Course: Computer Vision

Unit 4: Object recognition

Introduction to Neural Networks for Object Recognition

Luis Baumela

2018

Universidad Politécnica de Madrid

Introduction to Neural Networks

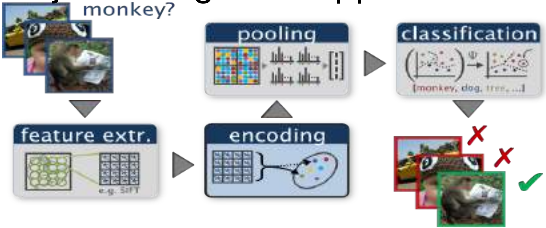
1. Introduction and motivation
    - Relevance
    - Neurons and neural nets
    - Historical approach

2. Linear classification
    - Multinomial logistic regression
    - Error functions. Cross Entropy
    - Optimizing model parameters

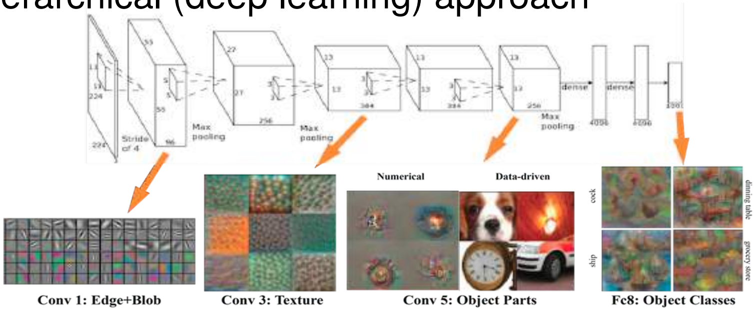
3. Neural Nets
    - Multilayer NNs
    - Training: backpropagation
    - Importance of depth

Introduction

- Standard (shallow) object recognition approach

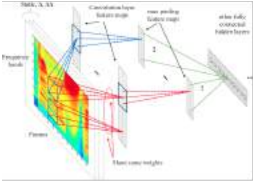


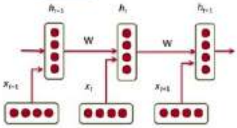
- Hierarchical (deep learning) approach



Introduction

- Deep learning is the state of the art for
  - Computer vision
  - Speech recognition
  - Natural language



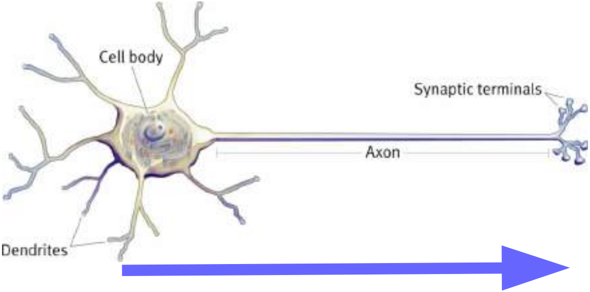


- Widely used in industry (Google, IBM, …). Why? Excels in presence of
  - Complex problems
  - Lots of data and
  - computing power

- Aplicable to wide spectrum of data & problems

Introduction. Biological motivation

- Biological neuron



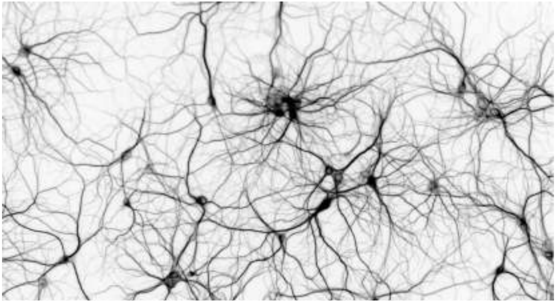
Flow of information

Neuron computation:

- Dendrites collect input from other neurons
- The neuron body adds these inputs to obtain an activation level
- The axon trasnsmits, through synaptic terminals, the neuron output
- The effectiveness of synapses can be changed

