Lecture 1
7th July2020- 1-5PM MTechDS SemIII

1. Syllabus: https://upscfever.com/upsc-fever/en/data/en-data-home.html (Course 3-7)
2. Books:
   Deep Learning, by Ian Goodfellow, Yoshua Bengio, Aaron Courville (e-book)
3. Online courses: Deep learning specialization by Andrew NG on coursera.org

Term work: 30 marks

1. 10 marks - Research paper (Team - max 2) - Target conferences: IIT
2. 10 marks - library/package development (Team - max 2) - http://r-pkgs.had.co.nz/
3. 10 marks - Lab and theory assignments

20 marks - ICA 1 and ICA 2

Background:
Writing a research paper - 10 marks
Team of max 2 people

1- ORCID - https://orcid.org/ (Create an account)
2- Google scholar
3- Researchgate
4-Github
5-https://www.scopus.com/home.uri (SCOPUS) and Science citation index (SCI)
https://publons.com/about/home/
6- overleaf.com (template) in latex

Identify conference
guide2research.com

Identify journals
https://www.scimagojr.com

Parts of the paper

1. Project template: Word https://www.ieee.org/conferences/publishing/templates.html or latex -use overleaf.com (IEEE conf template)
2. Paper [Page limit- 4/6]
    1. Title: Long and short version (Aspect based sentiment analysis for product recommendations)
    2. Abstract [125 - 150 word limit]
        1. Background - What is aspect based sentiment analysis (1 sentence)
        2. Objectives - To increase the accuracy of product recommendation systems
        3. Methods - Where is the dataset? Which techniques are used to solve the problem?
        4. Results - Compared with existing techniques the proposed model achieves ..... increase / decrease in performance metrics
        5. Conclusion - The proposed model was found appropraite / not found appropriate on .....
    3. Introduction [1 and half column]
        1. Background
        2. Motivation
        3. Summary of literature
        4. Contribution
        5. Outline of paper
    4. Literature survey [cover between 10-60 research papers from IEEE/ACM/Elsevier/Sciencedirect/Springer]
        1. State of the art - describe briefly the research papers and work done in them
        2. Research gaps - what problems did you find in the research papers
        3. Scope - which research gap is the focus of current paper
    5. Mathematical model
        1. mathematically represent your proposed model
        2. flow chart
        3. algorithm
    6. Experimental study
        1. Dataset
        2. metrics for performance
        3. environment
    7. Results and discussion
        1. results - compare with an existing baseline technique
        2. discussion and analysis
        3. summary

8. Conclusion
    1. future scope
9. References

July end - Introduction and literature survey
August end- Till Results and discussion
September end- Submit to conference

Background:

1. What is machine learning - Past data, Experience, Data points, model, optimize loss, convergence, training data, development set, test set
2. Experience, Performance,Task - EPT

Steps of solving a problem with ML (Research paper = Proj + Doc)

1. Define problem statement
2. Review solutions proposed by others
3. Decide strategy
4. Data collection
5. Wrangling
6. Exploratory data analysis / visualizations
7. Split train-dev-test
8. Train model on train set, validate on dev and evaluate on test
9. Performing experiments
10. Discussing results
11. Conclusion

Types of Data

1. Structured data -
    a. label (y/class/groundtruth/response variables/ dependent variables) and predictors (x/independent variables): rows and columns
    b. No labels with predictors (independent variables)
2. Unstructured data-
    a. label (response variables/ dependent variables) - audio/video/text/images
    b. No label (response variables/ dependent variables) - audio/video/text/images
3. Semi-structured data? - https://community.tealiumiq.com/t5/Customer-Data-Hub/Structured-Data-vs-Semi-Structured-Data/ta-p/15617

Limitations of ML

1. Handcrafting features
2. Dependence on ability of researcher

Deep learning v.s Machine Learning

1. Data driven features
2. Data -> features -> model -> classification (ML) and Data -> model -> classification (DL)
3. Limiting interpretability
4. Overcame brittleness of handcrafted features
5. Volume of data, complexity of models increased

ML : y^ = f(x)

y^ (Prediction) = X1W1 + X2W2 + ….. + XnWn + C
Where Parameters: W = weights C = intercept
X = features
f(.) = model

Y = price of house
X = area of house

Y= wx+c

Y = 3.5Area of house + 2

Exercises 1: Background -
https://colab.research.google.com/drive/1gM8vwiXAt4Ao-H-LlOIazB3z2Yqqf4gz

a. Introduction and Background.
    a. Import dataset store it as dataframe in python: `filename = "https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/auto.csv"`
    b. Add column headers to dataset
    c. Read the first 5 lines and display
    d. Are there missing data? Or illegal characters in the dataframe?
    e. Replace the character “?” with nan
    f. Count missing values in each column and print it with column name
        i. "normalized-losses": 41 missing data
        ii. "num-of-doors": 2 missing data
        iii. "bore": 4 missing data
        iv. "stroke" : 4 missing data
        v. "horsepower": 2 missing data
        vi. "peak-rpm": 2 missing data
        vii. "price": 4 missing data (Response)

g. Delete price rows that have missing data
h. Normalized losses,bore,stroke,horsepower,peak-rpm, - replace missing with mean of the column
i. Num-of-doors replace missing with most frequent value in the column
j. Reset the index of dataframe
k. Check datatype of columns and convert numeric/quantitative variables to float or int
l. Transform city-mpg and highway-mpg into liters/100km using conversion formula: L/100km = 235/mpg i.e. create two new column “city-L/100km” and “highway-L/100km”
m. Normalize columns length, width, height so that their values range from 0 to 1. Hint: Replace original values with original_value/max_value
m. Plot the histogram of horsepower to see its distribution
n. Create three equal sized bins “low”, “medium”, “high” and organize values in column horsepower into new column “horsepower-binned”
o. Plot distribution of “horsepower-binned”
p. Convert “fuel-type” into one-hot-encoded variables. Repeat same for “aspiration” and then drop columns “fuel-type” and “aspiration”

END - 10 mins

Exercise -2: Exploring variables
https://colab.research.google.com/drive/1gtU7GZEuUspZMHdhuh7rO7iWxm7YZAcU

1. Import the dataset:
   `path='https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/automobileEDA.csv'`
2. Import matplotlib, seaborn, numpy and pandas
3. See dimensions of data frame and its data types for each column
4. Calculate correlation between engine-size and price using corr function
5. Identify variables with positive or negative correlation with price
6. Identify datatype of “peak-rpm”
7. Using seaborn regplot() - plot relation between “engine-size” and “price”. Comment on your observation.
8. Identify using regplot() - which other variables can affect “price” and which do not affect it.
9. Use seaborn pairplot() to identify which variables can affect “price”
10. Draw a heatmap to plot the correlation in the dataframe
11. With seaborn boxplot() - compare “body-style” with “price”
12. Continue for other categorical variables in the dataset.
13. What do you infer from the boxplots about the relationship between the variables.
14. Use describe() to get descriptive statistics of numeric variables
15. Use describe() to get stats of categorical variables

16. Get unique values in each categorical variable along with their frequency. What do you understand by doing this?
17. Use groupby() to get the average price of “drive-wheels” wrt “price”. What do you understand by doing this?
18. Repeat step 17 for other categorical variables.
19. Use groupby() to find the average price for “drive-wheels”and”body-style” with price. Observation? Inference?
20. Use pivot() on the result of step 19 to get “drive-wheels” as index and “body-style” as columns. Observations? Inference?
21. Repeat step 19and20 for other combinations of independent variables wrt price. Observations? Inferences?
22. Draw heatmap for result of step20
23. Calculate the pearson correlation between “wheel-base” and “price”. What can you conclude from p-value (Hint: use stats from scipy which has pearsonr())
24. Perform one way ANOVA test using f_oneway() of stats to check if different groups of “drive-wheels” are correlated with “price”. What do you understand from F-test and p-value results?

END

Lecture 2
14th July2020- 1-5PM MTechDS SemIII

Notes:
https://docs.google.com/document/d/18JYvbD4L5NhQ8RYSq1xbLVL0bTISk8-0PYF8L3QQ7SY

Topic: Model development and data visualization

Exercise 1:
https://colab.research.google.com/drive/1yPUeIqjR_UImgf80BboZDicUQsaB73Ci

Topics:

1. Linear models
2. Regplot
3. Residual plots
4. Multivariate linear models
5. Polynomial models
6. Pipelining
7. R^2 - coefficient of determination
8. RMSE-Squared error

Exercise 2:
https://colab.research.google.com/drive/1sdE5oV7Fys89WsF31eFwclj4TEeE5GeE

Topics:

1. Model evaluation
2. Best fit
3. Regularize
4. Grid search

Exercise 3:
https://colab.research.google.com/drive/13HJawtK_mkspMFNoaw0qc2RWU0vWe8_r

Topics:

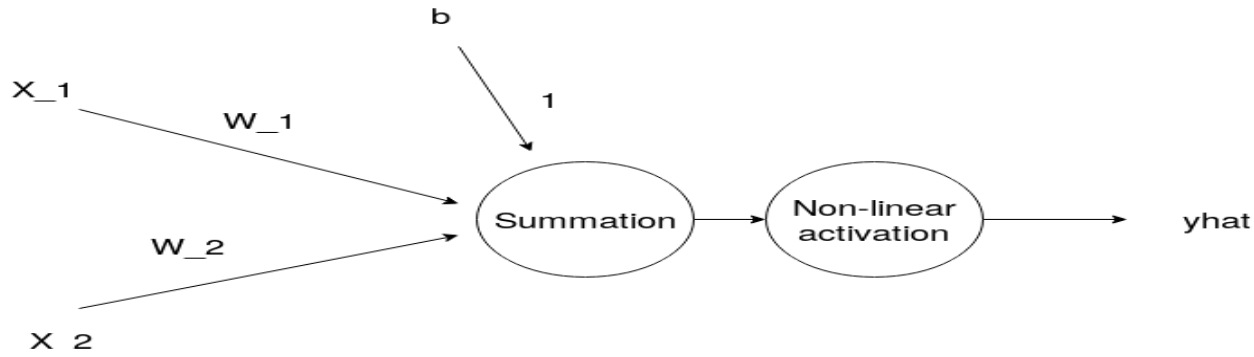
1. Hyper-parameters
2. Grid search and randomized search

What is Deep learning?

1. Deep learning is training of large neural networks

Neural networks?

1. Extension of logistic regression - a very simple neural network



Loss function
L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2

Training a neural network - just a logistic regression

1. Initialized W, b
2. Forward propagation
    a. Passed the training set and calculated loss of each observation in training set (L)
    b. Summed the losses of all observations into a total cost (J)
3. Backward propagation
    a. Calculate the derivatives wrt to parameters (dJ/dW, dJ/db)
    b. Get new values of parameters (W_new, b_new) by subtracting the old parameters (W_old, b_old) with derivatives (dJ/dW, dJ/db)
4. Step 2 and 3 to be continued till Cost of current iteration (J^(i)) is not less than cost of previous iterations (J^(i-1)) OR till a user criteria is met (no. of iterations (epochs) are over.. etc.)

