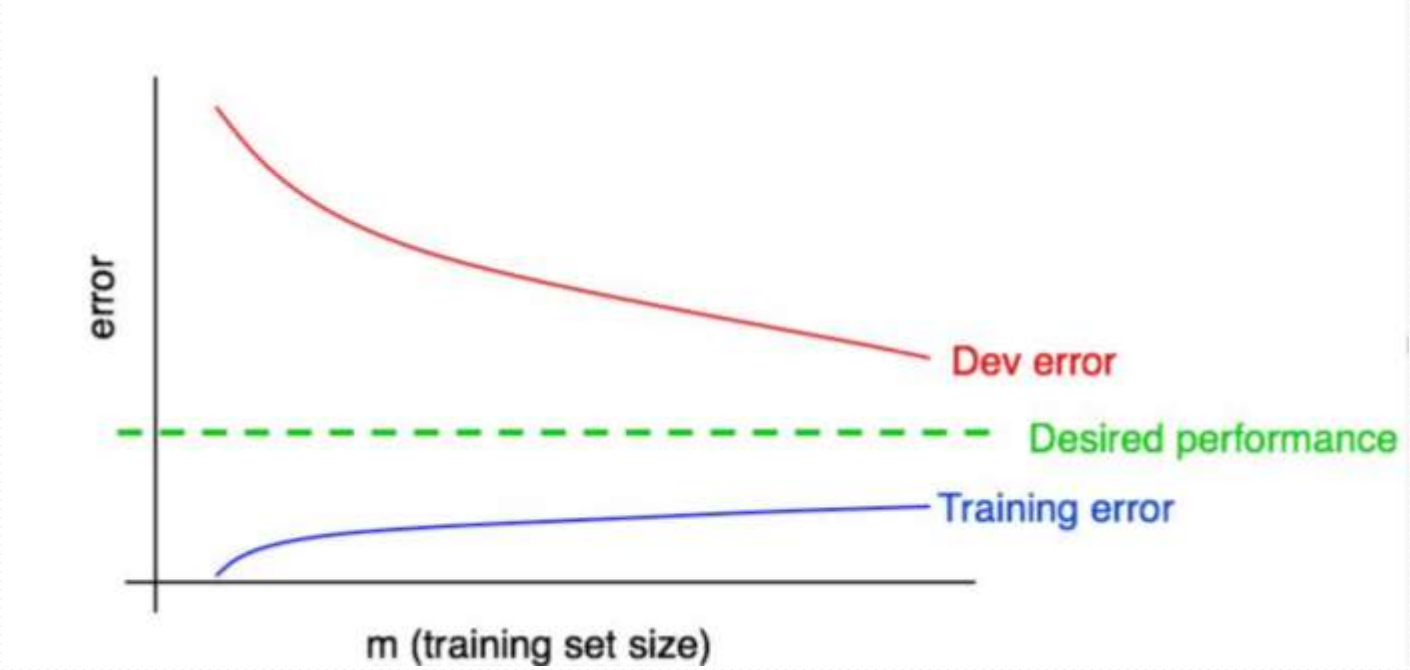
Increasing the training set size won't get you closer to the goal.

Desired Performance: Role of Training Curve



Why might the training error increase fast initially?

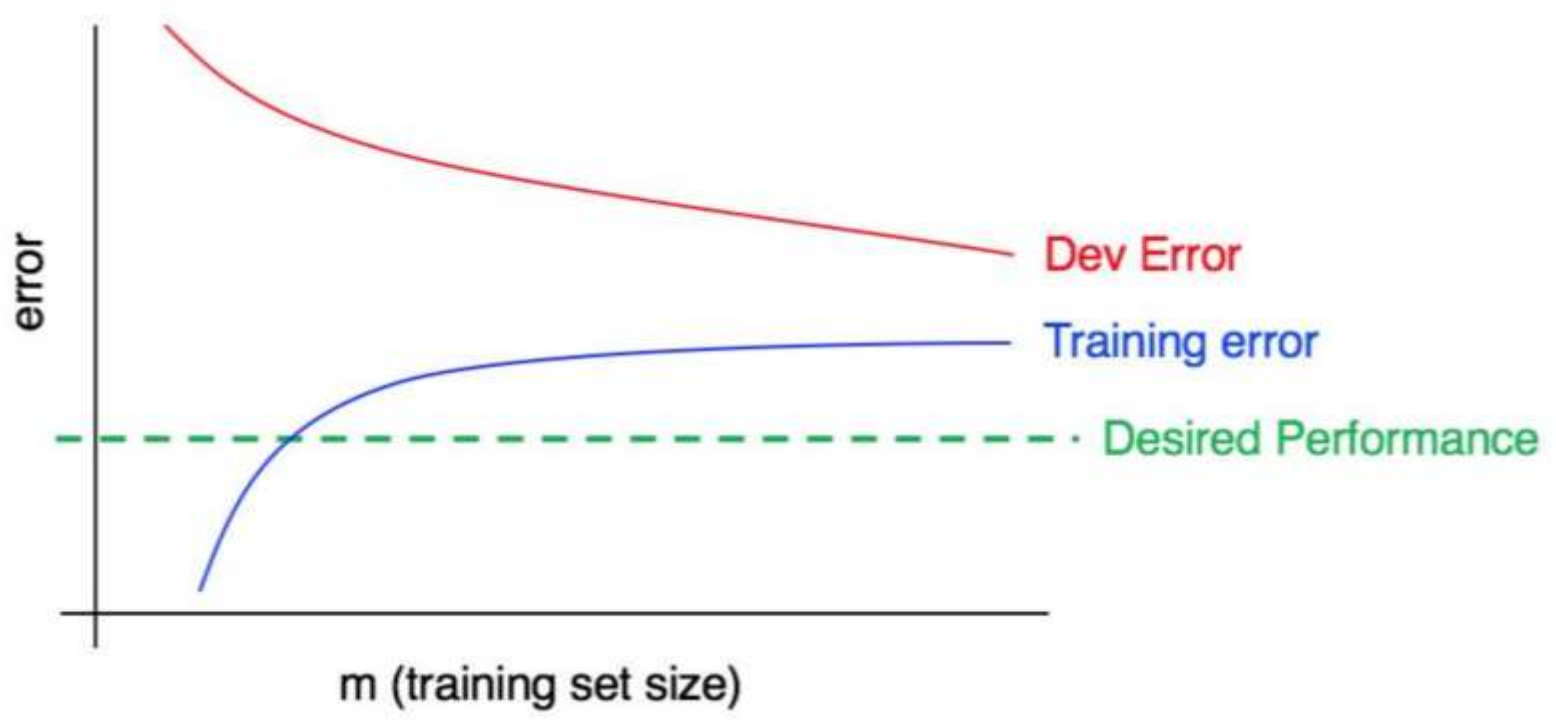
Desired Performance: Role of Training Curve



Too small training error size: memorizes correctly
Training error increases because Neural Network cannot memorize any more

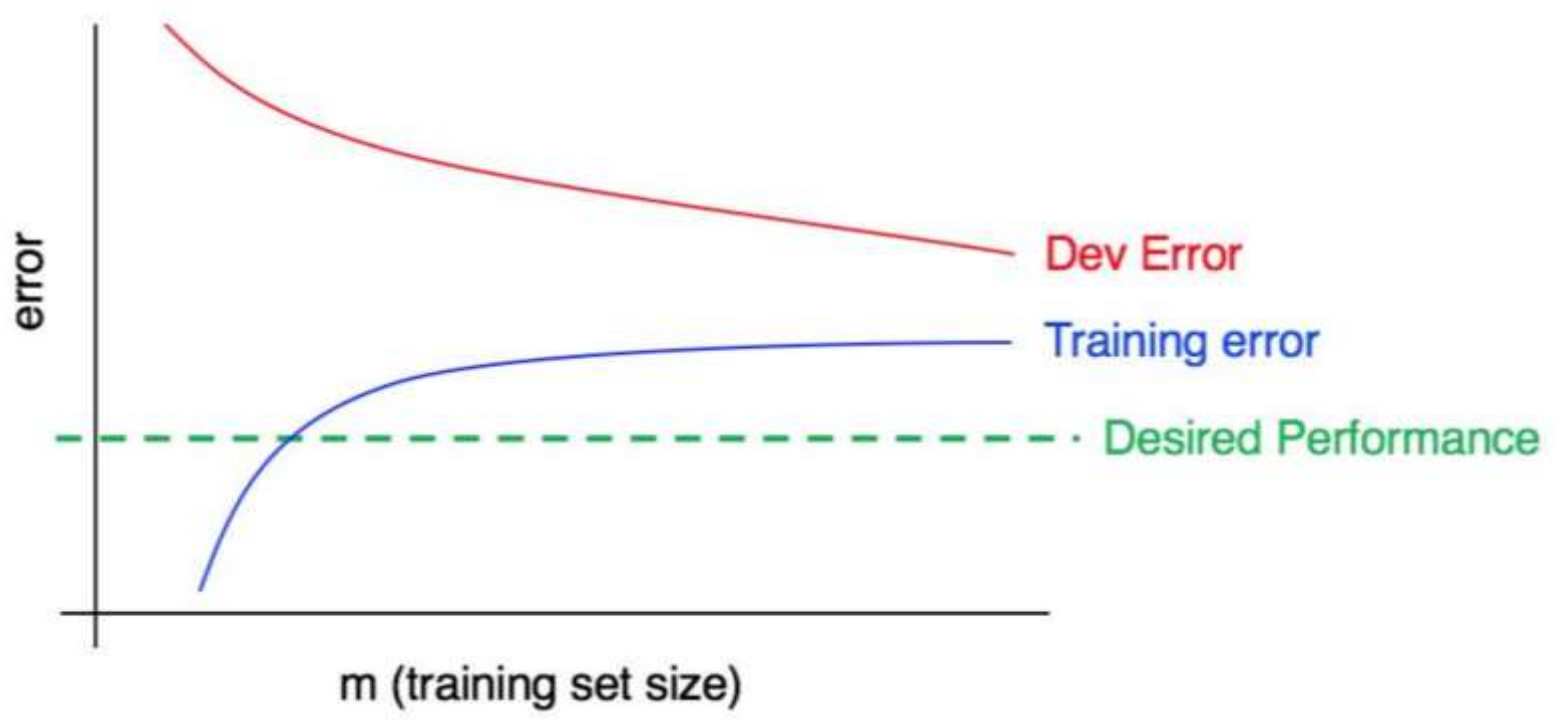

error
Dev Error
Desired performance
Training error
m (training set size)

High Bias & High Variance



Should we increase the size of training data?

High Bias & High Variance



- Adding more training data, training error can only get worse
- There's almost no way that adding more data would allow the red dev error curve to drop down to the desired level of performance when even the training error is higher than the desired level

Regularization

- Objective: To improve generalization of our model on unseen data
- How it works: Constrains our optimization problem to discourage complex models

Regularization 1: Penalizing weights

- Penalize large weights using penalties: constraints on their squared values (L2 penalty) or absolute values (L1 penalty)

- Neural networks have thousands (or millions of parameters)
  - Danger of overfitting

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L})) + \lambda\Omega(\theta)$$

Regularization 1: L1 and L2 regularization

- L2 regularization (most popular)

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L})) + \frac{\lambda}{2} \sum_{l} \|\theta_l\|^2$$

- L1 regularization

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L})) + \frac{\lambda}{2} \sum_{l} \|\theta_l\|$$

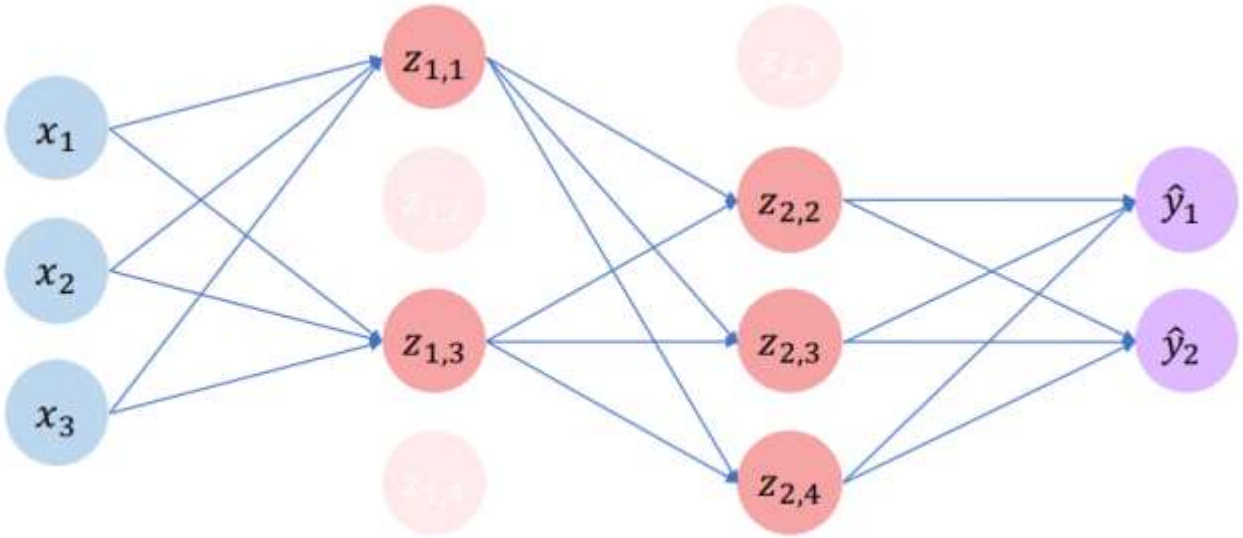
Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Regularization 3: Dropout

