Deep Learning

Nolan Reis

Abstract—Deep learning is a fast growing field in tech that is often described to have limitless potential. This paper describes its history, why the explosion in popularity, and how it works. An example of classifying images of handwritten digits (MNIST) will be explored using a fully connected network and a convolutional neural network. Next, a brief description of the tools necessary for the reader to implement his or her own network. Finally, a view of the state of the art being developed by companies such as Google, Facebook, and Baidu.

1 Trendy

Deep learning (DL) is one of the hottest terms in tech right now. Companies like Facebook, Google, YouTube, Tesla, Spotify, Yelp, and Microsoft are investing heavily into this tool. So what is it? The secret is that deep learning is just a re-branding of artificial neural networks (ANN), which have been around since the 1960s.

2 History

The earliest deep learning-like algorithms were invented by Ivakhnenko and Lapa in 1965. A lot of work and innovation happened in the 1980s (Fukushima's convolutional nueral networks) and 1990s (LeCunn's LeNet). [1] Many of these techniques are still used today. However, back then computers were slow and data sets were tiny. Researchers did not find many applications for neural networks (NN), so during the 2000's research dropped off. It was not until the last few years did NN make a resurgence.

The big shift was due to increased computational power and increased data. First was the introduction of the graphics processing unit (GPU). GPUs increased the computational processing speed by a factor of 1000 in the span of 10 years. [1] The second reason for NN's comeback was the exponential rise in data. Technology has allowed us to store more data and the internet has allowed us to share that data.

The combination of exponential technology and data has allowed deep learning to break record after record.

- *Speech Recognition*: In 2009, Microsoft and Toronto University improved speech recognition by 30% using DL. [2]
- *Computer Vision*: There is a yearly competition called ImageNet where teams compete to classify a library of 14 million images into 20,000 categories. In 2012, Alex Krizhevsky and Geoff Hinton submitted a deep learning algorithm, AlexNet, which achieved an error rate of 15% (40% better than state of the art). [3]
- *Drug Discovery*: In 2011, Geoff Hinton and a team from Toronto University won the "Merck Molecular Activity Challenge" for automatic drug discovery. They used deep learning to determine which molecule was most likely to be an effective drug agent. The amazing thing was that nobody on the team had a background in chemistry, biology, or life science and they did it in two weeks. [2]

3 Core Concepts

3.1 Machine Learning 101

Machine learning is a subfield of computer science where computers have the ability to learn from experience instead of being explicitly programmed. First we take some data, train a model off that data, then use that model to

make predictions on brand new data. Training is analogous to how humans learn. The model is exposed to new data, makes a prediction, and gets feedback about how accurate its prediction was. It uses that feedback to correct errors inside the model.[1] This process is repeated step by step multiple times through the entire data set.

The input data has n observations and m features. Features are attributes about the input data. For example, take a bank. There are n customers who have m features such as *does someone have a checking account? How much money is in that account?*

Machine learning models have the ability to predict either continuous values (*How much will someone spend per month?*) or classify k discrete values (*will someone open a savings account?*). These discrete values are referred to as *classes*. In the classification example, the output has $k = 2$ classes (*yes* or *no*). This paper will focus on classification.

3.2 Feature Engineering

Feature engineering is the process of using domain knowledge of data to extract useful features or patterns to make machine learning easier. For example, say you were training a model to predict if a photo was taken indoors or outside. You know that the sky is blue and so the percentage of blue pixels might be a good indicator (feature). By engineering that feature ahead of time, the model does not have to learn that the sky is in fact blue. This reduces the number of classes the model needs to consider (percentage of blue pixels vs. sky is blue or green or white). Examples of feature extractors for images are SIFT, HOG, RIFT.

While feature engineering is still a very important skill, it has its drawbacks. It requires expert knowledge of the problem, it can be very problem specific, and it takes a lot of hand tuning - which is time consuming.

1. In contrast, unsupervised learning eliminates the feedback portion and looks for unlabeled underlying structure.

3.3 Feature Learning

Feature learning is the process in which the algorithm autonomously finds distinguishing patterns, extracts them, and then feeds them to the classification layer. In other words, feature learning is feature engineering done automatically by algorithms. In deep learning, convolutional neural networks [CovNet] form a hierarchy of abstraction that grow in complexity (blobs→edges→eyes, noses, ears → face), see Fig 1. The final layer takes this generated feature and uses it for classification. [4]

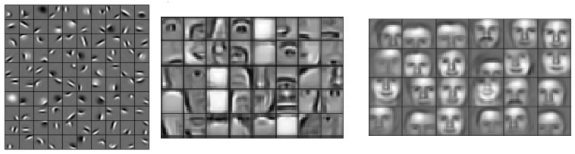


Fig. 1: Learned hierarchical feature learned by Deep Learning algorithm [4]

4 Logistic Classifier

The next two sections are going to explain the theory behind deep learning by starting with a logistic classifier and evolving it into a deep network. The purpose of a logistic classifier is predict a categorical class, given input data. *Is this an image of a 5? or a 4?*

Throughout all of deep learning the fundamental ingredients are a) Data b) Structure c) Loss and d) Optimizer

4.1 Data

As with all supervised machine learning algorithms, it is important to split the data into three sets: training, validation, testing. Normally, the data is split into 70% training, 20% validation, and 10% testing.

