Applied Deep Learning

Applied Deep Learning

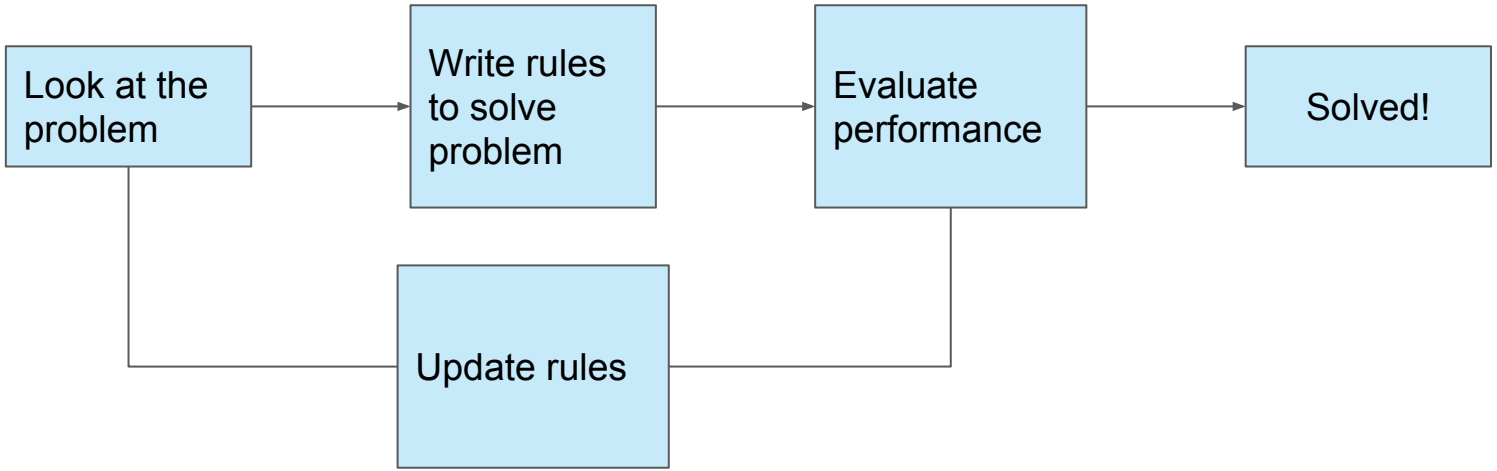
- Day 1 - Intro to ML and DL
  - Representation Learning
  - Training
  - NN enhancements
  - TensorFlow Intro
- Day 2 - Convolutional Neural Networks
  - Convolution operation
  - Network Structure
  - Representation Learning in ConvNets
  - Transfer Learning
  - Object Detection
  - TensorFlow with ConvNets
- Day 3 - Recurrent Neural Networks
  - RNN Structure
  - Representation Learning with Sequences
  - Advanced RNN structures (GRU, LSTM)
  - TensorFlow with RNNs
- Day 4 - GAN and Project
  - Generative Adversarial Networks
  - Create an application with TensorFlow

Day 1: Introduction to Deep Learning

Machine Learning

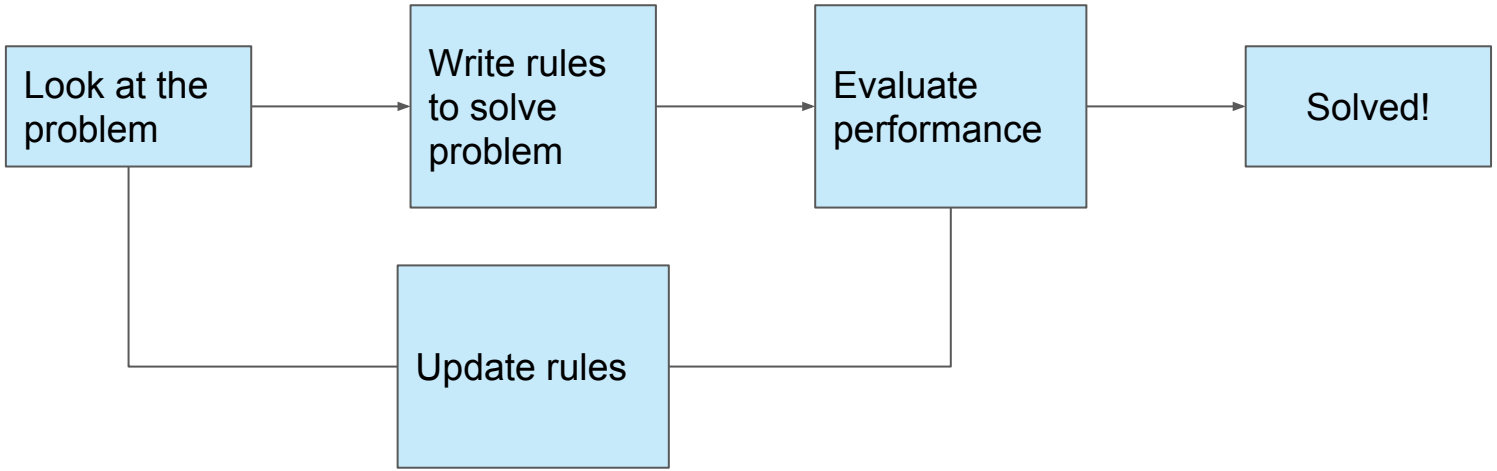
What is machine learning?

- Machine learning is about programming computers to learn from data
  - Giving computers the ability to learn without being explicitly programmed
- We want the machine to learn its own rules for solving some problem
- Typical problem solving loop looks like this

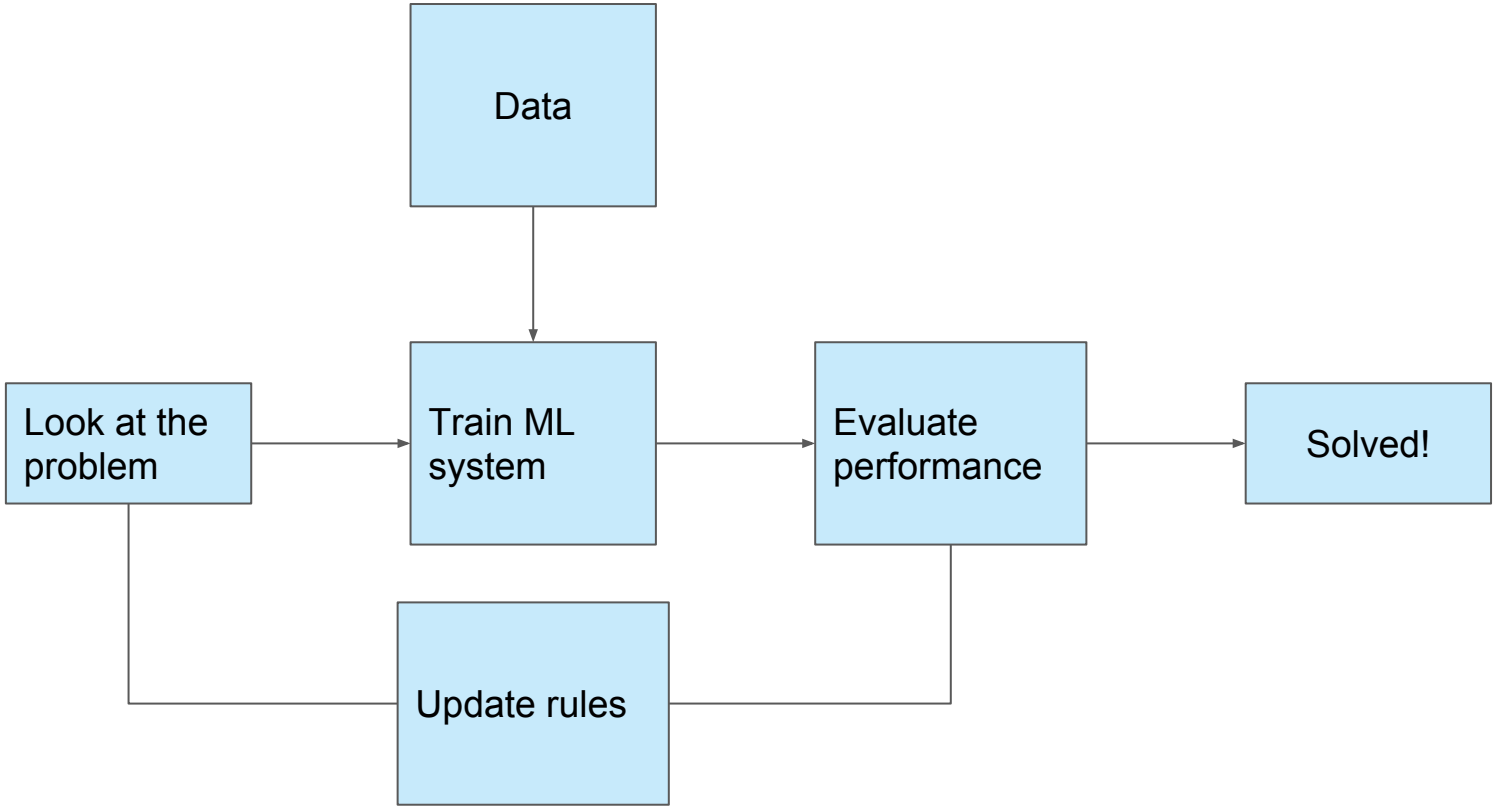


What is machine learning?

