

Portfolio of ML / Deep Learning

May 2016 Updated November 2018 Update March 2019

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, Intelligence, Quantitative Finance, and Risk Management. Singular Value Decomposition (SVD), Time Series, and Markov Chain Monte Carlo (MCMC) algorithms are established; while Deep Learning is rapidly improving and Reinforcement Learning is maturing. Therefore, I attempt to update the Deep Learning approaches of Convolutional Neural Networks, Natural Language Processing (NLP), Recurrent Neural Networks (RNN), Collaborative Filtering, and Reinforcement Learning to current best practices.

The work derives from my Top Secret work in the Intelligence Community and Corporate work protected by non-disclosure agreements. Therefore, I default to general descriptions expressed in business terms followed by a quantitative description of the approach. I executed this work leading small distributed teams using Agile methods on Cloud environments.

My recent 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 Stern paper on AI and the // Comparison to steam in Industrial Revolution
3. MIT is committing \$1 Billion for a new school of AI
4. U of Virginia received a gift of \$120 Million for New School of Data Science
5. Google has commercialized over 4,000 Deep Learning algorithms.
6. The 2009 \$1 Million dollar Netflix Competition Collaborative Filtering Algorithm winner: A Recommender system for movies based on peoples ratings.
7. The **2012 ImageNet** triumph of Hinton's group using Deep Learning:
The margin of victory was 11% over competitors on identifying images
8. **2018, Ruder, NLP's ImageNet Moment has Arrived**
9. Bayesian Approach focused on Data & Learning / Originated with Laplace in 18'th C
10. Progress in Convolution networks // **Image Detection**
11. Progress in **Recurrent Neural Networks** (LSTM - Long-short term memory)
12. Combining the two previous Deep Learning approaches has revolutionized Natural Language Processing (NLP).
13. I am collaborating on a paper on **Volatility forecasting** using LSTM's exploiting my experience in NLP.
14. I am executing a **chatbot with Encoder / Decoder**
15. I am exploring the TensorFlow 'add-on' for Reinforcement Learning 'the next big thing', because it dynamically adapts to external stimulus.
16. **ABCD of FINTECH** AI / Blockchain / Cloud / Data

Summary of Approaches

1. Anomaly Detection
2. Dimensionality Reduction (PCA / SVD)
3. Recommender Systems
4. Bayesian Statistics
5. Markov Chain Monte Carlo (MCMC)
6. Cloud (AWS / Google) no devOps
7. Scala Architecture
8. Scala / Kafka / Spark / Cassandra Real Time Streaming
9. Deep Learning
 - 9.1. Convolution Neural Networks (parameter sharing)
 - 9.2. Recurrent Neural Networks (LSTM) (time distributed equivalent to parameter sharing)
 - 9.3. Encoder / Decoder (ChatBox)
10. Repeatable Research {Jupyter / Spark / Zeppelin} Notebook
11. Reinforcement Learning
12. Blockchain

HISTORY OF DEEP LEARNING

Power Realized in NLP Conferences in 2015 Christopher D Manning

Hidden Layers can find features

Conv Nets

1. Edges // Lines / circles
2. Position of lines etc
3. Position to find face etc.

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

2018, Ruder, NLP's ImageNet Moment has Arrived

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>

The 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 researches now employed by Microsoft and Google!

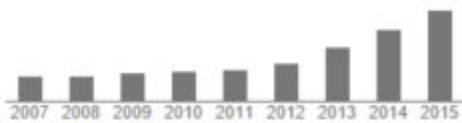
I have been working in this field for over 7 years, first in the Intelligence Community and recently at a startup, ETRADE, Comcast, and several Financial Firms.

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.

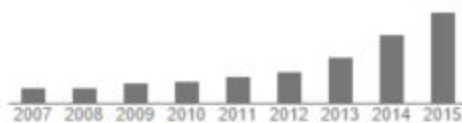
Citations for the Work of Deep Learning Pioneers ‘takes off’

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



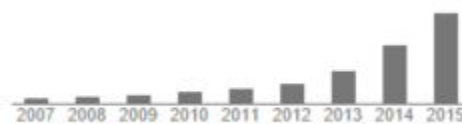
Geoffrey Hinton

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



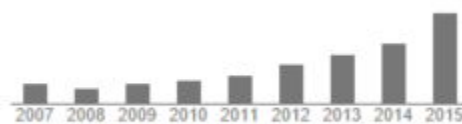
Yann LeCun

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

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 Commission, Comcast, and ETRADE.

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 : x_1 : number logins, x_2 : web pages visited, x_3 : number transactions x_4 : typing speed
Dataset : $\{x^1, x^2, \dots, x^m\}$ where x^m is a vector of features for a sample m of dimension n ($x \in \mathbb{R}^n$)

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).

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

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

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 squared

Efficient algorithms exist to calculate Singular Value Decomposition.

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

U, Σ, W are unique. U, W are column orthonormal $U^T U = I$, $W^T W = I$, Σ is diagonal with singular values $\sigma_1 \geq \sigma_2 \geq 0$

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

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

Recommender Systems

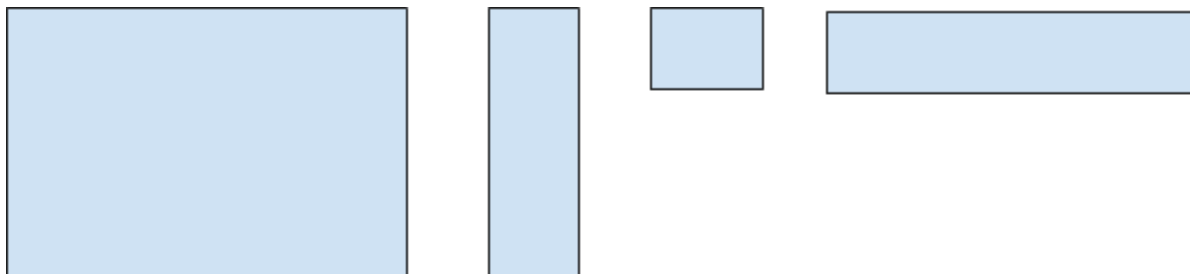
<https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75>

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) (If x If) (If x movies)



Bayesian Approach

<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.

$$p(\text{parameters} \mid \text{Data}) = \frac{P(\text{Data} \mid \text{parameters}) p(\text{parameters})}{p(\text{Data})} \quad \{\text{posterior} \propto \text{likelihood} \times \text{prior}\}$$

$$\text{with } p(\text{Data}) = \int p(\text{Data} \mid \text{parameters}) p(\text{parameters}) dp$$

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 ($p(\text{Data})$ - 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 ($p(\text{Data})$) using sampling.

Naive Bayes Classifier

Abstractly, naive Bayes is a [conditional probability](#) model: given a problem instance to be classified, represented by a vector $x = (x_1, \dots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities $p(C_k \mid x_1, \dots, x_n)$

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 $p(x_i \mid x_{i+1}, \dots, x_n, C_k) = p(x_i \mid C_k)$. Thus,

the joint model $p(C_k \mid x_1, \dots, x_n)$ is proportional to $p(C_k) \prod_{i=1}^n p(x_i \mid C_k)$

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.

Cloud Resources and Experience

My experience: All my enterprise analytics is performed in Cloud environments which support distributed streaming data (Kafka / Cassandra), analytics (Spark), and Deep Learning (Tensorflow / Keras). I have extensive experience on Google, Amazon, and private Clouds. My presentation to the Securities and Exchange Commission analyzes the advantages of serverless Cloud architectures.

Scala Architecture

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.

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.

I have implemented forward and back propagation in Python. I am currently exploring executing in Scala functional code.

Ng Deep Learning

Logistic Regression $\hat{y} = \sigma(W^T X + b)$

Do not use squared error because it is not convex

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

CROSS ENTROPY

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

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

Neural Network Notation

Do not count input layer in neural networks

For n_x neurons and m samples X is $x^{[1]}, \dots, x^{[m]}$ 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

W is oriented horizontally to prevent the need for a Transpose

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

Multiple Output using softmax for deep Neural Net

$Z^{[l]} = W^{[l]}X + b^{[l]}$ where $A^{[l]} = \text{softmax}(Z^{[l]})$ 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 number in range (0,1) which add up to 1.