The training and validation sets are used during training. The training set is used to adjust the weights of the model. While the validation set does not update the weights, it is used to validate that the model is not overfitting. *Overfitting* is when a model is overly complex - it has superfluous freedom to align with the specific data.

Overfitting can be seen in the following analogy. A student, analogous to our network model, takes two exams of the same subject repeatedly. Over many trials the student will improve. However, if the student's accuracy increases on Test 1 but not on Test 2, then he may be memorizing the answers, not learning the material. The same is true for our model. It could be overfitting the training data and not learning the underlying relationship.

The test set is new, unseen data that is only used for testing the final model's predictive power. To follow the student analogy, the test set is the real world.

4.2 Structure

4.2.1 Neuron or Node

The basic building block of a network is the neuron or node (Fig 2). It takes some input data, applies a linear function to those inputs by calculating a weighted sum, and applies an activation function to that sum.

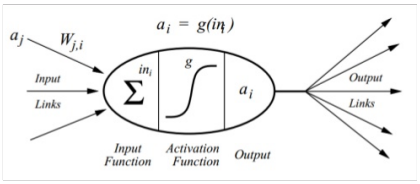


Fig. 2: Neuron or node: Basic unit of Deep Learning

The linear function is defined as

$$WX + b = Y \tag{1}$$

where X denotes an input vector, W denotes a matrix of weights, b denotes the biases, and Y denotes the *scores* or *logits*. The training happens by trying to find the weights and biases that are good at predicting the correct class. For example, take a model that is trying to learn handwritten digits with an input as an image of a handwritten "5." The linear function (Fig 3) takes that input and outputs logits. At first these outputs do not mean much. The task is to determine the probability the image belongs to each class (digit). The way to turn logits into probabilities is to apply a softmax as our activation function, see Fig 4.

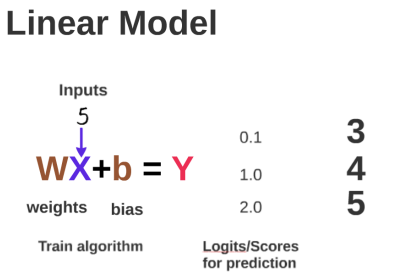


Fig. 3: Linear Function

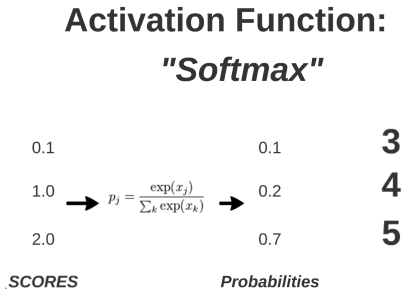


Fig. 4: To turn logits into probabilities, the activation function was chosen to be softmax

The softmax function outputs the probabilities the image belongs to each class (the most likely is close to 1 and the less likely are close to 0). The technique of One-hot Encoding is used to turn each label into a class-membership vector. This vector has the value 1 for the correct class, and 0 for the rest of entries. In the above example, the five is the correct label, so the one-hot encoded vector is [0,0,1].

There are now two vectors, one from the classifier (the probabilities) and one that represents the correct label (encoded vector).

4.3 Loss

For the feedback in the model to work, there must be a metric of success. The way to measure the distance between potential two vectors is called Cross Entropy, Eq 2. The goal is to have

a low distance for a correct class but a high distance for an incorrect one.

$$D(S(Y), L)) = \sum_k L_k log(S(Y_k))) \quad (2)$$

The Training Loss, Eq 3, is defined as the average cross entropy over the entire training set (i). A good model has a low training loss.

$$\mathfrak{L} = \frac{1}{N} \sum_i D(S(WX_i + b)), L_i) \quad (3)$$

The loss is a function of the weights and the biases, so we are going to minimize that function using an optimizer. [5]

4.4 Optimizer

One of the most popular optimizing techniques in machine learning is called Stochastic Gradient Descent (SGD). It takes small steps along the loss surface following the gradient until it finds a minimum. Recall the gradient is the multivariate slope of a function. The size of the step is called the *learning rate*. The bigger the learning rate the faster it learns, but it may not reach the absolute minimal loss. In practice, SGD is performed over multiple passes of the data set called epochs.

SGD is popular in machine learning because it scales well with data and model size. However, it comes with additional hyper-parameters. These are different from ordinary parameters that the model optimizes. Examples of hyper-parameters that the user must tune are:

- Learning Rate initialization
- Learning Rate decay
- Weight initialization
- Number of Epochs

4.5 Summary

To summarize, we have created a linear model that outputs probabilities [structure]. We evaluate how the model is doing by calculating the cross entropy [loss] and use SGD [optimizer] to minimize that loss. It is still a shallow model, but these are the fundamental tools for going deeper.

5 Deep Learning

5.1 MultiLayer Perceptron [MLP]

To turn the logistic classifier into a network, a second neuron is linked between the current neuron and the input (Fig 5). This is called a two-layer Neural Network (the input layer is not counted).

Layers are the highest level building block of a network. The new layer is called the Hidden Layer because its output values are not visible to the network output. The hidden layer gives the model the opportunity to represent the data in a simpler way.