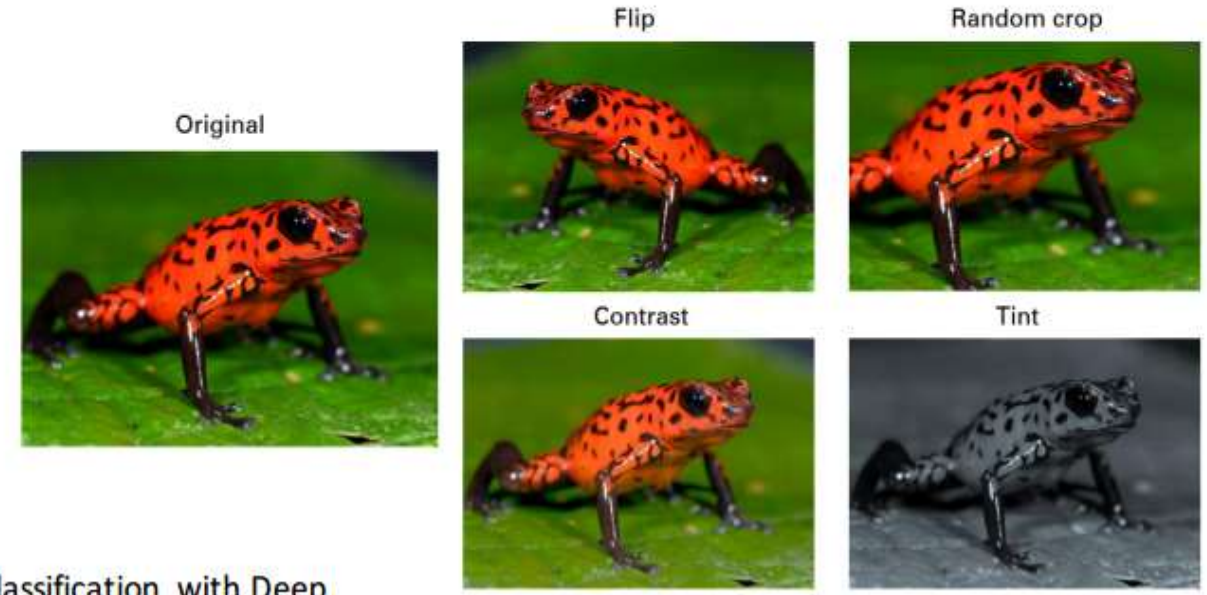
- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.nn.dropout(hiddenLayer, p=0.5)
```



Regularization 4: Data Augmentation

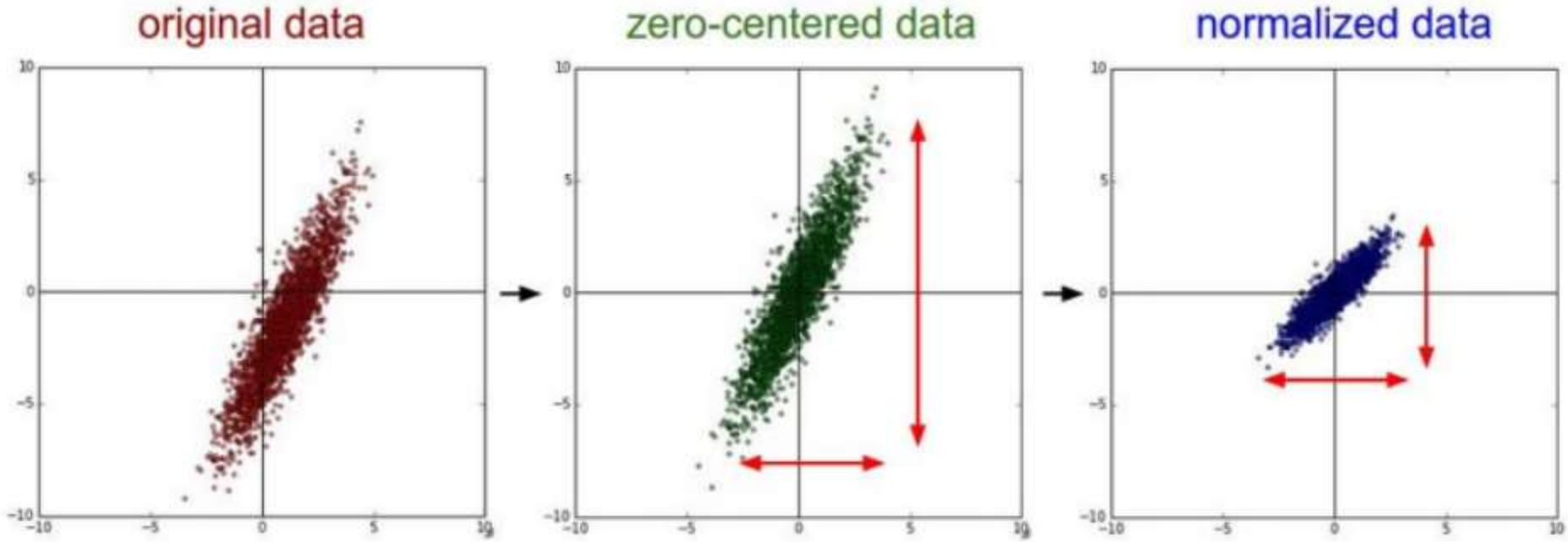
- Adding more data reduces overfitting
- Data collection and labelling is expensive
- Solution: Synthetically increase training dataset



Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, 2012

Normalizing inputs

- Normalized inputs helps for the learning process
- Subtract mean and normalize variances
- Use the same mean and variance to normalize the test (you want them to go through the same transitions)



Training and Testing on Different Distributions may be a Good Idea

- Available data (cat/non-cat):
  - 200,000 images from the Internet
  - 10,000 images uploaded by users
- How to choose training/dev/test data sets?

Is Training and Testing on Different Distributions a Good Idea?

- Available data (cat/non-cat):
  - 200,000 images from the Internet
  - 10,000 images uploaded by users
- Randomly picking training/dev/test data sets from 210,000 examples violates the basic principle:
  - Choose dev and test sets to reflect data you expect to get in the future and want to do well on.

Training and Testing on Different Distributions may be a Good Idea

- Available data (cat/non-cat):
  - 200,000 images from the Internet
  - 10,000 images uploaded by users
- Randomly picking training/dev/test data sets from 210,000 examples violates the basic principle:
  - Choose dev and test sets to reflect data you expect to get in the future and want to do well on.
- Distribute user images:
  - Training: 5,000
  - Dev/test: 5,000

Training and Testing on Different Distributions may be a Good Idea

- Available data (cat/non-cat):
  - 200,000 images from the Internet
  - 10,000 images uploaded by users
- Randomly picking training/dev/test data sets from 210,000 examples violates the basic principle:
  - Choose dev and test sets to reflect data you expect to get in the future and want to do well on.
- Distribute user images:
  - Training: 5,000
  - Dev/test: 5,000

Speech Recognition Example

- 20,000 examples of street addresses
- 500,000 examples of other audio clips with users speaking about other topics.
- Training set: Include: 10,000 examples of street addresses
- Dev/Test Set: include 10,000 examples of street addresses

Is Training and Testing on Different Distributions a Good Idea

- Predicting House Prices in different cities
  - Delhi
  - Udaipur
- Do we merge the two data sets?

Training and Testing on Different Distributions may be a Bad Idea

- Predicting House Prices in different cities
  - Delhi
  - Udaipur
- Do we merge the two data sets?
- Difference between the nature of two problems
  - prediction of house prices
  - prediction of cat/non-cat

Training and Testing on Different Distributions may be a Bad Idea

- Cat/Non-cat: given an input picture x, one can reliably predict the label y indicating whether there is a cat, even without knowing if the image is an internet image
- Price of a house: Without knowing the location, one cannot predict the price of a house

If Training and Testing on Different Distributions is a Bad Idea, can be made a better Idea

- Price of a house: Can we make the situations somewhat similar Cat/Non-cat so that we can use the entire dataset

Training and Testing on Different Distributions may be a Bad Idea

- Price of a house: Can we make the situations somewhat similar Cat/Non-cat so that we can use the entire dataset
- Consider additional attribute: city

How to decide whether to use all your data

- Available data (cat/non-cat):
  - 20,000 images from the Internet
  - 10,000 images uploaded by users
- Early Generation ML Algorithms
  - Handcrafted features
  - Internet Image inclusion: Real risk of performing worse
- Large NNs: Performance likely to improve
  - More examples of what cats do/do not look like
  - NN can apply some of the knowledge acquired from internet images to mobile app images.

How to decide whether to use all your data

- It forces the NN to expend some of its capacity to learn about properties that are specific to internet images
  - Examples: higher resolution, different distributions of how the images are framed, etc.
- These properties may differ greatly from mobile app images, if so
  - It will "use up" some of the representational capacity of the neural network
