Deep Learning; A Hands-on Introduction

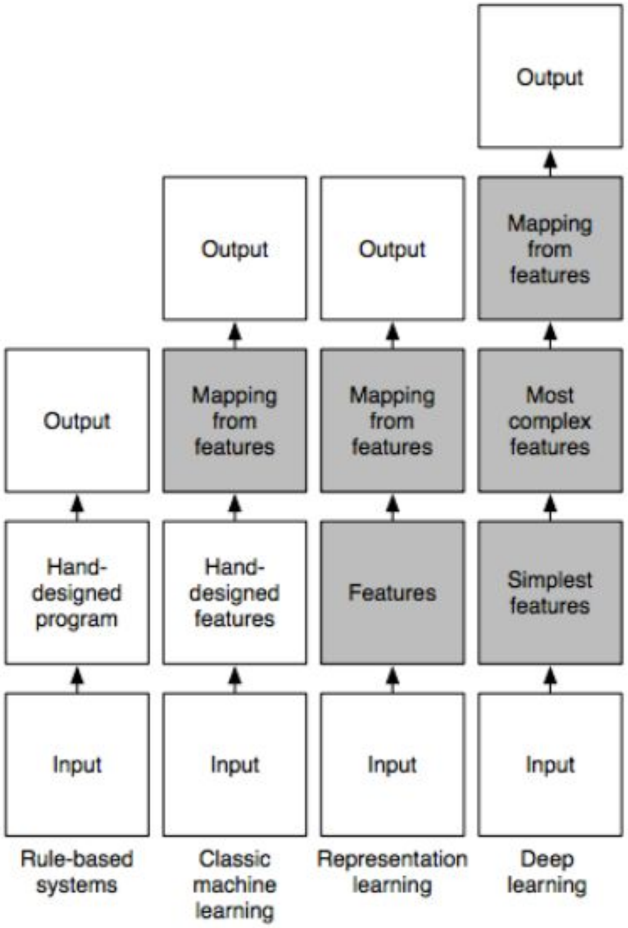
Hamid Mohammadi
Ph.D. Candidate at OHSU,
Research Scientist at ObEN Inc.

Advanced Topics in Speech Processing Course, UCLA
04/23/2018

Colab Code: https://drive.google.com/open?id=1_FxdrwqS8y8CuZEkFVKdxcLf0BIQgNdI

Artificial Intelligence Frameworks

- Rule-based:
  - Design all the rules manually by experts
  - E.g. Expert systems
- Classic Machine Learning:
  - Features are designed by experts
  - Models operate on the features
  - E.g. MFCCs for speech
- Modern ML, Representation Learning:
  - Learn representations using some techniques
  - Models operate on the learned features
  - E.g. Autoencoders
- Modern ML, Deep Learning:
  - Features/mappings are learned from raw data jointly

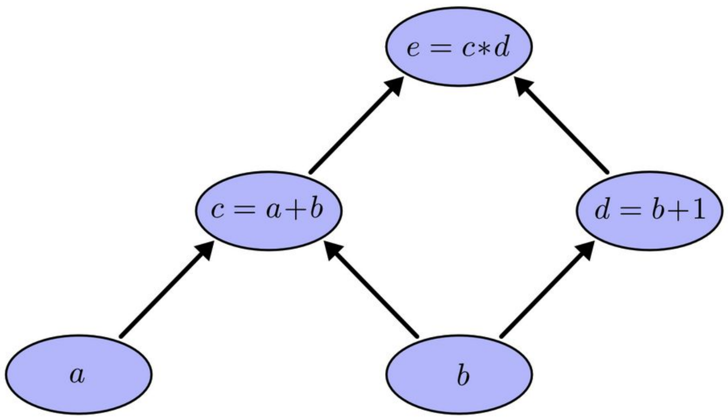


Outline

- Computational Graphs
- Linear Regression
- Logistic Regression (~Perceptron)
- Shallow Neural Networks
- Deep Neural Networks
- Convolutional NNs
- Recurrent NNs
- Future Readings

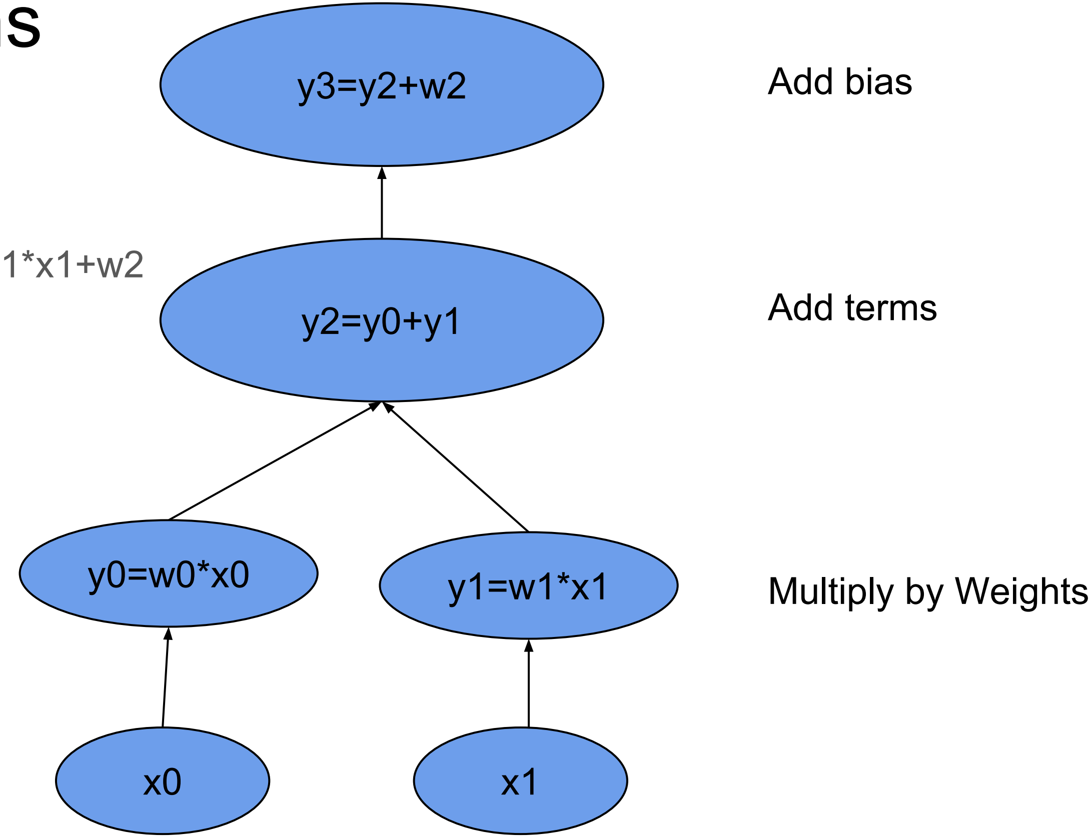
Computational Graphs

- A data structure to represent about mathematical expressions
- Example: $e = (a + b) * (b + 1)$



