Portfolio of Machine Learning and Deep Learning

May 2016 Update July 2019 Update May 2022

This portfolio documents the algorithms that I have executed at scale in government, finance and industry using enterprise data. These are relevant to Marketing, Business Optimization, Quantitative Finance, and Risk Management. While Bayesian Statistics, Time Series, Markov Chain Monte Carlo (MCMC), Gradient Descent, and Singular Value Decomposition (SVD) algorithms are established; Deep Learning and Reinforcement Learning are rapidly evolving. Therefore; I attempt to update the Deep Learning approaches of Convolutional Neural Networks, Natural Language Processing (NLP), Recurrent Neural Networks (RNN), Collaborative Filtering, Transfer Learning and Reinforcement Learning to state-of-art.

The work derives from my Top Secret work in the Intelligence Community and Corporate work for Fortune 50's and startups. I executed this work leading small distributed teams using Agile methods on Cloud environments.

My Thesis from the National Intelligence University on complexity and simulation on Fragile States won the National Security Award.

My benchmarks for the 'new electricity', Andrew Ng's characterization of Deep Learning, are:

1. Deep Learning (Neural Networks) as a ***universal function approximator***.
2. Harvard and University of Chicago paper on AI and the comparison to steam power which fueled the Industrial Revolution
3. MIT is committing $1 Billion for a new school of AI
4. University of Virginia received a gift of $120 Million for New School of Data Science
5. Google has commercialized over 4,000 {update} Deep Learning algorithms.
6. Laplace's 18'th Century Bayesian Approach focused on Data & Learning
7. The 2009 $1 Million dollar Netflix Competition Collaborative Filtering Algorithm winner: A Recommender system for movies based on peoples ratings.
8. The ***2012 ImageNet*** triumph of Hinton's group using Deep Learning: The margin of victory was 11% over competitors on identifying images
9. In 2018 Sebastian Ruder states NLP's ImageNet Moment has Arrived:
   NLP {OpenAI GPT-3, BERT} https://nlpprogress.com/
10. Progress in Convolution networks // ***Image Classification***
    EfficientNetV2, Normalizer-Free Networks {9x boost in training speed}
11. Progress in ***Recurrent Neural Networks*** Transformer architecture with Attention {BERT}
12. I am working on ***Forecasting Volatility*** using wavelets and my experience in Quantitative Finance, and Econometrics.
13. I am executing a ***Story Generation chatbot with Encoder / Decoder***
14. I have executed Reinforcement Learning with BERT on the GLUE Benchmark

Summary of Approaches

1. Anomaly Detection
2. Dimensionality Reduction (PCA / SVD)
3. Recommender Systems
4. Sentiment Analysis
5. Bayesian Statistics
6. Markov Chain Monte Carlo (MCMC)
7. XGBoost {weak learners}
8. Cloud (AWS / Google) ML Pipelines and eliminates devOps
9. Scala Architecture
10. Why Julia language
11. Scala / Kafka / Spark / Cassandra Real Time Streaming
12. Deep Learning
    - 12.1. Convolution Neural Networks (parameter sharing)
    - 12.2. Attention Mechanism
    - 12.3. Encoder / Decoder (ChatBox)
13. Reproducible Research {Jupyter / Spark / Zeppelin} Notebook
14. Transfer Learning
15. End to End Deep Learning

Deep Learning Fundamentals

Tuning Models

Optimization
1. Gradient Descent
2. Back Propagation
3. Automatic Differentiation

Summary of State-of-Art ML
Focus is on manageable training, transfer learning, and deployment to mobile

1. Attention, BERT, FINBert
2. MobileBERT
3. EfficientNetV2

Executed Projects

HISTORY OF DEEP LEARNING

1995 Support Vector Machines (handwritten recognition) / Random Forests
2006 Rebranded Neural Nets to Deep Learning // Triumphed over SVN in image recognition
2012 ImageNet Competition {ConvNets Triumph}
2016 XGBoost Gradient Boosting {weak learners}
2018 Ruder, NLP's ImageNet Moment has Arrived
2018 BERT {transformer with Attention}
2018 Julia Language {automatic differentiation}
2019 RoBERTa
2020 OpenAI Model from paper of 2005
2021 EfficientNetV2 // Normalizer-Free Networks

Geoffrey Hinton summarized the findings up to today in these four points:

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.
5.

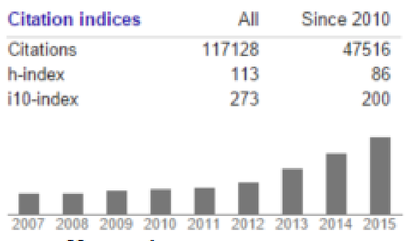
So here we are. Deep learning. The culmination of decades of research, all leading to this:

Deep Learning =

Lots of training data + Parallel Computation + Scalable, smart algorithms

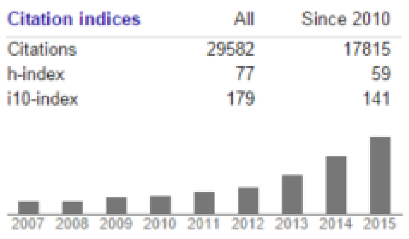
http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Citations for the Work of Deep Learning Pioneers 'takes off'



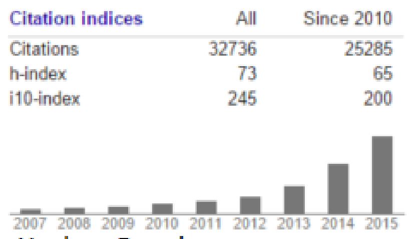
Citation indices	All	Since 2010
Citations	117128	47516
h-index	113	86
i10-index	273	200

Geoffrey Hinton



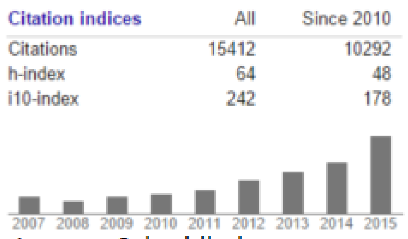
Citation indices	All	Since 2010
Citations	29582	17815
h-index	77	59
i10-index	179	141

Yann LeCun



Citation indices	All	Since 2010
Citations	32736	25285
h-index	73	65
i10-index	245	200

Yoshua Bengio



Citation indices	All	Since 2010
Citations	15412	10292
h-index	64	48
i10-index	242	178

Juergen Schmidhuber

Technology is a means to an end and only one factor to achieving success.

The striking competitive advantage gained by AI is the foundation of big tech (near trillion dollar market capitalization companies) GAFA {Google / **Apple** / Facebook / **Amazon**}.

'Silicon valley is coming' said JPMorgan Chase CEO Jamie Dimon in his April 2015 annual letter to shareholders. He was talking about the ABCD's of FINTECH {Artificial Intelligence / BlockChain / Cloud Computing / Data & Analytics}. I have mastered all 4 of these components.

Andrew Ng, the founder of Coursera, calls ML the New Electricity. A recent NBER (National Bureau of Economic Research) by 2 MIT & 1 U of C professors calls it the new 'steam' power with possible effects similar to the Industrial Revolution: Artificial Intelligence and the Modern Productivity Paradox:.

Google's focus is 'AI first' while Microsoft's CEO says AI is the 'ultimate breakthrough' technology. No wonder 2 of my prominent Machine Learning books are from researchers now employed by Microsoft and Google!

I have been working in this field for over 7 years, first in the Intelligence Community and recently at a startup, ETRADE, Comcast, and several Financial Firms.

I am currently demonstrating my Leadership / Technical Prowess with 2 Initiatives and Financial Consulting Engagements.

I know from experience it is a SEVERE CHALLENGE to go from $0 to $1 Billion for an established company.

Data Science Prototyping and Program Management
My experience: I successfully executed Data Science at several large Government and Commercial enterprises. Surprisingly, advanced technology is rarely the gating factor. Attached are Executive Presentations and the Program Management approach that I used at the Securities and Exchange Commision, Comcast, and ETRADE.

Summary of Approaches

1) Anomaly Detection

My experience: I lead a Cybersecurity Task force with a $100 million dollar budget across all 16 Agencies in the Intelligence Community. We implemented common algorithms for network security across all Agencies, and Anomaly Detection on Top Secret Networks. Anomaly Detection is a general approach usable in manufacturing, finance, and risk management.

Features are selected and quantified from the Domain.
$For\ Example\colon\ x_1\colon\ number\ logins,\ x_2\colon\ web\ pages\ visited,\ x_3\colon\ number\ transactions\ x_4\colon\ typing\ speed$
$Dataset\colon \{x^1,\ x^2,\ .\ .\ .\ x^m\}\ where\ x^m\ is\ a\ vector\ of\ features\ for\ a\ sample\ m$
$of\ dimension\ n\ (\ x\ \varepsilon\ R^n)$

Model probability of a feature and find anomalies by $p(x)\ <\ \epsilon\ signifies\ an\ anomaly.$
Assume independent Gaussian Distributions for the model. In practice this works even if the features are not independent (Stanford ML).

$$p(x)\ =\ \prod_{i=1}^{n} p(x_i\colon \mu_i,\ \sigma_i^2)\ =\ \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_i}} exp(-\ \frac{(x_i-\mu_i)^2}{2\sigma_i^2})$$

Plug the values and determine if $p(x)\ <\ \epsilon$ to identify anomalies.

2) *Dimensionality Reduction*

(PCA) Principal Component Analysis and (SVN) Singular Value Decomposition

My experience: Many problems are represented by a high number of dimensions which can be reduced to simplify the model and rank the factor causes. A correlation matrix of a large portfolio is one example. The number of elements in the portfolio can be significantly reduced with minimal impact on risk/return. SVN is the favored approach to Recommender Systems as documented in the next algorithm.

First perform mean normalization and perhaps feature scaling

$For\ m\ samples\ with\ n\ features\ \mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}\ \ Replace\ x_j^{(i)}\ with\ x_j - \mu_j\ For\ feature\ scaling\ divide\ by\ sigma\ s$

Efficient algorithms exist to calculate Singular Value Decomposition.

$X = U\Sigma W^T\ \ X\ is\ m\ x\ n,\ U\ is\ m\ x\ p,\ \ \Sigma\ is\ diagonal\ matrix\ p\ x\ p,\ \ W^T is\ \ p\ x\ n\ \ where$

$U, \Sigma, W\ are\ unique.\ U, W\ are\ column\ orthonormal\ U^TU = I,\ W^TW = I,\ \Sigma\ is\ diagonal\ with\ singular\ values\ \sigma_1 \geq$

$\Sigma^T\Sigma\ is\ a\ square\ diagonal\ matrix\ p\ x\ p\ with\ the\ Singular\ values\ squared\ of\ \ \Sigma\ which\ can\ reduce\ X's\ dimensions.$

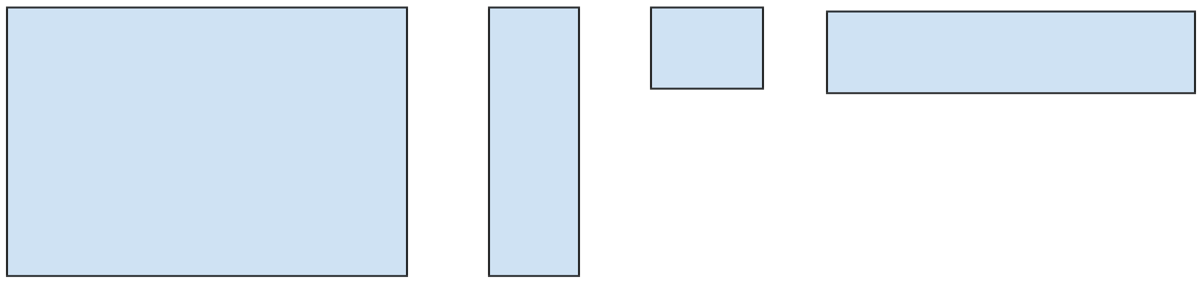
$Reduce\ dimensions\ to\ r\ \prec\ p\ by\ retaining\ \sum_{i=1}^{r}\sigma_{(i)}^2\ \geq\ 0.8\ set\ the\ other\ values\ to\ zero.$

3) ***Recommender Systems***
https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75

My experience: The benchmark example is improving the Netflix recommender for movies. I applied this to matching 'styles' of accessories and clothing to people rating a survey of 'styles'. The challenges to executing and my approach are outlined below:

1. It is difficult to formulate features/concepts segmenting 'styles' as it is to classify movies. Solution: Identify ***latent concepts*** (factors 'discoverable') through decomposing the matrix X {styles x people} into two 'skinny' matrices of {styles x concepts} (U) and {people x concepts} (W). Concepts are derived and ranked by importance.
2. The style matrix X is missing a large number of ratings (***sparse***). Solution: Extract training and test sets. Minimize the Root Mean Square (RMS) on the non-zero elements of the training set using Gradient Descent with Regularization. Regularization enables determining more than 2 factors by preventing overfitting. Then use SVN to discover the latent factors. What is a latent factor? It is a broad idea which describes a property or concept that a user or an item have. For instance, for music, latent factor can refer to the genre that the music belongs to. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension *r*. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. Thus use $X = U\Sigma W^T$ to interpolate missing values of X. Reduce the dimension of Σ for latent factors accounting for x% of SSE. Use the singular values of Σ to determine the dimensions of the 'skinny' matrices to capture a percentage of variance.
3. If Improved fit is required. Solution: Use a sequential combination of algorithms through Boosting.