  - There is less capacity for recognizing data drawn from the distribution of mobile app images.

How to decide whether to use all your data

Sherlock Holmes Quote

- Your brain is like an attic; it only has a finite amount of space.
- “for every addition of knowledge, you forget something that you knew before. It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones.”

Sherlock Holmes: May not be the last word

Sherlock Holmes: May not be the last word

- If you have the computational capacity needed to build a big enough neural network—i.e., a big enough attic—then this is not a serious concern.

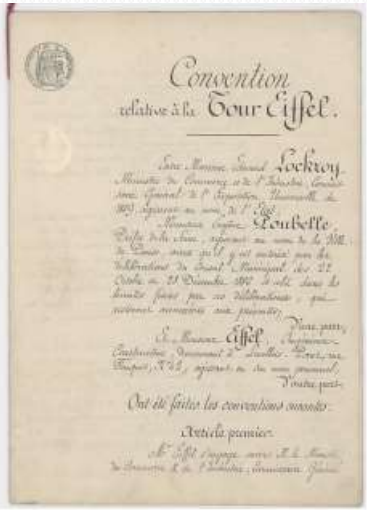
Sherlock Holmes: May not be the last word

- If you have the computational capacity needed to build a big enough neural network—i.e., a big enough attic—then this is not a serious concern.
- You have enough capacity to learn from both internet and from mobile app images, without the two types of data competing for capacity.
- Your algorithm's "brain" is big enough that you don't have to worry about running out of attic space.

Sherlock Holmes: Serious Advice

- If you do not have a big enough neural network (or another highly flexible learning algorithm), then you should pay more attention to your training data matching your dev/test set distribution.
- You may not have sufficient computational resources

Cat/Non-cat: Data Unlikely to contribute to learning



Convention
relative à la Tour Eiffel.

Cat/Non-cat: Data Unlikely to contribute to learning

- These documents don't contain anything resembling a cat.
- They also look completely unlike your dev/test distribution.
- There is no point including this data as negative examples:
  - There is almost nothing NN can learn from this data that it can apply to dev/test set distribution.
  - Including them would waste computation resources and representation capacity of the neural network.

Weighting Data

Internet Images: 200,000 Mobile images: 5,000

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} (h_\theta(x) - y)^2 + \sum_{(x,y)\in \text{InternetImg}} (h_\theta(x) - y)^2$$

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} (h_\theta(x) - y)^2 + \beta \sum_{(x,y)\in \text{InternetImg}} (h_\theta(x) - y)^2$$

β= ?

Generalizing from the training set to the dev set

- Cat/Non-cat: The algorithm has a much higher dev/test set error than you would like.

1. Does not do well on the training set. High (avoidable) bias on the training set distribution.
2. Does well on the training set, but does not generalize well to previously unseen data drawn from the same distribution as the training set. High variance.
3. Generalizes well to new data drawn from the same distribution as the training set, but not to data drawn from the dev/test set distribution.
4. We call this problem data mismatch, since it is because the training set data is a poor match for the dev/test set data.

Generalizing from the training set to the dev set

- Humans achieve near perfect performance on the cat recognition task.
- Your algorithm achieves this:
  - 1% error on the training set
  - 1.5% error on data drawn from the same distribution as the training set that the algorithm has not seen
  - 10% error on the dev set

Generalizing from the training set to the dev set

- Humans achieve near perfect performance on the cat recognition task.
- Your algorithm achieves this:
  - 1% error on the training set
  - 1.5% error on data drawn from the same distribution as the training set that the algorithm has not seen
  - 10% error on the dev set
- Solution: Try to make the training data more similar to the dev/test data.

Generalizing from the training set to the dev set

- To diagnose to what extent an algorithm suffers from each of the problems 1-3
- Do not give the algorithm all the available training data split it into two subsets
  - The actual training set which the algorithm will train on
  - A separate set, which we will call the “Training dev” set
- Training Set and “Training dev” set have same distribution

Generalizing from the training set to the dev set

- Training set. This is the data that the algorithm will learn from (e.g., Internet images + Mobile images). This does not have to be drawn from the same distribution as what we really care about (the dev/test set distribution).