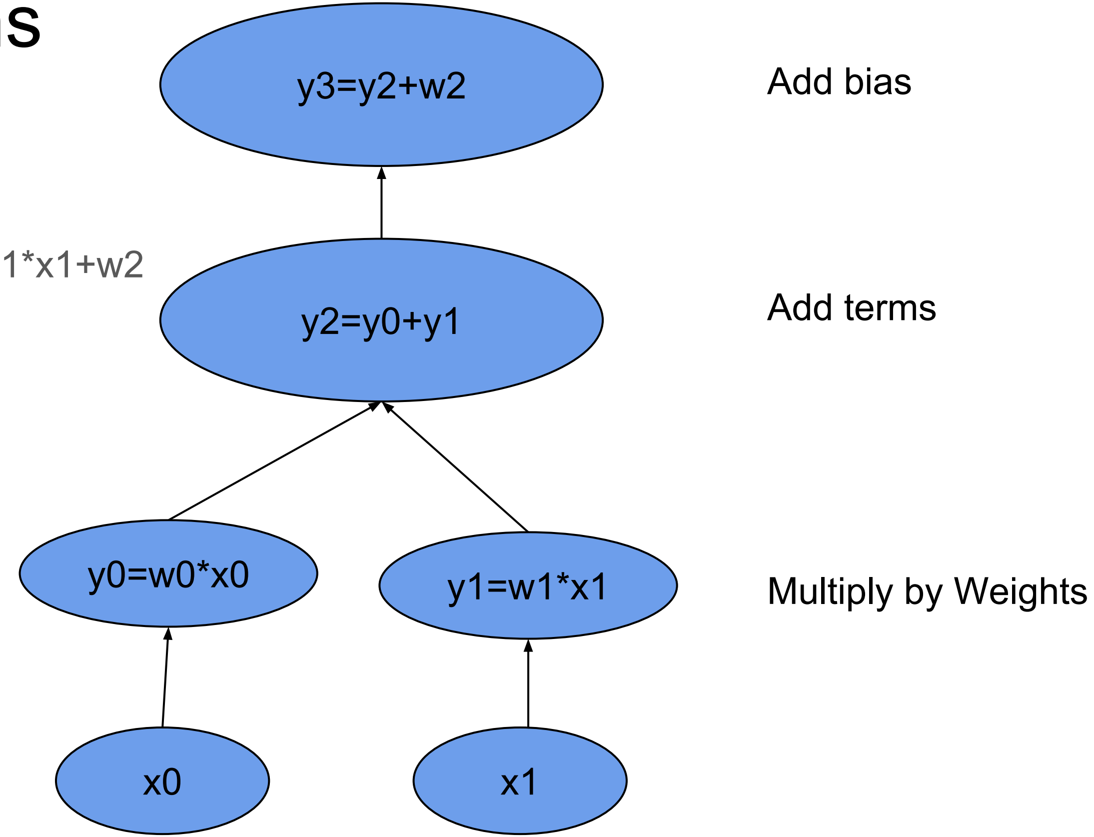
Computational Graphs

- Example, Linear Regression
    - Input: x0, x1
    - Predicted Output: y3=w0*x0+w1*x1+w2
    - Original Output: y
    - Parameters: W=w0, w1, w2



Computational Graphs

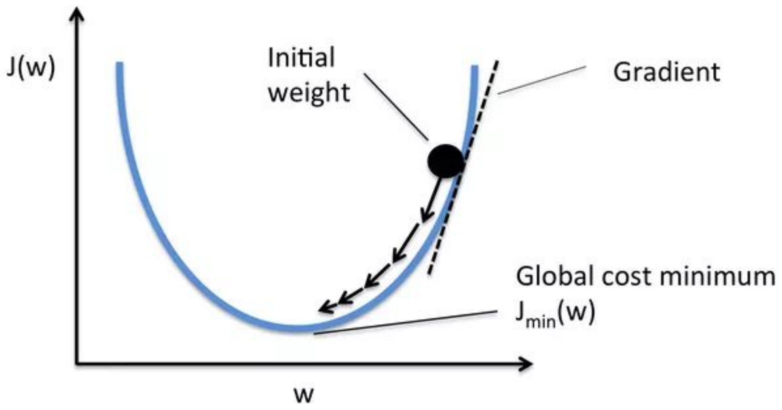
- Example, Linear Regression
    - Input: x0, x1
    - Predicted Output: y3=w0*x0+w1*x1+w2
    - Original Output: y
    - Parameters: W=w0, w1, w2
- How to optimize?
    - Minimize Mean-squared error:
        - Loss: J(W)=(y-y3)**2
- How to minimize:
    - Gradient Descent



Gradient Descent: Estimating Graph Parameters

- At each iteration,
  - take a step proportional to the negative of the gradient of the error function with respect to weights at point W
-
- repeat until convergence:

$$w0 := w0 - \alpha \frac{du}{dw0} J(W)$$
$$w1 := w1 - \alpha \frac{du}{dw1} J(W)$$
$$w2 := w2 - \alpha \frac{du}{dw2} J(W)$$



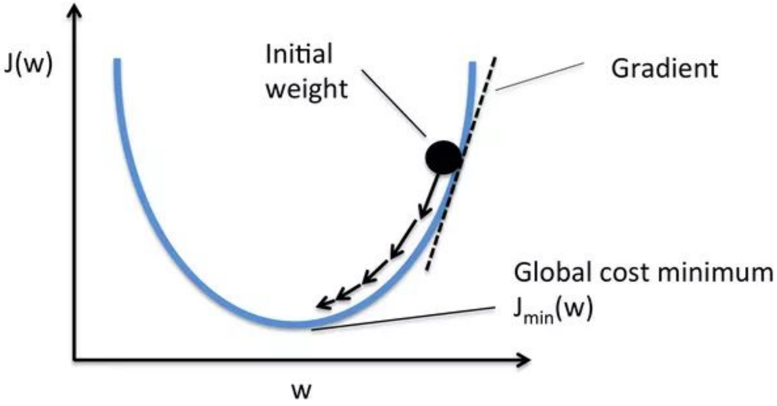
Gradient Descent: Estimating Graph Parameters

- At each iteration,
    - take a step proportional to the negative of the gradient of the error function with respect to weights at point W
-
- repeat until convergence:

$$w0 := w0 - \alpha \frac{du}{dw0} J(W)$$
$$w1 := w1 - \alpha \frac{du}{dw1} J(W)$$
$$w2 := w2 - \alpha \frac{du}{dw2} J(W)$$

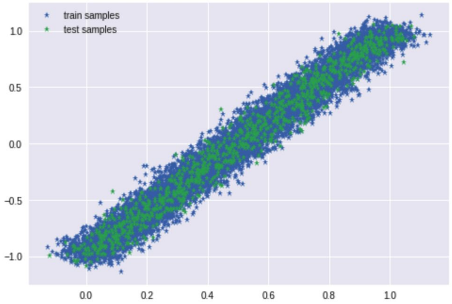
Learning rate

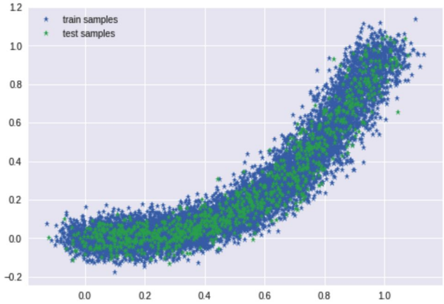
Gradient of cost function at w



Linear Regression

- A linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X
- Equation $y = w_0 * x_0 + w_1 * x_1 + ... + b$
- Some closed-form solutions exists, but we will use gradient descent to estimate the parameters on a linear/non-linear toy data:
-
-



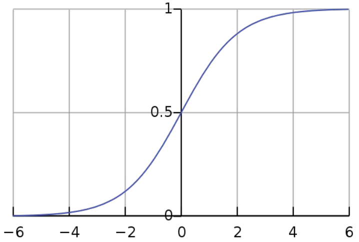


- Demo in TensorFlow:
  - https://colab.research.google.com/drive/1qO3iJhxC9HBee123LANiZqcGr6bf5OGH

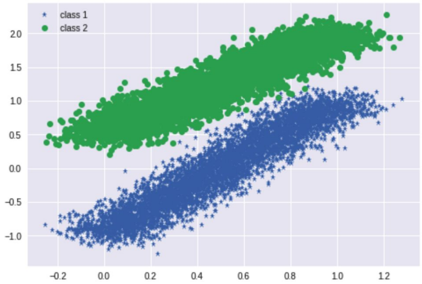
Single-Layer Neural Networks (Perceptron)

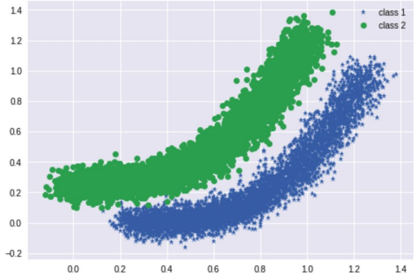
- A.k.a Logistic Regression, a simple linear classifier
- Equation

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



-
- The model has a hard-decision, to be able to compute gradients, apply sigmoid on w.x+b
- We intend to estimate the parameters using Gradient Descent





- Demo in TensorFlow:
  - https://colab.research.google.com/drive/11gXVnBPqnZTN8DXLEomRMj7hHubWjMwC
  -

Two-Layer Neural Network (Shallow NN)

- Two computations of type $f(W.x + b)$ where f is a nonlinear function
- Equation $y = g(f(x.W_1 + b_1).W_2 + b_2)$
- Regression demo on nonlinear regression data: https://colab.research.google.com/drive/1H8ms1Jzeze8ki7FWtszvcIN5bBfRJ8fw
- Classification demo on linearly non-separable data: https://colab.research.google.com/drive/1LBQ-EiH3d4hxly492DQdZOd5q7Tq8RBZ
-

Recap: Parameter Optimization in TensorFlow

- What are input and output? X and Y
- Specify the model architecture (How to connect X to Y using computations that have parameters)
- Specify a cost function to minimize.
- How to update the parameters? Gradient descent and its variants
- How to regularize the parameters? L1/L2 add to cost function, Dropout

