Introduction to Deep Learning

Samir Kanaan Izquierdo

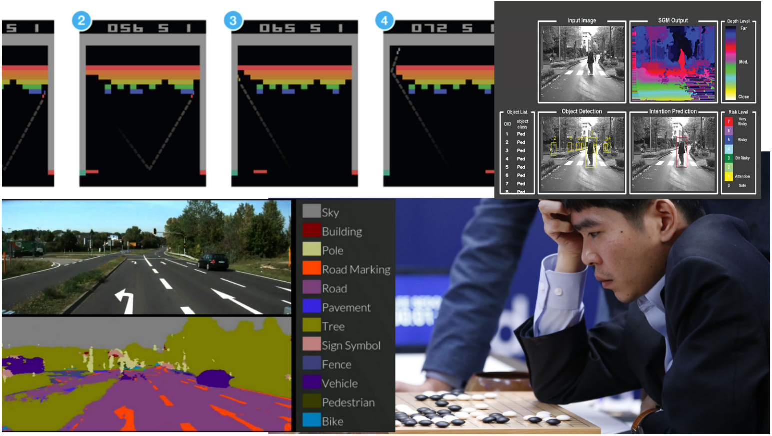
Bioinformatics and Biomedical Signals Laboratory (B2SLAB)
Centre de Recerca en Enginyeria Biomèdica (CREB)
Universitat Politècnica de Catalunya (UPC)

January 2017

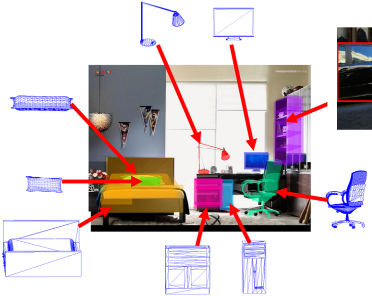
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Index of this presentation

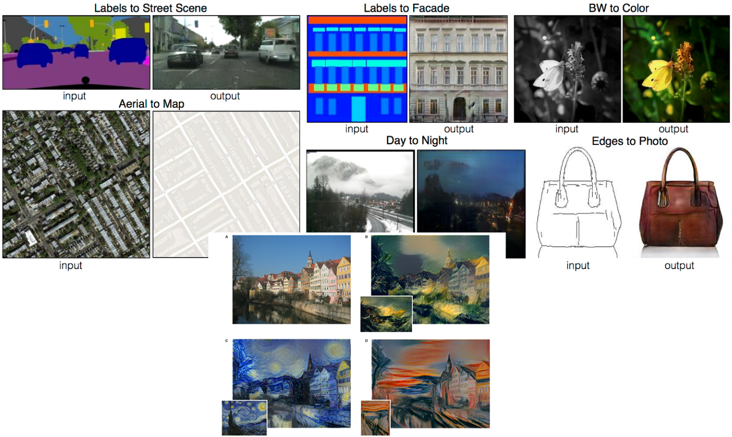
Deep Learning: A Breakthrough in AI?



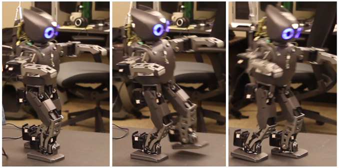
Deep Learning: A Breakthrough in AI?

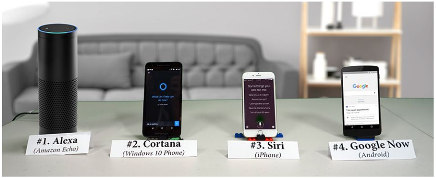


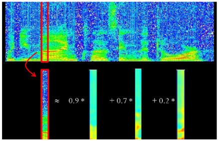
Deep Learning: A Breakthrough in AI?



Deep Learning: A Breakthrough in AI?







Deep Learning: A Breakthrough in AI?

