Apprentissage, réseaux de neurones et modèles graphiques (RCP209)
Neural Networks and Deep Learning

Nicolas Thome
Prenom.Nom@cnam.fr
`http://cedric.cnam.fr/vertigo/Cours/ml2/`

Département Informatique
Conservatoire Nationnal des Arts et Métiers (Cnam)

8 Weeks on Deep Learning and Strutured Prediction

- Week 1-5: Deep Learning
  - Neural nets and Backprop
  - Convolutionnal Neural Networks (ConvNets)
  - Visualization and Advanced Topics
- Week 6-8: Structured Prediction and Applications
  - Structural SVM and Structured Ranking
  - Deep Learning and Structured Prediction

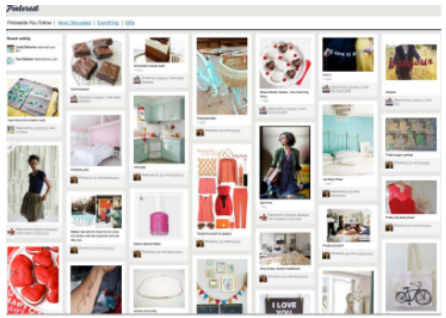
Outline

Context

Big Data

- Superabundance of data: images, videos, audio, text, use traces, *etc*

BBC: 2.4M videos

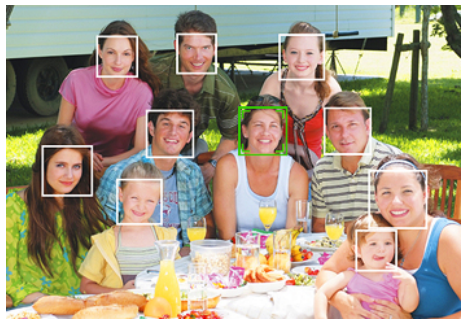


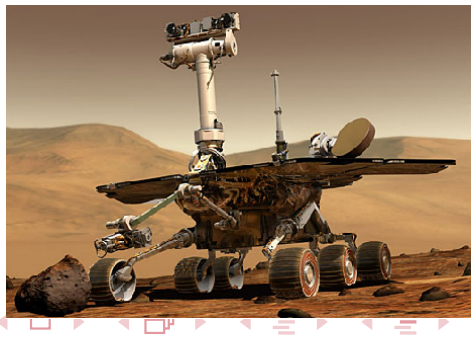
Facebook: 350B images

1B each day

100M monitoring cameras

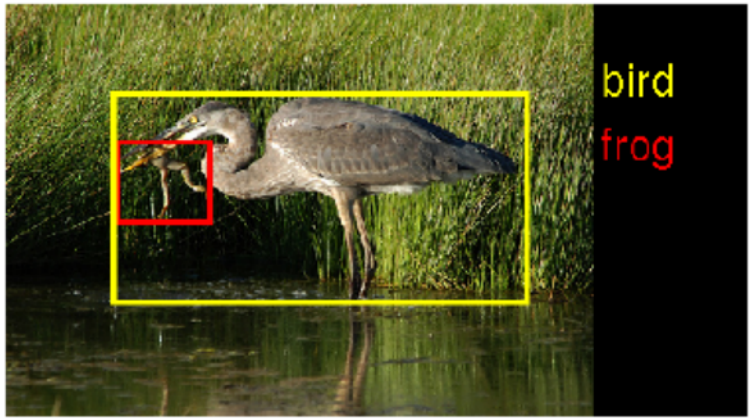
- Obvious need to access, search, or classify these data: **Recognition**
- Huge number of applications: mobile visual search, robotics, autonomous driving, augmented reality, medical imaging *etc*
- Leading track in major ML/CV conferences during the last decade

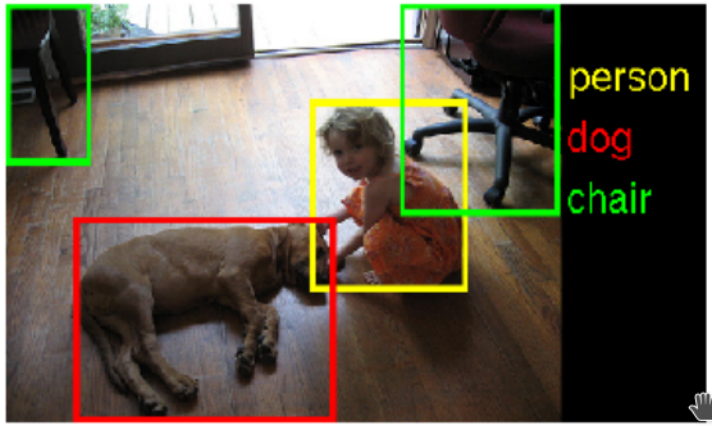




Recognition and classification

- Classification : assign a given data to a given set of pre-defined classes
- Recognition much more general than classification, *e.g.*
  - Object Localization in images
  - Ranking for document indexing
  - Sequence prediction for text, speech, audio, *etc*
- Many tasks can be cast as classification problems
  $\Rightarrow$ **importance of classification**

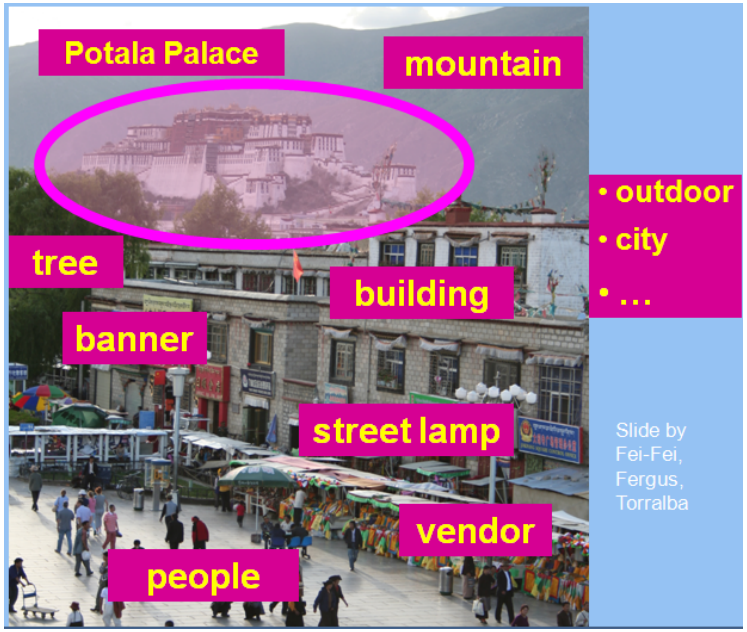




Focus on Visual Recognition: Perceiving Visual World

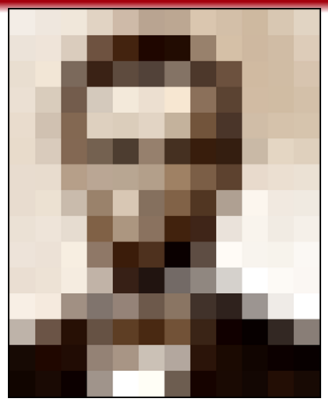
- Visual Recognition: archetype of low-level signal understanding
- Supposed to be a master class problem in the early 80's
- Certainly the most impacted topic by deep learning