Recap: Parameter Optimization in TensorFlow

- What are input and output? X and Y
- Specify the model architecture (How to connect X to Y using computations that have parameters)
- Specify a cost function to minimize.
- How to update the parameters? Gradient descent and its variants
- How to regularize the parameters? L1/L2 add to cost function, Dropout

```
# inputs
x = tf.placeholder(tf.float32, shape=[None, 1])
y = tf.placeholder(tf.float32, shape=[None, 1])
```

Recap: Parameter Optimization in TensorFlow

```
# paramteres
w1 = tf.Variable(tf.random_normal([1, hidden_layer_size]))
b1 = tf.Variable(tf.random_normal([hidden_layer_size]))

w2 = tf.Variable(tf.random_normal([hidden_layer_size, 1]))
b2 = tf.Variable(tf.random_normal([1]))

# model architecture (y=f(x))
h = tf.sigmoid( tf.add(tf.matmul(x, w1), b1) )
y_ = tf.add(tf.matmul(h, w2), b2)
```

Recap: Parameter Optimization in TensorFlow

- What are input and output? X and Y
- Specify the model architecture (How to connect X to Y using computations that have parameters)
- Specify a cost function to minimize.
- How to update the parameters? Gradient descent and its variants
- How to regularize the parameters? L1/L2 add to cost function, Dropout

```
# loss funstion
loss = tf.reduce_mean(tf.square(y - y_))
```

Recap: Parameter Optimization in TensorFlow

- What are input and output? X and Y
- Specify the model architecture (How to connect X to Y using computations that have parameters)

```
# update
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

accuracy = tf.reduce_mean(tf.abs(y - y_))
```

Static vs. Dynamic Computational Graphs

- **Static**: TensorFlow and Theano, **Dynamic**: PyTorch
- Both frameworks operate on tensors and view model as directed acyclic graph (DAG)
- Both view any model as a computational graph,
- They differ drastically on how you can define them.
- TF/Theano:
  - “Data as code and code is data”
  - Graph is defined statically before runtime
  - All communication with outer world is performed via tf.Session object and tf.Placeholder
  - Harder to debug and find issues
- PyTorch:
  - things are way more imperative and dynamic
  - you can define, change and execute nodes as you go
  - the framework is more tightly integrated with Python language and feels more native
  - Easier sequence modelling
  - Easier to debug

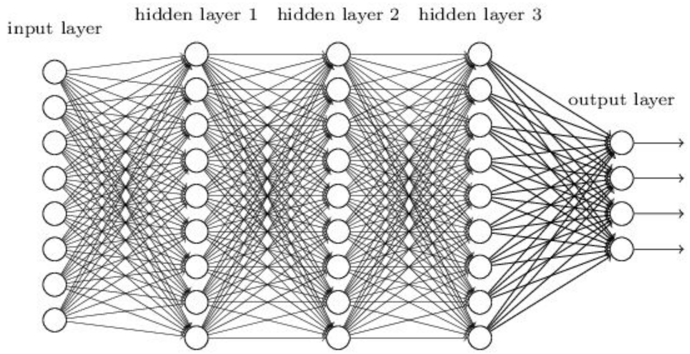
Deep Neural Networks

Artificial Neural Networks (ANNs) are a sequence of non-linear mappings

$$f(x) = s_K(w_k.(...s_2(w_2.s_1(w_1.x + b_1) + b_2)...) + b_K)$$

... ...

“Multilayer feedforward networks are **universal approximators**”, Hornik et al, 1989.



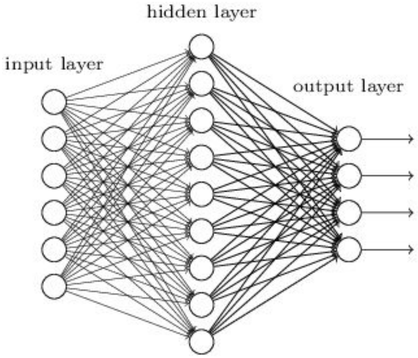
Deep Neural Networks

Artificial Neural Networks (ANNs) are a sequence of non-linear mappings

$$f(x) = s_K(w_k.(...s_2(w_2.s_1(w_1.x + b_1) + b_2)...) + b_K)$$

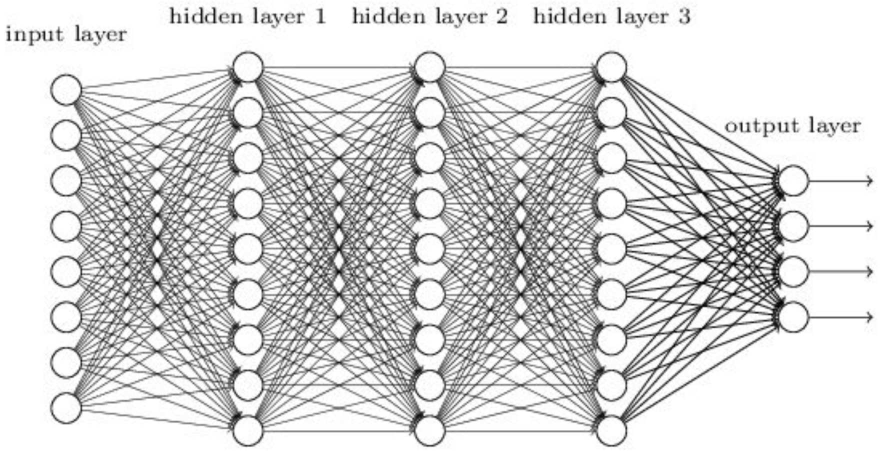
... ...

In fact, even two-layered (shallow) neural networks are **universal approximators**.



Deep Neural Networks

Deep neural networks refer to 3+ layered neural networks:



Why Deep?

If Shallow Neural Networks are universal approximators, why use deep architectures?

Why Deep?

If Shallow Neural Networks are universal approximators, why use deep architectures?

1- The brain has a deep architecture

2- Cognitive processes seem deep

3- Insufficient depth can hurt modeling

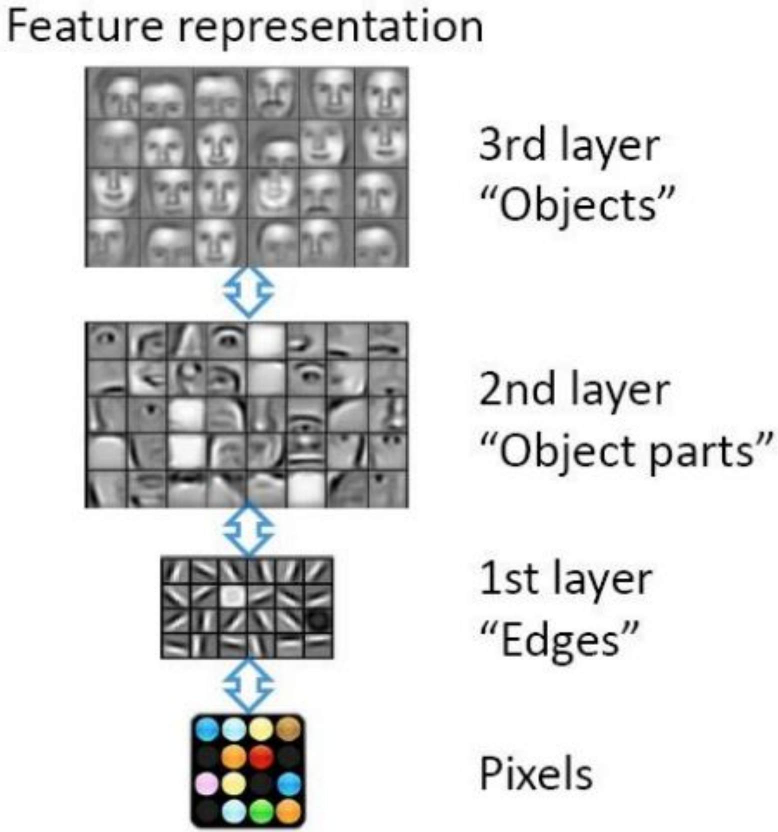
Why Deep?

If Shallow Neural Networks are universal approximators, why use deep architectures?

1- The brain has a deep architecture

- Visual cortex has a sequence of levels ,
- Each level represents the input at a different level of abstraction,
- More abstract features further up in the hierarchy, defined in terms of the lower-level ones.

Why Deep?