Contents lists available at ScienceDirect

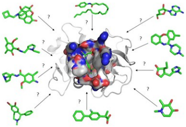
Information Sciences

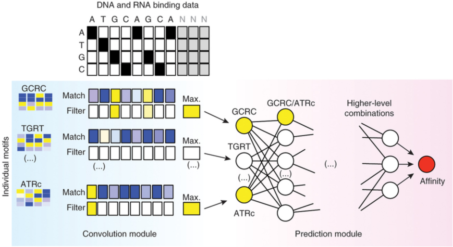
journal homepage: www.elsevier.com/locate/ins

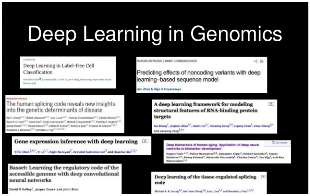
Deep learning approach for active classification of electrocardiogram signals

M.M. Al Rahhal[a], Yakoub Bazi[a,*], Haikel AlHichri[a], Naif Alajlan[a], Farid Melgani[b], R.R. Yager[c,1]

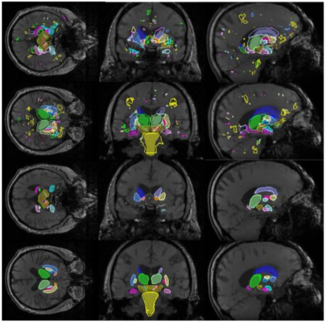
[a] ALISR Laboratory, College of Computer and Information Sciences, King Saud University, P. O. Box 51178, Riyadh 11543, Saudi Arabia
[b] Department of Information Engineering and Computer Science, University of Trento, Via Sommarive, 14, I-38123 Trento, Italy
[c] Machine Intelligence Institute, Iona College, New Rochelle, NY 10801, USA

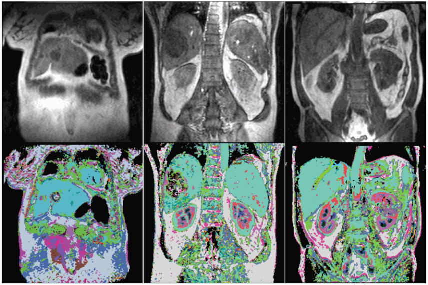






Deep Learning: A Breakthrough in AI?





(a) (b) (c)

Machine Learning (ML) very summarized

A diagram of what AI/ML does after all

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...
- Labeled or not?

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...
- Labeled or not?
  - Yes: Supervised Learning (SL)

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...
- Labeled or not?
  - Yes: Supervised Learning (SL)
  - No: Unsupervised Learning (UL)

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...
- Labeled or not?
  - Yes: Supervised Learning (SL)
  - No: Unsupervised Learning (UL)
  - Just a bit: Semisupervised Learning (SSL)

How's the input data of a ML system?

- Data types: attributes, signal/sound, image (2D/3D), video...
- Labeled or not?
  - Yes: Supervised Learning (SL)
  - No: Unsupervised Learning (UL)
  - Just a bit: Semisupervised Learning (SSL)
  - What's that?: Reinforcement Learning (RL)

Input data processing

1. Data normalization ✓

Input data processing

1. Data normalization ✓
2. Extract useful/well known features :S

Input data processing

1. Data normalization ✓
2. Extract useful/well known features :S
   - Image: moments, SIFT, HOG, SURF...
   - Signal: Fourier, Wavelets...
   - Text: bag of words, TfIdf vectors...
   - Genomics: differential expression tests, SNP association tests

Input data processing

1. Data normalization ✓
2. Extract useful/well known features :S
   - Image: moments, SIFT, HOG, SURF...
   - Signal: Fourier, Wavelets...
   - Text: bag of words, TfIdf vectors...
   - Genomics: differential expression tests, SNP association tests
3. **Problem**: losing potentially useful information

Input data processing

1. Data normalization ✓
2. Extract useful/well known features :S
   - Image: moments, SIFT, HOG, SURF...
   - Signal: Fourier, Wavelets...
   - Text: bag of words, TfIdf vectors...
   - Genomics: differential expression tests, SNP association tests
3. **Problem**: losing potentially useful information
   - Solution 1: use several features (multi-view / multi-feature)

Input data processing

1. Data normalization ✓
2. Extract useful/well known features :S
   - Image: moments, SIFT, HOG, SURF...
   - Signal: Fourier, Wavelets...
   - Text: bag of words, TfIdf vectors...
   - Genomics: differential expression tests, SNP association tests
3. **Problem**: losing potentially useful information
   - Solution 1: use several features (multi-view / multi-feature)
   - Solution 2: use the whole data (*sensory* data)

Desired output: ML tasks

- SL: classification, regression...

Desired output: ML tasks

- SL: classification, regression...
- UL: dimensionality reduction, clustering...

