FROM LINEAR REGRESSION TO NEURAL NETWORKS

Afi Escuela de Finanzas

ABOUT ME

75% Math 25% Computers

NLP

Computer Vision

SOME AREAS OF EXPERTISE

SOME FRAMEWORK PROFICIENCIES

Machine Learning Stuff

Video Games stuff

Actually playing Video Games

Analytical Index

1. **Intro: What is Deep Learning?**
2. **From Logistic Regression to Neural Nets**
3. **Backpropagation**
4. **Stochastic Gradient Descent**
5. **Network Examples**
6. **Use Cases and Techniques**

What is Deep Learning?

ARTIFICIAL INTELLIGENCE

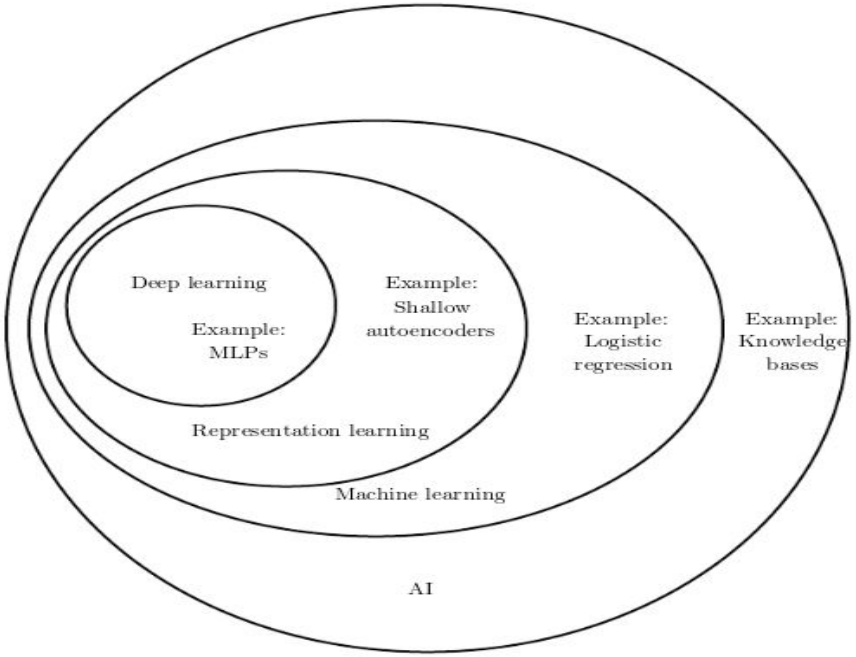


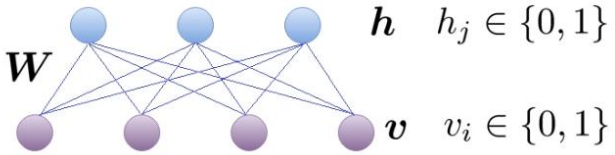
Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

From the book “Deep Learning”, Ian Goodfellow and Yoshua Bengio and Aaron Courville

SOME APPLICATIONS

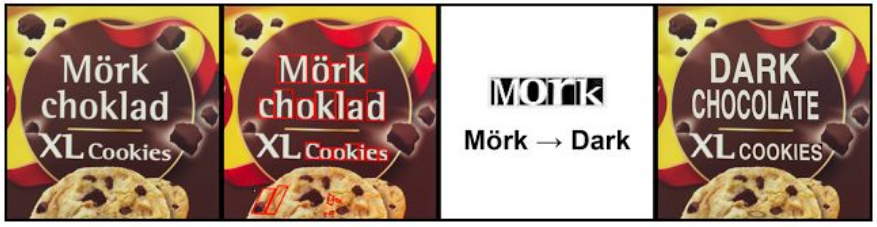
Object Classification and Detection

Restricted Boltzmann Machines (RBM)



RBM on Recommender Systems

Instant Visual translation



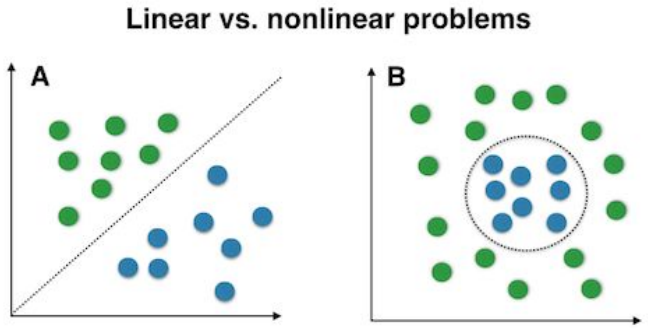
Film scripts. Yes, it's for real

CNN + LSTM to describe pictures

From Logistic Regression to Neural Nets

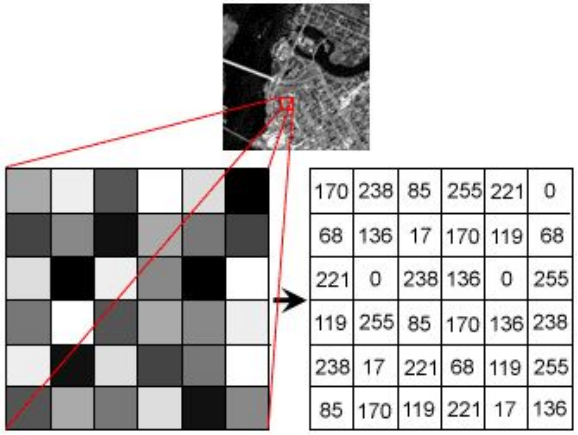
Motivation: The need for non-linear decision boundaries

Decision boundary = set of points that are equally likely to belong to any of two classes
Simple classifiers such as **linear regression and logistic regression** can only find linear boundaries

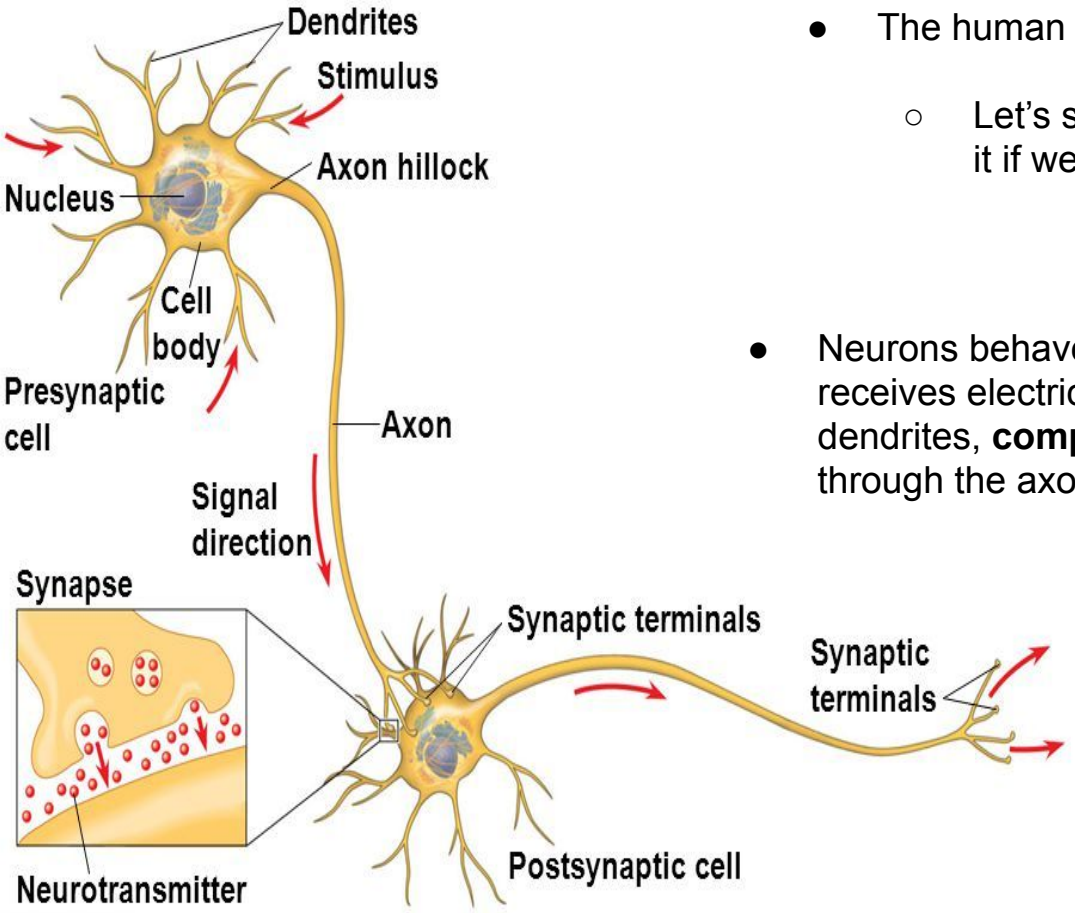


- Trick: create new features as non-linear combinations of existing features, and give them to the linear classifier
  - Eg: use x_1 and x_2 but also x_1x_2 , x_1^2 , x_2^2
  - Pros: still using a simple (white-box) classifier, understandable by business people (econometrics)

- What happens if we have *n* = 50 features?
  - Too many new features generated
- Even worse: what if we have an image, where each pixel is a feature, and want to learn the concept being displayed?