The depth of the network is defined by the number of hidden layers.

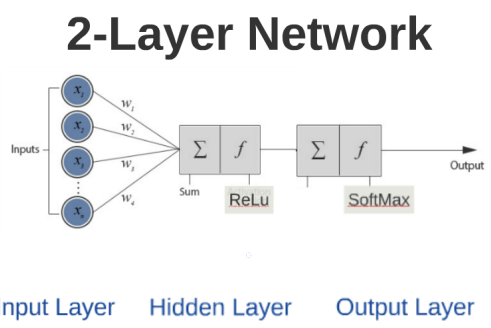


Fig. 5: Basic two layer Neural Network

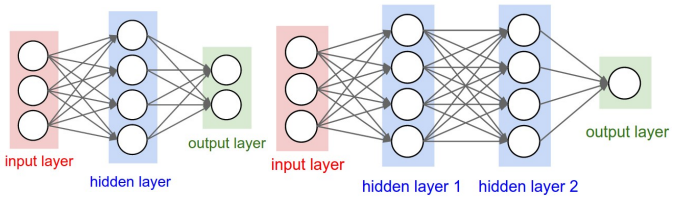
In addition to layers, the number of nodes per layer can increase as well. The number of nodes on a layer represents the degree of freedom of that layer.

When the output of every node on one layer is connected to the input of every node on the next layer, the network is called *Fully Connected* or *Dense*.

The size of a network is defined by the number of layers and the number of nodes or parameters. Fig 6a has 2 layers, 4+2=6 nodes (do not count input) or [3x4]+[4x2] = 20 weights and 4+2=6 biases for a total of 26 parameters. Fig 10b has 3 layers, 9 nodes, and 42 learnable parameters.

Modern convolutional neural networks have 100 million parameters and 20 layers (hence deep learning).

Evolving the structure from a single node into a network has allowed the model more



(a) 2-Layer (b) 3-Layer

Fig. 6: Two Fully connected Neural Networks [6]

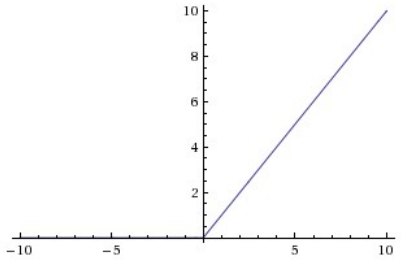


Fig. 7: Rectified Linear Unit is the most popular nonlinear function.

opportunities to represent the data in a simpler way (layers) and more degrees of freedom (nodes). However, the model is still linear. Because of superposition, stacking a 100 purely linear transformations can be simplified to a single layer. The solution is to introduce non-linear functions.

5.2 Non-Linearities

To preserve the network's structure (and the benefits gained with this structure), each hidden layer is given a non-linear activation function. By adding non-linearity, the entire model is now non-linear and cannot be simplified down to a single transformation. This creates a hierarchy of abstraction that grows in complexity with every layer. [4] [7]

This is the foundation for building deep models.

There are multiple types of non-linear activation functions: softmax, sigmoidal/logistic, tanh, and the rectified linear unit [ReLU]. A ReLU is a very simple, very powerful non-linearity. Its output is linear for x greater than zero and zero everywhere else (Fig 7). Since its introduction in 2012, ReLu has become the most popular non-linearity because it does not face gradient vanishing problems as with sigmoid and tanh function. [7]

5.3 Summary

By constructing this MLP network we have given the model a better structure

- Hidden Layers - number of moves to figure out a simpler way to represent the data
- Number of nodes per hidden layer - the degrees of freedom for that move
- Non-linearities allow increasing feature complexity with each layer

We then told the network to learn the best parameters in order to correctly classify the input. This is the core to Deep Learning. [5] [7] [8]

6 Convolutional Neural Networks

Deep networks are powerful but can quickly increase in complexity. Back to the example of classifying handwritten digits. If the input image is 32x32 pixels with 3 colors and the network has 2 fully connected layers with 2 outputs (similar to Fig 6a) then there would be a total of 9.4 millions learnable parameters. That is a lot of parameters for a small image and a simple structure. To help out the model, the user can use his or her domain knowledge (the fact that it is an image).

Take an image of a cat. It does not matter where in the image the cat is, it is still an image of a cat. This is called translational invariance. Identifying invariant structure is a key aspect in machine learning because it is a direct path to efficient learning. In a fully connected network, the model learns weights for cats in the right corner and different weights for cats in the left corner. Instead, the user would like the model to learn features by sharing weights across the entire image. This is called *convolutions*.

6.1 Convolutions

Fig 8 is an example of an input image (X) with a cat in it. The image has a width, height, and a depth (represented by the RGB colors). Take a small patch of the image and run a tiny neural network on it with k outputs. Sliding that patch across the entire image creates a new image with a new width, height, and a depth of k. If the patch was the size of the original image, it would be no different than a fully connected layer. However, by sweeping a smaller patch across the image there are fewer weights and the weights are shared across space.