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

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

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

- 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: $\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$

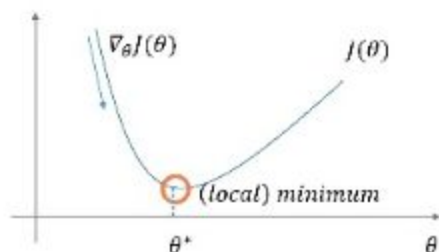


Figure: Optimization with gradient descent

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Difference: Amount of data used per update

- Performs update for every **mini-batch** of n examples.
- Update equation: $\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(
            loss_function, batch, params)
        params = params - learning_rate * params_grad
```

Listing 3: Code for mini-batch gradient descent update

- Pros
 - **Reduces variance** of updates.
 - Can exploit **matrix multiplication** primitives.
- Cons
 - **Mini-batch size** is a hyperparameter. Common sizes are 50-256.
- Typically the algorithm of choice.
- Usually referred to as SGD even when mini-batches are used.
- Choosing a **learning rate**.
- Defining an **annealing schedule**.
- Updating features to **different extent**.
- **Avoiding suboptimal minima**.

Annealing is making the learning rate smaller

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

Table: Update equations for the gradient descent optimization algorithms.

- Adaptive learning rate methods (Adagrad, Adadelata, RMSprop, Adam) are **particularly useful for sparse features**.
- Adagrad, Adadelata, RMSprop, and Adam work well in similar circumstances.
- [Kingma and Ba, 2015] show that bias-correction helps Adam **slightly outperform RMSprop**.
- Shuffling and Curriculum Learning [Bengio et al., 2009]
 - Shuffle training data after every epoch to **break biases**
 - Order training examples to **solve progressively harder problems**; infrequently used in practice
- Batch normalization [Ioffe and Szegedy, 2015]
 - **Re-normalizes every mini-batch** to zero mean, unit variance
 - Must-use for computer vision
- Early stopping
 - *"Early stopping (is) beautiful free lunch"* (Geoff Hinton)
- Gradient noise [Neelakantan et al., 2015]
 - Add Gaussian noise to gradient
 - Makes model **more robust to poor initializations**
- Many recent papers use **SGD with learning rate annealing**.
- SGD with tuned learning rate and momentum is **competitive with Adam** [Zhang et al., 2017b].
- Adam **converges faster**, but **underperforms SGD** on some tasks, e.g. Machine Translation [Wu et al., 2016].
- Adam with **2 restarts and SGD-style annealing** converges faster and outperforms SGD [Denkowski and Neubig, 2017].
- **Increasing the batch size** may have the same effect as decaying the learning rate [Smith et al., 2017].
- Deep Biaffine Attention for Neural Dependency Parsing [Dozat and Manning, 2017]
 - Adam with $\beta_1 = 0.9$, $\beta_2 = 0.9$
 - Report large positive impact on final performance of lowering β_2
- Attention is All You Need [Vaswani et al., 2017]
 - Adam with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$, learning rate η
 - $\eta = d_{\text{model}}^{-0.5} \cdot \min(\text{step_num}^{-0.5}, \text{step_num} \cdot \text{warmup_steps}^{-1.5})$
 - $\text{warmup_steps} = 4000$

Larger batch sizes improves training efficiency because they ship more data to computation units (e.g. GPU) at a time.

- Batch size
 - 32 to 1024 on GPUs. Pick numbers that are powers of two.
 - Increasing batch size by factor of N requires epoch number increase by factor of N to maintain number of updates.

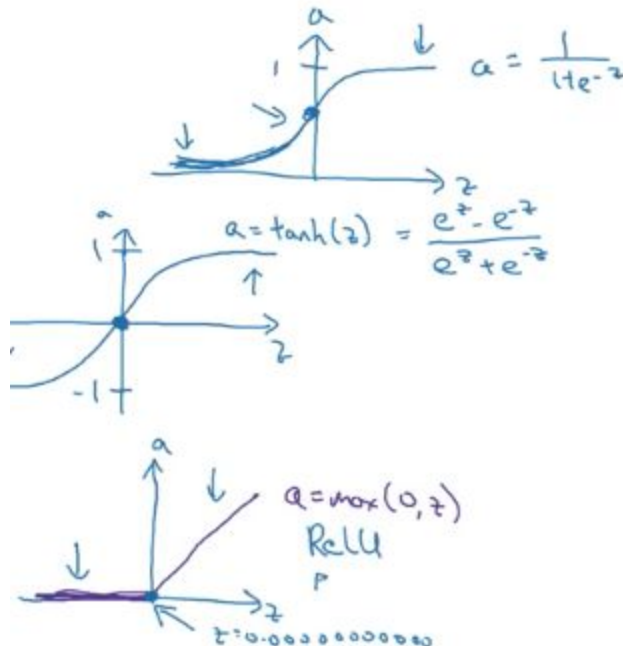
Regularization

Prevents overfitting and parameters 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 rates
 - Use with **AdaGrad**, **SGD**
- **Dropout**
 - Temporarily sets activation to 0
 - Works with all NN types
 - Avoid using on first layer, risks losing 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



Yellow tab For Forward and Back Propagation CALCS

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 (λ)

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 that 60% of the challenge.

From my reading of the papers NLP is 6 years behind Convolution.

- 2012 ImageNet triumph by a margin of 11%
- 2013 Mikolov word2vec
- 2018 Ruder NLP's ImageNet moment has arrived.

Github for Keras Beam Search
Keras {Functional, Sequential} API

Basic Seq to Seq Architecture

1. GloVe 500 dimensional embedding
2. 2 Stacked LSTM
3. Time Distributed (parameter sharing equivalent to convolution)
4. Softmax {Greedy vs Beam Search}

Classification / Sentiment

Naïve Bayes or 2-layer LSTM sequence classifier (n dimensional output)

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

Trigger Word Detection from spectrogram

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)

DialogFlow (Google)

<https://console.dialogflow.com/api-client/#/agent/f0df7906-77f0-4e36-a4d9-02e182ace8be/intents>

Encoder / decoder chatbox

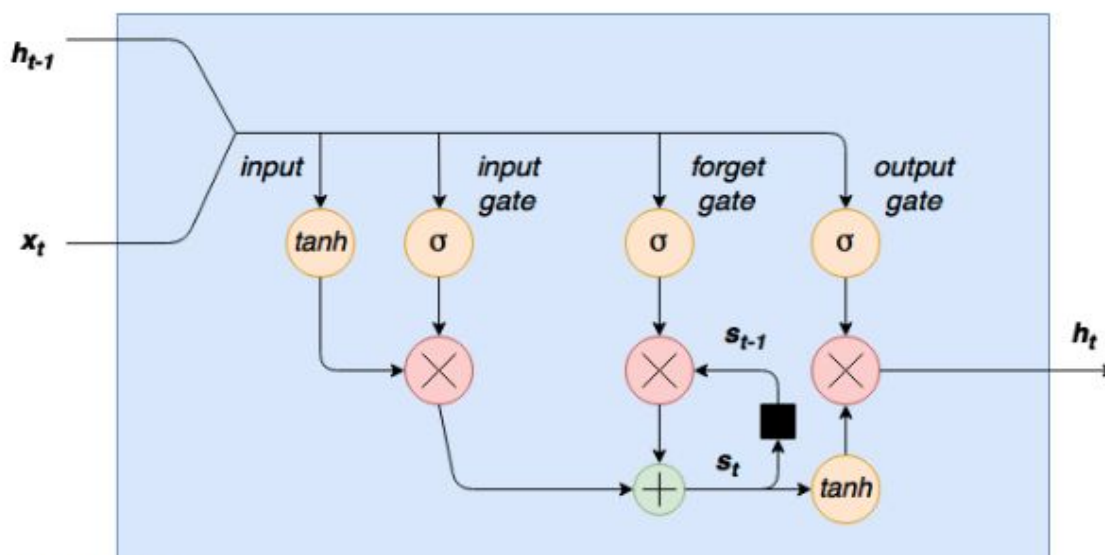
Attention Mechanism

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 outputting 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 through the process

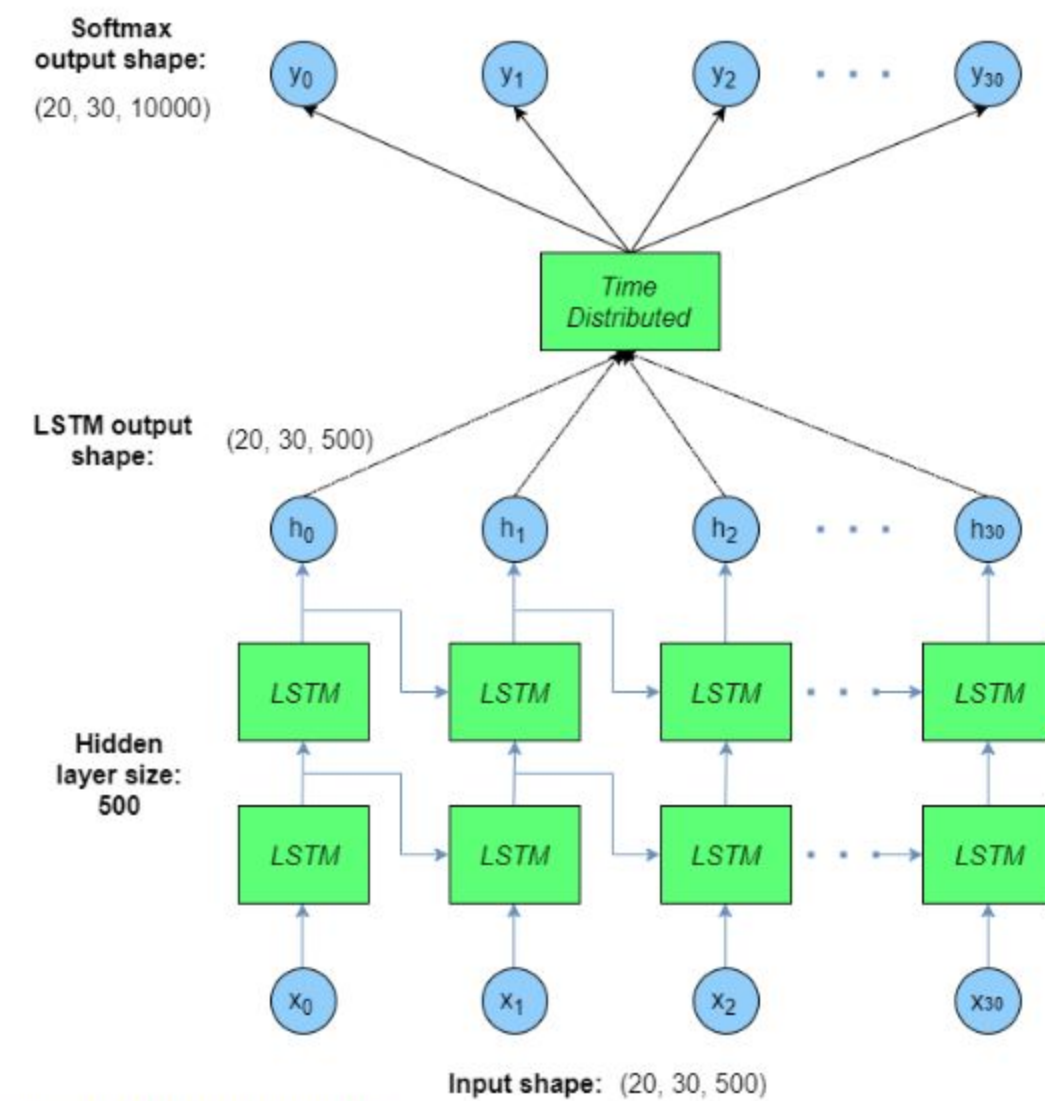
<http://adventuresinmachinelearning.com/keras-lstm-tutorial/>



LSTM cell diagram

Github repository for Keras Beam Search

Used Keras Functional API



Keras LSTM tutorial architecture

The input shape of the text data is ordered as follows : (batch size, number of time steps, hidden size). In other words, for each batch sample and each word in the number of time steps, there is a 500 length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. The input data is then fed into two "stacked" layers of LSTM cells (of 500 length hidden size) – in the diagram above, the LSTM network is shown as unrolled over all the time steps. The output from these unrolled cells is still (batch size, number of time steps, hidden size).

This output data is then passed to a Keras layer called TimeDistributed, which will be explained more fully below. Finally, the output layer has a *softmax* activation applied to it. This output is compared to the training *y* data for each batch, and the error and gradient back propagation is performed from there in Keras. The training *y* data in this case is the input *x* words advanced one time step – in other words, at each time step the model is trying to predict the very next word in the sequence. However, it does this at *every* time step – hence the output layer has the same number of time steps as the input layer. This will be made more clear later.

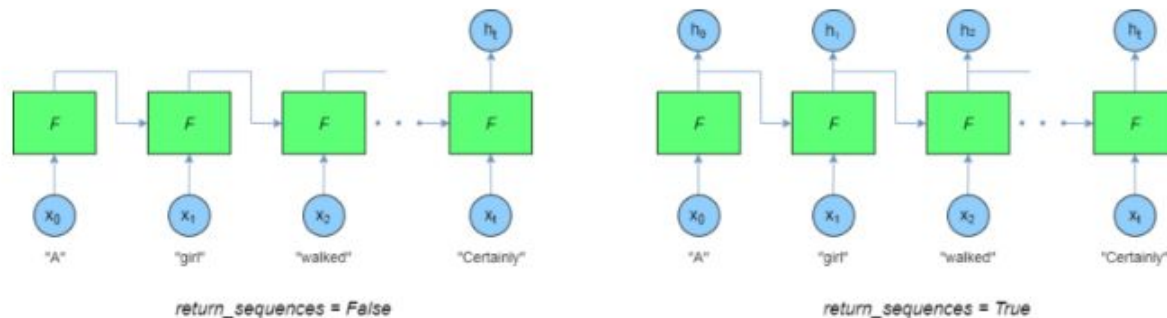
In this example, the Sequential way of building deep learning networks will be used. This way of building networks was introduced in my [Keras tutorial – build a convolutional neural network in 11 lines](#). The alternate way of building networks in Keras is the Functional API, which I used in my [Word2Vec Keras tutorial](#). Basically, the sequential methodology allows you to easily stack layers into your network without worrying too much about all the tensors (and their shapes) flowing through the model. However, you still have to keep your wits about you for some of the more complicated layers, as will be discussed below. In this example, it looks like the following:

```
1 model = Sequential()
2 model.add(Embedding(vocabulary, hidden_size, input_length=num_steps))
3 model.add(LSTM(hidden_size, return_sequences=True))
4 model.add(LSTM(hidden_size, return_sequences=True))
5 if use_dropout:
6     model.add(Dropout(0.5))
7 model.add(TimeDistributed(Dense(vocabulary)))
8 model.add(Activation('softmax'))
```

The first step involves creating a Keras model with the `Sequential()` constructor. The first layer in the network, as per the architecture diagram shown previously, is a word embedding layer. This will convert our words (referenced by integers in the data) into meaningful embedding vectors. This `Embedding()` layer takes the size of the vocabulary as its first argument, then the size of the resultant embedding vector that you want as the next argument. Finally, because this layer is the first layer in the network, we must specify the “length” of the input i.e. the number of steps/words in each sample.

It’s worthwhile keeping track of the Tensor shapes in the network – in this case, the input to the embedding layer is `(batch_size, num_steps)` and the output is `(batch_size, num_steps, hidden_size)`. Note that Keras, in the Sequential model, always maintains the batch size as the first dimension. It receives the batch size from the Keras fitting function (i.e. `fit_generator` in this case), and therefore it is rarely (never?) included in the definitions of the Sequential model layers.

The next layer is the first of our two LSTM layers. To specify an LSTM layer, first you have to provide the number of nodes in the hidden layers within the LSTM cell, e.g. the number of cells in the forget gate layer, the *tanh* squashing input layer and so on. The next argument that is specified in the code above is the *return_sequences=True* argument. What this does is ensure that the LSTM cell returns all of the outputs from the unrolled LSTM cell through time. If this argument is left out, the LSTM cell will simply provide the output of the LSTM cell from the last time step. The diagram below shows what I mean:



Keras LSTM return sequences argument comparison

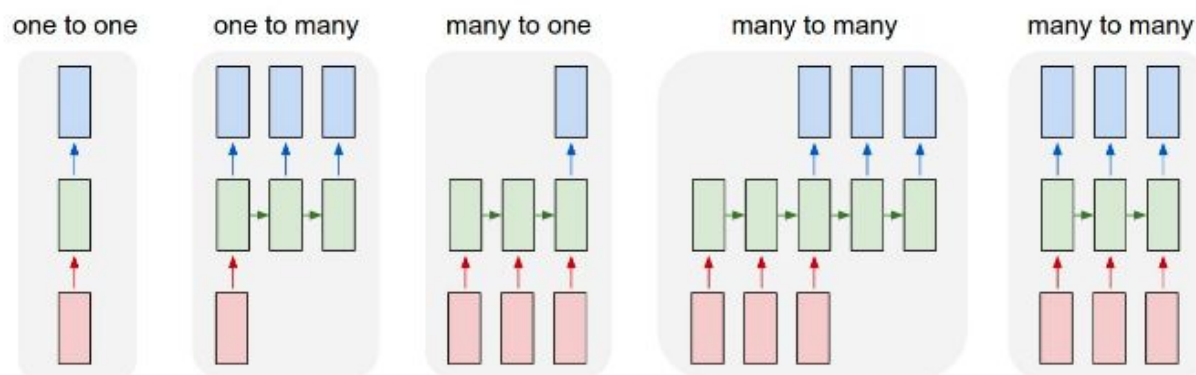
As can be observed in the diagram above, there is only one output when *return_sequences=False* – h_t . However, when *return_sequences=True* all of the unrolled outputs from the LSTM cells are returned $h_0 \dots h_t$. In this case, we want the latter arrangement. Why? Well, in this example we are trying to predict the very next word in the sequence. However, if we are trying to train the model, it is best to be able to compare the LSTM cell output at each time step with the very next word in the sequence – in this way we get *num_steps* sources to correct errors in the model (via **back-propagation**) rather than just one for each sample.

Therefore, for both stacked LSTM layers, we want to return all the sequences. The output shape of each LSTM layer is *(batch_size, num_steps, hidden_size)*.

The next layer in our Keras LSTM network is a dropout layer to prevent overfitting. After that, there is a special Keras layer for use in recurrent neural networks called TimeDistributed. This function adds an independent layer for each time step in the recurrent model. So, for instance, if we have 10 time steps in a model, a TimeDistributed layer operating on a Dense layer would produce 10 independent Dense layers, one for each time step. The activation for these dense layers is set to be softmax in the final layer of our Keras LSTM model.

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['categorical_accuracy'])
```