- Scene categorization
- Object localization
- Context & Attribute recognition
- Rough 3D layout, depth ordering
- Rich description of scene, *e.g.* sentences



Recognition of low-level signals

Challenge: filling the semantic gap



243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	8
17	26	12	160	255	255	109	22	26	19	35	24

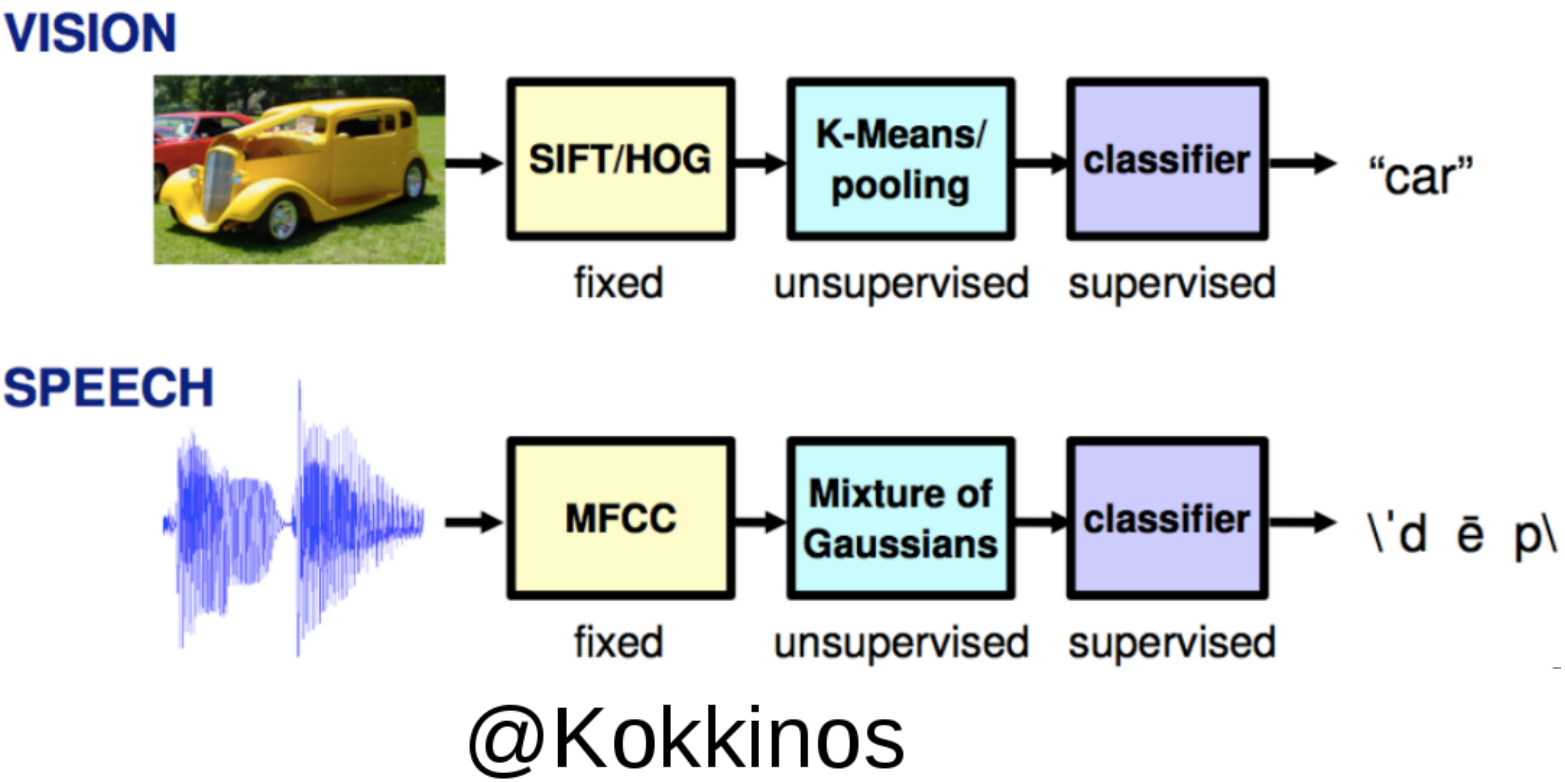
What we perceive *vs*
What a computer sees

- Illumination variations
- View-point variations
- Deformable objects
- intra-class variance
- *etc*

⇒ **How to design "good" intermediate representation ?**

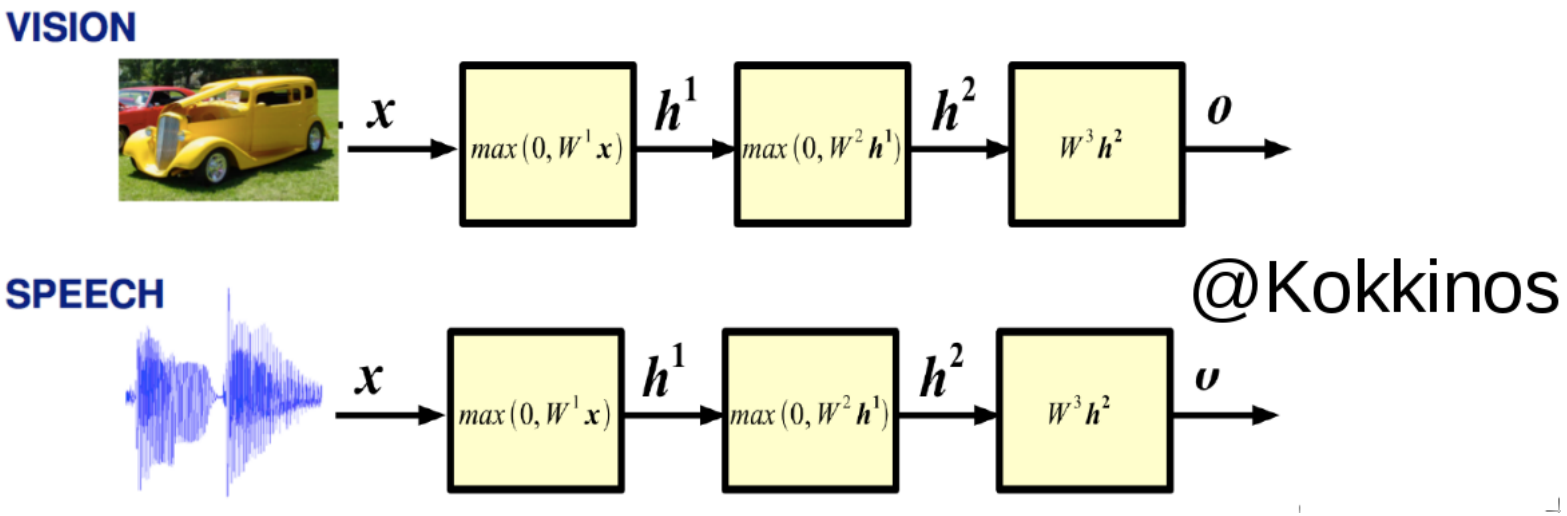
Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Before DL: handcrafted intermediate representations for each task
  - ⊖ Needs expertise (PhD level) in each field
  - ⊖ Weak level of semantics in the representation