Singular Value Decomposition of a X (people x movies)
(people x movies) = (people x latent factors) (lf x lf) (lf x movies)



4) Sentiment Analysis

Naïve Bayes or 2-layer ~~LSTM~~ sequence classifier (n dimensional output). Update from LSTM.

1. GloVe 50- dimensional embedding
a. Word-to-index into dense vector of dimension 50
b. Need to update embedding with key domain words
c. (batch_size, max_input_length, zero padded)
2. 2 stacked LSTM with Dropout in between
3. Dropout
4. Softmax

Naive Bayes Classifier

Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $x =$ $x = (x_{1,} ... x_n)$ representing some n features (independent variables), it assigns to this instance probabilities $p(C_k | x_1, ... x_n)$

for each of K possible outcomes or *classes* C_k. Now the "naive" conditional independence assumptions come into play: assume that each feature x is conditionally independent of every other feature x_j *for* $j \neq i$, *given the category* C_k . This means that $p(x_i | x_{i+1}, ... x_{n,} C_k) = p(x_i | C_k)$.

Thus, the joint model $p(C_k | x_i, ... x_n)$ is proportional to $p(C_k) \prod_{i=1}^{n} p(x_i | C_k)$

Logistic Regression $\hat{y} = \sigma(W^T X + b)$
Does not use squared error because it is not convex

Cross Entropy Loss Function: $L(\hat{y} - y) = -(y log\hat{y} + (1 - y) log(1 - \hat{y}))$

Cost Function = $J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L(\widehat{y^{(i)}}, y^{(i)})$

$\sigma = \frac{1}{1+e^{-z}}$ sigmoid

5) Bayesian Statistics http://www.mit.edu/~9.520/spring10/

My experience: I use the Bayesian approach because its natural fit to the primacy of data and learning from sequential trials. Successful applications abound in machine learning, medicine and physics (Bishop). This approach develops an assessment with quantified uncertainty. Additionally, new evidence can be added to revise the assessment and uncertainty.

$$p(parameters \mid Data) \;=\; \frac{P(Data \mid parameters)\, p(parameters)}{p(Data)} \{posterior\ \alpha\ likelihood\ x\ prior\}$$

with $p(Data) \;=\; \int p(Data \mid parameters)\, p(parameters)\, dp$

In both the Bayesian and Frequentist approaches the likelihood is central; however, the interpretation is different.

Frequentists consider the parameters, determined by an estimator, fixed with error derived from a distribution of ‘possible’ data sets.

Bayesians consider the data an observable (a single data set) with uncertainty over the distribution of parameters (p(Data) - above.

Support for the Bayesian approach is: Laplace determining the mass of Saturn within 1% from observations in 1815, calculating the probability of a rare disease when receiving a positive test result (medicine gets this wrong), and revising estimates through additional trials (sequential coin tosses).

MCMC (next) enables Bayesian methods by calculating intractable integrals such as the partition function (p(Data)) using sampling.

6) Markov Chain Monte Carlo (MCMC)

My experience: I have applied MCMC in statistical physics (exploring large dimensional spaces), Quantitative Finance, and Risk Management.

MCMC sampling is effective in calculating intractable integrals over multi-dimensional spaces for applications in Bayesian statistics, computational physics and linguistics. The Metropolis-Hastings algorithm generates the random sample from a 'proposal' distribution such as a uniform or Gaussian distribution. Samples, generated using a rejection criteria, create a correlated random walk which explores the space. Intuitions developed in 3 dimensions do not generalize to high-dimensional spaces. For instance, most of the volume of a D dimensional sphere of high dimension is concentrated on the shell (curse of dimensionality).

Support Vector Machines (SVM)
My experience: SVM's are an algorithm which can be applied to non-linear classification. It was successful in image detection until supplanted by Convolution in 2012. I study SVM's to understand non-linear algorithms and the power of transforming problems to higher dimensional spaces. There are SVM algorithms (kernels) that transform a problem to an infinite dimensional space.

SVM is a Binary classification model; however, using the 'kernel trick' it can perform non-linear classification by mapping inputs into higher-dimension feature spaces. Dot products are replaced by a non-linear kernel function.

The kernel is related to the transform $\phi(x_i)\ through\ k(x_i, x_j) = \phi(x_i) * \phi(x_j)$
The Kernel is a similarity function. The Gaussian radial basis kernel:
$k(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$ transforms the problem to an infinite dimensional feature space!

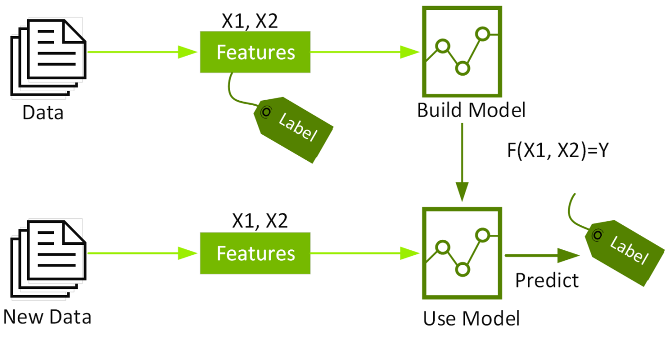
However; higher-dimension feature spaces increase the generalization error requiring a greater number of samples. The classifier is a hyperplane in the transformed space.

7) XGBoost: https://www.gormanalysis.com/blog/gradient-boosting-explained/
https://www.nvidia.com/en-us/glossary/data-science/xgboost/

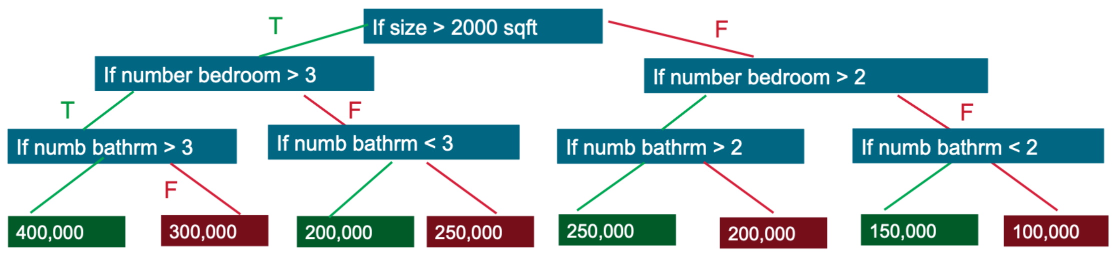
"If linear regression was a Toyota Camry, then gradient boosting would be a UH-60 Blackhawk Helicopter. A particular implementation of gradient boosting, XGBoost, is consistently used to win machine learning competitions on Kaggle. Unfortunately many practitioners (including my former self) use it as a black box. It's also been butchered to death by a host of drive-by data scientists' blogs." The following is from nvidia and is missing a description of the critical Regularization.

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems. The algorithms that XGBoost builds upon are: supervised machine learning, decision trees, ensemble learning, and gradient boosting.

Supervised machine learning uses algorithms to train a model to find patterns in a dataset with labels and features and then uses the trained model to predict the labels on a new dataset's features.

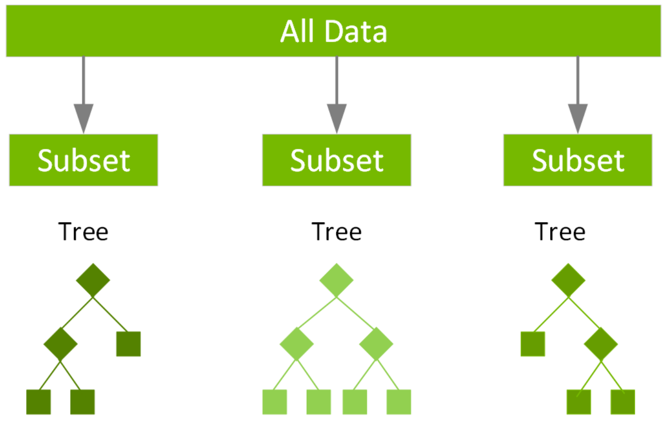


Decision trees create a model that predicts the label by evaluating a tree of if-then-else true/false feature questions, and estimating the minimum number of questions needed to assess the probability of making a correct decision. Decision trees can be used for classification to predict a category, or regression to predict a continuous numeric value. In the simple example below, a decision tree is used to estimate a house price (the label) based on the size and number of bedrooms (the features).



A Gradient Boosting Decision Trees (GBDT) is a decision tree ensemble learning algorithm similar to random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model.

Both random forest and GBDT build a model consisting of multiple decision trees. The difference is in how the trees are built and combined.



Random forest uses a technique called bagging to build full decision trees in parallel from random bootstrap samples of the data set. The final prediction is an average of all of the decision tree predictions.

The term “gradient boosting” comes from the idea of “boosting” or improving a single weak model by combining it with a number of other weak models in order to generate a collectively strong model. Gradient boosting is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function. Gradient boosting sets targeted outcomes for the next model in an effort to minimize errors. Targeted outcomes for each case are based on the gradient of the error (hence the name gradient boosting) with respect to the prediction.

GBDTs iteratively train an ensemble of shallow decision trees, with each iteration using the error residuals of the previous model to fit the next model. The final prediction is a weighted sum of all of the tree predictions. Random forest “bagging” minimizes the variance and overfitting, while GBDT “boosting” minimizes the bias and underfitting.

XGBoost is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms, being built largely for energizing machine learning model performance and computational speed. With XGBoost, trees are built in parallel, instead of sequentially like GBDT. It follows a level-wise strategy, scanning across gradient values and using these partial sums to evaluate the quality of splits at every possible split in the training set.

8) Cloud Resources and Experience

Pipeline: Data cleaning and labeling, Model Build and experiment, Hyperparameter tuning, and implementation at Production speed and quantity. This is where my experience with Cloud Pipelines: ***Amazon SageMaker*** and ***Google MLOps***.

9) Scala Architecture (update in June 2022 for Scala 3)

My experience: I am a Scala Architect with an Architectural Principles Document which I abstract as:

1. Functional Language
   Immutable / Lambdas / Closures / Higher Order Functions
2. Statically Typed
   Errors at compilation time versus Python run time
3. Favor Expressions for Composability
   Futures over Actors / Syntactic sugar For Comprehensions
4. Collection Library
   Foundation of Spark / Lazy Evaluation
5. Category Theory through Scalaz (Effective for DSL's)
   Functors / Monads

I have extensive Python experience with all the major libraries and a command of Tensorflow and Keras.