The brain as the most perfect “universal” learning algorithm



- The human brain can learn almost any task
  - Let’s see how it is structured and try to imitate it if we want a really good learning machine

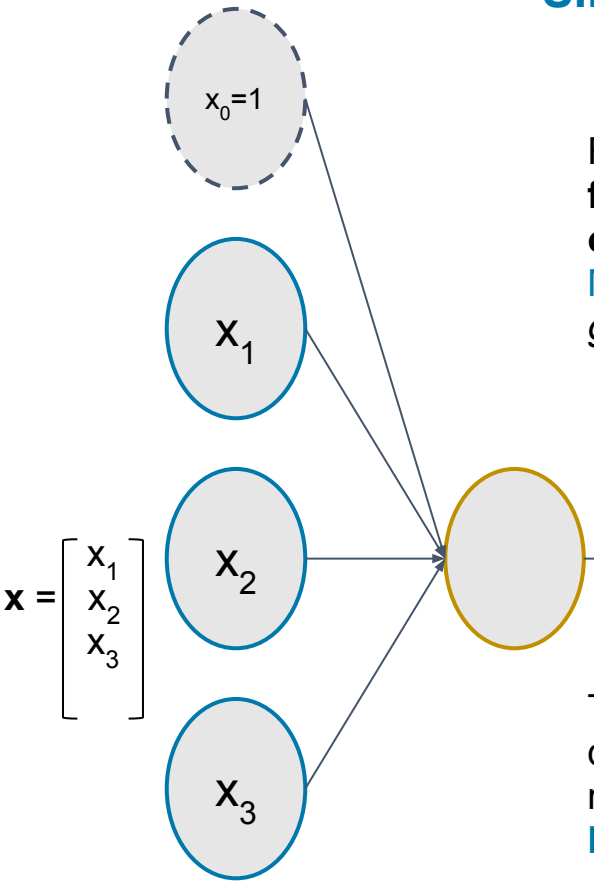
- Neurons behave as computing units. Each neuron receives electric input signals (called *spikes*) through the dendrites, **computes** an output with them and sends it through the axon to other neurons connected to it

- Axon-dendrite transmission is done in a mechanism called synapse.
- This process never changes, yet the brain is always learning

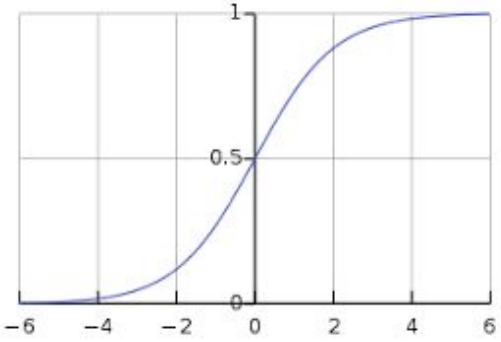
Computational model of a neural network

SIMPLE PERCEPTRON

$g(w^T\mathbf{x}) = 1 / (1 + e^{-(w0 + w1x1 + w2x2 + w3x3)})$



Function $g(\cdot)$ is known as the **activation function** and it is a **non-linear function of the input**.
Most often, it is the **sigmoid function**:
$g(z) = 1 / (1 + e^{-z})$



Output (called ***activation***):
$g(\mathbf{w}^T\mathbf{x}) = g(w_0 + w_1x_1 + w_2x_2 + w_3x_3)$

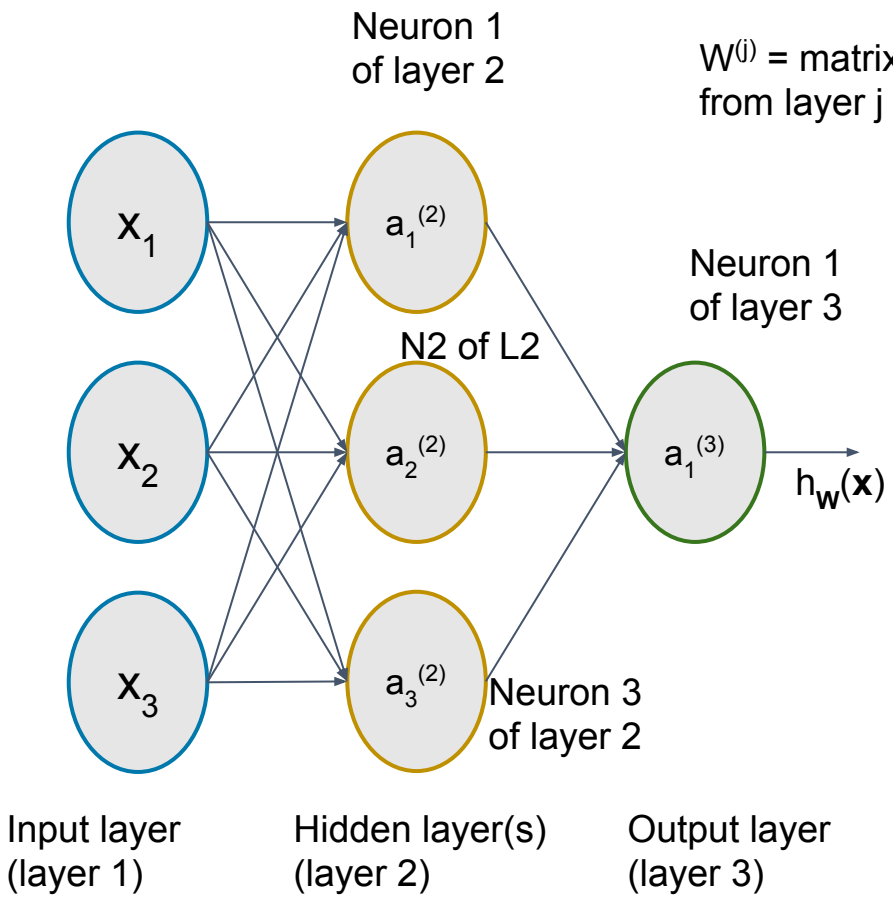
The value of the parameters $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$ depends on the neuron (they represent "**what the neuron has learned upt to now**").

Conclusion: with the sigmoid, each neuron actually learns a **logistic regression** for a (mysterious) sub-task which contributes appropriately to the network task.

Equation ("hypothesis") of a small neural network

$a_i^{(j)}$ = activation of the i-th neuron of layer j

$W^{(j)}$ = matrix of parameters multiplied by the inputs (activations) from layer j to compute the activations of layer j+1

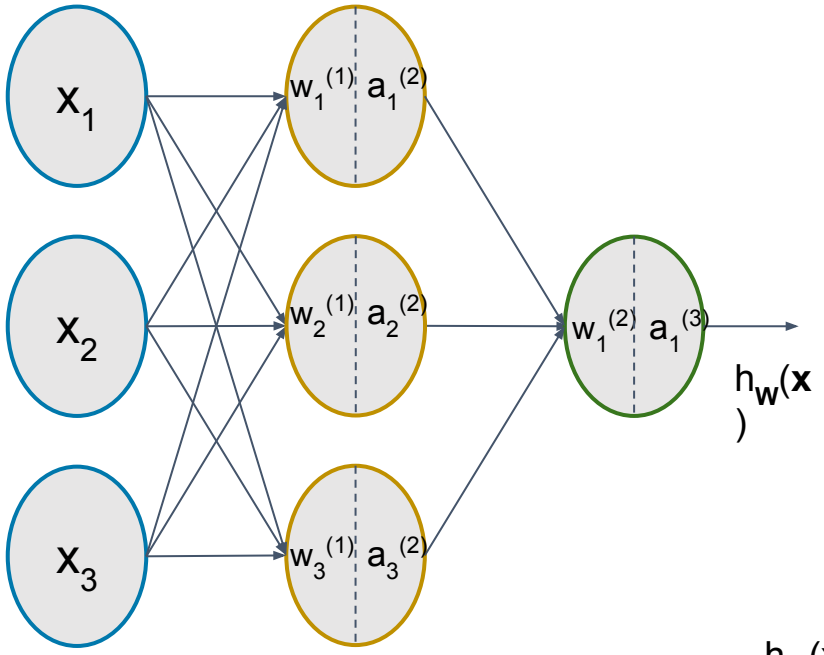


MULTILAYER PERCEPTRON (MLP)

Equation ("hypothesis") of a small neural network

$a_i^{(j)}$ = activation of the i-th neuron of layer j

$W^{(j)}$ = matrix of parameters to be multiplied by the inputs (activations) from layer j to compute the activations of layer j+1



Input layer (layer 1)

Hidden layer(s) (layer 2)

Output layer (layer 3)

$$W^{(1)} = \begin{bmatrix} (w_1^{(1)})^T \\ (w_2^{(1)})^T \\ (w_3^{(1)})^T \end{bmatrix} = \begin{bmatrix} w_{1,0}^{(1)} & w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,0}^{(1)} & w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,0}^{(1)} & w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} (w_1^{(2)})^T \end{bmatrix} = \begin{bmatrix} w_{1,0}^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = g(w^{(1)}_{10} + w^{(1)}_{11} x_1 + w^{(1)}_{12} x_2 + w^{(1)}_{13} x_3)$$

$$a_2^{(2)} = g(w^{(1)}_{20} + w^{(1)}_{21} x_1 + w^{(1)}_{22} x_2 + w^{(1)}_{23} x_3)$$

$$a_3^{(2)} = g(w^{(1)}_{30} + w^{(1)}_{31} x_1 + w^{(1)}_{32} x_2 + w^{(1)}_{33} x_3)$$

$$h_{\mathbf{w}}(\mathbf{x}) = a_1^{(3)} = g(w^{(2)}_{10} + w^{(2)}_{11} a_1^{(2)} + w^{(2)}_{12} a_2^{(2)} + w^{(2)}_{13} a_3^{(2)})$$

which is a logistic regression with new variables a_1, a_2, a_3 created as non-linear transformations of x_1, x_2, x_3

A matricial compact form to compute the output

$a_i^{(j)}$ = activation of the i-th neuron of layer j

$B^{(j)}$ = matrix of parameters multiplied by the inputs (activations) from layer j to compute the activations of layer j+1

$$W^{(1)} = \begin{bmatrix} (w_1^{(1)})^T \\ (w_2^{(1)})^T \\ (w_3^{(1)})^T \end{bmatrix} = \begin{bmatrix} w_{1,0}^{(1)} & w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,0}^{(1)} & w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,0}^{(1)} & w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix}$$

$$W^{(2)} = (w_1^{(2)})^T = \begin{bmatrix} w_{1,0}^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = g(w^{(1)}_{10} + w^{(1)}_{11} x_1 + w^{(1)}_{12} x_2 + w^{(1)}_{13} x_3) = g(z_1^{(2)})$$
$$a_2^{(2)} = g(w^{(1)}_{20} + w^{(1)}_{21} x_1 + w^{(1)}_{22} x_2 + w^{(1)}_{23} x_3) = g(z_2^{(2)})$$
$$a_3^{(2)} = g(w^{(1)}_{30} + w^{(1)}_{31} x_1 + w^{(1)}_{32} x_2 + w^{(1)}_{33} x_3) = g(z_3^{(2)})$$

$$\mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = W^{(1)} \mathbf{a}^{(1)} = W^{(1)} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

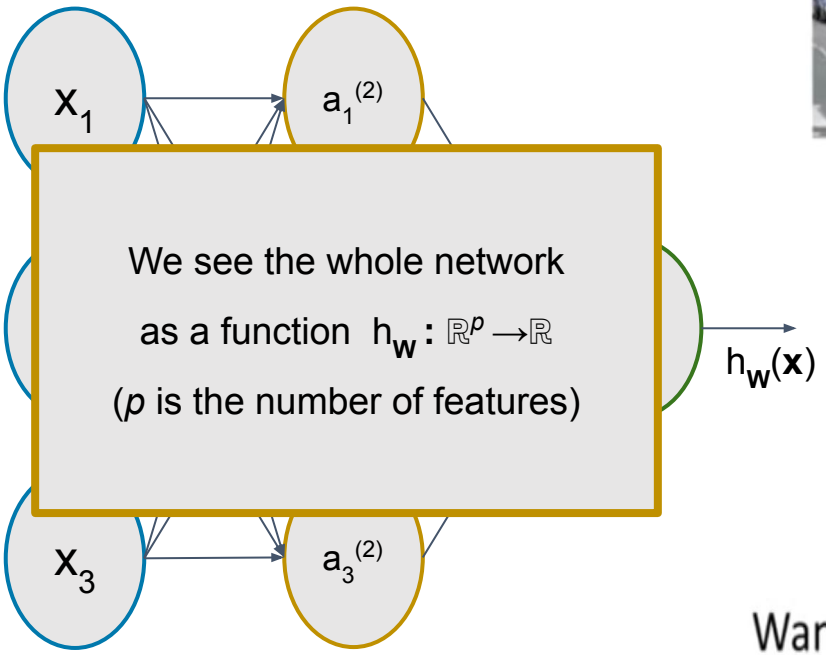
$$\mathbf{a}^{(2)} = (1, g(\mathbf{z}^{(2)}))^T = \begin{bmatrix} 1 \\ g(z_1^{(2)}) \\ g(z_2^{(2)}) \\ g(z_3^{(2)}) \end{bmatrix} \quad \text{(apply } g \text{ element-wise)}$$

$$\mathbf{z}^{(3)} = B^{(2)} \mathbf{a}^{(2)}$$

$$\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)}) \quad \text{(apply } g \text{ element-wise)}$$

$$h_{\mathbf{w}}(\mathbf{x}) = a_1^{(3)} = g(w^{(2)}_{10} + w^{(2)}_{11} a_1^{(2)} + w^{(2)}_{12} a_2^{(2)} + w^{(2)}_{13} a_3^{(2)})$$

Two-class and multiclass classification with a neural network



2 classes = 1 neuron in the output layer

Multiple output units: One-vs-all.