- We want to automate this process
  - Need to add data to the loop and design the system to make use of that data



What is machine learning?

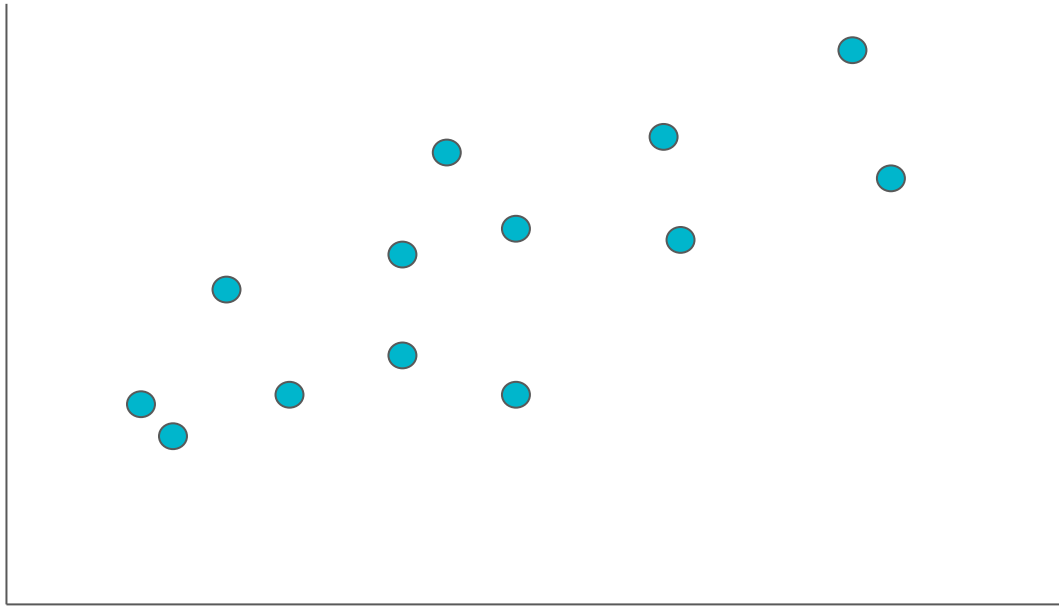


What is machine learning?

- To get a machine learning system to learn on its own we have to set up our problem in some clever ways
  - Typically use mathematics to produce some output given the data
    - Multiply data by some values
  - Evaluate that output with another mathematical function (evaluation metric)
    - Score model on how much generated values deviate from the desired output
  - Update the machine learning system to perform better on the evaluation metric
    - Change the values that we multiply to the data so the generated output gets closer to the desired output on the next pass

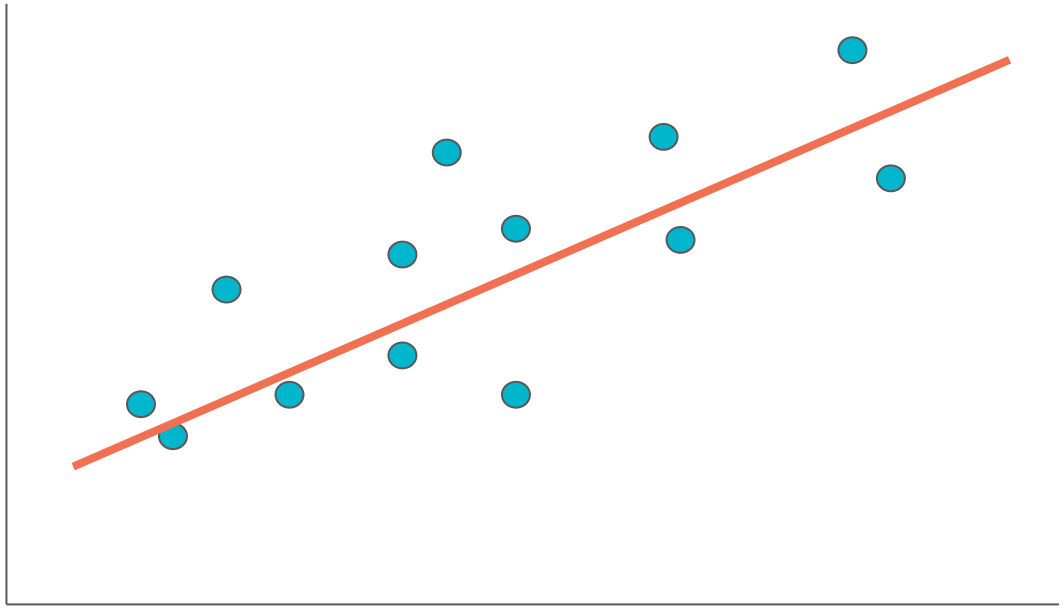
What is machine learning?

- Calculating the line of best fit can be thought of as the simplest case of machine learning



What is machine learning?

- Calculating the line of best fit can be thought of as the simplest case of machine learning



What is machine learning?

- This type of machine learning is typically referred to as supervised learning
  - We have some dataset and corresponding true output values for each data point
  - We want our machine learning system to get as close to the true output for each example as possible
- Other types of machine learning exist, and will be introduced as needed
  - Unsupervised learning
  - Reinforcement learning
  - Semi-supervised learning

Deep Learning

What is deep learning?

- Deep learning is a subset of machine learning
  - Essentially doing machine learning with deep neural networks
- A more creative application of neural networks
  - Arrangement of ‘neurons’ or computational units is more complex and unique to each problem type
    - Neural networks are now commonly designed as sequences of computational blocks where the arrangement has some significance to the problem.
    - Image classification network will look much different than a language translation network
  - Effective training algorithms and specific hardware

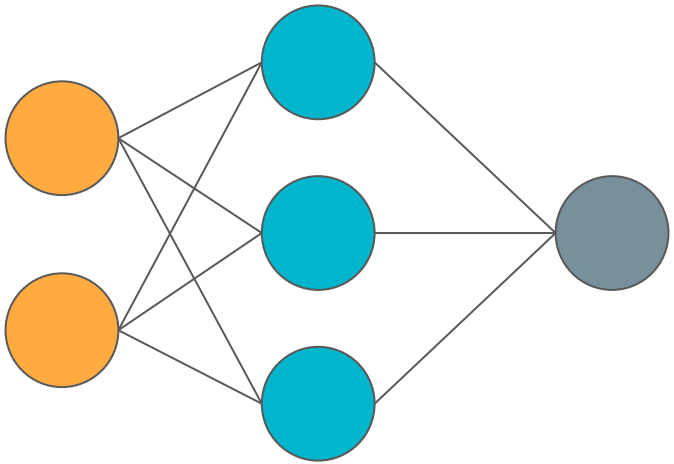
What is deep learning?

- Conventional machine learning algorithms have some problems with feature engineering:
  - Time consuming
  - Error prone
  - Difficult for complex data types (images, text)
  - Requires domain expertise
- With Deep Learning:
  - Model learns its own way to represent the features
  - Able to work on data in a more raw form

http://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

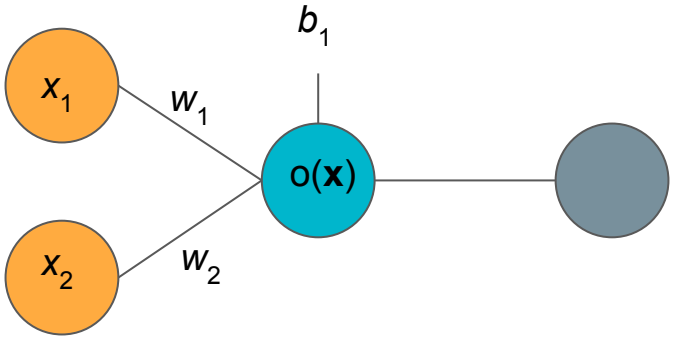
What is deep learning?

- Basically, it's a rebranding of neural networks.