Deep Learning (DL) & Recognition of low-level signals

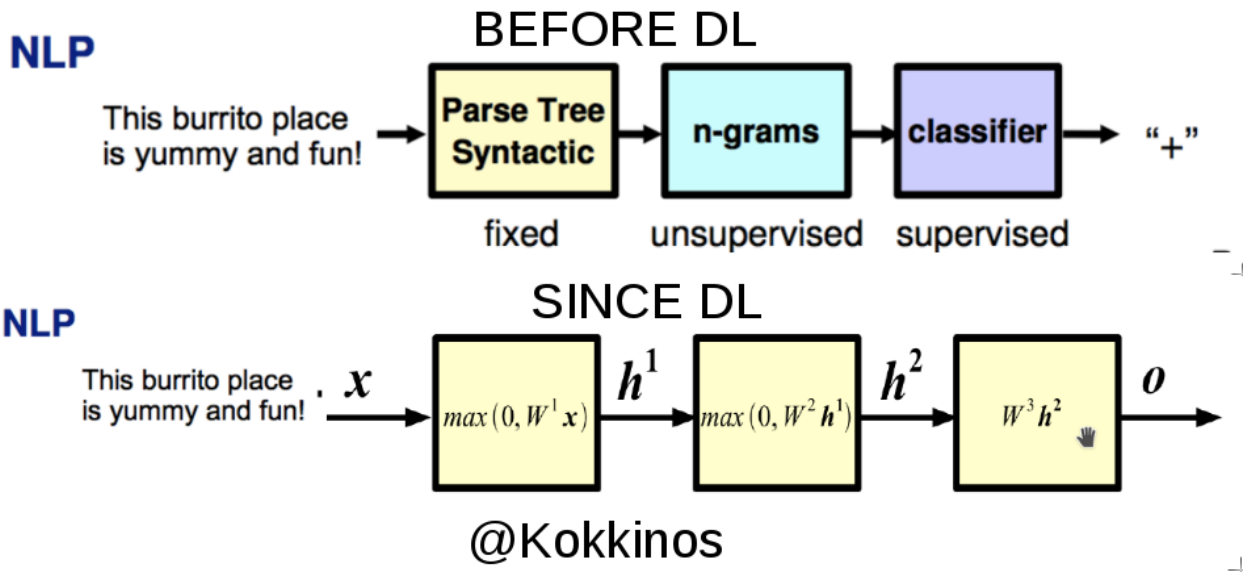
- DL: breakthrough for the recognition of low-level signal data
- Since DL: automatically **learning intermediate representations**
  - ⊕ Outstanding experimental performances >> handcrafted features
  - ⊕ Able to learn high level intermediate representations
  - ⊕ Common learning methodology ⇒ field independent, no expertise

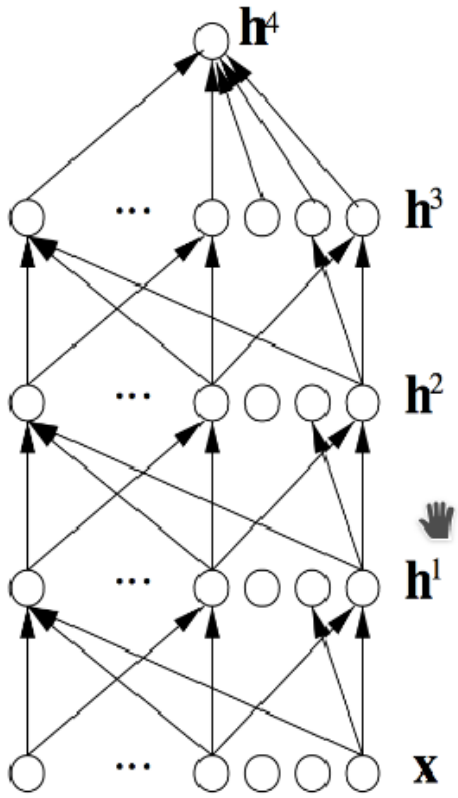


Deep Learning (DL) & Representation Learning

- DL: breakthrough for representation learning
  - Automatically learning intermediate levels of representation

- Ex: Natural language Processing (NLP)





Outline

The origins

The formal neuron, basis of the neural networks

- **1943:** The formal neuron [MP43]

x_i: inputs
w_i, b: weights
f: activation function
y: output of the neuron

$$y = f(w^\top x + b)$$

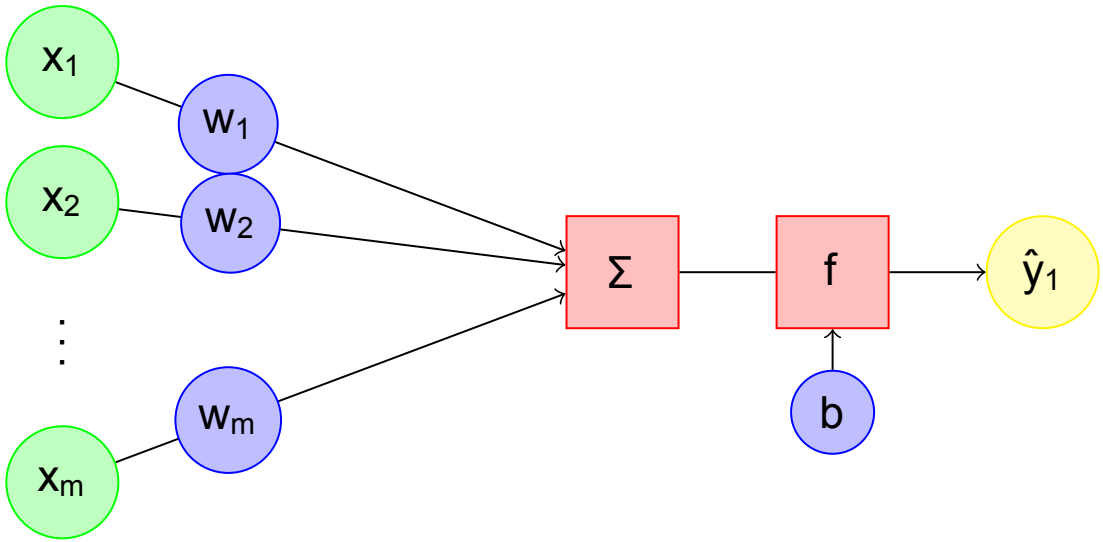


Figure : The formal neuron – **Credits:** R. Herault

Supervised Learning & Binary Classification

Supervised Learning

- Input $\mathbf{x}$, output $\mathbf{y}$
- Supervised context: during training, a set of N annotated pairs $(\mathbf{x}_i, \mathbf{y}_i^*)$
- A parametrized model $f_{\mathbf{w}}(\mathbf{x}_i) = \hat{\mathbf{y}}_i$
- Goal of supervised learning : minimizing an objective function :