10) Why Julia language

1. Excels at High Performance Computing
2. Excels at Analytics {Tensors, Differential Equations, Simulation, Time Series, Spectral Analysis}
3. Explosive growth of scientific libraries
4. Debugging and Introspection
5. Flexible Deployment {TPU's GPU's}
6. Best-of-class Automatic Differentiation (AD)
7. Integration with Python, C, C++

11) Scala / Kafka / Spark / Cassandra Real Time Streaming

ETrade, Comcast, and Celebrity Execution

12) Deep Learning

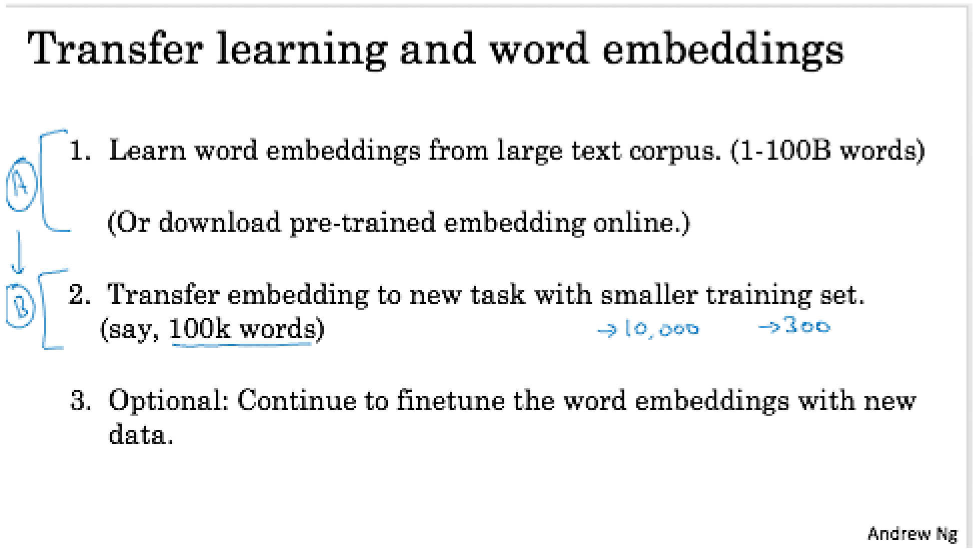
My experience: I have applied Convolution and Recurrent Neural Networks for Image Detection, Natural Language Processing, and Quantitative Finance in large Government and Commercial Enterprises. Typical solutions required preprocessing large distributed data, and implementing multi-stage Deep Learning Models. I accelerate implementation by employing pre-trained models and integrating with established Deep Learning products.

13) Reproducible Research

All my work in Python, Julia, and Scala is executed in notebooks permitting reproducible research. I use Google colabs to experiment with analytics and minimize costs.

14) Transfer Learning

Pre-trained models were applied in many different domains and started to be considered critical for ML research. In computer vision, supervised pre-trained models such as Vision Transformer have been scaled up and self-supervised pre-trained models have started to match their performance. The latter have been scaled beyond the controlled environment of ImageNet to random collections of images. In speech, new models have been built based on wav2vec 2.0 such as W2v-BERT.



Transfer learning and word embeddings

A
1. Learn word embeddings from large text corpus. (1-100B words)

   (Or download pre-trained embedding online.)

B
2. Transfer embedding to new task with smaller training set. (say, 100k words) →10,000 →300

3. Optional: Continue to finetune the word embeddings with new data.

Andrew Ng

Transfer Learning

Transfer learning refers to using the neural network knowledge for another application.

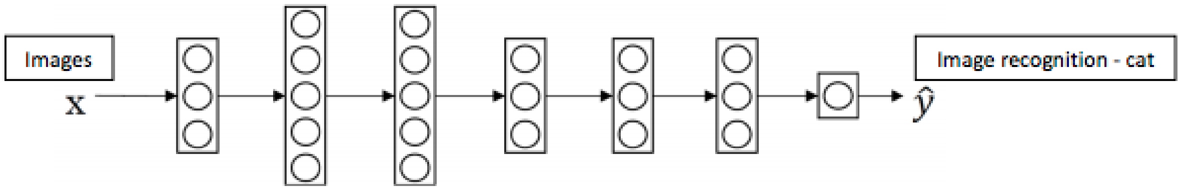
When to use transfer learning

- Task A and B have the same input x
- A lot more data for Task A than Task B
- Low level features from Task A could be helpful for Task B

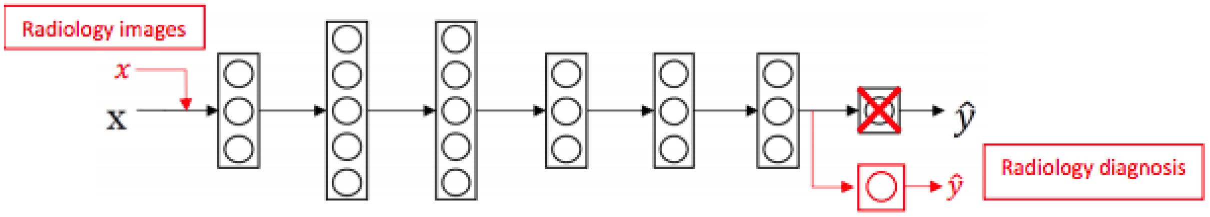
Example 1: Cat recognition - radiology diagnosis

The following neural network is trained for cat recognition, but we want to adapt it for radiology diagnosis. The neural network will learn about the structure and the nature of images. This initial phase of training on image recognition is called pre-training, since it will pre-initialize the weights of the neural network. Updating all the weights afterwards is called fine-tuning.

For cat recognition
Input x: image
Output y – 1: cat, 0: no cat



Radiology diagnosis
Input x: Radiology images – CT Scan, X-rays
Output y :Radiology diagnosis – 1: tumor malign, 0: tumor benign



Guideline

- Delete last layer of neural network
- Delete weights feeding into the last output layer of the neural network
- Create a new set of randomly initialized weights for the last layer only
- New data set (x, y)

Multi-task learning

Multi-task learning refers to having one neural network do simultaneously several tasks.

When to use multi-task learning

- Training on a set of tasks that could benefit from having shared lower-level features
- Usually: Amount of data you have for each task is quite similar
- Can train a big enough neural network to do well on all tasks

Example: Simplified autonomous vehicle

The vehicle has to detect simultaneously several things: pedestrians, cars, road signs, traffic lights, cyclists, etc. We could have trained four separate neural networks, instead of train one to do four tasks. However, in this case, the performance of the system is better when one neural network is trained to do four tasks than training four separate neural networks since some of the earlier features in the neural network could be shared between the different types of objects.

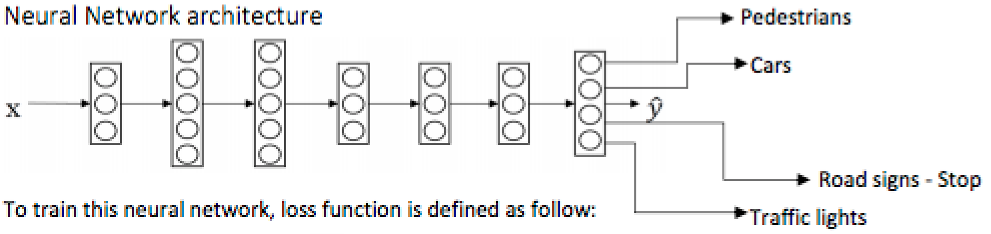
The input $x^{(i)}$ is the image with multiple labels
The output $y^{(i)}$ has 4 labels which are represents:

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \begin{array}{l} \text{Pedestrians} \\ \text{Cars} \\ \text{Road signs - Stop} \\ \text{Traffic lights} \end{array}$$

$$\downarrow$$

$$Y = \begin{bmatrix} | & | & | & | \\ y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \\ | & | & | & | \end{bmatrix} \qquad \begin{array}{l} Y = (4, m) \\ Y = (4,1) \end{array}$$

Neural Network architecture



To train this neural network, loss function is defined as follow:

$$-\frac{1}{m}\sum_{i=1}^{m}\boxed{\sum_{j=1}^{4}}\left(y_j^{(i)}\log\left(\hat{y}_j^{(i)}\right)+\left(1-y_j^{(i)}\right)\log\left(1-\hat{y}_j^{(i)}\right)\right)$$

Also, the cost can be compute such as it is not influenced by the fact that some entries are not labeled. Example:

$$Y = \begin{bmatrix} 1 & 0 & ? & ? \\ 0 & 1 & ? & 0 \\ 0 & 1 & ? & 1 \\ ? & 0 & 1 & 0 \end{bmatrix}$$

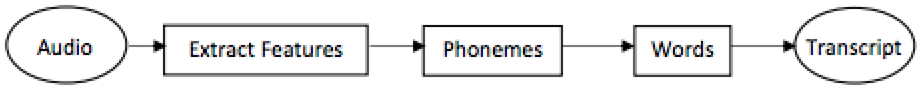
15) End to End Deep Learning

What is end-to-end deep learning

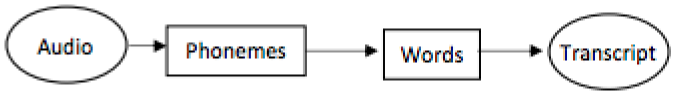
End-to-end deep learning is the simplification of a processing or learning systems into one neural network.

Example - Speech recognition model

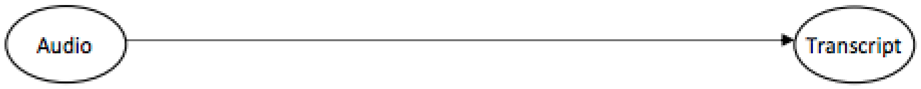
The traditional way - small data set



The hybrid way - medium data set



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



End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc.

Whether to use end-to-end deep learning

Before applying end-to-end deep learning, you need to ask yourself the following question: Do you have enough data 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.

Deep Learning Basics

Neural Network Notation

Do not count input layer in neural networks

For n_x $neurons\ and\ m\ samples\ \ X\ is\ x^{[1]},\ ...\ x^{[m]}\ column\ vectors\ \ activation\ a^{[0]}\ =\ x$

$Z^{[i]} = W^{[i]}X + b^{[i]}\ \ where\ A^{[i]} = \sigma(Z^{[i]})\ \ Z\ rows\ go\ across\ training\ examples$
$\sigma\ is\ a\ non-linear\ activation\ preventing\ the\ model\ to\ collapse\ t$
$W\ is\ oriented\ horizontally\ to\ prevent\ the\ need\ for\ a\ Transpose$

Cross Entropy Loss $L(\widehat{y} - y)\ =\ -(ylog\widehat{y}\ +\ (1-y)log(1-\widehat{y}))$

Multiple Output using softmax for deep Neural Net

$Z^{[l]} = W^{[l]}X + b^{[l]}\ \ where\ A^{[l]} = softmax(Z^{[l]})$ For l'th and final layer apply softmax activation

Softmax function takes an N-dimensional vector of real numbers and transforms it into a vector of real numbers in range (0,1) which add up to 1.

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^{a}}$$

Initialization and Hyperparameters

Neural Network by Input

Image	CNN (Convolutional neural network)
Sequential	RNN / LSTM (Recurrent neural network, long-short-term-memory)
Audio	RNN / LSTM
Video	CNN + RNN hybrid network

Bias	can initialize to zero
Weights	NOT ZERO / random small values OR XAVIER

Loss Functions

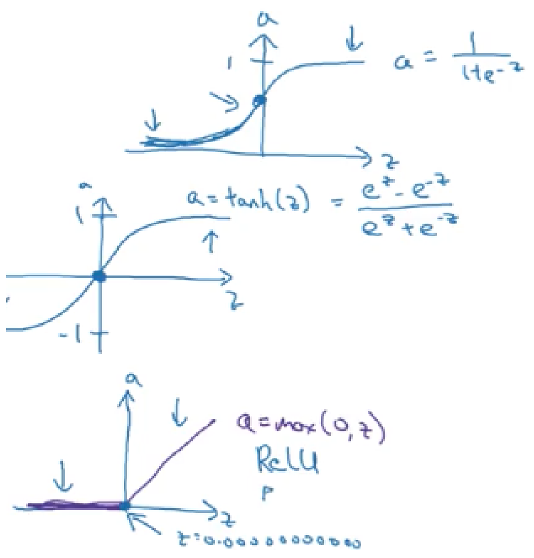
Squared Loss	Regression
Cross Entropy	Sigmoid Binary Classification
Softmax	Multiple Classification NOT A LOSS FUNCTION
Root MSE	Feature engineering

Regularization

Prevents overfitting and parameteres becoming too large.

- `L2`
  - Sparse models
  - More heavily penalizes large weights, but doesn't drive small weights to 0.
- `L1`
  - Dense models
  - Has less of a penalty for large weights, but leads to many weights being driven to 0 (or very close to 0), meaning that the resultant weight vector can be sparse.
- `Max-norm`
  - Alternative to `L2`, good with large learning ratesj
  - Use with `AdaGrad`, `SGD`
- `Dropout`
  - Temporarily sets activation to 0
  - Works with all NN types
  - Avoid using on first layer, risks loosing information.
  - Increases training times x2, x3, not a good fit for millions of training records.
  - Use with `SGD`
  - Influences choice of momentum: 0.95 or 0.99
  - Values (per layer type)
    - Input: [0.5, 1.0)
    - Hidden: 0.5
    - Output: don't use.