Feature representation
3rd layer
“Objects”
2nd layer
“Object parts”
1st layer
“Edges”
Pixels

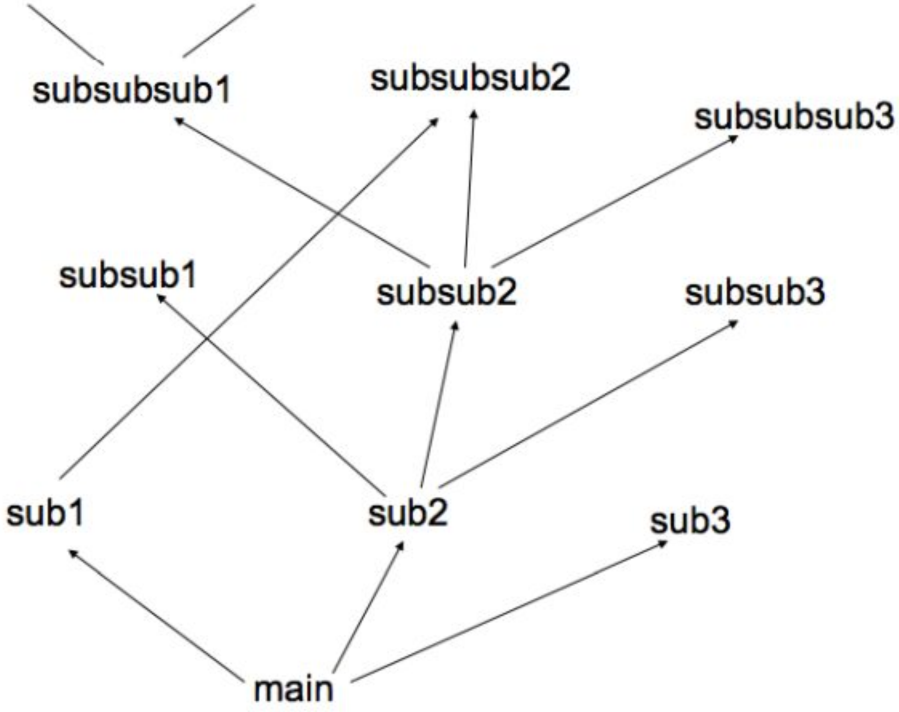
Why Deep?

If Shallow Neural Networks are universal approximators, why use deep architectures?

2- Cognitive processes seem deep

- Humans organize their ideas and concepts hierarchically,
- Humans first learn simpler concepts and then compose them to represent more abstract ones,
- Engineers break-up solutions into multiple levels of abstraction and processing

Why Deep?


subsubsub1
subsubsub2
subsubsub3
subsub1
subsub2
subsub3
sub1
sub2
sub3
main

Why Deep?

If Shallow Neural Networks are universal approximators, why use deep architectures?

3- Insufficient depth can hurt modeling

- there exist function families which the required number of nodes may grow exponentially with the input size [Hastad 1986]
  “An Average-case Depth Hierarchy Theorem for Higher Depths”, Hastad 1986.
- Some families of functions which can be efficiently (compactly) represented with *O(n)* nodes (for *n* inputs) for depth *d*
  but for which an exponential number (*O(2^n)*) of nodes is needed if depth is restricted to *d-1*

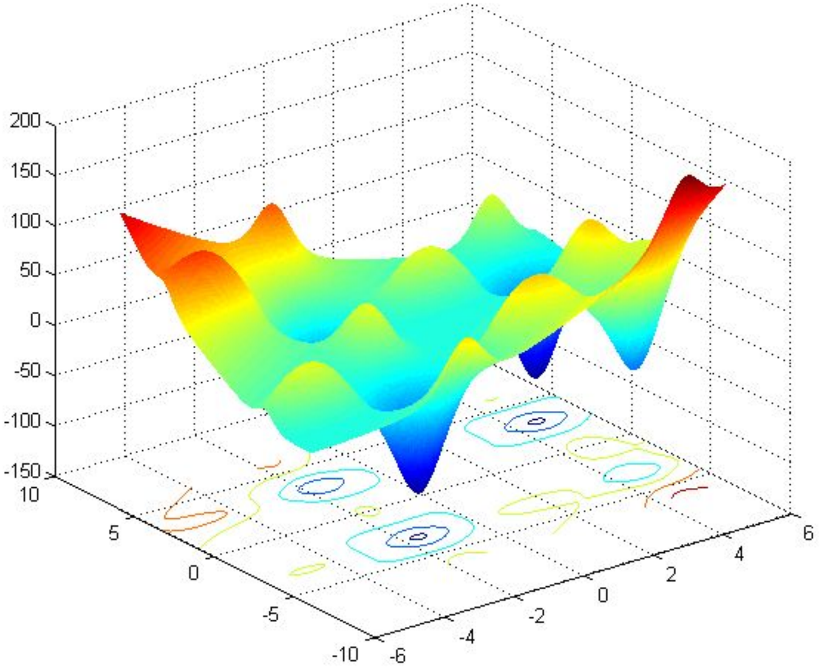
Difficulties of training deep architectures

The reason deep

- Not enough
- Not enough
- Primitive Tec primitive)
- Hard to impl e easier)

Technical issues:

- Vanishing gr or gets smaller)
- Prone to loc inear
- Regularizati



Difficulties of training deep architectures

The reason deep neural networks did not work before ~2007

- Not enough computational power (cheap GPUs today)
- Not enough data even if computation was not an issue
- Primitive Technology (especially regularization was more primitive)
- Hard to implement (open-source toolkits help reusing code easier)

Technical issues:

- Vanishing gradient: as the error back-propagated, the error gets smaller
- Prone to local minima: more local minima and more complex cost function
- Regularization: more memorizing and less generalizing

Solutions

- *Pre-training (obsolete)*: Rather than random initialization, initialize from an unsupervised network. Typically using Autoencoders or Restricted Boltzmann Machines (RBMs)
    -

Solutions

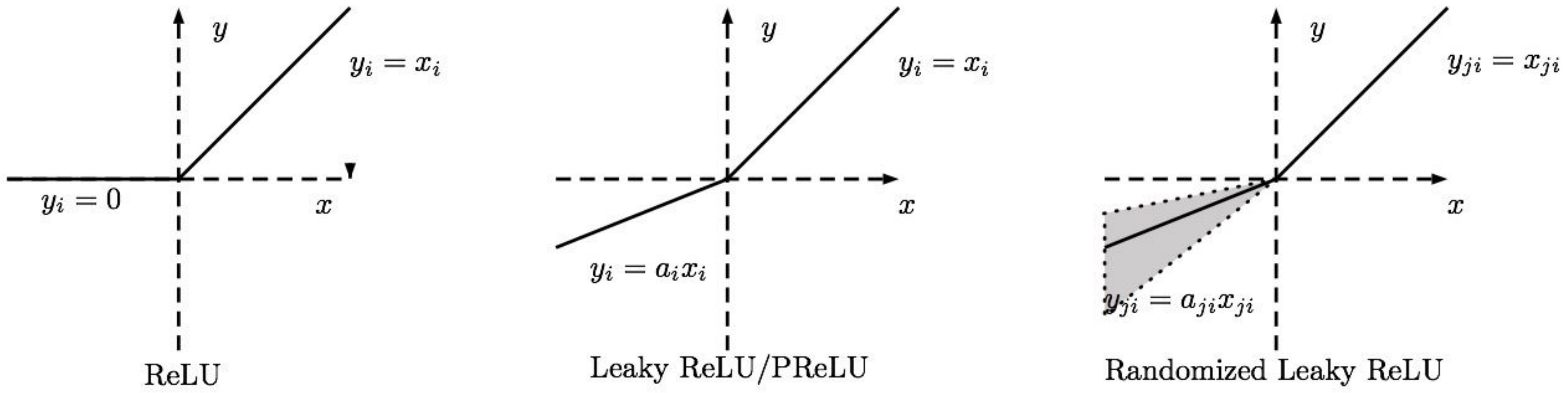
- *Pre-training (obsolete)*: Rather than random initialization, initialize from an unsupervised network. Typically using Autoencoders or Restricted Boltzmann Machines (RBMs)
- *Better transfer function*: ReLU, Leaky ReLU, R/PReLU, Maxout
-

DNN with Relu

ReLU and the variance activation functions have gained popularity recently

ReLU well for speech and image processing tasks

Faster convergence, Better convergence. Goes well with Dropout.



