Project Proposal

- due TODAY @ 11:59pm
- one page maximum stating:
  - student names
  - problem
  - data you will use
  - draft of proposed solution (can be **updated later**)
  - experiments you will run (can be **updated later**)
  - references (you can use an additional page for this)
- send me pdf by email (mvasconcelos@eng.ucsd.edu) with:
  - Subject: Group X Proposal, where X is the group number in this list
  - cc to all group members

This assignment is worth 5% of your class grade. **If you submit the proposal in time and make a serious attempt at addressing the bullet points above, you will get full score. I'm not, at this point, grading projects on their merits.** I will look at the proposal and give you some feedback. This will be mostly on issues that I think may become serious obstacles and you need to consider urgently. For example, if I find the problem you propose to be outside the scope of the class, that you may not be able to find data to train the methods you are proposing, etc. Note that if I say "OK", it just means that I see no such problems. It does not mean that you will receive an A just by doing what you proposed. I see these proposals more as a "direction to where the project is going." The projects themselves will be evaluated at the end of the quarter, according to the guidelines published.

1. **Hussain**, Tanvir; **Lewis**, Cameron; **Villamar**, Sandra
2. **Dong**, Meng; **Long**, Jianzhi; **Wen**, Bo; **Zhang**, Haochen
3. **Chen**, Yuzhao; **Li**, Zonghuan; **Song**, Yuze; **Yan**, Ge
4. **Li**, Jiayuan; **Xiao**, Nan; **Yu**, Nancy; **Zhou**, Pei
5. **Li**, Zheng; **Tao**, Jianyu; **Yang**, Fengqi
6. **Bian**, Xintong; **Jiang**, Yufan; **Wu**, Qiyao
7. **Chen**, Yongxing; **Yao**, Yanzhi; **Zhang**, Canwei
8. **Nukala**, Kishore; **Pulleti**, Sai; **Vaidyula**, Srikar
9. **Baluja**, Michael; **Cao**, Fangning; **Huff**, Mikael; **Shen**, Xuyang
10. **Arun**, Aditya; **Long**, Heyang; **Peng**, Haonan
11. **Cowin**, Samuel; **Liao**, Albert; **Mandadi**, Sumega
12. **Jia**, Yichen; **Jiang**, Zhiyun; **Li**, Zhuofan
13. **Dandu**, Murali; **Daru**, Srinivas; **Pamidi**, Sri
14. **He**, Bolin; **Huang**, Yen-Ting; **Wang**, Shi; **Wang**, Tzu-Kao
15. **Chen**, Luobin; **Feng**, Ruining; **Wu**, Ximei; **Xu**, Haoran
16. **Chen**, Rex; **Liang**, Youwei; **Zheng**, Xinran
17. **Aguilar**, Matthew; **Millhiser**, Jacob; **O'Boyle**, John; **Sharpless**, Will
18. **Wang**, Haoyu; **Wang**, Jiawei; **Zhang**, Yuwei
19. **Chen**, Yinbo; **Di**, Zonglin; **Mu**, Jiteng
20. **Chowdhury**, Debalina; **He**, Scott; **Ye**, Yiheng
21. **Lin**, Wei-Ru; **Ru**, Liyang; **Zhang**, Shaohua
22. **Bhavsar**, Shivad; **Blazej**, Christopher; **Bu**, Yinyan; **Liu**, Haozhe
23. **Chen**, Claire; **Hsieh**, Chia-Wei; **Lin**, Jui-Yu; **Tsai**, Ya-Chen
24. **Cheng**, Yu; **Yu**, Zhaowei; **Zaidi**, Ali
25. **Assadi**, Parsa; **Brugere**, Tristan; **Pathak**, Nikhil; **Zou**, Yuxin
26. **Candassamy**, Gokulakrishnan; **Dixit**, Rajeev; **Huang**, Joyce
27. **Kok**, Hong; **Wang**, Jacky; **Yan**, Yijia; **Yuan**, Zhouyuan
28. **Luan**, Zeting; **Yang**, Zheng
29. **Cuawenberghs**, Kalyani; **Mojtahed**, Hamed

ECE 271B – Winter 2022

Neural Networks (cont.)

Disclaimer:
This class will be recorded
and made available to students asynchronously.