Equations of the Neural network:

$$Z = \sum_{i=1}^{n} W_i X_i + b$$
$$\hat{y} = sigmoid(Z)$$
$$cross - entropyloss$$
$$binaryorcategorical$$
$$L(\hat{y}, y) = -ylog(\hat{y}) - (1-y)log(1-\hat{y})$$
$$J(w, b) = -\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$
$$\frac{\partial J}{\partial W} = \frac{1}{m} X(A - Y)^T$$
$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)$$
$$W_{new} = W_{old} - \alpha\frac{\partial J}{\partial W}$$
$$b_{new} = b_{old} - \alpha\frac{\partial J}{\partial b}$$

Exercise : 1
https://colab.research.google.com/drive/1fwMwtOQP0PmFD7PlZV3TR1xOzjzBelQF#scrollTo=5CNhRFg0gGY_

Lecture 3
21st July2020- 1-5PM MTechDS SemIII

Sigmoid

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

sigmoid(x) = \frac{1}{1+e^{-x}}

Notebook:
https://colab.research.google.com/drive/1tF4DLF1m_7CRfkMc14e8VSeG1g-Lc82a

From logistic regression to neural networks

Forward propagation

$$\begin{aligned} z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= sigmoid(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= sigmoid(Z^{[2]}) \\ z^{[3]} &= W^{[3]}A^{[2]} + b^{[3]} \\ \hat{y} &= A^{[3]} = sigmoid(Z^{[3]}) \end{aligned}$$

Computing the loss and the cost (Cross-entropy loss / bernoullis cross entropy loss)

$$L(\hat{y}, y) = -ylog(\hat{y}) - (1-y)log(1-\hat{y})$$

m=observations in the datasets

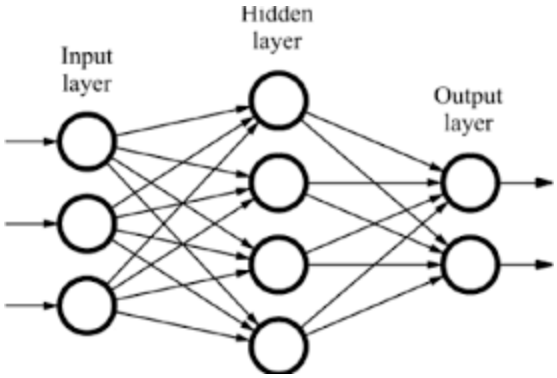
$$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i)$$

Back propagation of gradients

$$W_{new} = W_{old} - \alpha \frac{\partial J}{\partial W}$$

$$b_{new} = b_{old} - \alpha \frac{\partial J}{\partial b}$$

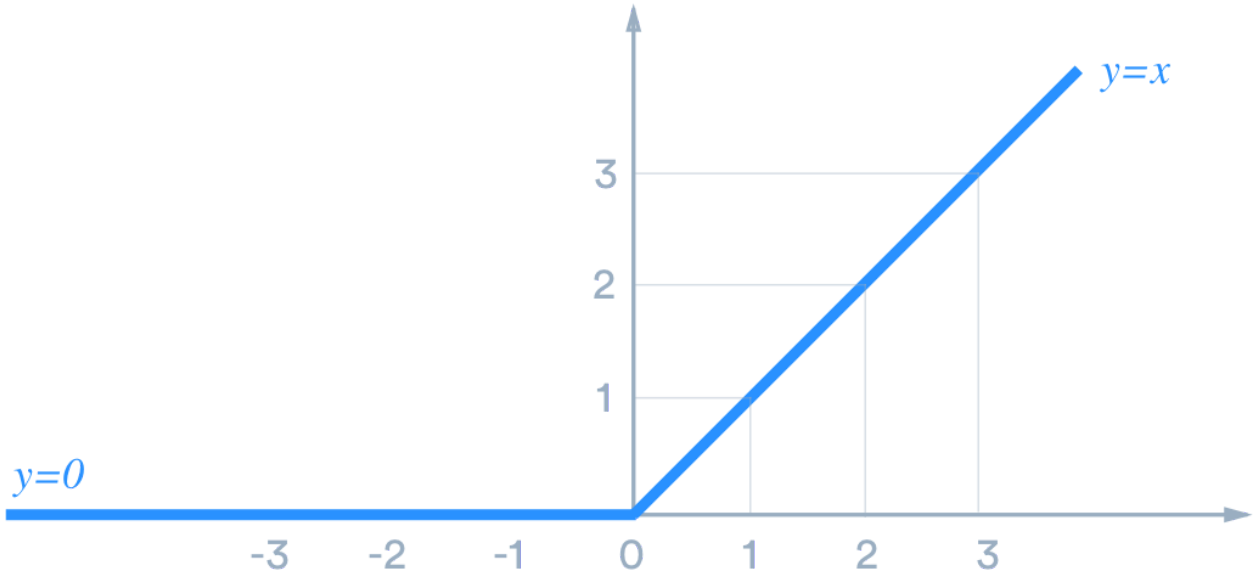
Neural network building blocks

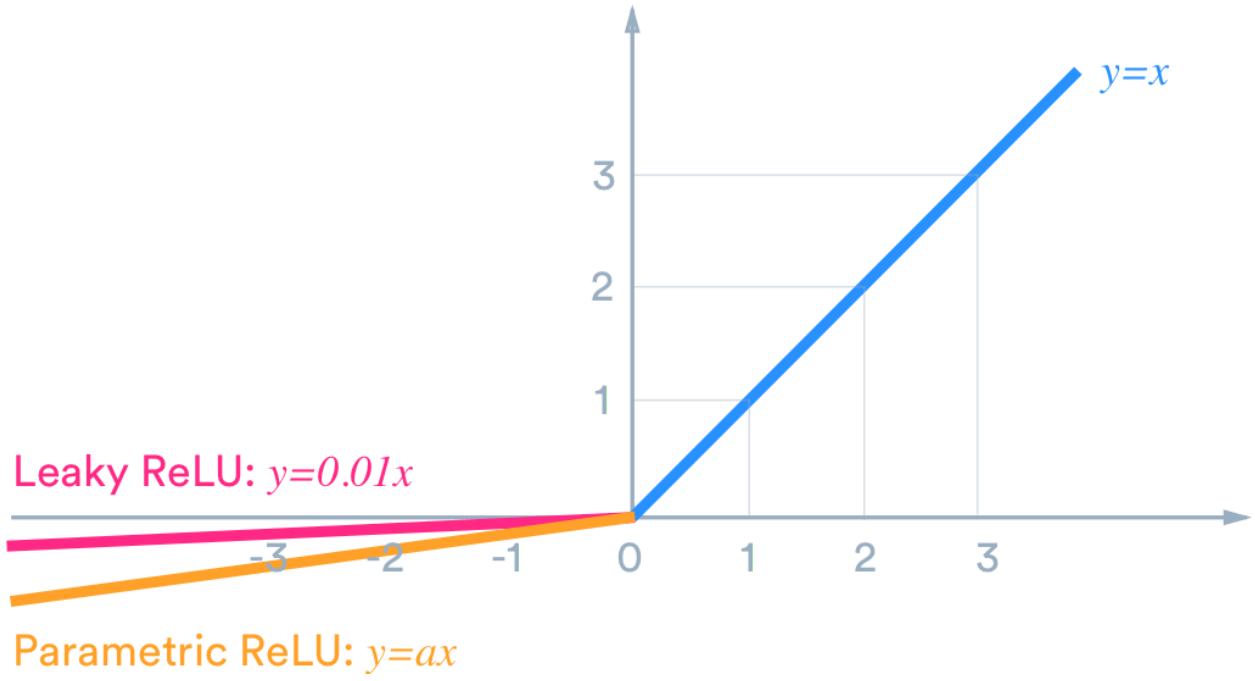
1. Number of nodes - ranges 1/3rd size of input to size of input
2. How many hidden layers - start from 1 and keep adding one
3. Deep learning - no feature engineering
4. 

5. Keras model building blocks:
    a. Dense - units, input_dim, activation
    b. Compile - loss, metrics, optimizer
    c. Add
    d. Fit - x, y, validation_split
    e. Plot - val_accuracy vs accuracy , val_loss vs loss
    f. Summary
    g. SVG

6. Innovations in Deep learning
    a. Layers
    b. Activations
    c. Data generations
    d. Optimizations
    e. Loss
    f. Metrics
    g. Iterating over the dataset
    h. Divide into train-dev-test
    i. Sampling the dataset

Thumb rule in neural network architectures

1. Use softmax for final layer when more than two classes are there in data
2. Use sigmoid for final layer when more than two classes are there in data
3. Tanh is activation superior to sigmoid in most cases for layers before the output (symmetric around zero)
4. ReLU (max(0,z)) for all layers other than output and softmax (class>3)/sigmoid(class=2) for output layers - Baseline*******
5. LeakyReLU (max(0.01*z,z)) for all layers other than output and softmax (class>3)/sigmoid(class=2) for output layers
6. SeLU (max(a*z,z)) where scalar 'a' is trainable
7. 



Initialization is keras for W (kernel) and b (bias)

1. Never initialize both to zeros

Assg -3

1. Which initialization is suitable for a Monsoon dataset?
2. Model for cifar10 which gives 70-75% val_acc. (hyper-parameters)

Lecture 4
28st July2020- 1-5PM MTechDS SemIII

Innovations in Deep learning

j. Layers
k. Activations
l. Data generations
m. Optimizations
n. Loss
o. Metrics
p. Iterating over the dataset
q. Divide into train-dev-test
r. Sampling the dataset

Different initializers for kernel ‘W’ and bias ‘b’

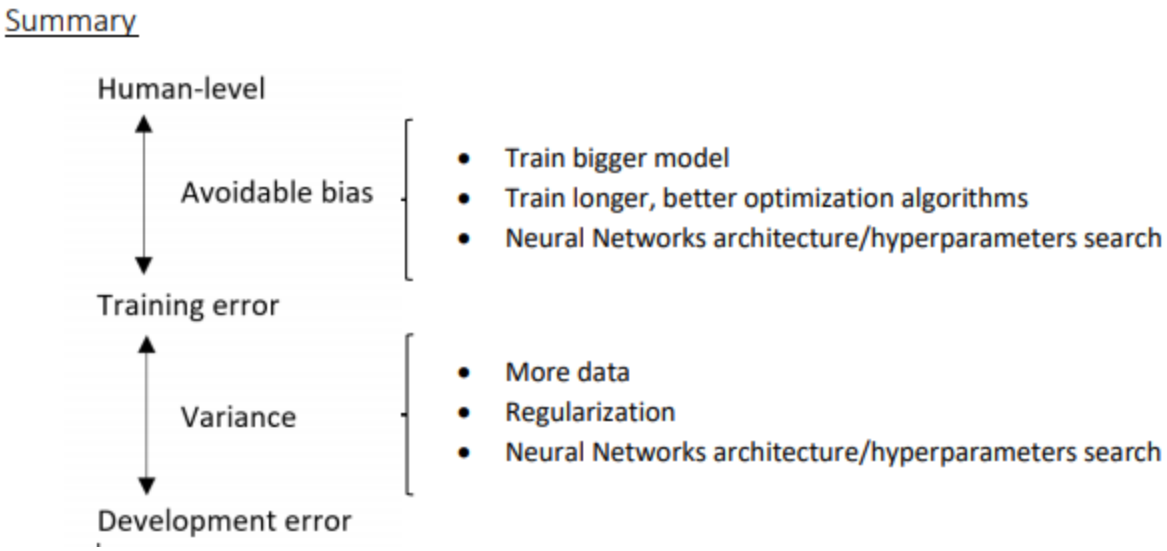
1. In keras we have kernel_initializer and bias_initializer to set these arguments
2. To initialize W and b to small values for initial iteration
3. Types:
    a. Random_uniform
    b. Zeros
    c. Ones
    d. Constant
    e. Random normal
    f. Random uniform with cutoff for upper and lower bounds
    g. Truncated normal
    h. Variance scaling (popular)
    i. Orthogonal
    j. Identity
    k. Lecun_uniform (popular)
    l. `Glorot_normal (popular starting point)`
    m. Glorot_uniform (popular)
    n. He_normal (classical)
    o. Lecun_normal(popular)

p. he_uniform(classical)

Dataset

1. CIFAR10 - 50000 (32*32*3), 10 animals, 10000 test images (Canadian Institute For Advanced Research)
2. CIFAR100 - 50000 (32*32*3), 100 animals, 10000 test images (Canadian Institute For Advanced Research)
3. MNIST (Modified National Institute of Standards and Technology") - 60000 (28*28*1) 10 handwritten numbers, 10000 test images
4. Fashion MNIST - 60000 (28*28*1) 10 fashion categories, 10000 test images

Summary

1. First ask how much will a human level or bayes optimal error on the dataset (Assumptions)
2. Bigger models means more depth or more layers - add layers one at a time
3. Hyperparameters:
    a. Deciding layers, units, initializers, optimizers, loss, metrics, epochs, validation split
4. 