Starting Simple

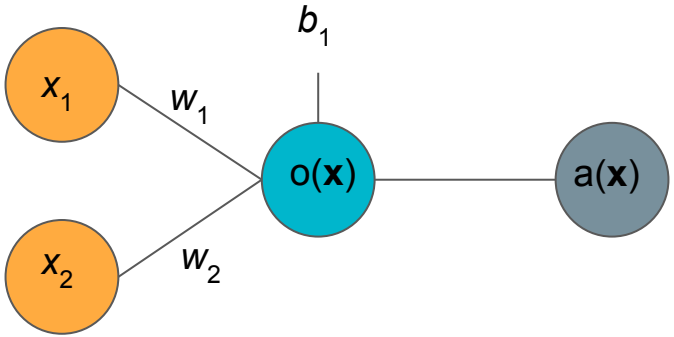
- The first neural network was basically a single mathematical function that was inspired by biological neurons.



$$o(x) = b + \sum_{i}^{n} x_i w_i$$

Starting Simple

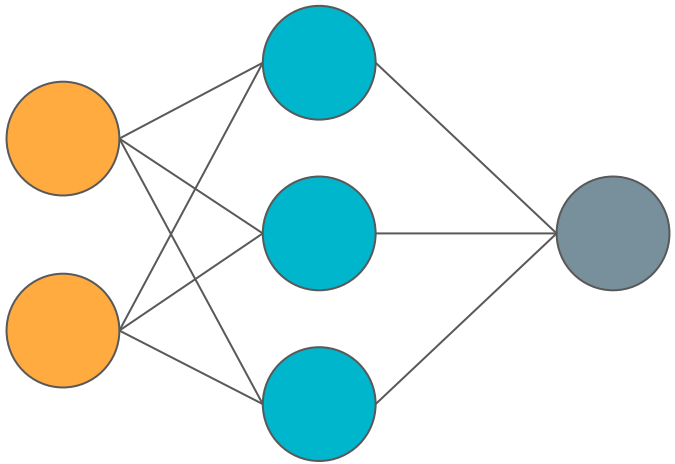
- We typically apply a function to the output of that weighted sum



$$a(x) = a(o(x)) = a(b + \sum_{i}^{n} x_i w_i)$$

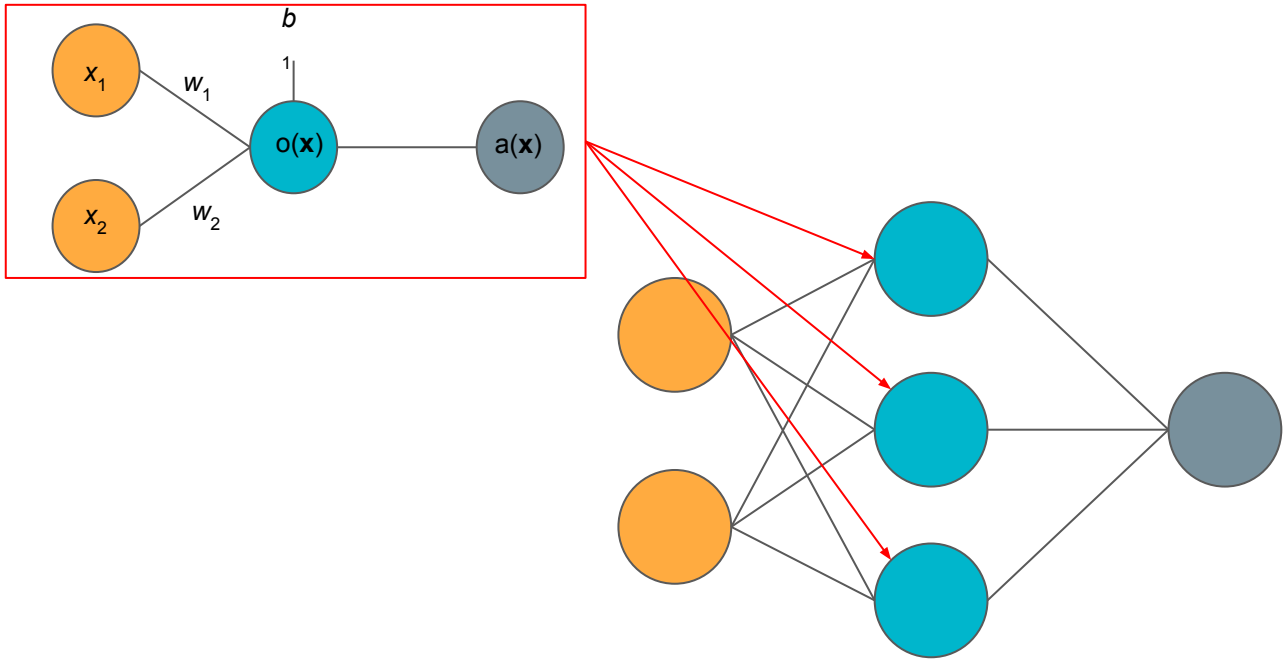
Starting Simple

- We can arrange these ‘neurons’ in parallel



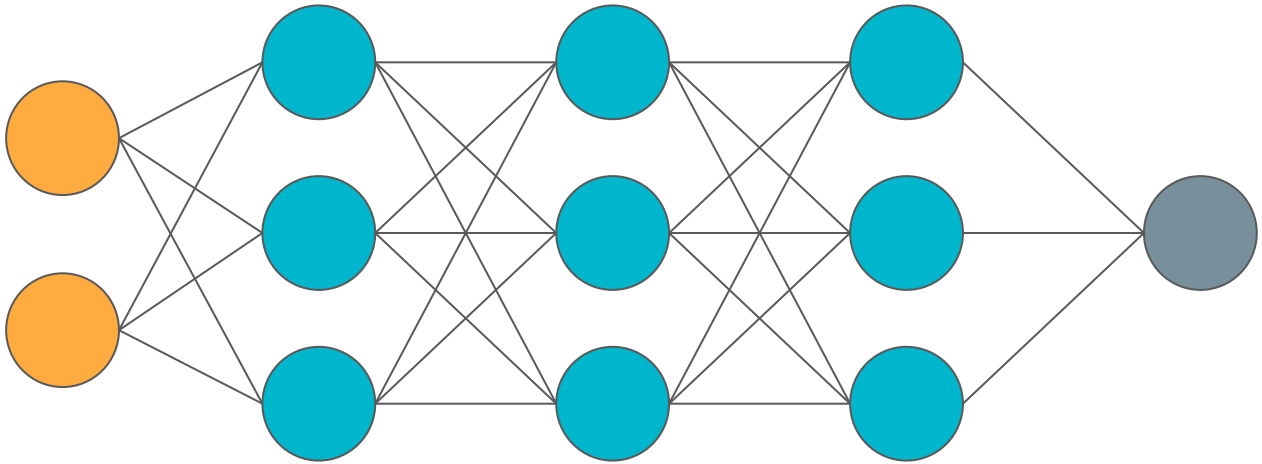
Starting Simple

- We can arrange these ‘neurons’ in parallel



Starting Simple

- We can arrange these ‘neurons’ in parallel and sequentially

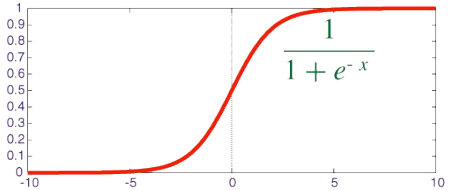


Activation Functions

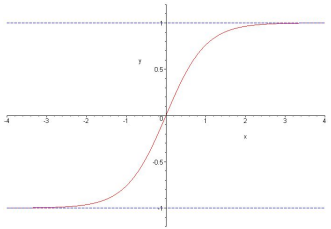
- There are a handful of common activation functions that are applied to each unit in a neural network.
    - Used to increase the flexibility and expressivity of neuron outputs

Activation Functions

- There are a handful of common activation functions that are applied to each unit in a neural network.
  - Used to increase the flexibility and expressivity of neuron outputs

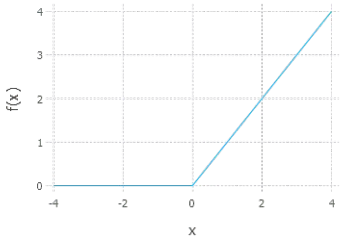


Sigmoid Function - popular in the past but has fallen out of favor



$$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

Tanh Function - similar to sigmoid except has the benefit of being centered on 0, still common in RNN



Rectified Linear Unit (ReLU) - Most common activation function now. It's simply max(x, 0)

Neural Network - References

Hugo Larochelle - Neural Network Youtube playlist

https://www.youtube.com/watch?v=SGZ6BttHMPw&list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH

Deep Learning book chapter 6

http://www.deeplearningbook.org/contents/mlp.html

Three blue one brown - what is a neural network?

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Why create networks in this way?