Desired output: ML tasks

- SL: classification, regression...
- UL: dimensionality reduction, clustering...
- Both: novelty or anomaly detection

Desired output: ML tasks

- SL: classification, regression...
- UL: dimensionality reduction, clustering...
- Both: novelty or anomaly detection
- RL: game playing, auto driving, robotics

Desired output: ML tasks

- SL: classification, regression...
- UL: dimensionality reduction, clustering...
- Both: novelty or anomaly detection
- RL: game playing, auto driving, robotics
- Structured data generation (image, text, translations...)

What's inside the AI/ML box?

Machine learning

Train on some data, build a model, apply it on new data

What's inside the AI/ML box?

Machine learning

Train on some data, build a model, apply it on new data

Other ML methods

PCA, SVM, K-means, logistic regression, LME, tSNE............

What's inside the AI/ML box?

Machine learning

Train on some data, build a model, apply it on new data

Other ML methods

PCA, SVM, K-means, logistic regression, LME, tSNE............

Deep learning

- Evolution of neural networks
- Big family of methods, applications and architectures
- "Component" philosophy

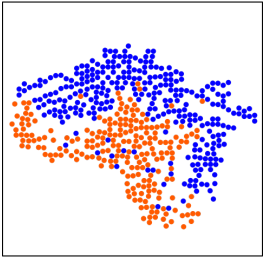
ML concepts: supervised case

Train/validation/test protocol

ML concepts: supervised case

Train/validation/test protocol

Error types



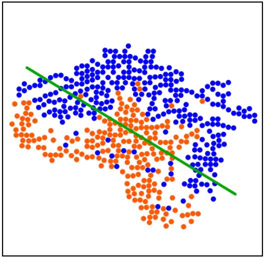
ML concepts: supervised case

Train/validation/test protocol

TRAIN	VALID.	TEST

Error types

- Train error (underfitting)



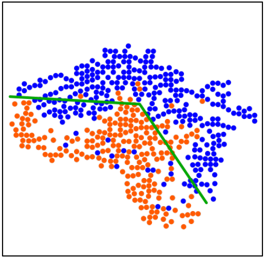
ML concepts: supervised case

Train/validation/test protocol

TRAIN	VALID.	TEST

Error types

- Train error (underfitting)

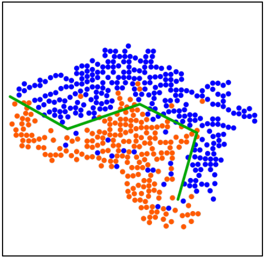


ML concepts: supervised case

Train/validation/test protocol

Error types

- Train error (underfitting)

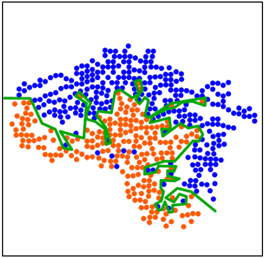


ML concepts: supervised case

Train/validation/test protocol

Error types

- Train error (underfitting)
- Validation error (overfitting)



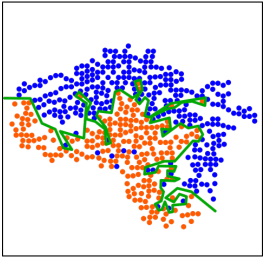
ML concepts: supervised case

Train/validation/test protocol

TRAIN	VALID.	TEST

Error types

- Train error (underfitting)
- Validation error (overfitting)
- Test error ($\neq$ distribution)

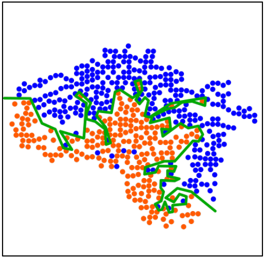


ML concepts: supervised case

Train/validation/test protocol

Error types

- Train error (underfitting)
- Validation error (overfitting)
- Test error ($\neq$ distribution)
- Human error :o

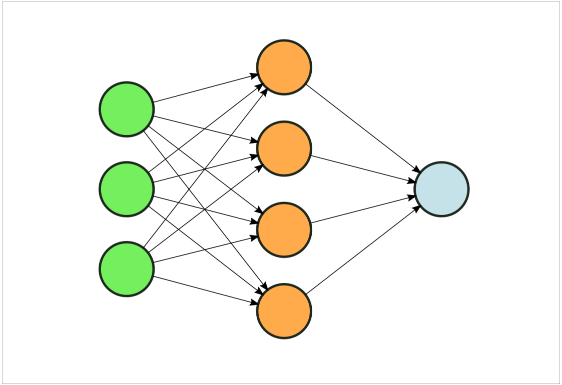


Deep learning concepts

- Neural networks (metaphor)
- Now we have huge datasets
- Now we have powerful hardware
- Deep architectures deliver much better performance
- Theoretical improvements (optimization)