We see the whole network
as a function $h_{\mathbf{w}}: \mathbb{R}^p \rightarrow \mathbb{R}^K$
(*p* is the number of features)
(*K* is the number of classes)

$(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

K classes = K neurons in the output layer (K > 2)

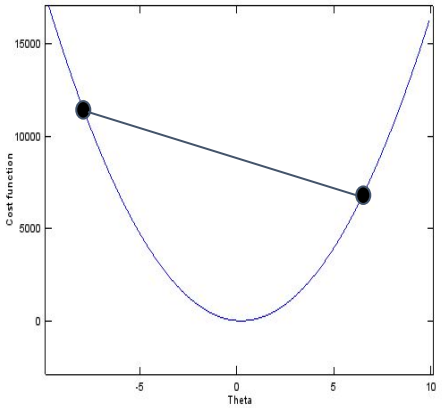
The concept of cost function in Machine Learning

- In any Machine Learning model, fitting a model means finding the best values of its parameters.
- The *best* model is that whose parameter values *minimize* the total error with respect to the actual outputs
- Since the error depends on the parameters chosen, it is a *function* called *cost function* J. The most common error measure is the MSE (Mean Squared Error):

$J(w_0, w_1, \ldots, w_R) = [\ 1/(2m)\ ] \sum_{i=1}^{m} (\ \hat{y}_i\ - y_i)^2 =$ m = number of training examples

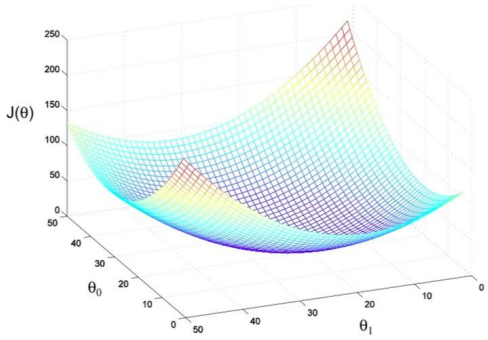
$= [\ 1/(2m)\ ] \sum_{i=1}^{m} (h_{w0,\ w1,\ \ldots,\ wR}(\mathbf{x}_i) - y_i)^2$ $h_{w0,\ w1,\ \ldots,\ wR}$ = hypothesis (the equation of the model)

- Instead of minimizing the total sum, we minimize the average error, (1/m).
- Finding the optimum of any function *f* is equivalent to finding the optimum of *f* / 2.
- Hence we divide by 2 because it eases further calculations.



Cost function of one parameter

Ideally the **cost function is convex**: for every pair of points, the curve is always below the line (or hyper-plane) between them



Cost function of two parameters

Gradient descent with one variable w

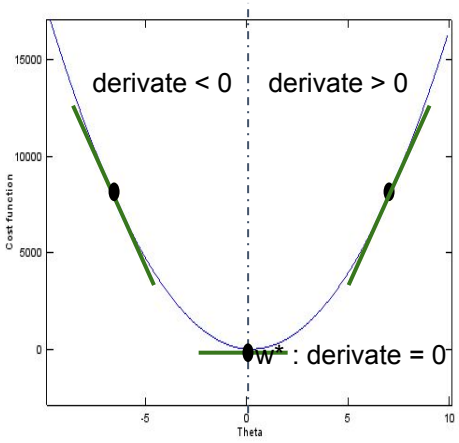
- **If the cost function is convex**: only **one global optimum, which is the global minimum**.
- Closed form of the minimum:
  - Compute the derivative of the model equation and find the point where it is 0 (exact solution).
  - Multiple variables: partial derivatives, and solve an equation system to find where all are 0 simultaneously.
- Can be difficult if the model equation is complicated, has many variables (large equation system) or is not derivable.
- Solution: approximate iterative algorithm called *gradient descent* (also valid for non-convex functions!)

GRADIENT DESCENT ALGORITHM

$w_0 \leftarrow$ some initial value
$\eta \leftarrow$ some fixed (small) constant
(η is called the *learning ratio*)
$t \leftarrow 0$
tolerance $\leftarrow$ small value (e.g: 0.000001)

while $dJ/dw \mid_{w = w(t)} >$ tolerance :

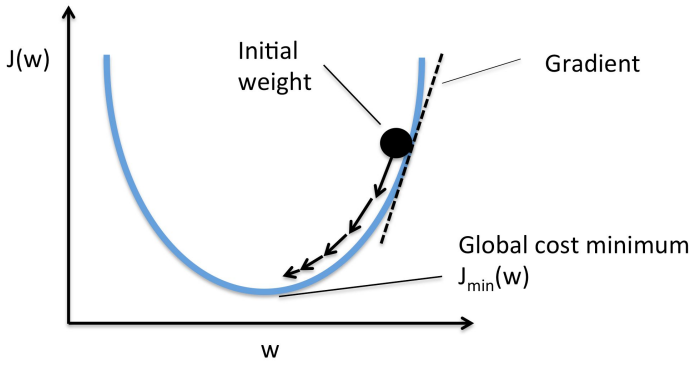
$$w^{(t+1)} \leftarrow w^{(t)} - \eta \left(dJ/dw \mid_{w = w(t)} \right)$$



w ⟶ **⟵ w**

dJ/dw < 0 :
w(t+1) > w(t):
w increases

dJ/dw > 0 :
w(t+1) < w(t):
w decreases



Gradient descent in general (variables w = (w0, w1, …, wp))

- With multiple variables: evaluate the modulus (noted ||·||) of the gradient vector to test the stopping criterion

GRADIENT DESCENT ALGORITHM (p variables)

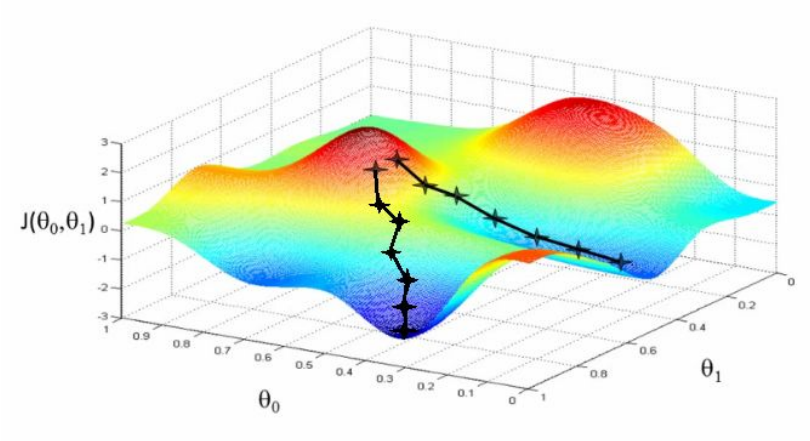
$\mathbf{w}_0 \leftarrow$ some initial vector $(w_{01}, \ldots, w_{0p})$
$\eta \leftarrow$ some fixed (small) constant
(η is called the *learning ratio*)
$t \leftarrow 0$
tolerance $\leftarrow$ small value (e.g: 0.000001)

while $||\nabla g|_{\mathbf{w} = \mathbf{w}(t)}|| >$ tolerance:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \, || \nabla g|_{\mathbf{w} = \mathbf{w}(t)} ||$$

NOTE: $\nabla g = (\partial J/\partial w_0 , \partial J/\partial w_1, \ldots , \partial J/\partial w_p) \in \mathbb{R}^p$

- In general, cost functions are not convex: many local optima. Gradient descent does not guarantee the minimum
- The solution found depends on the starting point



In summary: we need to compute the partial derivatives at each parameter point

Gradient descent with two variables

- Note that the error function J **is determined uniquely by the dataset and the shape model being fitted**.
- The function (and hence its derivative) does not change during the algorithm, we just evaluate the derivate function at different values of the model parameters, which are the variables of that function.

- E.g: linear regression with 2 variables x_1, x_2. The model is

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

Imagine this dataset: Sample 1: **x** ; **y**= (2, 3; 1.8)
Sample 2: **x** ; **y**= (4, 5; 3.2) (having only m = 2 examples)

Then

$$J(w_0, w_1, w_2) = (1/(2\cdot 2))\,[(w_0 + 2w_1 + 3w_2 - 1.8)^2 + (w_0 + 4w_1 + 5w_2 - 3.2)^2]$$

$$\partial J/\partial w_0 = (1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8) + (w_0 + 4w_1 + 5w_2 - 3.2))$$
$$\partial J/\partial w_1 = (1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8)\cdot 2 + (w_0 + 4w_1 + 5w_2 - 3.2)\cdot 4)$$
$$\partial J/\partial w_2 = (1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8)\cdot 3 + (w_0 + 4w_1 + 5w_2 - 3.2)\cdot 5)$$

$$\nabla g = ((1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8) + (w_0 + 4w_1 + 5w_2 - 3.2)),$$
$$(1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8)\cdot 2 + (w_0 + 4w_1 + 5w_2 - 3.2)\cdot 4),$$
$$(1/2)\,((w_0 + 2w_1 + 3w_2 - 1.8)\cdot 3 + (w_0 + 4w_1 + 5w_2 - 3.2)\cdot 5))$$

and now we can start evaluating ∇g at different points $\mathbf{w}^{(t)}$, each point being a vector of $\mathbb{R}^3$.

