Introduction to Artificial Learning and Computer Vision

Andreu Casas

Session Outline

1 Artificial Learning

- **Notes** Artificial Intelligence. Sounds fancy, but how does it work?
- **Module** From traditional to artificial learning: predicting which bills will become law

2 Computer Vision

- **Notes** Convolutional Neural Networks
- **Module** Classifying images with a CNN trained on ImageNet

Artificial Learning
Why?

In the last few years Artificial Neural Networks and deep learning have drastically improved machine-learning performance.

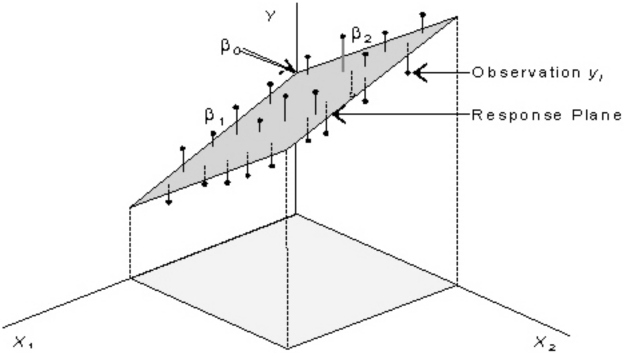
- Speech-recognition (e.g. Siri, Echo, Alexa)
- Translation (e.g. Google translator)
- Image recognition (e.g. Facebook's facial recognition photo tagging)

Artificial Learning

In "conventional" machine learning, we only use a single parameter matrix: 1 variable = 1 coefficient.

Linear Model: regression formula

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon_i$$



Artificial Learning

In "conventional" machine learning, we only use a single parameter matrix: 1 variable = 1 coefficient.

Linear Model: compact matrix form

$$\mathbf{Y} = \mathbf{X}\beta$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{22} \\ 1 & x_{41} & x_{22} \\ \vdots & \vdots & \\ 1 & x_{n1} & x_{n2} \end{bmatrix} * \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

Artificial Learning

In "conventional" machine learning, we only use a single parameter matrix: 1 variable = 1 coefficient.

- Interested in finding the parameter matrix β that minimizes predictive error
- This is easy when using a Least Square regression because there is an analytic solution

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'y$$

- We can use MLE to find this parameter matrix for more complex general linear models

$$\mathbf{Y} = \mathit{logit}(\mathbf{X}\beta)$$

Artificial Learning

Conventional machine learning only take us so far... what about extending the learning process?

What if we use the output of a first model as input of a second model...

$$\hat{\mathbf{Y}}_{\mathbf{1}} = \mathbf{X}_1 \beta_1$$

$$\hat{\mathbf{Y}}_{\mathbf{2}} = \mathbf{X}_2 \beta_2$$

where

$$\hat{\mathbf{Y}}_{\mathbf{1}} = \mathbf{X}_{\mathbf{2}}$$

and we try to minimize $\mathbf{Y} - \hat{\mathbf{Y}}_{\mathbf{2}}$ instead of $\mathbf{Y} - \hat{\mathbf{Y}}_{\mathbf{1}}$?

This is what we call a Neural Network or Artificial Neural Network!

Artificial Learning

Matrix multiplication is the key to understand neural nets!

When you multiply matrices remember that:

- the number of columns in the first matrix has to be the same than the number of rows in the second matrix

- the number of rows of the resulting matrix will equal the number of rows of the first matrix, and the number of columns will equal the number of columns of the second matrix

$$A[n, k] * B[k, z] = C[n, z]$$

Artificial Learning

Matrix multiplication is the key to understand neural nets!

Instead of a simple linear or general linear model we can have a model that looks like this...

$$\text{Sigmoid}\boldsymbol{(}\mathbf{X_1}[1000, 4]\ \beta_1[4, 250]\boldsymbol{)}\ \beta_2[250, 1] = \mathbf{Y}[1000, 1]$$

...