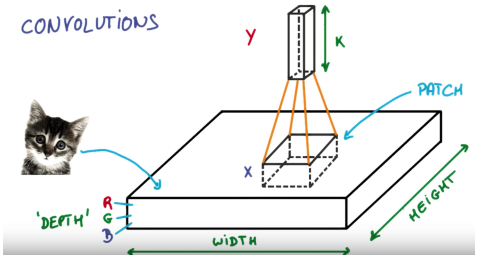


Fig. 8: Sketch of how a Convolution passes over an image [5]

6.2 Network

Convolutions are stacked on top of each other to form a convolutional pyramid, Fig 9. The layers progressively squeeze the spatial dimensions, while increasing the network depth. The depth can be thought of as the semantic representation. At the end, a fully connected classifier is attached. Through training, these convolutional layers form the hierarchy of abstraction seen in Fig 1. [5] [7]

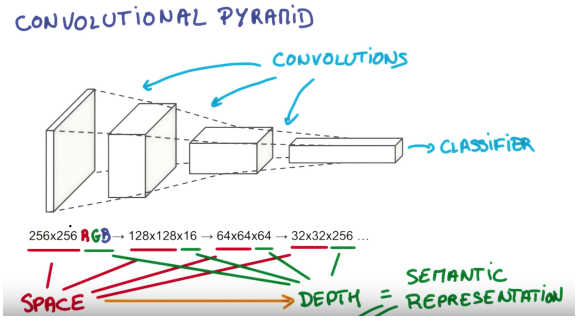


Fig. 9: Structure of a ConvNet [5]

7 Example: MNIST

The challenge of classifying handwritten digits is a classic machine learning problem. The dataset used is called MNIST and it was one of the first real world problems solved by neural networks.

7.1 Data

MNIST contains 60,000 training images and 10,000 test images of handwritten digits from 500 different writers. Each image is a grey scale 28x28 pixel image. 10% of the training data was reserved for validation.

7.2 Structure

Two structures were evaluated: MultiLayer Perceptron [MLP] and a ConvNet.

7.2.1 MLP

A 2-Layer Perceptron [MLP] was built with fully connected layers. The input was an image flattened to a vector of length 784 (28x28). This input was fully connected to a hidden layer with 512 nodes with a ReLU activation function. The hidden layer was fully connected to the output layer of length 10 (0-9) with a softmax.[2]

7.2.2 ConvNet

The input image was kept in its original form 1x28x28. It was inputted into two convolutional layers which squeezed it to a shape of 32x14x14. That image was fed into the same MLP classifier as described above.

7.3 Loss and Optimizer

Both models defined the loss as cross entropy and used RMSprop as their optimizer. [RMSprop is a version of SGD with an adaptive learning rate].

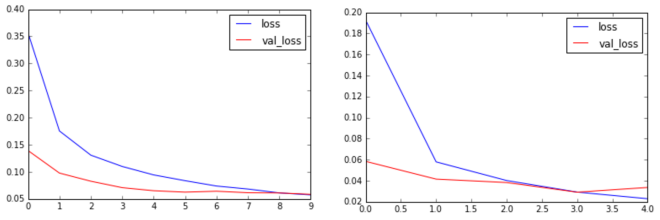
2. Dropout was applied to combat overfitting

7.4 Results

The results are shown in the Table 1 below. Both algorithms performed excellent. Out of the 10,000 test points, the MLP missed 188. The ConvNet misclassified only 93; it was twice as accurate in half the number of epochs. Fig 10 shows the loss curves for each, demonstrating that neither model overfit the training data. The time per epoch was listed because this example was done on the author's personal computer (2014 13in Macbook Pro running OS X 10.11.4, 2.6GHz 8GB, Intel Iris 1536MB). At the time of purchase, the author had no intention of demanding more than basic performance from his machine. Had this experiment been run on a GPU the training time would have been an order of magnitude faster.

TABLE 1: Results from Classifying MNIST

			Accuracy		
	Epochs	time per Epoch [s]	Training Set	Validation Set	Test Set
MLP	10	10	0.9822	0.9828	0.9812
CNN	5	360	0.9928	0.9915	0.9907



(a) MLP (b) ConvNet

Fig. 10: Loss as a function of epoch. The fact that the training and validation losses converge is evidence that the model is not overfitting

8 Tools

The good news is that the field of DL is exploding and a majority of it is open-source. The bad news is that this field is in its infancy so there are a lot of options and it is difficult to configure your system.

8.1 Programming

Deep learning is programmed in mostly Python or C++. Since it is just math, one can program all of the operations from scratch. However, many common functions have been built into open-source toolkits. The community has not consolidated yet, so there are over 50 different toolkits, each with its advantages and disadvantages. The most popular include

- TensorFlow
- Keras
- Theano
- Caffe
- Torch
- CNTK

Google open-sourced its toolkit called TensorFlow and it has gained a lot of traction in the six months since its release (November, 2015). It is a very powerful toolkit that they are writing all of their algorithms on.

The best place to start is a toolkit called Keras. It is built to run on top of TensorFlow or Theano. The purpose of this toolkit is to enable fast, easy prototyping. While it is not as powerful as other options it allows beginners to get their hands dirty quickly. [3]

In addition to toolkits, existing pre-trained networks are usually open-sourced. Winning networks, such as AlexNet (the first ConvNet submitted to ImageNet), open-source their learned parameters and structure.