Gradient descent with two variables

- In logistic regression, $h_{\boldsymbol{\beta}}(\mathbf{x}) = 1 / (1 + e^{-(\boldsymbol{\beta}^T\mathbf{x})})$ and so the cost function of a logistic regression,

$$J(\beta_0, \beta_1, \ldots, \beta_R) = [\ 1/(2m)\ ] \sum_{i=1}^{m} (\ 1 / (1 + e^{-(\boldsymbol{\beta}^T\mathbf{x})}) - y_i)^2$$

is non-convex. A somewhat equivalent, convex cost function in logistic regression is

$$J(\beta_0, \beta_1, \ldots, \beta_R) = -(1/m) [\ \sum_{i=1}^{m} y_i \log(h_{\boldsymbol{\beta}}(\mathbf{x}_i)) + (1-y_i)\log(1 - h_{\boldsymbol{\beta}}(\mathbf{x}_i))\ ]$$ (we ignore any regularization term)

Recall that a neural network can be seen as an aggregation of logistic regressions. In a NN with **K neurons in the output layer** (no matter how many are inside), the cost function is

$$J(\mathbf{B}) = -(1/m) [\ \sum_{i=1}^{m} \sum_{k=1}^{K} y_i \log(h_{\mathbf{B}}(\mathbf{x}_i))_k + (1-y_i)\log(1 - (h_{\mathbf{B}}(\mathbf{x}_i))_k)\ ] \qquad \mathbf{B} = \{B^{(1)}, B^{(2)}, \ldots\}$$

which is again non-convex.

How can we compute the partial derivatives $\partial J / \partial B_{ij}^{(\ell)}$ at each step… and not die along the way? The task seems a bit (computationally) heavy...

Backpropagation

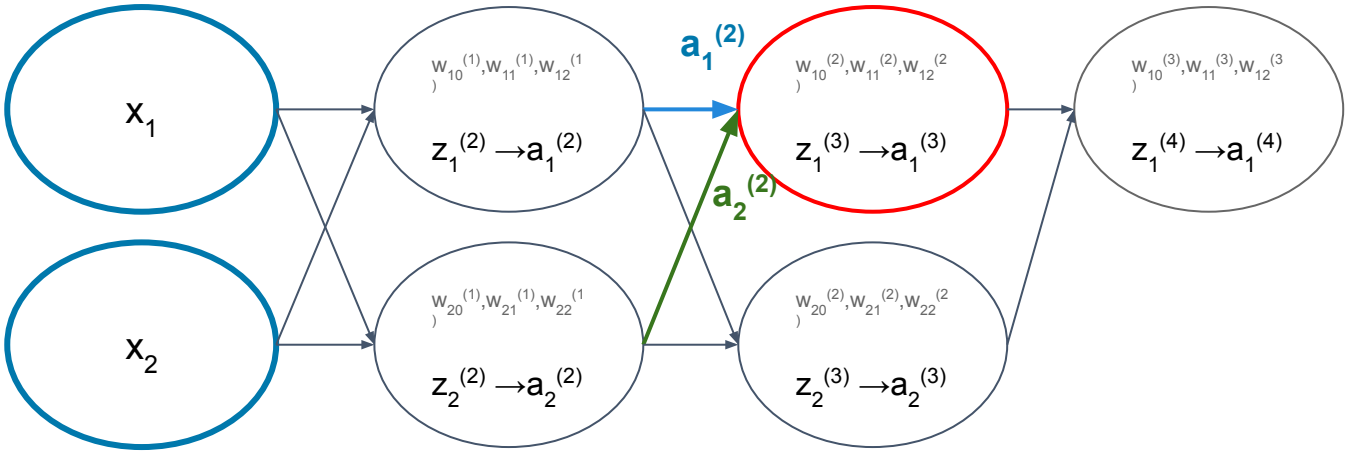
Backpropagation

Algorithm to compute the partial derivatives of the cost function with respect to each parameter.

- **Intuition**: compute the contribution of each neuron to the final error, and change its weights accordingly
  - Compute the contribution (*deltas*) of each neuron to the error of each example separately, and then accumulate over all examples to obtain the total contribution of each neuron to the total error.

Example: we first apply forward propagation to compute every $a_j^{(\ell)}$ and the output of the network $h_{\mathbf{w}}(\mathbf{x})$

$$z_1^{(3)} = w_{10}^{(2)} + w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)}$$
$$a_1^{(3)} = g(z_1^{(3)})$$



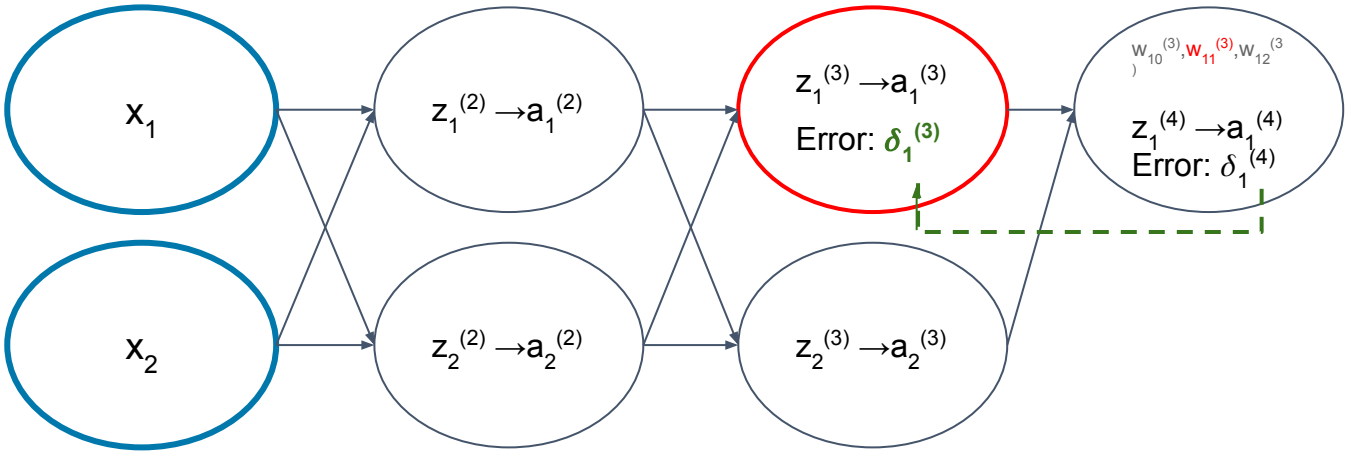
Backpropagation: contribution of a1(3) to the error

How wrong is $a_1^{(3)}$? In other words:
How much did neuron 1 of layer 3 contribute to the network error on a given example $(\mathbf{x}_i , y_i)$?

$\delta_1^{(4)} = y_i - a_1^{(4)}$
$\delta_1^{(3)} = w_{11}^{(3)} \delta_1^{(4)}$

because $a_1^{(3)}$ had contributed to $a_1^{(4)}$ with the term $a_1^{(3)} w_{11}^{(3)}$

(recall: $a_1^{(4)} = g(w_{10}^{(3)} + a_1^{(3)} w_{11}^{(3)} + a_2^{(3)} w_{12}^{(3)}))$

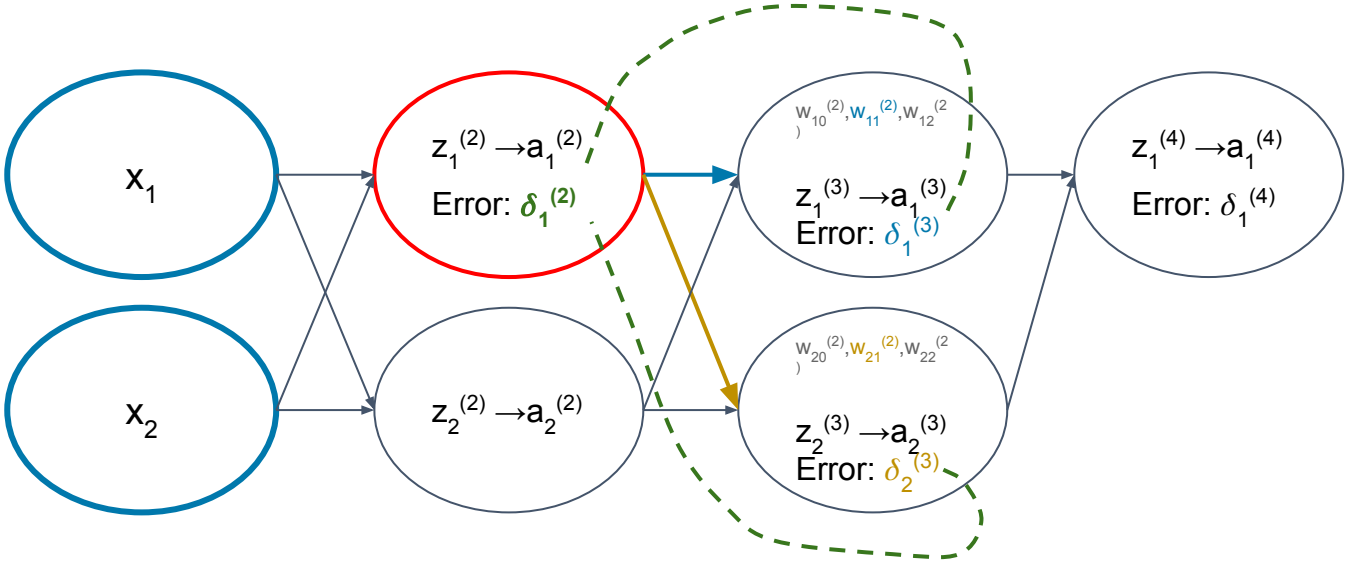


Backpropagation: contribution of a1(2) to the error

How wrong is $a_1^{(2)}$? In other words:
How much did neuron 1 of layer 2 contribute to the network error on a given example $(\mathbf{x}_i , y_i)$?