(1) $\mathbf{X_1}[1000, 4]\ \beta_1[4, 250] \rightarrow \mathbf{X_2}[1000, 250]$

(2) $\text{Sigmoid}(\mathbf{X_2}[1000, 250]) \rightarrow \mathbf{X_{2b}}[1000, 250]$

(3) $\mathbf{X_{2b}}[1000, 250]\ \beta_2[250, 1] \rightarrow \hat{\mathbf{Y}}[1000, 1]$

We calculate the parameters in the matrices β_1 and β_2 using e.g. Stochastic Gradient Descent $\rightarrow$ iterating until convergence

Artificial Learning

Some basic terminology... different words for some familiar concepts

- **input layer**: the original data matrix ($\mathbf{X}$)
- **weight/s**: a single parameter (β_{ij}) / parameter matrix (β)
- **bias**: the intercept parameter matrix (α or β_0)
- **ReLu, Sigmoid, Tanh**: non-linear transformation we apply to $\mathbf{X}$ matrices. Also known as **activation functions**
- **hidden layer**: $\mathbf{X_1}, \mathbf{X_2}, ...$ a new intermediate representation of the input
- **loss function**: the function we want to minimize (e.g. $\hat{\mathbf{Y}} - \mathbf{Y}$)
- **regularization**: transformations we apply to the loss function (e.g. $|\hat{\mathbf{Y}} - \mathbf{Y}| \rightarrow$ L1 and $(\hat{\mathbf{Y}} - \mathbf{Y})^2 \rightarrow$ L2)
- **dropout**: setting some β_{ij} from a β matrix to 0 at random
- **forward propagation**: performing all matrix multiplications
- **backpropagation**: calculating Stochastic Gradient Descent
- **graph**: a model

Artificial Learning

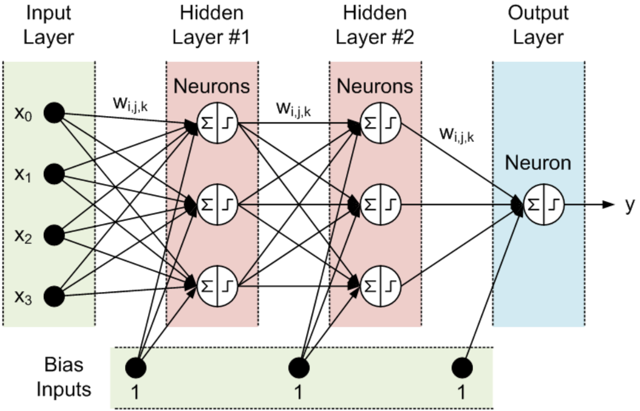
Some more terminology and... hyperparemeters, the dark mysteries of neural nets

- **train-validation-test split**: 80-10-10? 50-25-25?
- **batch size**: the number of training observations we use for training in a given iteration
- **epochs**: number of training iterations
- **dropout rate**: the probability re-initializing a given weight
- **learning rate**: by how much we update the weights at each training iteration

There are some conventions people follow. Since we are preforming supervised training, we always look for the hyperparemeters that achieve the highest out-of-sample accuracy.

Artificial Learning

Neural nets are often represented this way



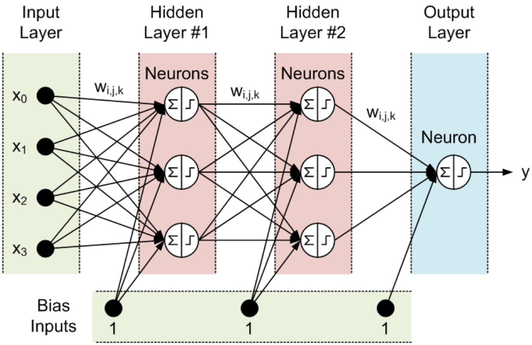
To be fair, the term “deep learning” should be used only when the neural networks have a several hidden layers. But how deep does a neural net to be in order to be considered deep learning?

Artificial Learning

Fine tuning or transfer learning

Slightly tweaking an already trained neural net to predict a different outcome

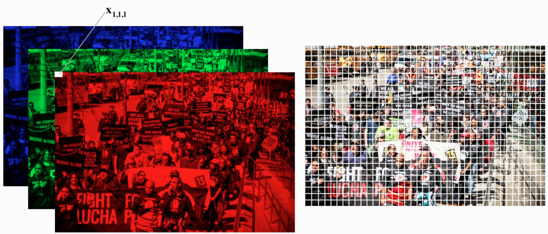
- Retraining the whole neural net with new data
- Retraining part of the neural net with new data
- Adding or changing layers



[Module]
From traditional to artificial learning: predicting which bills will become law

Convolutional Neural Nets for Computer Vision
Two main differences

(1) Images as inputs: 3-dimensional matrices (width x height x depth)