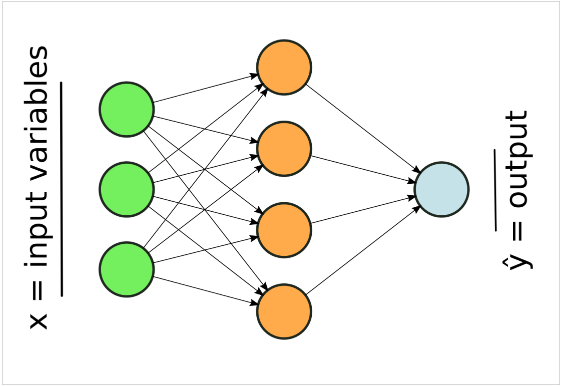
Not extensively exploited in Bioinfo!

A simple (but complete) example neural network



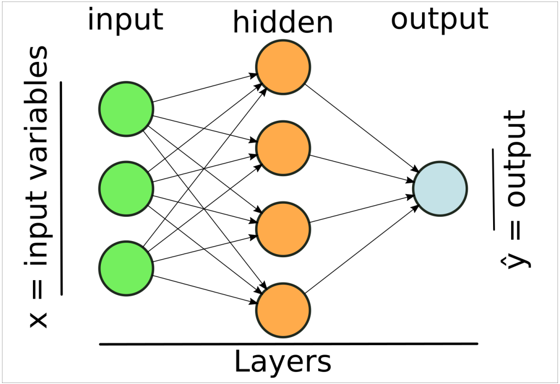
Inputs $*$ *weights* capture interactions between input variables.
Enough hidden units allow to model any Borel-measurable function.
Not deep yet!

A simple (but complete) example neural network



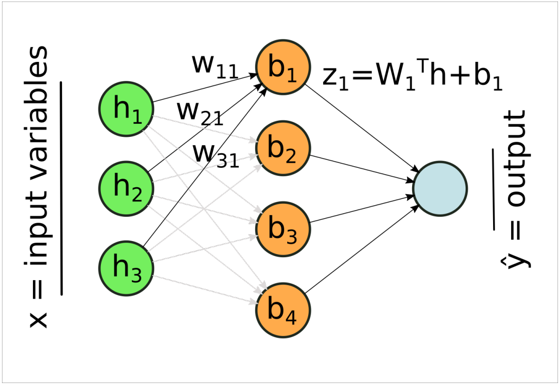
Inputs ∗ *weights* capture interactions between input variables.
Enough hidden units allow to model any Borel-measurable function.
Not deep yet!

A simple (but complete) example neural network



Inputs ∗ *weights* capture interactions between input variables.
Enough hidden units allow to model any Borel-measurable function.
Not deep yet!

A simple (but complete) example neural network



Inputs $*$ *weights* capture interactions between input variables.
Enough hidden units allow to model any Borel-measurable function.
Not deep yet!

Training a NN: two stages (1)

Forward propagation

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*
3. Compute the inputs to the hidden units $z_i = W_i^T h + b_i$

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*
3. Compute the inputs to the hidden units $z_i = W_i^T h + b_i$
4. Compute the activation function $g_1(z)$ of each hidden unit

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*
3. Compute the inputs to the hidden units $z_i = W_i^T h + b_i$
4. Compute the activation function $g_1(z)$ of each hidden unit
5. Feed the activation values to the output unit(s)

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*
3. Compute the inputs to the hidden units $z_i = W_i^T h + b_i$
4. Compute the activation function $g_1(z)$ of each hidden unit
5. Feed the activation values to the output unit(s)
6. Compute the inputs to the output unit(s) $z_j = W_j^T h + b_j$

Training a NN: two stages (1)

Forward propagation

1. Initialize weights and biases with random (small < 0.1) values
2. Feed train data on the inputs *(later: epochs, batches)*
3. Compute the inputs to the hidden units $z_i = W_i^T h + b_i$
4. Compute the activation function $g_1(z)$ of each hidden unit
5. Feed the activation values to the output unit(s)
6. Compute the inputs to the output unit(s) $z_j = W_j^T h + b_j$
7. Compute the activation function of each output unit $g_2(z)$

Training a NN: two stages (2)

Backpropagation

Goal: adjust weights and biases to make the output as close as the expected as possible.