$\delta_1^{(2)} = w_{11}^{(2)}\delta_1^{(3)} + w_{21}^{(2)}\delta_2^{(3)}$ because $a_1^{(2)}$ contributed to

- $a_1^{(3)}$ with term $a_1^{(2)}w_{11}^{(2)}$ (recall: $a_1^{(3)} = g(w_{10}^{(2)} + a_1^{(2)}w_{11}^{(2)} + a_2^{(2)}w_{12}^{(2)})$)
- $a_2^{(3)}$ with term $a_1^{(2)}w_{21}^{(2)}$ (recall: $a_2^{(3)} = g(w_{10}^{(2)} + a_1^{(2)}w_{11}^{(2)} + a_2^{(2)}w_{12}^{(2)})$)



Backpropagation algorithm (complete)

INPUT: training set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$

Initialize $\Delta_{ij}^{(\ell)} \leftarrow 0$ for every i, j, ℓ

For $i = 1$ to m:

$a^{(1)} \leftarrow \mathbf{x}_i$

Perform forward propagation to compute every $a_j^{(\ell)}$ for $\ell = 1, 2, \ldots, L$ (recall the $a_j^{(L)}$ are the outputs of the network)

$\boldsymbol{\delta}^{(L)} \leftarrow a^{(L)} - y_i$ (errors of the output layer)

Compute the $\boldsymbol{\delta}^{(L-1)}, \boldsymbol{\delta}^{(L-2)}, \ldots, \boldsymbol{\delta}^{(2)}$ as explained before

$\Delta_{ij}^{(\ell)} \leftarrow \Delta_{ij}^{(\ell)} + a_j^{(\ell)} \delta_i^{(\ell+1)}$

$D_{ij}^{(\ell)} = (1/m) \Delta_{ij}^{(\ell)}$ (assuming no regularization)

Finally…….. the D's are the components of the gradient vector evaluated at the current values of the parameters:

$\partial J / \partial W_{ij}^{(\ell)} \mid_{Bij(\ell) = Bij(\ell)(t)} = D_{ij}^{(\ell)}$

and now we can use them to update the parameter values $\mathbf{W}^{(t)}$ to obtain $\mathbf{W}^{(t+1)}$, either as in gradient descent or any other optimization method

Stochastic Gradient Descent

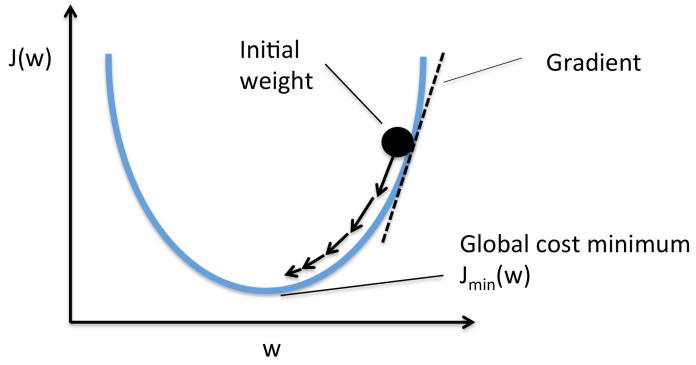
STOCHASTIC GRADIENT DESCENT

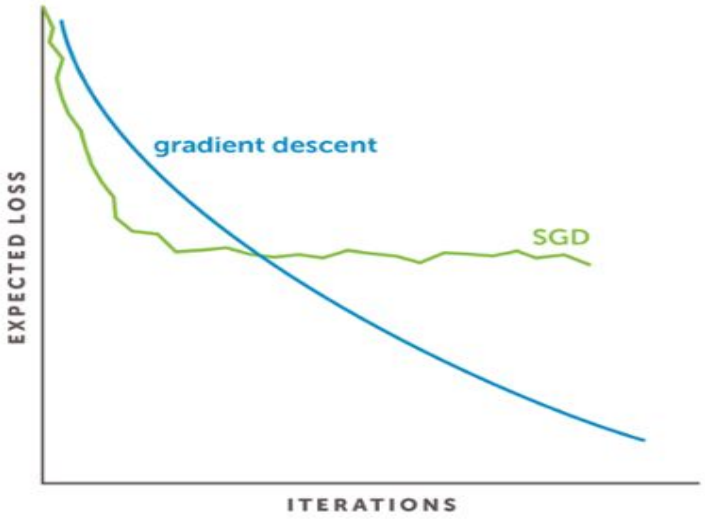
Gradient Descent

- Goes over the whole training set.
- Very expensive.
- There isn't an easy way to incorporate new data to training set.

$$\widehat{J}(W) = \frac{1}{m} \cdot \sum_i \mathcal{L}(\widehat{y}_i, y_i)$$

$$w^{t+1} \leftarrow w^t - \eta \cdot \nabla_w J(w^t)$$





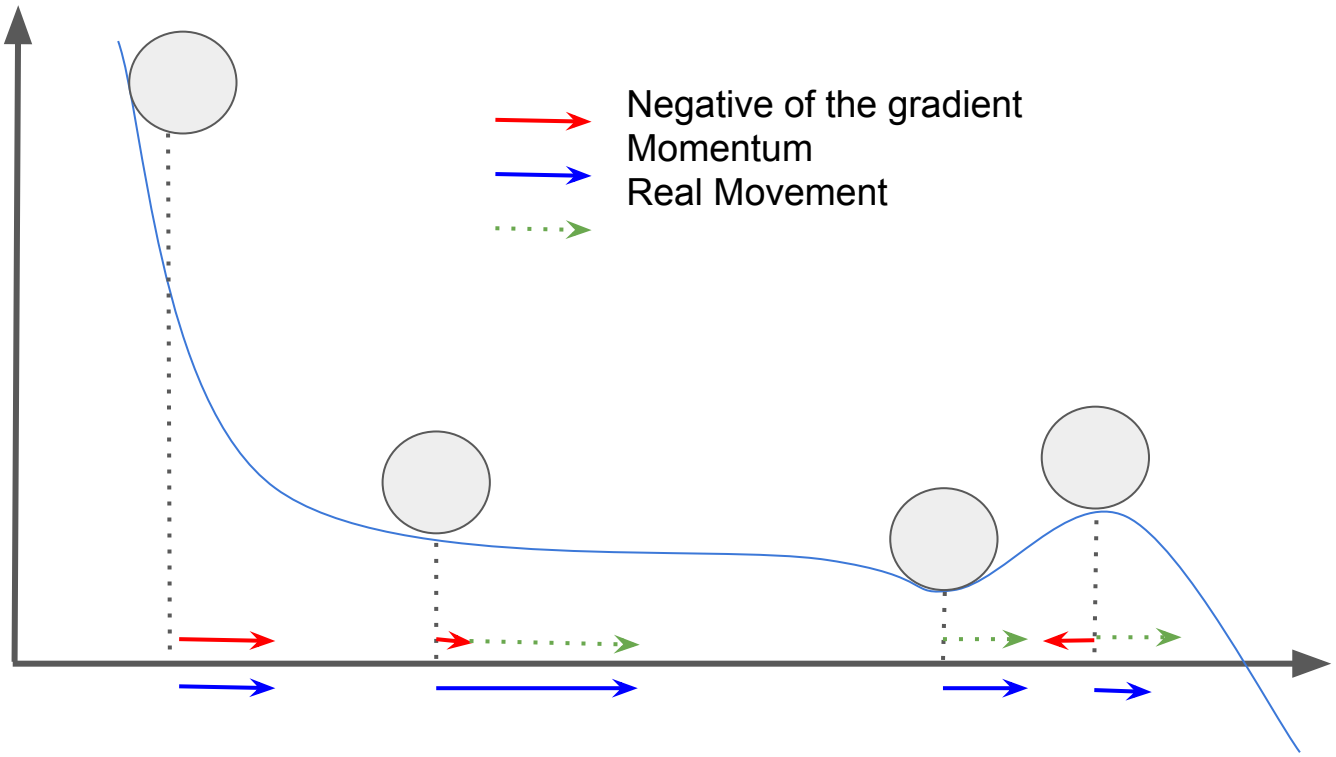
Stochastic Gradient Descent (SGD)

- Randomly sample a small number of examples (minibatch)
- Estimate cost function and gradient:

$$\widehat{J}(W) = \frac{1}{m} \cdot \sum_i \mathcal{L}(\widehat{y}_i, y_i)$$

- **Batch size:** Length of the minibatch
- **Iteration:** Every time we update the weights
- **Epoch:** One pass over the whole training set.
- k = 1 => online learning
- Small batches => regularization

MOMENTUM IDEA



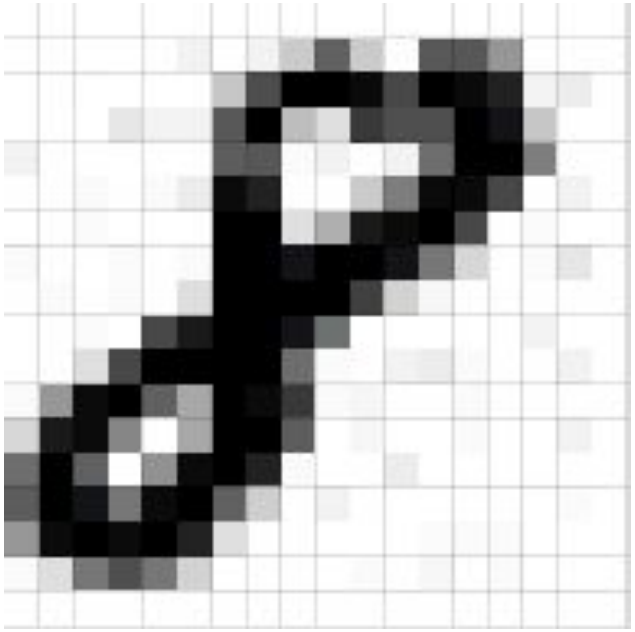
MOMENTUM IDEA

https://github.com/fermaat/afi_deep_learning_intro

https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html

Network Examples

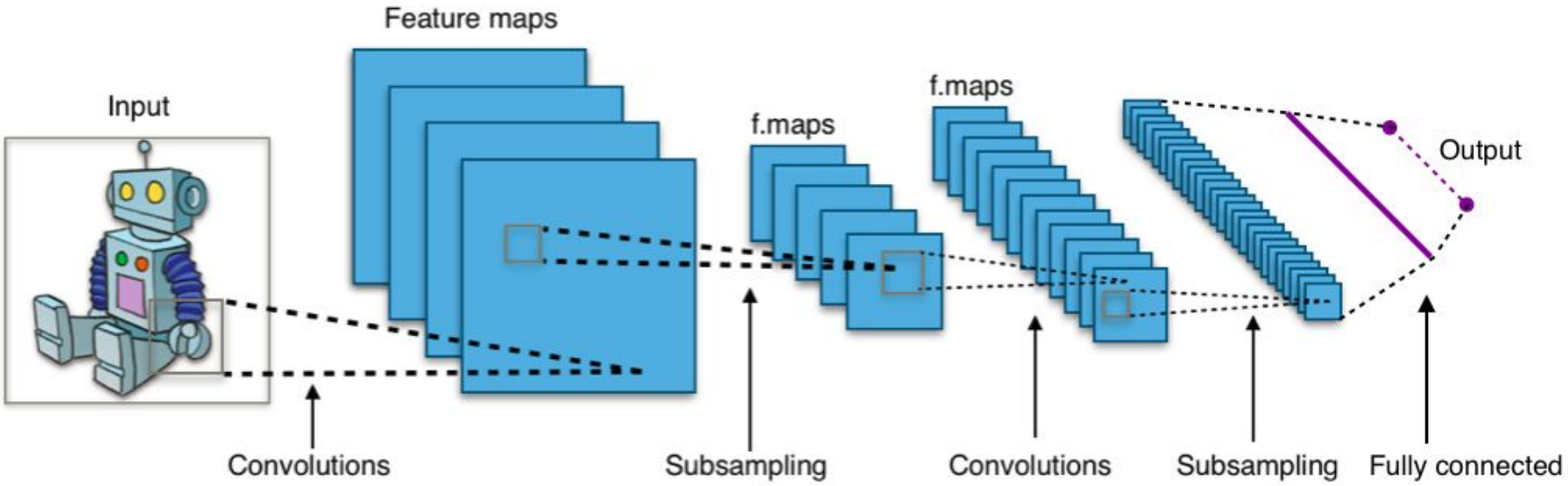
IMAGE DATA



- Images are composed by pixels.
- Grayscale images can be seen as matrices

- Coloured images are usually represented as mixes of three colours: Red, Green and Blue
- Each one can be seen as a grayscale-like filter.

