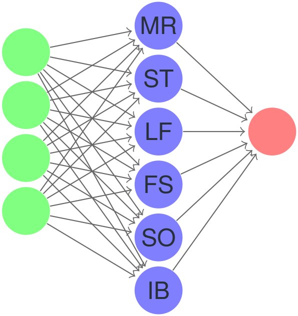
SCAI
SuperComputing Applications and Innovation

Introduction to Deep Learning and Tensorflow

Day 2



CINECA Roma - SCAI Department

Marco **R**orro
Luca **F**erraro
Sergio **O**rlandini
Stefano **T**agliaventi
Francesco **S**alvadore
Isabella **B**accarelli

Rome, 7th-9th Nov 2018

CINECA

Outline

Outline

Deep learning

- Representation learning: extract high-level abstract features from raw data (i.e. pixels composing an image).
- Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations.
- Deep learning allows the computer to build complex concepts out of simpler concepts.
- Deep Learning try to solve the problem of the *curse of dimensionality*.
    - in very high-dimensional spaces, traditional machine learning algorithms fail to generalize.
    - fail to extract complicated function
    - high-computational costs.

Deep Learning

A recently published review[1] can help on summarizing main aspects of deep learning.

1. Models are composed of multiple processing layers:
   - multiple layers of abstraction to learn data representations.
2. Improved state-of-the-art in:
   - speech recognition, object recognition, object detection;
   - drug discovery, genomics.
3. Discovers complex patterns in large datasets:
   - backpropagation to change layer parameters;
   - representation in each layer is based on previous layer results;
4. Specialized networks for different data;
   - deep convolutional networks: image, video, speech;
   - recurrent networks: sequential data (text, speech).

[1] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, **Deep Learning**, Nature 2015

Limits of conventional ML techniques

LeCun Bengio and Hinton stress that:

Limits of conventional ML techniques

LeCun Bengio and Hinton stress that:

- conventional machine-learning techniques were limited in their ability to process natural data in their raw form;

Limits of conventional ML techniques

LeCun Bengio and Hinton stress that:

- conventional machine-learning techniques were limited in their ability to process natural data in their raw form;
- feature extraction is a necessary step for transforming raw data into an internal representation;

Limits of conventional ML techniques

LeCun Bengio and Hinton stress that:

- conventional machine-learning techniques were limited in their ability to process natural data in their raw form;
- feature extraction is a necessary step for transforming raw data into an internal representation;
- considerable domain expertise is needed to pick a representation suitable to the task.

Limits of conventional ML techniques

LeCun Bengio and Hinton stress that:

- conventional machine-learning techniques were limited in their ability to process natural data in their raw form;
- feature extraction is a necessary step for transforming raw data into an internal representation;
- considerable domain expertise is needed to pick a representation suitable to the task.

On the other side, they consider deep learning methods as representation-learning methods.

"Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representation needed for detection or classification"

Outline

1. Deep Learning
   - Deep Learning: a review
   - Representation Learning methods
   - DL Applications
   - Layers and Features
   - Convolutional Networks
   - Binary Classification Example
2. Deep topics
   - Norm Penalties
   - Early stopping
   - Dropout
   - Batch Normalization

Neural networks as representation learning methods

Data flow:

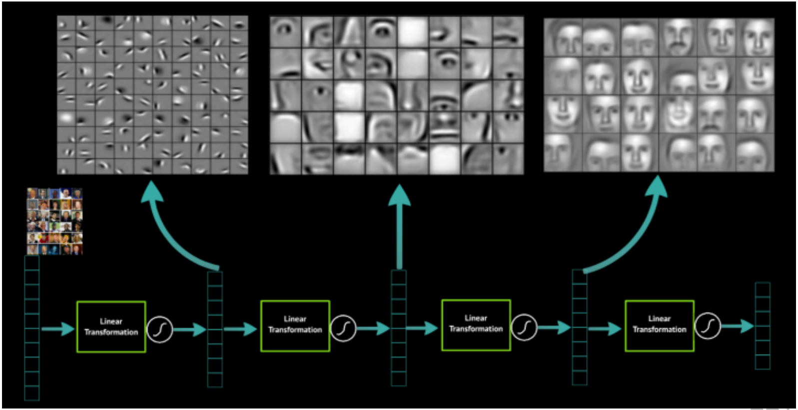
- input: raw data;
- output: detection/classification distribution probabilities;
- in the process: a layer is fed with data representation learned from previous layer.

Key aspects:

- no a-priori design of features;
- they are learned from data using a general purpose procedure.

Image Example

edges motifs familiar objects



Outline

Deep learning main results (I)

- Good at discovering intricate structures in high-dimensional data.
- Exhibits superior performances (compared to other ML techniques):
  - image and speech recognition;
  - prediction of the activity of potential drug molecules;
  - analyzing particle accelerator data;
  - reconstructing brain circuits;
  - predicting the effects of mutations in non-coding DNA on gene expression and disease.
- Shows promising results in natural language processing (NLP):
  - topic classification, sentiment analysis, question answering and language translation.