Solutions

- *Pre-training*: Rather than random initialization, initialize from an unsupervised network. Typically using Autoencoders or Restricted Boltzmann Machines (RBMs)
- *Better transfer function*: ReLU, Leaky ReLU, R/PReLU, Maxout
- *Regularization*: L1, L2, Sparseness, Dropout
    - Adding a penalty term to the cost function

L1 vs. L2

L1: $R(\theta) = \|\theta\|_1 = \sum_{i=1}^{n} |\theta_i|$

L2: $R(\theta) = \|\theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2$

Solutions

- *Pr* ... sed
  ne
  (R
- *Be*
- *Re*
  -

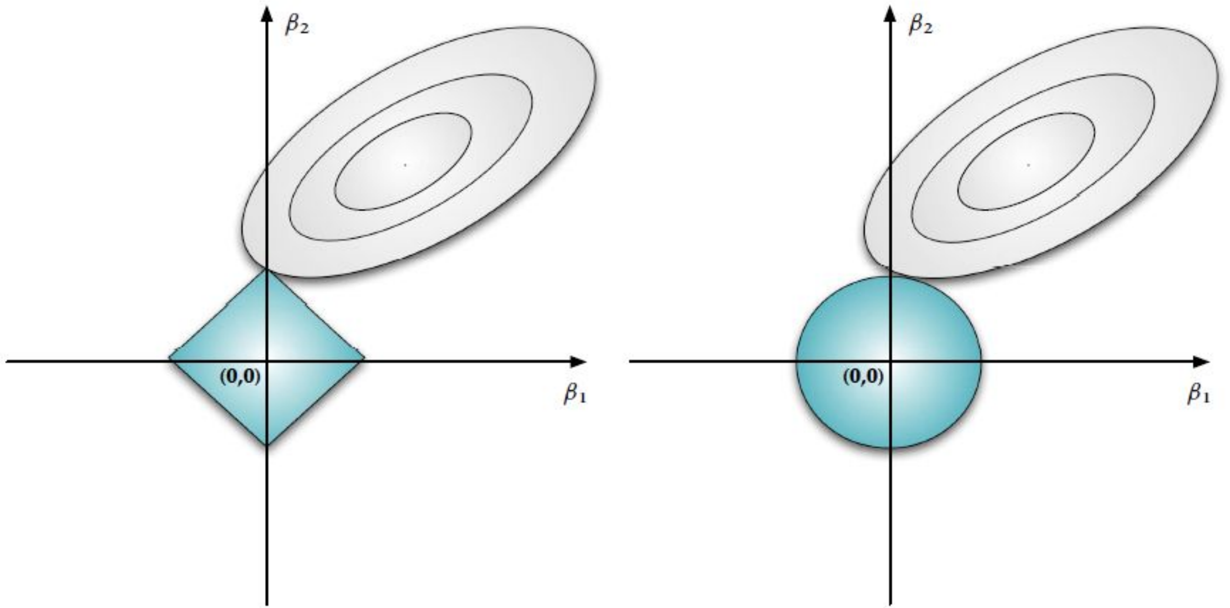
```
# TF: Deep Neural Network; L1
w1 = tf.Variable(tf.random_normal([1, hidden_layer_size]))
b1 = tf.Variable(tf.random_normal([hidden_layer_size]))
w2 = tf.Variable(tf.random_normal([hidden_layer_size, 1]))
b2 = tf.Variable(tf.random_normal([1]))
h = tf.sigmoid( tf.add(tf.matmul(x, w1), b1) )
y_ = tf.add(tf.matmul(h, w2), b2)
# loss function
loss = tf.reduce_mean(tf.square(y - y_)+a*(tf.abs(w1)+tf.abs(w2)))
```

Solutions

- *Pr* ... sed
  ne
  (R
- *Be*
- *Re*
  -

```
# TF: Deep Neural Network; L2
w1 = tf.Variable(tf.random_normal([1, hidden_layer_size]))
b1 = tf.Variable(tf.random_normal([hidden_layer_size]))
w2 = tf.Variable(tf.random_normal([hidden_layer_size, 1]))
b2 = tf.Variable(tf.random_normal([1]))
h = tf.sigmoid( tf.add(tf.matmul(x, w1), b1) )
y_ = tf.add(tf.matmul(h, w2), b2)
# loss function
loss = tf.reduce_mean(tf.square(y - y_)+a*(w1**2+w2**2))
```

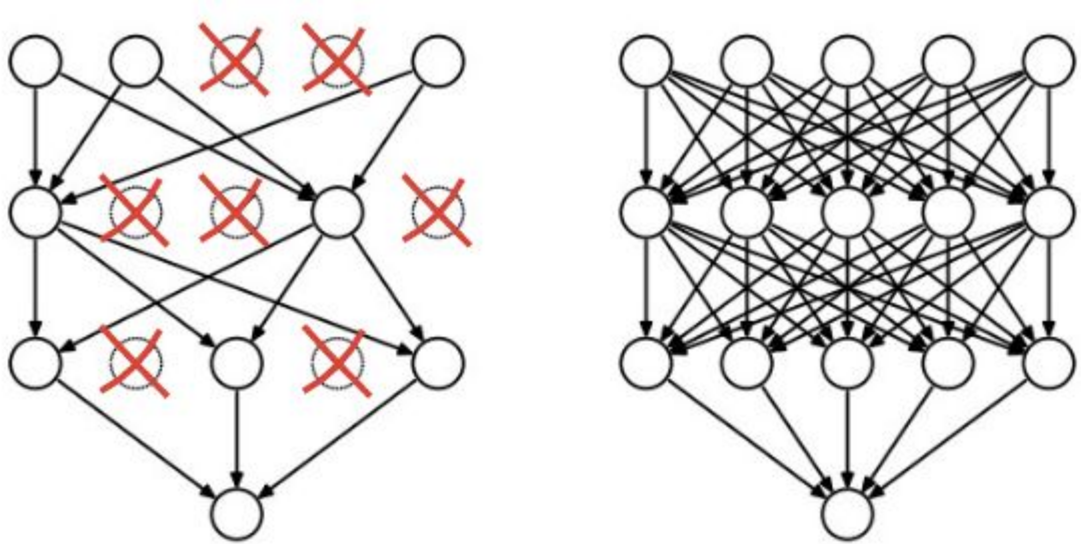
L1 vs. L2 regularization



Dropout

Randomly drop nodes with probability *p*

It is a form of model averaging (averaging a lot of models)