CONVOLUTIONAL NEURAL NETWORKS



- Exceed on feature extraction
- Composed of a feature extractor (mainly based on convolutions) and a classifier
- Convolutions are linear transformations of the data, applied through all the filters on each stage
- Subsampling processes are performed to improve generalization
- Widely used on image classification, but not strictly

CONVOLUTION EXAMPLES

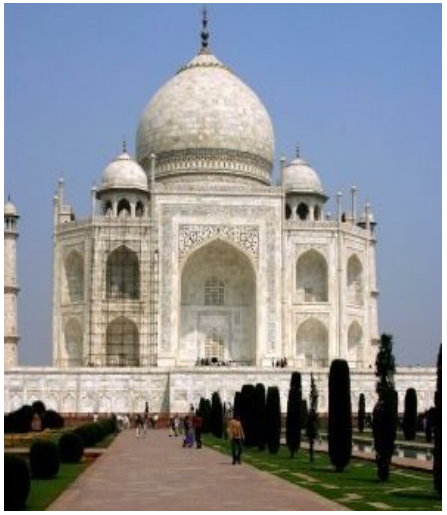
35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

×

0	1	0
0	0	0
0	0	0

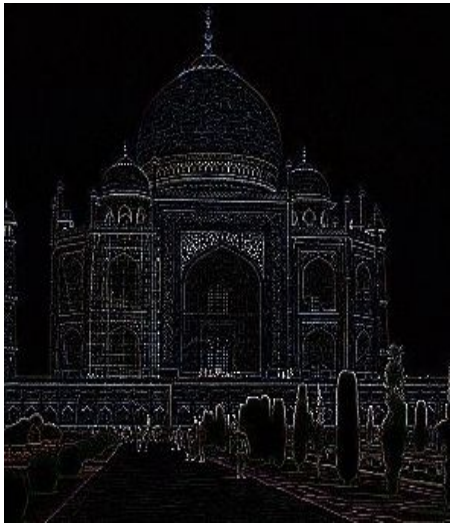
=

		42		



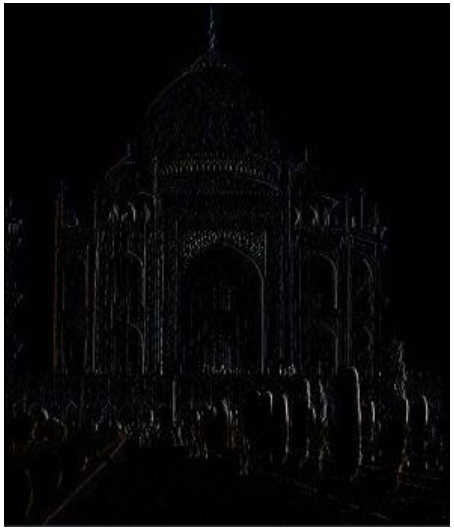
Edge Detection

0	1	0	
1	-4	1	
0	1	0	

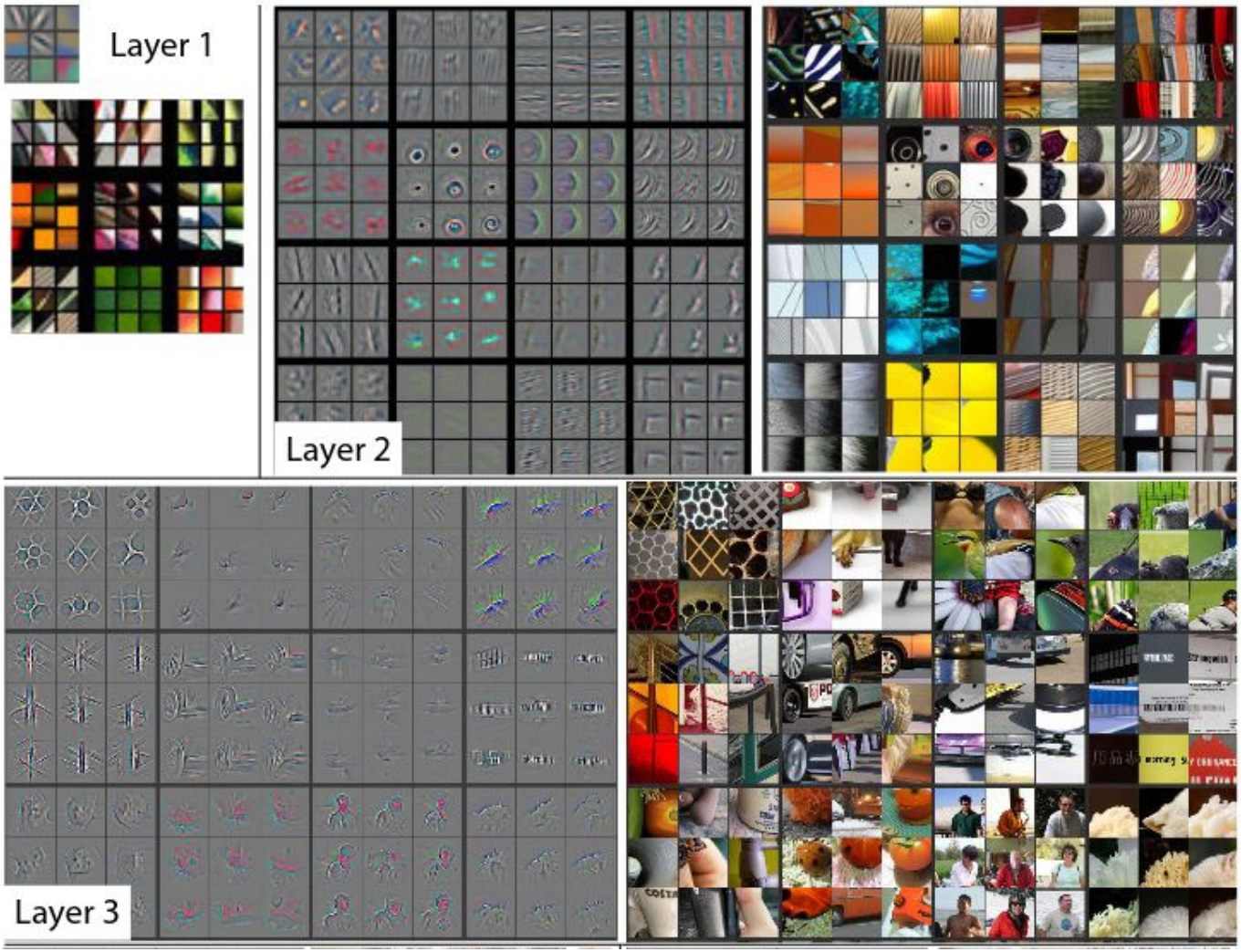


Edge Enhance (right)

0	0	0	
-1	1	0	
0	0	0	



Inside the layers



Zailer and Fergus. *Visualizing and Understanding Convolutional Networks*

Inside the layers

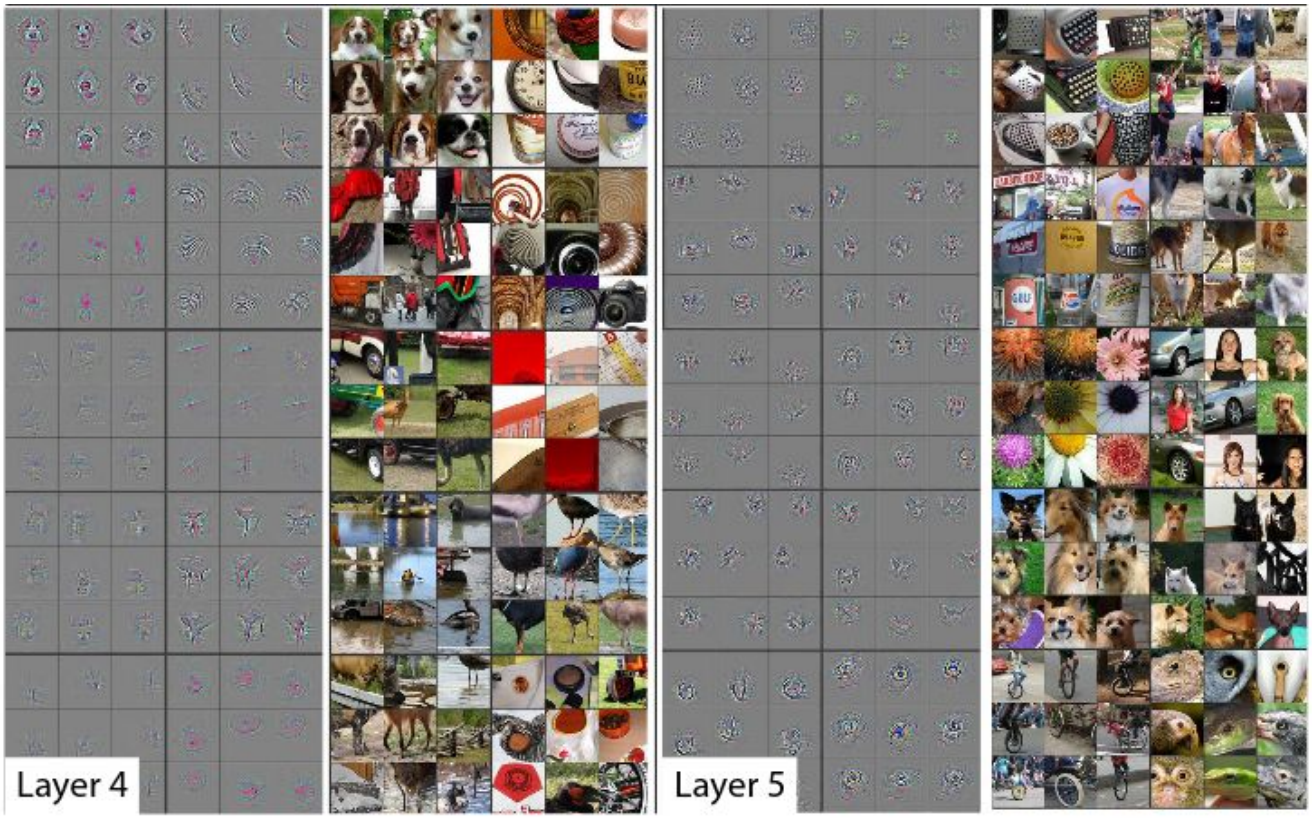
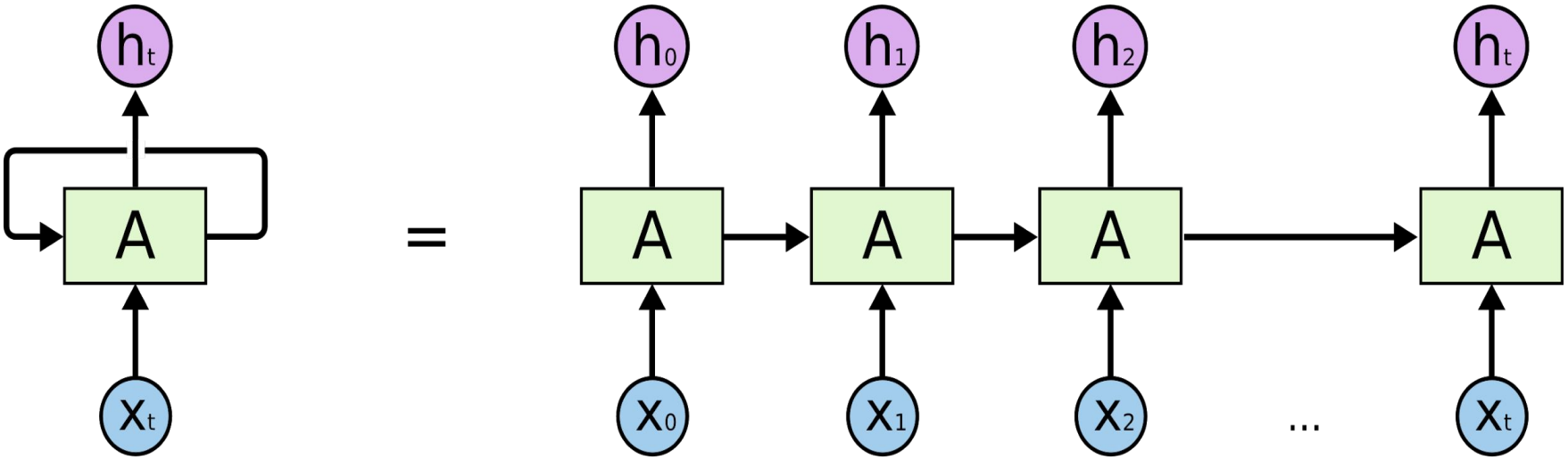


Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

Zailer and Fergus. *Visualizing and Understanding Convolutional Networks*

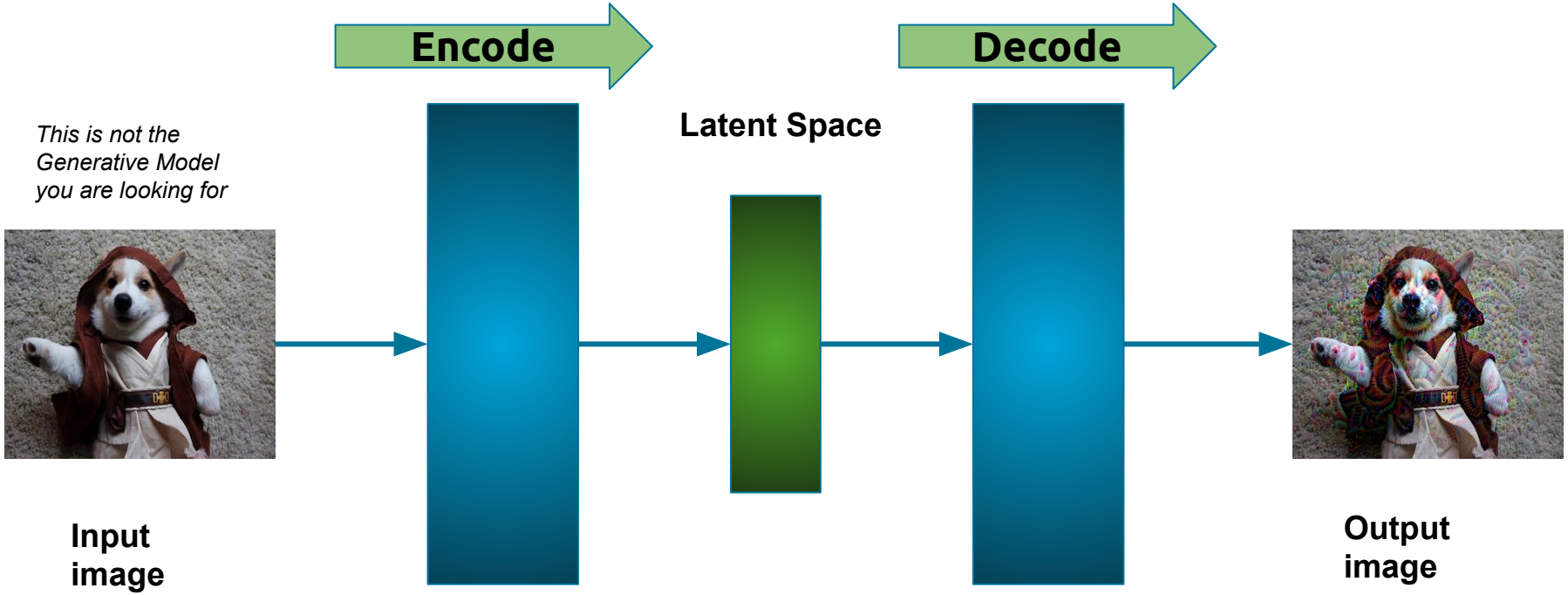
RECURRENT NEURAL NETWORKS



- Neural Networks with recurrent connections, specialized in processing sequential data.
- Recurrent connections allows a 'memory' of previous inputs.
- Can scale to long sequences (variable length), not practical for other types of nets.
- Same parameters for every timestep (t) => generalize

RNN images by Christopher Olah

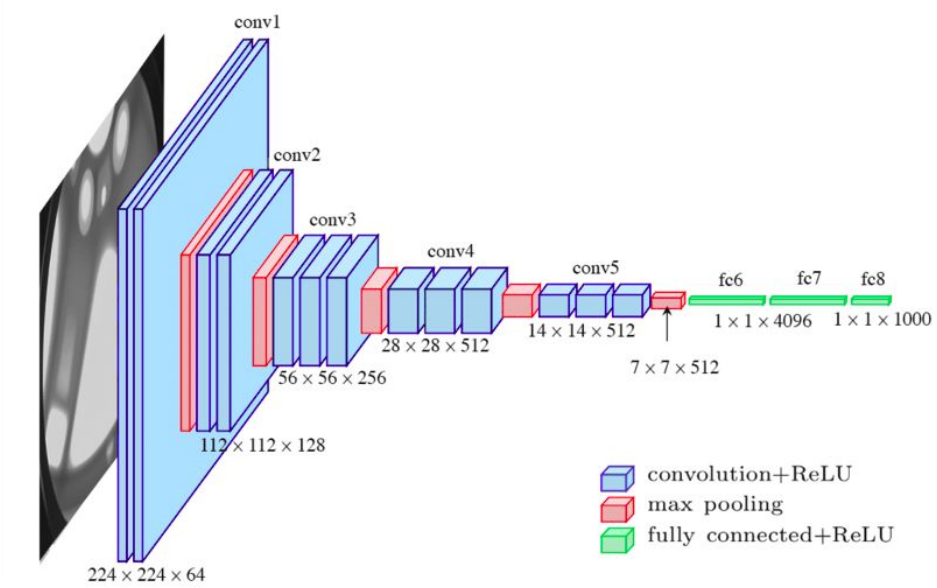
AUTOENCODERS

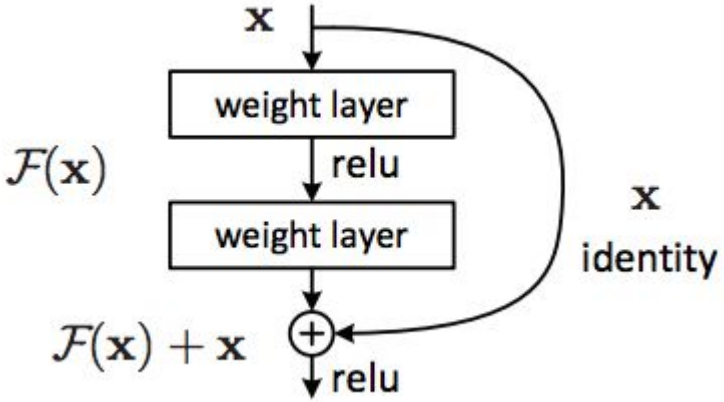


- It tries to predict **x** from **x**, but no labels are needed.
- The idea is learning an approximation of the identity function.
- Along the way, some restrictions are placed: typically the hidden layers *compress* the data.
-

- Encoder and decoder can share the same (inversed) structure or be different.
- Each one can have its own depth (number of layers) and complexity.
- The original input is represented at the output, even if it comes from noisy or corrupted data.