Notebook
https://colab.research.google.com/drive/1auFpITv2drTFLG6KuxOIXiZbd15xWNP1

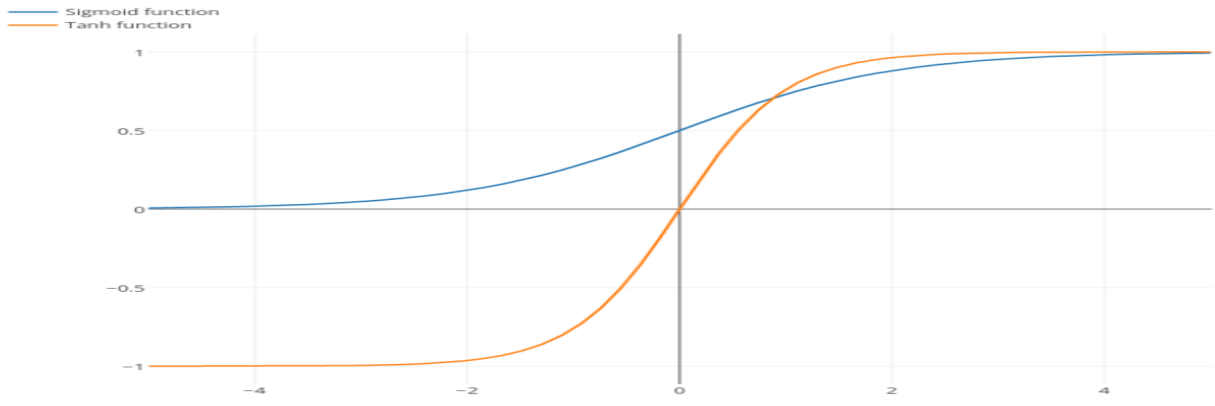
Assignment:

1. Build model for CIFAR100, MNIST, Fashion MNIST (test data - accuracy of +80%) - plot metrics accuracy, sparse_cat_acc, top_k, categorical_acc, sparse_topk
2. Use activation as layers in the model
3. EarlyStopping and ModelCheckPoint

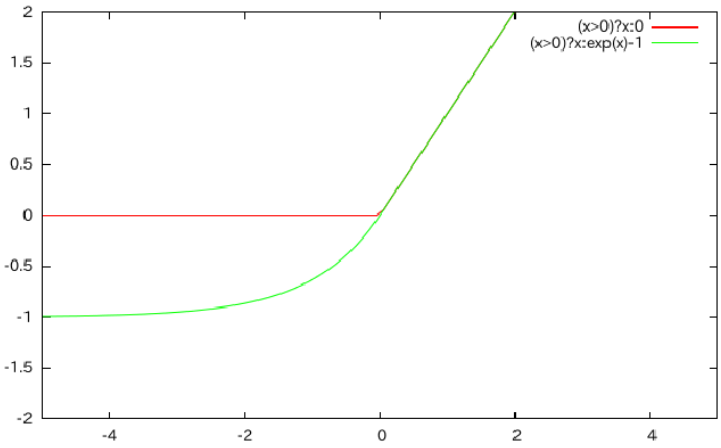
Dense Layers

1. Units: positive int
2. Activation
3. Input_dim / input_shape
4. Use_bias: boolean
5. Kernel_initializer: string/object
6. Bias_initializer: string/object
7. Kernel_regularizer: string/object
8. Bias_regularizer:string/object
9. Activity_regularizers: (applied to the output of a layer)
10. Kernel_constraint:string/object
11. Bias_constraint: string/object

Activation:

1. Relu for all layers other than output
2. Output layer - softmax (multiclass - class>2), sigmoid (binary/multilabel)
3. 

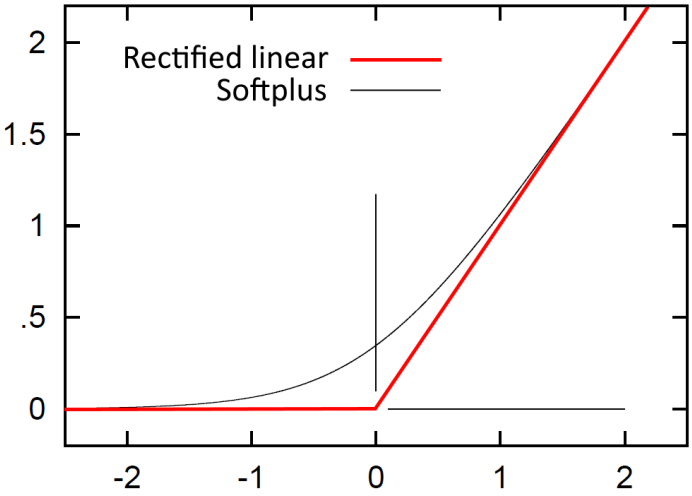
4. Types of activations
    a. Elu x>0? X : alpha * (exp(x)-1)

b. 

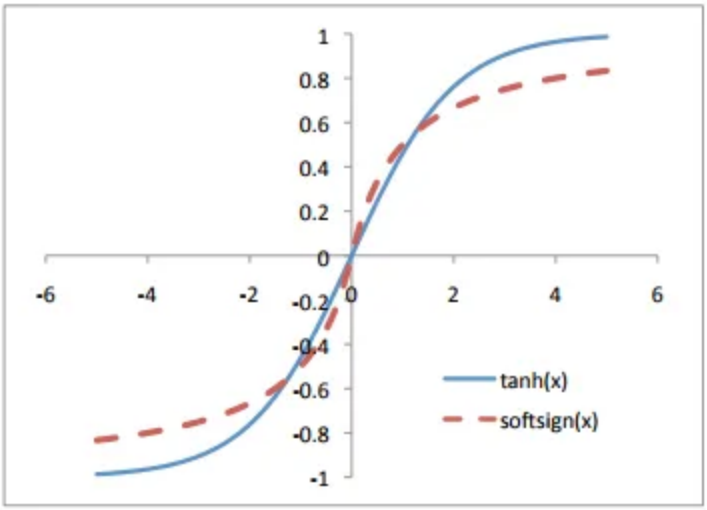
c. Softmax

d. Selu (scaled exponential unit) - scale * elu() - used with initialization lecun_normal and dropout alphadropout

e. Softplus - log(exp(x) - 1)



f. Softsign: x / (abs(x)+1)

g. 

h. Relu
i. Tanh
j. Sigmoid
k. Hard_sigmoid: (faster to compute than sigmoid)
    i. 0 if x<-2.5
    ii. 1 if x>2.5
    iii. -2.5 < x < 2.5 then 0.2*x + 0.5
l. Exponential
m. Linear
n. LeakyRELU
    i. x<0 we use alpha*x
    ii. x>=0 we use x
o. PReLU (parametric relu)
    i. alpha*x for x<0
    ii. X for x>=0

Metrics (Plot performance with epochs)

1. Decide whether to train further or stop
2. Types
    a. Regression - mae, mse
    b. Classification - mae, mse, accuracy [popular]
        i. Binary_accuracy
        ii. Categorical_accuracy
        iii. Sparse_categorical_accuracy [label encoded - response variable, to be used with loss function 'sparse categorical cross entropy']
        iv. Top_k_categorical_accuracy
        v. Sparse_top_k_categorical_accuracy
        vi. Cosine_proximity

Loss

1. Regression:
    a. 'Mean_squared_error' (popular)
    b. Mean_squared_logarithmic_error
    c. Mean_absolute_percentage_error
    d. Mean_absolute_error (second popular)
2. Classification:
    a. 'categorical_crossentropy' [multiclass i.e. class>2] and 'binary_crossentropy' [class = 2 or multilabel] (popular)
    b. 'Cosine_proximity' (word embeddings)
    c. 'Poisson'
    d. Kullback_leibler_divergence' (autoencoder)
    e. Logcosh

f. Categorical_hinge
g. Hinge
h. Squared_hinge

Callbacks (stopping criteria)

1. From keras.callbacks import EarlyStopping, ModelCheckPoint
2. EarlyStopping
3. ModelCheckPoint
4. Callbacks
5. load_weights()
6. load_model()

Lecture 5
11 August 2020- 1-5PM MTechDS SemIII

Topics

1. Data generators
2. Optimizers
3. Imagenet
4. Flow from directory
5. Flow from dataframe
6. Multilabel classification

Notebook

https://colab.research.google.com/drive/1WE1DgU09gbHMXxOU8ImS2QD1UWxBiTPf

Optimizers

1. Converge to the right values of weights (W, b)
2. Gradient descent - Tried and tested, requires a pass over the entire dataset for small change to W, b
3. Innovations focus on converging to right values of W and b, with minimum time without passing over entire dataset
4. List
    a. Gradient descent (classical - dataset is small)
    b. Stochastic GD
    c. Mini batch SGD (more popular)
    d. RMSProp
    e. Adagrad
    f. Adadelta
    g. Adam (new, recommend - Andrew NG and Ian goodfellow)
    h. Adamax
    i. Nadam

5. Good starting point - small datasets use Gradient descent and adjust learning rate, Big datasets we use adam and adjust learning rate

Assignment

1. Multiclass classification on UCMErced dataset
2. Data augmentations - find best practices

Multilabel classification

1. Each image can have multiple labels.
2. We use sigmoid as the activation for output layer
3. We use binary_crossentropy as loss function

Data augmentation

1. Data augmentation is used we have very little data
2. Reduce overfitting
3. Robust model
4. Accuracy on the train data will decrease. Model building is more challenging
5. Kaggle Competitions
6. Types of data augmentations
    a. Randomly Rotate Images by a degree
    b. Vertical flip
    c. Shifting horizontally by a percentage
    d. Shifting vertically by a percentage
    e. Shifting both horizontally and vertically by a percentage
    f. Brightness shifting
    g. Horizontal flip
    h. Zoom
    i. Scale
    j. fillmode= constant, nearest
    k. Shearing (displacement)
7. Mathematical data augmentation
    a. Featurewise Center - mean to 0
    b. Samplewise Center - mean to 0
    c. Featurewise normalization - divide each pixel by standard deviation
    d. Samplewise normalization - divide each pixel by standard deviation
    e. ZCA whitening
    f. ZCA epsilon

To save augmented images for later use

```
save_to_dir='images', save_prefix='aug', save_format='png'
```

How to use data augmentations with model building

1. Build a baseline model
2. Import ImageDataGenerator
3. Create an ImageDataGenerator object for train, validation and test
4. Fit object to data
5. Use fit_generator() for training, datagen.flow() to pass training data

Lecture 6
12 August 2020- 530-730PM MTechDS SemIII

Notebook

Flow from dataframe
https://colab.research.google.com/drive/1m3OW0y7OBYpoJxfTJOHPFTYksZ5JkeXk

Flow from directory
https://colab.research.google.com/drive/1ufmzlmZBL-hrGmL8-dGi8tF-er0EPpyv

Topic

1. Imagenet
2. Flow from directory
3. Flow from dataframe

How to get Images from ImageNet with Python in Google Colaboratory

The first step to train a model for image recognition is finding images that belong to the desired class (or classes), and ImageNet is very useful for this because it currently has 14,197,122 images with 21841 synsets indexed. ImageNet aims to provide on average 1000 images to illustrate each one of their 100,000 synsets, the majority of the synsets are nouns (80.000+).