Introduction. Biological motivation

- Brain structure

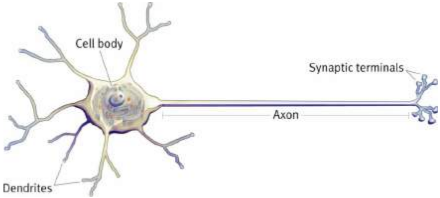


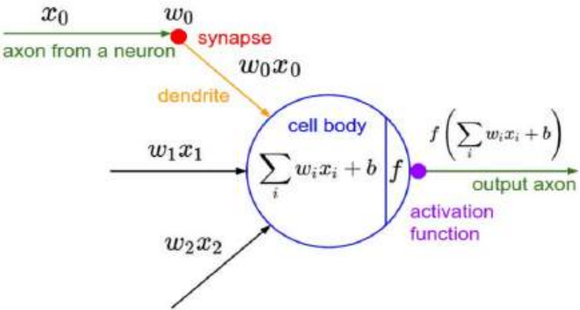
Networks of neurons:

- Each neuron receives input from other neurons
- The effect of each input is controlled by the synaptic weight
- Synaptic weights adapt so that the network performs useful computations
- We have about 10^{11} neurons each with 10^4 synapses

Introduction. Biological motivation

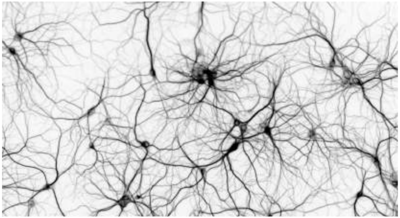
- Computational neuron



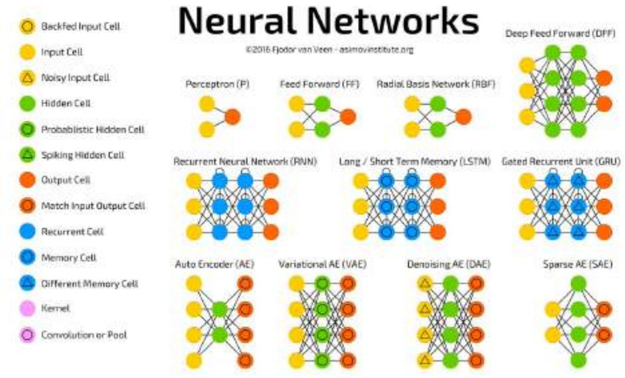


Introduction. Biological motivation

- Neural network



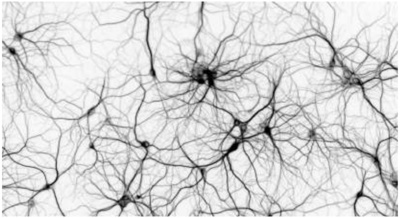
Models



asimovinstitute.org

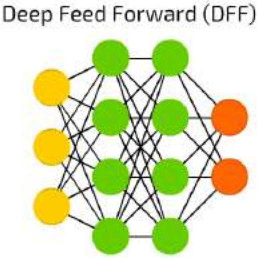
Introduction. Biological motivation

- Neural network

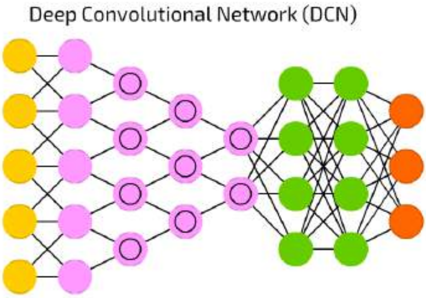


We are interested in:

Feedforward model

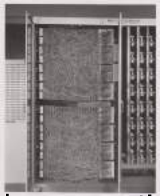


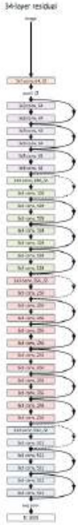
Convolutional model

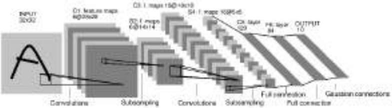


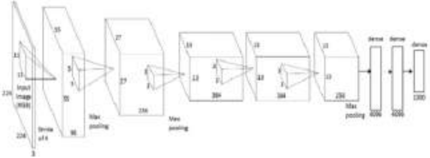
Introduction. Historical overview

- Deep Learning: the “third emergence” of neural nets
  - 1957. F. Rosemblat’s Perceptron
  - 1969. M. Minsky, “Perceptrons”
  - 1980’s.
    - P. Werbos, G. Hinton, D. Rumerhart, Backpropagation algorithm
    - K. Fukushima Neocognitron
  - 1990s. Y. LeCun Convolutional Neural Networks
  - 2012. A. Krizhevsky. AlexNet
  - 2015 K. He. Deep Residual Networks









Introduction to Neural Networks

1. Introduction and motivation
    - Relevance
    - Neurons and neural nets
    - Historial approach

2. Linear classification
    - **Multinomial logistic regression**
    - **Error functions. Cross Entropy**
    - **Optimizing model parameters**

3. Neural Nets
    - Multilayer NNs
    - Training: backpropagation
    - Importance of depth

Linear classification

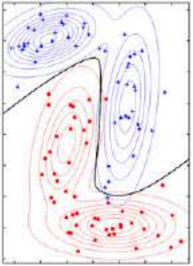
- Classification

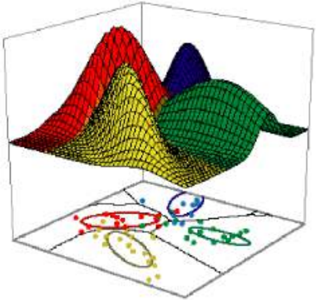
  The assignment of a label to a given input data $\mathbf{x} \in \mathbb{R}^m$.

  We represent the class label with a set of discrete values

  $$\{c_1, \ldots, c_k\}$$

  So, classification may be seen as the asignment of $\mathbf{x}$ to a region in the space of features:





Linear classification

- Classification
  The assignment of a label to a given input data $\mathbf{x} \in \mathbb{R}^m$.
  We represent the class label with a set of discrete values
  $$\{c_1, \dots, c_l\}$$
  So, classification may be seen as the asignment of $\mathbf{x}$ to a region in the space of features:
- Supervised classification
  We have a set of labels $C = \{c_1, \dots, c_l\}$, and a set of training data with labels $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^m, \; y_i \in C$
  Our goal is to learn a model g such that, given a data with unknown label, $\mathbf{x}$, we can estimate a label asigment
  $$c = g(\mathbf{x}|\mathcal{D})$$

Linear classification

- Multinomial logistic classification

Let's assume $\mathcal{D} = \{(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)\}, \mathbf{x}_i \in \mathbb{R}^m,\ y_i \in \{c_1 \dots c_l\}$

We want to find a set of **weights** $\mathtt{W}_{l \times m}$ and **biases** $\mathbf{b}_{l \times 1}$

$$\mathbf{z}_i = \mathbf{g}(\mathbf{x}_i) = \mathtt{W}\,\mathbf{x}_i + \mathbf{b} = \begin{Bmatrix} 2.8 & & \mathbf{x} \in c_1 \\ \vdots & \text{if} & \vdots \\ 0.1 & & \mathbf{x} \notin c_l \end{Bmatrix}$$

We also want these scores, $\mathbf{z}_i = z_i^j, j = 1 \dots l$, to be probabilities

$$\mathbf{S}(\mathbf{z}_i) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{l} e^{z_i^j}} = \begin{cases} 0.9 \\ \vdots \\ 0.03 \end{cases}, \text{ where } \sum_j S(z^j) = 1$$

Linear classification

- Labels: "one-hot" encoding

  How do we code label values, $y_i \in \{c_1 \dots c_l\}$?

$$\mathbf{S}(\mathbf{z}_i) = \begin{bmatrix} 0.9 \\ \vdots \\ 0.03 \end{bmatrix} \in \mathbb{R}^l \quad \mathbf{y}_i = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^l \Leftrightarrow y_i = c_1$$

- Error function: **average cross entropy** (ACE)