Deep learning main results (II)

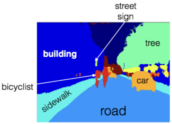
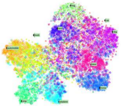
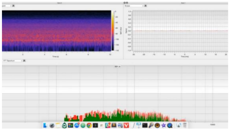


Image recognition

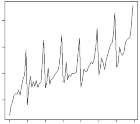


Natural language processing



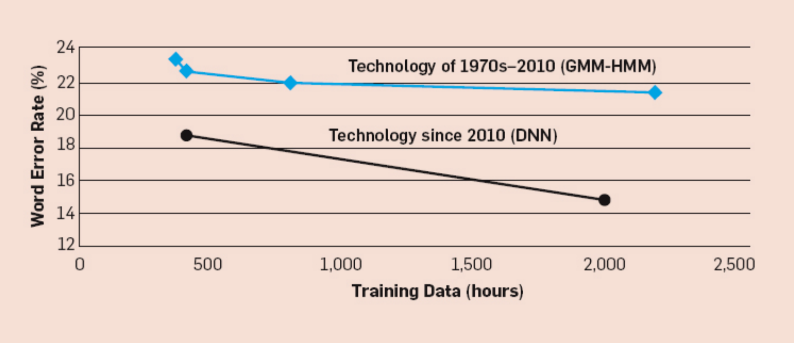
Speech recognition

Video activity detection



Tabular and time-series data applications

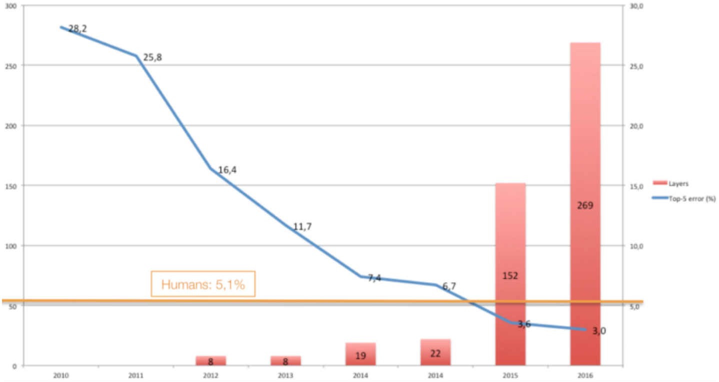
Accuracy on Speech Recognition



Source: Huang, Baker, Reddy, **A Historical Perspective of Speech Recognition**, Communications of the ACM, January 2014

GMM: Gaussian Mixture Models, **HMM:** Hidden Markov Models, **DNN:** Deep Neural Networks

How deep is deep learning?



Number of layers in ILSVRC (ImageNet Large Scale Visual Recognition Competion) winners, compared to accuracy.

Outline

How deep learning works?

In the following, we will see:

- the effect of adding a fully connected layer to an existing classifier;
- the effect of describing our data in a "wider" hyperspace.

How deep learning works?

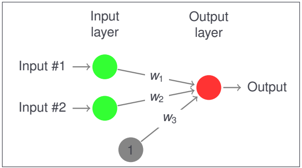
In the following, we will see:

- the effect of adding a fully connected layer to an existing classifier;
- the effect of describing our data in a "wider" hyperspace.

Idea from a blog post: Olah, **Neural Networks, Manifolds, and Topology**: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

2d example (I)

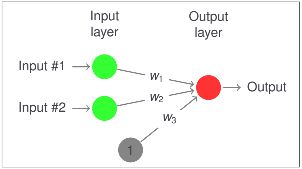
Define a simple network:



$$o_i = < [x_i \; y_i], [w_1 \; w_2] > + w_3$$

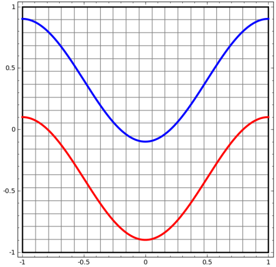
2d example (I)

Define a simple network:



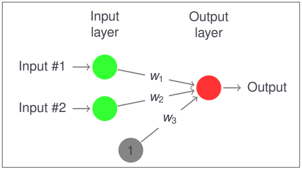
$$o_i = < [x_i \; y_i] , [w_1 \; w_2] > + w_3$$

Labeled observations: $\forall i \; (x_i, y_i) \rightarrow l_i$



2d example (I)

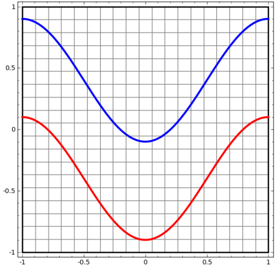
Define a simple network:



$$o_i = < [x_i \ y_i], [w_1 \ w_2] > + w_3$$

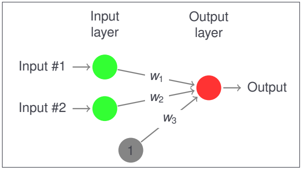
Labeled observations: $\forall i \ (x_i, y_i) \rightarrow l_i$

optimize: $w = \arg\min \sum_i (l_i - o_i)^2$



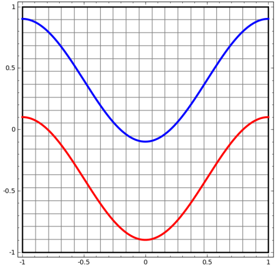
2d example (I)