Training a NN: two stages (2)

Backpropagation

Goal: adjust weights and biases to make the output as close as the expected as possible.

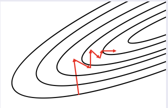
1. Let $\theta \in \mathbb{R}^d$ be all the parameters of the NN (weights + biases)
2. **Cost** function $J(\theta)$: usually the error/loss between the outputs of the NN and the expected outputs

Training a NN: two stages (2)

Backpropagation

Goal: adjust weights and biases to make the output as close as the expected as possible.

1. Let $\theta \in \mathbb{R}^d$ be all the parameters of the NN (weights + biases)
2. **Cost** function $J(\theta)$: usually the error/loss between the outputs of the NN and the expected outputs
3. Adjust θ using **Gradient Descent** optimization
   - $\theta = \theta - \eta \Delta \nabla_\theta J(\theta)$
   - η is the **learning rate** (step size)

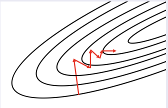


Training a NN: two stages (2)

Backpropagation

Goal: adjust weights and biases to make the output as close as the expected as possible.

1. Let $\theta \in \mathbb{R}^d$ be all the parameters of the NN (weights + biases)
2. **Cost** function $J(\theta)$: usually the error/loss between the outputs of the NN and the expected outputs
3. Adjust θ using **Gradient Descent** optimization
   - $\theta = \theta - \eta \Delta \nabla_\theta J(\theta)$
   - η is the **learning rate** (step size)



4. Possibly re-run the NN with new or shuffled train data (**epoch**)

Gradient Descent (GD): considerations

Learning rate: yet another hyperparameter

Tradeoff: large step (overstepping) vs small step (slow learning)

- Decreasing η
- Momentum / Nesterov momentum
- Adagrad / RMSProp / ADAM

Gradient Descent (GD): considerations

Learning rate: yet another hyperparameter

Tradeoff: large step (overstepping) vs small step (slow learning)

- Decreasing η
- Momentum / Nesterov momentum
- Adagrad / RMSProp / ADAM

Granularity of gradient updates

- **Batch** GD. All train examples per update. *Expensive*
- **Stochastic** GD. One (random) example per update. *High variance*
- **Mini-batch** GD. Some (power of 2) examples per update. *Best option*

Output units

General considerations

- Type depends on the task type
- Must not saturate easily (i.e. give meaningful gradients)
- Easy to optimize (≈linear), compatible with cost function
- Cost function: **cross-entropy**

$$C = -\frac{1}{|x|}\sum_{x}(y \ln \hat{y} + (1-y)\ln(1-\hat{y}))$$

$\hat{y}$: output of the NN; y: expected output; x: training samples

Output units

General considerations

- Type depends on the task type
- Must not saturate easily (i.e. give meaningful gradients)
- Easy to optimize (≈linear), compatible with cost function
- Cost function: **cross-entropy**

$$C = -\frac{1}{|x|}\sum_{x}(y \ln \hat{y} + (1-y)\ln(1-\hat{y}))$$

$\hat{y}$: output of the NN; y: expected output; x: training samples

- Regression (Gaussian) → **linear** unit
- Binary classification (Bernoulli) → **sigmoid** unit
- Multiclass classif. (Multinoulli) → **softmax** units (one unit per class, outputs sum 1, initially fuzzy)

Hidden and input units

General considerations

- Basically hidden $\equiv$ input
- Easy to optimize (but not necessarily differentiable everywhere)
- Desirable $\approx$ linear behaviour

Hidden and input units

General considerations

- Basically hidden $\equiv$ input
- Easy to optimize (but not necessarily differentiable everywhere)
- Desirable $\approx$ linear behaviour

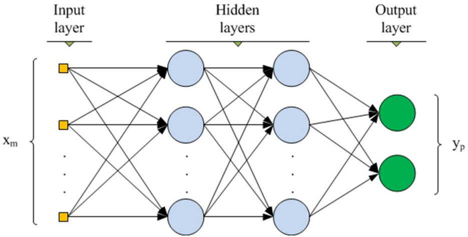
- Logistic sigmoid / hyperbolic tangent $\rightarrow$ *deprecated**
- Rectified Linear Unit (ReLU) $\rightarrow$ **best**: $g(z) = \max\{0, z\}$
- Maxout $\rightarrow$ good for some problems
  - Split input in blocks
  - Pick the highest input as output

How can we reduce the validation error?