  How do we compare classifier responses with labels?

$$d(\mathbf{S}(\mathbf{g}(\mathbf{x}_i)), \mathbf{y}_i) = -\sum_{j=1}^{l} y_i^j \log S^j(\mathbf{g}(\mathbf{x}_i))$$

$$d(\mathbf{S}(\mathbf{z}_i), \mathbf{y}_i) = -\log S^{y_i}(\mathbf{g}(\mathbf{x}_i))$$

$$\boxed{\mathcal{L}(\mathbf{g}, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} d(\mathbf{S}(\mathbf{z}_i), \mathbf{y}_i) = -\frac{1}{n}\sum_{i=1}^{n} \log S^{y_i}(\mathbf{g}(\mathbf{x}_i))}$$

Linear classification

- Why cross-entropy?

  Other error measures: classification error (CE), mean squared error (MSE).

  Let us analyze one example:

```
computed       | targets              | correct?
-----------------------------------------------
0.3  0.3  0.4  | 0   0   1 (democrat)   | yes
0.3  0.4  0.3  | 0   1   0 (republican) | yes
0.1  0.2  0.7  | 1   0   0 (other)      | no
```

CE = 0.33
MSE = 0.81
ACE = -2 log0.4 – log0.1 = 1.38

```
computed       | targets              | correct?
-----------------------------------------------
0.1  0.2  0.7  | 0   0   1 (democrat)   | yes
0.1  0.7  0.2  | 0   1   0 (republican) | yes
0.3  0.4  0.3  | 1   0   0 (other)      | no
```

CE= 0.33
MSE = 0.34
ACE = -2 log0.7 – log0.3 = 0.64

  - CE: too coarse error measure
  - MSE: better than CE, but emphasizes errors too much
  - ACE: best approach!

But not conmutative!

https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/

Linear classification

- Training

  Minimize ACE loss

$$\arg\min_{\mathtt{W},\mathbf{b}} \mathcal{L}(\mathbf{g}, \mathcal{D}) = \arg\min_{\mathtt{W},\mathbf{b}} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log S^{y_i}(\mathtt{W}\mathbf{x}_i + \mathbf{b}) \right\}$$

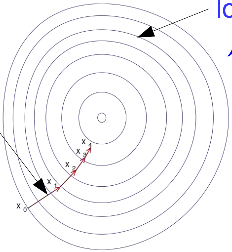
  Numerical optimization. **Gradient descent**

loss function

$\mathcal{L}(\mathbf{g}, \mathcal{D})$

descent step

$-\alpha \nabla_{\mathtt{W},\mathbf{b}} \mathcal{L}(\mathbf{g}, \mathcal{D})$



$$\mathtt{W}_{k+1} = \mathtt{W}_k - \alpha \nabla_{\mathtt{W}} \mathcal{L}(\mathbf{g}, \mathcal{D})$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \alpha \nabla_{\mathbf{b}} \mathcal{L}(\mathbf{g}, \mathcal{D})$$

Linear classification

- Training. **Stochastic gradient descent**

  At each step compute

$$\mathtt{W}_{k+1} = \mathtt{W}_k - \alpha \nabla_{\mathtt{W}} \mathcal{L}(\mathbf{g}, \mathcal{D}_u)$$
$$\mathbf{b}_{k+1} = \mathbf{b}_k - \alpha \nabla_{\mathbf{b}} \mathcal{L}(\mathbf{g}, \mathcal{D}_u)$$

  where $\mathcal{D}_u, u = 1 \ldots U$, is a U-fold partition of $\mathcal{D}$, i.e. $\{\mathcal{D}_i \cap \mathcal{D}_j\} = \Phi$, $\{\mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_U\} = \mathcal{D}$.

  The sample distribution in each $\mathcal{D}_u$ should be similar to that in $\mathcal{D}$.

  When $\mathcal{D}$ is very large and redundant SGD has several advantages:

  - Computationally much more efficent
  - May generalize better, since in each iteration it only uses part of the data set.

  However, it is an approximation!