Why a non-linear activation function? Without non-linear activation multiple layers collapse into a linear transformation.
Sigmoid, tanh, Relu



Steps for Training a Neural Net

1. Random Initialize weights
2. Forward propagation
3. Implement cost function
4. Backpropagation with partial derivatives
5. Gradient checking to validate
6. Gradient descent to minimize cost function (Definition of derivative)

Training Set (60%) / Cross Validation (20%) / Test Set (20%)

Bias (underfitting) vs Variance (overfitting) Regularization Procedure (lambda)

Learning Curves error vs training set size versus desired performance

1. Bias if train error is larger than desired performance
2. Variance if train error ok; however, test error is too far above desired performance

Metrics

P Precision = True Positives / Predicted Positives
R Recall = True Positives / Actual Positives
F Score = 2 PR/(P+R)

Hyperparameters

Learning Rate: α
Number Iterations
Number of hidden layers
Number of neurons per layer
Activation Functions

My strength would be on Deep Learning / Machine Learning and Real-time Data streaming (Scala/Spark/Cassandra).

My experience is Organization and data is more than 60% of the challenge.

Natural Language Processing (NLP) Convolution
Implementation Notes & Andrew Ng Machine Learning

Supervised / Unsupervised

MATLAB (conventions) Octave

Trade off between more data / improving model

Multivariate Regression
- Cost Function
- Gradient Descent
- Vectorization
- Feature Normalization
- Analytic Calculations Normal Equation

Logistic Regression (Percent certainty)
- Sigmoid Function (overtaken in Neural networks)
- Cost Function (looks the same)
- Gradient Descent
- Regularization to penalize for overfitting (too many parameters)
- Multiclass One-vs-all

2012 Hinton’s team won ImageNet by 11% margin on competitors

Neural Networks (nonlinear hypothesis) combinatorics demonstrates why so powerful

- Cost Function
- Backpropagation
- Unrolling Parameters
- Gradient Checking
- Random Initialization

Hyperparameter tuning

Bias (underfitting) vs Variance (overfitting) Regularization Procedure (lambda)

Learning Curves error vs training set size versus desired performance

3. Bias if train error is larger than desired performance
4. Variance if train error ok; however, test error is too far above desired performance

Metrics

P Precision = True Positives / Predicted Positives
R Recall = True Positives / Actual Positives
F Score = 2 PR/(P+R)

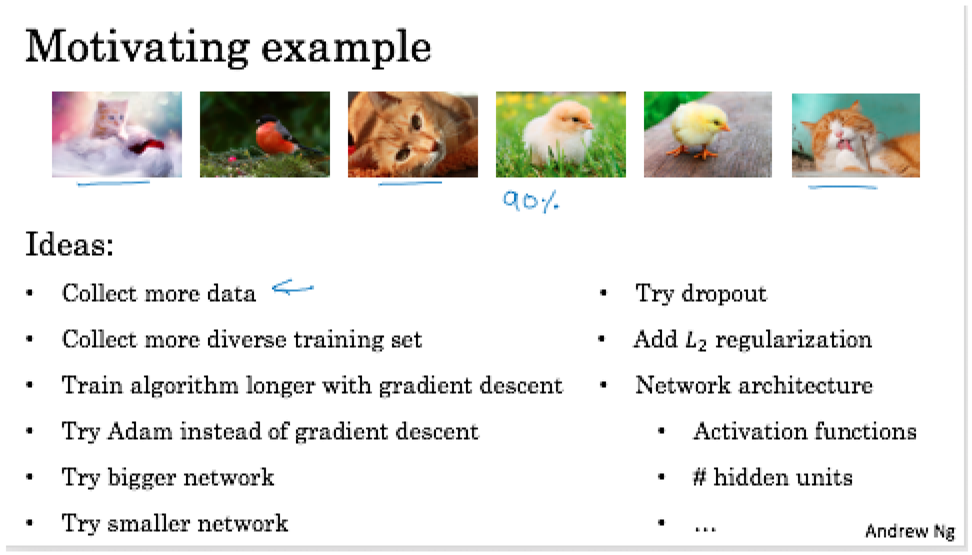
UNSUPERVISED

Clustering K-Means Algorithm

- Random Initialization

Elbow method for choosing number of clusters

Tuning Models



Motivating example

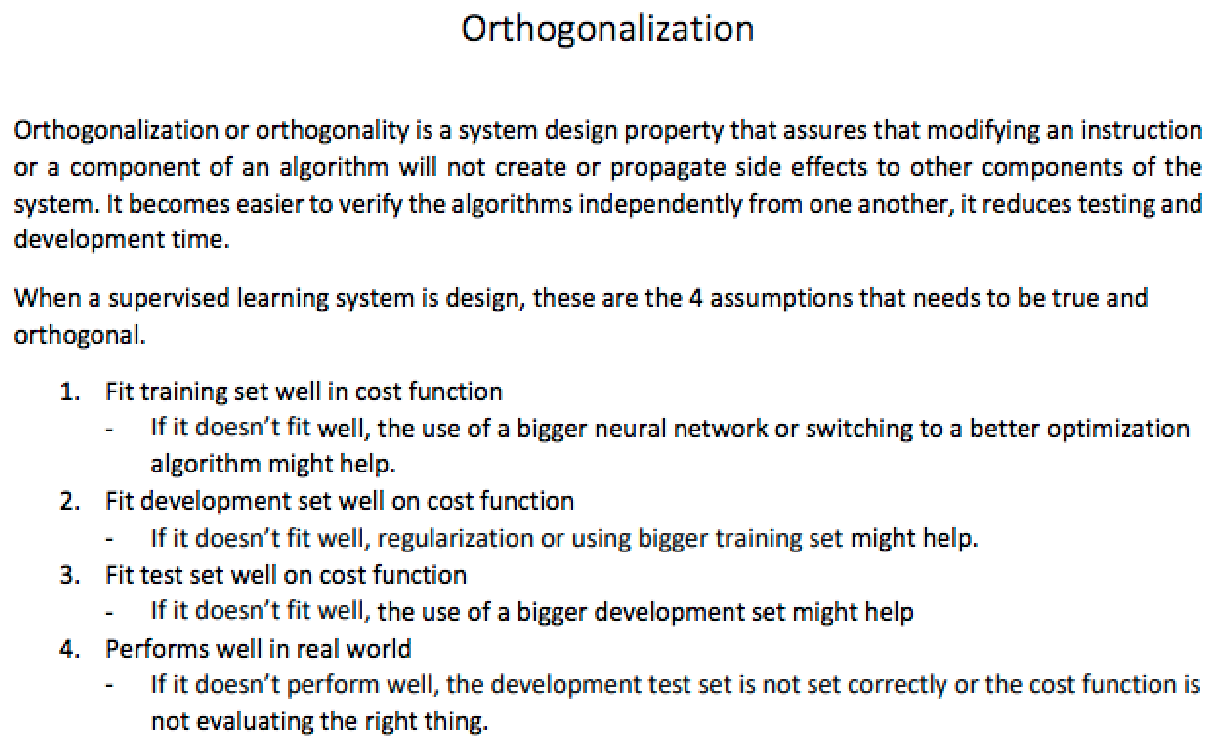
90%

Ideas:

- Collect more data ←
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
  - Activation functions
  - # hidden units
  - …

Andrew Ng

Orthogonalization to focus on determining the Problem



Orthogonalization

Orthogonalization or orthogonality is a system design property that assures that modifying an instruction or a component of an algorithm will not create or propagate side effects to other components of the system. It becomes easier to verify the algorithms independently from one another, it reduces testing and development time.

When a supervised learning system is design, these are the 4 assumptions that needs to be true and orthogonal.

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.

Binary Evaluation

Confusion Matrix

	P’ Predicted	N’ Predicted
P Actual	True Positive	False Negative Type II error
N Actual	False Positive Type I error	True Negative

Sample Results

	P’ Predicted	N’ Predicted
P Actual 10	True Positive 8’	False Negative Type II error 2’
N Actual 10	False Positive Type I error 4’	True Negative 6’

Measures

Accuracy = (TP + TN) / (TP + FP + TN + FN) = 14 / 20	Percent correctly identified
Precision = TP / (TP + FP) = 8 / 12	Positive Prediction Value
Recall = TP / (TP + FN) = 8 / 10	How well model avoids false negatives
F1 = 2TP/(2TP + FP + FN) = 16/(16+4+2) = 8/11	harmonic mean of Precision & Recall

Single number evaluation metric

To choose a classifier, a well-defined development set and an evaluation metric speed up the iteration process.

Example : Cat vs Non- cat

y = 1, cat image detected

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

Precision

Of all the images we predicted y=1, what fraction of it have cats?

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

Recall

Of all the images that actually have cats, what fraction of it did we correctly identifying have cats?

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

Let's compare 2 classifiers A and B used to evaluate if there are cat images:

Classifier	Precision (p)	Recall (r)
A	95%	90%
B	98%	85%

In this case the evaluation metrics are precision and recall.

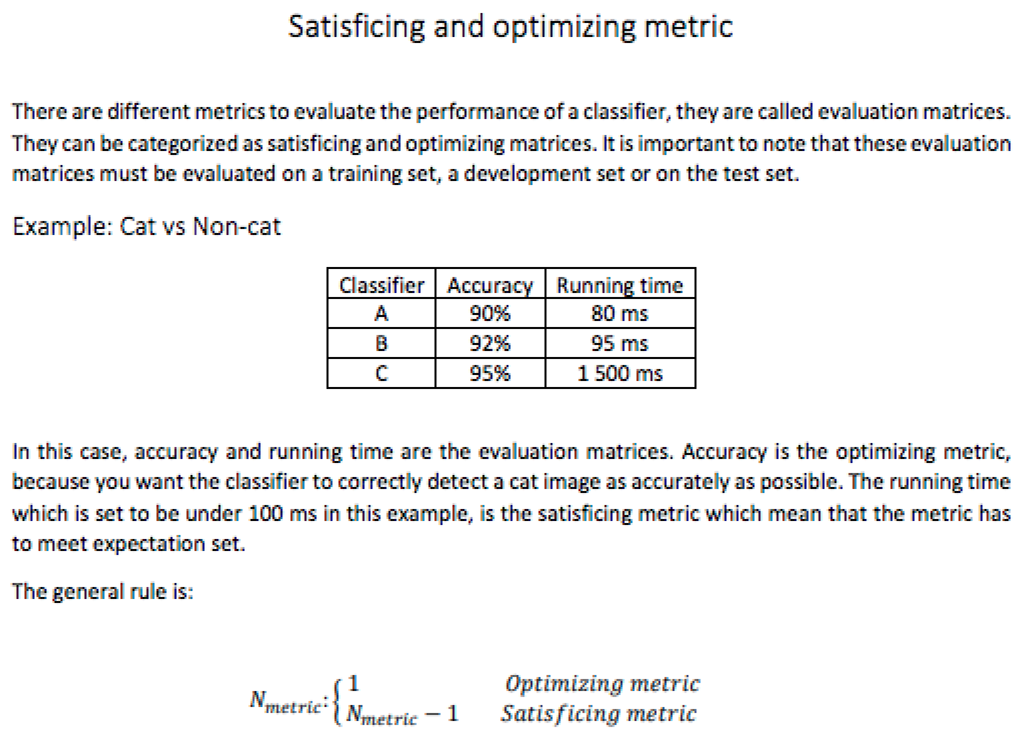
For classifier A, there is a 95% chance that there is a cat in the image and a 90% chance that it has correctly detected a cat. Whereas for classifier B there is a 98% chance that there is a cat in the image and a 85% chance that it has correctly detected a cat.

The problem with using precision/recall as the evaluation metric is that you are not sure which one is better since in this case, both of them have a good precision et recall. F1-score, a harmonic mean, combine both precision and recall.

F1-Score= $\frac{2}{\frac{1}{p}+\frac{1}{r}}$

Classifier	Precision (p)	Recall (r)	F1-Score
A	95%	90%	92.4 %
B	98%	85%	91.0%

Classifier A is a better choice. F1-Score is not the only evaluation metric that can be use, the average, for example, could also be an indicator of which classifier to use.