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$$

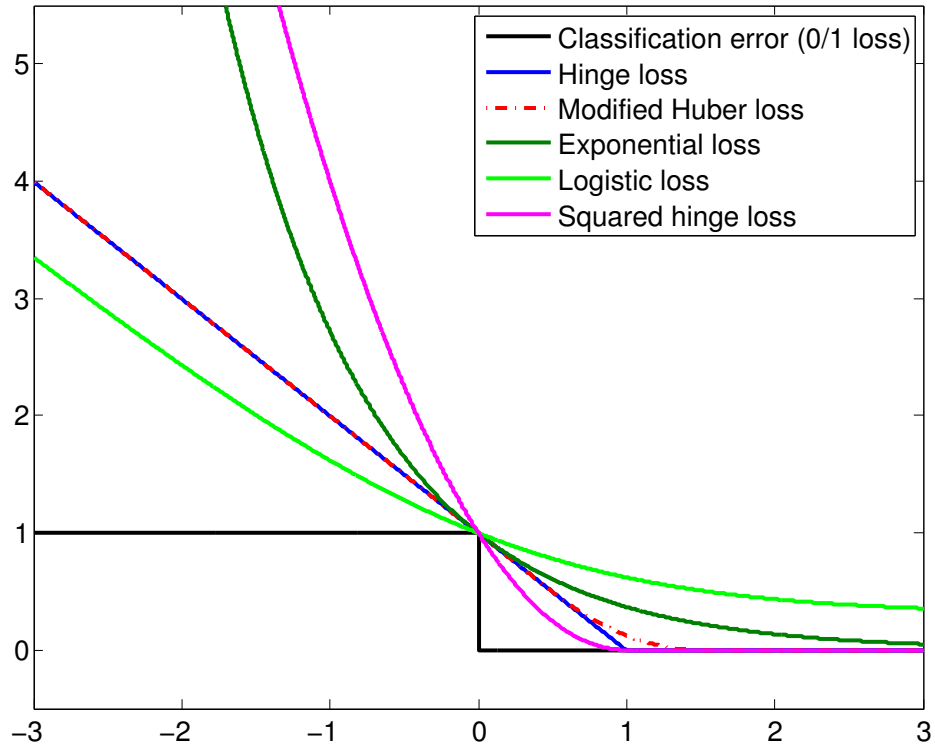
- Very general formulation, a specific task needs:
  - Define $\mathbf{y}$, *e.g.* $\mathbf{y} \in \mathbb{R}$, $\mathbf{y} \in \mathbb{R}^d$ or $\mathbf{y}$ graph (structured prediction)
  - Define ℓ: classification, regression, structured loss,

Example for classification

- Linear Classification : $f_{\mathbf{w}}(\mathbf{x}_i) = \hat{\mathbf{y}}_i = \langle \mathbf{w}, \mathbf{x}_i \rangle$, $\mathbf{y} \in \mathbb{R}$
- $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = 1_{\hat{y}_i y_i^* < 0}$ 0/1 loss
- Trick to tackle minimization of 0/1 loss with gradient descent: non differentiable

Binary classification

- Solution: define convex upper bound of 0/1 loss
- Hinge loss:
  $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = \max[0, 1 - \langle \mathbf{w}, \mathbf{x}_i \rangle y_i^*]$
  - Used in Support Vector Machines (SVM), perceptron
- logistic loss (cross-entropy)
  - Used in logistic regression, neural networks



Neural networks

- Single neuron: scalar prediction, *e.g.* binary classifcation
- Neural network: a set of output neurons $\Rightarrow$ vectorial prediction
  - *e.g.* classification: Logistic Regression (LR)

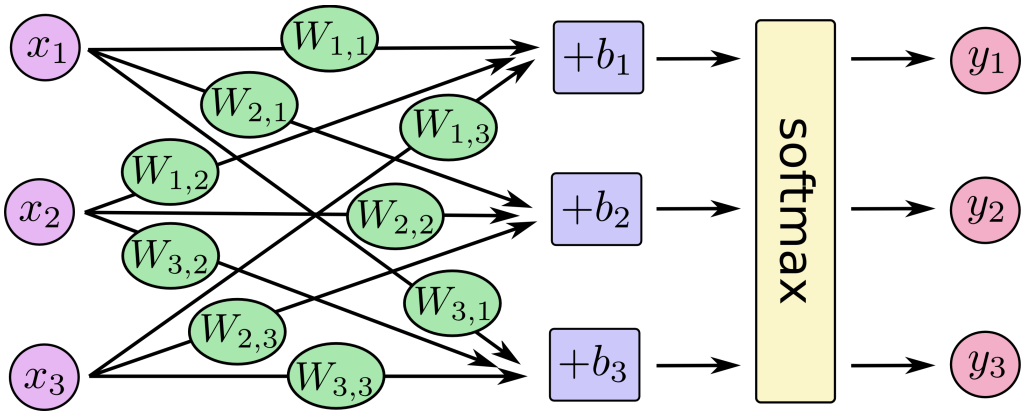
- Ex: $\mathbf{x_i}$ of size (1,d) (d=3)

$$p(\hat{y_{c,i}}|\mathbf{x}_i)) = \frac{e^{\langle \mathbf{x_i}; \mathbf{w_c} \rangle + b_c}}{\sum_{c'=1}^{K} e^{\langle \mathbf{x_i}; \mathbf{w_{c'}} \rangle + b_{c'}}}$$

Decomposed into:

- Linear Projection: $\tilde{\mathbf{y}}_\mathbf{i} = \mathbf{x_i W} + \mathbf{b}$
  $\tilde{\mathbf{y}}_\mathbf{i}$ (1,K), $\mathbf{x_i}$ (1,d) and $\mathbf{W}$ (d,K)
- Soft-max activation:

$$\hat{y_{c,i}} = \frac{e^{\tilde{y}_{c,i}}}{\sum_{c'=1}^{K} e^{\tilde{y}_{c',i}}}$$