Linear classification

- Training. Helping the optimization
  - Parameters initialization
    $$w_0 \sim \mathcal{N}(0, \sigma^2) \qquad b_0 \sim \mathcal{N}(0, \sigma^2)$$
    for a small σ
  - Input data nomalization
    $$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{n}\sum_j \mathbf{x}_j$$
  - Batch size, card($\mathcal{D}_u$)
    Largest that fits in GPU memory!

Introduction to Neural Networks

1. Introduction and motivation
    - Relevance
    - Neurons and neural nets
    - Historial approach

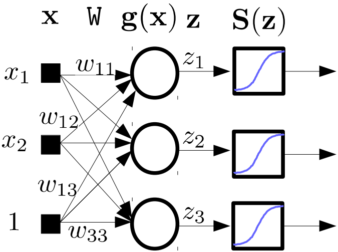
2. Linear classification
    - Multinomial logistic regression
    - Error functions. Cross Entropy
    - Optimizing model parameters

3. Neural Nets
- **Multilayer NNs**
- **Training: backpropagation**
- **Importance of depth**

Multilayer Neural Network

- Single-layer neural network

The multinomial logistic regressor is a single-layer NN



Response function

$$\mathbf{S}(\mathbf{g}(\mathbf{x})) = \mathbf{S}(\mathtt{W}\mathbf{x})$$

Training

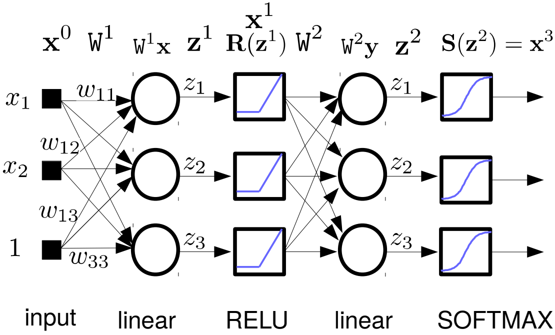
$$\arg\min_{\mathtt{W}} \mathcal{L}(\mathtt{W}, \mathcal{D})$$

Linear models:

- Are very stable
- Constant derivative
- May be efficiently computed in GPUs
- Only solve linearly separable problems
- Add more linear layers? **useless !** $\mathbf{S}(\mathtt{W}_1\mathtt{W}_2\mathtt{W}_3\mathbf{x}) = \mathbf{S}(\mathtt{W}'\mathbf{x})$

Multilayer Neural Network

- Multi-layer neural network



Response function

$$\mathbf{S}(\mathtt{W}^2\mathbf{R}(\mathtt{W}^1\mathbf{x}))$$

Training

$$\arg\min_{[\mathtt{W}]} \mathcal{L}([\mathtt{W}], \mathcal{D})$$

where $[\mathtt{W}] = [\mathtt{W}_1 \ldots \mathtt{W}_s]$

Neural network

- Solve non-linearly separable problems
- May be efficiently computed in GPUs
- Stack many layers. Hierarchical representation.
- How do we train it?

Multilayer Neural Network

- **Error backpropagation**

Gradient descent approach to train a NN solving

$\arg\min_{[\mathtt{W}]} \mathcal{L}([\mathtt{W}], \mathcal{D})$ through the iterative scheme

$$[\mathtt{W}]_{k+1} = [\mathtt{W}]_k - \alpha \nabla_{[\mathtt{W}]} \mathcal{L}([\mathtt{W}], \mathcal{D})$$

Steps:

- Compute $\nabla_{[\mathtt{W}]} \mathcal{L}([\mathtt{W}], \mathcal{D})$
- Update network parameters $[\mathtt{W}]$

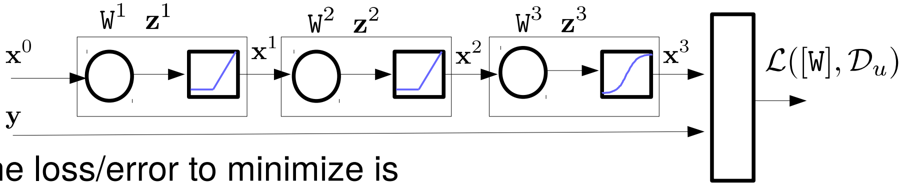
Provides an efficient approach to estimate $\nabla_{[\mathtt{W}]} \mathcal{L}([\mathtt{W}], \mathcal{D})$