Define a simple network:

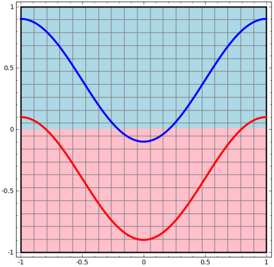


$$o_i = < [x_i \;\; y_i]\,,[w_1 \;\; w_2] > + w_3$$

Labeled observations: $\forall i \; (x_i, y_i) \to l_i$

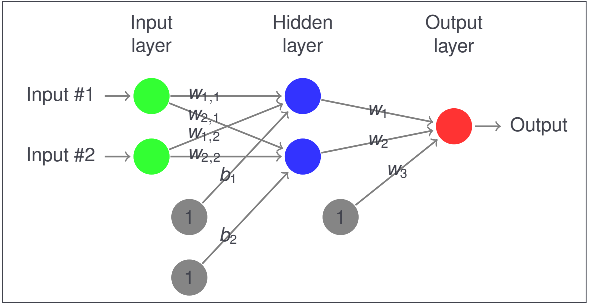


optimize: $w = \arg\min \sum_i (l_i - o_i)^2$



2d example (II)

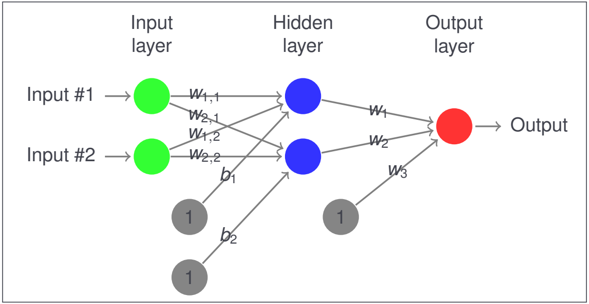
Add an hidden layer:



$$o_i = < f\left([x_i \ \ y_i] \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}^T \right), [w_1 \ \ w_2] > + w_3$$

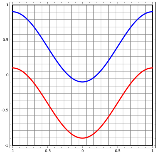
2d example (II)

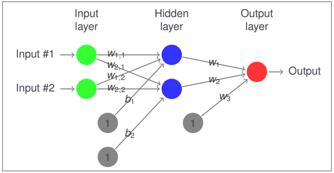
Add an hidden layer:

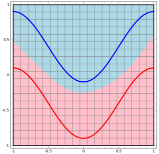


$$o_i = < \boxed{f\left([x_i \; y_i] \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}^T \right)}, [w_1 \; w_2] > + w_3$$

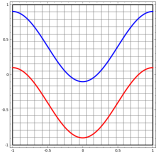
Hidden layer: evaluated features

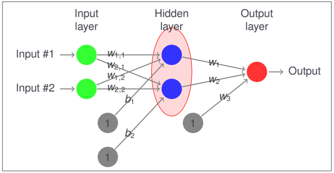


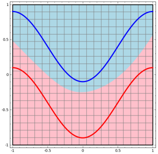


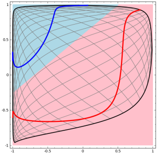


Hidden layer: evaluated features

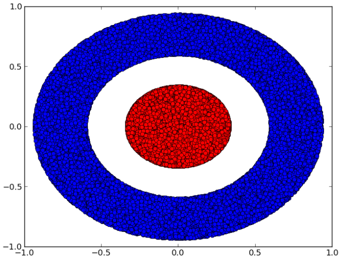






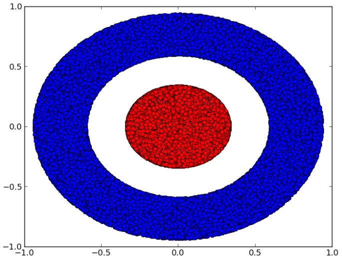


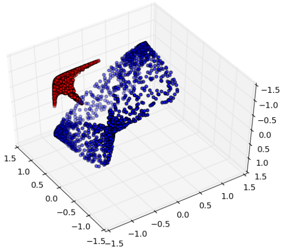
Increase the dimensionality



- It is impossible for a neural network to classify this dataset without having a layer that has 3 or more hidden units, regardless of depth
- Even if it can get an 80% of classification accuracy

Increase the dimensionality





- It is impossible for a neural network to classify this dataset without having a layer that has 3 or more hidden units, regardless of depth
- Even if it can get an 80% of classification accuracy

Neural Network Internals: classification problem

- Given a single input, a trained neural network is able to predict a distribution of probability:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W})$$

prediction input NN parameters

Neural Network Internals: classification problem

- Given a single input, a trained neural network is able to predict a distribution of probability:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W})$$