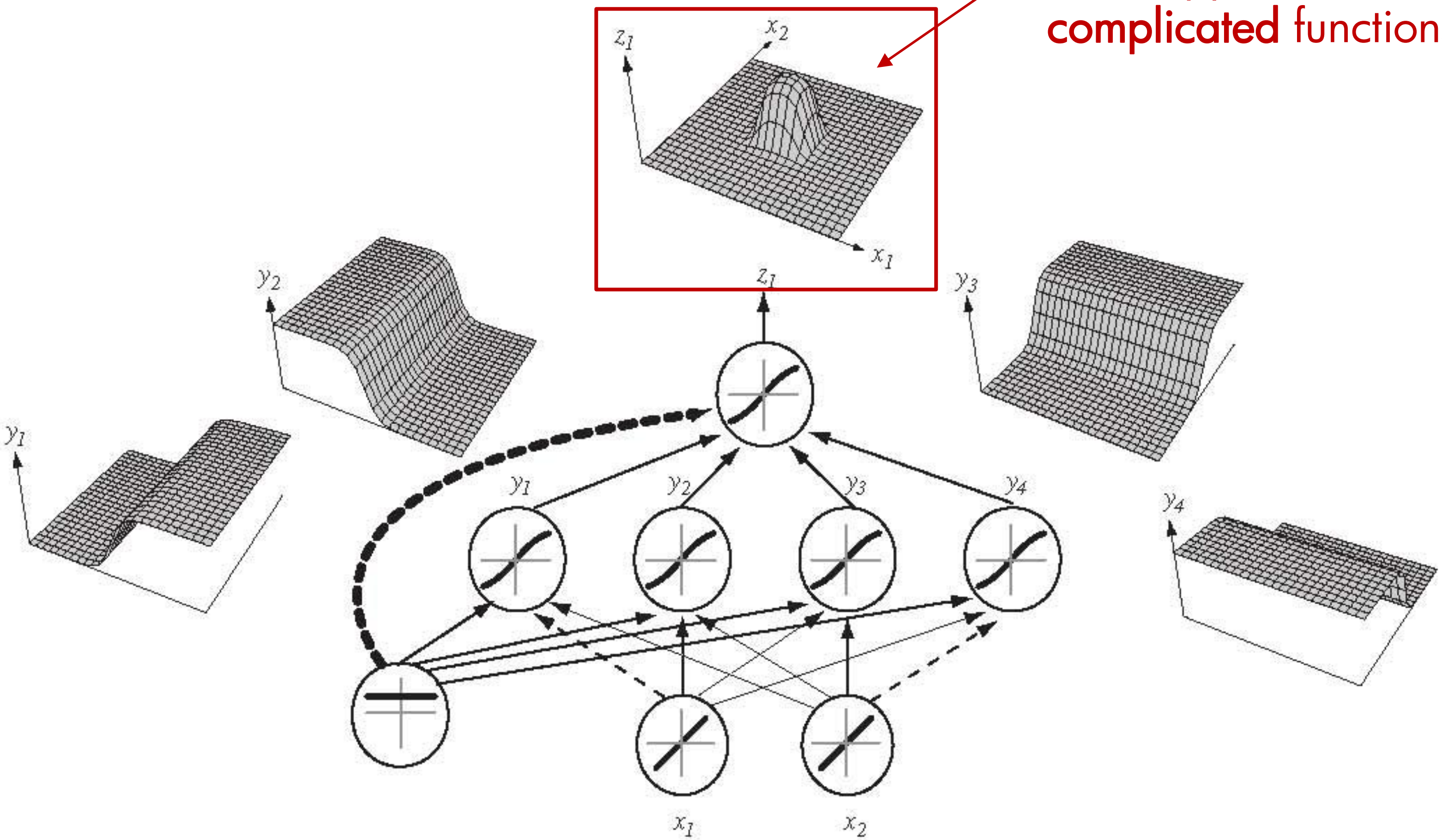
Manuela Vasconcelos

ECE Department, UCSD

Neural Network

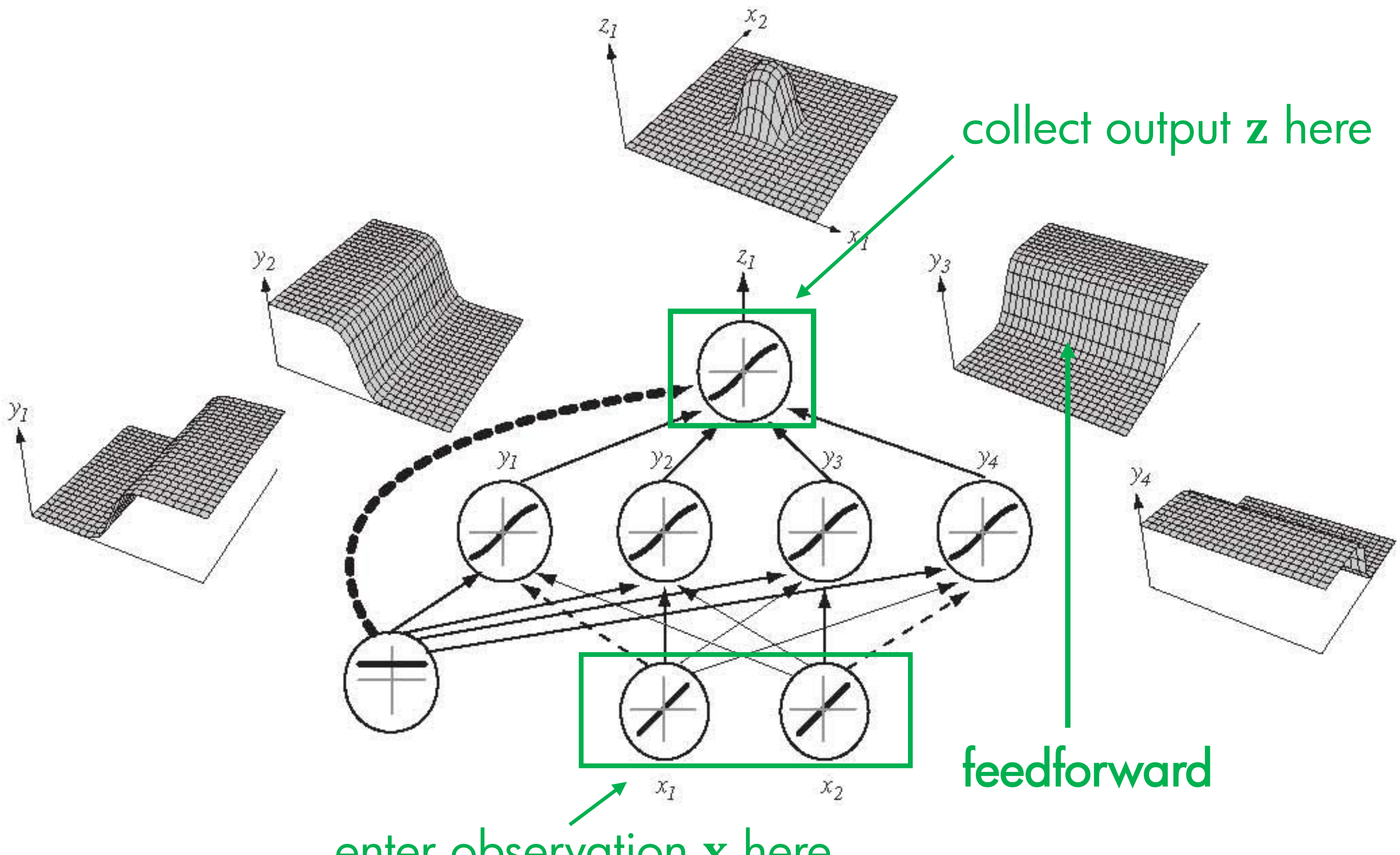
- the MLP as function approximation

even with just 2 layers,
it is possible
to approximate
complicated functions!