- All necessary information is available locally
- Scales linearly with the number of parameters

Multilayer Neural Network

- **Error backpropagation**

For the network



the loss/error to minimize is

$$\mathcal{L}([\mathtt{W}], \mathcal{D}_u) = \frac{1}{|\mathcal{D}_u|} \sum_{\forall i \in \mathcal{D}_u} d(\mathbf{x}_i^3, \mathbf{y}_i) = \frac{-1}{|\mathcal{D}_u|} \sum_{\forall i \in \mathcal{D}_u} \log(\mathbf{x}_i^3)^{y_i}$$

At each iteration we compute

$$\mathtt{W}_{k+1}^i = \mathtt{W}_k^i - \alpha \nabla_{\mathtt{W}^i} \mathcal{L}$$

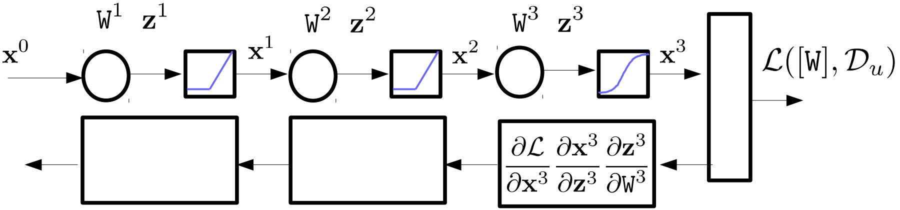
Steps:

1. Forward pass $\mathbf{x}_i \in \mathcal{D}_u$ to the net and compute $\mathcal{L}([\mathtt{W}], \mathcal{D}_u)$
2. Back propagate the error $\mathcal{L}([\mathtt{W}], \mathcal{D}_u)$ and compute $\nabla_{\mathtt{W}^i} \mathcal{L}$
3. Estimate $\mathtt{W}_{k+1}^i$

Multilayer Neural Network

- **Error backpropagation**

Estimate $\nabla_{\mathtt{W}^3}\mathcal{L}$



From the chain rule of derivation $\nabla_{\mathtt{W}^i}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathtt{W}^i} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^i}\frac{\partial \mathbf{x}^i}{\partial \mathbf{z}^i}\frac{\partial \mathbf{z}^i}{\partial \mathtt{W}^i}$

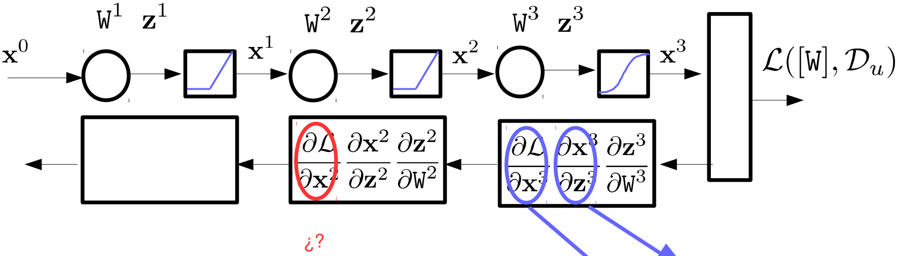
estimating $\nabla_{\mathtt{W}^3}\mathcal{L}$ is trivial

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^3} = \frac{1}{|\mathcal{D}_u|}\sum_{\forall i \in \mathcal{D}_u}\frac{\partial \log(\mathbf{x}_i^3)^{y_i}}{\partial \mathbf{x}^3}; \quad \frac{\partial \mathbf{x}^3}{\partial \mathbf{z}^3} = \frac{\partial \mathbf{S}(\mathbf{z}^3)}{\partial \mathbf{z}^3}; \qquad \frac{\partial \mathbf{z}^3}{\partial \mathtt{W}^3} = \frac{\partial \mathtt{W}^3\mathbf{x}^2}{\partial \mathtt{W}^3} = \mathbf{x}^2$$