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.

Recurrent neural network (RNN)

1. RNN & LSTM (GRU)
2. Architecture
 - 2.1. Input Sequence, Activation, Cell State (solves vanishing gradient)
 - 2.2. Tanh, sigmoid (No relu)
 - 2.3. Dimensionality
 - 2.3.1. Many to Many
 - 2.3.2. Many to One
 - 2.3.3. One to Many
3. Descriptive Examples

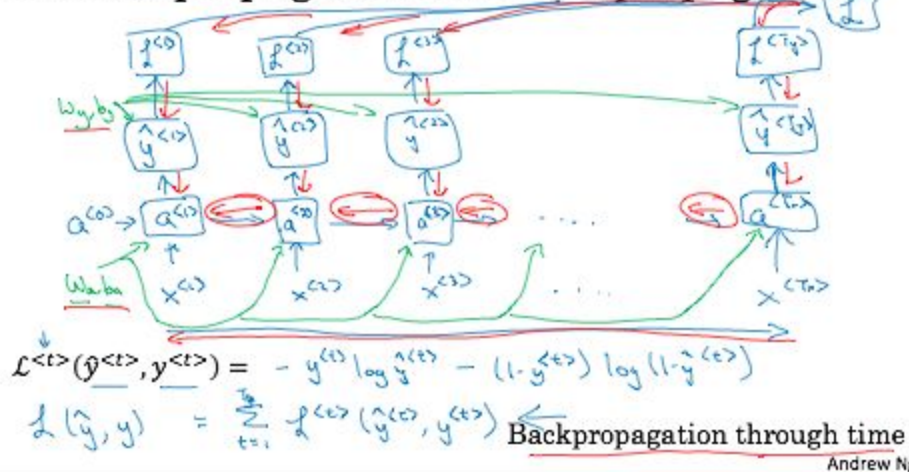
Examples of sequence data

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	→ AGCCCTGTGAGGAACTAG	→ AGCCCTGTGAGGAACTAG
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

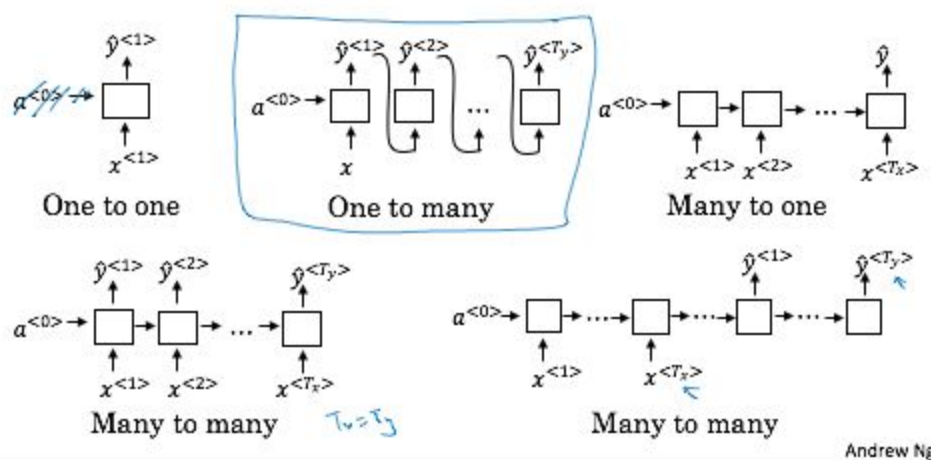
Commercial Vocabulary on order of 50,000 to 100,000

Backpropagation

Forward propagation and backpropagation



Summary of RNN types



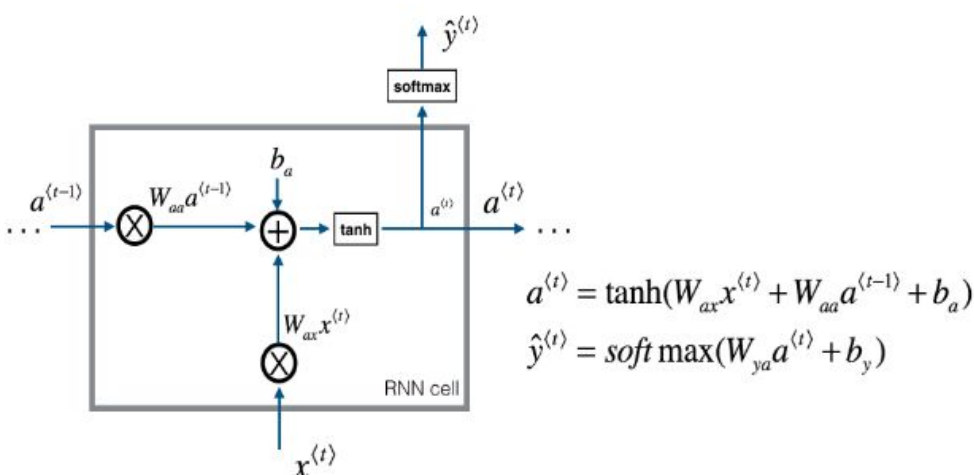
A language model determines the probability of a sentence using principle of multiplicative probabilities

$$\begin{aligned}
 P(y^{<1>}, y^{<2>}, y^{<3>}) &\leftarrow \text{way} \\
 &= \frac{P(y^{<1>}) P(y^{<2>} | y^{<1>})}{P(y^{<2>} | y^{<1>}, y^{<2>})}
 \end{aligned}$$