- Use more train data (I wish I had it!)
- Data augmentation
  - Data synthesis
  - Add noise: input data, weights of hidden layers, outputs
- Parameter normalization (L_1 or L_2)
- Early stopping (train error stable, validation error grows)
- Dropout: randomly deactivate units

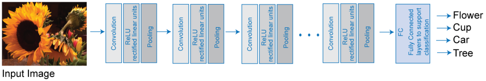
Multi-layer perceptron / feed-forward NN

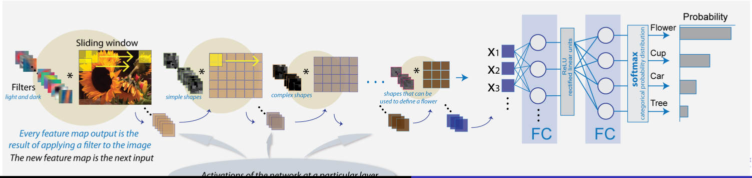
- Tasks: **regression**, **classification**
- The outputs of one layer are the inputs of the next (no loops)
- Totally connected (dense) layers by default
- Efficient, easy to optimize



Convolutional neural networks

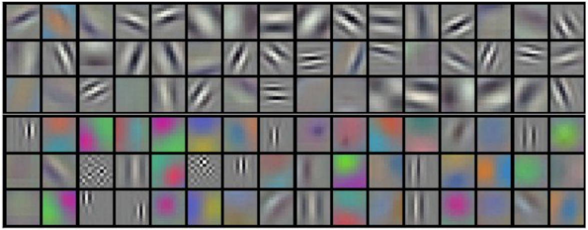
- Tasks: **image processing** (locally-structured data)
- Convolutional layers: specialized filters
- Pooling layers: reduce size (can be maxout)
- Parameter sharing: same filters applied to whole input (dim. reduction, no overfitting)



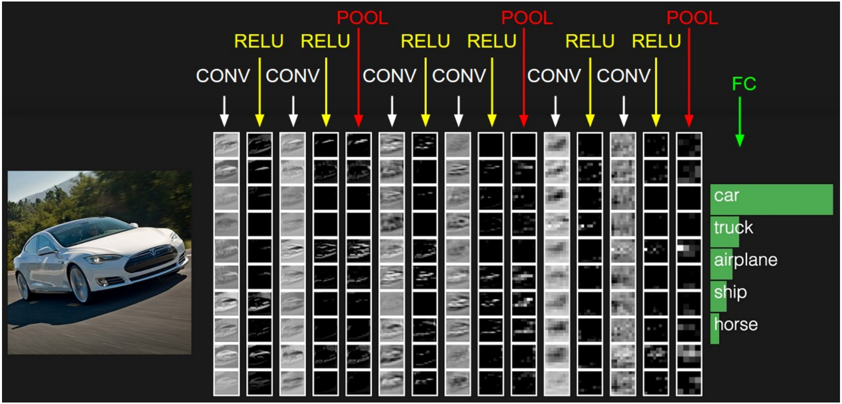


Convolutional neural networks

- Tasks: **image processing** (locally-structured data)
- Convolutional layers: specialized filters
- Pooling layers: reduce size (can be maxout)
- Parameter sharing: same filters applied to whole input (dim. reduction, no overfitting)

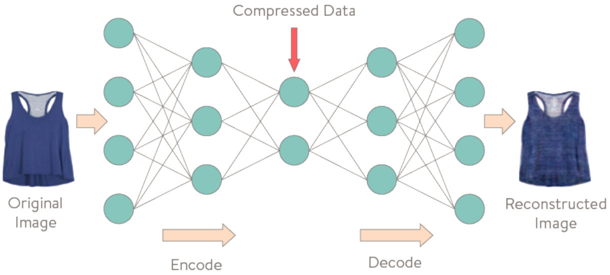


CNN filters in action

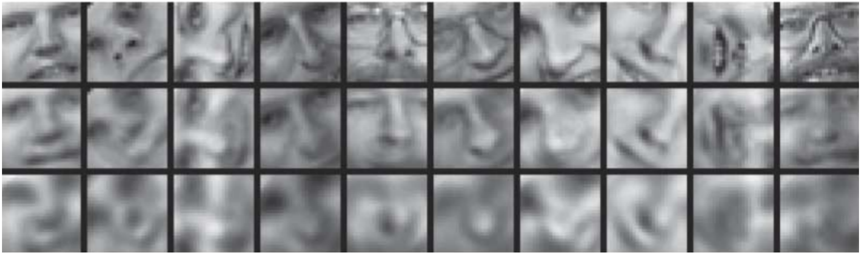


Autoencoders

- Tasks: dimensionality reduction, noise filtering
- Output = input. Weird!!
- Trick: middle layer(s) are smaller than input, so a "compressed" representation is forced