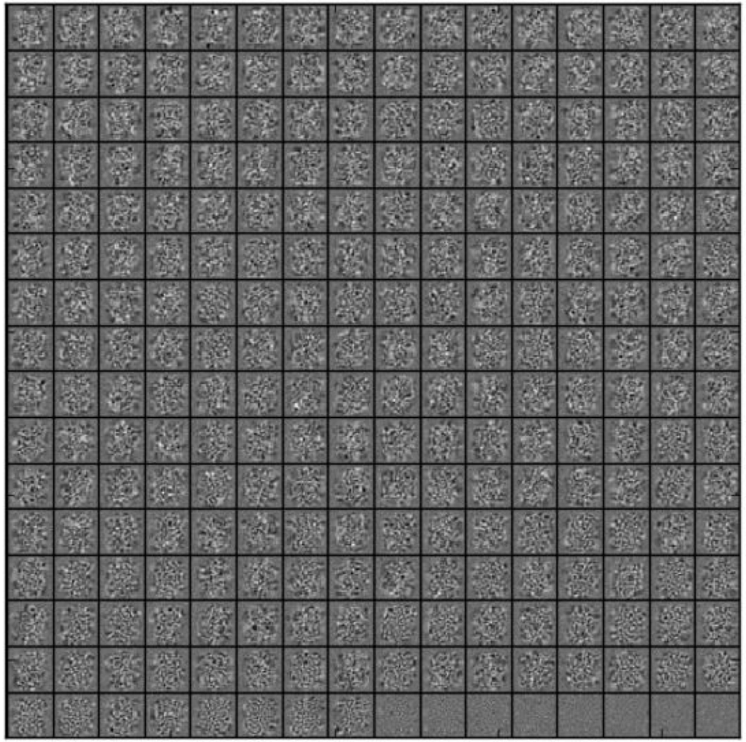


Solutions

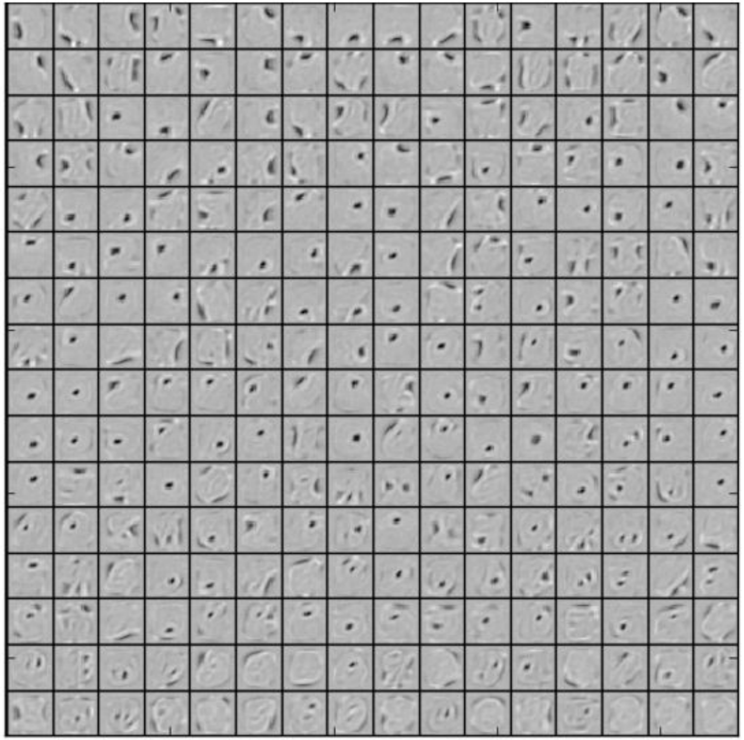
- *Pr* ... sed
  ne
  (R
- *Be*
- *Re*
  -

```
# TF: Deep Neural Network; Dropout
w1 = tf.Variable(tf.random_normal([1, hidden_layer_size]))
b1 = tf.Variable(tf.random_normal([hidden_layer_size]))
w2 = tf.Variable(tf.random_normal([hidden_layer_size, 1]))
b2 = tf.Variable(tf.random_normal([1]))
h = tf.sigmoid( tf.add(tf.matmul(x, w1), b1) )
dropped = tf.nn.dropout(h,0.5)
y_ = tf.add(tf.matmul(h, w2), b2)
# loss function
loss = tf.reduce_mean(tf.square(y - y_))
```

Dropout



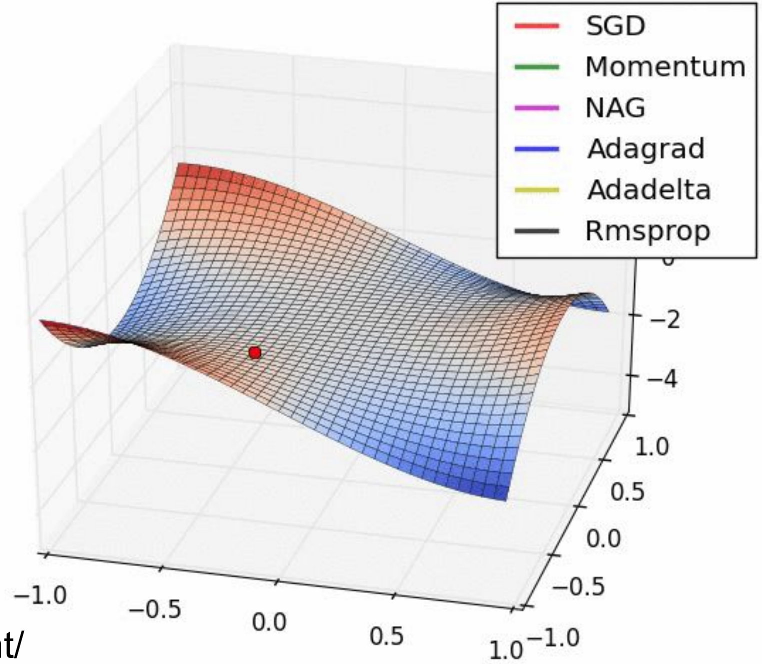
(a) Without dropout



(b) Dropout with $p = 0.5$.

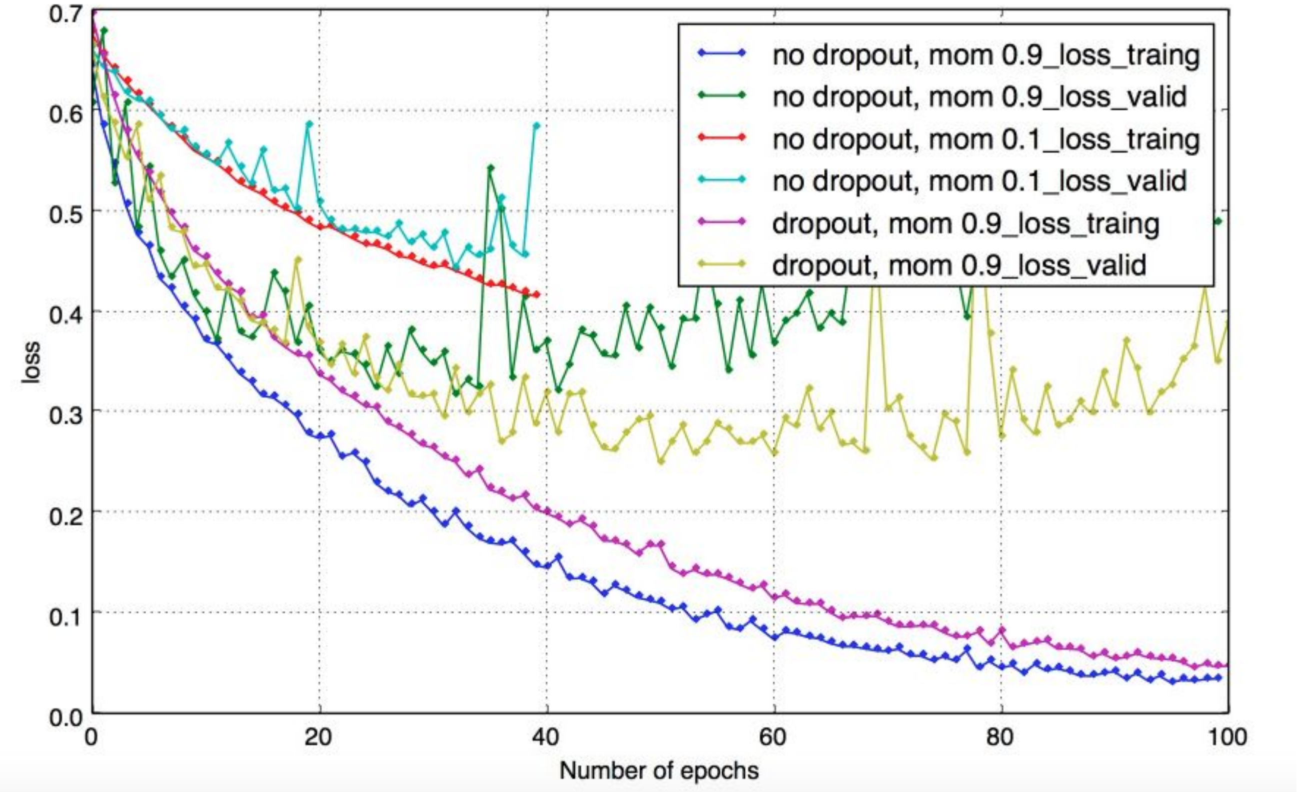
Gradient Descent variants

- Stochastic Gradient Descent (SGD): Compute gradients over a batch of samples and average them.
- Momentum
- Adadelta
- Adagrad
- Rmsprop
- BFGS



http://sebastianruder.com/optimizing-gradient-descent/

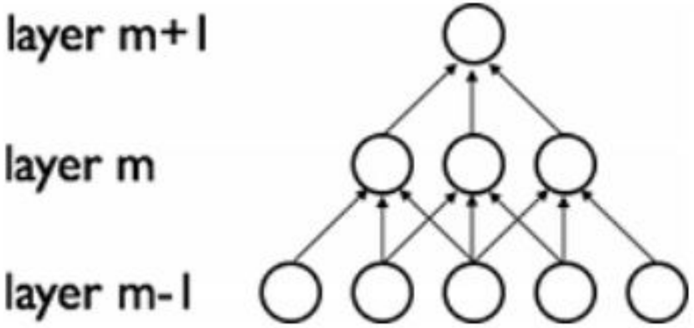
The effect of dropout and momentum



Convolutional Neural Networks (CNNs)