Satisficing and optimizing metric

There are different metrics to evaluate the performance of a classifier, they are called evaluation matrices. They can be categorized as satisficing and optimizing matrices. It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.

Example: Cat vs Non-cat

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1 500 ms

In this case, accuracy and running time are the evaluation matrices. Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example, is the satisficing metric which mean that the metric has to meet expectation set.

The general rule is:

$$N_{metric}:\begin{cases} 1 & Optimizing\ metric \\ N_{metric}-1 & Satisficing\ metric \end{cases}$$

TRY DIFFERENT MODELS ON TRAINING SET, EVALUATE ON DEV SET -- PICK ONE AND EVALUATE ON TEST SET.

Perplexity

Entropy is $-\sum_i P_i log P_i$

Defined such that likely events have low entropy, unlikely high entropy and independent events are additive.

Compare a toss of a fair coin with a roll of a fair dice

Coin S = $-\ 2\ x\ (1/2 log_2 2^{-1}\ =\ 1$

Dice S = $-\ 6\ x\ 1/6\ log_2 6^{-1}\ =\ 2.58$ More disorder

The best language model is one that best predicts an unseen test set

Gives the highest Probability for sentence $PP(W)\ =\ P(w_1 w_2 ... w_N)^{-1/N}$ =

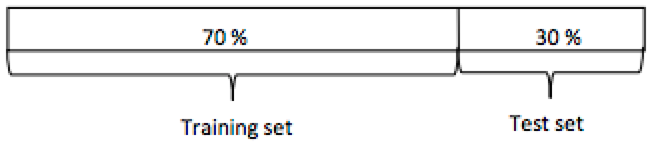
Perplexity which is: inverse probability of the test set normalized by number of words = $\sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$

Minimizing perplexity is equivalent to maximizing probability

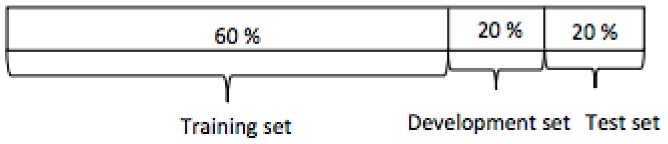
Size of the development and test sets

Old way of splitting data

We had smaller data set therefore we had to use a greater percentage of data to develop and test ideas and models.

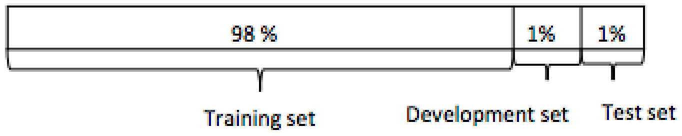


Or



Modern era – Big data

Now, because a large amount of data is available, we don't have to compromised as much and can use a greater portion to train the model.



Guidelines

- Set up the size of the test set to give a high confidence in the overall performance of the system.
- Test set helps evaluate the performance of the final classifier which could be less 30% of the whole data set.
- The development set has to be big enough to evaluate different ideas.

When to change development/test sets and metrics

Example: Cat vs Non-cat

A cat classifier tries to find a great amount of cat images to show to cat loving users. The evaluation metric used is a classification error.

Algorithm	Classification error [%]
A	3%
B	5%

It seems that Algorithm A is better than Algorithm B since there is only a 3% error, however for some reason, Algorithm A is letting through a lot of the pornographic images.

Algorithm B has 5% error thus it classifies fewer images but it doesn't have pornographic images. From a company's point of view, as well as from a user acceptance point of view, Algorithm B is actually a better algorithm. The evaluation metric fails to correctly rank order preferences between algorithms. The evaluation metric or the development set or test set should be changed.

The misclassification error metric can be written as a function as follow:

$$Error : \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} \mathcal{L}\{(\hat{y}^{(i)} \neq y^{(i)}\}$$

This function counts up the number of misclassified examples.

The problem with this evaluation metric is that it treats pornographic vs non-pornographic images equally. On way to change this evaluation metric is to add the weight term $w^{(i)}$.

$$w^{(i)} = \begin{cases} 1 & if\ x^{(i)}\ is\ non-pornographic \\ 10 & if\ x^{(i)}\ is\ pornographic \end{cases}$$

The function becomes:

$$Error : \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{(\hat{y}^{(i)} \neq y^{(i)}\}$$

Guideline

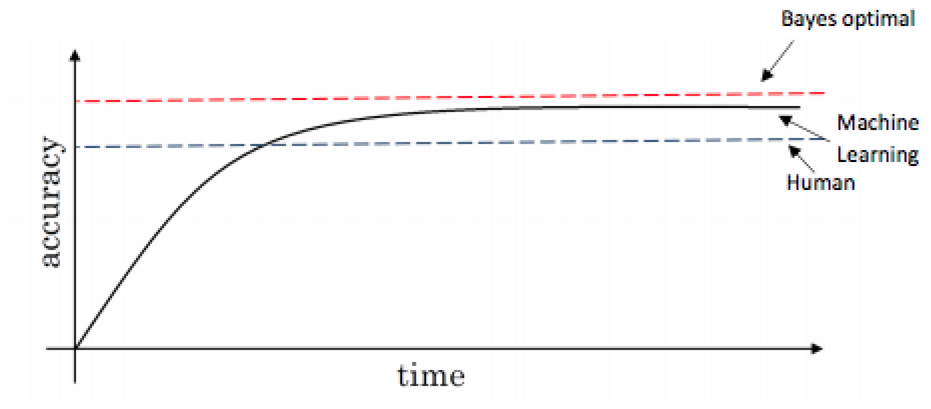
1. Define correctly an evaluation metric that helps better rank order classifiers
2. Optimize the evaluation metric

Why human-level performance?

Today, machine learning algorithms can compete with human-level performance since they are more productive and more feasible in a lot of application. Also, the workflow of designing and building a machine learning system, is much more efficient than before.

Moreover, some of the tasks that humans do are close to "perfection", which is why machine learning tries to mimic human-level performance.

The graph below shows the performance of humans and machine learning over time.



The

Machine learning progresses slowly when it surpasses human-level performance. One of the reason is that human-level performance can be close to Bayes optimal error, especially for natural perception problem.

Bayes optimal error is defined as the best possible error. In other words, it means that any functions mapping from x to y can't surpass a certain level of accuracy.

Also, when the performance of machine learning is worse than the performance of humans, you can improve it with different tools. They are harder to use once its surpasses human-level performance.

These tools are:

- Get labeled data from humans
- Gain insight from manual error analysis: Why did a person get this right?
- Better analysis of bias/variance.

Avoidable bias

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

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.

Understanding human-level performance

Human-level error gives an estimate of Bayes error.

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

Surpassing human-level performance

Example1: Classification task

	Classification error (%)	
	Scenario A	Scenario B
Team of humans	0.5	0.5
One human	1.0	1
Training error	0.6	0.3
Development error	0.8	0.4

Scenario A

In this case, the Bayes error is 0.5%, therefore the available bias is 0.1% et the variance is 0.2%.

Scenario B

In this case, there is not enough information to know if bias reduction or variance reduction has to be done on the algorithm. It doesn't mean that the model cannot be improve, it means that the conventional ways to know if bias reduction or variance reduction are not working in this case.

There are many problems where machine learning significantly surpasses human-level performance, especially with structured data:

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals

Improving your model performance

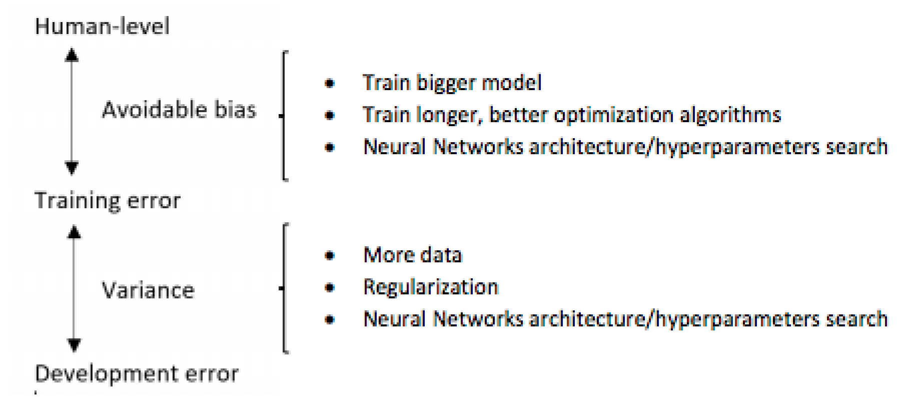
The two fundamental assumptions of supervised learning

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.

Summary


Human-level
Avoidable bias
Train bigger model
Train longer, better optimization algorithms
Neural Networks architecture/hyperparameters search
Training error
Variance
More data
Regularization
Neural Networks architecture/hyperparameters search
Development error

Build system quickly, then iterate

Depending on the area of application, the guideline below will help you prioritize when you build your system.

Guideline

1. Set up development/ test set and metrics
   - Set up a target
2. Build an initial system quickly
   - Train training set quickly: Fit the parameters
   - Development set: Tune the parameters
   - Test set: Assess the performance
3. Use Bias/Variance analysis & Error analysis to prioritize next steps

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.

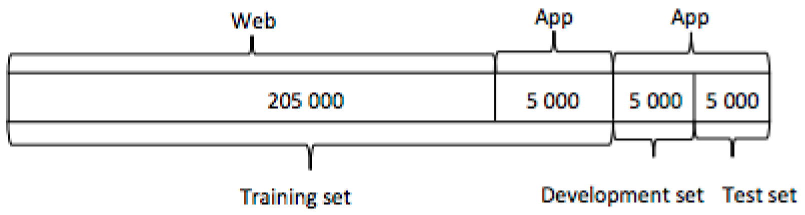
There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded 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	Scenar
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

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 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 what 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.

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.

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

$$\begin{tikzcd}
\text{Bayes error} \arrow[d, leftrightarrow, "\text{Avoidable Bias}"] \\
\text{Training set error} \arrow[d, leftrightarrow, "\text{Variance}"] \\
\text{Development - Training set error} \arrow[d, leftrightarrow, "\text{Data mismatch}"] \\
\text{Development set error} \arrow[d, leftrightarrow, "\text{Degree of overfitting to the development set}"] \\
\text{Test set error}
\end{tikzcd}$$

Addressing data mismatch

This is a general guideline to address data mismatch:

- Perform manual error analysis to understand the error differences between training, development/test sets. Development should never be done on test set to avoid overfitting.

- 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 is 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.

Optimization

Gradient Descent An overview of gradient descent optimization algorithms
Sebastian Ruder Sept 2016 updated in 2018

Abstract: Gradient descent is an optimization algorithm for finding a local minimum of a differential function. Cauchy in 1847 is attributed to this algorithm. Several approaches to optimize the algorithm for Deep Learning are: 1) Stochastic gradient descent calculates the gradient on each training sample for fast convergence, 2) Executing on mini-batches reduces the variance and leads to stable convergence, 3) Best results from combining Stochastic gradient descent on mini-batches, 4) a Learning rate η which often varies (annealing) is used to improve convergence. Adam is a good choice for adapting the learning rate.