- Training dev set: This data is drawn from the same distribution as the training set (e.g., Internet images + Mobile images). This is usually smaller than the training set; it only needs to be large enough to evaluate and track the progress of our learning algorithm.
- Dev set: This is drawn from the same distribution as the test set, and it reflects the distribution of data that we ultimately care about doing well on. (e.g., mobile images.)
- Test set: This is drawn from the same distribution as the dev set. (e.g., mobile images.)

Identifying Bias, Variance, and Data Mismatch Errors

- Suppose humans achieve almost perfect performance (≈0% error) on the cat detection task
- Optimal error rate: about 0%.
- Suppose you have:
  - 1% error on the training set.
  - 5% error on training dev set.
  - 5% error on the dev set.

Identifying Bias, Variance, and Data Mismatch Errors

- Suppose humans achieve almost perfect performance (≈0% error) on the cat detection task
- Optimal error rate: about 0%.
- Suppose you have:
  - 1% error on the training set.
  - 5% error on training dev set.
  - 5% error on the dev set.
- High Variance

Identifying Bias, Variance, and Data Mismatch Errors

- Now, suppose your algorithm achieves:
  - 10% error on the training set.
  - 11% error on training dev set.
  - 12% error on the dev set.

Identifying Bias, Variance, and Data Mismatch Errors

- Suppose your algorithm achieves:
  - 10% error on the training set.
  - 11% error on training dev set.
  - 12% error on the dev set.
- High avoidable bias
- Suppose your algorithm achieves:
  - 10% error on the training set.
  - 11% error on training dev set.
  - 20% error on the dev set.

Identifying Bias, Variance, and Data Mismatch Errors

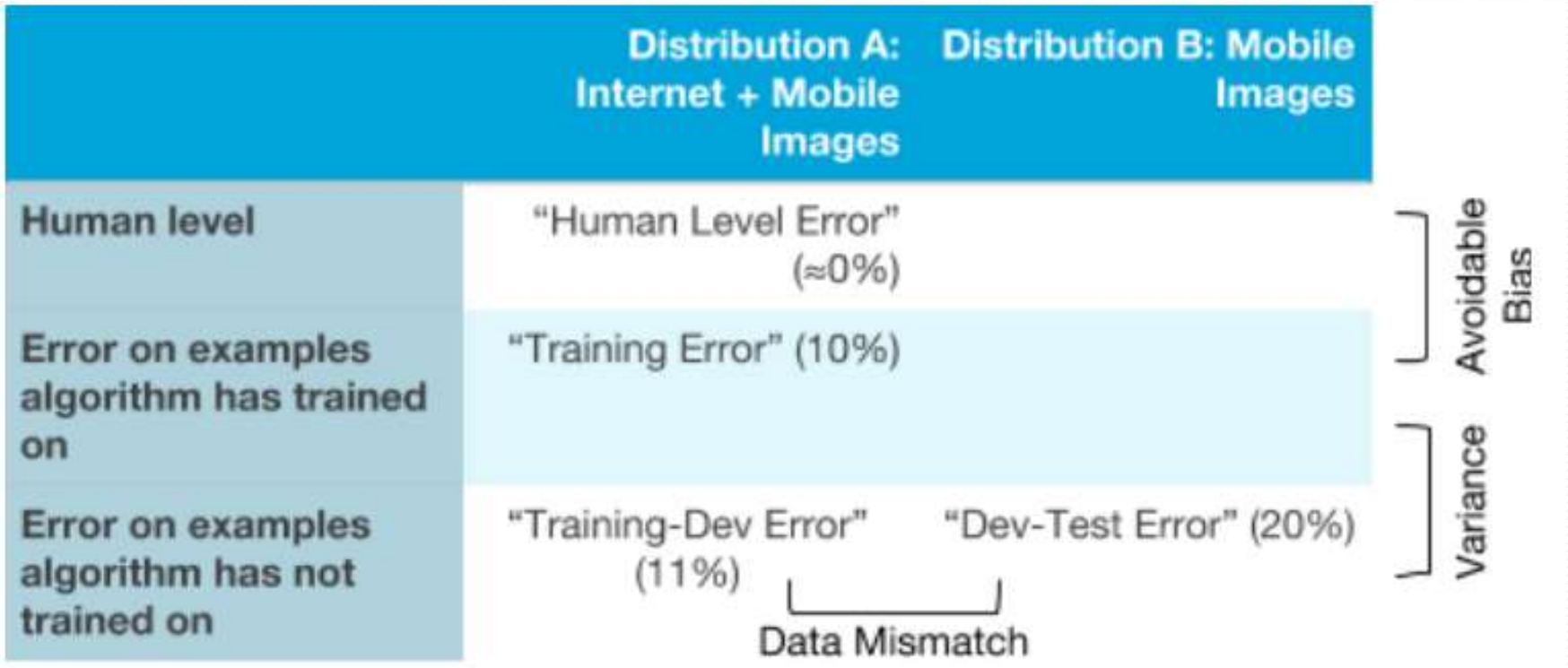
- Suppose your algorithm achieves:
  - 10% error on the training set.
  - 11% error on training dev set.
  - 12% error on the dev set.
- High avoidable bias

Identifying Bias, Variance, and Data Mismatch Errors

- Suppose your algorithm achieves:
  - 10% error on the training set.
  - 11% error on training dev set.
  - 20% error on the dev set.

- High avoidable bias and data mismatch
- High variance?

Identifying Bias, Variance, and Data Mismatch Errors



Addressing Data Mismatch

- Situation: A speech recognition system works well on a training set as well as training dev set, but does not work well on the dev set.
- Analysis: ?

Addressing Data Mismatch

- Situation: A speech recognition system works well on a training set as well as training dev set, but does not work well on the dev set.
- Analysis: Manually Examine 100 samples:
- Finding: most of the audio clips in the dev set are taken within a car, whereas most of the training examples were recorded against a quiet background.

Addressing Data Mismatch

- Situation: A speech recognition system works well on a training set as well as training dev set, but does not work well on the dev set.
- Analysis: Manually Examine 100 samples:
- Finding: most of the audio clips in the dev set are taken within a car, whereas most of the training examples were recorded against a quiet background.
- Action: Add more data having characteristics like dev set
- Challenge: Such data may not be readily available
- Solution: Generate artificial data

Artificial Data Synthesis

- Already available: speech data
- Collect: data of car noise
- Add: noise to available speech data
- Care: If we have just one hour of car noise, the algorithm may overfit. Data so generated may appear realistic to a human, but not to a machine!
- Mobile (dev set) images: Blurred
- Central Idea: Avoid giving the synthesized data properties that makes it possible for a learning algorithm to distinguish synthesized from non-synthesized examples

The Optimization Verification test

- Problem: Convert an audio clip (say, A) to text
- Evaluate: $\text{Score}_A(S)$, for each possible output sentence

$$\text{Output} = \arg\max_S \text{Score}_A(S)$$

- Suppose English language has 50,000 words
- How many sentences of length N to be evaluated?

The Optimization Verification test

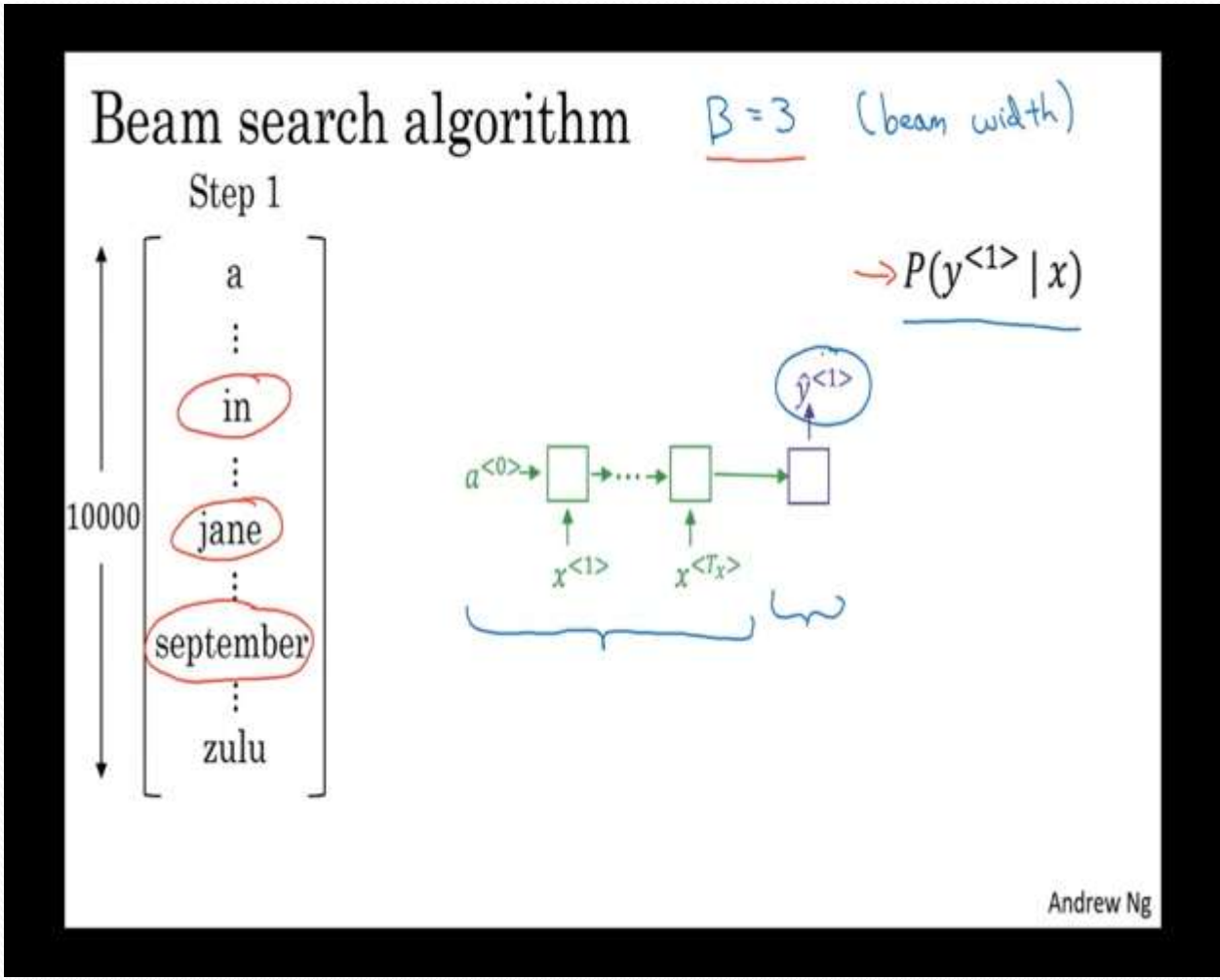
- Problem: Covert an audio clip to text
- Evaluate: $\text{Score}_A(S)$, for each possible output sentence

$$\text{Output} = \arg \max_S \text{Score}_A(S)$$

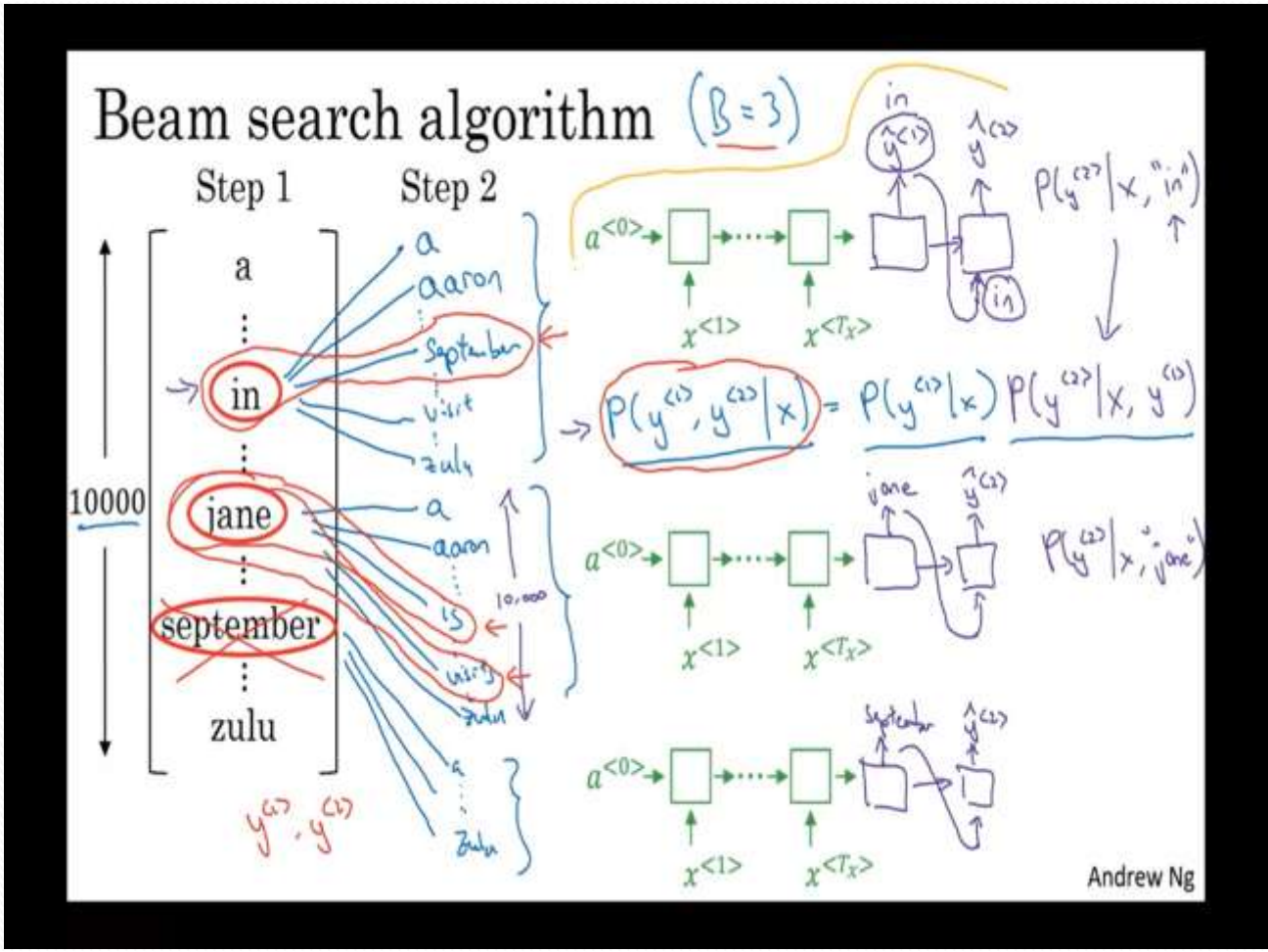
- Suppose English language has 50,000 words
- How many sentences of length N to be evaluated?
- $50{,}000^N = (5X10^4)^N = 5^N \text{ X } (10^4)^N = 5^N \text{ X } 10^{4N}$