- NN paramaters are chosen in order to minimize the average error on a given training set $\left\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right\}_{i=1,\dots,N}$:

$$J(\boldsymbol{W}) = \frac{1}{N}\sum_{1}^{N} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{W}), \boldsymbol{y}^{(i)}\right)$$

Neural Network Internals: classification problem

- Given a single input, a trained neural network is able to predict a distribution of probability:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W})$$

- NN paramaters are chosen in order to minimize the average error on a given training set $\left\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right\}_{i=1,\dots,N}$:

$$J(\boldsymbol{W}) = \frac{1}{N}\sum_{1}^{N} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{W}), \boldsymbol{y}^{(i)}\right)$$

- a Stochastic Gradient Descent (SGD) algorithm step is:

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} - \epsilon_k \hat{\boldsymbol{g}}^{(k)} \qquad \text{where} \qquad \hat{\boldsymbol{g}}^{(k)} = \frac{1}{n}\nabla_W \sum_{l \in \text{batch}} L(f(\boldsymbol{x}^{(l)}; \boldsymbol{W}^{(k)}), \boldsymbol{y}^{(l)}),$$

Backpropagation

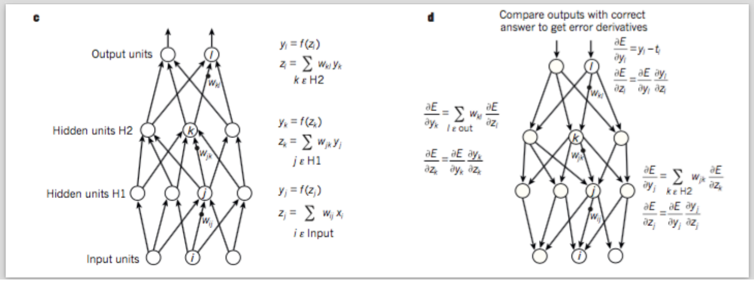
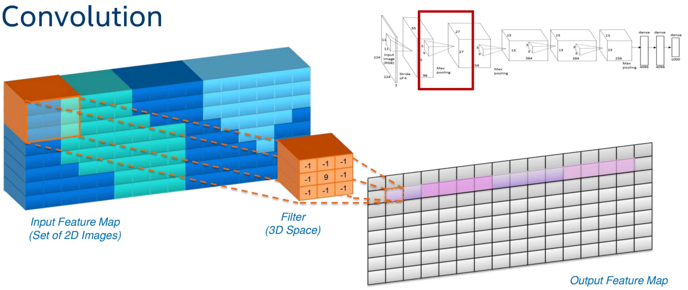


Image from Yann LeCun, Yoshua Bengio, Geoffrey Hinton, **Deep Learning**, Nature 2015.

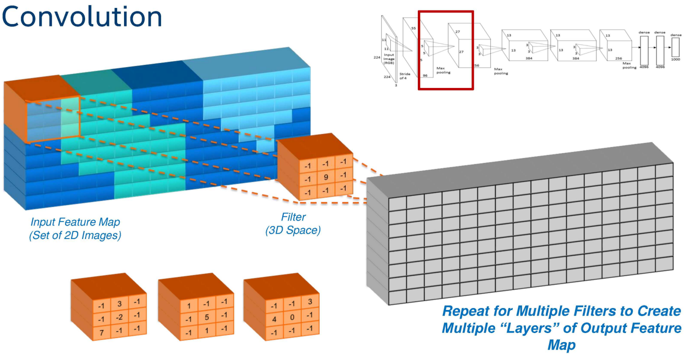
See also:
https://google-developers.appspot.com/machine-learning/crash-course/backpro

Outline

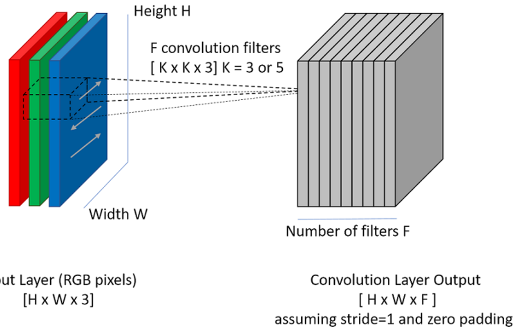
Convolutional layer (I)



Convolutional layer (I)

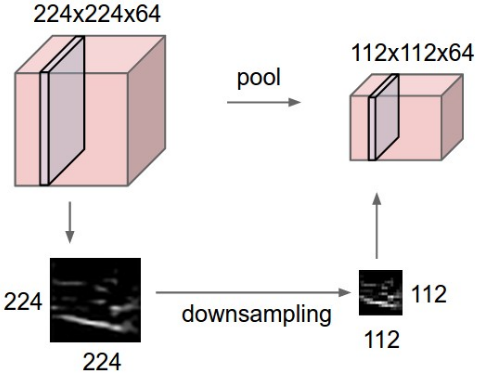


Convolutional layer (II)