8.2 Transfer Learning

One of the continuing limitations of DL is data. While the amount of data is increasing exponentially, getting good, clean data is hard to come by. Large corporations like Facebook or Google can pay for manual labeling of data, but everyone else uses shared data sets like MNIST over and over again. Few DL models are trained from scratch. It is logical to assume that is would stall innovation; however, this is not true. It has been shown that CovNets learned from large data sets can learn generic features and be repurposed for other smaller databases. [9]

3. It is this author's opinion that creating an environment inside anaconda was the most successful way to get started.

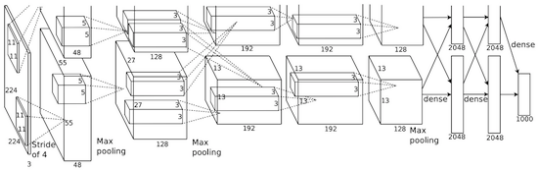


Fig. 11: The structure of AlexNet [10]

8.2.1 Fine Tuning

Take a ConvNet pre-trained on ImageNet and cut off the last fully connected layer (that classifies the 1000 classes defined by ImageNet). Retrain the CovNet by fine-tuning the existing weights for this new dataset. Essentially, the original weights are used as an initialization for the new task. The motivation behind this strategy is that the lowest levels of the ConvNet contain generic features (edge detector or blob) that can be useful for many task; however, later layers become progressively more specific to the nuances of the classes for the original dataset. For example, ImageNet has a number of dog breeds, so AlexNet likely has a number of later filters that can distinguish between the breeds. [6]

8.3 Technology

As previously stated, the exponential rise in computational power has enabled DL to grow at an incredible rate. In the 2000s, researchers recognized that the GPU inside gaming computers was perfect for quickly multiplying very large matrices. They originally rode the rise of gaming computers, but lately, computer companies, like Nvidia, have taken notice of DL and begun building chips specifically designed for DL. Deciding which GPU specifications is beyond the scope of this report but there are many resources for those who are interested. [11]

It is this author's opinion that if the reader is planning to do deep learning, then he or she should spend time researching a sufficient computer. While it may be possible to train models using CPUs, it is not practical. Even basic GPUs are 10x faster, which means faster iterating.

Another option is Amazon Web Services [AWS]. A user can rent time on Amazon's GPUs to run an algorithm for a couple hours. This is great for testing models out without the investment in hardware.

Currently, most of DL is done on clusters of GPUs by big companies and requires the cloud. Movidius is trying to change that. Last month they announced their Fathom Neural Compute Stick - a modular deep learning accelerator in the form of a standard USB stick. This chip allows DL to be embedded in new places like robots, drones, cameras, VR. The goal to able to add a visual cortex to any device. This will take the learning out of the cloud and allow the devices to be more natively intelligent. [12] [13]

9 State of the Art

Here is a snapshot of the state of the art at the time this report was written. (note: new advances are announced almost every week)

9.1 Google

9.1.1 Google Photos [Sept 2014]

Google Photos is a downloadable app that stores and learns the user's photos in the cloud. It is built off Google's first place finish in the 2014 ImageNet: googLeNet. [14] The program learns your friends through facial recognition, as well as learns about photo context - this allows for easy searching.

9.1.2 Deep Dream [July 2015]

The Google engineers wanted a way to visualize what the network was visualizing on the middle layers of googLeNet, so they invented a technique called Inceptionism. The network is fed an arbitrary image and asked to enhance whatever it detected on a selected layer. Each layer deals with a different level of abstraction, so lower layers tend to produce strokes (Fig 12) while higher levels identify more sophisticated features (even objects), see Fig 13 and Fig 14.

> This creates a feedback loop: if a cloud looks a little bit like a bird, the network will make it look more like a

bird. This in turn will make the network recognize the bird even more strongly on the next pass and so forth, until a highly detailed bird appears, seemingly out of nowhere. [15]

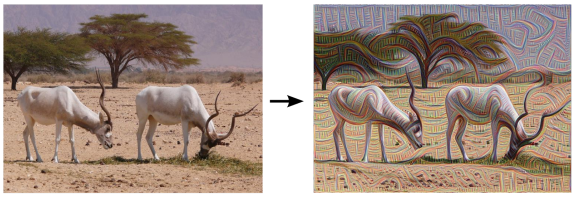


Fig. 12: Visualization of a lower layer produces strokes [15]

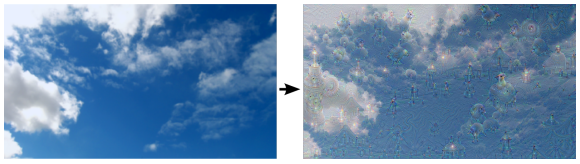


Fig. 13: Visualization of a higher level produces more complex objects [15]

Fig. 14: Zoomed in view of the sky visualization in Fig 13 shows the advanced objects [15]

Due to their trippy, psychedelic nature the engineers joke that this might be what a computer brain's daydreams might actually look like. Google has open-sourced the code, DeepDream, for anyone to create their own images.

Building off DeepDream, a paper [16] was published using a ConvNet to factor images into style and content. This allows the creation of new images that combine the style of one image with the content of another, Fig 15.