  $= 5^8 \text{ X } 10^{32} = 390625 \text{ X } 10^{32} = 3.9X\ 10^{37}$

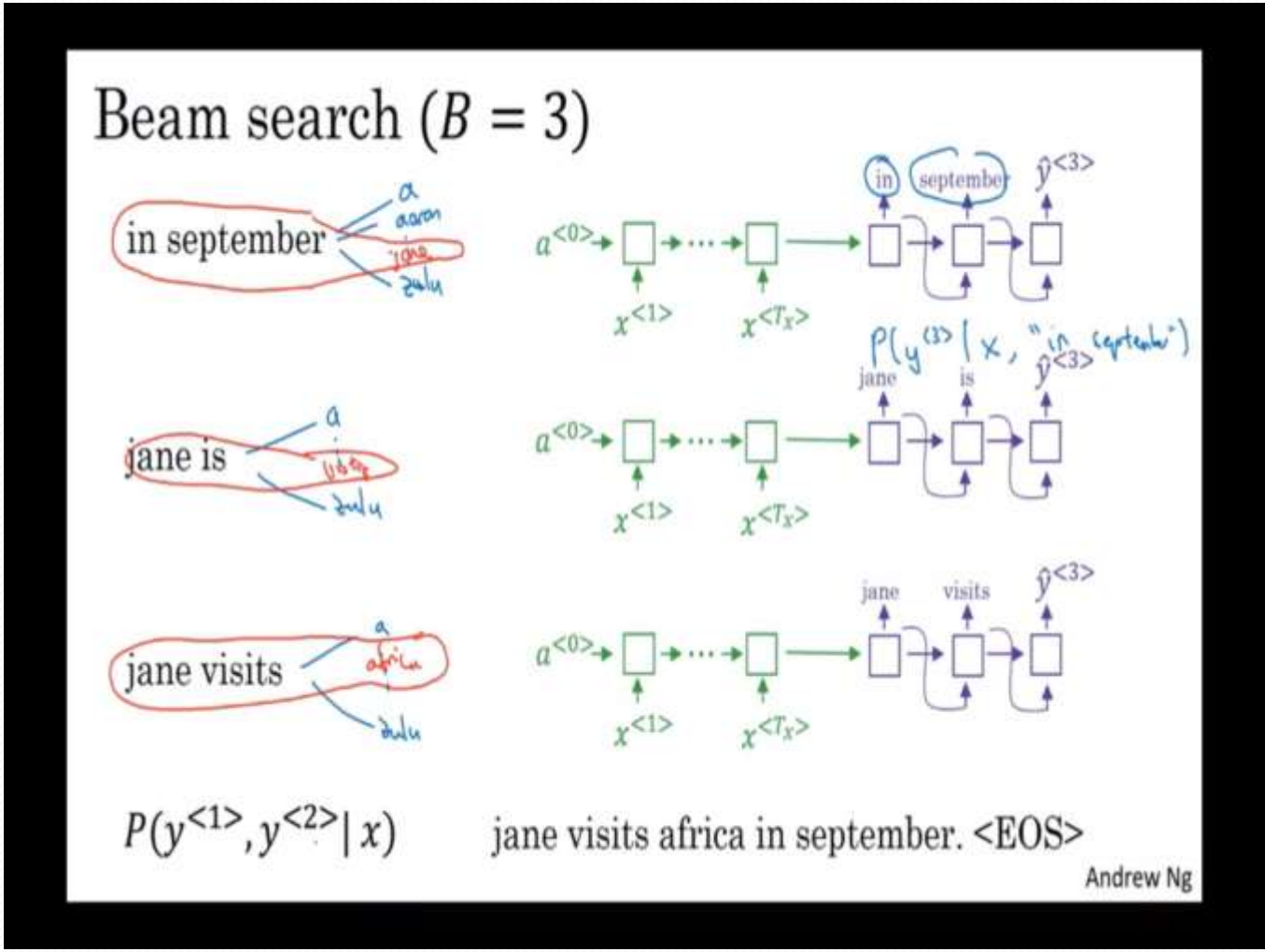
- Beam search picks not just the best word, but B (width of beam search) best words



- As B= 3, the algorithm finds three best words at every step.
- In step 1, in, jane, and September are picked up.
- In step 2, $P(y^{<1>}, y^{<2>}| x)$ is evaluated for each of the three choices of $y^{<1>}$ and three best matches are selected for each of three choices of $y^{<1>}$.
- Next, out of nine $y^{<1>}$, $y^{<2>}$ pairs, best three are selected.
- Suppose, beam search decides that the most likely choices for the first and second words are **in September**, or **Jane is**, or **Jane visits**.
- Then Beam search rejects September as a candidate for the first word for further search.
- Thus, at every step three copies of the network are evaluated



- Notice that it is only by chance that Beam search has picked one word from each of the three networks.
- Sentence at the bottom is likely output when Beam search is continued.

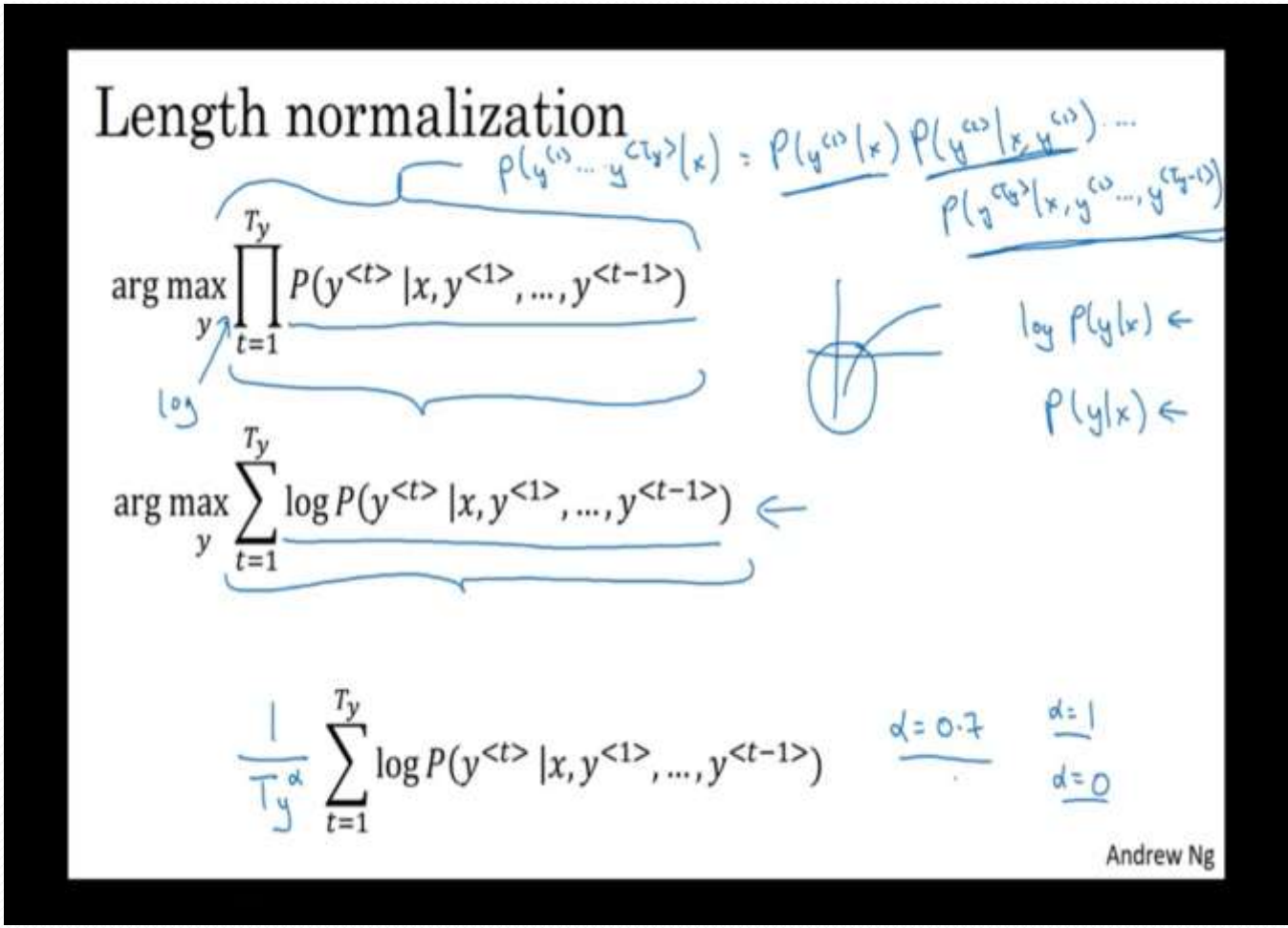


Refinements of Beam Search

- As multiplying a number of small probabilities may lead to underflow, use log of the product of probabilities.
- This changes product to summation
- Normalize by length. In fact one uses $(T_y)^{\alpha}$ where $\alpha = 0.7$

How to choose B

- Application dependent, but 1000 to 3000 is not uncommon



Beam search discussion

large B: better result, slower

small B: worse result, faster

Beam width B?

1 → 3 → 10, 100, 1000, → 3000

Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$.

The Optimization Verification test: Beam Search

- Original Sentence (S^*): "I love machine learning"
- Output Sentence (S_{out}): "I love robots"
- $Score_A(S^*) > Score_A(S_{out})$?
- Case 1: $Score_A(S^*) > Score_A(S_{out})$
- Scoring algorithm has correctly rated S^*, S_{out}
- Culprit: Beam Search
- Case 2: $Score_A(S^*) <= Score_A(S_{out})$
- Culprit: Scoring algorithm

The Optimization Verification test: Beam Search

- In practice,
  - Beam search may be at fault for some sentences
  - Scoring function may be at fault for some other sentences
- Example:
  - 95% of the errors due to the scoring function
  - 5% due to search optimization algorithm.

Reinforcement learning example
Find trajectory (T) for safe landing

Reinforcement learning example Find trajectory (T) for safe landing

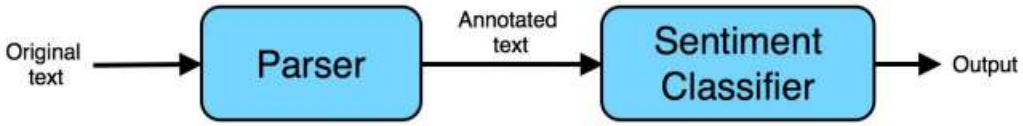
- $R(T_{human}) > R(T_{out})$?
- Case 1: $R(T_{human}) > R(T_{out})$
  - Reward function R(.) is correctly rating T_{human} as superior to T_{out}
  - Reinforcement learning algorithm is finding the inferior T_{out}
- Case 2: $R(T_{human}) <= R(T_{out})$
  - R(.) assigns a worse score to T_{human} even though it is the superior trajectory.
  - We should work on improving R(.) to better capture the tradeoffs that correspond to a good landing.

Audio clip transcription vs helicopter landing

- Audio clip transcription: Comparing with well defined optimal performance
- helicopter landing: Comparing with human performance
- In general, so long as you have some y* (in this example, T_{human}) that is a superior output to the performance of your current learning algorithm — even if it is not the "optimal" output—then the Optimization Verification test can indicate whether it is more promising to improve the optimization algorithm or the scoring function.

Sentiment classifier: traditional approach

- Sentiment classification: a system (classifier) to examine online product reviews and automatically tell you if the writer liked or disliked that product.
- positive: This is a great mop!
- negative: This mop is low quality--I regret buying it.
- Solution: Build a pipeline of two components

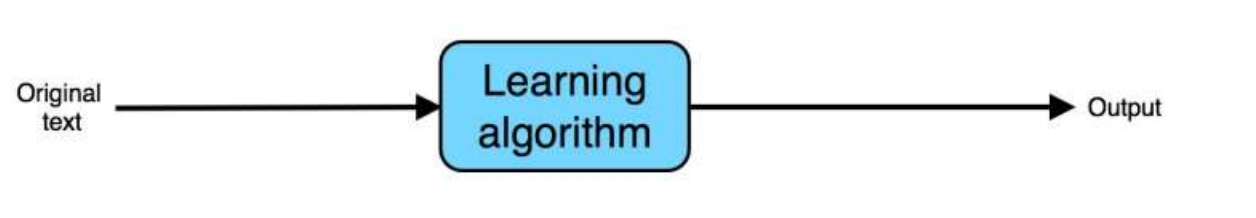


Sentiment classifier: traditional approach

- Parser: Annotates the text with information identifying the most important words. For example, use the parser to label all the nouns and adjectives.
  - This is a great$_{\text{Adjective}}$ mop$_{\text{Noun}}$!
- Sentiment classifier: A learning algorithm that takes as input the annotated text and predicts the overall sentiment.
- The parser's annotation could help the learning algorithm greatly: By giving adjectives a higher weight, your algorithm will be able to quickly hone in on the important words such as "great," and ignore less important words such as "this."