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $\theta \in \mathbb{R}^d$: model parameters
  - η: learning rate
  - $\nabla_\theta J(\theta)$: gradient of the objective function with regard to the parameters
- Updates parameters **in opposite direction** of gradient.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

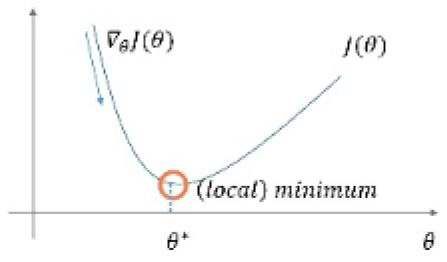
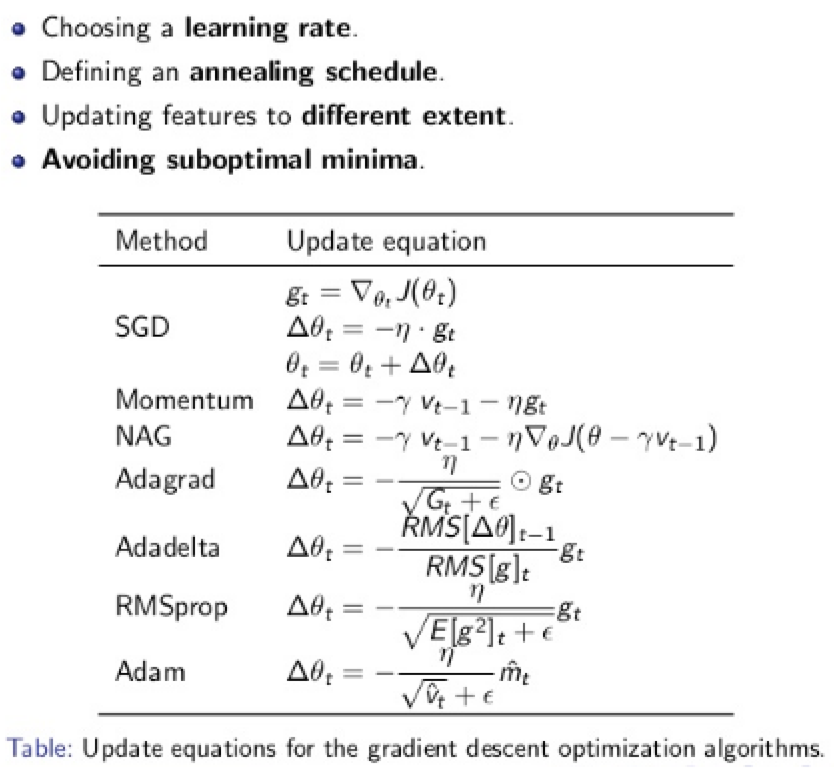


Figure: Optimization with gradient descent

Annealing is making the learning rate smaller

- Adaptive learning rate methods (Adagrad, Adadelta, RMSprop, Adam) are **particularly useful for sparse features**.
- Adagrad, Adadelta, RMSprop, and Adam work well in similar circumstances.
- [Kingma and Ba, 2015] show that bias-correction helps Adam **slightly outperform RMSprop**.



- Choosing a **learning rate**.
- Defining an **annealing schedule**.
- Updating features to **different extent**.
- **Avoiding suboptimal minima**.

Method	Update equation
	$g_t = \nabla_{\theta_t} J(\theta_t)$
SGD	$\Delta\theta_t = -\eta \cdot g_t$
	$\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\ v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\ v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

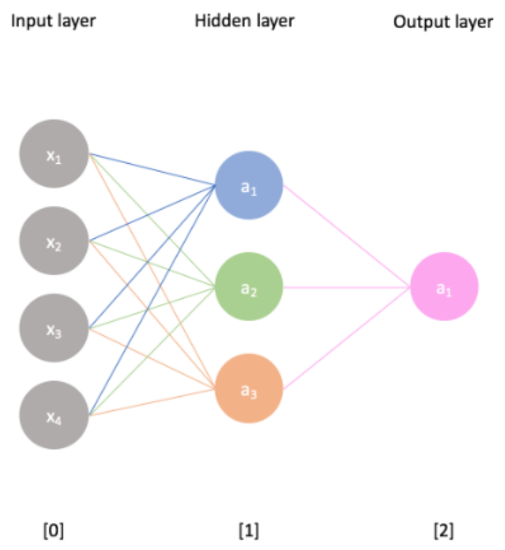
Table: Update equations for the gradient descent optimization algorithms.

Development and extensive options are available at:
https://ruder.io/optimizing-gradient-descent/index.html

“Early stopping (is) beautiful free lunch” Geoff Hinton

Forward and Back Propagation

Deep Learning Representation



I'll use the superscript $[l]$ to refer to the l^{th} layer of the network and the subscript i to refer to the i^{th} neuron in a layer.

For example, $a_2^{[1]}$ represents the activation of the second neuron in the first hidden layer. We can calculate this value by first combining the proper weights and bias with the previous layer's values

$$z_2^{[1]} = w_2^{[1]\mathrm{T}} a^{[0]} + b_2$$

and then passing this through our activation function, $g(z)$. Notice how each neuron combines *every* value from the previous layer as input.

Note: Our input vector, x, can also be referred to as the activations of the 0^{th} layer.

More generally, we can calculate the activation of neuron i in layer l.

$$z_i^{[l]} = w_i^{[l]\mathrm{T}} a^{[l-1]} + b_i^{[l]}$$

$$a_i^{[l]} = g\left(z_i^{[l]}\right)$$

Similarly, we can calculate all of the activations for a given layer l by using our weight matrix $W^{[l]}$.

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g\left(Z^{[l]}\right)$$

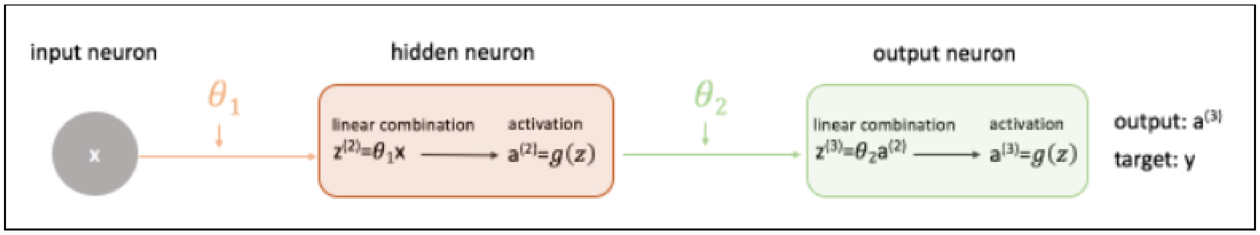
Steps for Training a Neural Net

1. Random Initialize weights
2. Forward propagation
3. Implement cost function
4. Backpropagation with partial derivatives
5. Gradient checking to validate
6. Gradient descent to minimize cost function

Forward Propagation: Given randomly initialized weights (now defined as θ) connecting each of the neurons, apply the matrix of observations to calculate the outputs of the neural network. Given the random initial weights, the output will probably be far off.

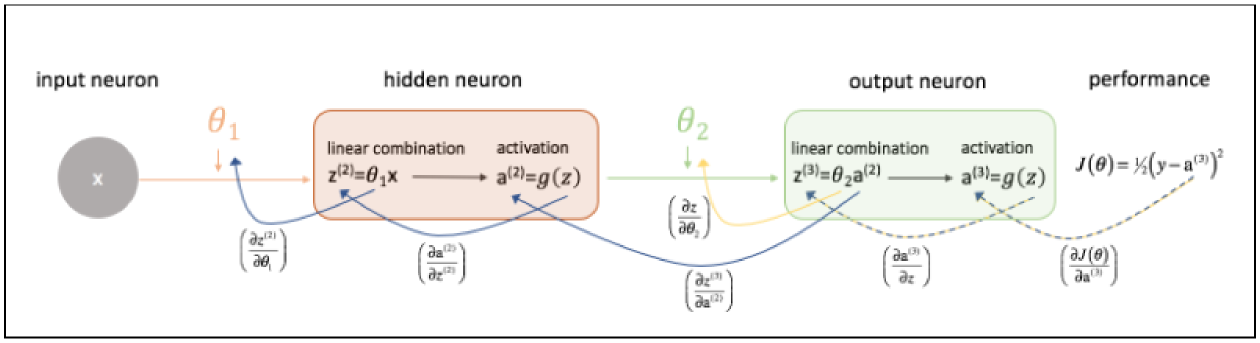
Simplify the network complexity by showing a linear network with a weight matrix $W^{[l\}}$ with elements θ . Define a cost function:

$$J(\theta) = \frac{1}{2}(y - a^{(3)})^2$$



Back Propagation: Now we use the chain rule for partial derivatives to move 'backward' from the cost function to define each of the weights.

$$\frac{\partial J(\theta)}{\partial \theta_2} = \left(\frac{\partial J(\theta)}{\partial a^{(3)}}\right)\left(\frac{\partial a^{(3)}}{\partial z}\right)\left(\frac{\partial z}{\partial \theta_2}\right)$$


input neuron
hidden neuron
output neuron
performance
x
θ_1
θ_2
linear combination
$z^{(2)}=\theta_1 x$
activation
$a^{(2)}=g(z)$
linear combination
$z^{(3)}=\theta_2 a^{(2)}$
activation
$a^{(3)}=g(z)$
$J(\theta)=\frac{1}{2}(y-a^{(3)})^2$
$\left(\frac{\partial z^{(2)}}{\partial \theta_1}\right)$
$\left(\frac{\partial a^{(2)}}{\partial z^{(2)}}\right)$
$\left(\frac{\partial z}{\partial \theta_2}\right)$
$\left(\frac{\partial z^{(3)}}{\partial a^{(2)}}\right)$
$\left(\frac{\partial a^{(3)}}{\partial z}\right)$
$\left(\frac{\partial J(\theta)}{\partial a^{(3)}}\right)$

It is more complex because there are multiple rows (neurons) and cross terms, but the idea is clear.

Automatic Differentiation

Dual Numbers can be used to calculate the exact derivative of any function without requiring the calculation of derivatives. Implementing Dual Numbers is straightforward; Define a Dual Type, and define the methods for every operator {+, *, ^, trigonometric etc}.

Automatic Differentiation (AD) is a method to calculate the derivative of functions. This is critical because Derivatives play a fundamental role in mathematics, physics, and Data Science. Examples are:

- Gradients of cost functions used in optimization, parameter estimation and training of machine learning algorithms.
- Jacobian matrices required to solve systems of differential equations.
- Uncertainty and error propagation through models.

The advantages of AD with respect to other techniques are:

1. It works with any model as long as the individual functions and data structures are supported by the AD implementation.
2. It introduces no numerical error and the derivatives are calculated with the same accuracy as the outputs.

Therefore; an AD package can calculate the derivatives of any function accurately and easily. It can even calculate the derivatives of functions for which the source code is not available or is too complex to analyze for a human. The type of function does not matter: it could be a simple formula, a system of differential equations, an agent based model or any other kind of algorithm.

There are different types of AD techniques (the website http://www.autodiff.org has a list of AD tools), but here we address AD built on Dual Numbers.

1) A Dual Number y is defined as: y = a + b*ϵ such that ϵ > 0 and ϵ^2 = 0
This is analogous to complex numbers with ϵ playing the role of i; however, remember i^2 = -1. Therefore; multiplication and division is defined as with complex numbers.

Here is how we apply Dual Numbers to Differentiation:

For y = *f*(x) we add an infinitesimal to each side:

2) y + $\frac{\partial y}{\partial x} dx$ = *f*(x) + *f'*(x)dx where *f'*(x) = $\frac{\partial y}{\partial x}$. This means we can represent y as a Dual Number ***where the attached dimension represents the derivative!***

Using the definition of Dual Number (1) we have a = ***y = f(x)*** and using (2) ***f'(x) =*** $\frac{\partial y}{\partial x}$!

A Julia implementation is in my github.

State-of-Art ML

A summary of major Innovations is:

1. Models are trained on large corpuses using Unsupervised training
2. Models are customizable to multiple tasks by fine-tuning one final layer
3. Transfer learning is performed from Supervised Data {my FINBert}

The problem is that BenchMarks, until recently, did not account for training time and number of parameters. Therefore; many models are only of academic interest. MobileBERT is designed for mobile devices and is task agnostic as is BERT and is 4.3x smaller and 5.5x faster then BERT.

Attention: https://arxiv.org/abs/1706.03762
BERT: https://arxiv.org/abs/1810.04805
MobileBERT: https://arxiv.org/abs/2004.02984
For VISION
EfficientNetV2: https://arxiv.org/pdf/2104.00298.pdf

My 'benchmark' for language understanding is a novel. A novel has character and plot development, the interaction of characters and plot and most difficult for NLP is relationships spanning ***multiple pages***. Current NLP is far from this 'benchmark'. However; Transformer takes a significant step towards understanding at the 'paragraph' level versus the prior champion LSTM. ***Transformer {BERT}*** leverages ***Attention*** and is the dominant analytics for NLP and increasingly used for Vision.

An important note to the following is that convolution is the 'multiplication' of two functions over the Real Numbers. Convolution originated in D'Alembert's work of 1754 and was later extensively employed by the famous French mathematicians, Pierre Simon Laplace, Jean-Baptiste Joseph Fourier, and Simeon Denis Poisson.

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU, ByteNet and ConvS2S, all of which use convolutional neural networks as basic building blocks, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention.

Self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution.