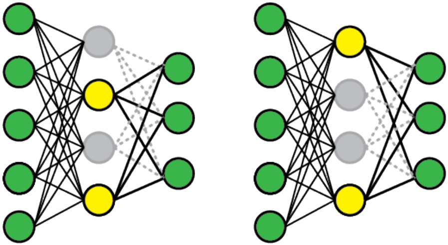
- Convolution Layer output: $H - \frac{K-1}{2}$, $W - \frac{K-1}{2}$ with $stride = 1$ and without padding
- For each filter we have $K \times K \times 3$ weights
- The filter convolves over all spatial locations, producing a scalar for each location
- An activation function is finally applied

Pooling layer



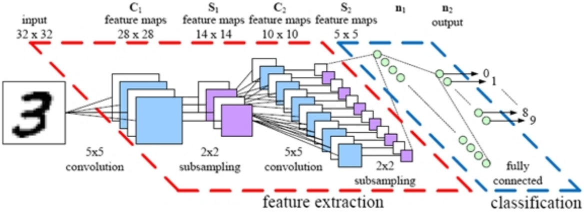
- makes the representations smaller and more manageable
- operates over each activation map independently
- Example: max pooling computes the maximum with $K \times K$ filter

Dropout layer



- In each forward pass, randomly set some neurons to zero.
- Probability of dropping is a hyperparameter; 0.5 is common

Convolutional network



- LeNet[2] . The first successful convolutional neural network
- Designed to identify hand-written digits in the MNIST dataset
- LeNet-5 takes a single-channel 2D input
- Performs 6 convolution (5×5) , then subsamples by max-pooling (2×2).
- The convolution-pooling layer sequence occurs again
- Finally 2 fully connected layer followed by a fully connected softmax layer is performed

[2] Yann Lecun and LÃ©on Bottou and Yoshua Bengio and Patrick Haffner, **Gradient-based learning applied to document recognition**, 1998

CNN comparison

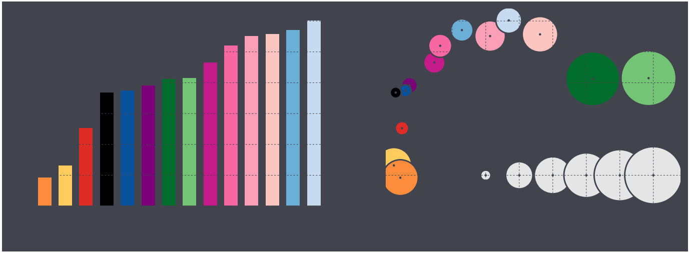


Figure from: Canziani, Alfredo; Paszke, Adam; Culurciello, Eugenio; An Analysis of Deep Neural Network Models for Practical Applications, 2017

- Inception-v4: Resnet + Inception
- VGG High memory and operations
- GoogLeNet very efficient
- Alexnet few operations but high memory and low accuracy
- Resnet moderate efficency and high accuracy

Links and credits

- Fei-Fei Li, Justin Johnson, Serena Yeung - CS231n
- https://developers.google.com/machine-learning/crash-course/
- https://eu.udacity.com/course/deep-learning–ud730
- https://www.kaggle.com/competitions

Outline

Binary Classification Example

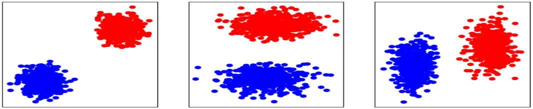
The goal of the example is to feedforward network to solve a binary classification problem

- **Classification** is the process in which ideas and objects are recognized
- If only 2 groups are involved the classification is a **Binary Classification**
- The imaginary lines that separate the groups are called **Decision Boundaries**
- The aim of the classification is to learn the decision function from a set of labeled samples.
  This set of samples is called **Training Data**

Binary Classification Example

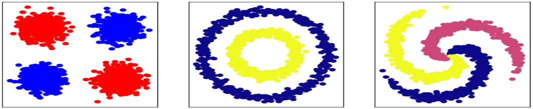
Linearly separable

- Data sets are linearly separable if exists at least one line that separate all the points of a type on one side of the line and all the points of the other type on the other side.
  A linear boundary (e.g. a straight line) is enough to separate the data into 2 groups



Non-Linearly separable

- Data sets that are not linearly separable.
  The decision boundary is not a linear boundary but a polygonal or a circular line



Binary Classification Example

- **Input layer**
  supply the input data or features to the network
- **Output layer**
  give out predictions
- **Hidden layers**
  apply a series of functions to the input
  NB: With multiple hidden layer we can compute complex functions by cascading simpler functions

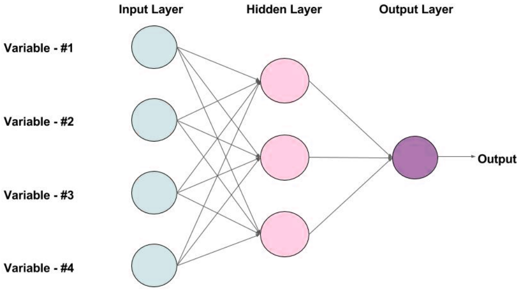
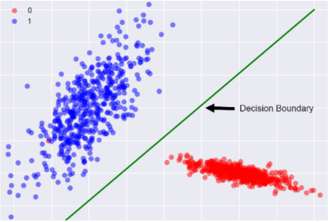


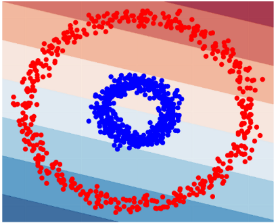
Figure: Feed-forward network with one hidden-layer and 3 neurons

Why use hidden layer in Binary Classification ?