Representation Learning

Representation learning

- Deep learning models are representation learning systems
  - The models learn useful ways to represent the data in order to best solve the given problem
  - We will refer to the ‘deep’ part of the network as the representation learning portion
  - The output layer is responsible for producing the desired output based on the learned representations

Representation learning

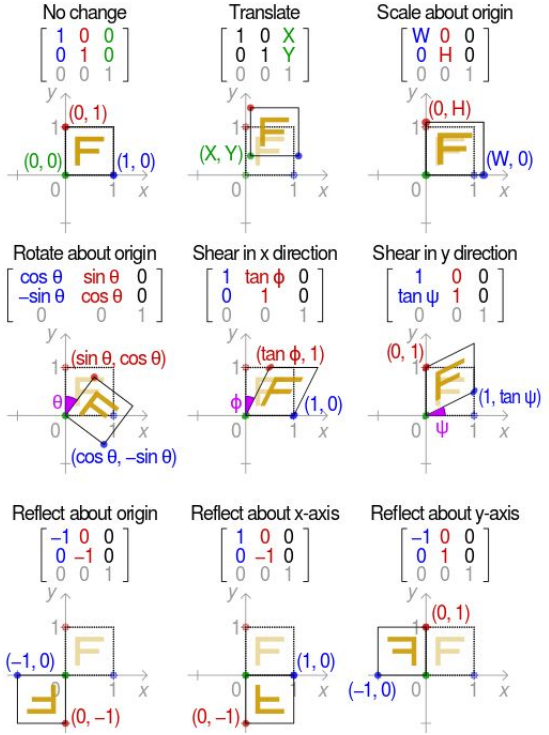
- Deep learning models are representation learning systems
  - The models learn useful ways to represent the data in order to best solve the given problem
  - We will refer to the ‘deep’ part of the network as the representation learning portion
  - The output layer is responsible for producing the desired output based on the learned representations

How does the representation happen?

Each of the units in a neural network is doing a linear combination of weights and input data.

The type of linear combination that we are doing in a neural network looks like something that we call an Affine Transformation.

$$x \mapsto Wx + b$$



Note: in affine transformations the lines stay parallel

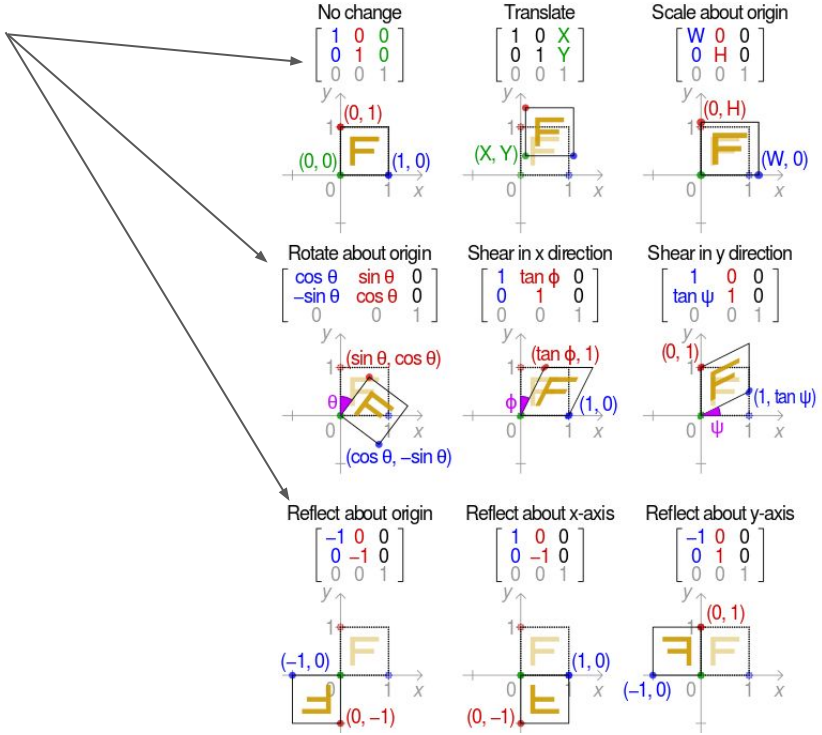
Source: https://en.wikipedia.org/wiki/Affine_transformation

How does the representation happen?

Each of the matrices can be applied to the matrix representing the letter “F” to perform some transformation on it.

In out case the transformation matrix are the weights of the neural network.

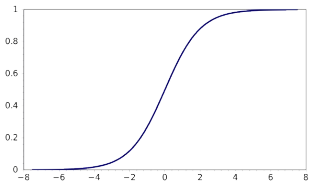
$$x \mapsto Wx + b$$



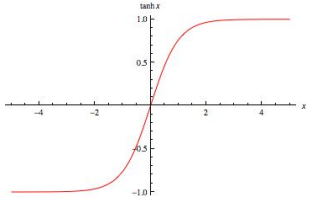
Note: in affine transformations the lines stay parallel

Source: https://en.wikipedia.org/wiki/Affine_transformation

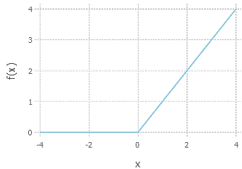
How does the representation happen?



http://tikalon.com/blog/blog.php?article=2011/sigmoid



http://mathworld.wolfram.com/HyperbolicTangent.html



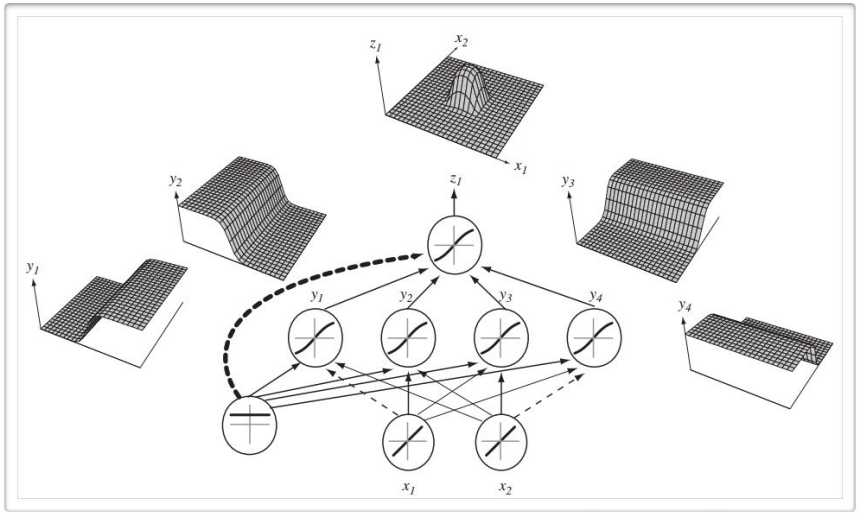
http://int8.io/neural-networks-in-julia-hyperbolic-tangent-and-relu/

Using Representations

- By combining simple equations together in parallel and sequentially we can produce some really useful projections of our data
  - Each neuron produces an affine transformation followed by some non-linearity (activation function)
  - The output of one layer becomes the input to the next, each transformation builds on the previous layer's transformations

Using Representations

- By combining simple equations together in parallel and sequentially we can produce some really useful projections of our data



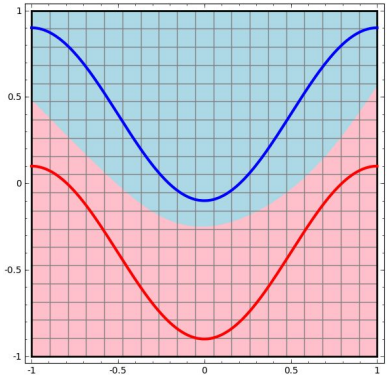
(from Pascal Vincent's slides)

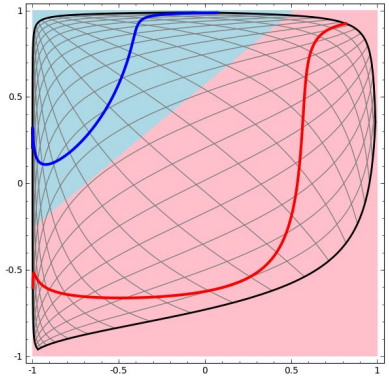
http://www.cs.cmu.edu/~rsalakhu/10807_2016/Lectures/Lecture_NN_Part1.pdf

Who in turn took this slide from another presentation (Pascal Vincent)

Using Representations

- The final output layer of a neuron network is generally constrained to be some type of linear model
- The transformations become necessary to ‘warp’ the data into a space that a linear model can solve



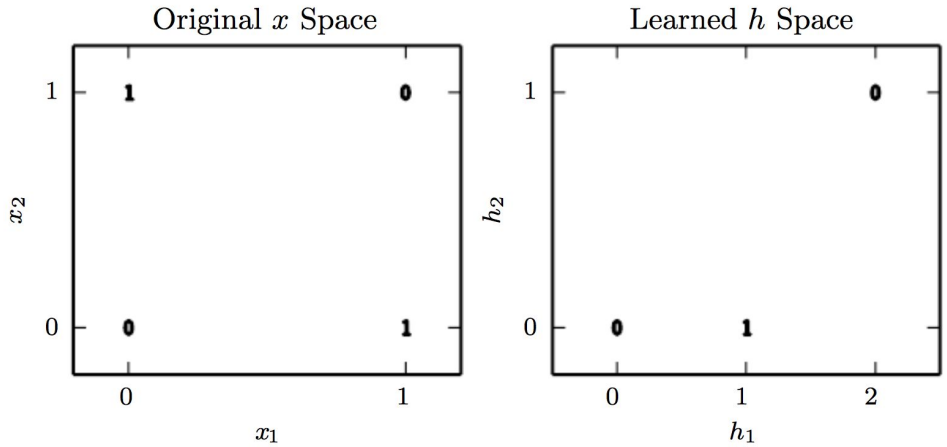


A linear model can only attempt to separate the spaces between the two classes of data with a straight line. Before the transformation this is impossible. But, the ‘deep’ layers of a neural network can learn to project the space in such a way that a straight line can perfectly separate the two spaces.