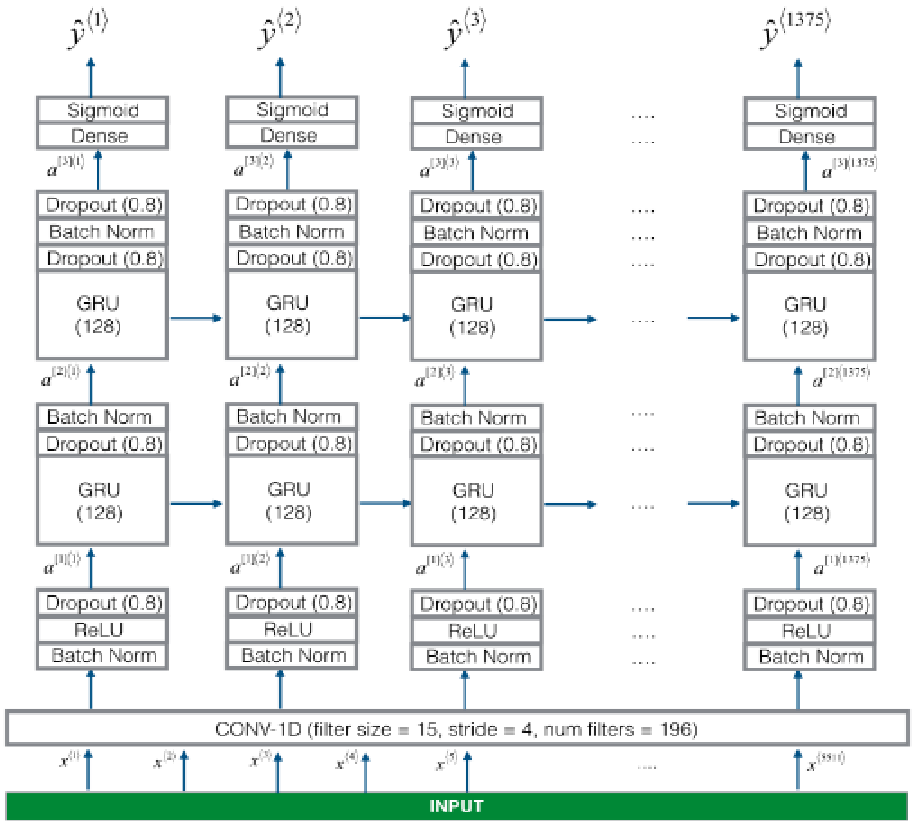
Most competitive neural sequence transduction models have an encoder-decoder structure. Here, the encoder maps an input sequence of symbol representations $(x_1, . . ., x_n)$ to a sequence of continuous representations z = $(z_1, . . . , z_n)$. Given z, the decoder then generates an output sequence $(y_1, . . . , y_n)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

Both the encoder and decoder are constructed with 6 identical layers employing a ‘normalized’ dot product and a final Softmax layer for transformation to probabilities. I will spare you the headache from the architectural diagram which is available here:
http://nlp.seas.harvard.edu/annotated-transformer/

Executed Projects

Trigger Word Detection from spectrogram*:* Convert audio to spectrogram (fourier transform). Generate training examples over background noise tape using positive and negative non-overlapping samples. Uses convolution to down-sample spectrograms.

1. CONV-1D processes spectrogram (downsizes 5511 to 1375)
2. Batch Norm / ReLU / Dropout(0.8)
3. GRU (gated recurrent unit) / Dropout(0.8) / Batch Norm
4. GRU (gated recurrent unit) / Dropout(0.8) / Batch Norm
5. Dense / Sigmoid (YES / NO)


$\hat{y}^{\langle 1 \rangle}$
$\hat{y}^{\langle 2 \rangle}$
$\hat{y}^{\langle 3 \rangle}$
$\hat{y}^{\langle 1375 \rangle}$
Sigmoid
Dense
Dropout (0.8)
Batch Norm
Dropout (0.8)
GRU (128)
Batch Norm
Dropout (0.8)
GRU (128)
Dropout (0.8)
ReLU
Batch Norm
CONV-1D (filter size = 15, stride = 4, num filters = 196)
INPUT

One key step of this model is the 1D convolutional step (near the bottom of Figure 3). It inputs the 5511 step spectrogram, and outputs a 1375 step output, which is then further processed by multiple layers to get the final $T_y = 1375$ step output. This layer plays a role similar to the 2D convolutions you saw in Course 4, of extracting low-level features and then possibly generating an output of a smaller dimension.

Computationally, the 1-D conv layer also helps speed up the model because now the GRU has to process only 1375 timesteps rather than 5511 timesteps. The two GRU layers read the sequence of inputs from left to right, then ultimately uses a dense+sigmoid layer to make a prediction for $y^{\langle t \rangle}$. Because y is binary valued (0 or 1), we use a sigmoid output at the last layer to estimate the chance of the output being 1, corresponding to the user having just said "activate."

Note that we use a uni-directional RNN rather than a bi-directional RNN. This is really important for trigger word detection, since we want to be able to detect the trigger word almost immediately after it is said. If we used a bi-directional RNN, we would have to wait for the whole 10sec of audio to be recorded before we could tell if "activate" was said in the first second of the audio clip.

Dinosaurus Island -- Creating dinosaur names from characters

Description: Using corpus of dinosaur names break these into characters (27 w /n -- for softmax). RNN with $sample(y^{(t)}) = x^{(t-1)}$(diagram). Train the model on the corpus using stochastic gradient descent.

- **Step 1**: Pass the network the first "dummy" input $x^{\langle 1 \rangle} = \vec{0}$ (the vector of zeros). This is the default input before we've generated any characters. We also set $a^{\langle 0 \rangle} = \vec{0}$
- **Step 2**: Run one step of forward propagation to get $a^{\langle 1 \rangle}$ and $\hat{y}^{\langle 1 \rangle}$. Here are the equations:

$$a^{\langle t+1 \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t \rangle} + b) \tag{1}$$

$$z^{\langle t+1 \rangle} = W_{ya}a^{\langle t+1 \rangle} + b_y \tag{2}$$

$$\hat{y}^{\langle t+1 \rangle} = softmax(z^{\langle t+1 \rangle}) \tag{3}$$

Note that $\hat{y}^{\langle t+1 \rangle}$ is a (softmax) probability vector (its entries are between 0 and 1 and sum to 1). $\hat{y}_i^{\langle t+1 \rangle}$ represents the probability that the character indexed by "i" is the next character. We have provided a `softmax()` function that you can use.

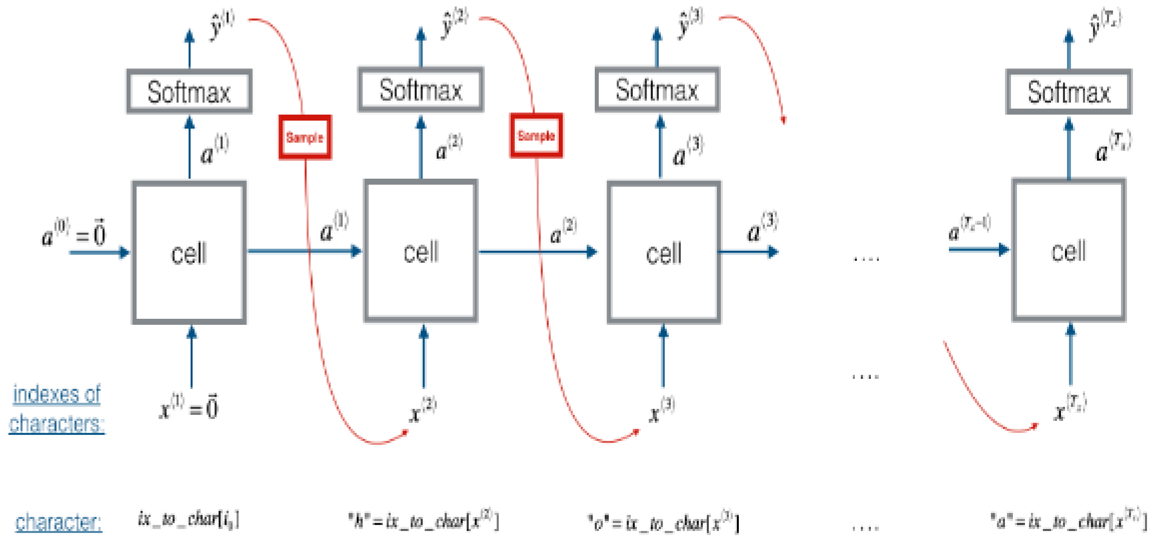
- **Step 3**: Carry out sampling: Pick the next character's index according to the probability distribution specified by $\hat{y}^{\langle t+1 \rangle}$. This means that if $\hat{y}_i^{\langle t+1 \rangle} = 0.16$, you will pick the index "i" with 16% probability. To implement it, you can use `np.random.choice`.

Here is an example of how to use `np.random.choice()`:

```
np.random.seed(0)
p = np.array([0.1, 0.0, 0.7, 0.2])
index = np.random.choice([0, 1, 2, 3], p = p.ravel())
```

This means that you will pick the `index` according to the distribution:
$P(index = 0) = 0.1, P(index = 1) = 0.0, P(index = 2) = 0.7, P(index = 3) = 0.2.$

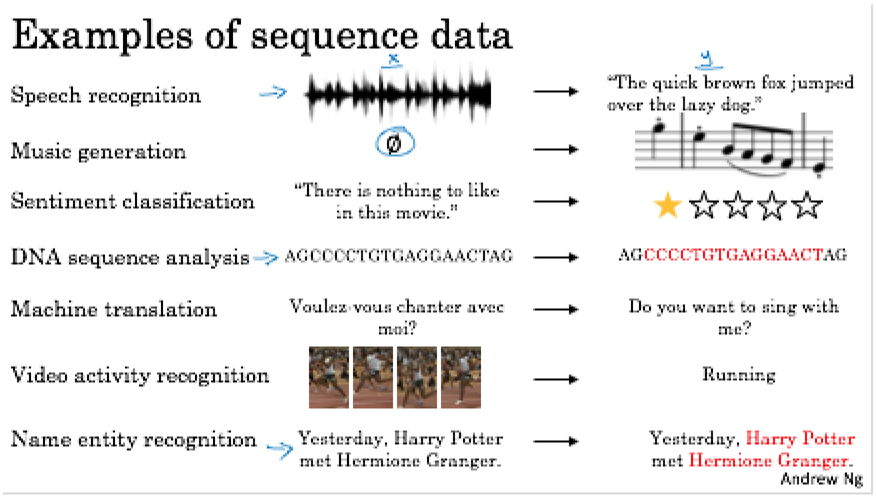
- **Step 4**: The last step to implement in `sample()` is to overwrite the variable `x`, which currently stores $x^{\langle t \rangle}$, with the value of $x^{\langle t+1 \rangle}$. You will represent $x^{\langle t+1 \rangle}$ by creating a one-hot vector corresponding to the character you've chosen as your prediction. You will then forward propagate $x^{\langle t+1 \rangle}$ in Step 1 and keep repeating the process until you get a "\n" character, indicating you've reached the end of the dinosaur name.


Softmax
Sample
cell
$a^{\langle 0 \rangle} = \vec{0}$
$x^{\langle 1 \rangle} = \vec{0}$
$\hat{y}^{\langle 1 \rangle}$
$a^{\langle 1 \rangle}$
$x^{\langle 2 \rangle}$
$a^{\langle T_x \rangle}$
$\hat{y}^{\langle T_x \rangle}$
indexes of characters:
character:
$ix\_to\_char[i_0]$
"h" = $ix\_to\_char[x^{\langle 2 \rangle}]$
"o" = $ix\_to\_char[x^{\langle 3 \rangle}]$
"a" = $ix\_to\_char[x^{\langle T_x \rangle}]$

ChatBox DialogFlow (Google)
https://console.dialogflow.com/api-client/#/agent/f0df7906-77f0-4e36-a4d9-02e182ace8be/intents

Generate Shakespeare Generate Shakespeare poems from corpus of The Sonnets

Generate Music (LSTM) Generate music from corpus of music. Notes are represented by 78 unique values. Model.fit trains the model.