9.1.3 PlaNet [February 2016]

Google created a network, PlaNet, that has the "superhuman" ability to determine the location of almost any image. They trained a deep

Fig. 15: Using a NN to cross a photo with a painting style: for example Neil deGrasse Tyson in the style of Kadinskys Jane Rouge Bleu. [17]

learning network to work out the location of a photo using only the image's pixels. Their approach was to divide the world into a grid of 26,000 squares. The size of those squares varied based on the number of images taken in that location. Big Cities had more fine-grained grid structures; while oceans and the poles were ignored. The data set consisted of 126M photos with Exif geolocations mined from all over the web (training: 91M; validation: 34M). To test the model's localization accuracy, they fed it 2.3M geotagged Flickr photos from across the world, see Fig 16.

PlaNet is able to localize 3.6% of the images at street-level accuracy and 10.1% at city-level accuracy. 28.4% of the photos are correctly localized at country level and 48.0% at continent level. [18]

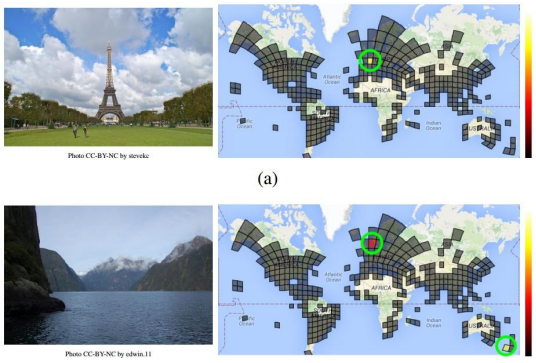


Fig. 16: Google's PlaNet geolocating a picture in one of its 26,000 zones [19]

9.1.4 AlphaGo [March 2016]

In March 2016, Google's AlphaGo won 4-1 against the Lee Sedol, the legendary Go champion for that last decade.

The game of Go is 2,500 year old Chinese game. Players take turns placing black or white stones on the board, trying to capture the opponent's stones or empty territory. The game is incredibly complex with more possible positions than there are atoms in the universe. Go is a googol times more complex than chess. The game is played primarily through intuition and feel.

In 1997 IBM's DeepBlue beat the world champion chess player using a brute force method, Search Tree, to calculate all possible positions. This is not possible with Go. Instead, AlphaGo used a combination of Monte Carlo Tree Search with deep neural networks to play out the rest of the game in its imagination. Not only was the match one sided, it was 10 years before experts predicted a computer would win at Go. [20] [21] [22]

9.1.5 SyntaxNet [May 2016]

Google just released SyntaxNet, an open-source neural network framework that provides a foundation for Natural Language Understanding (NLU). Not only did they provide all the code (written in TensorFlow) needed to train models on individual data, they included Parsey McParseface, an English parser that has been pre-trained to analyze English text.

SyntaxNet is built off syntactic parsing. Given a sentence as input, it tags each word as a part-of-speech with its syntactic function. Then it determines the syntactic relationship between the words, which is related to the underlying meaning of the sentence. Parsey McParseface can handle complex sentences like Fig 17, this allows users to ask questions like *whom did Alice see?*, *when did Alice see Bob?*

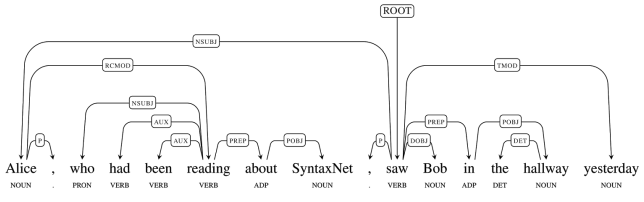


Fig. 17: Parsey McParseface understanding a sample sentence [23]

Unfortunately, natural language is full of ambiguities. In a moderate length sentence (20-30 words), there can be tens of thousands of syntactic relationships. For example the following sentence (Fig 18) can be read two ways. First, the correct understanding is Alice is driving the car. The second (absurd, but possible) interpretation is where the street is located in the car. The preposition *in* can modify *drove* or *street*, causing ambiguity. From our vast experience, humans do a great job navigating these ambiguous cases. SyntaxNet uses deep learning.

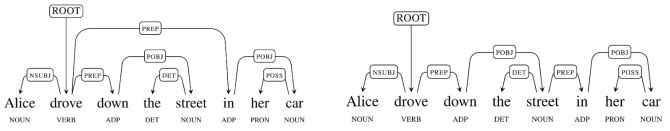


Fig. 18: An example of a prepositional phrase attachment ambiguity. [23]

Parsey McParseface understands sentences from news articles with an accuracy of 94%. On sentences scraped from the internet, Parsey understands 90%.

The goal of Natural Language Understanding is to make our interactions with computers more natural. Instead of memorized phrases, soon the user will be able to just talk with a computer. [23] [24]

9.2 Facebook [2014]

Facebook AI Research (FAIR) is a research team at Facebook advancing the field of machine learning. One of their big projects was called DeepFace. DeepFace performs facial verification (it recognizes that two images show the same face), not facial recognition (putting a name to a face).

To derive a facial representation, they created a nine-layer deep neural network. This network involved more than 120 million parameters using several locally connected layers but without the weight sharing as in standard convolutional layers.