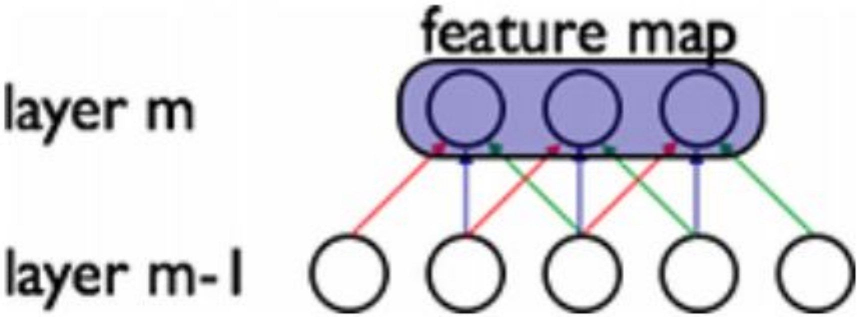
In CNNs, layers have sparse connectivity by design

Better suited for correlated features (Image, Spectrogram, etc)



CNNs

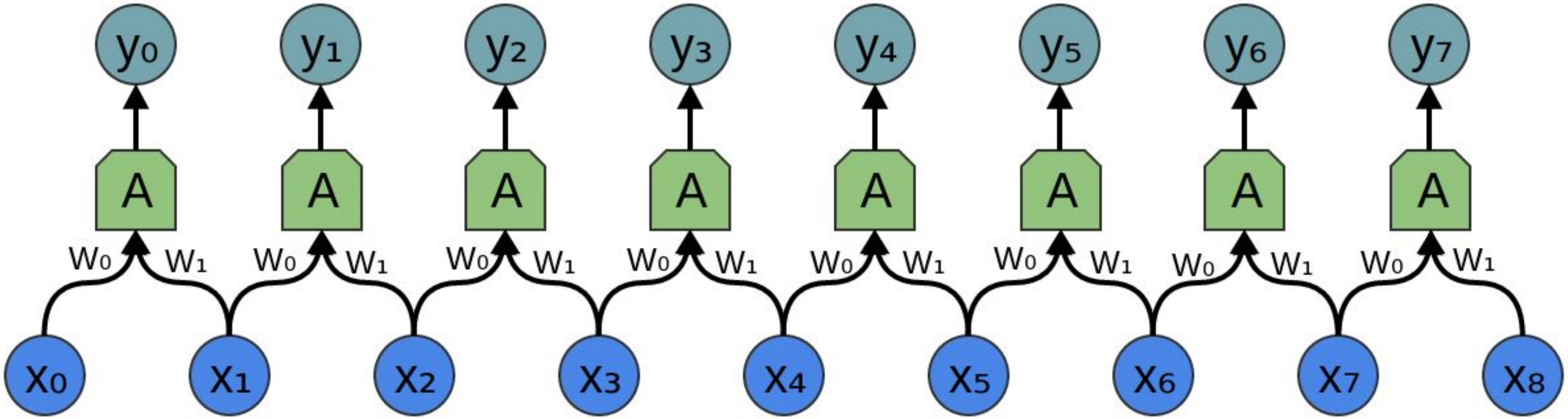
Weight sharing



https://colah.github.io/posts/2014-07-Understanding-Convolutions/

https://colah.github.io/posts/2014-07-Conv-Nets-Modular/

CNNs


y0
y1
y2
y3
y4
y5
y6
y7
A
A
A
A
A
A
A
A
W0
W1
W0
W1
W0
W1
W0
W1
W0
W1
W0
W1
W0
W1
W0
W1
x0
x1
x2
x3
x4
x5
x6
x7
x8

CNNs

Weight matrix computation W.x

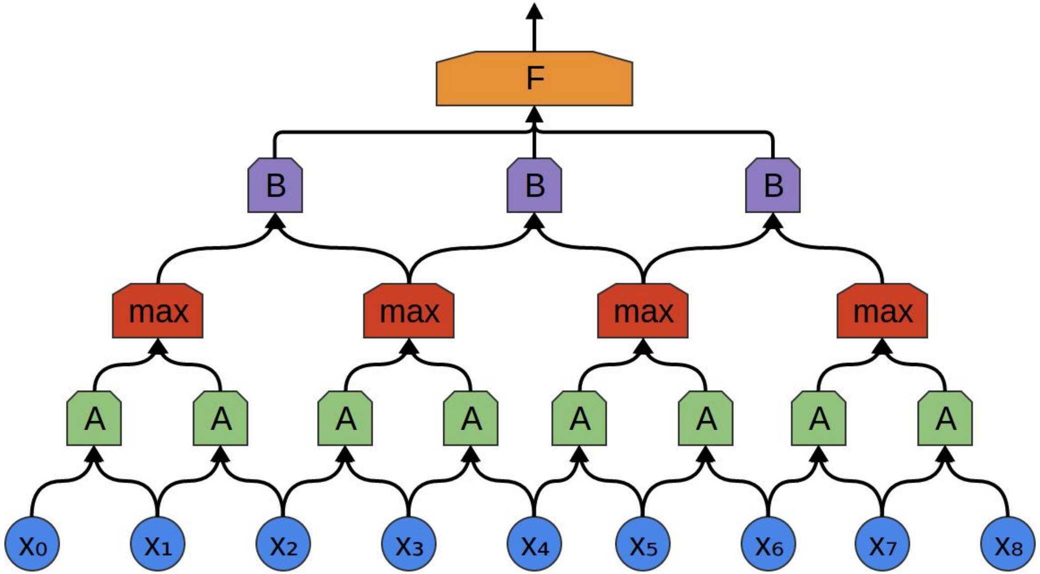
$$W = \begin{bmatrix} W_{0,0} & W_{0,1} & W_{0,2} & W_{0,3} & \dots \\ W_{1,0} & W_{1,1} & W_{1,2} & W_{1,3} & \dots \\ W_{2,0} & W_{2,1} & W_{2,2} & W_{2,3} & \dots \\ W_{3,0} & W_{3,1} & W_{3,2} & W_{3,3} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Regular fully-connected weight matrix

$$W = \begin{bmatrix} w_0 & w_1 & 0 & 0 & \dots \\ 0 & w_0 & w_1 & 0 & \dots \\ 0 & 0 & w_0 & w_1 & \dots \\ 0 & 0 & 0 & w_0 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Convolutional weight matrix

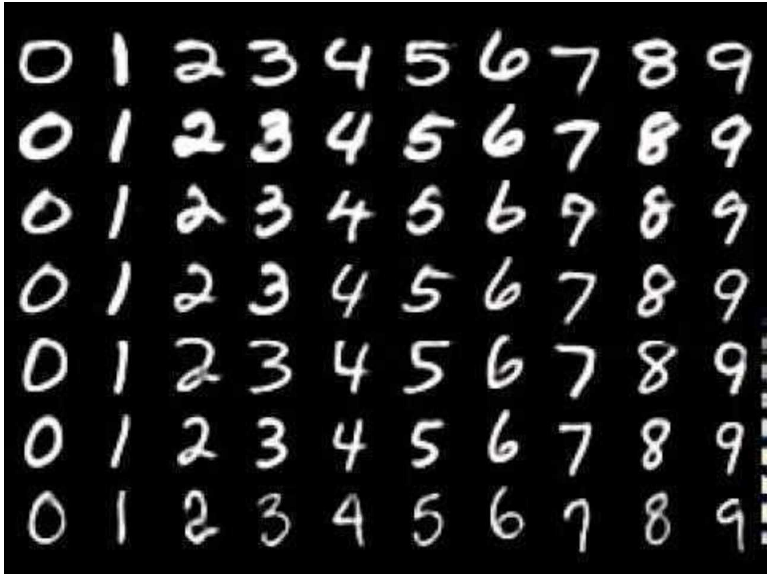
CNNs



CNNs: interactive demo

Demo in Keras:

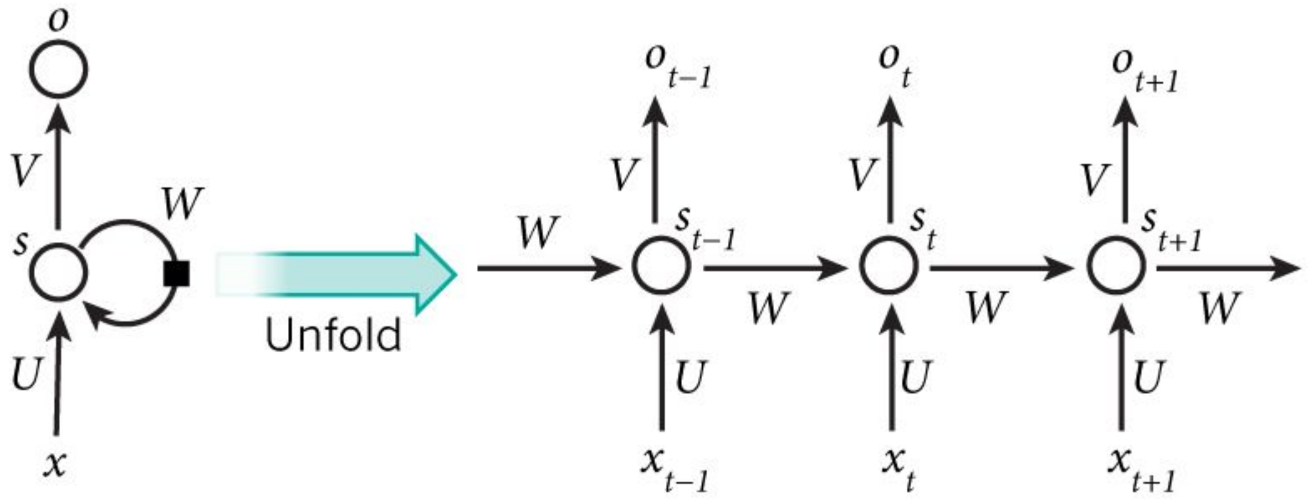
https://colab.research.google.com/drive/1rsjU9s0_JcNzi7DSBDNSl-WwQrqEWQUS#scrollTo=BFiZwlTJoVhW