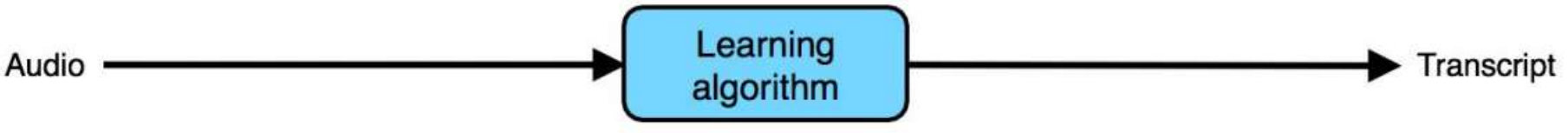
Sentiment classifier: An end-to-end learning algorithm

- End-to-end learning: go directly from the input to the desired output
  - Typically deploy large neural networks
  - Require abundant data



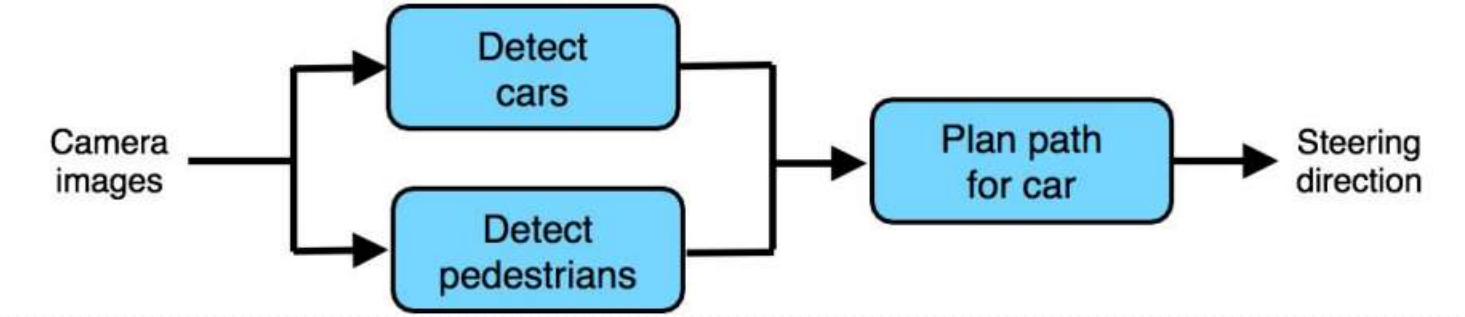
Speech recognition system

- Compute features: Extract hand-designed features, such as MFCC (Mel-frequency cepstrum coefficients) features, which try to capture the content of an utterance while disregarding less relevant properties, such as the speaker's pitch.
- Phoneme recognizer: Some linguists believe that there are basic units of sound called "phonemes." For example, the initial "k" sound in "keep" is the same phoneme as the "c" sound in "cake." This system tries to recognize the phonemes in the audio clip.
- Final recognizer: Take the sequence of recognized phonemes, and try to string them together into an output transcript.
- End to end learning:

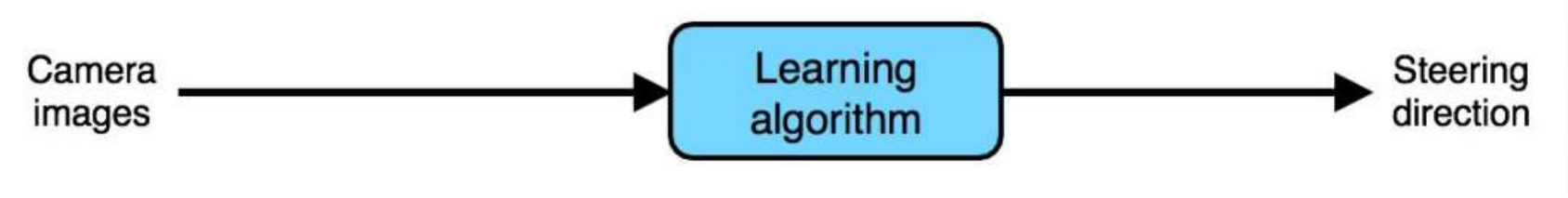


Non-linear ML pipeline

Traditional approach:



End-to-end approach:



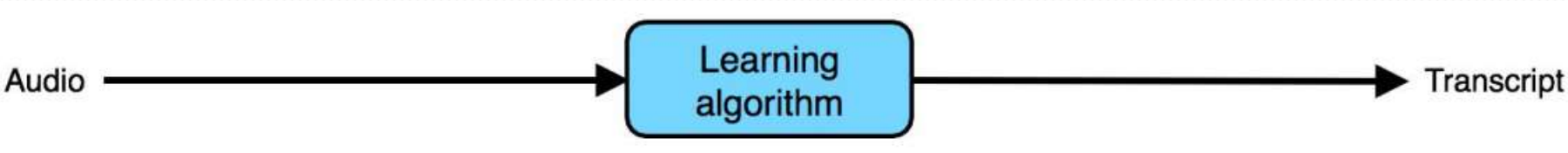
Pros and cons of end-to-end learning

- MFCCs : A set of hand-designed audio features.
  - they provide a reasonable summary of the audio input,
  - they also simplify the input signal by throwing some information away.
- Phonemes: An invention of linguists.
  - an imperfect representation of speech sounds.
  - a poor approximation of reality,
  - forcing an algorithm to use a phoneme representation will limit the speech system's performance.

Allowing hand-engineered components: Advantages

- MFCC features are robust to some properties of speech that do not affect the content,
  - thus, they help simplify the problem for the learning algorithm.
  - examples: speaker pitch.
- To the extent that phonemes are a reasonable representation of speech, they can also help the learning algorithm understand basic sound components and therefore improve its performance.
- Having more hand-engineered components generally allows a speech system to learn with less data.
- The hand-engineered knowledge captured by MFCCs and phonemes "supplements" the knowledge our algorithm acquires from data.
- When we don't have much data, this knowledge is useful.

End-to-end system



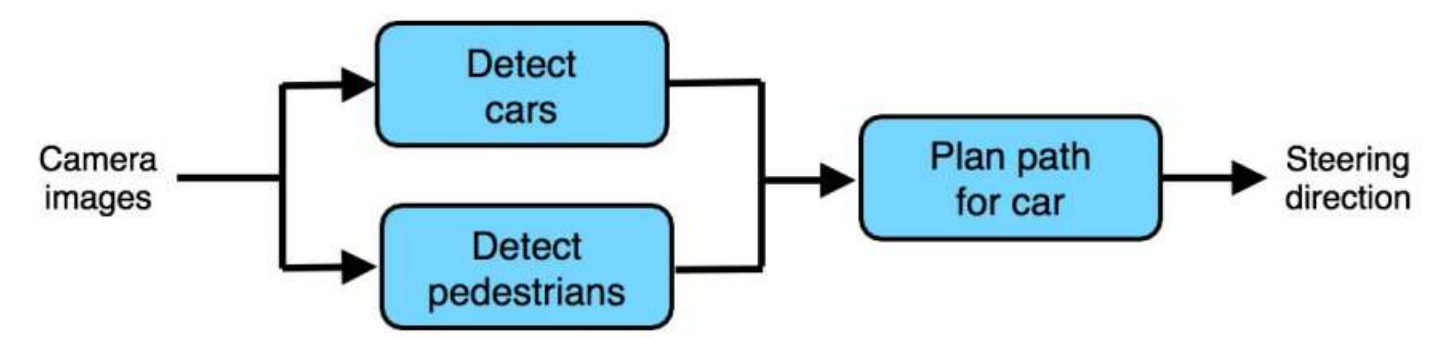
- This system lacks the hand-engineered knowledge.
- When the training set is small,
  - it might do worse than the hand-engineered pipeline.
- When the training set is large,
  - it is not hampered by the limitations of an MFCC or phoneme-based representation.

End-to-end system

- This system lacks the hand-engineered knowledge.
- When the training set is small, it might do worse than the hand-engineered pipeline.
- When the training set is large,
  - it is not hampered by the limitations of an MFCC or phoneme-based representation.
  - If the learning algorithm is a large-enough neural network and if it is trained with enough training data, it has the potential to do very well, and perhaps even approach the optimal error rate.

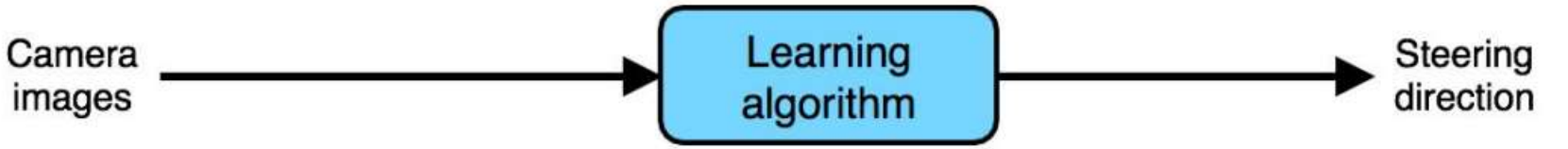
Choosing pipeline components: Data availability

- An autonomous driving architecture



- Use machine learning to detect cars and pedestrians.
  - it is not hard to obtain data for these
  - numerous computer vision datasets with large numbers of labeled cars and pedestrians.
  - use crowdsourcing (such as Amazon Mechanical Turk) to obtain even larger datasets.
- It is thus relatively easy to obtain training data to build a car detector and a pedestrian detector.

A pure end-to-end approach



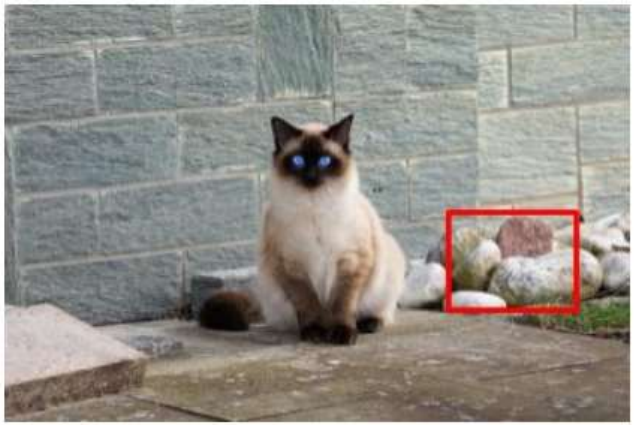
- Need a large dataset of (Image, Steering Direction) pairs
  - very time-consuming and expensive to have people drive cars around and record their steering direction to collect such data.
  - need a fleet of specially-instrumented cars, and a huge amount of driving to cover a wide range of possible scenarios.

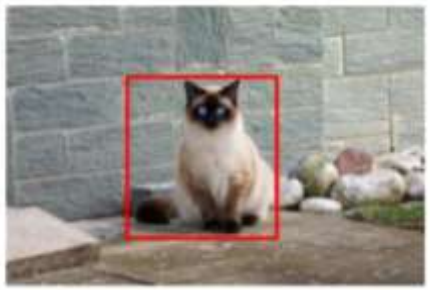
Pipeline vs end-to-end approach

- A lot of data available for training “intermediate modules” of a pipeline (such as a car detector or a pedestrian detector)
  - consider using a pipeline with multiple stages

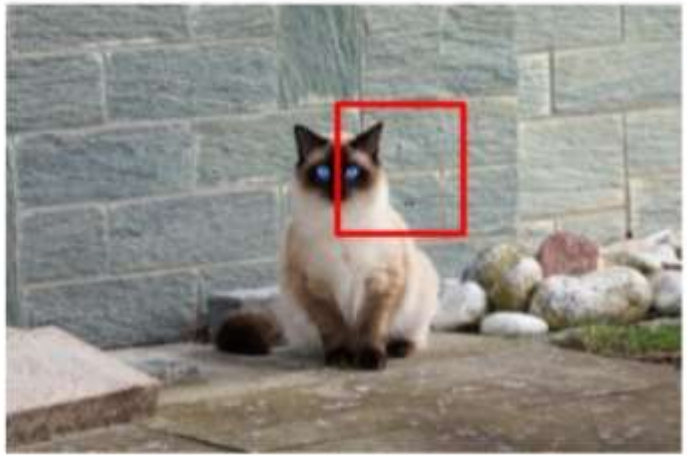
Error Analysis by Parts

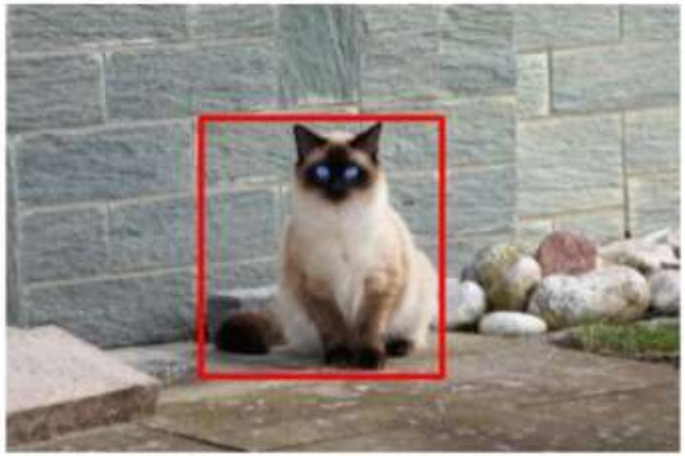
Image → Cat detector → cat breed classifier → label 0/1