If we try to solve the binary classification problem without hidden layers we are using a simple neuron which is able of learning only a **linear decision boundary**.

In case of linearly separable data a single neuron is able to learn. But in case of non-linearly separable data we can use feature transformations as a model for neuron learning.





Exercise **Reminder**:

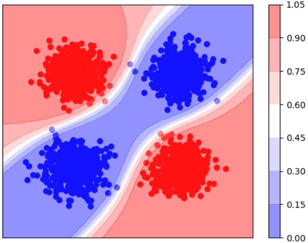
- A single neuron can only learn a linear decision boundary
- For non linear datasets feature transformations (like product of features or square of them) have to be involved.
  This approach is not general and can be tricky for data not easy to visualize

Why use hidden layer in Binary Classification ?

A neural network with a single hidden layer with nonlinear activation functions is a "**Universal Function Approximator**" (i.e. capable of learning any function)

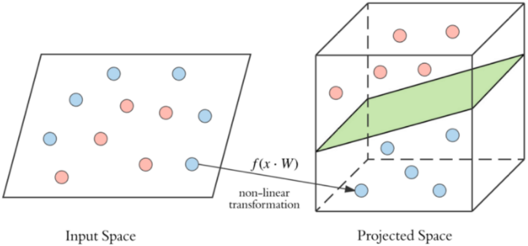
The universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n (under mild assumptions on the activation function)

A feedforward network is a powerful deep learning model as a universal function approximator able to model any complex function

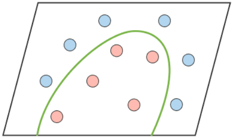


Why use hidden layer in Binary Classification ?

By performing **non-linear transformations** by layer the input space is projected into a new vector space. It is projected into a higher dimensional space where it becomes linearly separable.



This is equivalent to learn a complex decision boundary in the original input space.



Outline

Outline

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Many strategies: restriction on parameters, on training algorithm, extra terms in objective function and more.

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Many strategies: restriction on parameters, on training algorithm, extra terms in objective function and more.
- In practical complex scenarios, as image or waveform analysis, the search space is huge.

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Many strategies: restriction on parameters, on training algorithm, extra terms in objective function and more.
- In practical complex scenarios, as image or waveform analysis, the search space is huge.
- Very often the optimal choice is not find the minimum size of the space generating the data.

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Many strategies: restriction on parameters, on training algorithm, extra terms in objective function and more.
- In practical complex scenarios, as image or waveform analysis, the search space is huge.
- Very often the optimal choice is not find the minimum size of the space generating the data.
- Optimal solution: bigger spaces with regularization.

Norm Penalties

- Adding a term to the objective function.

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \Omega(\boldsymbol{\theta})$$

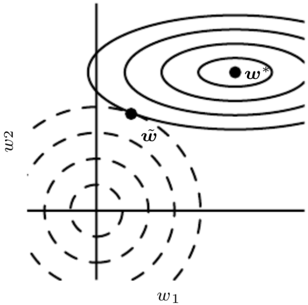
- $\alpha \in [0, \infty)$ is an hyperparameter controlling the weight of the regularization.
- With neural networks, typically only weights are regularized, not biases.

L^2 norm regularizer

- Weight decay.
- Tikhonov regularization.

$$\Omega(\boldsymbol{w}) = \frac{1}{2}||\boldsymbol{w}||_2^2$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\frac{1}{2}||\boldsymbol{w}||_2^2$$

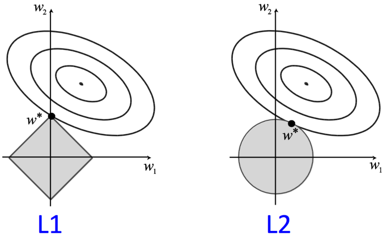


- Regularizer acts on slow changing directions
- Encourage small weights, mostly on "unimportant weights"

L^1 norm regularizer I

$$\Omega(\boldsymbol{w}) = ||\boldsymbol{w}||_1$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha||\boldsymbol{w}||_1$$



- Encourages sparsity
- Moves small weight directions towards zero.
- Enhance directions of big weight values.
- A feature selection mechanism.