Sequence Generation you simple sample from the softmax

Vanishing Gradients LSTM. Exploding Gradients Gradient Clipping

RNN



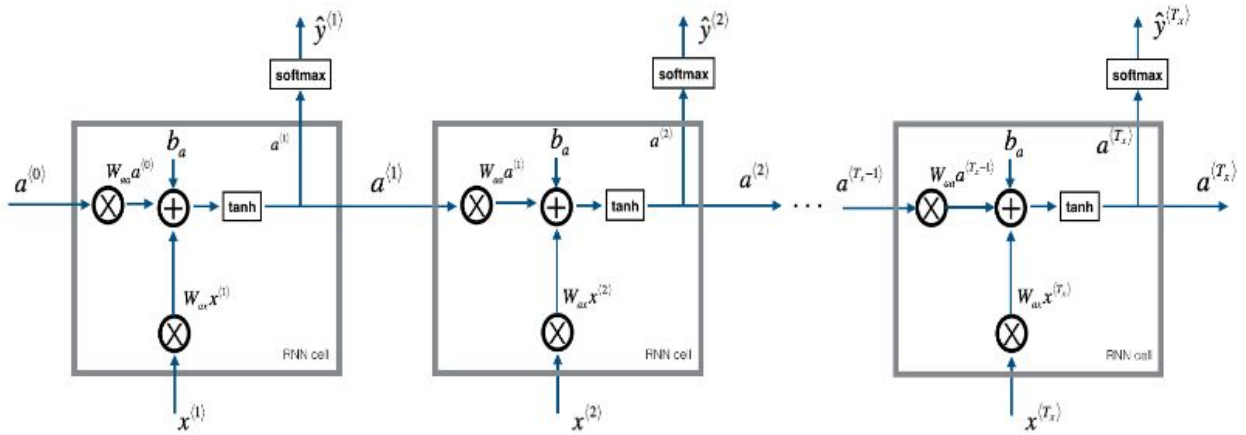


Figure 3: Basic RNN. The input sequence $x = (x^{(1)}, x^{(2)}, \dots, x^{(T_x)})$ is carried over T_x time steps. The network outputs $y = (y^{(1)}, y^{(2)}, \dots, y^{(T_x)})$.

Long Short-Term Memory (LSTM)

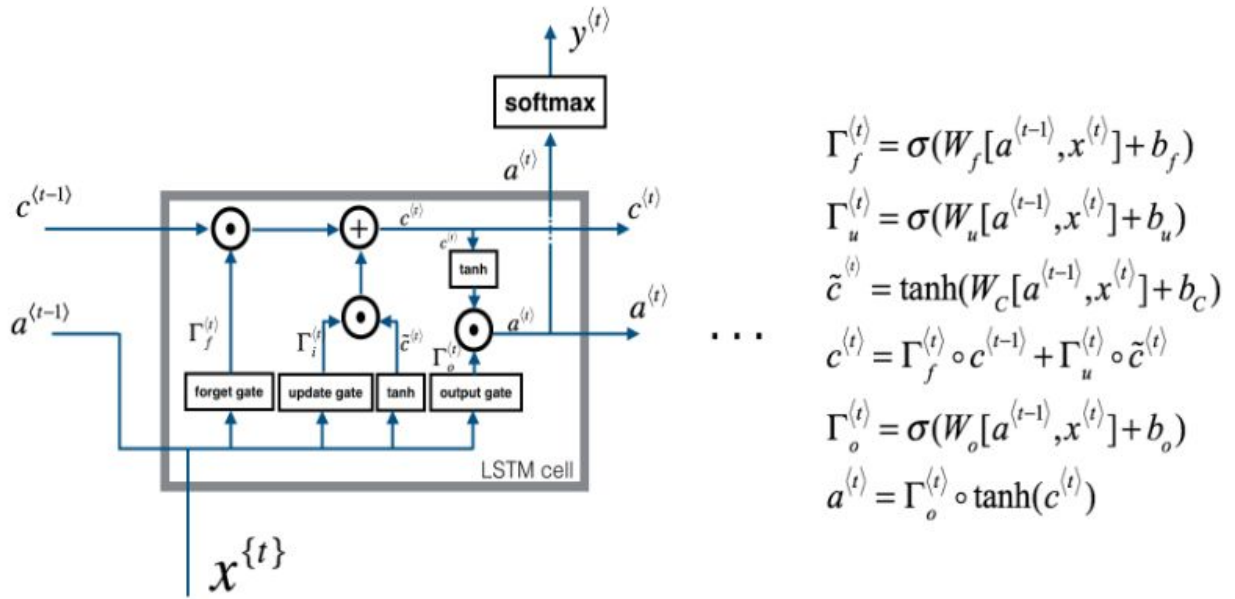


Figure 4: LSTM-cell. This tracks and updates a "cell state" or memory variable $c^{(t)}$ at every time-step, which can be different from $a^{(t)}$.

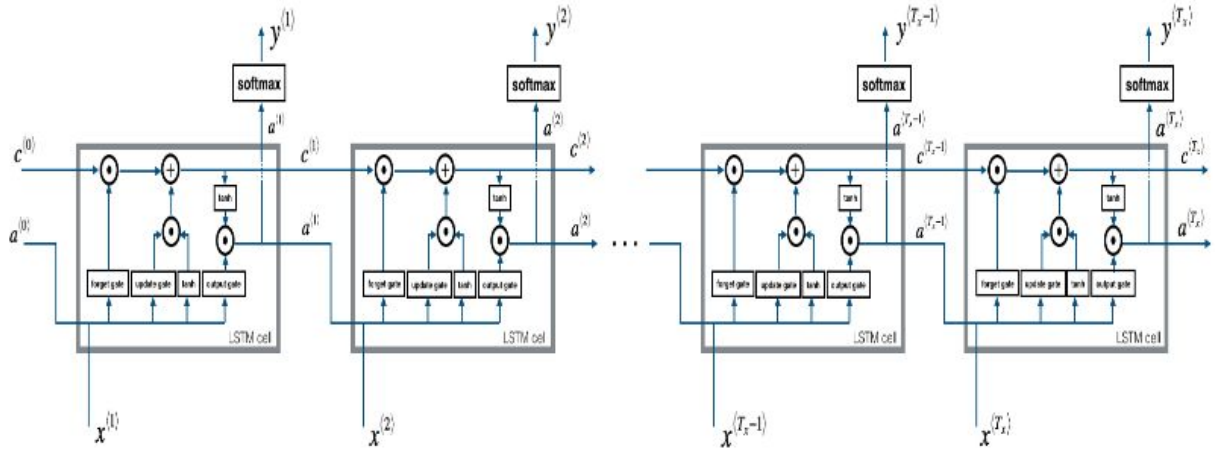


Figure 4: LSTM over multiple time-steps.

- Forget gate

For the sake of this illustration, let's assume we are reading words in a piece of text, and want to use an LSTM to keep track of grammatical structures, such as whether the subject is singular or plural. If the subject changes from a singular word to a plural word, we need to find a way to get rid of our previously stored memory value of the singular/plural state. In an LSTM, the forget gate lets us do this:

$$\Gamma_f^{(t)} = \sigma(W_f[a^{(t-1)}, x^{(t)}] + b_f) \quad ($$

Here, W_f are weights that govern the forget gate's behavior. We concatenate $[a^{(t-1)}, x^{(t)}]$ and multiply by W_f . The equation above results in a vector $\Gamma_f^{(t)}$ with values between 0 and 1. This forget gate vector will be multiplied element-wise by the previous cell state $c^{(t-1)}$. So if one of the values of $\Gamma_f^{(t)}$ is 0 (or close to 0) then it means that the LSTM should remove that piece of information (e.g. the singular subject) in the corresponding component of $c^{(t-1)}$. If one of the values is 1, then it will keep the information.