Source: Christopher Olah (Google Brain) http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Using Representations

- The final output layer of a neuron network is generally constrained to be some type of linear model
- The transformations become necessary to 'warp' the data into a space that a linear model can solve

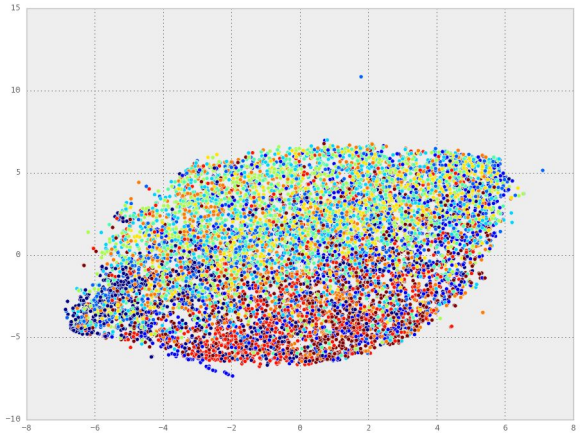


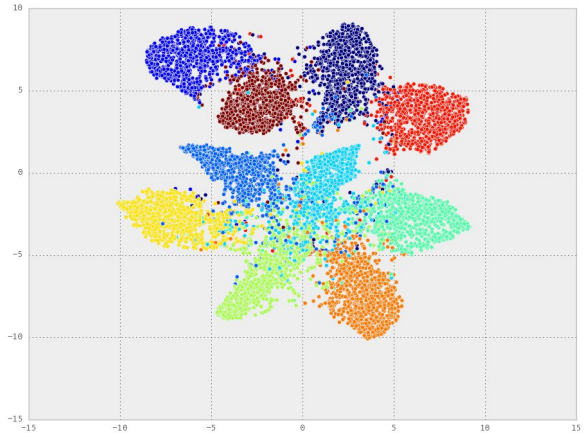
Another example is the XOR problem. On the left we have the original dataset, on the right we have the learned feature representations that allows for a linear separation.

Source deep learning book: http://www.deeplearningbook.org/

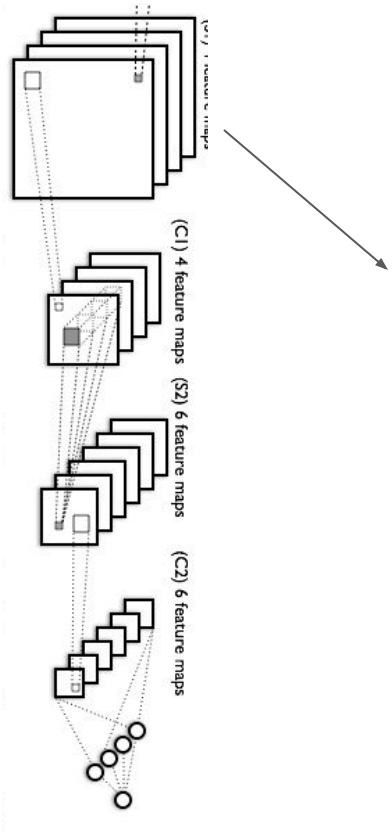
Exploring Representations

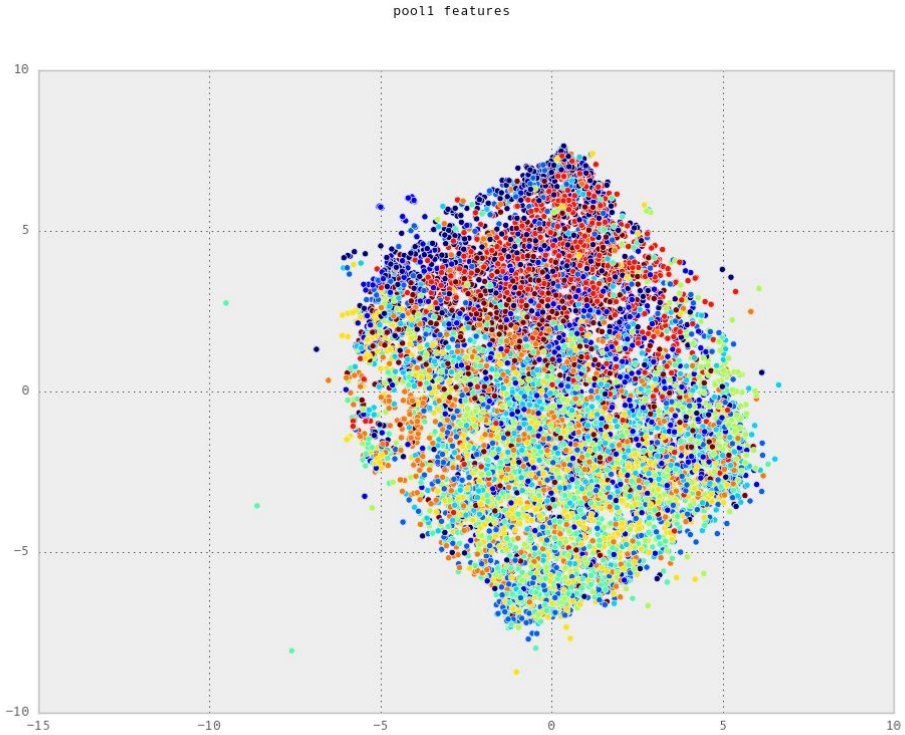
- Attempting to find correct clusters in an image dataset is difficult with raw data
- Instead we can cluster the images using the representations learned by a neural network