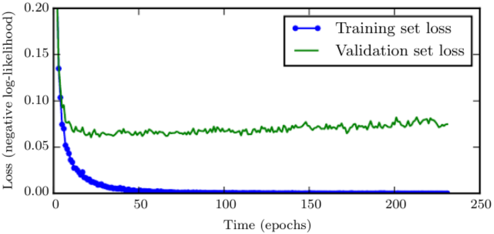
L^1 norm regularizer II

- With tensorflow:

```
import tensorflow as tf
import numpy as np

total_loss = tf.losses.mean_squared_error( labels, predictions)
l1_regularizer = tf.contrib.layers.l1_regularizer(scale=0.005, scope=None)

weights = tf.trainable_variables() # all vars of your graph
regularization_penalty = tf.contrib.layers.apply_regularization(l1_regularizer, weights)

regularized_loss = total_loss + regularization_penalty # this loss needs to be minimized
train_step = tf.train.GradientDescentOptimizer(0.05).minimize(regularized_loss)
```

Early stopping I

- With large overfittig models, often validation error begin to rise after decreasing, while training error decrease.



- Obvious solution: get the model parameters at the minimum of the validation error.
- Simple and efficient.
- Store a copy of the parameters and update it with the best one.
- Possible hyperparameters: begin-end iteration evaluation, threshold, evaluation step.
- It reduces the parameter space.

Early stopping II

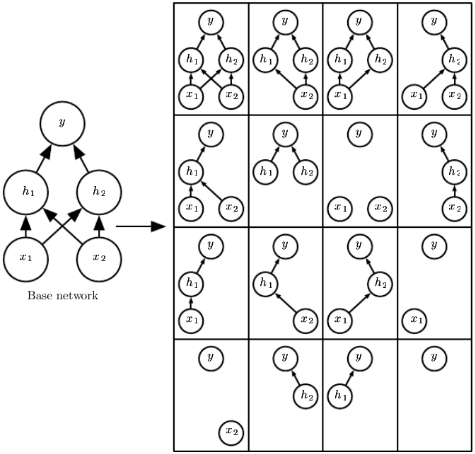
- Using tensorflow basics, for example, take the test error at the end of a epoch.
- Using tensorflow **Estimators** with **`make_early_stopping_hook`**
- Using Keras with

```
keras.callbacks.EarlyStopping(monitor='val_loss',          #quantity to be monitored
                              min_delta=0,                 #minimum change in to qualify
                                                           #    as an improvement
                              patience=0,                  #number of epochs with no
                                                           #    improvement after which
                                                           #    training will be stopped.
                              verbose=0,
                              mode='auto',                 #{auto, min, max}. stop when
                                                           #    the quantity monitored
                                                           #    has stopped decreasing
                              baseline=None,               #Value to reach.Training will
                                                           #    stop if the model doesn't
                                                           #    show improvement over
                                                           #    the baseline.
                              restore_best_weights=False)
```

Dropout I

- Computationally inexpensive.
- Powerfull on many family models.
- It can be thought as model selection over a family of models.
- Dropout trains the ensemble consisting of all subnetworks that can be formed by removing non output units from an underling base network.

Dropout II



- It can be defined as a mask that select nodes to be included in the training process. For example by multiplying the units(the outputs of nodes) by the mask.

Dropout III

- Nodes in the mask are sampled independently.
- Hyperparameter: sampling probability:
  - usually 0.5 for hidden units, 0.8 for inputs
- It also can be viewed as a training procedure of an ensemble of models that share hidden units.
  - Hidden units must be able to perform well regardless of which other hidden units are in the model.
  - Hinton et al.(2012):ideas form sexual reproduction,which involves swapping genes between two different organisms, creates evolutionary pressure for genes to become not just good but readily swapped between organism. This produces genes robust to environment changes not follow unusual features of the different organisms.
- with tensorflow, simply add a dropout layer with the function **`tf.nn.dropout`**
- For example:

```
layer_1 = tf.nn.relu(tf.add(tf.matmul(x, weights_hidden), biases_hidden))
# apply DropOut to hidden layer
drop_out = tf.nn.dropout(layer_1, 0.6 )  #  60% of units will be active
out_layer = tf.matmul(drop_out, weights_out) + biases_out
```

Batch Normalization I

- Gradient descent update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \boldsymbol{g}$ is true as a first order approximation.
- Parameters update are made independently algong each direction, i.e simultaneously.
- What if weights are > 1 ? It is no more valid.
- Especially in deep networks where many weights are involved in computation of higher orders.
- A net effect is the the loss could not decrease at all, since updates to latest layers are done according to previous state of the previous layers.
- Renormalize hidden inputs in order to have small changing weights, in order to have a ditribution with zero mean and unit standard deviation.
- If $\boldsymbol{H}$ is a desing matrix of a minibatch every row corresponds to the inputs related to an example of the minibatch
- Hidden units of a layer, at minibatch step, are the update according to:

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

- $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of each unit

Batch Normalization II

- The common form of the normalization involves the change:

$$\gamma \boldsymbol{H}' + \beta$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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

- γ andβ are hyperparameters

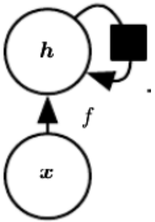
Outline

Recurrent Neural Network

- Well-suited to classifying, processing and making predictions based on time series data
  - Speech recognition
  - Image captioning
  - Time series prediction
  - Music composition
  - Machine translation
- The basic idea is the use of parameters sharing across different parts of a model.
- They are networks with loops in them, allowing information to persist.
- A convenient way of think RNN is using computational graphs, i.e. a way to formalize the structure of set of computations.