For instance if the synset needed is pictures of ships it can be found by searching for ship on the imagenet website and the result will be the following page which has the wnid: n04194289 Get the list of URLs for the images of the synset:

Said list of URLs can be downloaded from the URL
http://www.image-net.org/api/text/imagenet.synset.geturls?wnid= followed by the wnid so in the case of ships it would be
“http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n04194289" this can be done with the Python library BeautifulSoup

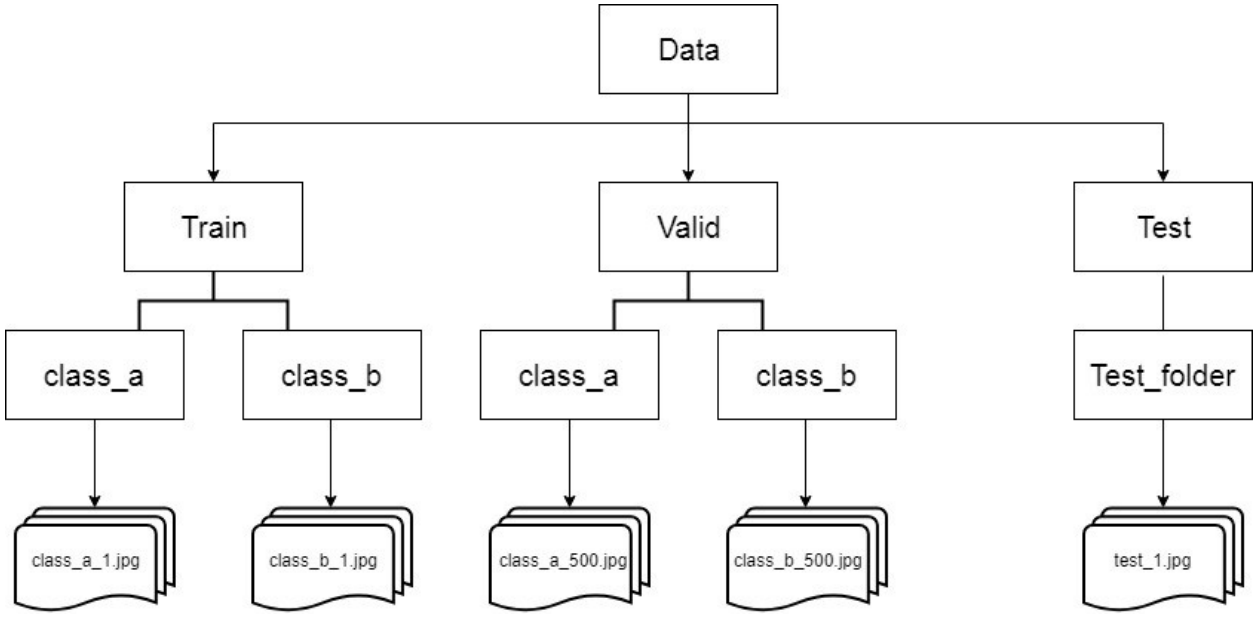
1. flow_from_dataframe(): when our dataset has all images in one folder for train and another folder for test
2. flow_from_directory():
3. 

Figure : 1

Download from imagenet

1. Find wnid of synset of your choice
2. Call the library function

Point to remember

1. Organize your data as all images in one folder for train and another folder for test
2. Organize your data as Figure 1

Summarize data augmentation

1. Needed when overfitting, robust model, competitions, having less data
2. Challenging as model building become difficult
3. flow() - increase the images in your dataset. Save augmented images for further use. Using inbuilt datasets
4. flow_from_directory(): Most popular method
5. flow_from_dataframe(): Second most popular method
6. Caution: for images you need one conv2d layers at beginning as generators dont give flatten output
7. Have separate datagen objects for train, test
8. Create generators for train, val and test
9. Use fit_generator(), evaluate_generator() and test_generator()
10. Before using test_generator() remember to reset it

11. Choose augmentations by testing on your data
12. While creating traingenerators and validation generators use classmode, colormode, shuffle, seed, target but dont shuffle test generators and use classmode=none in test generator

Assignment-5 [19/8/2020]

1. Google open images / imagenet for model building - classifier on any three classes (synsets) of cars [any one of two flow from functions]

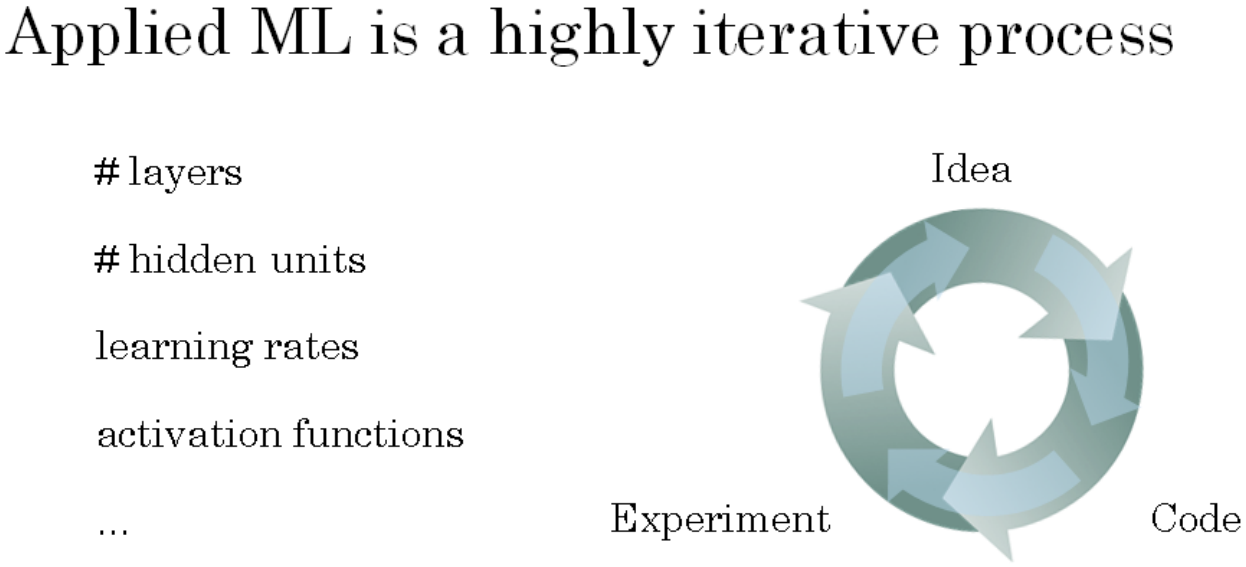
Lecture 7
19 August 2020- 1230-430PM MTechDS SemIII

Topics:

1. Best practices in ML projects
2. Hyper-parameter Tuning
3. Improving ML models performance

Reference:

1. Ian Goodfellows book
2. Andrew NG Deep learning course



So in the previous era of machine learning, it was common practice to take all your data and split it according to a 70/30% in terms of a people often talk about the 70/30 train test splits. If

you don't have an explicit dev set or maybe a **60/20/20% split in terms of 60% train, 20% dev and 20% test.**

And several years ago this was widely considered best practice in machine learning.

If you have maybe 100 examples in total, maybe 1000 examples in total, maybe after 10,000 examples. These sorts of ratios were perfectly reasonable rules of thumb.

But in the modern big data era, where, for example, you might have a million examples in total, then the trend is that your dev and test sets have been becoming a much smaller percentage of the total.

Because remember, the goal of the dev set or the development set is that you're going to test different algorithms on it and see which algorithm works better.

So the dev set just needs to be big enough for you to evaluate, say, two different algorithm choices or ten different algorithm choices and quickly decide which one is doing better.

And you might not need a whole 20% of your data for that. So, for example, if you have a million training examples you might decide that just having 10,000 examples in your dev set is more than enough to evaluate which one or two algorithms does better.

And in a similar vein, the main goal of your test set is, given your final classifier, to give you a pretty confident estimate of how well it's doing.

And again, if you have a million examples maybe you might decide that 10,000 examples is more than enough in order to evaluate a single classifier and give you a good estimate of how well it's doing.

So in this example where you have a million examples, if you need just 10,000 for your dev and 10,000 for your test, your ratio will be more like this 10,000 is 1% of **1 million** so you'll have **98% train, 1% dev, 1% test.**

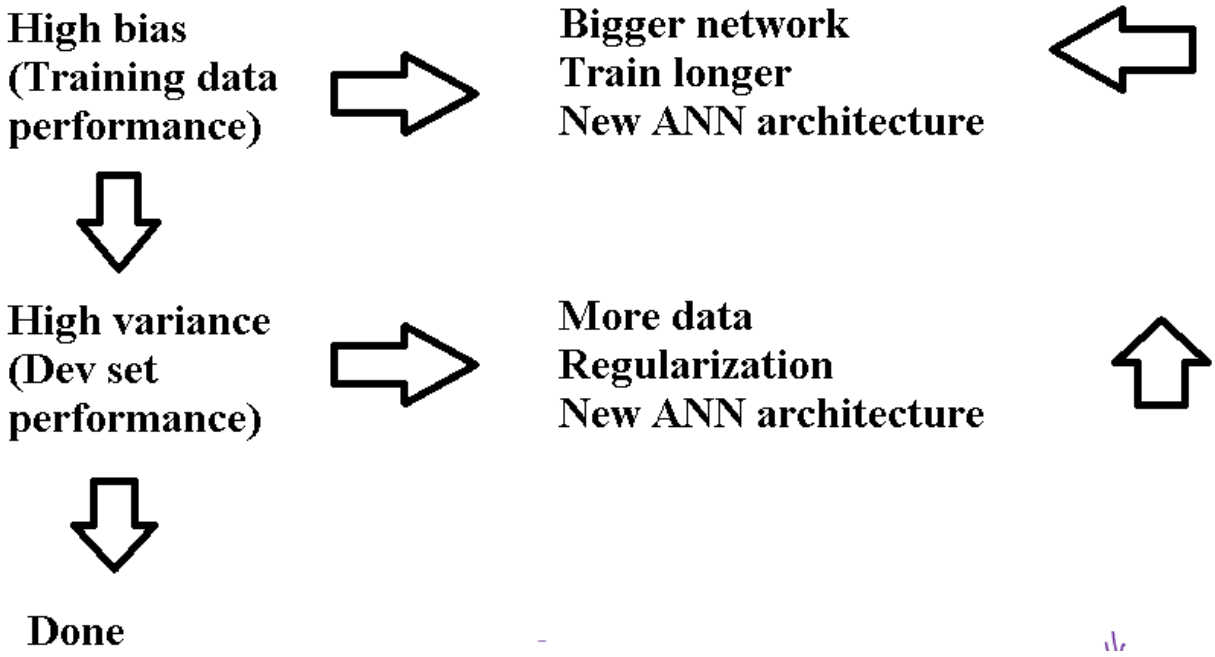
And if you have even more than a million examples, you might end up with 99.5% train and 0.25% dev, 0.25% test. Or maybe a 0.4% dev, 0.1% test.