Examples of sequence data

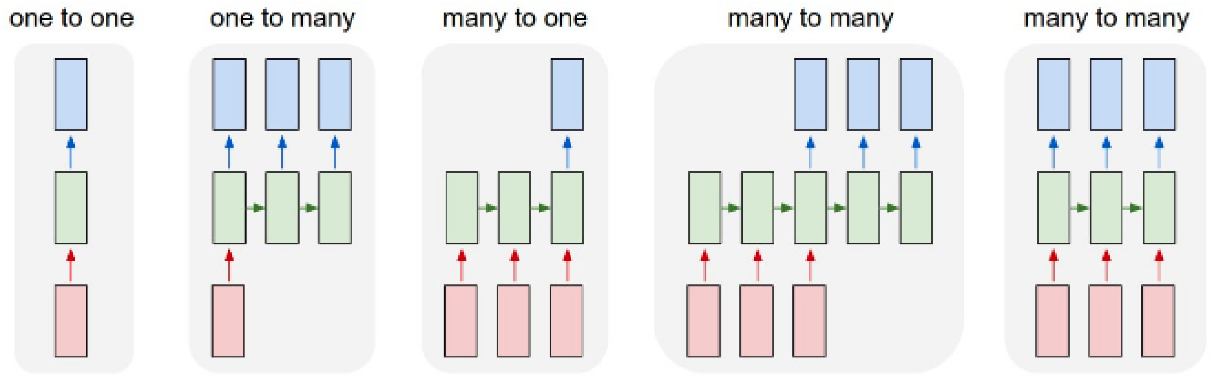
Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation	∅	
Sentiment classification	"There is nothing to like in this movie."	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

Encoder / Decoder SeqToSeq Architecture with Attention & Beam Search
https://guillaumegenthial.github.io/sequence-to-sequence.html

1. Encoder 'captures meaning' in 1 vector
   1.1. Words are input in a sequence outputing 1 vector
   1.2. ***Attention*** introduces context vector c which through a function (dot, Tensor, concat) generates unnormalized weights which are normalized by softmax
      1.2.1. Weights near 1 for 'relevant' words 0 for others
2. Decoder produces sequence of words
   2.1. Training approach is to feed in the next word as input
   2.2. Greedy Search simply performs softmax giving percentages across ALL vocabulary elements and selects argmax
   2.3. ***Beam Search*** retains k hypothesis th

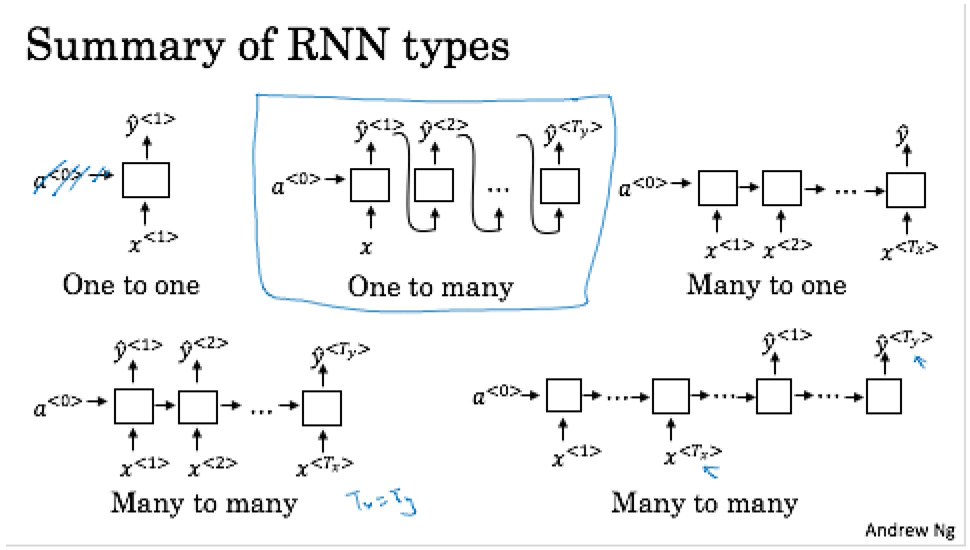
RNNs are capable of a number of different types of input / output combinations, as seen below



The `TimeDistributedDense` layer allows you to build models that do the *one-to-many* and *many-to-many* architectures. This is because the output function for each of the "many" outputs must be the same function applied to each timestep. The `TimeDistributedDense` layers allows you to apply that Dense function across every output over time. This is important because it needs to be the *same* dense function applied at every time step.

If you didn't not use this, you would only have one final output - and so you use a normal dense layer. This means you are doing either a *one-to-one* or a *many-to-one* network, since there will only be one dense layer for the output.

rough the process



Commercial Vocabulary on order of 50,000 to 100,000

A language model determines the probability of a sentence using principle of multiplicative probabilities

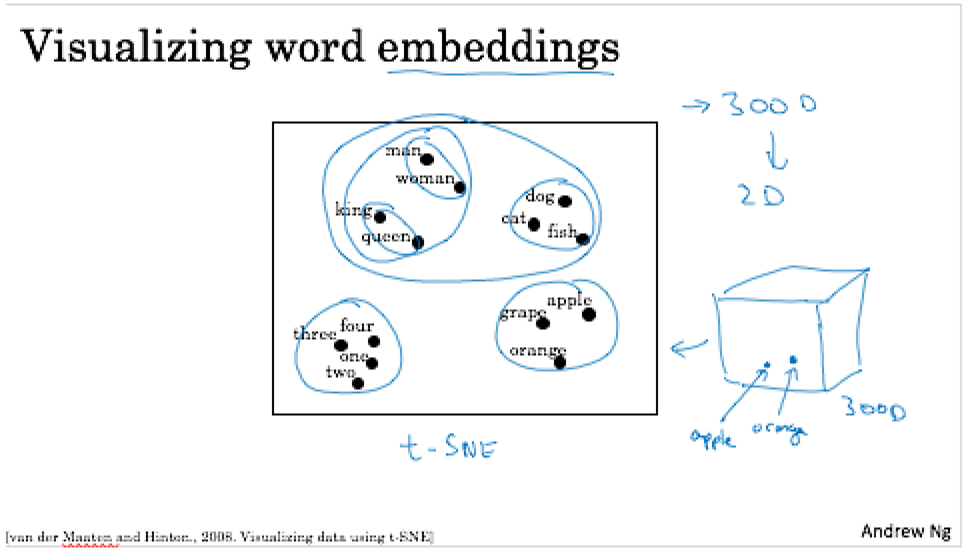
$$P(y^{<1>}, y^{<2>}, y^{<3>}) \xleftarrow{\text{way}}$$
$$= \underline{P(y^{<1>})} \, \underline{P(y^{<2>} | y^{<1>})}$$
$$\underline{P(y^{<3>} | y^{<1>}, y^{<2>})}$$

Sequence Generation you simple sample from the softmax

Vanishing Gradients LSTM. Exploding Gradients Gradient Clipping

Word Embeddings 50 - 1,000 dimensions

Visualize word embeddings with t-SAE maps 300 dimensions in non-linear way to 2D



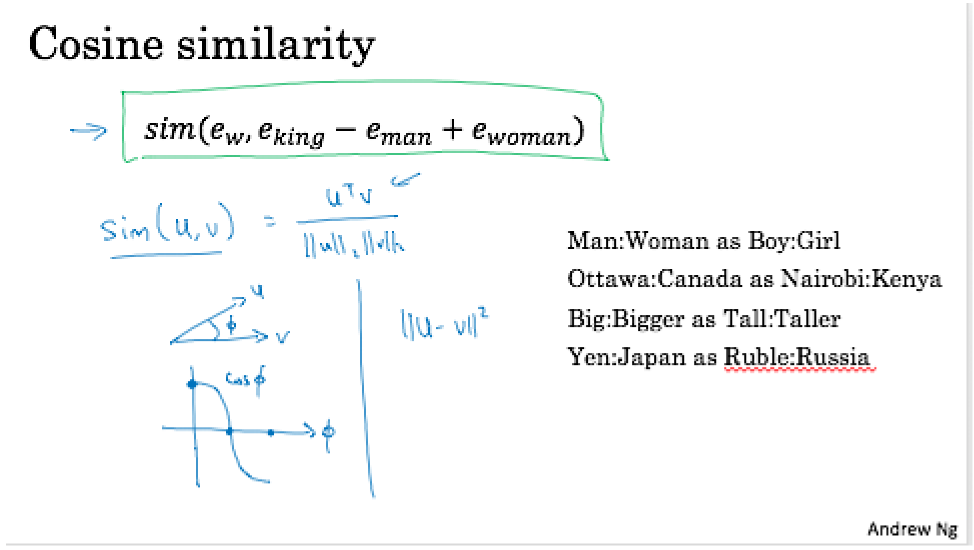
Visualizing word embeddings

[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Andrew Ng

Approaches to Learning Word Embeddings

Cosine similarity



Cosine similarity

$$\rightarrow \boxed{sim(e_w, e_{king} - e_{man} + e_{woman})}$$

$$\underline{sim(u,v)} = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

Man:Woman as Boy:Girl

Ottawa:Canada as Nairobi:Kenya

Big:Bigger as Tall:Taller

Yen:Japan as Ruble:Russia

Andrew Ng

In Keras, the embedding matrix is represented as a "layer", and maps positive integers (indices corresponding to words) into dense vectors of fixed size (the embedding vectors). It can be trained or initialized with a pretrained embedding. In this part, you will learn how to create an Embedding() layer in Keras, initialize it with the GloVe 50-dimensional vectors loaded earlier in the notebook. Because our training set is quite small, we will not update the word embeddings but will instead leave their values fixed. But in the code below, we'll show you how Keras allows you to either train or leave fixed this layer.

The `Embedding()` layer takes an integer matrix of size (batch size, max input length) as input. This corresponds to sentences converted into lists of indices (integers), as shown in the figure below.

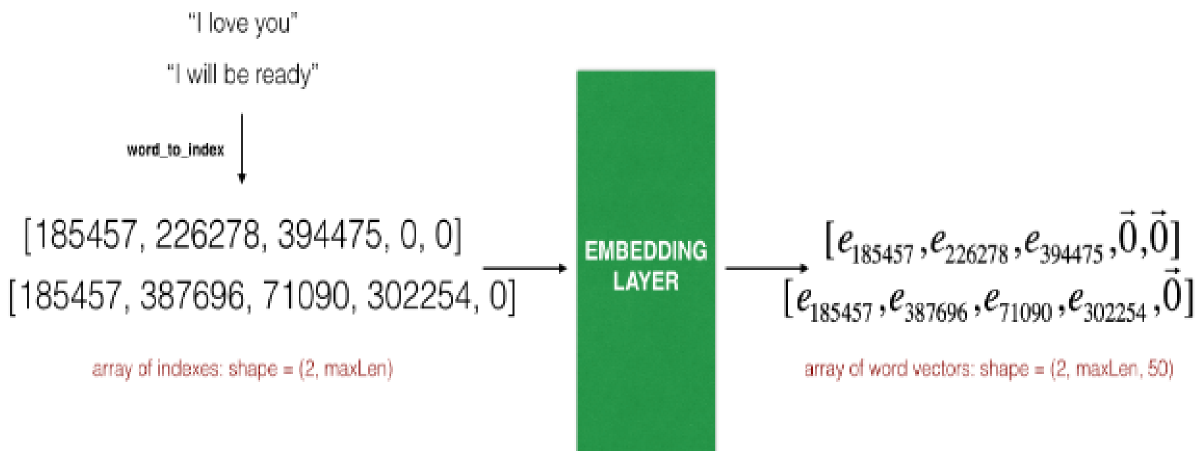

"I love you"
"I will be ready"
word_to_index
[185457, 226278, 394475, 0, 0]
[185457, 387696, 71090, 302254, 0]
array of indexes: shape = (2, maxLen)
EMBEDDING LAYER
$[e_{185457}, e_{226278}, e_{394475}, \vec{0}, \vec{0}]$
$[e_{185457}, e_{387696}, e_{71090}, e_{302254}, \vec{0}]$
array of word vectors: shape = (2, maxLen, 50)

Figure 4: Embedding layer. This example shows the propagation of two examples through the embedding layer. Both have been zero-padded to a length of `max_len=5`. The final dimension of the representation is `(2,max_len,50)` because the word embeddings we are using are 50 dimensional.

Fundamental Books

Machine Learning A Probabilistic Perspective
by Kevin P. Murphy

Deep Learning
By Ian Goodfellow, Yoshua Bengio, and Aaron Courville -- papers before recently published book.

Probabilistic Graphical Models: Principles and Techniques
Daphne Koller and Nir Friedman

Learning From Data
By Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuant-Tien Lin

Data Analysis: A Bayesian Tutorial
By D.S. Sivia

Advances in Financial Machine Learning
By Marcos Lopez De Prado