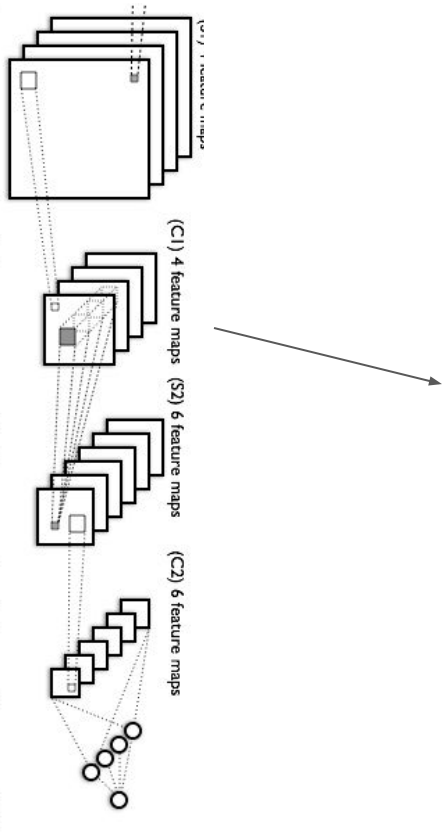


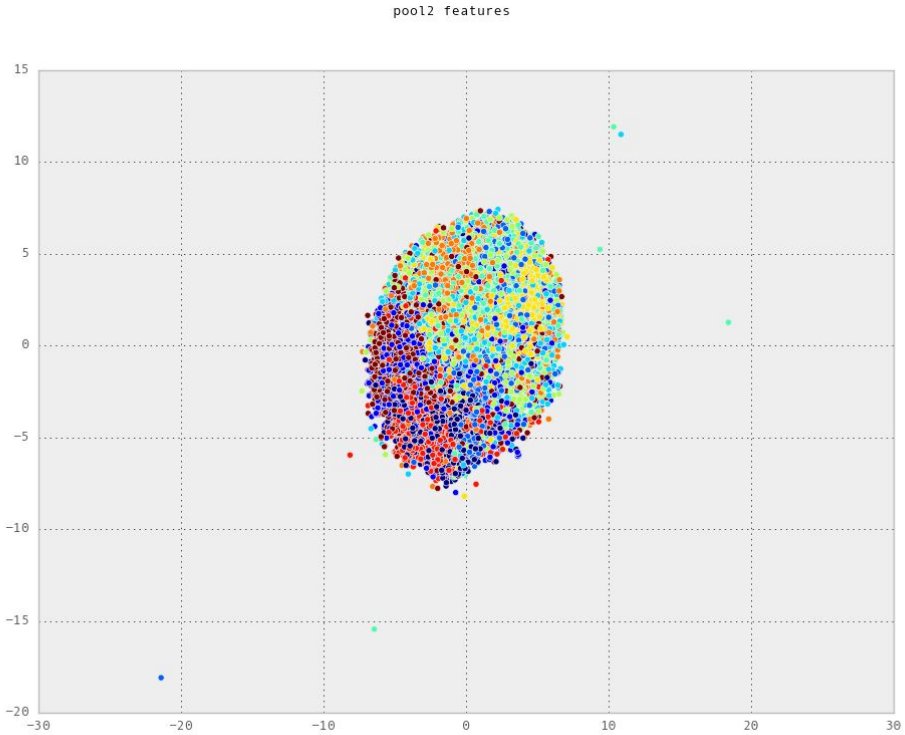
Exploring Representations



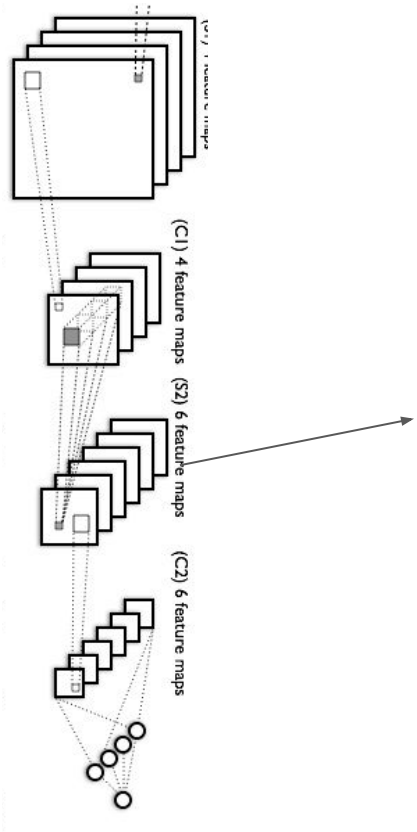


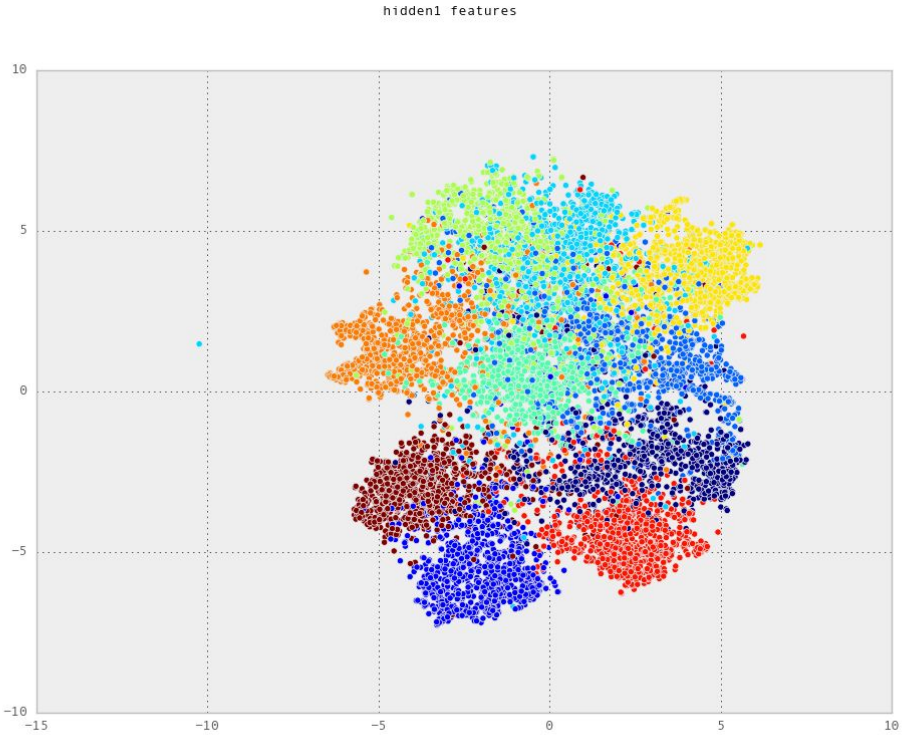
Exploring Representations



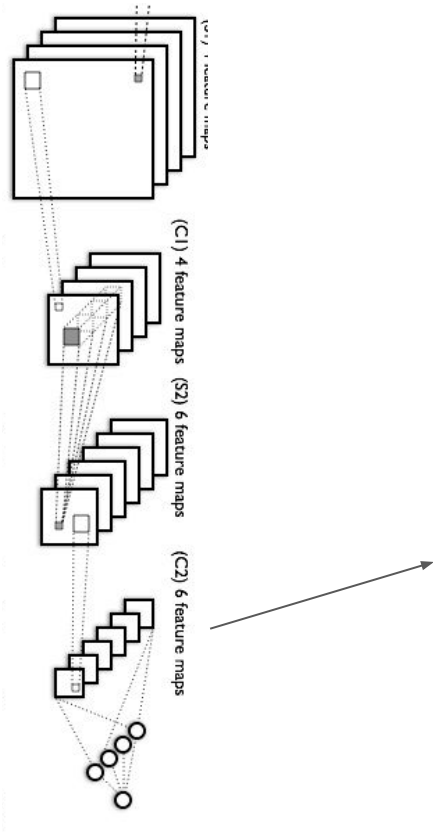


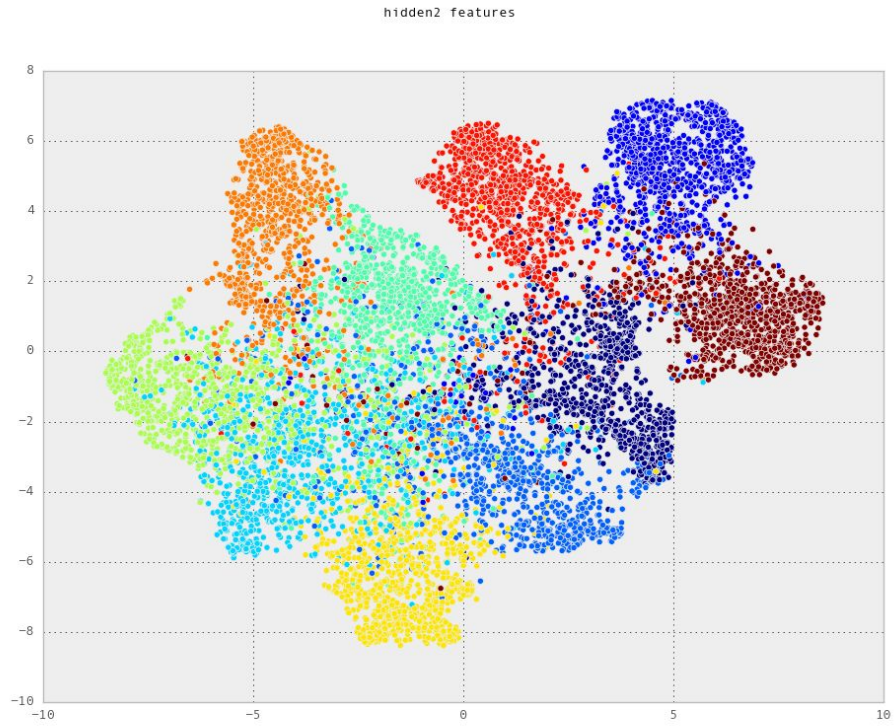
Exploring Representations



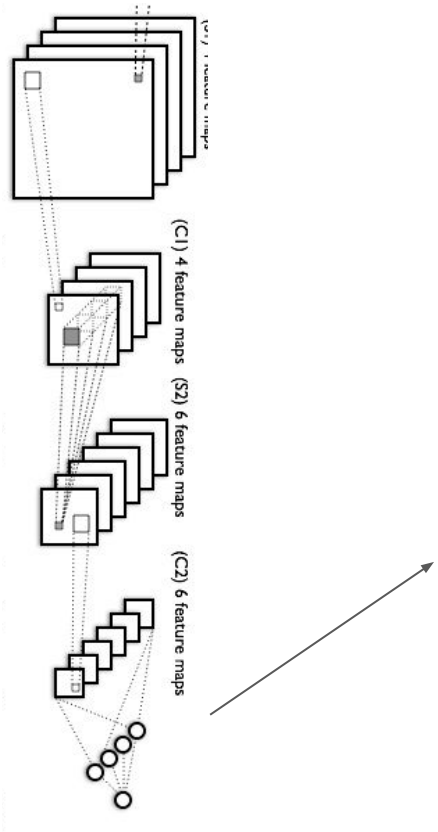


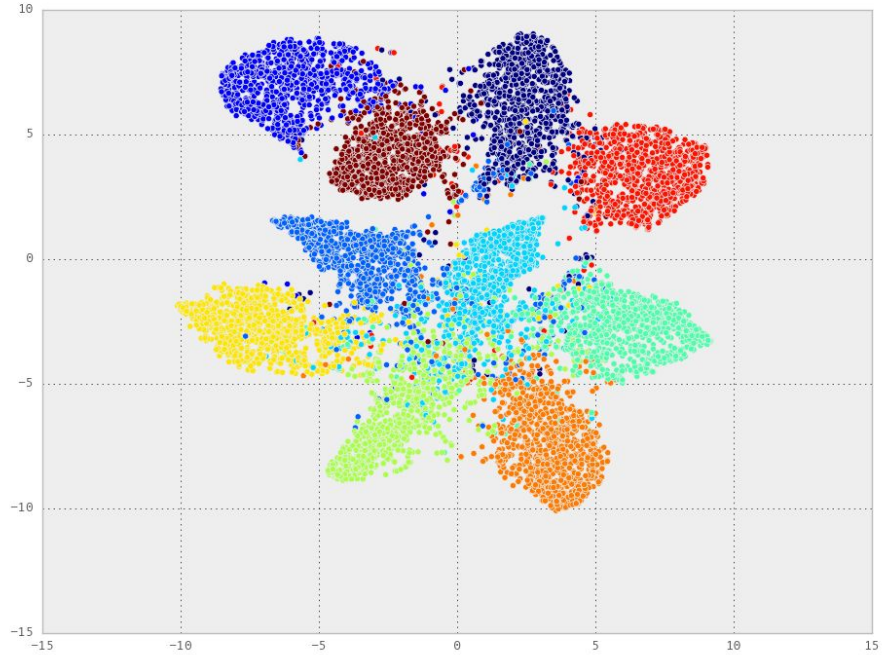
Exploring Representations





Exploring Representations





Representation Learning

- This phenomenon has implications for training neural networks both from scratch and fine tuning already trained neural networks
  - Design neural networks to leverage this observation
  - Sometimes need to create intermediate representations for the end goal (object detection networks)
  - We can re-use generic representations learned from one task for another

- Think of the deep parts of a neural network as the parts of the network responsible for the representation learning
  - The final layers of the network are responsible for using those representations for some task
  - Most of the layers in a neural network are working to make the problem easier to solve

Representation Learning - References

Decoding the Thought Vector

http://gabgoh.github.io/ThoughtVectors/

Christopher Olah: Neural Network Manifolds

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Learning book chapter 1

http://www.deeplearningbook.org/contents/intro.html

Representation Learning - References

Linear Algebra Review (highly recommended) - Three blue one brown, essence of linear algebra

https://www.youtube.com/watch?v=kjBOesZCoqc&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

Training a Network

Training

- In order to get these representations to happen we need to set our weights to meaningful values
  - The wrong values won't help us solve the problem and at worst make the problem harder

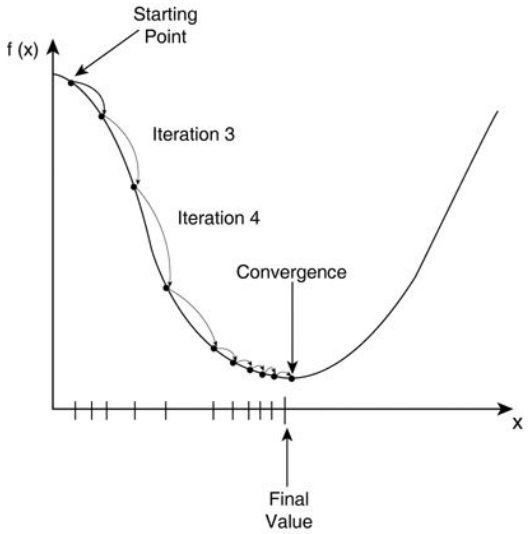
- To get the appropriate weight values we set up our system in a way that we can exploit some useful properties of mathematics
  - Set up a simple equation to score the network on some training set
  - Set up an algorithm to evaluate which direction to nudge weights to improve on the scoring function
  - Set up an algorithm to nudge the weights in the desired direction

Training - Loss Function

- The loss function is used to evaluate how well the network is doing during training
  - Needs to be differentiable
  - Should be a good metric for the problem that you are trying to solve
    - Cross entropy loss for classification
    - Mean squared error for regression
  - Higher loss values should mean that we are doing worse on the problem
    - This allows us to use calculus to get the derivative of the loss function with respect to the weights
    - In doing so we know which direction to nudge the weights to improve on our task

Training - Gradient Descent

- An iterative process
  - Evaluate loss function at current weight values