Bias and Variance

Cat classification


Train set error: Dev set error	1% 11%	15% 16%	15% 30%	0.5% 1%
	High variance	High bias	High bias High variance	Low bias Low variance

Basic recipe for machine learning



Regularizing your neural network

If you suspect your neural network is over fitting your data. That is you have a high variance problem, one of the first things you should try per probably regularization.

The other way to address high variance, is to get more training data that's also quite reliable.

But you can't always get more training data, or it could be expensive to get more data.

But adding regularization will often help to prevent overfitting, or to reduce the errors in your network.

L2 regularization is weight decay.

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{y}^{(i)}) - (1-y^{(i)})log(1-\hat{y}^{(i)})]$$

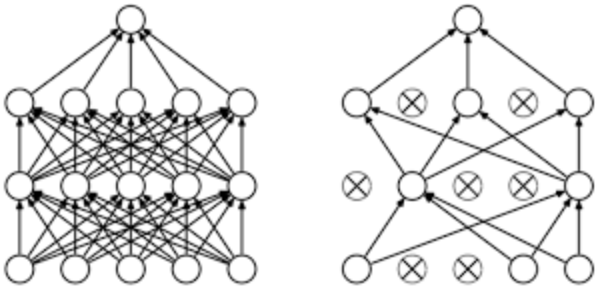
L2 Regularization term (**Frobenius norm**)

$$\frac{\lambda}{2m}||w||_2^2.$$

L1 Regularization term

$$\frac{\lambda}{2m}||w||_1.$$

Dropout



(a) Standard Neural Net (b) After applying dropout.

Additionally, as recommended in the original paper on Dropout, a constraint is imposed on the weights for each hidden layer, **ensuring that the maximum norm of the weights does not exceed a value of 3**. This is done by setting the kernel_constraint argument on the Dense class when constructing the layers.

The **learning rate was lifted by one order of magnitude and the momentum was increase to 0.9**. These increases in the learning rate were also recommended in the original Dropout paper.

Paper by Geoffrey Hilton

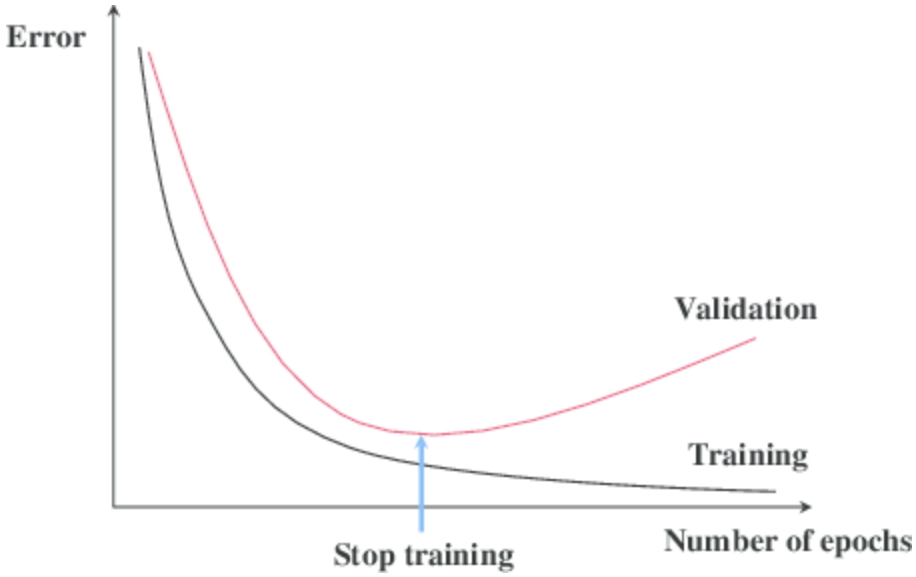
Input layers

```
model = Sequential()
model.add(Dropout(0.2, input_shape=(60,)))
model.add(Dense(60, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dense(30, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dense(1, activation='sigmoid'))
# Compile model
sgd = SGD(lr=0.1, momentum=0.9)
```

Hidden layers

```
model = Sequential()
model.add(Dense(60, input_dim=60, activation='relu', kernel_constraint=maxnorm(3)
model.add(Dropout(0.2))
model.add(Dense(30, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
# Compile model
sgd = SGD(lr=0.1, momentum=0.9)
```

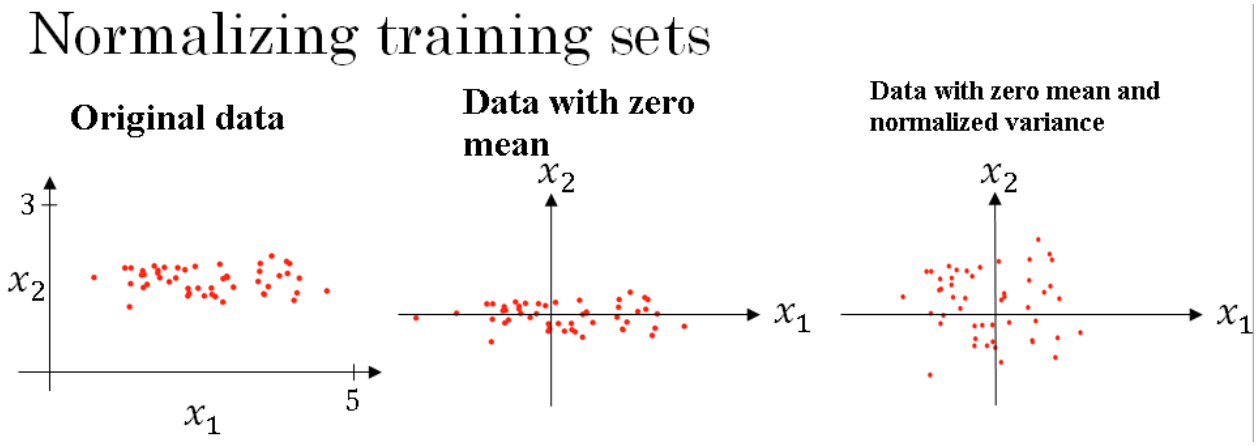
Early stopping and Data augmentation



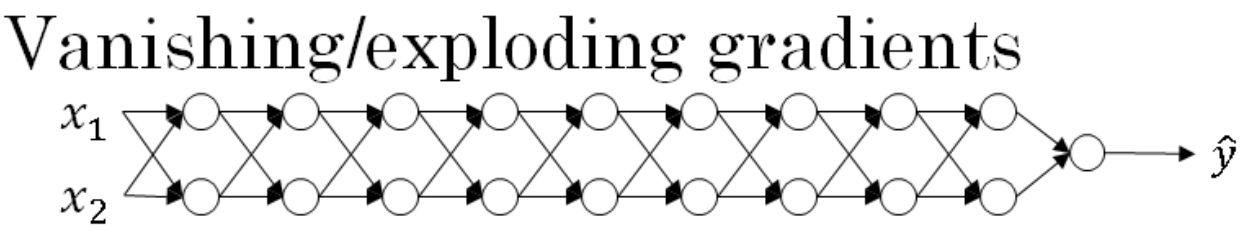
Normalizing inputs

When training a neural network, one of the techniques **that will speed up your training** is if you normalize your inputs.

Images =/255.0



Vanishing and exploding gradients



Forward pass

$$Z^{[1]} = W^{[[1]]}X^{[0]}$$
$$A^{[1]} = g(Z^{[1]}) = Z^{[1]}$$
$$Z^{[2]} = W^{[[2]]}A^{[1]}$$
$$A^{[2]} = g(Z^{[2]}) = Z^{[2]}$$
$$..$$
$$..$$
$$..$$
$$Z^{[L-1]} = W^{[[L-1]]}A^{[L-2]}$$
$$A^{[L-1]} = g(Z^{[L-1]}) = Z^{[L-1]}$$
$$Z^{[L]} = W^{[[L]]}A^{[L-1]}$$
$$\hat{y} = A^{[L]} = g(Z^{[L]}) = Z^{[L]}$$

Assume linear activation function

$$\hat{y} = W^{[[L]]} * W^{[[L-1]]} * W^{[[L-2]]} * \ldots * W^{[[1]]}X$$

Exploding gradients

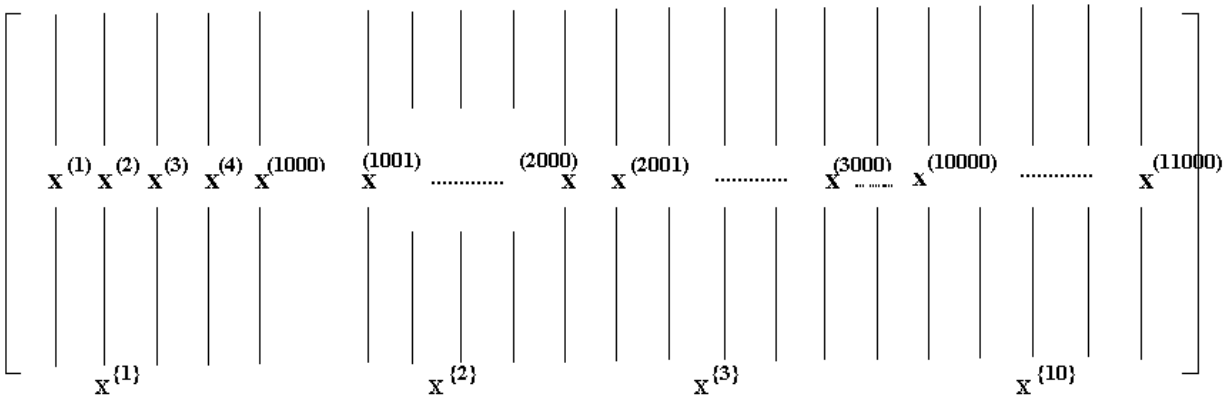
$$\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L}$$

Vanishing gradients

$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L}$$

It turns out there's a partial solution that doesn't completely solve this problem but it helps a lot which is **careful choice of how you initialize the weights**

Mini batch gradient descent



repeat

{

for t = 1, ..., 5000

{

Forward propagation on $X^{\{t\}}$

$Z^{[1]} = W^{[1]}X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

.

. .

.