Two Modes of Operation

- normal mode, after training: feedforward

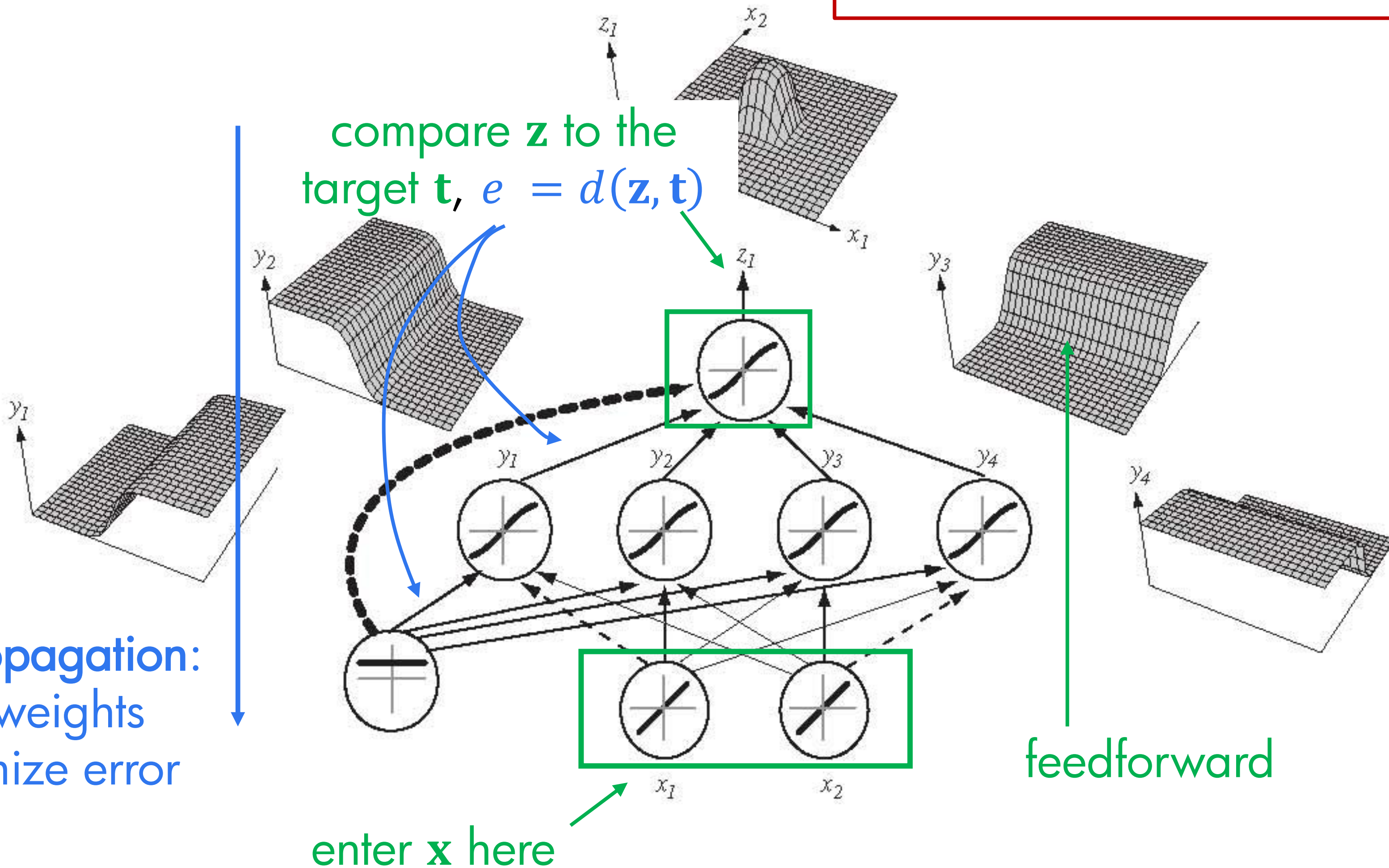


Two Modes of Operation

- training mode: backpropagation

iterative:

- propagate $\mathbf{x}$ feedforward
- compute error e
- backpropagate to adjust weights



backpropagation:
update weights
to minimize error

Backpropagation

- is just gradient descent
- at the end of the day, the output $\mathbf{z}$ is just a big function of
  - input vector $\mathbf{x}$
  - weights, which we can be represent by a "big" vector $\mathbf{W}$
  - e.g.

$$\mathbf{z} = s\left[\sum_j v_j\, s\left(\sum_i w_{ji} x_i\right)\right] = \mathbf{z}(\mathbf{x};\mathbf{W}) \quad \text{with} \quad \mathbf{W} = (\mathbf{v},\mathbf{w})$$

- **objective:** given a dataset $\mathcal{D} = \{(\mathbf{x}_1,\mathbf{t}_1),\dots,(\mathbf{x}_n,\mathbf{t}_n)\}$, determine

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} J(\mathbf{W})$$

$$J(\mathbf{W}) = \sum_{i=1}^{n} L\big(\mathbf{t}_i, \mathbf{z}(\mathbf{x}_i;\mathbf{W})\big)$$

$$L(\mathbf{t},\mathbf{z}) = \frac{1}{2}\sum_k [t_k - z_k]^2$$

squared−error with the multi−label classifier

Backpropagation: Summary

- for <u>any</u> pair (i, j)

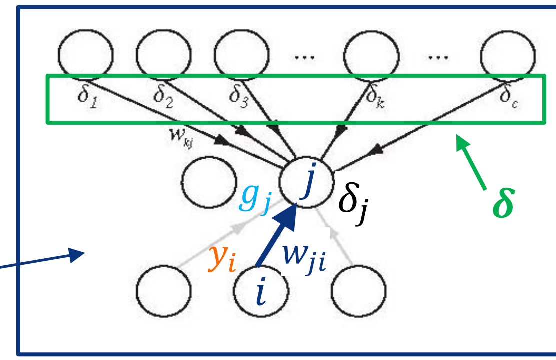
Ramelhart, David; Hinton, Geoffrey; Williams, Ronald; Learning Internal Representations by Backpropagating Errors. In *Parallel Distributed Processing*, Volume 1, MIT Press, 1986.