- Update gate

Once we forget that the subject being discussed is singular, we need to find a way to update it to reflect that the new subject is now plural. Here is the formula for the update gate:

$$\Gamma_u^{(t)} = \sigma(W_u[a^{(t-1)}, x^{(t)}] + b_u) \quad (2)$$

Similar to the forget gate, here $\Gamma_u^{(t)}$ is again a vector of values between 0 and 1. This will be multiplied element-wise with $\tilde{c}^{(t)}$, in order to compute $c^{(t)}$.

- Updating the cell

To update the new subject we need to create a new vector of numbers that we can add to our previous cell state. The equation we use is:

$$\tilde{c}^{(t)} = \tanh(W_c[a^{(t-1)}, x^{(t)}] + b_c) \quad (3)$$

Finally, the new cell state is:

$$c^{(t)} = \Gamma_f^{(t)} * c^{(t-1)} + \Gamma_u^{(t)} * \tilde{c}^{(t)} \quad (4)$$

- Output gate

To decide which outputs we will use, we will use the following two formulas:

$$\Gamma_o^{(t)} = \sigma(W_o[a^{(t-1)}, x^{(t)}] + b_o) \quad (5)$$

$$a^{(t)} = \Gamma_o^{(t)} * \tanh(c^{(t)}) \quad (6)$$

Where in equation 5 you decide what to output using a sigmoid function and in equation 6 you multiply that by the tanh of the previous state.

Bidirectional / Deep (3 layers is quite deep)

Wk1 Building a Recurrent Neural Network - Step by Step - V3 has Backpropagation

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

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

$$z^{(t+1)} = W_{ya}a^{(t+1)} + b_y \quad (2)$$

$$\hat{y}^{(t+1)} = \text{softmax}(z^{(t+1)}) \quad (3)$$

Note that $\hat{y}^{(t+1)}$ is a (softmax) probability vector (its entries are between 0 and 1 and sum to 1). $\hat{y}_i^{(t+1)}$ 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 $\hat{y}^{(t+1)}$. This means that if $\hat{y}_i^{(t+1)} = 0.16$, you will pick the index "i" with 16% probability. To implement it, you can use `np.random.choice`.