$A^{[L]} = g^{[L]}(Z^{[L]})$

Compute cost on $X^{\{t\}}$ as $J^{\{t\}}$

Backpropagation to compute gradients

Update W and b

} // after completion of loop 1 pass over training set over i.e. 1 epoch

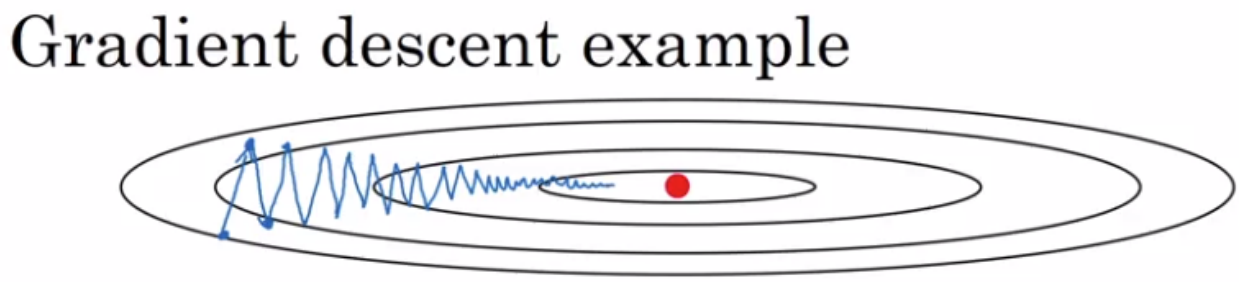
} // for multiple iterations over training set

Now **one of the parameters you need to choose is the size of your mini batch**. So m was the training set size on one extreme, if the mini-batch **size, = m**, then you just end up with **batch gradient descent.**

Alright, so in this extreme you would just have one mini-batch X1,Y1 , and this mini-batch is equal to your entire training set. So setting a mini-batch size m just gives you batch gradient descent. The other extreme would be if your mini-batch size, **Were = 1.**

This gives you an algorithm called stochastic gradient descent.

Momentum



Two hyperparameters of the learning rate alpha, as well as this parameter Beta. **Most common value for Beta is 0.9.**

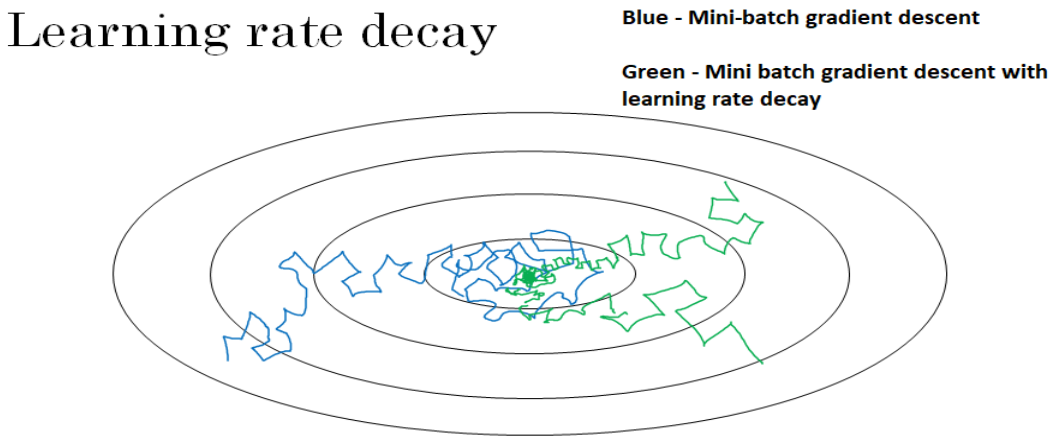
So, for example when β goes 0.9 you could think of this as averaging over the last 10

$\frac{1}{1-\beta} = \frac{1}{1-0.9} = 10.$

RMS Prop and
Adaptive moment estimation (Adam)

1. A common choice really the default choice for **ß1 is 0.9**.
2. The hyper parameter for ß2, the authors of the Adam paper, inventors of the Adam algorithm recommend **0.999**.
3. And then Epsilon, the choice of epsilon doesn't matter very much. But the authors of the Adam paper recommended it **10 to the minus 8**.

Learning rate decay



References: https://keras.io/api/optimizers/learning_rate_schedules/

```
initial_learning_rate = 0.1
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels, epochs=5)
```

```
starter_learning_rate = 0.1
end_learning_rate = 0.01
decay_steps = 10000
learning_rate_fn = tf.keras.optimizers.schedules.PolynomialDecay(
    starter_learning_rate,
    decay_steps,
    end_learning_rate,
    power=0.5)

model.compile(optimizer=tf.keras.optimizers.SGD(
                    learning_rate=learning_rate_fn),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels, epochs=5)
```

```
...
initial_learning_rate = 0.1
decay_steps = 1.0
decay_rate = 0.5
learning_rate_fn = keras.optimizers.schedules.InverseTimeDecay(
  initial_learning_rate, decay_steps, decay_rate)

model.compile(optimizer=tf.keras.optimizers.SGD(
                  learning_rate=learning_rate_fn),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels, epochs=5)
```

```
step = tf.Variable(0, trainable=False)
boundaries = [100000, 110000]
values = [1.0, 0.5, 0.1]
learning_rate_fn = keras.optimizers.schedules.PiecewiseConstantDecay(
    boundaries, values)

# Later, whenever we perform an optimization step, we pass in the step.
learning_rate = learning_rate_fn(step)
```

Importance in hyper-param tuning

And then when using the Adam algorithm I actually pretty much never tuned β_1, β_2, and ϵ. Pretty much I always use 0.9, 0.999 and 10^{-8} although you can try tuning those as well if you wish.

So the chronological order of importance for hyper-parameters: α

β

$\beta_1, \beta_2, \epsilon$

Number of hidden layers

Number of hidden units

learning rate decay

Mini batch size

Normalizing activations in a network

a. In the rise of deep learning, one of the most important ideas has been an algorithm called batch normalization, created by two researchers, Sergey Ioffe and Christian Szegedy.

b. **Batch normalization** makes your hyperparameter search problem much easier, makes your neural network much more robust.

c. It will also enable you to much more easily train even very deep networks.

$$\mu = \frac{1}{m}\sum_i z^{[i]}$$
$$\sigma^2 = \frac{1}{m}\sum_i (z^{[i]} - \mu)$$
$$z^{[i]}_{norm} = \frac{(z^{[i]} - \mu)}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{[i]} = \gamma z^{[i]}_{norm} + \beta$$

gamma and beta are learnable parameters of your model
In practice, **normalizing z** is done much more often than activation.

Ref:
https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

Batch Normalization is used to normalize the input layer as well as hidden layers by adjusting mean and scaling of the activations.

Because of this normalizing effect with additional layer in deep neural networks, **the network can use higher learning rate** without vanishing or exploding gradients.

Furthermore, batch normalization **regularizes the network** such that it is easier to generalize, and it is thus **unnecessary to use dropout** to mitigate overfitting.

```
# instantiate model
model = Sequential()

# we can think of this chunk as the input layer
model.add(Dense(64, input_dim=14, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))

# we can think of this chunk as the hidden layer
model.add(Dense(64, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))

# we can think of this chunk as the output layer
model.add(Dense(2, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('softmax'))

# setting up the optimization of our weights
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='binary_crossentropy', optimizer=sgd)
```

Summary

1. Batch size
2. Learning rate decay
3. Batch norm
4. Adam, RMSProp, Momentum
5. Normalize features
6. Use L1, L2 regularization, Dropout, Data augmentation, Early stopping

Assignment 6: (nextweek)

Use Concepts 1-6 to improve accuracy on CIFAR100 to +90%

- **When a supervised learning system** is design, these are the 4 assumptions that needs to be true

    1. **Fit training set well in cost function** - If it doesn't fit well, the use of a bigger neural network or switching to a better optimization algorithm might help.
    2. **Fit development set well on cost function** - If it doesn't fit well, regularization or using bigger training set might help.
    3. **Fit test set well on cost function** - If it doesn't fit well, the use of a bigger development set might help
    4. **Performs well in real world** - If it doesn't perform well, the development test set is not set correctly or the cost function is not evaluating the right thing

Single number evaluation metric - Summary

- **To choose a classifier**, a well-defined development set and an evaluation metric speed up the iteration process.
- **F1-score**, a harmonic mean, combine both precision and recall.

Example : Cat vs Non- cat

Predict class $\hat{y}$	Actual class y = 1	Actual class y = 0
1	True positive	False positive
0	False negative	True negative

$$\text{Precision (\%)} = \frac{True\ positive}{Number\ of\ predicted\ positive} x\ 100 = \frac{True\ positive}{(True\ positive + False\ positive)}\ x\ 100$$

$$\text{Recall (\%)} = \frac{True\ positive}{Number\ of\ predicted\ actually\ positive} x\ 100 = \frac{True\ positive}{(True\ positive + False\ negative)}\ x\ 100$$

Satisficing and Optimizing metric

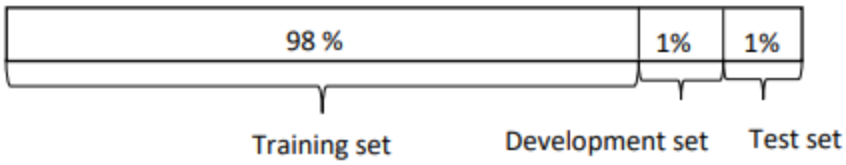
1. Accuracy is an optimizing metric because you want to maximize accuracy
2. Running time is what we call a satisficing metric. **Meaning that it** just has to be good enough, it just needs to be less than 100 milliseconds and beyond that you don't really care, or at least you don't care that much
3. **So more generally**, if you have N metrics that you care about it's sometimes reasonable to pick one of them to be optimizing. So you want to do **as well as is possible** on that one.
4. **And then N minus 1** to be satisficing, meaning that so long as they reach some threshold **you don't care how much better it is in that threshold, but they have to reach that threshold**.

Train/dev/test distributions

Workflow in machine learning is that you try a lot of ideas, train up different models on the training set, and then use the dev set to evaluate the different ideas and pick one.

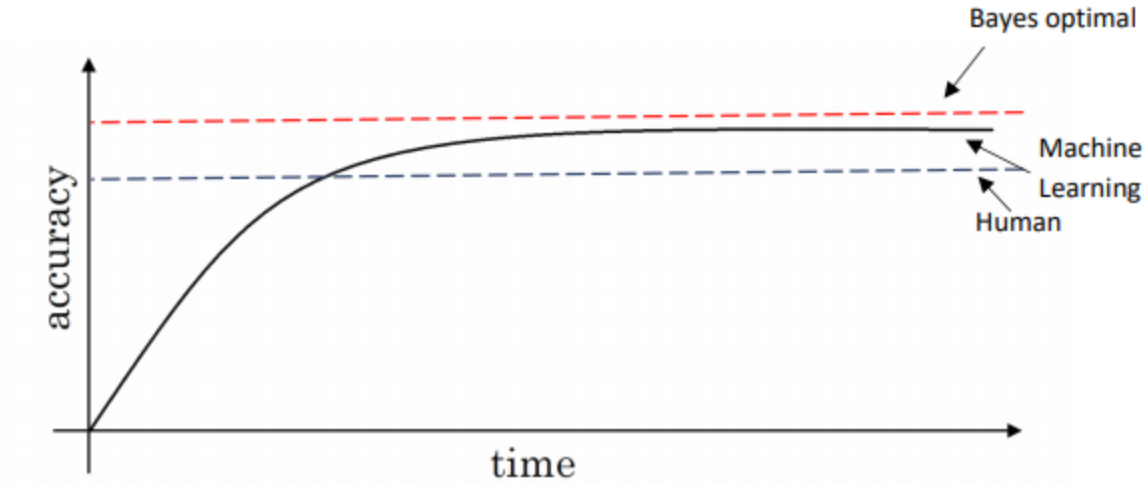
- To make your dev and test sets come from the same distribution
- **Now**, let's say, by way of example, that you're building a cat classifier, and you are operating in these regions: in the U.S, U.K, other European countries, South America, India, China, other Asian countries, and Australia.