Multilayer Neural Network

- **Error backpropagation**

Estimate $\nabla_{\mathtt{W}^2}\mathcal{L}$



$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}^2}=\frac{\partial\mathcal{L}}{\partial\mathbf{x}^3}\frac{\partial\mathbf{x}^3}{\partial\mathbf{z}^3}\frac{\partial\mathbf{z}^3}{\partial\mathbf{x}^2}=\frac{\partial\mathcal{L}}{\partial\mathbf{x}^3}\frac{\partial\mathbf{x}^3}{\partial\mathbf{z}^3}\mathtt{W}^3$$

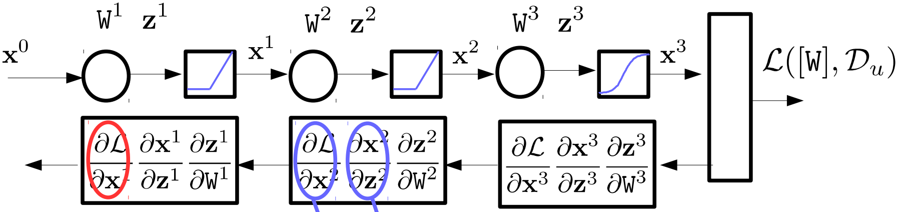
$$\frac{\partial\mathbf{x}^2}{\partial\mathbf{z}^2}=\frac{\partial\mathbf{R}(\mathbf{z}^2)}{\partial\mathbf{z}^2}$$

$$\frac{\partial\mathbf{z}^2}{\partial\mathtt{W}^2}=\frac{\partial\mathtt{W}^2\mathbf{x}^1}{\partial\mathtt{W}^2}=\mathbf{x}^1$$

Multilayer Neural Network

- **Error backpropagation**

Estimate $\nabla_{\mathtt{W}^1}\mathcal{L}$



¿?

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}^2} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}^3}\frac{\partial\mathbf{x}^3}{\partial\mathbf{z}^3}\mathtt{W}^3$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}^1} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}^2}\frac{\partial\mathbf{x}^2}{\partial\mathbf{z}^2}\frac{\partial\mathbf{z}^2}{\partial\mathbf{x}^1} = \frac{\partial\mathcal{L}}{\partial\mathbf{x}^2}\frac{\partial\mathbf{x}^2}{\partial\mathbf{z}^2}\mathtt{W}^2$$

$$\frac{\partial\mathbf{x}^1}{\partial\mathbf{z}^1} = \frac{\partial\mathbf{R}(\mathbf{z}^1)}{\partial\mathbf{z}^1}; \qquad \frac{\partial\mathbf{z}^1}{\partial\mathtt{W}^1} = \frac{\partial\mathtt{W}^1\mathbf{x}^0}{\partial\mathtt{W}^1} = \mathbf{x}^0$$

Multilayer Neural Network

- The importance of depth

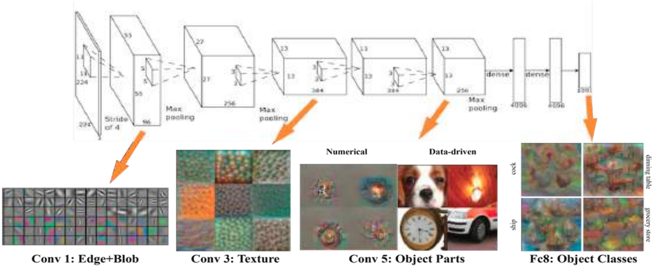
Deep learning builds models of the world around us

- complex enough to represent real world situations,
- with large (but always limited) amounts of data,

that require strong regularization techniques.

Deep learning provides means of regularization other than smoothing:

- Distributed data representation
- Deep hierarchical representation



A Deep Neural net can discover features independently of each other that better generalize to unseen samples, hence requiring less training data.