– **Credits**: https://www.tensorflow.org/get_started/mnist/beginners

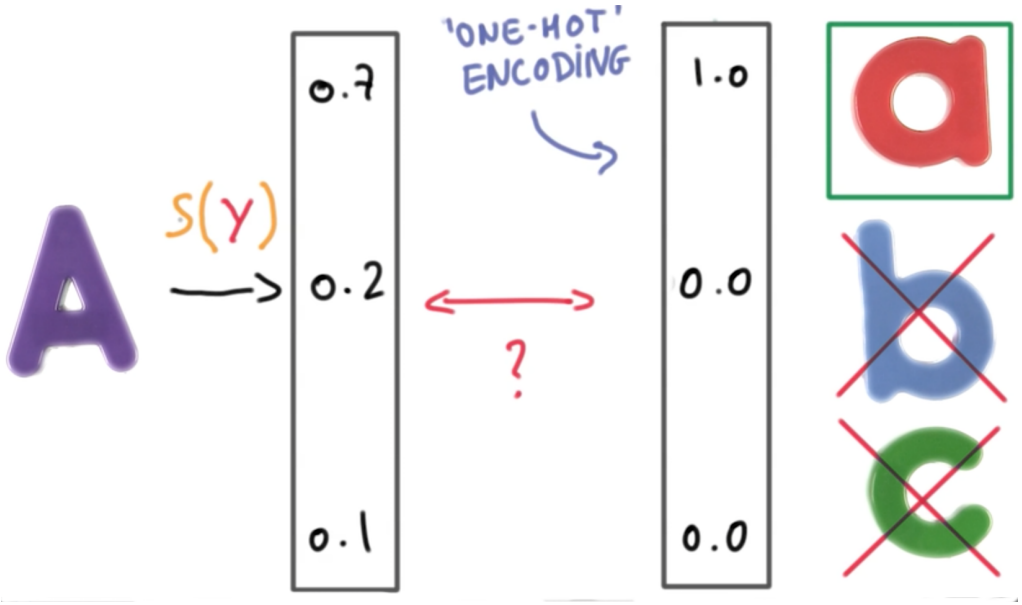
- Example in MNSIT: $K = 10$ classes
- # parameters: $784 * 10 + 10 = 7850$

Logistic Regression (LR)

LR Training

- Output: one hot-encoding for ground truth supervision

$$y^*_{c,i} = \begin{cases} 1 & \text{if c is the groud truth class for } \mathbf{x}_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$



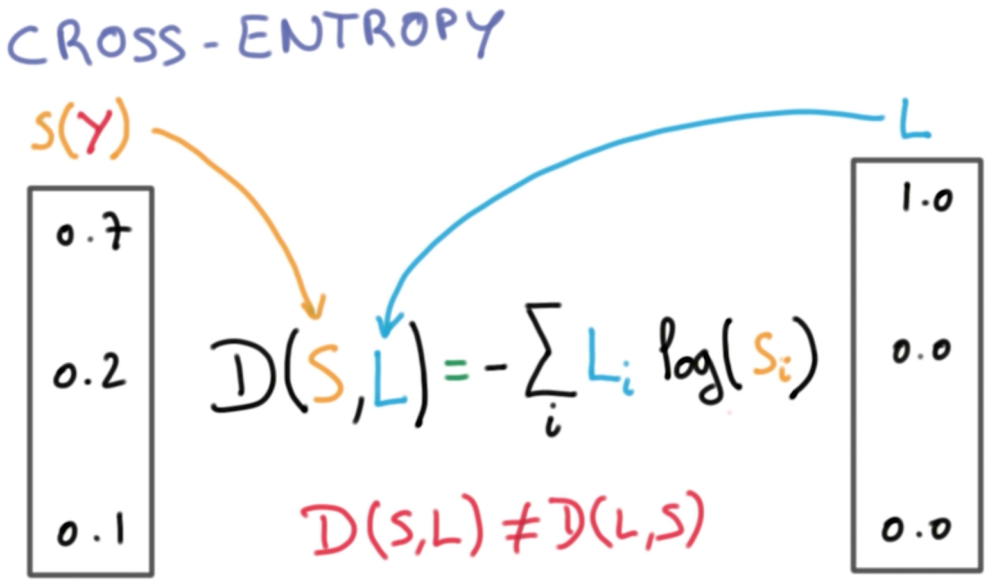
– Credits: V. Vanhoucke

Logistic Regression (LR)

LR Training

- Loss function between $\hat{\mathbf{y}}_i$ and $\mathbf{y}_i^*$: cross entropy

$$\mathcal{L}_{\mathbf{W}}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\sum_{c=1}^{10} y_{c,i}^* log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i}) \quad (2)$$



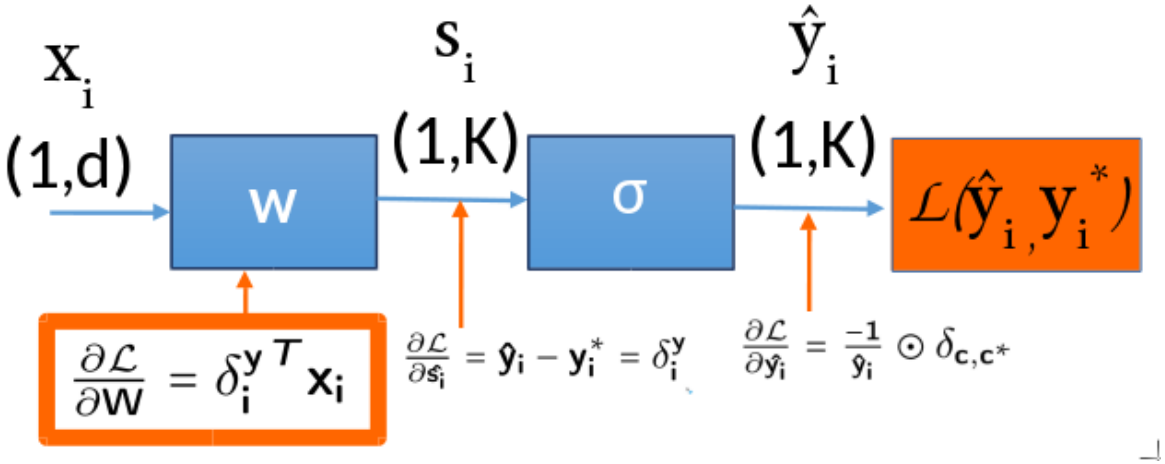
– Credits: V. Vanhoucke

- Loss function on whole training dataset $\mathcal{L}_{\mathbf{W}}(\hat{\mathbf{Y}}_i, \mathbf{Y}_i^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
  - $\mathcal{L}_{\mathbf{W}}$ convex wrt $\mathbf{W}$ parameters, extension of logistic loss to multi-class

Logistic Regression (LR) Training

Backpropagation

- $\mathcal{L}_{\mathbf{W}}$ convex wrt $\mathbf{W}$ parameters, extension of logistic loss to multi-class
- Can be optimized using gradient descent
- Intro to Backpropagation of gradient error: chain rule $\frac{\partial \mathbf{x}}{\partial z} = \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial z}$



Prove it !