- **So**, how do you set up your dev set and your test set? Well, one way you could do so is to pick four of these regions. And say, that data from these **four regions will go into the dev set. And, the other four regions, will go into the test set. [Dont do this!!!]**
- What I recommend instead is that you take all this **randomly shuffled data** into the dev and test set. So that, **both the dev and test sets have data from all eight regions** and that the dev and test sets really come from the same distribution, which is the distribution of all of your data mixed together.



Understanding human-level performance

Human-level error gives an estimate of Bayes error.



Avoidable bias

- By knowing what the human-level performance is, it is possible to tell **<u>when a training set is performing well or not.</u>**

  Example: Cat vs Non-Cat

	Classification error (%)	
	Scenario A	Scenario B
Humans	1	7.5
Training error	8	8
Development error	10	10

  In this case, the human level error as a proxy for Bayes error since humans are good to identify images.

If you want to improve the performance of the training set but you can't do better than the Bayes error otherwise the training set is overfitting.

By knowing the Bayes error, it is easier to focus on whether bias or variance avoidance tactics will improve the performance of the model.

Scenario A There is a 7% gap between the performance of the training set and the human level error. It means that the **algorithm isn't fitting well with the training set** since the target is around 1%. To resolve the issue, we use **bias reduction technique such as training a bigger neural network or running the training set longer.**

Scenario B The training set is doing good since there is only a 0.5% difference with the human level error. The **difference between the training set and the human level error** is called avoidable bias. The focus here is to **reduce the variance** since the difference between the **training error and the development error** is 2%.

To resolve the issue, we use variance reduction technique such as **regularization or have a bigger training set.**

Example 1: Medical image classification This is an example of a medical image classification in which the input is a radiology image and the output is a diagnosis classification decision.

	Classification error (%)
Typical human	3.0
Typical doctor	1.0
Experienced doctor	0.7
Team of experienced doctors	0.5

The definition of human-level error depends on the purpose of the analysis, in this case, by definition the Bayes error is ***lower or equal to 0.5%.***

Example 2: Error analysis

	Classification error (%)		
	Scenario A	Scenario B	Scenario C
Human (proxy for Bayes error)	1	1	0.5
	0.7	0.7	
	0.5	0.5	
Training error	5	1	0.7
Development error	6	5	0.8

Scenario A In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 4%-4.5% and the variance is 1%. Therefore, the focus should be on **bias reduction technique.**

Scenario B In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 0%-0.5% and the variance is 4%. Therefore, the focus should be on *variance reduction* technique.

Scenario C In this case, the estimate for Bayes error has to be 0.5% since you can't go lower than the human-level performance otherwise the training set is overfitting. Also, the avoidable bias is 0.2% and the variance is 0.1%. Therefore, the focus should be on bias reduction technique.

Summary of bias/variance with human-level performance

Human - level error – proxy for Bayes error If the difference between human-level error and the training error is bigger than the difference between the training error and the development error. The focus should be on bias reduction technique

If the difference between training error and the development error is bigger than the difference between the human-level error and the training error. The focus should be on variance reduction technique

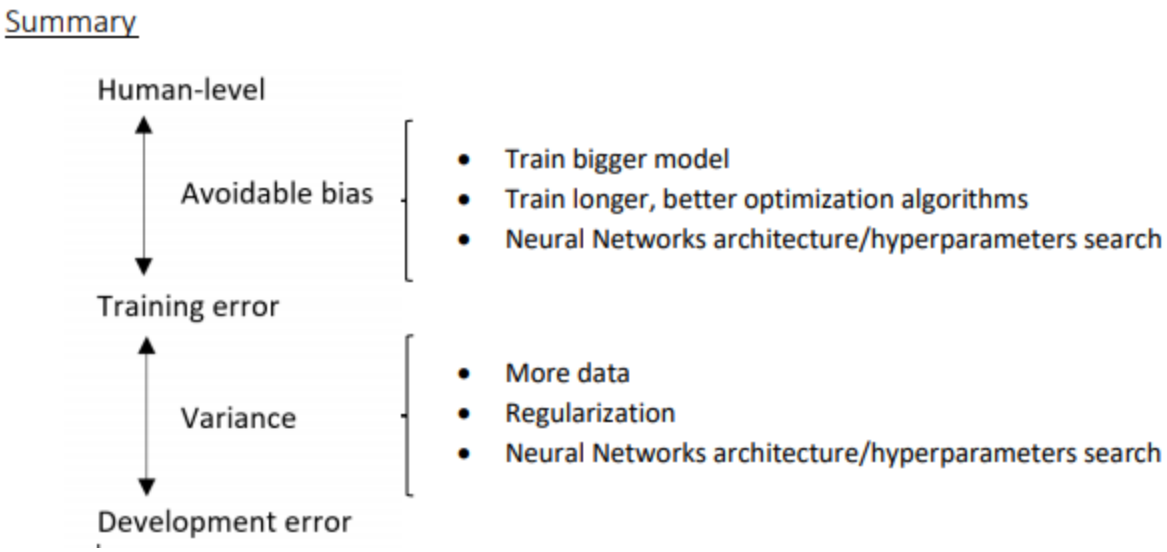
Improving your model performance

The two fundamental assumptions of supervised learning There are 2 fundamental assumptions of supervised learning.

The first one is to have a low avoidable bias which means that the training set fits well.

The second one is to have a low or acceptable variance which means that the training set performance generalizes well to the development set and test set.
If the difference between human-level error and the training error is bigger than the difference between the training error and the development error, the focus should be on bias reduction technique which are training a bigger model, training longer or change the neural networks architecture or try various hyperparameters search.

If the difference between training error and the development error is bigger than the difference between the human-level error and the training error, the focus should be on variance reduction technique which are bigger data set, regularization or change the neural networks architecture or try various hyperparameters search.



Carrying out error analysis

During error analysis, you're just looking at **dev set** examples that your algorithm has misrecognized.

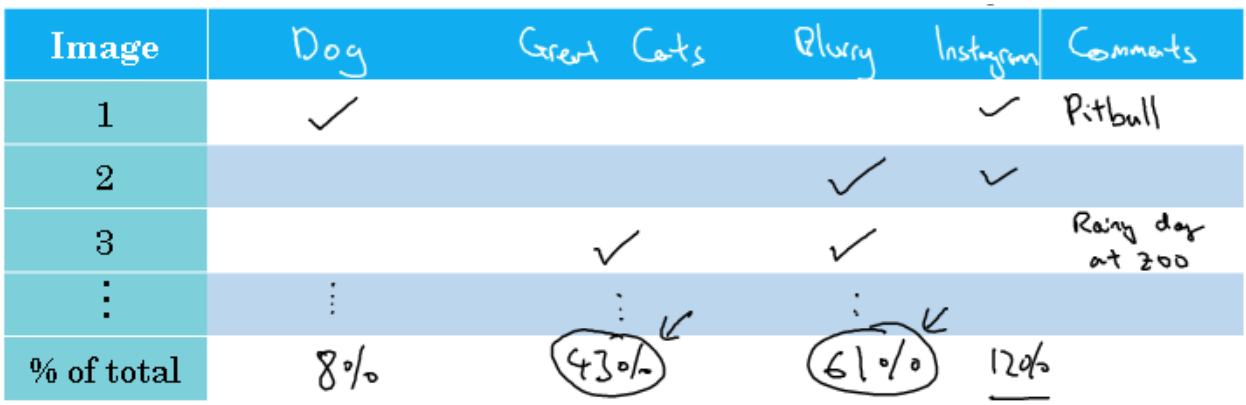


Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮		
% of total	8%	43%	61%	12%	

Cleaning up incorrectly labeled data

Deep learning algorithms are robust to random errors. They are less robust to systematic errors.

If error analysis reveals that **incorrectly labeled** have large proportion in mislabeled examples then go ahead and solve it otherwise ignore

Error analysis

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

Training and testing on different distributions

Example: Cat vs Non-cat

In this example, we want to create a mobile application that will classify and recognize pictures of cats taken and uploaded by users.

There are two sources of data used to develop the mobile app. The first data distribution is small, **10 000 pictures** uploaded from the mobile application.

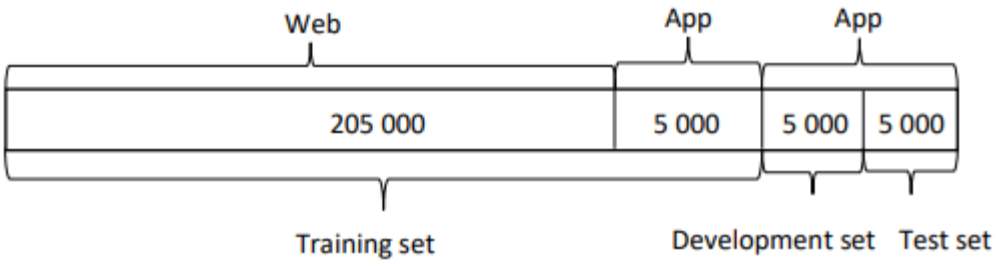
Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded **200 000 pictures** where cat's pictures are professionally framed and in high resolution.

The problem is that you have a different distribution:

1. **small data set** from pictures uploaded by users. This distribution is important for the mobile app.
2. **bigger data set** from the web.

- **The guideline** used is that you have to choose a development set and test set to reflect data you expect to get in the future and consider important to do well.

- **The data is split as follow:**



- **The advantage** of this way of splitting up is that the target is well defined. The disadvantage is that the training distribution is different from the development and test

set distributions. However, this way of splitting the data has a better performance in long term

Bias and variance with mismatched data distributions

- **Example:** Cat classifier with mismatch data distribution
  When the training set is from a different distribution than the development and test sets, the method to analyze bias and variance changes.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

Scenario A

1. **If the development data** comes from the same distribution as the training set, then there is a large variance problem and the algorithm is not generalizing well from the training set.

   However, since the training data and the development data come from a different distribution, this conclusion cannot be drawn.

   There isn't necessarily a variance problem. The problem might be that the development set contains images that are more difficult to classify accurately. When the training set, development and test sets distributions are different, two things change at the same time.

   First of all, the algorithm is trained in the training set but not in the development set.

   Second of all, the distribution of data in the development set is different. It's difficult to know which of these two changes produces this 9% increase in error between the training set and the development set.

   To resolve this issue, we define a new subset called training - development set. This new subset has the same distribution as the training set, but it is not used for training the neural network.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

- **Scenario B**

The error between the training set and the training- development set is 8%.

In this case, since the training set and training-development set come from the same distribution, the only difference between them is the neural network sorted the data in the training and not in the training development.