  - Take gradient (derivative) of loss function w.r.t. Weights
  - Nudge weights so that we do better on the next pass



$$F(k) = (t(k) - a(k))^2$$

Training - Backpropagation

- With gradient descent we know which direction we need to nudge the weights
  - We have many layers that are connected and dependent on each other
  - When we nudge the weights in each layer we need to make sure that we are sensitive to these dependencies
  - Need to figure out how each intermediate layer impacts the output and subsequent loss calculation

Training - Backpropagation

- Suppose that we have functions
  - ***y = g(x)***
  - ***z = f(g(x)) = f(y)***
- To find the derivative of ***z*** w.r.t. ***x*** we have to move through the intermediate function ***y()***.

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- In the case of deep learning, each of these intermediate functions are the layers
  - Calculating the derivative in this way gives us the direction that we need to update the weights for that layer in order to improve on the task

Training - Backpropagation

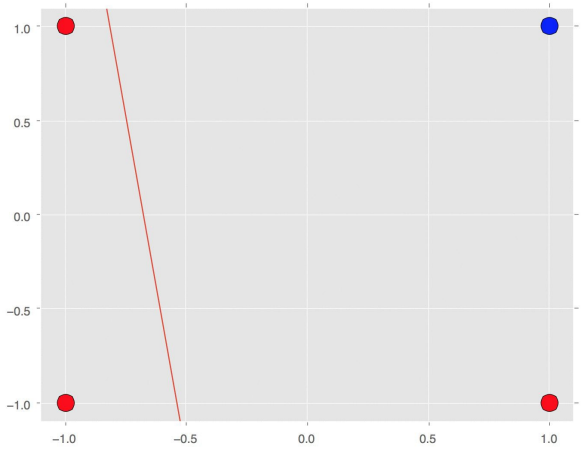
- If we think of each layer of the neural network as being some function ***f()*** then we just have to apply that same chain rule property to get the direction of the gradient for each layer.
    - Each time we apply the chain rule and arrive at the weights of a layer we can update them in the opposite direction of the gradient

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

- We call this back propagation because we propagate the error signal backwards through the network from output layer to input layer
    - This process is handled by all modern deep learning frameworks automatically (automatic differentiation).

Training - Stochastic Gradient Descent

- Training a neural network is an iterative process
  - Push some data through the network (randomly chosen subset of data - hence stochastic)
  - Calculate the error using the chosen loss function
  - Use the backpropagation algorithm to update the weights at each layer in order to decrease the loss on the next pass (gradient descent part, we move in the opposite direction of the grad.)
  - Repeat until satisfied



Training - Stochastic Gradient Descent

- Training a neural network is an iterative process
  - Push some data through the network (randomly chosen subset of data - hence stochastic)
  - Calculate the error using the chosen loss function
  - Use the backpropagation algorithm to update the weights at each layer in order to decrease the loss on the next pass (gradient descent part, we move in the opposite direction of the grad.)
  - Repeat until satisfied

Neural Network - References

Hugo Larochelle - Neural Network Youtube playlist

https://www.youtube.com/watch?v=SGZ6BttHMPw&list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH

Deep Learning book chapter 6

http://www.deeplearningbook.org/contents/mlp.html

Three blue one brown - what is a neural network?