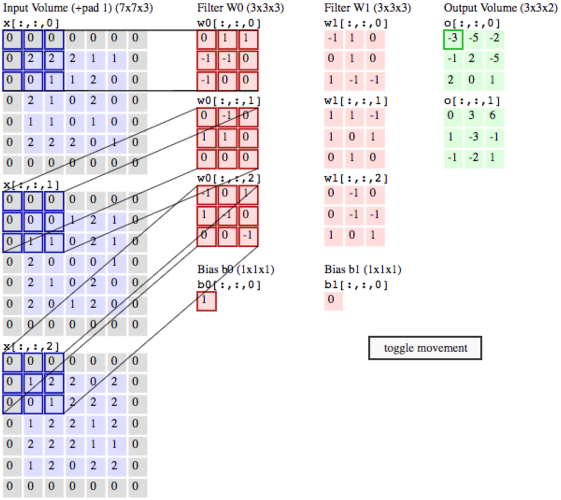
$\mathbf{X} =$

$$
\begin{bmatrix} x_{111} & x_{112} & \dots & x_{11n} \\ x_{121} & x_{122} & \dots & x_{12n} \\ x_{131} & x_{132} & \dots & x_{13n} \\ x_{141} & x_{142} & \dots & x_{14n} \\ \vdots & \vdots & & \\ x_{1n1} & x_{1n2} & \dots & x_{1nn} \end{bmatrix},
\begin{bmatrix} x_{211} & x_{212} & \dots & x_{21n} \\ x_{221} & x_{222} & \dots & x_{22n} \\ x_{231} & x_{232} & \dots & x_{23n} \\ x_{241} & x_{242} & \dots & x_{24n} \\ \vdots & \vdots & & \\ x_{2n1} & x_{2n2} & \dots & x_{2nn} \end{bmatrix},
\begin{bmatrix} x_{311} & x_{312} & \dots & x_{31n} \\ x_{321} & x_{322} & \dots & x_{32n} \\ x_{331} & x_{332} & \dots & x_{33n} \\ x_{341} & x_{342} & \dots & x_{34n} \\ \vdots & \vdots & & \\ x_{3n1} & x_{3n2} & \dots & x_{3nn} \end{bmatrix}
$$

Convolutional Neural Nets for Computer Vision

Two main differences

(2) Weights (**filters**) are not connected to the whole **input volume**: convolution.



Convolutional Neural Nets for Computer Vision

Some new terminology... and more hyperparameters

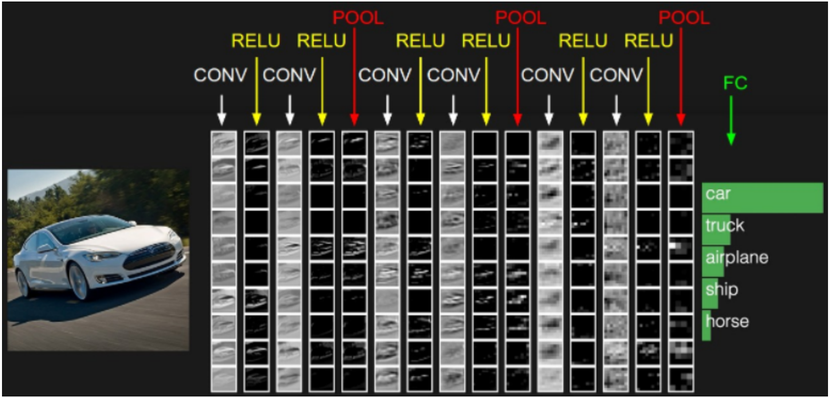
- **input volume**: a 3-dimensional input
- **convolutional layer**: a 3-dimensional layer where convolutional filters are applied to the input volume. FxFxK where F is the width and height of the filter, and K is the number of filters in the layer → 3x3x2 in the previous example
- **stride**: the number of pixels we move the filter at a time. This is 2 in the previous example
- **zero-padding**: adding zeros around the input border (it's still unclear to me why we do this)
- **pooling layer**: a layer where we reduce the size the output of a convolutional layer. From 224x224x64 to 112x112x64 for example.

Convolutional Neural Nets for Computer Vision
Some new terminology... and more hyperparameters

- **fully connected layer**: a layer of weights that are connected to the whole input volume. These are usually at the end of a network
- **softmax**: a multi-class classifier. This is basically a multinomial logit model that uses the output of the last fully-connected layer to predict the final classes of interest

Convolutional Neural Nets for Computer Vision

This is how a ConvNet looks like



Convolutional Neural Nets for Computer Vision
VGG16's architecture

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Convolutional Neural Nets for Computer Vision

Let's practice!

[Module]
Classifying images with VGG16