The neural network is not generalizing well to data from the same distribution that it hadn't seen before Therefore, we have really a variance problem.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

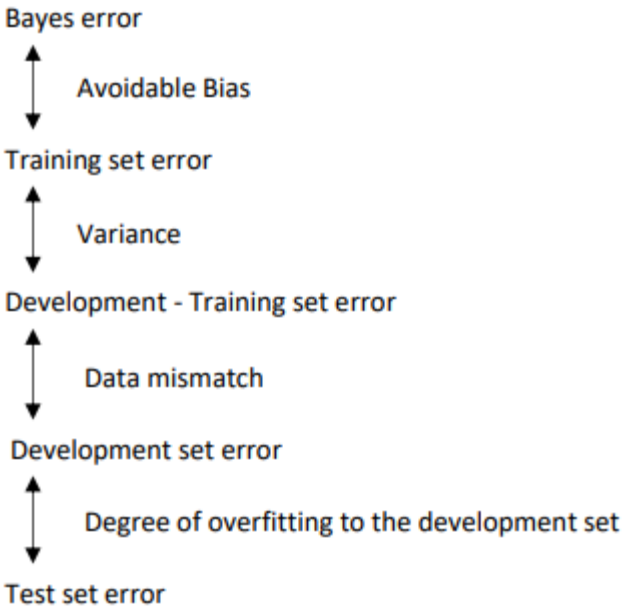
Scenario C In this case, we have a **mismatch data** problem since the 2 data sets come from different distribution.

Scenario D In this case, the avoidable bias is high since the difference between Bayes error and training error is 10 %.

Scenario E In this case, there are **2 problems**. The first one is that the avoidable bias is high since the difference between Bayes error and training error is 10 % and the second one is a data mismatched problem.

Scenario F Development should never be done on the test set. However, the difference between the development set and the test set gives the degree of overfitting to the development set.

General formulation

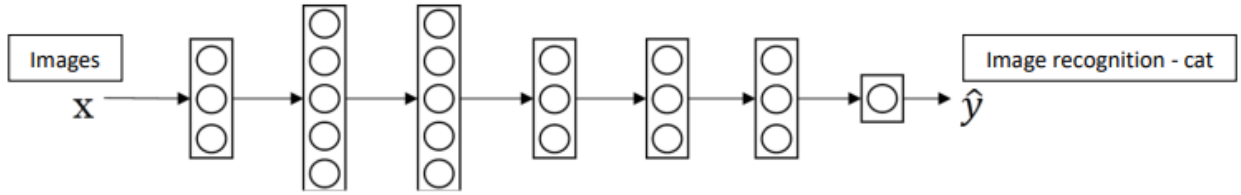


Addressing data mismatch

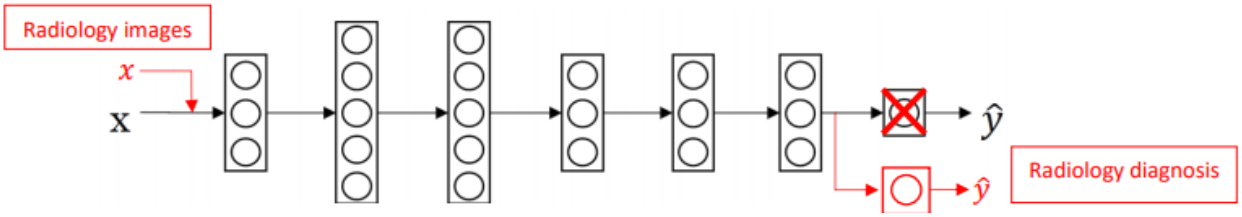
1. **This is a general guideline** to address data mismatch:
2. **Perform manual error analysis** to understand the error differences between training, development/test sets. Development should never be done on a test set to avoid overfitting.
3. **Make training data or collect data** similar to development and test sets. To make the training data more similar to your development set, you can use artificial data synthesis. However, it is possible that if you might be accidentally simulating data only from a tiny subset of the space of all possible examples

Transfer learning

One of the most powerful ideas in deep learning is that sometimes you can take knowledge the neural network has learned from one task and apply that knowledge to a separate task.



If you want to take this neural network and adapt, or we say transfer, what is learned to a different task, such as radiology diagnosis, meaning really reading X-ray scans, what you can do is take this last output layer of the neural network and just delete that and delete also the weights feeding into that last output layer and create a new set of randomly initialized weights just for the last layer and have that now output radiology diagnosis.



You might, if you have a small radiology dataset, you might want to just retrain the weights of the last layer, and keep the rest of the parameters fixed.

If you have enough data, you could also retrain all the layers of the rest of the neural network.

And if you retrain all the parameters in the neural network, then this initial phase of training on image recognition is sometimes called pre-training, because you're using image recognitions data to pre-initialize or really pre-train the weights of the neural network.

And then if you are updating all the weights afterwards, then training on the radiology data sometimes that's called fine tuning.

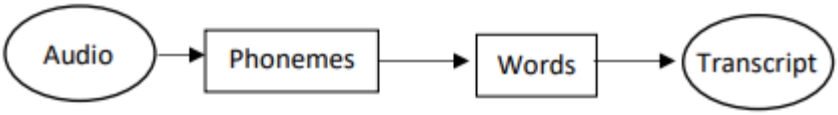
And the rule of thumb is maybe if you have a small data set, then just retrain the one last layer at the output layer. Or maybe that last one or two layers.

1. **Guideline**
    1. **Delete** last layer of neural network
    2. **Delete** weights feeding into the last output layer of the neural network
    3. **Create** a new set of randomly initialized weights for the last layer only
    4. **New** data set (x, y)

Whether to use <u>end-to-end deep learning</u>

- **The traditional way** - small data set

The hybrid way - medium data set



The End-to-End deep learning way – large data set

Before applying end-to-end deep learning, you need to ask yourself the following question:
<u>Do you have enough data</u> to learn a function of the complexity needed to map x and y?

Pro:

Let the data speak - By having a pure machine learning approach, the neural network will learn from x to y. It will be able to find which statistics are in the data, rather than

being forced to reflect human preconceptions.

Less hand-designing of components needed - It simplifies the design work flow.

Cons:

Large amount of labeled data - It cannot be used for every problem as it needs a lot of labeled data.

Excludes potentially useful hand-designed component - Data and any hand-design's components or features are the 2 main sources of knowledge for a learning algorithm. If the data set is small than a hand-design system is a way to give manual knowledge into the algorithm.

Convolutional Networks

COURSE 6: Convolutional Neural Networks

https://upscfever.com/upsc-fever/en/data/en-data-home.html

Innovations in Computer vision

1. Convolutional operation
2. Padding
3. Stride
4. Pooling
5. Fully connected layers

1. Computer Vision

2. Edge Detection Example

3. More Edge Detection

4. Padding

5. Strided Convolutions

6. Convolutions Over Volume

7. One Layer of a Convolutional Network

8. Simple Convolutional Network Example

9. Pooling Layers

10. CNN Example

11. Why Convolutions?

Assignment 7 (15/8/2020):

Implement VGG16, LENET5, AlexNet on Cifar10

Recurrent Networks

Convolutional neural network

1. **Convolutional operation**
2. **Stride**
3. **Padding - valid (no padding), same (output and inputs have same dimension)**
4. **Pooling - Max, Average**
5. **Filters**
6. **Residual blocks**
7. **Inception networks**
8. **1x1 convolution**

Keras: Conv2D

Keras: 2D CNN for MNIST

Keras: 2D CNN for Arabic alphabets

Keras: 2D CNN for Fashion MNIST

Keras: 2D CNN for Flower recognition

Keras: Transfer learning

Keras: Using bottleneck features

Keras: Using Fine Tuning

Keras: Resnet for EmotionAI

Keras: Facial expression detection

Keras: Classify Radio Signals

Inception networks and 1x1 convolutiions

https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html

Assignment 8 (22/9/2020)

Download dataset of indian elephant, african elephant and tusker (100 images each). Find which pretrained model get highest accuracy

Lecture 12: 1230-430PM

Time series, and Object detection, Segmentation, Face recognition, Neural style transfer, CNN for Graphs, GAN

1. Single output
2. Multi-output
3. Single step
4. Multi-step

Single output and Single time step model

1. Create object of Windowgenerator
    a. Input width = 1
    b. offset/shift = 1
    c. label_width=1
    d. Label_column = ‘name of a column’

2. Multistep single output
    a. Input width = n
    b. offset/shift = 1
    c. label_width=1
    d. Label_column = ‘name of a column’

3. Multi-output models
    a. Input width = 1/n
    b. offset/shift = 1
    c. label_width=1
    d. Label_column = ‘name of a column’,’name of col 2’,.....,’name of col n

4. Multi-step models [Single shot predictions]
    a. Input width = 1/n
    b. offset/shift = 1
    c. label_width=1
    d. Label_column = ‘name of a column’,’name of col 2’,.....,’name of col n

Assignment 9 (13/10) - build 4 time series models

UCI Time series dataset

```
!wget -O AirQualityUCI.zip
```
https://archive.ics.uci.edu/ml/machine-learning-databases/00360/AirQualityUCI.zip

```
!unzip AirQualityUCI.zip

df = pd.read_excel('/content/AirQualityUCI.xlsx', parse_dates=[['Date',
'Time']])

df.replace(to_replace=-200.0, value=0.0, inplace=True)

wv = df['NMHC(GT)']

bad_wv = wv == -200.0

wv[bad_wv] = 0.0
```

Object Detection

Neural style transfer

https://colab.research.google.com/drive/1Sk7fT7jhO95yCw2Dkfghlr4gjJkOFsVe

Generative adversarial networks

https://colab.research.google.com/drive/185B4asJbcz7A0xqM3YClcuy0p3WRlAVV#scrollTo=JHga1nihlD6q

Lec 13: Sequence models

Recurrent neural networks learn temporal relationship (audio signals, language, time series, video)

Types of architectures

- **One to one: time series**
- **One to many: text generation**
- **Many to one: sentiment analysis, classification**
- **Many to many: translation**

1. RNN - Short sentences
2. Long short term memory - Long sentences
3. Gated recurrent unit - Long sentences

Bidirectional - Language models

Unidirectional - Time series

RNN suffer from vanishing gradient so we prefer LSTM, GRU

Task1: Single time step model - Predict using RNN, LSTM, GRU (unidirectional) use wide_window and single_step_window

Task 2: Multi-output models - Predict using RNN, LSTM, GRU (unidirectional) use wide_window and single_step_window

https://colab.research.google.com/drive/1kedWo7qbJc4PDhGnr-bk3YpeD11vRJFm

---

Assignment 10 (20/10/2020)

Task 4: Train autoregressive model using RNN, LSTM, GRU (unidirectional) - on multi_window

Next week

Task 3: Multi-step single shot models - Predict using RNN, LSTM, GRU (unidirectional) - multi_window

Time series and Text classification

1. Unidirectional LSTM, GRU, SimpleRNN
2. Bidirectional LSTM, GRU, SimpleRNN
3. Stacking multiple layers of RNN

Note:

1. LSTM with return_sequences = true and followed by LSTM is equivalent to LSTM with return_sequences = false and followed by RepeatVector() followed by LSTM

2. LSTM with return_sequences = false and followed by Dense is equivalent to LSTM with return_sequences = true and followed by TimeDistributed(Dense)