Using a dataset of 4 million photos of 4,000 individuals, DeepFace achieved a 97.25% accuracy at predicting if two images showed the

same person. This is remarkable result as it is almost at human-level performance (97.53%). [25] [26]

9.3 FaceYou [October, 2015]

FaceYou is an entertainment app that allows users to merge their own face with another face. It is able to capture facial expressions and speech in real-time.

Baidu researchers developed FaceYou as a demonstration to show the sophistication of deep learning on a smartphone. Traditionally, this level of real-time face tracking was only possible on large systems used in film and animation studios. [27] Another application of this technology could be fashion. Instead of merging two faces, a shirt or dress could be projected onto a potential buyer.

Fig. 19: The author with a skeleton merged onto his face

10 Future of Deep Learning

By now it is clear that deep learning has incredible potential. However, what makes DL special from other machine learning techniques? Why is it causing renowned scientists to make bold claims?

> Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is
> - Geoff Hinton [28]

Andrew Ng, Cofounder of Google Brain and Chief Scientist at Baidu Research, describes it with the following graph, Fig 20. While all other machine learning algorithms get better with more data, at some point their performance plateaus. In deep learning, the exponential growth of both data and computation power allow models to evolve in structure and complexity, thus increasing their performance.

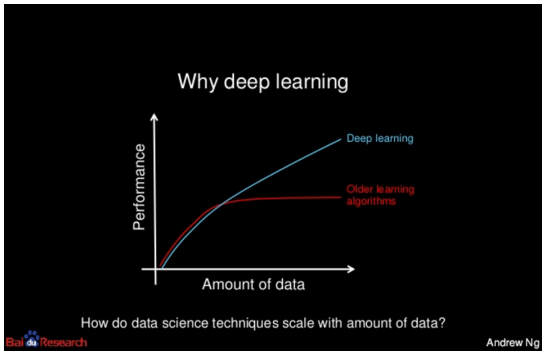


Fig. 20: Andrew Ng's slide about the immense potential of DL over traditional machine learning algorithms [29]

11 Summary

This paper described the history of deep learning and how it is just a rebranding of artificial neural networks. However, the reason for DL's recent explosion in popularity is due to the exponential growth of both computing power and data.

Next, we walked through the foundation of what makes a deep network: *data*, *structure*, *loss*, and an *optimizer*. We built a multilayer perceptron by adding nodes, layers, and non-linearities to a simple logistic classifier. By using domain knowledge (the input was an image), we showed how convolutional neural networks form a hierarchy of abstraction that grow in complexity.

We talked about how tools like TensorFlow, transfer learning, and GPUs can greatly increase productivity when training DIY models. We examined the state-of-art problems big companies are solving using deep learning.

Finally, we explained why experts are claiming the limitless potential of Deep Learning.

REFERENCES

[1] T. Dettmers. (2015) Deep learning in a nutshell: History and training. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training/

[2] J. Markoff. (2012) Scientists see promise in deep learning programs. [Online]. Available: http://www.nytimes.com/2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html

[3] (2012). [Online]. Available: http://image-net.org/challenges/LSVRC/2012/results.html

[4] T. Dettmers. (2015) Deep learning in a nutshell: Core concepts. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/feature-engineering

[5] V. Vanhoucke. (2015) Udacity: Deep learning. [Online]. Available: https://www.udacity.com/course/deep-learning--ud730

[6] F.-F. Li and A. Karpathy. (2016) Cs231n: Convolutional neural networks for visual recognition. [Online]. Available: http://cs231n.stanford.edu/

[7] C. Mellina, "Deep learning workshop," 2016.

[8] D. Smilkov and S. Carter. (2016) Tensorflow playground. [Online]. Available: http://playground.tensorflow.org

[9] T.-Y. Lin, Y. Cui, S. Belongie, and J. Hays, "Learning deep representations for ground-to-aerial geolocalization," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 5007–5015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[11] T. Dettmers. (2014) Which gpu(s) to get for deep learning: My experience and advice for using gpus in deep learning. [Online]. Available: http://timdettmers.com/2014/08/14/which-gpu-for-deep-learning/

[12] A. Souppouris. (2016) Artificial intelligence now fits inside a usb stick. [Online]. Available: http://www.engadget.com/2016/04/28/movidius-fathom-neural-compute-stick/

[13] (2016) Embedded neural network compute framework:fathom. [Online]. Available: http://www.movidius.com/solutions/machine-vision-algorithms/machine-learning

[14] C. Szegedy. (2014) Building a deeper understanding of images. [Online]. Available: http://googleresearch.blogspot.com/2014/09/building-deeper-understanding-of-images.html

[15] A. Mordvintsev, C. Olah, and M. Tyka. (2015) Inceptionism: Going deeper into neural networks. [Online]. Available: http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html

[16] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[17] M. Tyka. (2016) Exploring the intersection of art and machine intelligence. [Online]. Available: http://googleresearch.blogspot.ca/2016/02/exploring-intersection-of-art-and.html

[18] T. Weyand, I. Kostrikov, and J. Philbin, "Planet - photo geolocation with convolutional neural networks," *CoRR*, vol. abs/1602.05314, 2016. [Online]. Available: http://arxiv.org/abs/1602.05314