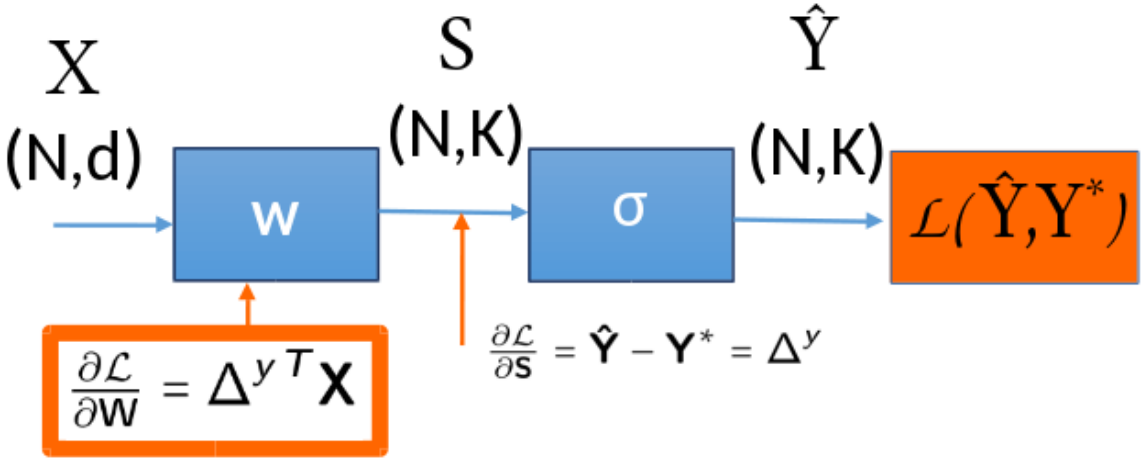
- $\mathcal{L}_{\mathbf{W}}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -log(\hat{y}_{\mathbf{c}^*,i})$
- Chain rule: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_i} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}}$
- $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_i} = \frac{-1}{\hat{y}_{\mathbf{c}^*,i}} = \frac{-1}{\hat{\mathbf{y}}_i} \odot \delta_{\mathbf{c},\mathbf{c}^*}$
- $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i} = \hat{\mathbf{y}}_i - \mathbf{y}_i^* = \delta_i^{\mathbf{y}}$
- $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \delta_i^{\mathbf{y}\,T} \mathbf{x}_i$

Logistic Regression (LR) Training

Backpropagation

- Loss function over the whole trainign dataset:

$$\mathcal{L}_W(\hat{\mathbf{Y}}, \mathbf{Y}^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$$



- Gradient descent: $W_i^{(t+1)} = W_i^{(t)} - \eta \frac{\partial L}{\partial W_i}$
- Convex function, convegence ensured (provided a sufficiently small η)
- Matrix multiplication, efficiently sloved with Python (numpy)

Kernel Methods and Deep Learning

- Regularization in SVM: learning theory

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 + \sum_{i=1}^{N} \max\left[0, 1 - \langle \mathbf{w}, \mathbf{x}_i \rangle y_i^*\right]$$

  - $||\mathbf{w}||^2$ max margin classifiers
  - Provides theoretical guarantees on the test performance of the learned model
- Kernel methods: extending these linear models to non-linear decision functions
- BUT: Model maps input $\mathbf{x}$ to output (target) $\mathbf{y}$ with a single layer
  $\Rightarrow$ **shallow model** $\neq$ **deep learning**

The Multi-Layer Perceptron (MLP) & Deep Learning

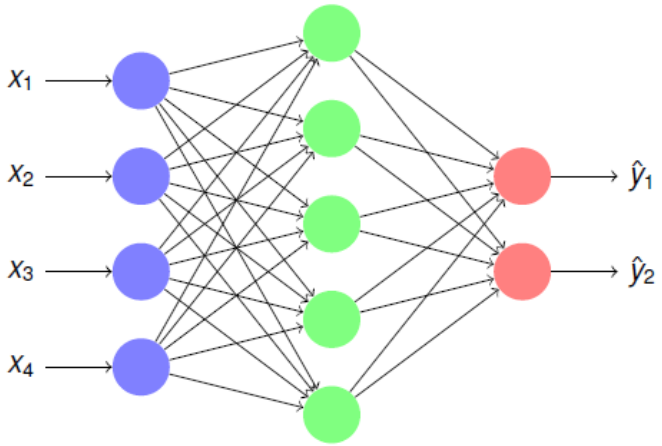


Figure : Perceptron with 1 hidden layer – **Credits: R. Herault**

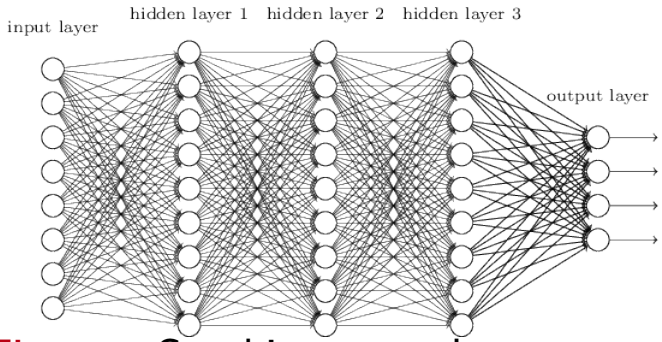
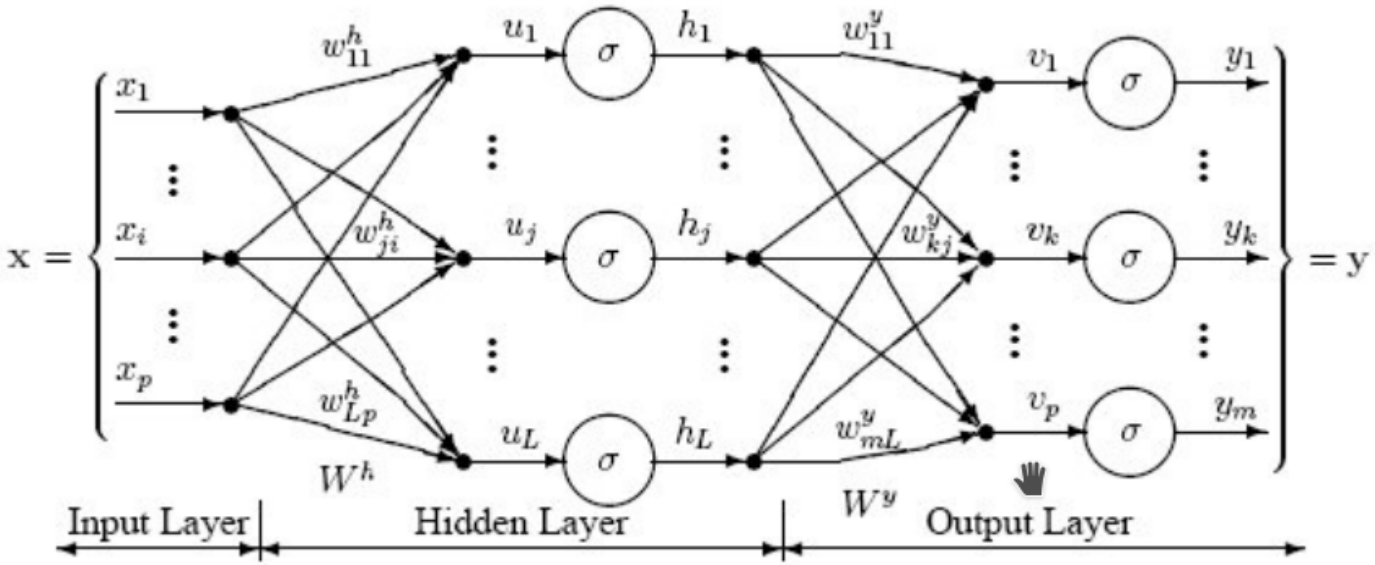