Common Architecture tricks





VGG16 Architecture

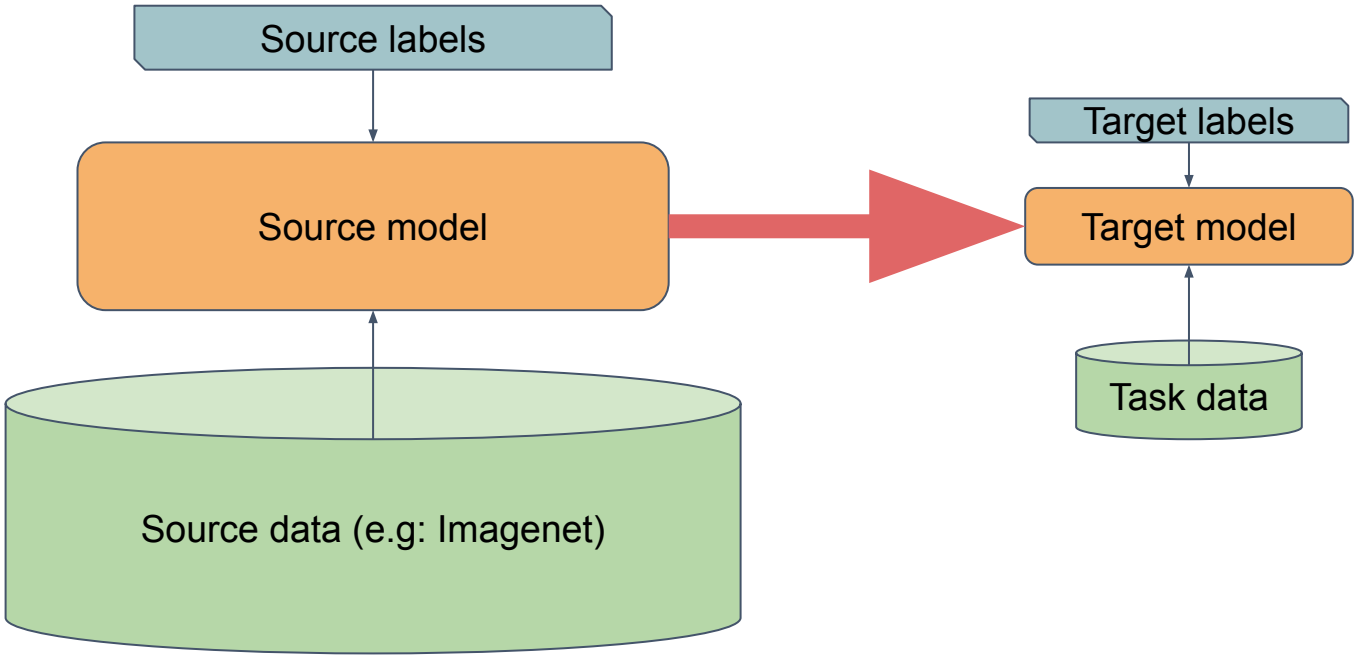
- Not a very deep network, but a very successful one
- its name is due to the 16 layers (excluding pooling)
- Widely used on transfer learning

Residuals

- A clever way to keep previous information on new layers
- prevents the network to learn identity function
- Has a different *dense* version, which adds concatenation

Ferguson et al. *Automatic localization of casting defects with convolutional neural networks*

Transfer Learning



Instead of training a Deep Neural Net from scratch for your task:

- Take a network trained on a different domain for a different task
- Adapt it for your domain and your task

Variations:

- Same domain, different task
- Different domain, same task

Use Cases and Techniques

GENERATIVE MODELS

Why are generative models harder to create than discriminative models?

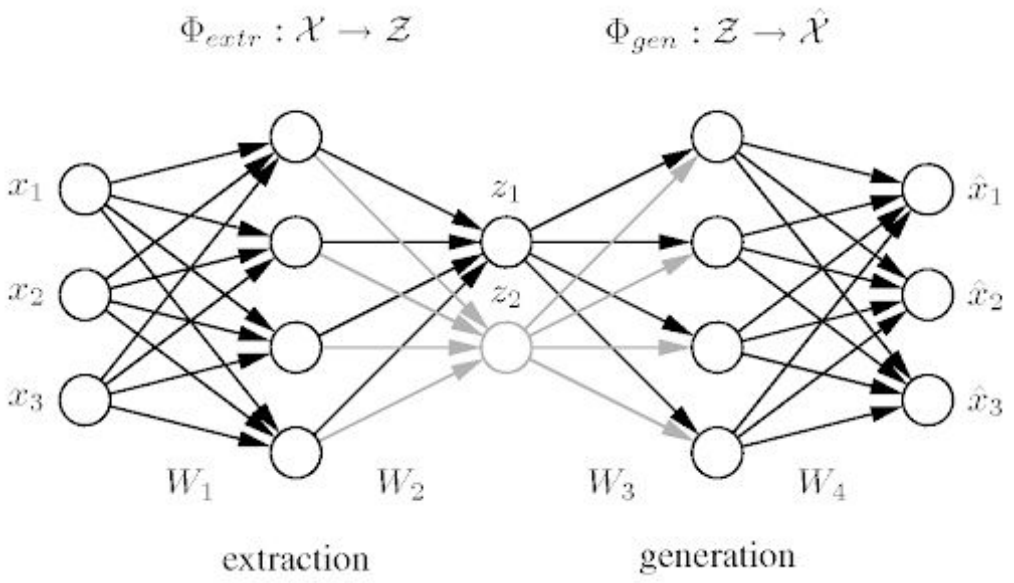
3 Answers

Ian Goodfellow, I invented generative adversarial networks.
Answered Apr 10, 2016 · Upvoted by Amir H. Jadidinejad, Expert in Machine Learning, Assistant Professor @QIAU and Viresh Ranjan

Can you look at a painting and recognize it as being the Mona Lisa? You probably can. That's discriminative modeling. Can you paint the Mona Lisa yourself? You probably can't. That's generative modeling.

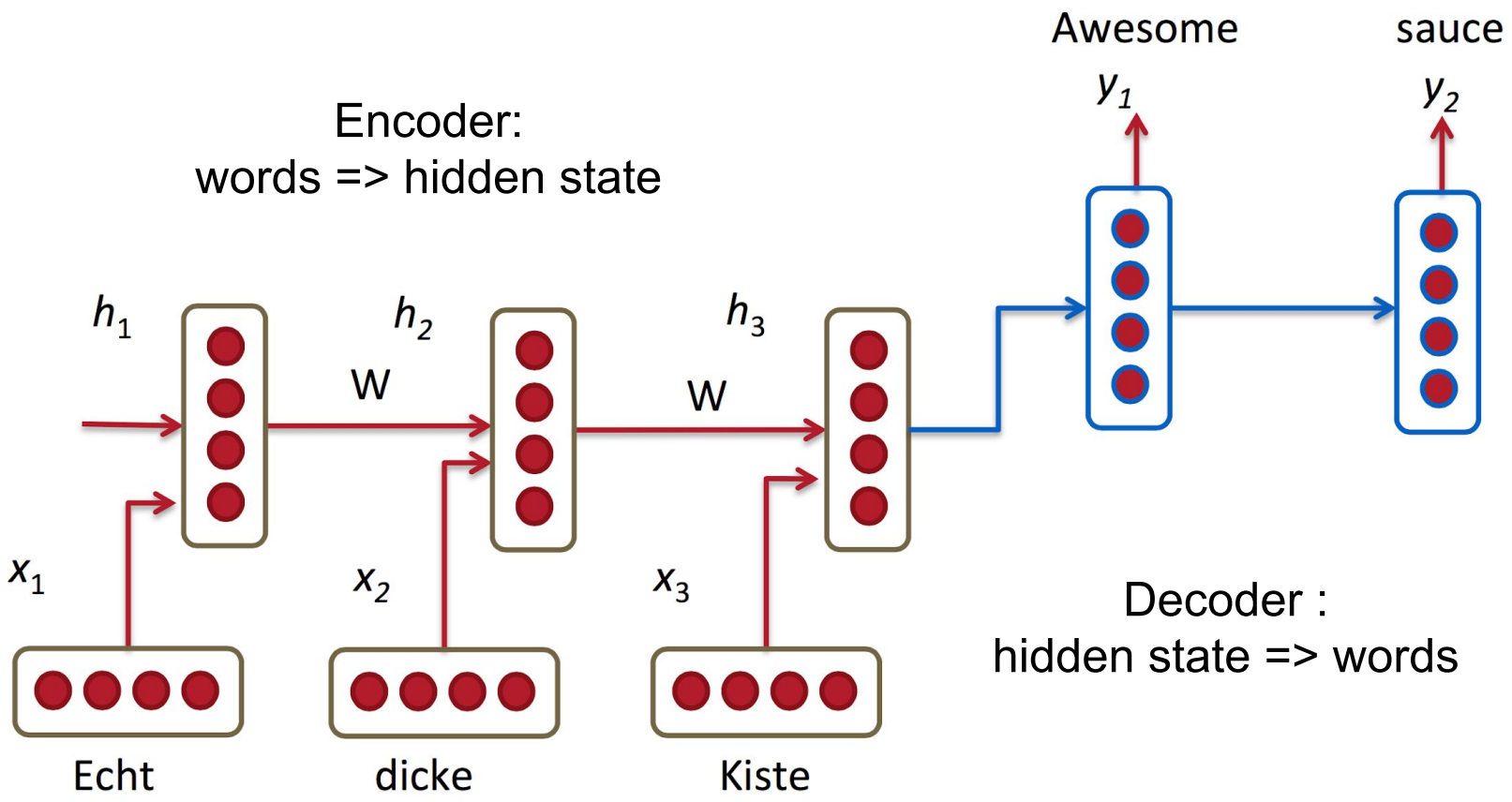
- Generate potentially unfeasible examples for *Reinforcement Learning*
- Denoising/Pretraining
- Structured prediction exploration in RL
- Entirely plausible generation of images to depict image/video
- Feature understanding

Generative neural nets (II)



- Musenet, by openAI. **Music generation**
- GPT2 online demo. A transformer structured **text generator**
- A living Dalí, created by his museum in florida. **Image/Video generation**

MACHINE TRANSLATION



Hidden states are not entirely universal languages!!

Deep Learning on Images

- **Image Segmentation**
- **Fashion Matching**

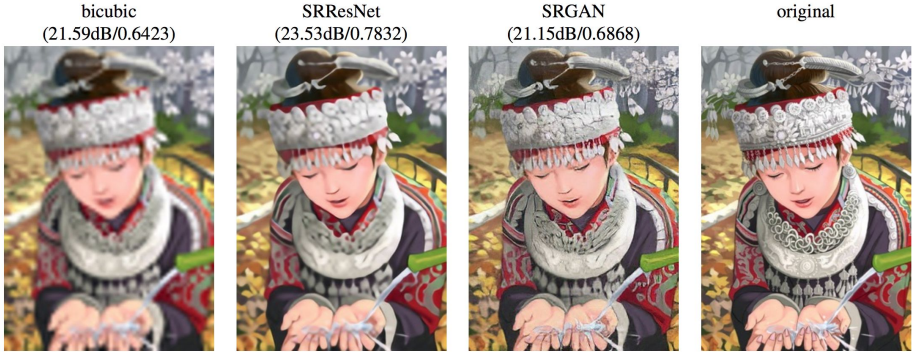


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

- **Super Resolution generated images**
- **Denoising**

Ledig et al. *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*

Bonus track: about ethics

Some Ethical concerns

"That's not 20/20 hindsight. The scale of this problem was significant and it was already apparent."

NEWS / ROHINGYA

UN: Facebook had a 'role' in Rohingya genocide

'I'm afraid that Facebook has now turned into a beast.'

13 Mar 2018

- Ethical issues are complex and don't have clear answers
- It is good to think about how you'd handle a situation before you are on it
- Consider the next step impact of what you do
- Be responsible for what you program
- Volkswagen engineer sentenced to prison for creating software to cheat on emissions test (he was following the chain of command)
- Algorithms are used differently than human decision makers: Algorithms are more likely to be implemented without appeals process in place

Jeremy Howard and Rachel Thomas.
On ethics/TED talk