Autoencoder vs PCA

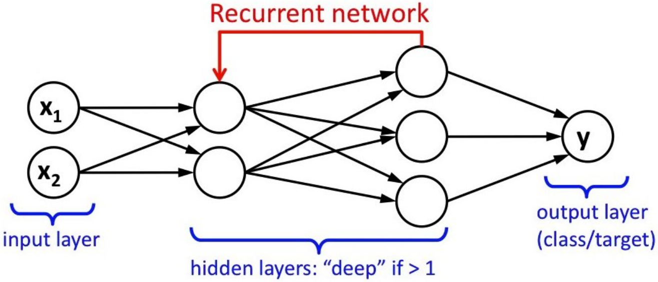


.

First row: original image; second row: autoencoder representation; third row: PCA reconstruction

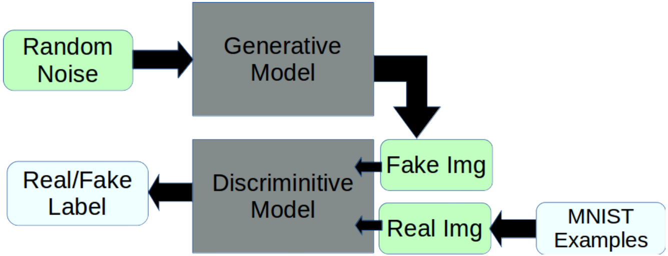
Recurrent neural networks

- Tasks: sequence (signal) processing
- There are loops and/or units with memory
- Harder to optimize/computationally more expensive
- Long-short term memory (LSTM): very effective



Generative adversarial networks

- Tasks: data generation
- One network (generative) trying to *cheat* the other (discriminative)
- Can create lifelike images/others



How many hidden layers? How wide?

- Layer width: combinatorial, no generalization

How many hidden layers? How wide?

- Layer width: combinatorial, no generalization
- Deeper net (more hidden layers): layers extract advanced features from the previous ones (better generalization)

How many hidden layers? How wide?

- Layer width: combinatorial, no generalization
- Deeper net (more hidden layers): layers extract advanced features from the previous ones (better generalization)
- How to decide? (This is an art)

How many hidden layers? How wide?

- Layer width: combinatorial, no generalization
- Deeper net (more hidden layers): layers extract advanced features from the previous ones (better generalization)
- How to decide? (This is an art)
  - Train error: increase width and depth

How many hidden layers? How wide?

- Layer width: combinatorial, no generalization
- Deeper net (more hidden layers): layers extract advanced features from the previous ones (better generalization)
- How to decide? (This is an art)
  - Train error: increase width and depth
  - Validation error: increase depth

DL libraries

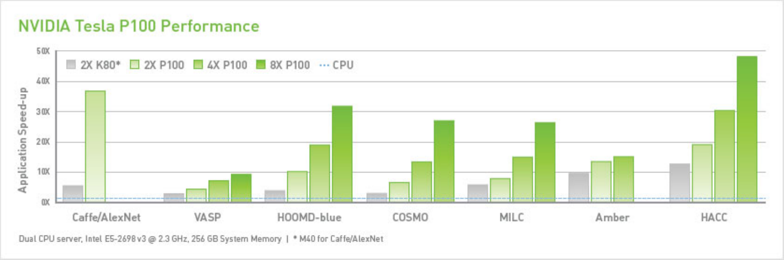
- Theano
- Tensorflow
- Caffe
- Lasagne*
- Keras*
- CNTK
- dl4j
- ...

Example: CNN on MNIST using Keras

```
model = Sequential()

model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Where do you train your DL experiments?

- CPU: slooooow
- GPU: mainstream, really fast
- FPGA: ???
- Multi-core CPUs: ???



About to finish, so...

Any questions?