Figure : Stacking more layers, toward "deep learning" – **Credits: M. Nielsen**

- Basis of the "deep learning" field
- Principle: Stacking layers of neural networks to allow more complex and rich functions
- **Not limited to linear prediction**
- With a hidden layer, **can approximate any function** given enough hidden units [Cyb89]
- Can be seen as **different levels of abstraction** from low-level features to the high-level ones

The Multi-Layer Perceptron (MLP)

Perceptron: Neural Network with 1 Hidden Layer

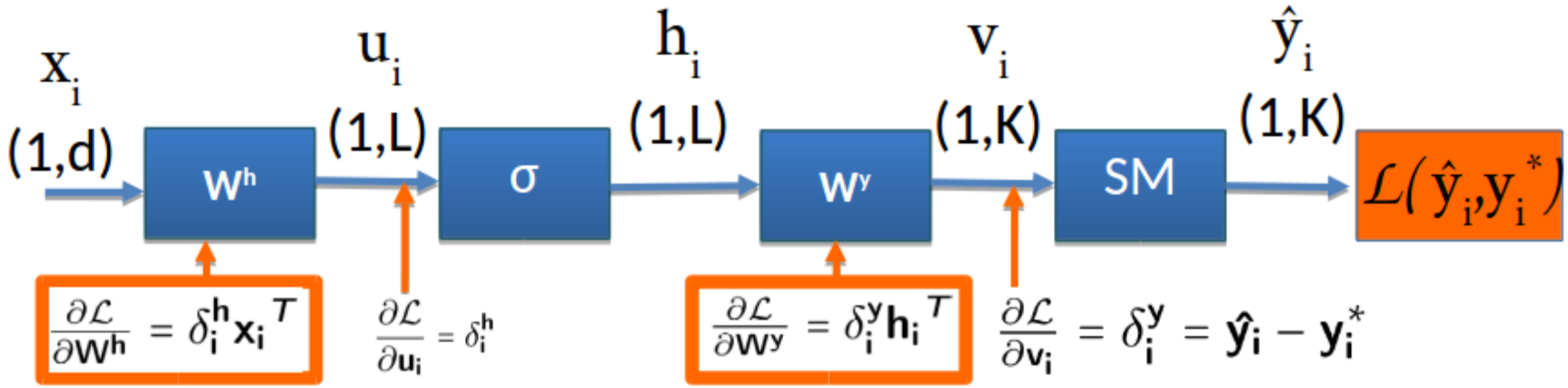


- Linear projection to L hidden units: $\hat{\mathbf{u}}_\mathbf{i} = \mathbf{x}_\mathbf{i}\mathbf{W}^\mathbf{h} + \mathbf{b}^\mathbf{h}$
  - Sigmoid $\forall j \in \{1; L\}\ h_{i,j} = \frac{1}{1+exp(-u_{i,j})}$
- Linear projection to K output units: $\hat{\mathbf{v}}_\mathbf{i} = \mathbf{h}_\mathbf{i}\mathbf{W}^\mathbf{y} + \mathbf{b}^\mathbf{y}$
  - Soft-max activation function: $\forall j \in \{1; K\}\ y_{i,j} = \frac{exp(v_{i,j})}{\sum_{i=1}^{K} exp(v_{i,k})}$

Perceptron

Training with Backpropagation

- Loss with a single example: $\mathcal{L}_{\mathbf{W}}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -log(\hat{y}_{c^*,i})$
- Backward pass: until first hidden layer: $\sim$ Logistic Regression
- $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i} = \delta_i^{\mathbf{y}} = \hat{\mathbf{y}}_i - \mathbf{y}_i^*$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\mathbf{y}}} = \delta_i^{\mathbf{y}} \mathbf{h}_i{}^T$
- **Go backward**: need to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{u}_i} = \delta_i^{\mathbf{h}}$
- If we know $\delta_i^{\mathbf{h}} \Rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{\mathbf{h}}} = \delta_i^{\mathbf{h}} \mathbf{x}_i{}^T \sim$ Logistic Regression

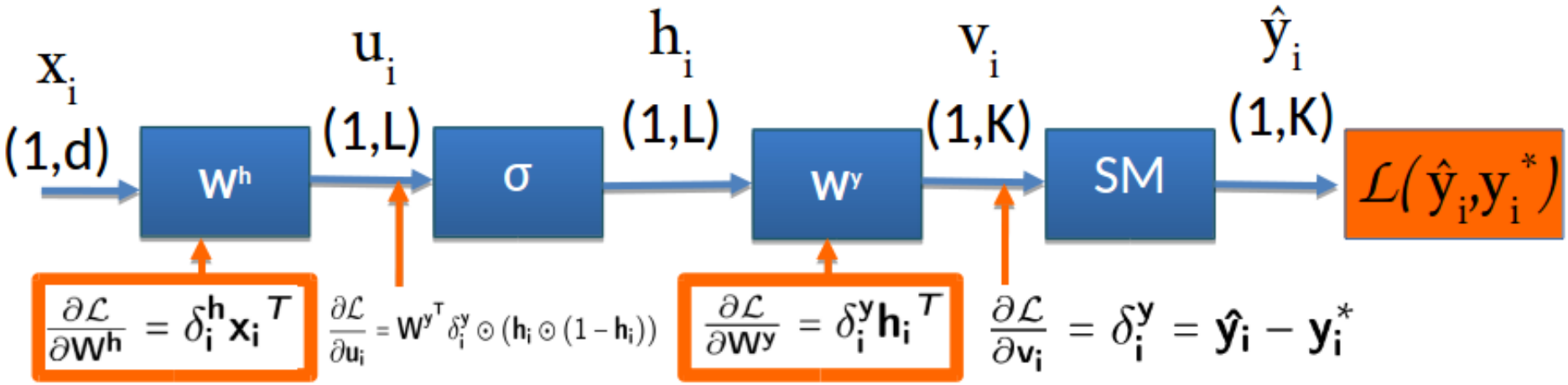


Perceptron

Training with Backpropagation

- Computing $\frac{\partial \mathcal{L}}{\partial \mathbf{u_i}} = \delta_i^{\mathbf{h}} \Rightarrow$ use chain rule: $\frac{\partial \mathcal{L}}{\partial \mathbf{u_i}} = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial \mathbf{v_k}} \frac{\partial \mathbf{v_k}}{\partial \mathbf{u_i}}$
- ... Leading to:

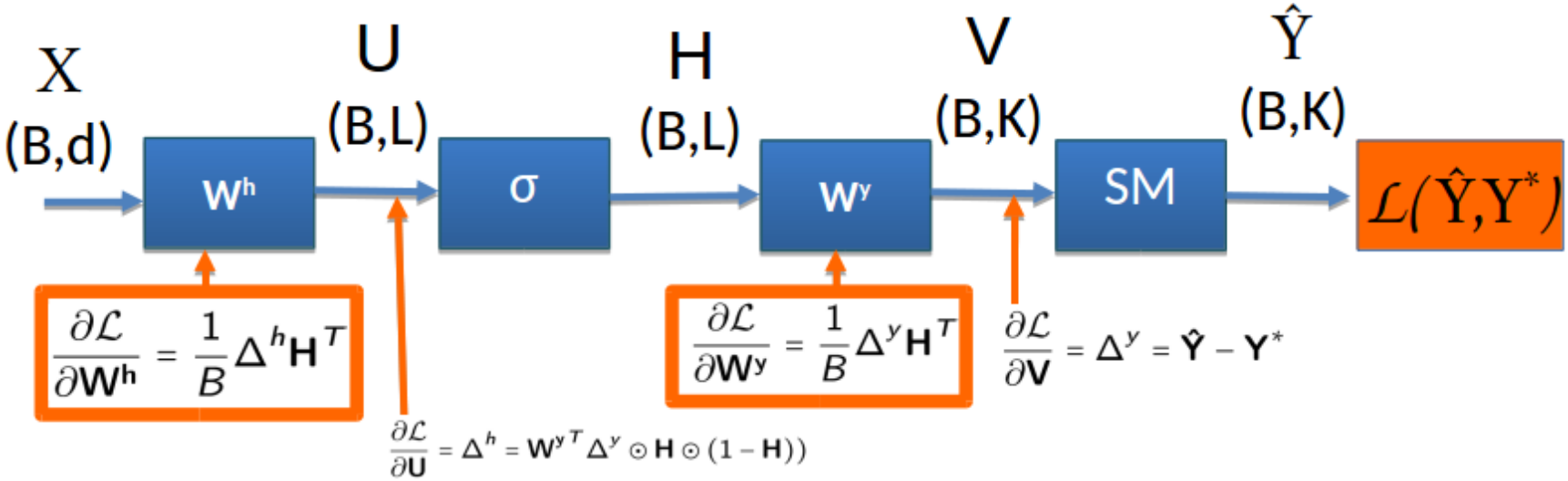
$$\frac{\partial \mathcal{L}}{\partial \mathbf{u_i}} = \delta_i^{\mathbf{h}} = \mathbf{W^y}^T \delta_i^{\mathbf{y}} \odot \sigma'(\mathbf{h_i}) = \mathbf{W^y}^T \delta_i^{\mathbf{y}} \odot (\mathbf{h_i} \odot (1 - \mathbf{h_i}))$$



Perceptron

Training with Backpropagation: Optimization

- Training with gradient descent: too slow even for moderate dimensionality dataset size
- Using Stochatic Gradient Descent (SGD): but noisy gradient
- Solution: gradient estimated on a subset (BATCH) of B examples



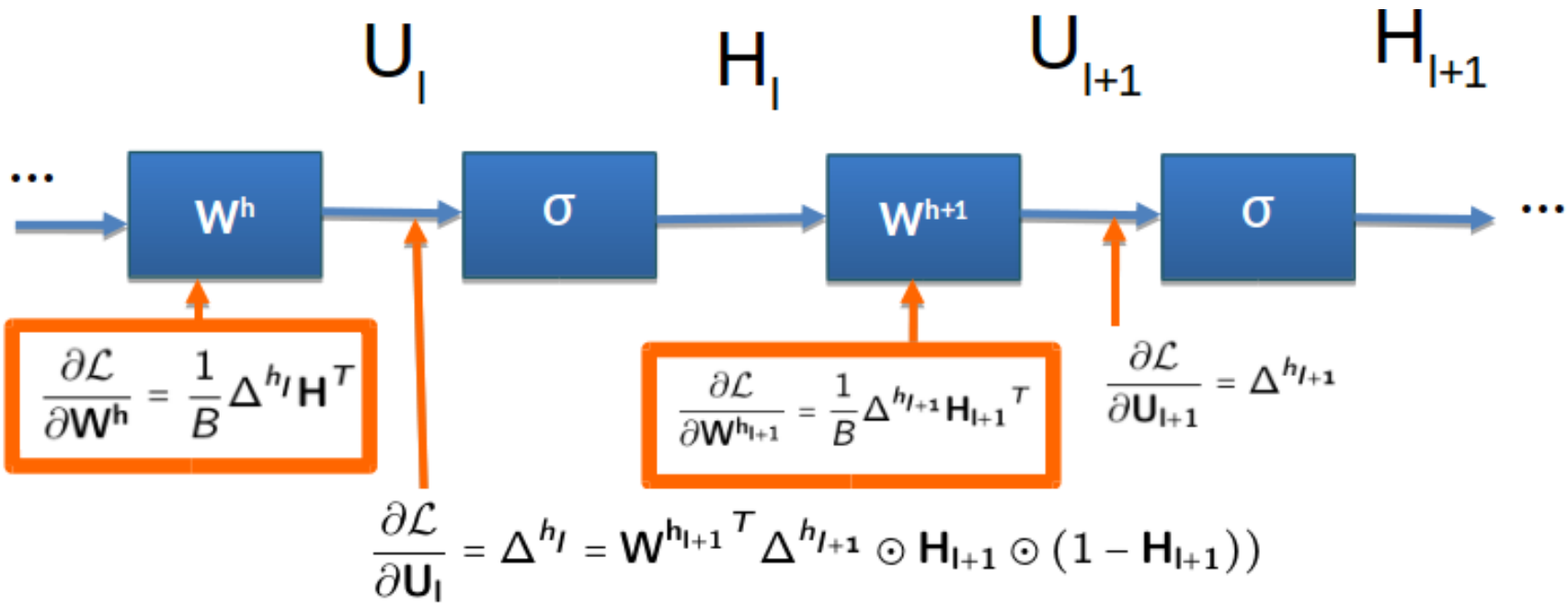
Controlling Overfitting

- Early stopping in validation set
- Weight decay: $\sim \ell_2$ regularization

The Multi-Layer Perceptron (MLP)

MLP: Adding more layers

- Easy to add more layers: fully connected + non-linearity
- Need to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_l} = \Delta^{h_l}$ from $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_{l+1}} = \Delta^{h_{l+1}}$



References I


George Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems 2 (1989), no. 4, 303–314.


Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics 5 (1943), no. 4, 115–133.