Recurrent Neural Networks

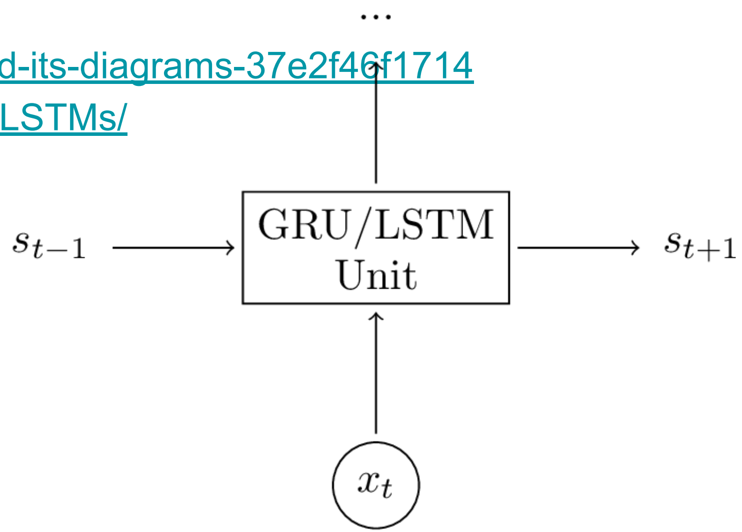
RNN: For modeling sequence where adjacent frames are not independent from each other

Models dynamics by having a state which is computed from the previously seen samples



Gated RNNs

- Long Short Term memory Networks (LSTMs)
- Gated Recurrent Units (GRUs)
- They model next hidden state in a more compact manner
- They are a black box “memory unit”
- Further reading:
    - https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714
    - https://colah.github.io/posts/2015-08-Understanding-LSTMs/

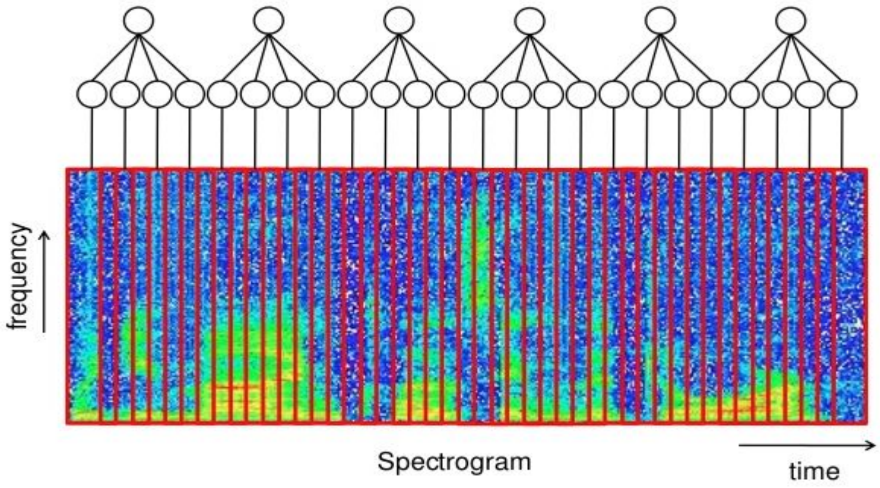


RNNs: Interactive demo

- Sequence classifications using LSTMs
- Input: IMDB review text
- Output: User rating, positive or negative
- Demo in Keras:
    - https://colab.research.google.com/drive/14nioF__2bTxeiOsIXRyAq8Ribj3ic9Eh
    -

CNN/RNNs for Audio

Spectrogram can be treated as image, A recurrent layer models the sequence
- No feature engineering (MFCC computation)
- No adding delta or appending frames to capture context

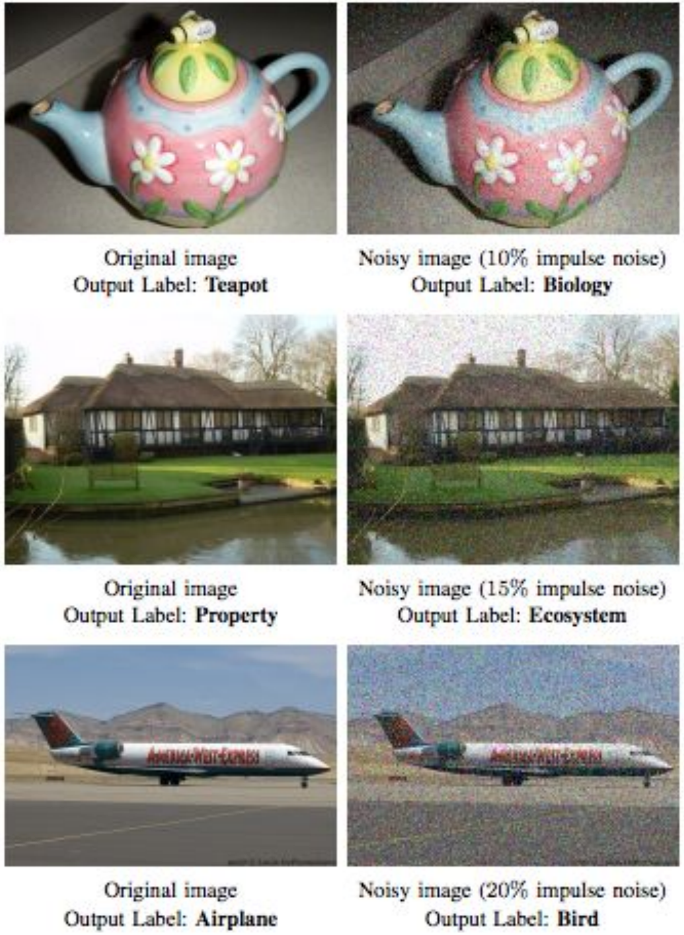


Challenges

The systems are still vulnerable:

- Image recognition tasks are vulnerable to Noise that doesn't affect human perception

- Speech recognition in cars? Not near perfect.



Deceiving Google's Cloud Video Intelligence API Built for Summarizing Videos, 2017.
Google's Cloud Vision API Is Not Robust To Noise, 2017.

Further Reading: Generative Models

- Generative models: definition
- Popular models:
  - Generative Adversarial Networks (GANs):
  - Variational Autoencoders (VAEs):
  - Autoregressive Models:
    - Wavenet for audio generation
    - PixelRNN/PixelCNN for image generation

Further Reading: Miscellaneous

- Batch Normalization: https://arxiv.org/pdf/1502.03167v3.pdf
- Residual/Highway/Dense Nets: https://chatbotslife.com/resnets-highwaynets-and-densenets-oh-my-9bb15918ee32
- Sequential models with Attention: https://distill.pub/2016/augmented-rnns/
- Connectionist Temporal Classification (CTC) for end-to-end Speech Recognition
- Capsule Nets by Prof. Hinton
- Meta Learning
- Bayesian/Probabilistic Neural Networks

ObEN Inc.

Email: hamid@oben.com

Creating Personal Avatars:

- 3D Face/Body/Dance
- Personalized Speech/Singing
- Chatbot

Internship/Full-time positions:
Projects are a combination of:

- Speech Signal Processing
  (With a focus on Speech Synthesis)
- Machine Learning (Deep Learning)

ObEN’s AI generated poem using Deep Learning

Oben all the time

The story of it of mine

I said you wanna be your mind with you

I wanna see you say it let me see you love me

I said you see the stars to the way you want to be all the way

They can't see you say

I see the way you see

All I see that we'll be better and I can see

And I said you want to be all the world