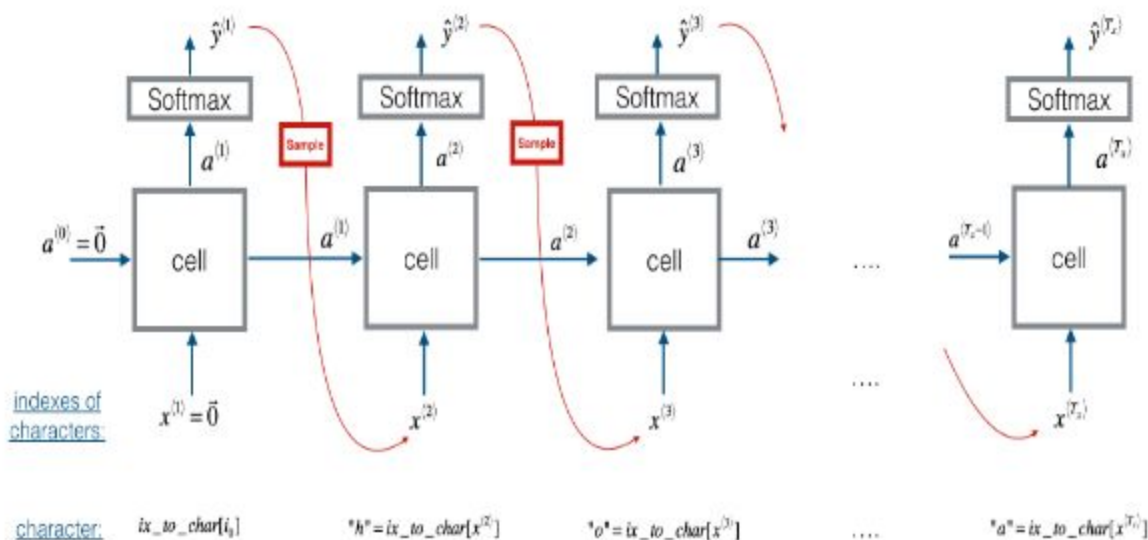
Here is an example of how to use `np.random.choice()`:

```
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:

$P(\text{index} = 0) = 0.1, P(\text{index} = 1) = 0.0, P(\text{index} = 2) = 0.7, P(\text{index} = 3) = 0.2$.

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



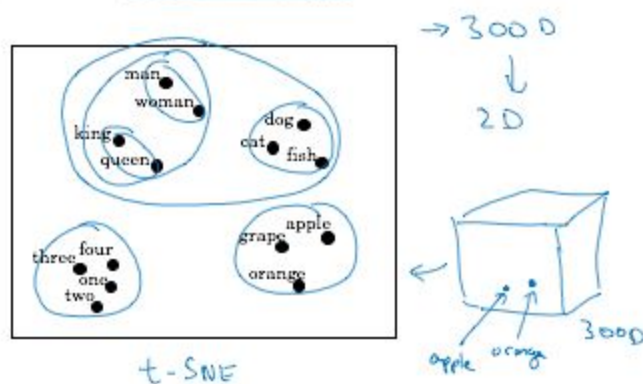
Generate Shakespeare Generate Shakespeare poems from corpus of The Sonnets

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

Word Embeddings 50 - 1,000 dimensions

Visualize word embeddings with t-SNE 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

Transfer learning and word embeddings

1. Learn word embeddings from large text corpus. (1-100B words)
(Or download pre-trained embedding online.)
2. Transfer embedding to new task with smaller training set.
(say, 100k words) $\rightarrow 10,000$ $\rightarrow 300$
3. Optional: Continue to finetune the word embeddings with new data.

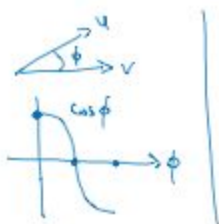
Andrew Ng

Cosine similarity

Cosine similarity

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

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



$$\|u - v\|^2$$

Man:Woman as Boy:Girl
 Ottawa:Canada as Nairobi:Kenya
 Big:Bigger as Tall:Taller
 Yen:Japan as Ruble:Russia

Andrew Ng

Emojifier-V2 2-layer LSTM sequence classifier

Use GloVe 50 dimensional Vector embedding without additional training. Output is 5 dimensional emoji.

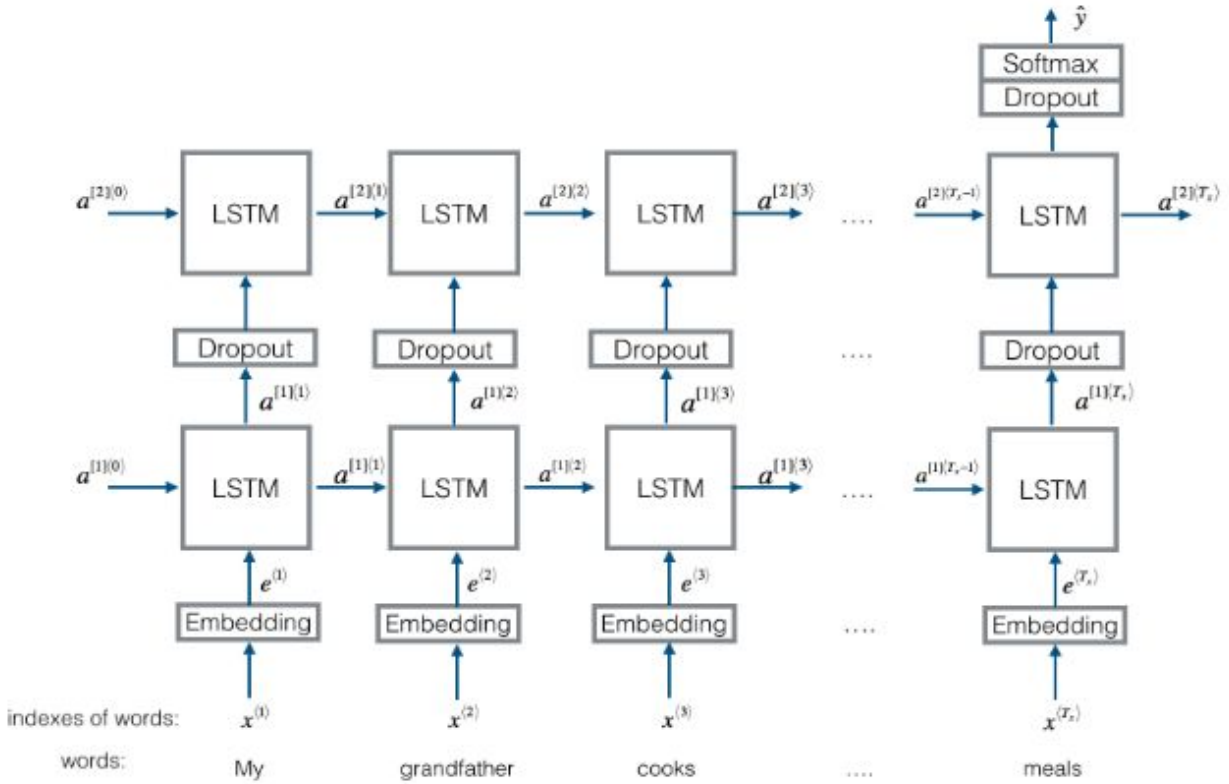


Figure 3: Emojifier-V2. A 2-layer LSTM sequence classifier.

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.

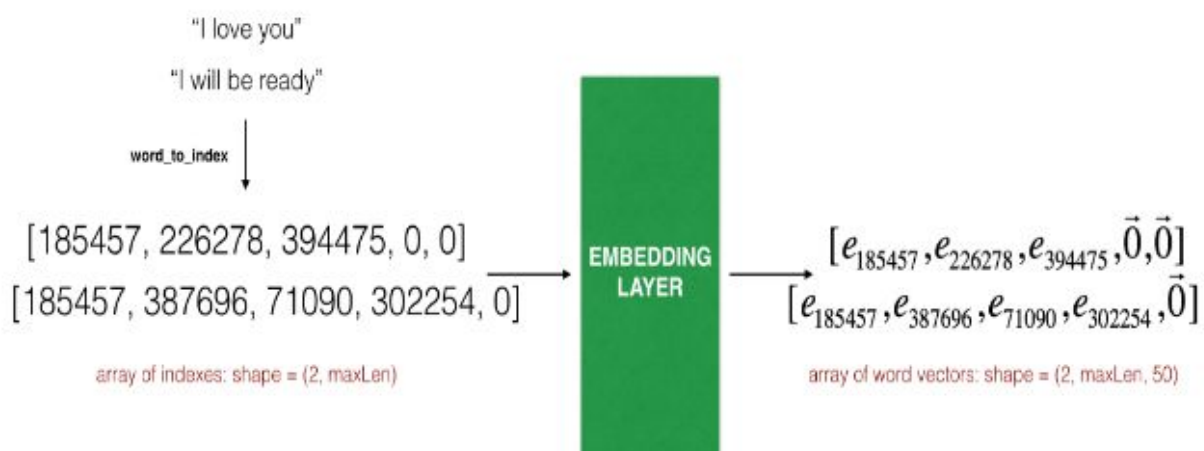
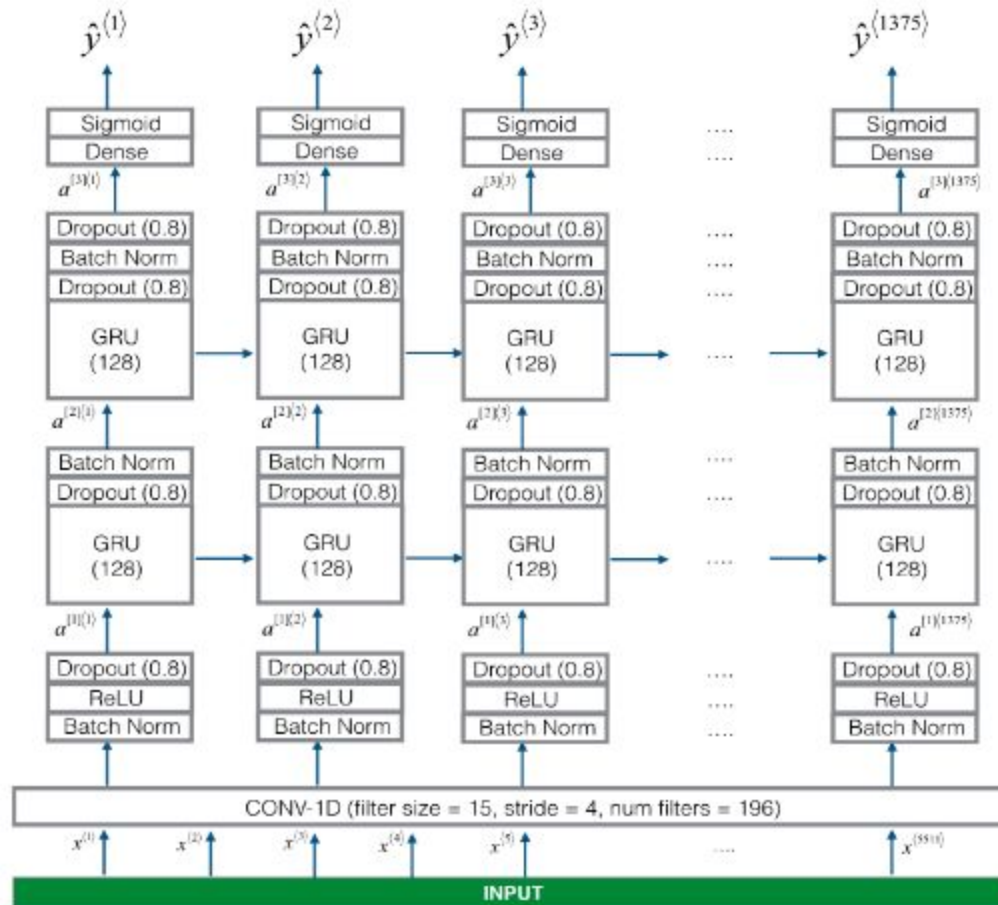


Figure 4: Embedding layer. This example shows the propagation of two examples through the embedding layer. Both have been zero-padded to a length of `max_len=5`. The final dimension of the representation is $(2, \text{max_len}, 50)$ because the word embeddings we are using are 50 dimensional.

Trigger Word Detection 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 spectrogram.



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 $y^{(t)}$. 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.

Motivating example



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.

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

Predict class \hat{y}	Actual class y	
	1	0
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?

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

Recall

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

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

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.

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

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:

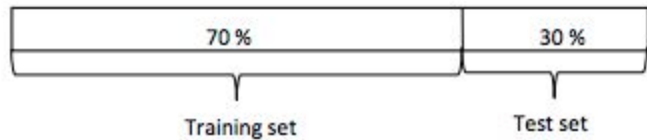
$$N_{metric} : \begin{cases} 1 & \text{Optimizing metric} \\ N_{metric} - 1 & \text{Satisficing metric} \end{cases}$$

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

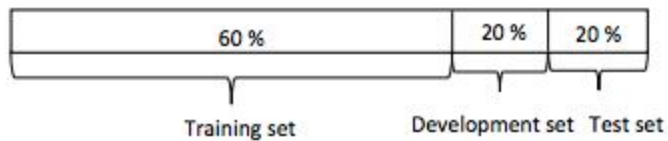
Size of the development and test sets

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 compromise 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:

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

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. One way to change this evaluation metric is to add the weight term $w^{(l)}$.

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

The function becomes:

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

Guideline

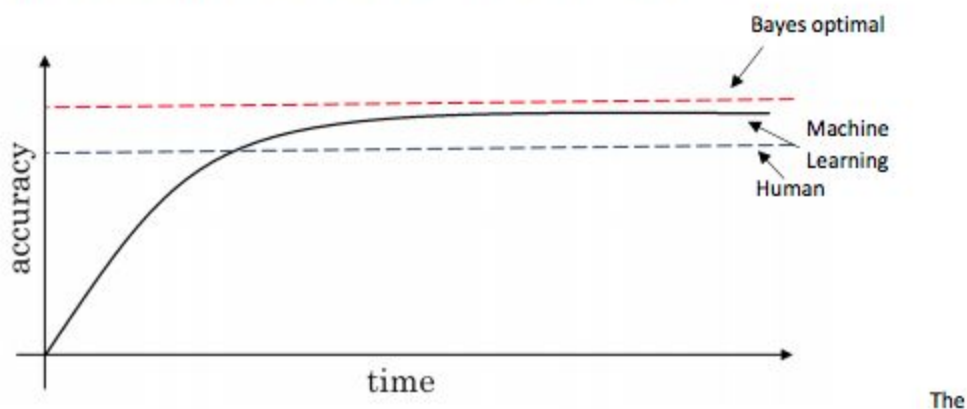
1. Define correctly an evaluation metric that helps better rank order classifiers
2. Optimize the evaluation metric

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.



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

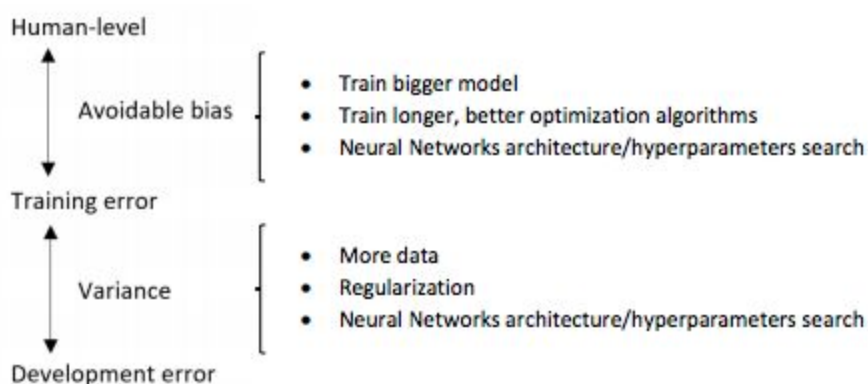
The two fundamental assumptions of supervised learning

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



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.

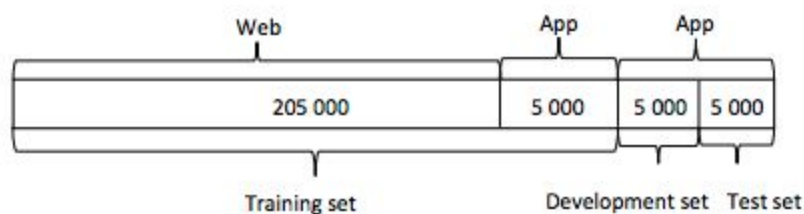
There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded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