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

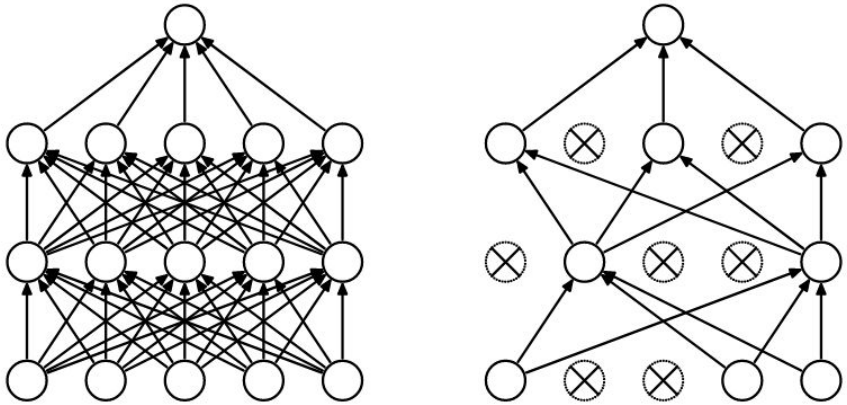
Neural Network Enchancements

Ad-hoc Enhancements

- Over the years some techniques were developed to help with NN training and generalization
- Two big ones
  - Dropout
  - Batch Normalization

Dropout

- Dropout was introduced in 2014 in the paper “Dropout: A simple way to prevent neural networks from overfitting” by Srivastava et al.
  - During training randomly drop a certain percentage of ‘neurons’ from the network on some epoch

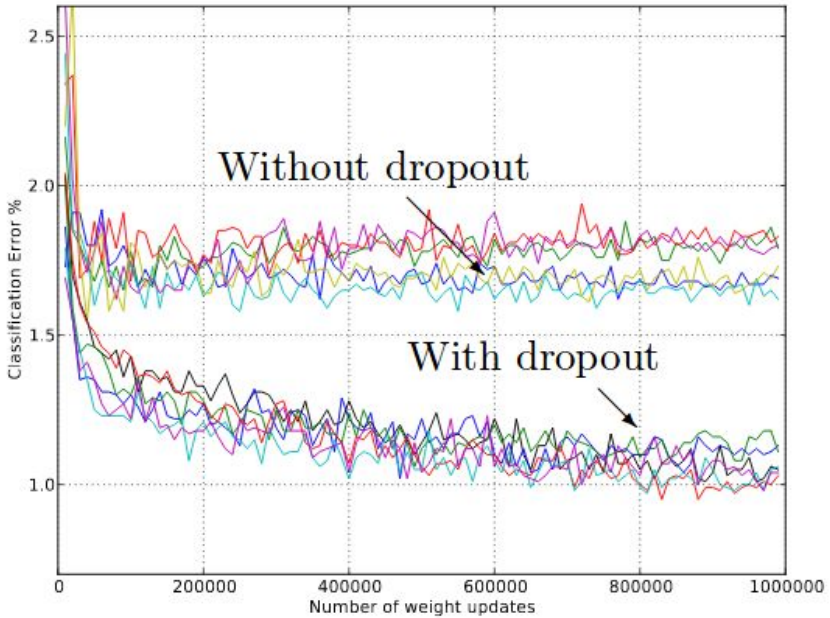


(a) Standard Neural Net (b) After applying dropout.

http://jmlr.org/papers/v15/srivastava14a.html

Dropout

- Can significantly improve generalization



http://jmlr.org/papers/v15/srivastava14a.html

Batch Normalization

- Batch Norm was introduced in 2015 in the paper “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift” by Ioffe et al.
    - During training normalize the output of a neural network layer

https://arxiv.org/pdf/1502.03167.pdf

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

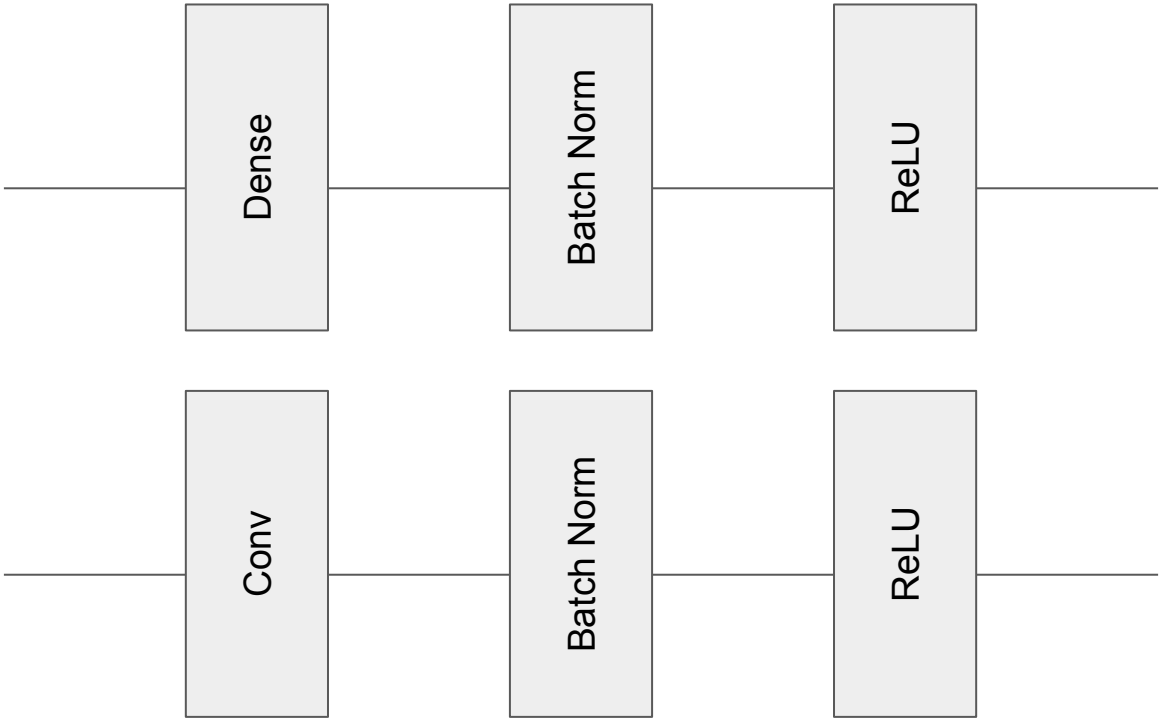
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

https://arxiv.org/pdf/1502.03167.pdf

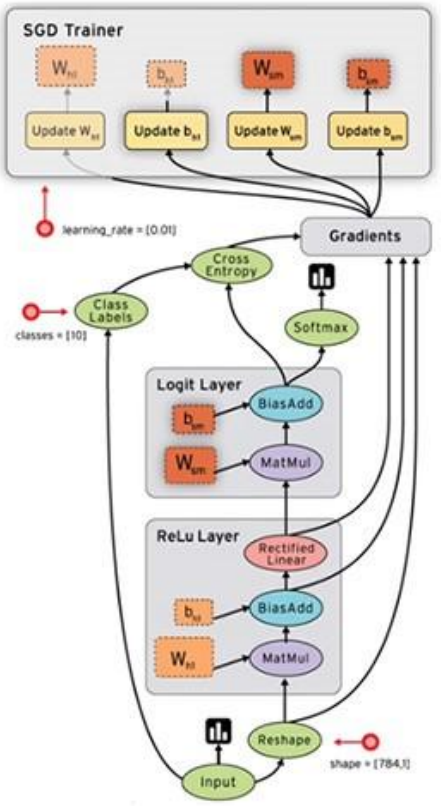
Batch Normalization



Getting Started with TensorFlow

TensorFlow

- TensorFlow is a framework for defining and running computational graphs
  - Primarily used for deep learning
  - Automatically calculate gradients w.r.t the loss function
    - Allows user to set up any graph with any functions that are differentiable
    - Train the weights in that graph with some kind of numerical loss minimization (SGD)



http://sdsawtelle.github.io/blog/output/getting-started-with-tensorflow-in-jupyter.html

TensorFlow - Basics

- **`tensorflow_tutorials/getting_started.ipynb`**
- TensorFlow graphs, scopes, and Sessions
- Working with various data types and buckets in TF
    - tf.constants - immutable numeric objects (scalars, vectors, matrices, or tensors)
    - tf.Variable - mutable numeric objects, values often change during gradient descent
    - tf.placeholder - empty buckets of predefined shape that user will provide actual numbers for
- Working with various mathematical operations
    - tf.matmul(x,y) - dot product of tensors x and y
    - tf.mul(x,y) - element-wise multiplication of two tensors
    - tf.add(x,y) - add two tensors
    - tf.sub(x,y) - subtract two tensors
- Thinking about tensor shapes and operations when designing models
- Set up linear regression to predict housing prices based on house features using only basic TF operations

TensorFlow - Basic Neural Network

- **`tensorflow_tutorials/basic_neural_netsipynb`**
- Designing neural network layers in TensorFlow
  - Emphasis on designing generic and reusable layers to aid in experimentation and prototyping
- Using TensorFlow optimizers and automatic differentiation to train a neural network
  - Mini-batch gradient descent
- Saving TensorFlow sessions as checkpoints
  - Loading checkpoint to continue training
  - Making predictions with a trained neural network in a checkpoint