Attributing Error to One Part

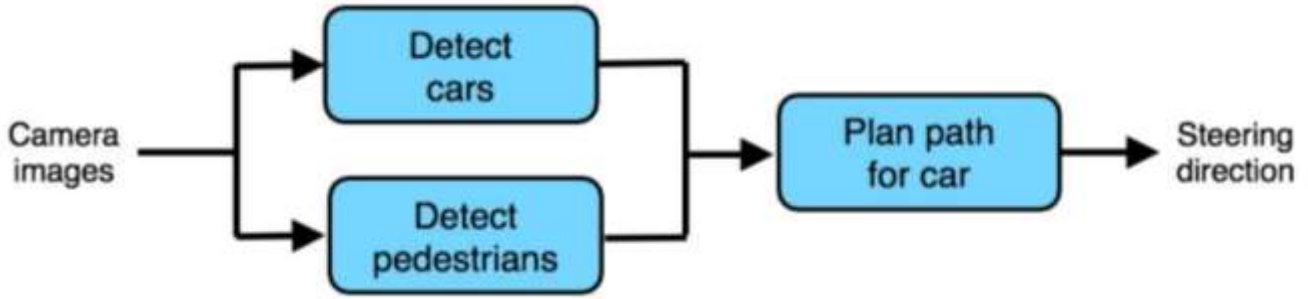




- Replace the detector with a hand made bounding box
- Run the classifier again

General Case of Error Attribution

- A → B → C→ output

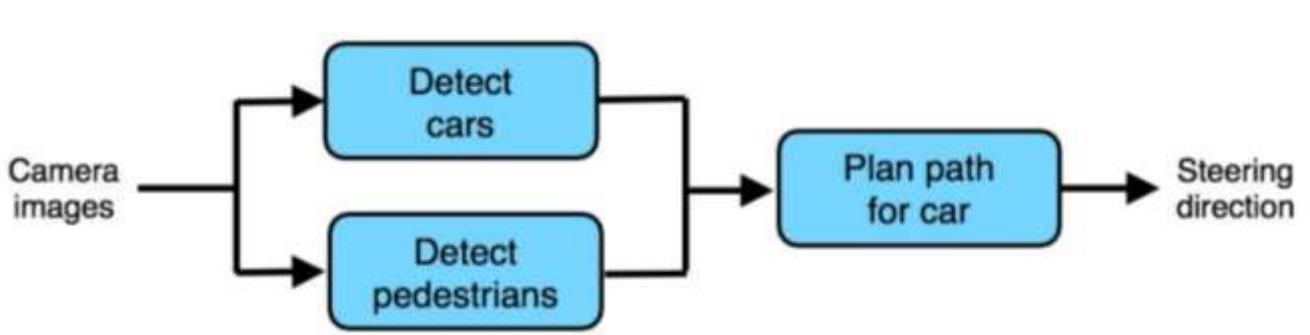


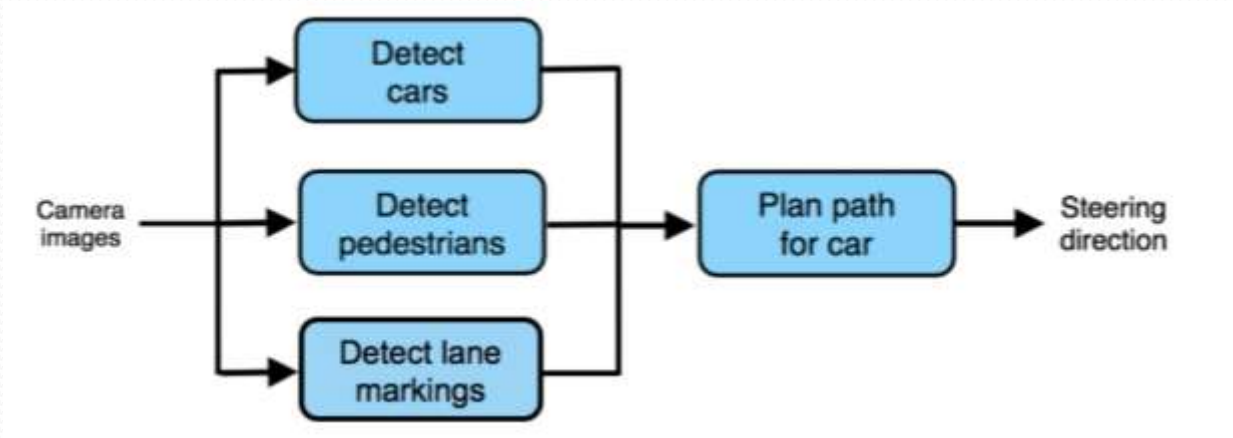
- DAG ordering of the steps in ML pipeline

Informal Analysis

How far is the performance of each component from human-level

Spotting a Flawed ML Pipeline



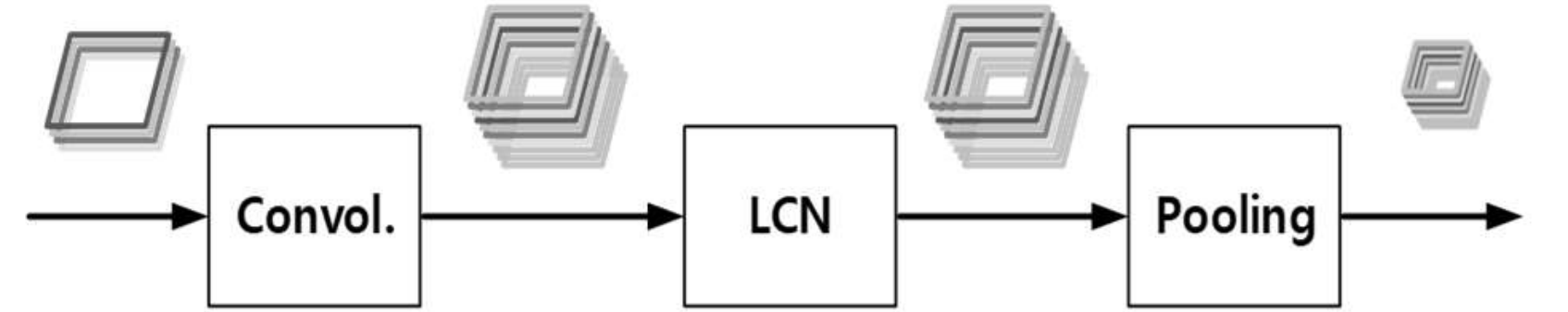


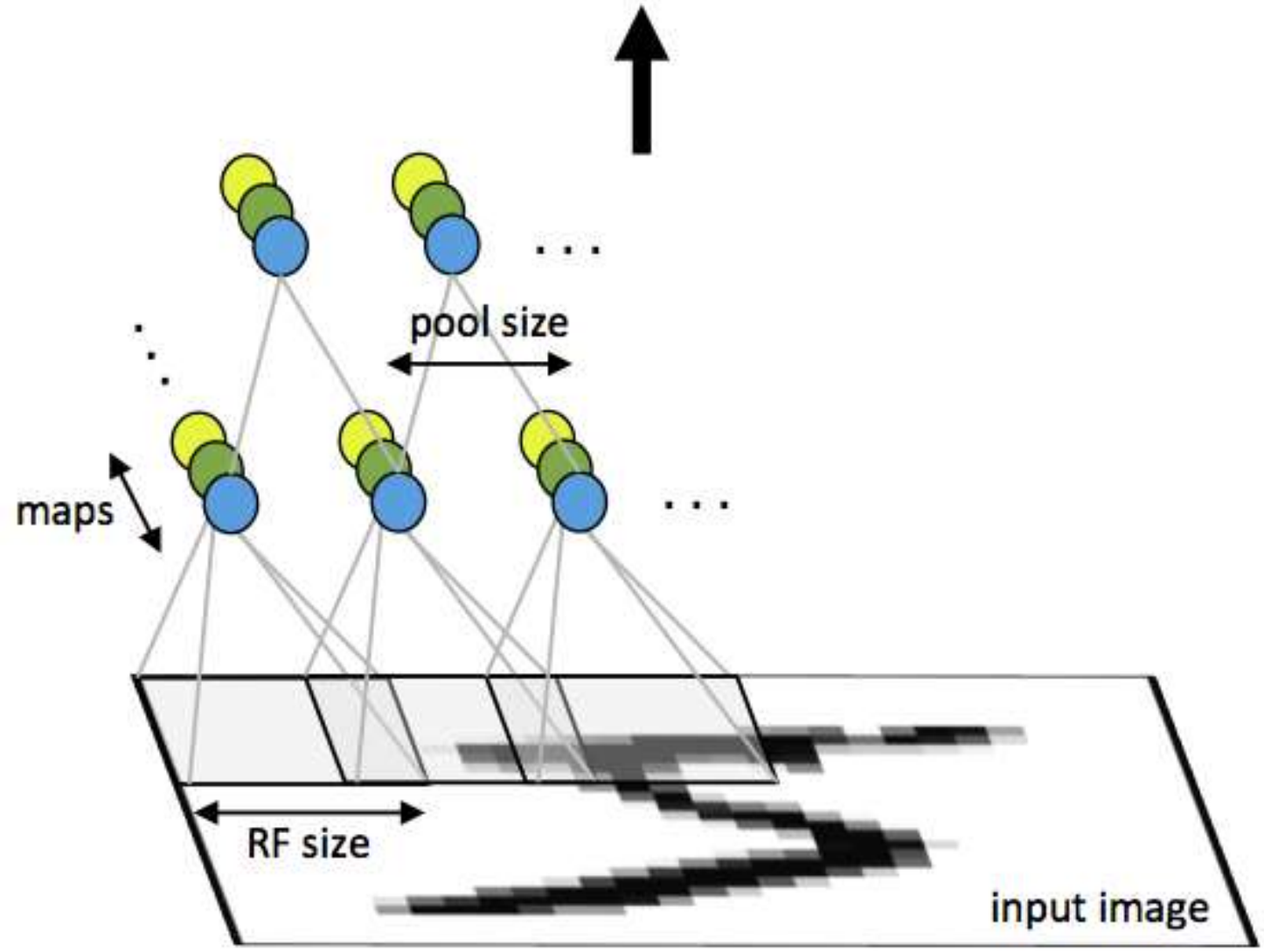
Convolutional Neural Network (CNN)

- A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers.
- The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels.
- Convolution is used to find the same feature in different places of an image
- Convolution is conducted using learnable filters/kernels (that have small receptive fields) that are passed (convoluted) through the input data/image
- Each filter moves sequentially across (convolved) the input data/image to make a 2-dimensional activation map based on each filter
- Feature Maps are made from the activation maps of the filters

CNN Processing Characteristics

- Convolution layer increases number of feature maps
- LCN improves the optimization results and the image's invariance
- LCN can be used on the image after convolution and before pooling (subsampling) (i.e., characteristic of not changing after transformation or processing)
- Pooling (subsampling) layer deceases spatial resolution





A full layer in a CNN consisting of convolutional and subsampling sublayers

CNN Characteristics

- ReLU (Rectified Linear Units) activation functions are often used

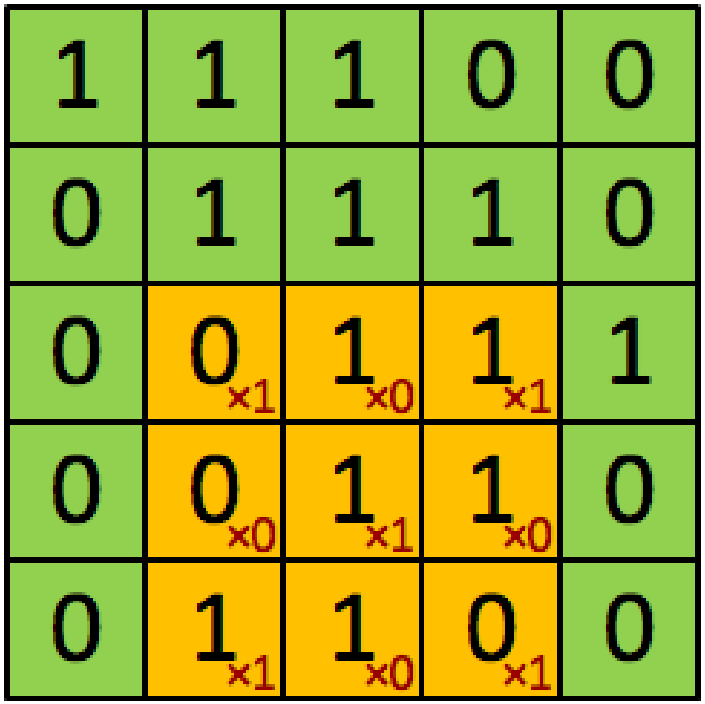
- CNNs are used in image/video recognition, recommender systems, natural language processing, Chess, Go, etc.

Feature Extraction Using Convolution

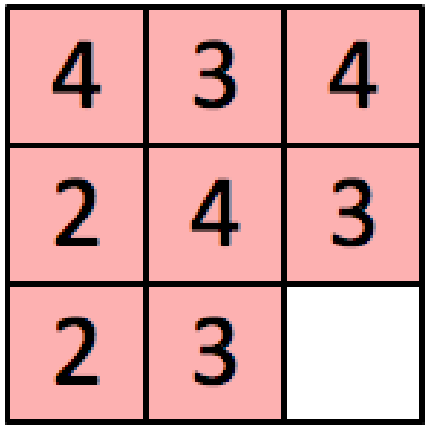
- Natural images have the property of being "'stationary"', meaning that the statistics of one part of the image are the same as any other part.
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.
- More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image

Example

- To give a concrete example, suppose you have learned features on 8x8 patches sampled from a 96x96 image.
- Suppose further this was done with an auto-encoder that has 100 hidden units.
- To get the convolved features, for every 8x8 region of the 96x96 image, that is, the 8x8 regions starting at (1,1),(1,2),…(89,89) (1,1),(1,2),…(89,89), you would extract the 8x8 patch, and run it through your trained sparse auto-encoder to get the feature activations.
- This would result in 100 sets 89x89 convolved features.



Image

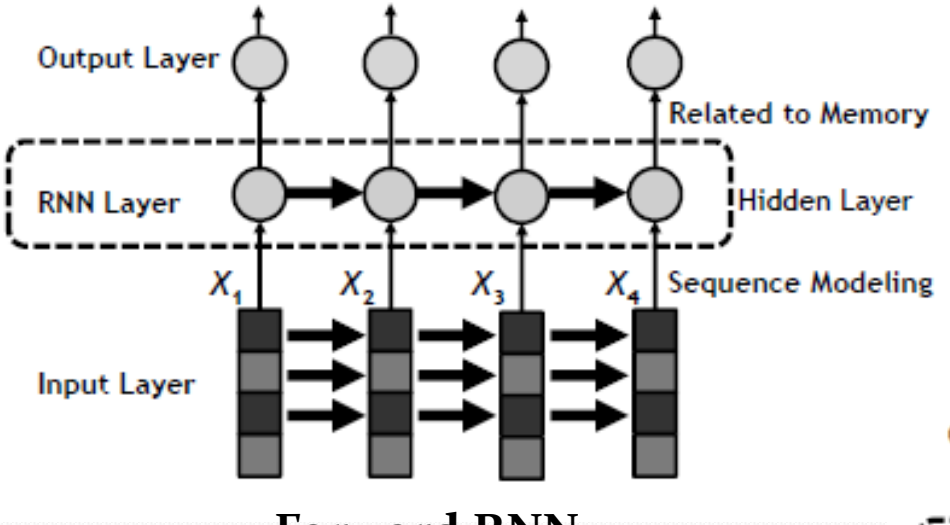


Convolved Feature

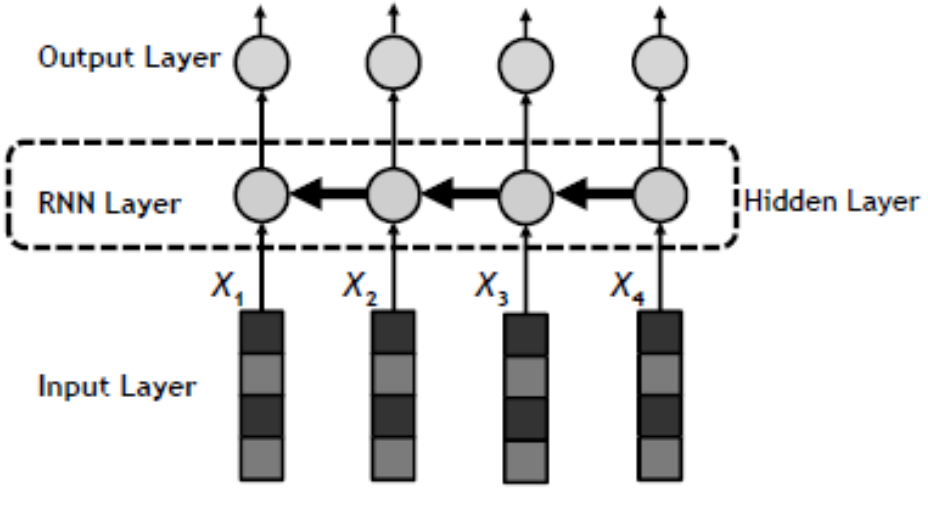
RNN (Recurrent Neural Network)

- It is a Neural Network that uses directed cyclic connections between neurons.
- Directed cyclic connections create internal states with dynamic temporal characteristics
- Internal memory is used to process arbitrary input data sequences
- Sequence Modeling is used for data sequence Classification & Clustering
- Sequence Modeling structure based on S2S (Sequence to Sequence) Learning
  - *N* inputs are transformed into *M* outputs

Forward and Backward RNN



Forward RNN



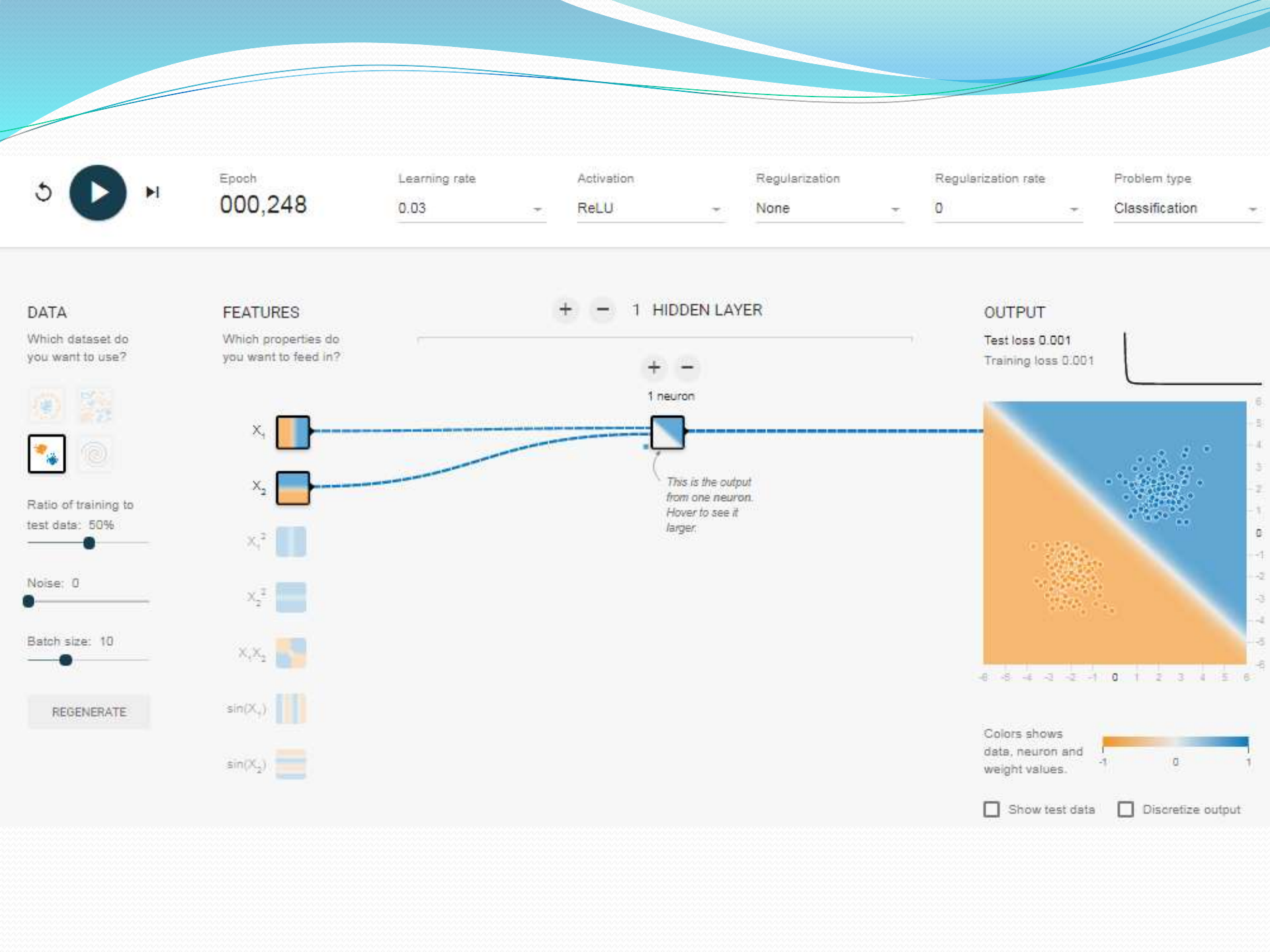
Backward RNN

Project 1:Classification of two sets of clusters of data

- TensorFlow Playground Project Setup
  - Learning rate : 0.03
  - Activation : ReLU
  - Regularization : None
  - Regularization rate : 0
  - Problem type : Classification
  - Ratio of training to test data : 50%
  - Noise : 0
  - Batch size : 10

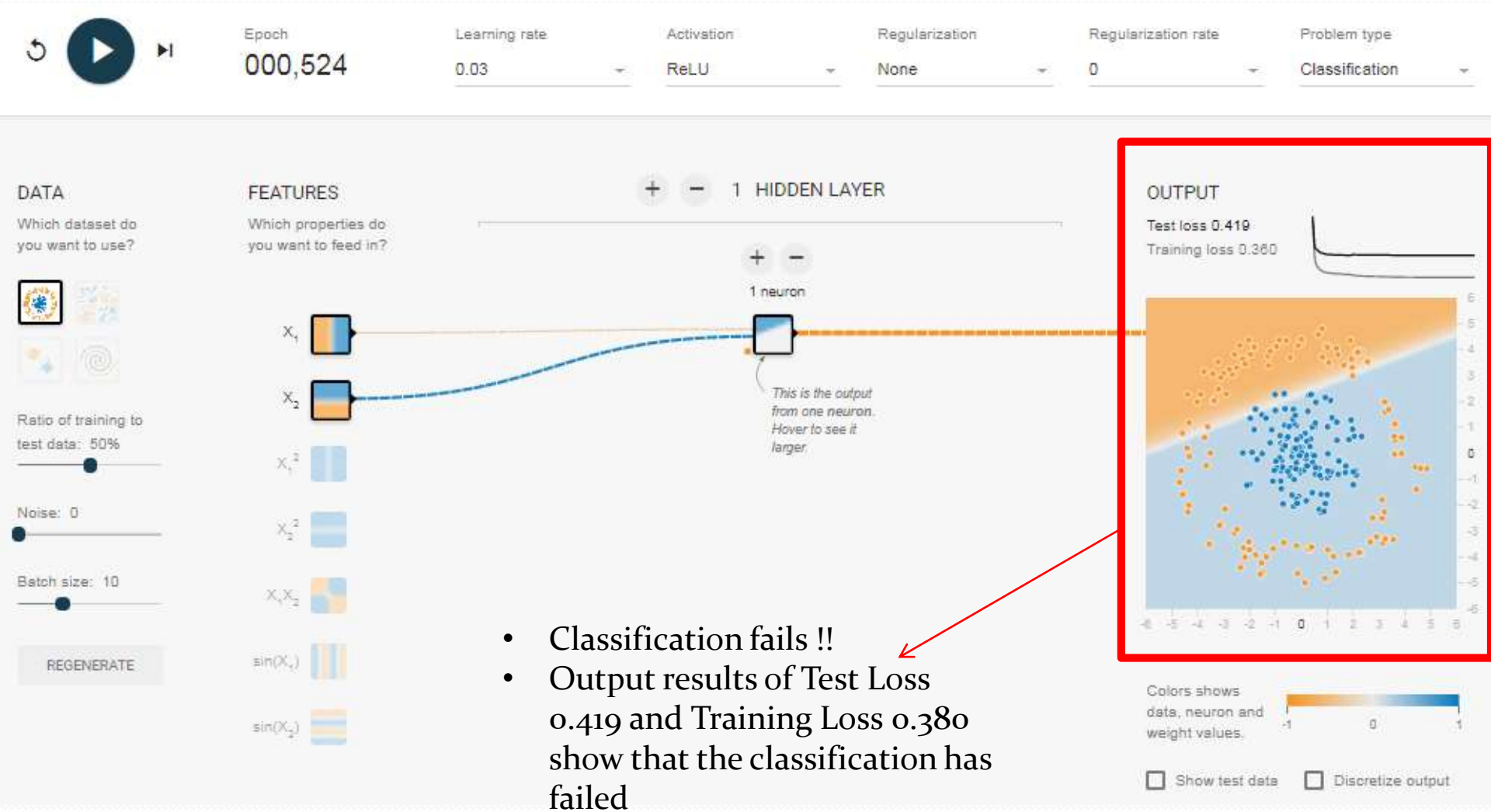
Regularization is not needed in solving a simple problem, because overfitting most likely will not occur

Noise is set to zero to make it easy to find the solution. Try to practice more with higher noise levels

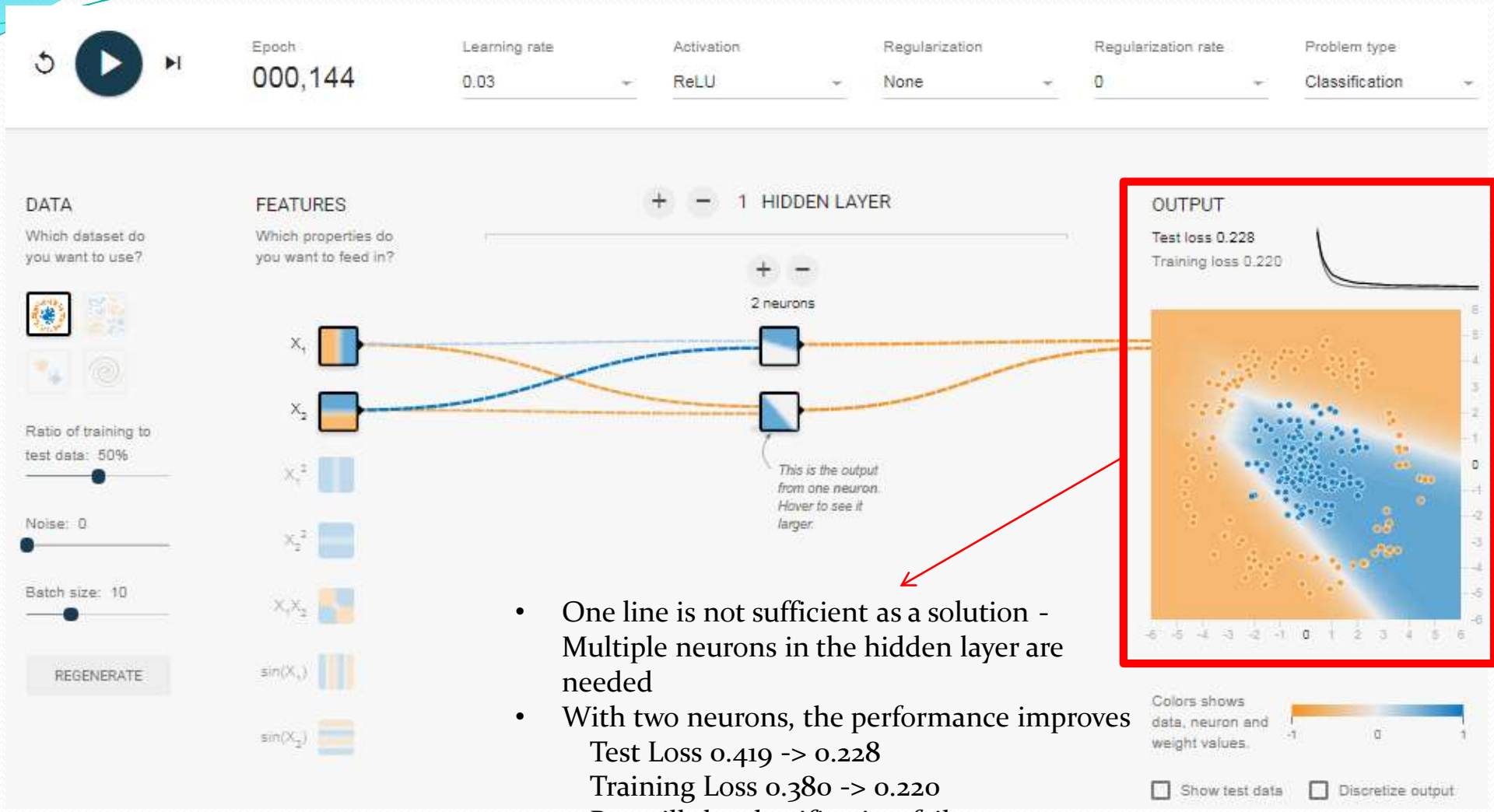

Epoch
000,248
Learning rate
0.03
Activation
ReLU
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.001
Training loss 0.001
Colors shows data, neuron and weight values.
Show test data
Discretize output

Project 2

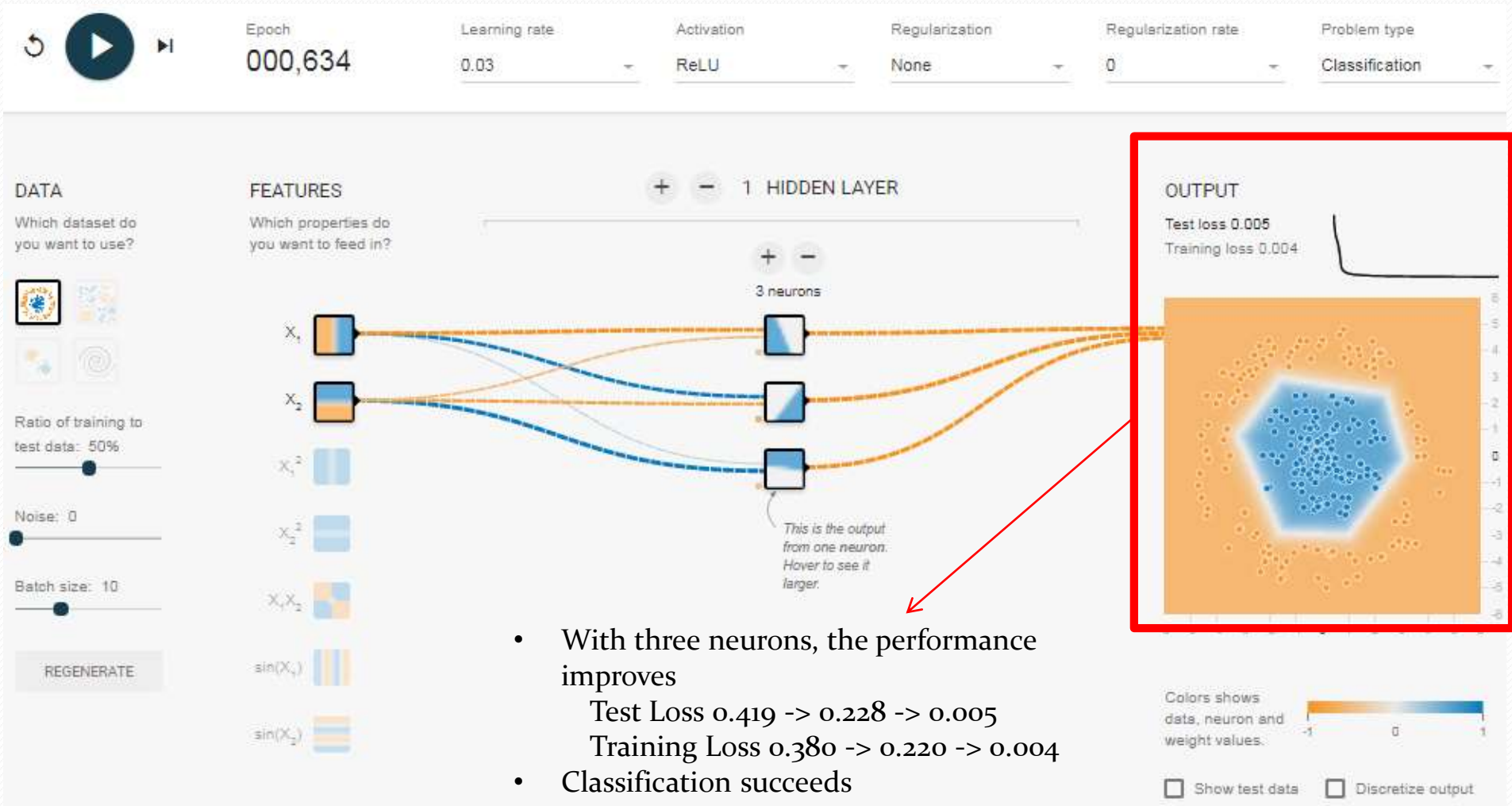
- Classification of two Data sets
- Orange data surrounds the blue data in a circular shape



- Classification fails !!