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



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 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

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

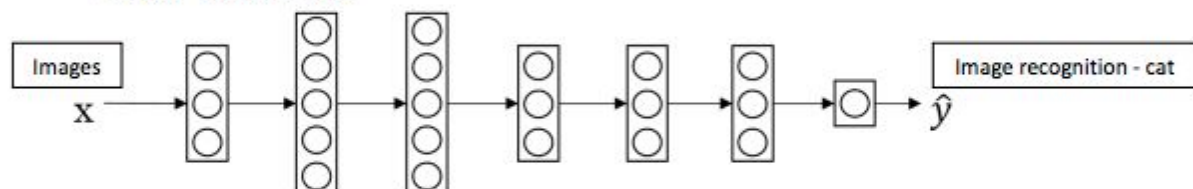
Example 1: Cat recognition - radiology diagnosis

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

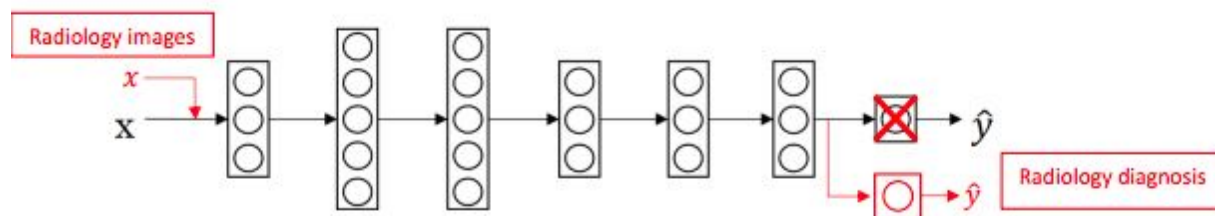
Output y - 1: cat, 0: no cat



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.

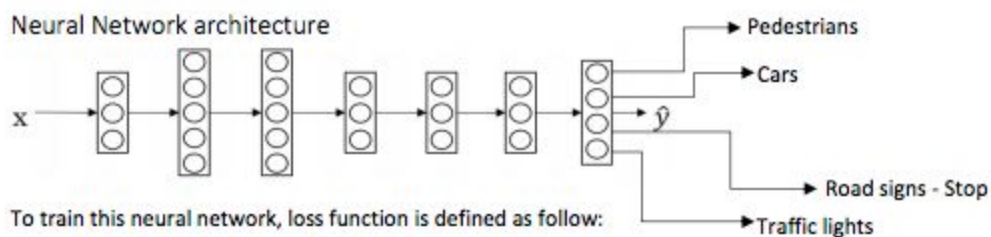
The input $x^{(i)}$ is the image with multiple labels
The output $y^{(i)}$ has 4 labels which are represents:

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

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



Neural Network architecture



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

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

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

Example:

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

What is end-to-end deep learning

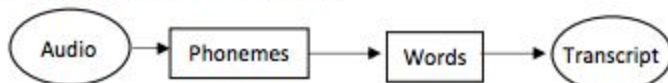
End-to-end deep learning is the simplification of a processing or learning systems into one neural network.

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.

Tools

Tensorflow Neural Network Playground: <https://goo.gl/3pmeKj>

Natural Language Processing (NLP) Convolution

Recurring Neural Nets (LSTM) Long short term memory

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

Do hyperparameter tuning WEEK 1

Steps for Training a Neural Net

7. Random Initialize weights
8. Forward propagation
9. Implement cost function
10. Backpropagation with partial derivatives

11. Gradient checking to validate
 12. Gradient descent to minimize cost function
- Training Set (60%) / Cross Validation (20%) / Test Set (20%)

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

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 = $2PR/(P+R)$

UNSUPERVISED

Clustering K-Means Algorithm

- Random Initialization
- Elbow method for choosing number of clusters

Stochastic Gradient Descent

- Randomly shuffle training examples
- Do gradient descent on 1 sample
- May not get to global minimum
- Wanders

Volatility Forecasting with LSTM (long short term memory)

My experience:

Clustering on Transaction Processing

My experience: I have extensive experience in analyzing and marketing to customers through supermarket and retail transactions linked to customer cards. I segment customers on transactions to enable tailored marketing programs. This work included store placement using census tract data.

Keras Functional API

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```

Word embeddings from co-occurrence matrices -- GloVe

Creating vector space models of word semantics (word embeddings)

Pennington, Socher, Manning (<http://www.aclweb.org/anthology/D14-1162>) 2014

Previous approaches

1. Rows (number of words) in Columns (paragraphs) use SVD (Singular Value Decomposition) and dot products
2. Pass Window over Corpus and predict:
Used in word2vec
 - 2.1. Surrounding words (Skip-gram)
 - 2.2. Word given surroundings (continuous bag-of-words)

Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.

CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.

Authors of Glove say these don't learn from the global corpus statistics

<https://towardsdatascience.com/emnlp-what-is-glove-part-ii-9e5ad227ee0>

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\sum_k X_{ik}},$$

X_{ij} = number of times word j
occurs in the context of word i .

VECTOR DIFFERENCE MODEL

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}.$$

Encoder / decoder Beam vs greedy search.

<https://guillaumegenthial.github.io/sequence-to-sequence.html>

Sequence tagging

<https://guillaumegenthial.github.io/sequence-to-sequence.html>

<https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html>

<https://hackernoon.com/beam-search-a-search-strategy-5d92fb7817f>

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-Ismael, Hsuan-Tien Lin

Data Analysis: A Bayesian Tutorial

By D.S. Sivia

Advances in Financial Machine Learning

By Marcos Lopez De Prado