[19] (2016) Google unveils neural network with superhuman ability to determine the location of almost any image. [Online]. Available: https://www.technologyreview.com/s/600889/google-unveils-neural-network-with-superhuman-ability-to-determine-the-location-of-almost/

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 01 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[21] (2016). [Online]. Available: https://deepmind.com/alpha-go

[22] D. Silver and D. Hassabis. (2016) Alphago: Mastering the ancient game of go with machine learning. [Online]. Available: http://googleresearch.blogspot.com/2016/01/alphago-mastering-ancient-game-of-go.html

[23] S. Petrov. (2016) Announcing syntaxnet: The worlds most accurate parser goes open source. [Online]. Available: http://googleresearch.blogspot.com/2016/05/announcing-syntaxnet-worlds-most.html

[24] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, "Globally normalized transition-based neural networks," *CoRR*, vol. abs/1603.06042, 2016. [Online]. Available: http://arxiv.org/abs/1603.06042

[25] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1701–1708.

[26] T. Simonite. (2014) Facebook creates software that matches faces almost as well as you do. [Online]. Available: https://www.technologyreview.com/s/525586/facebook-creates-software-that-matches-faces-almost-as-well-as-you-do/

[27] (2015) Happy halloween! baidu research introduces faceyou. [Online]. Available: http://research.baidu.com/happy-halloween-baidu-research-introduces-faceyou/

[28] P. Pallath. (2015) Making the most of predictive analytics: Exploring deep learning. [Online]. Available: http://blog-sap.com/analytics/2015/06/11/making-the-most-of-predictive-analytics-exploring-deep-learning/L

[29] A. Ng, "2015 extract data conference: What data scientists should know about deep learning."

APPENDIX A
MLP CODE FOR LEARNING MNIST

```
#———MNIST – MLP
#MNIST: grayscale hand-written digits 28x28 pixels.
# there are 10 classes in the dataset corresponding to the digits 0-9.

import matplotlib.pyplot as plt
import numpy as np

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import SGD, Adam, RMSprop
from keras.utils import np_utils


#load the data and split it into train and test sets

(X_train, y_train), (X_test, y_test) = mnist.load_data()
print X_train.shape

#show a random example
plt.imshow(X_train[np.random.randint(len(X_train))], cmap='Greys')


## flatten the 28x28 images to a 784 dimensional vector.
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')


# encode our labels as one-hot vectors.
y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)


#function to plot the Training Loss
def plot_loss(hist):
    loss = hist.history['loss']
    val_loss= hist.history['val_loss']
    #plt.plot(range(len(loss)), loss)
    plt.plot(range(len(loss)), loss, 'b', val_loss, 'r')
    plt.legend(['loss','val_loss'])
```

MLP

```
## ---Construst the MLP structure
model = Sequential()
model.add(Dense(512, input_dim=784)) #784 = 28x28 input; and 512 nodes
model.add(Activation('relu')) #define the activation func as 'RELU'
model.add(Dropout(.5)) #add dropout with a probability of 50%
model.add(Dense(10)) #add a fully connected layer with 10 outputs (0-9 digits)
model.add(Activation('softmax'))

model.summary()          #print out structure

#define LOSS and OPTIMIZER
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, nb_epoch=10,
                    batch_size=128, verbose=1,
                    validation_split=0.1)
plot_loss(history)

# Final test evaluation
score = model.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

APPENDIX B
COVNET PYTHON CODE FOR LEARNING MNIST

```
#
#  Classifying MNIST with CNNs
#

import matplotlib.pyplot as plt
import numpy as np

from keras.datasets import mnist
from keras.models import Sequential
from keras.optimizers import SGD, RMSprop
from keras.utils import np_utils

# FORMAT INPUT DATA
# Keep the data in its original shape.
#note: when we reshape the data below, we add a dimension of 1.
#       this is the number of **channels** in the image,
#       which is just 1 because these are grayscale images.
#       If they were color, this would be 3 for RGB.


(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print X_train.shape

y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)


#function to plot the Training Loss
def plot_loss(hist):
    loss = hist.history['loss']
    val_loss= hist.history['val_loss']
    #plt.plot(range(len(loss)), loss)
    plt.plot(range(len(loss)), loss, 'b', val_loss, 'r')
    plt.legend(['loss','val_loss'])



from keras.layers.core import Dense, Dropout, Activation
```

```
from keras.layers import Convolution2D, MaxPooling2D, AveragePooling2D,Flatten

numb_labels = 10
#
# ### design structure of CNN###
model = Sequential()
model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=(1, 28, 28),
#2x2 pooling cuts in image in half
model.add(MaxPooling2D(pool_size=(2, 2), strides=None, border_mode='same'))
model.add(Convolution2D(32, 3, 3, border_mode='same'))
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(.4))
model.add(Dense(numb_labels))
model.add(Activation('softmax'))

model.summary()

#define LOSS and OPTIMIZER
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, nb_epoch=5,
                    batch_size=128, verbose=1,
                    validation_split=0.1)
plot_loss(history)


score = model.evaluate(X_test, y_test, verbose=1)
print('Test score:', score[0])
print('Test accuracy:', score[1])


# ## Saving a trained model
with open('mnist_cnn.json', 'w') as f:
    f.write(model.to_json())

model.save_weights('mnist_cnn_weights.h5')
```