- Output results of Test Loss 0.419 and Training Loss 0.380 show that the classification has failed

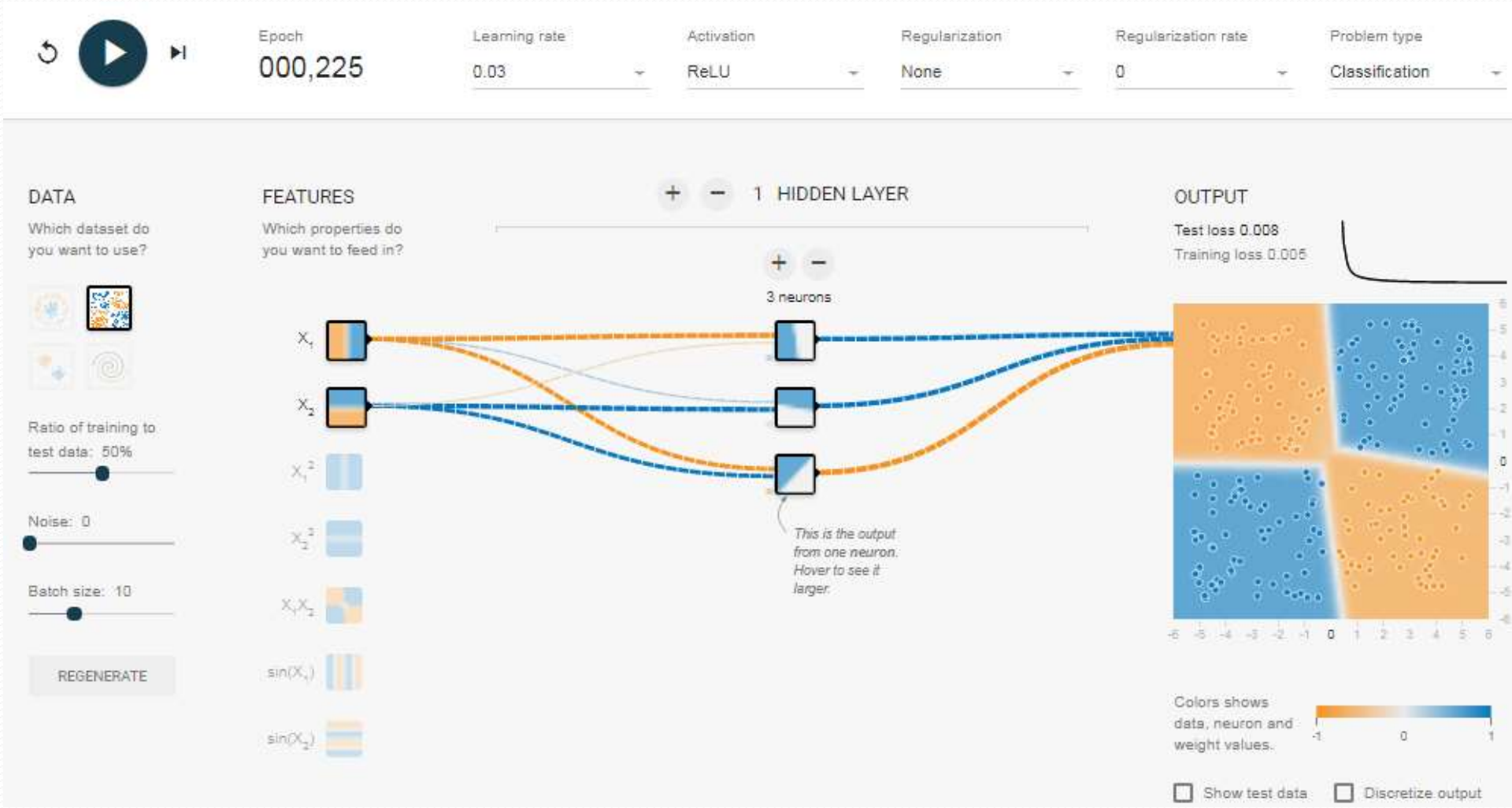


- One line is not sufficient as a solution - Multiple neurons in the hidden layer are needed
- With two neurons, the performance improves
  - Test Loss 0.419 -> 0.228
  - Training Loss 0.380 -> 0.220
  - But still the classification fails



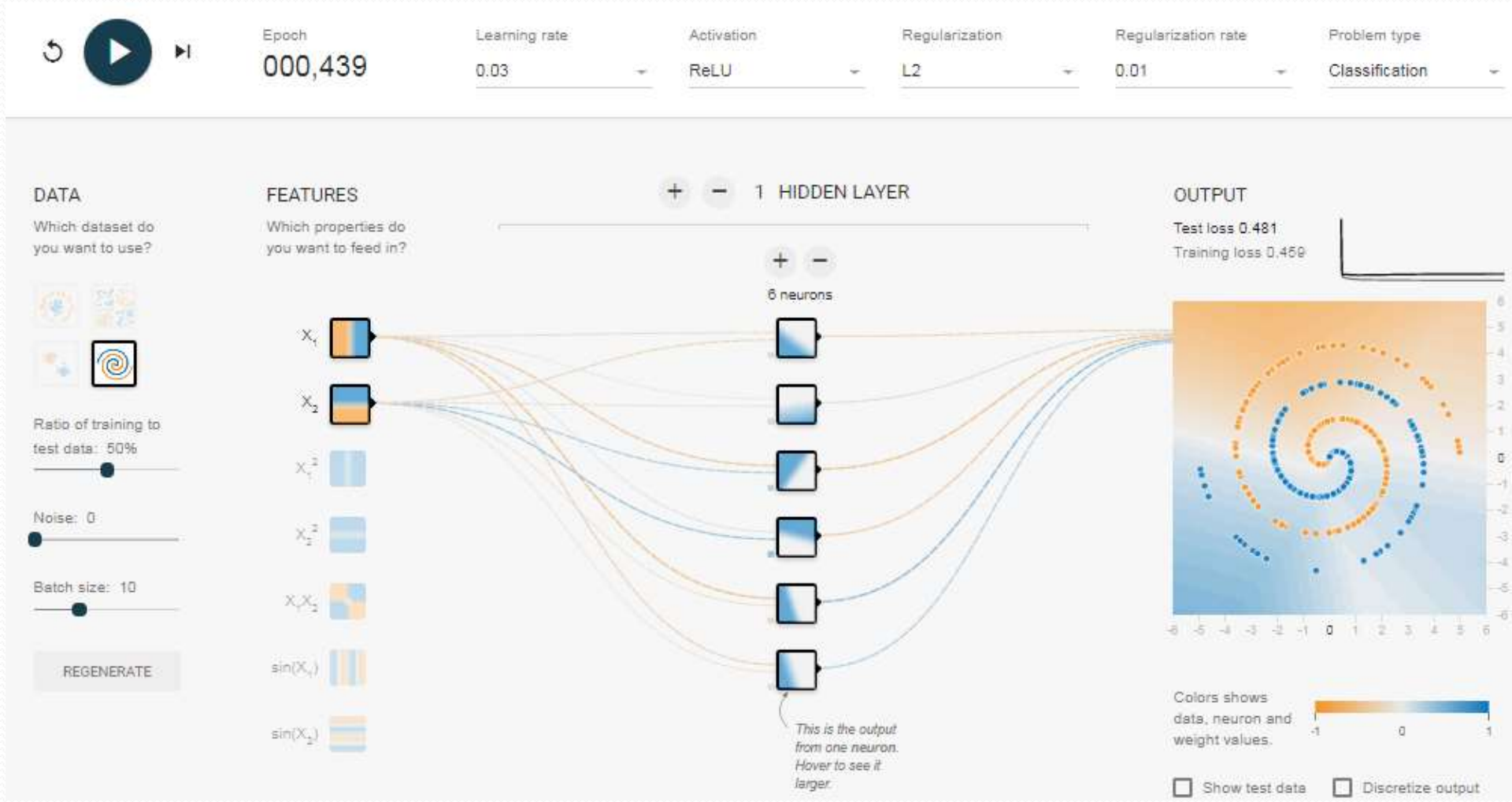
- With three neurons, the performance improves
  - Test Loss 0.419 -> 0.228 -> 0.005
  - Training Loss 0.380 -> 0.220 -> 0.004
- Classification succeeds

Project 3

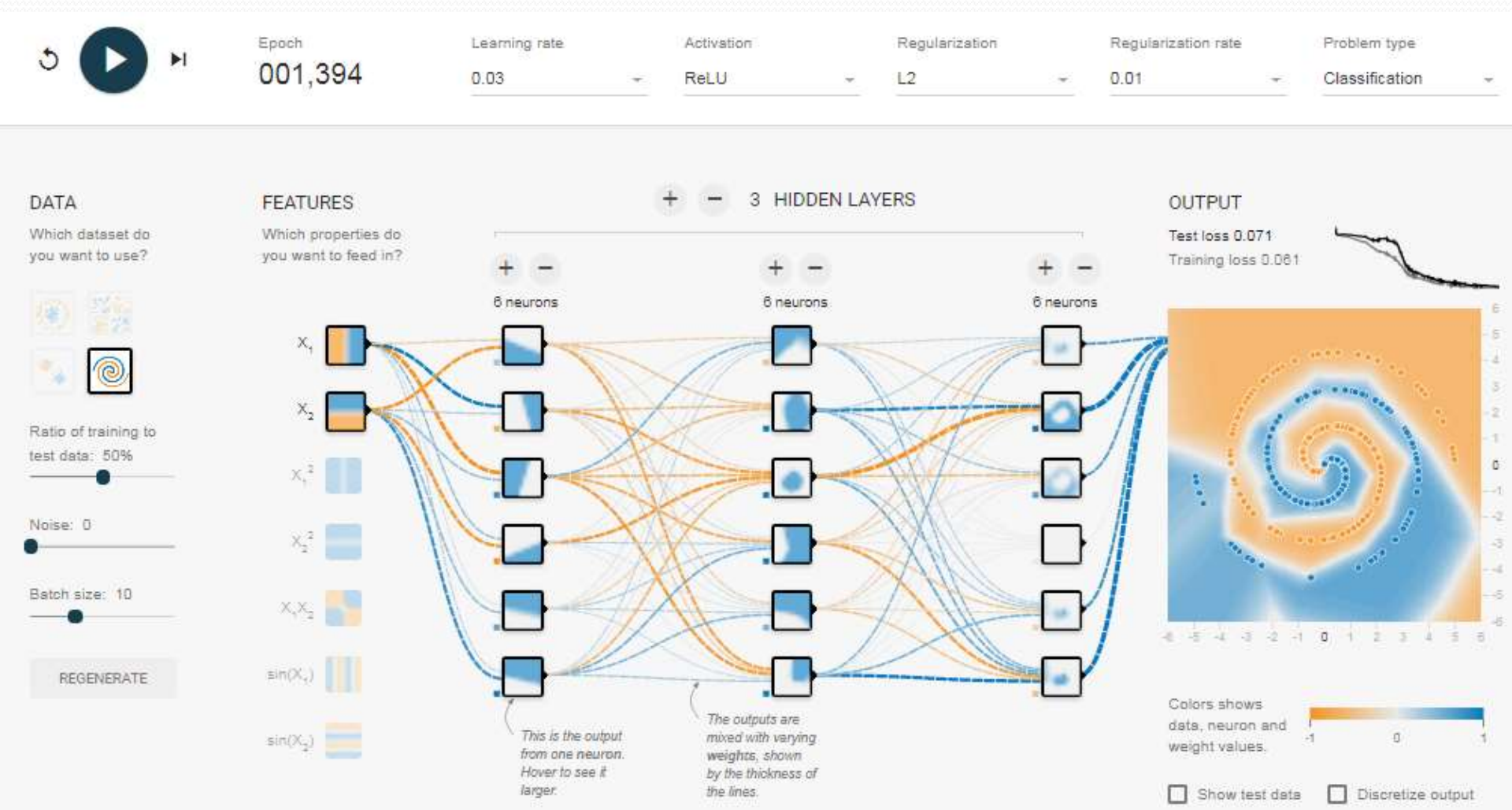

Epoch
000,225
Learning rate
0.03
Activation
ReLU
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.008
Training loss 0.005
Colors shows data, neuron and weight values.
Show test data
Discretize output

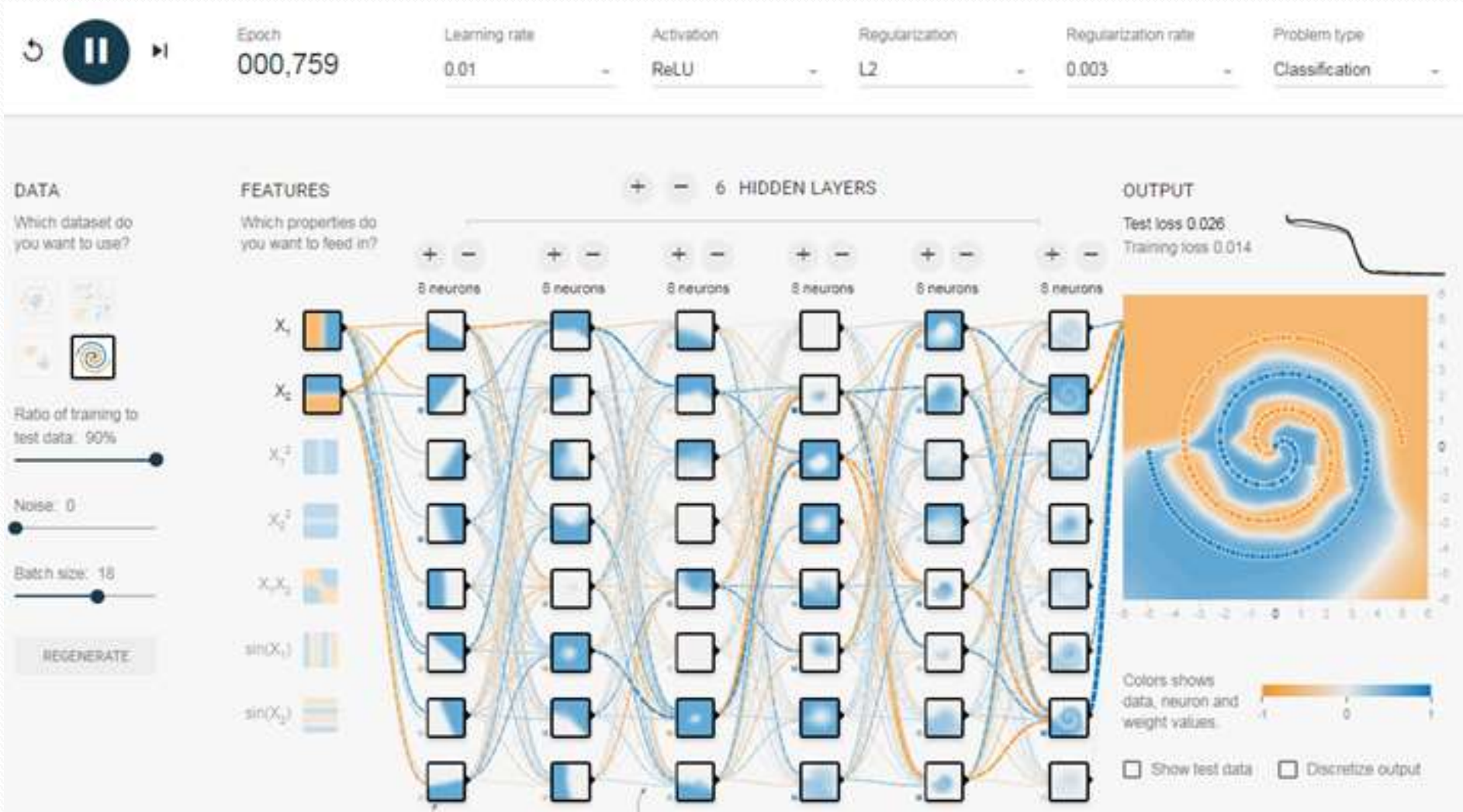
Project 4

- Classify swirled structure of orange and blue data



Classification fails even if we add number of neurons. Thus, now we will have to add hidden layers.

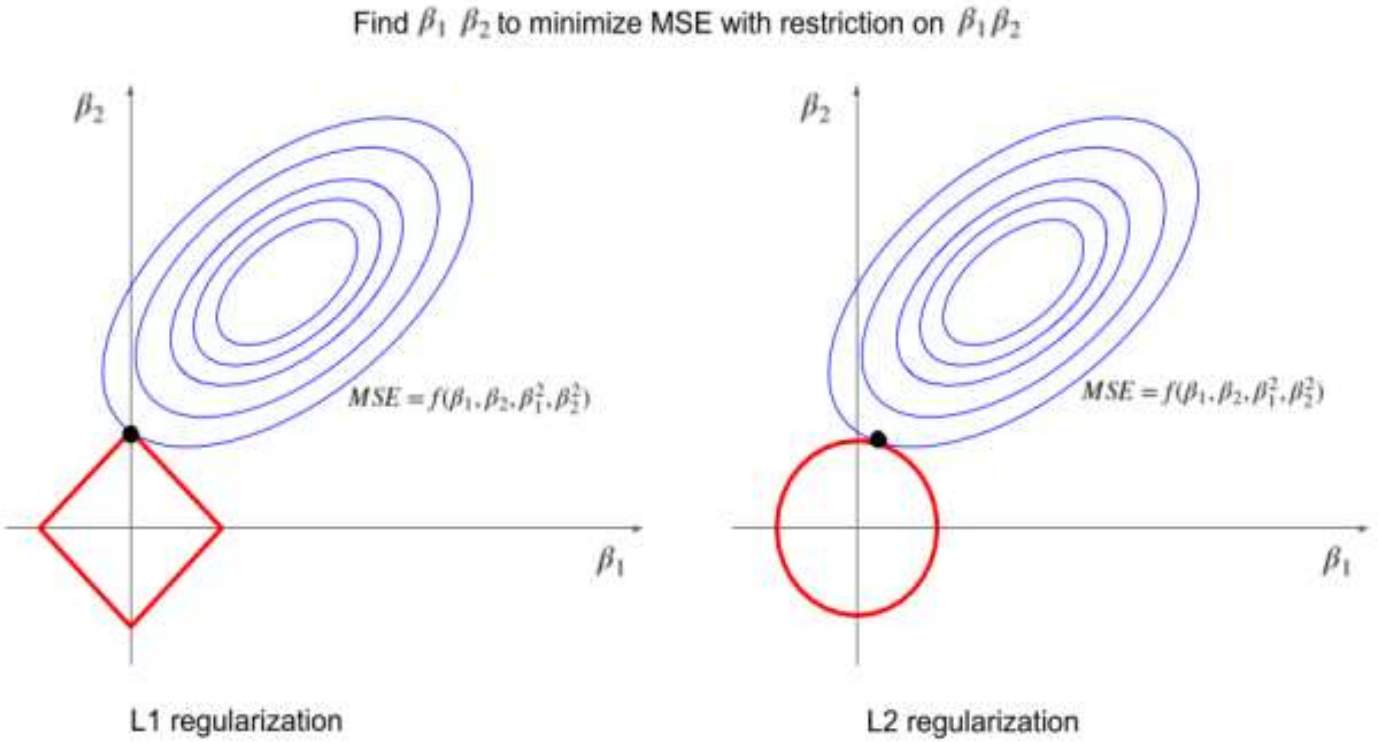

Epoch
001,394
Learning rate
0.03
Activation
ReLU
Regularization
L2
Regularization rate
0.01
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
3 HIDDEN LAYERS
6 neurons
6 neurons
6 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.071
Training loss 0.061
Colors shows data, neuron and weight values.
Show test data
Discretize output


Epoch
000,759
Learning rate
0.01
Activation
ReLU
Regularization
L2
Regularization rate
0.003
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 90%
Noise: 0
Batch size: 18
REGENERATE
FEATURES
Which properties do you want to feed in?
6 HIDDEN LAYERS
8 neurons
8 neurons
8 neurons
8 neurons
8 neurons
8 neurons
OUTPUT
Test loss 0.026
Training loss 0.014
Colors shows data, neuron and weight values.
Show test data
Discretize output

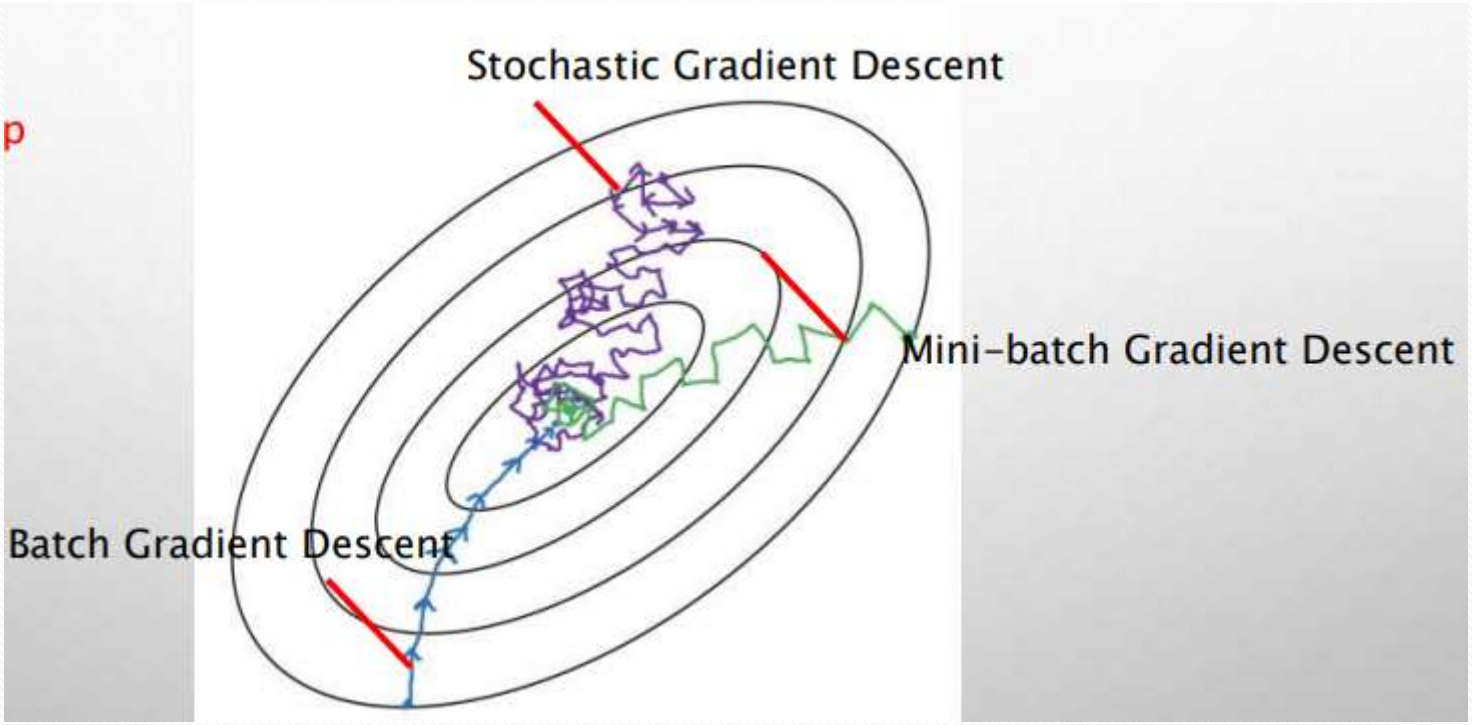

Epoch
000,343
Learning rate
0.03
Activation
ReLU
Regularization
L2
Regularization rate
0.003
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 90%
Noise: 0
Batch size: 18
REGENERATE
FEATURES
Which properties do you want to feed in?
6 HIDDEN LAYERS
8 neurons
8 neurons
8 neurons
8 neurons
8 neurons
8 neurons
OUTPUT
Test loss 0.006
Training loss 0.026
Colors shows data, neuron and weight values.
Show test data
Discretize output

Formally, given some large $r \times c$ images x_{large}, we first train a sparse autoencoder on small $a \times b$ patches x_{small} sampled from these images, learning k features $f = \sigma(W^{(1)} x_{small} + b^{(1)})$ (where σ is the sigmoid function), given by the weights $W^{(1)}$ and biases $b^{(1)}$ from the visible units to the hidden units. For every $a \times b$ patch x_s in the large image, we compute $f_s = \sigma(W^{(1)} x_s + b^{(1)})$, giving us $f_{convolved}$, a $k \times (r - a + 1) \times (c - b + 1)$ array of convolved features.

L1 vs L2 regularization



Comparing Convergence: Batch Sizes



Identifying Bias, Variance, and Data Mismatch Errors

- Consider filling up the remaining two boxes in the table
  - Ask a person to examine samples from dev-test set
  - Include part of dev-test data in the training set

The Optimization Verification test: Beam Search (Width = 3)

Slots: 1 w_{13} 2 3 4 5 6 7 8 (N= 8)

Slot 1: w_{11} , w_{12}, w_{13}: Most likely to begin the sentence

Slot 2: (Based on step 2 output w_{11} , w_{12}, w_{13}) 150,000 candidates

w_{11}: $P(x_1|w_{11})$ … $P(x_{50,00}|w_{11})$: $w_{11}\,x_{2369}$

w_{12}: $P(x_1|w_{12})$ … $P(x_{50,00}|w_{12})$: $w_{12}x_{25694}$ $w_{12}x_{35694}$

w_{13}: $P(x_1|w_{13})$ … $P(x_{50,00}|w_{13})$: $w_{13}x_{43669}$

Step3 (Based on step 2 output w_{21} , w_{22}, w_{23})

w_{11}: $P(x_1|w_{21})$ … $P(x_{50,00}|w_{21})$

w_{12}: $P(x_1|w_{22})$ … $P(x_{50,00}|w_{22})$

w_{13}: $P(x_1|w_{33})$ … $P(x_{50,00}|w_{33})$

Output: w_{31} , w_{32}, w_{33}

Mobile App: Cat vs Non-Cat

- Observation: 90% accuracy, app mistakes dogs for cats
- What to do?
  - Get a sample of 100 misclassified dev examples
  - Count what fraction of these are dogs
- Finding: 50% of misclassified images are dogs
- Best expected?

Mobile App: Cat vs Non-Cat

- Observation: 90% accuracy, app mistakes dogs for cats
- What to do?
  - Get a sample of 100 misclassified dev examples
  - Count what fraction of these are dogs
- Finding: 50% of misclassified images are dogs
- Best expected? 50% of 10 is 5
  - Accuracy moves up from 90 % to 95%