RNN Computational Graph I

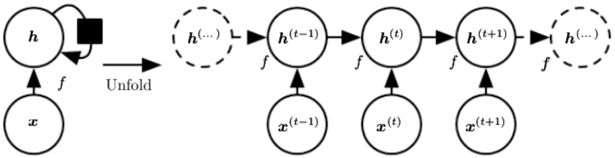
- Suppose that the network is operating a sequence x_t with t ranging form 1 to τ



- It is a piece of a neural network and *h* is the output.
- The loop allows the information to be passed from one step of the network to the next.
- A recurrent neural network can be thought as multiple copies of the same network, each passing a message to a successor.

RNN Computational Graph II

- The loop graph can be **unfolded** to a graph that has a repetitive structure.

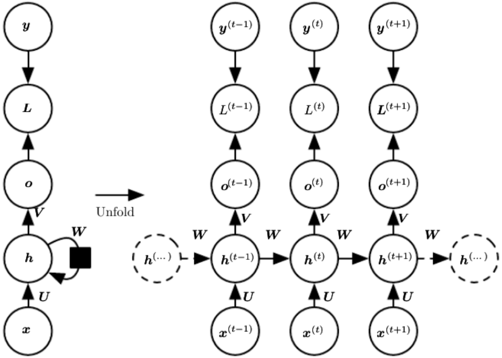


- in general the hidden units can be represented as:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{(t-1)}, \boldsymbol{x}_t; \boldsymbol{\theta})$$

- The gradient is computed with the back-propagation algorithm
- Think of unfolded network

Feed forward RNN I



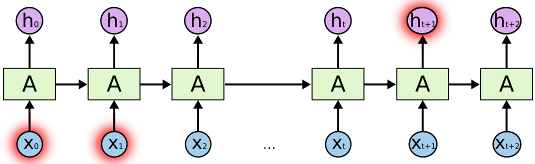
- ***U*** is the weigth matrix of input-to-hidden connections
- ***W*** is the weigth matrix of hidden-to-hidden connections
- ***V*** is the weigth matrix of hidden-to-output connections

Feed forward RNN II

- The forward propagation for each step from $t = 1$ to $t = \tau$:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$
$$\hat{y}^{(t)} = softmax(\boldsymbol{o}^{(t)})$$

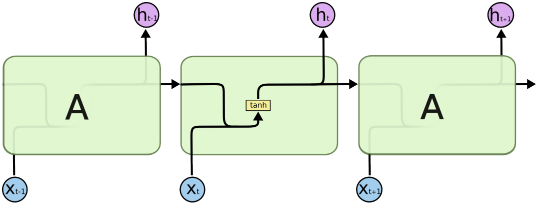
- The BPTT (back-propagation through time) runtime is $O(\tau)$
- RNN naive suffers from long-term dependency problem.
- As gap grows RNN is not able to connect informations.



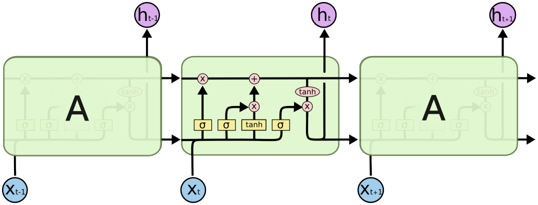
- One solution: Long-Short-Term-Memory networks

LSTM

- It is a special case of RNN
- Designed to remember information for long periods of time.
- A simple RNN is a repetition of simple tanh layers

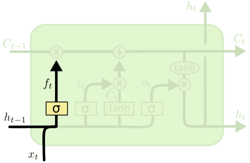


- LSTM uses a more complex block to repeat



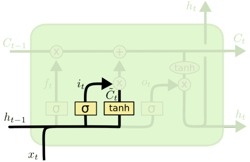
LSTM I

1. First step: "forget gate layer" , a sigmoid function to decide whether to keep(1) or discard(0) the information



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

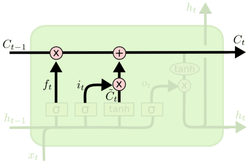
2. A sigmoid layer called the "input gate layer" decides which values we'll update. Then a tanh layer creates a vector of new candidates $\tilde{C}_t$ that could be added to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

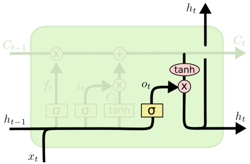
LSTM II

3. Now update the cell state from C_{t-1} to C_t
4. Multiply the old state by f_t and adding the new term $i * \hat{C}_t$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

5. Finally the output: a sigmoid layer to select wihch part of the cell state to output multiplied by a tanh((-1,1) range) to select only the part we are going to output.



$$o_t = \sigma \left(W_o \; [h_{t-1}, x_t] \; + \; b_o \right)$$
$$h_t = o_t * \tanh \left(C_t \right)$$

[1] From `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Attention

- Next step ?

Attention

- Next step ?
- The Attention mechanism

Attention

- Next step ?
- The Attention mechanism
- More on this, the next course.