$$\boxed{\frac{\partial L}{\partial w_{ji}} = -\delta_j \, y_i}$$

with

$$\delta_j = (t_j - z_j) s'[u_j] \qquad \text{if } j \text{ is output}$$

$$\delta_j = \left[\textstyle\sum_k \delta_k w_{kj}\right] s'[g_j] \qquad \text{if } j \text{ is hidden}$$



the weight updates are

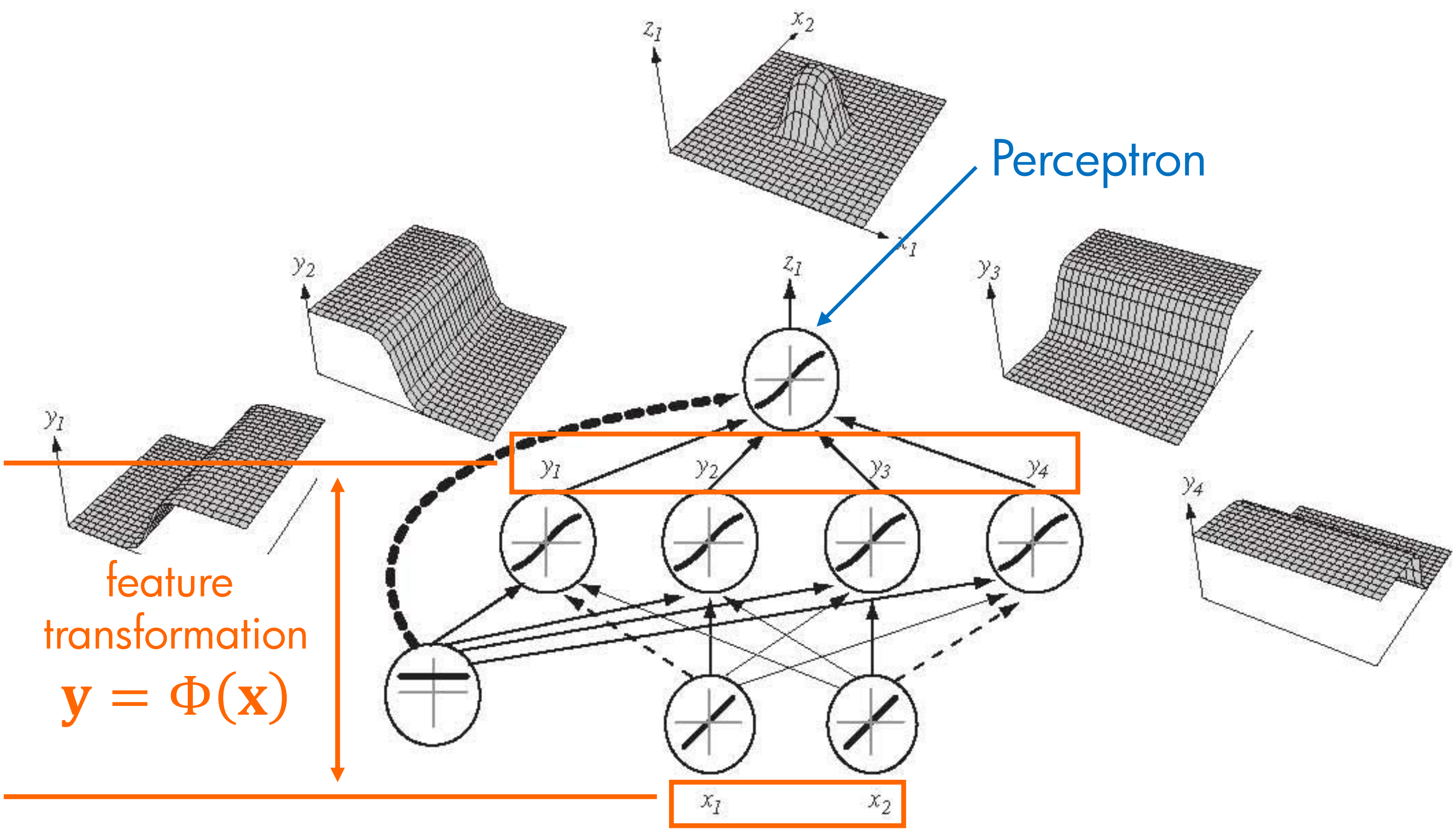
$$\boxed{w_{ji}^{(n+1)} = w_{ji}^{(n)} - \eta \frac{\partial L}{\partial w_{ji}}}$$

- the error is "backpropagated" by local message passing!

Feature Transformation

- MLP can be seen as:

  non−linear feature transformation + linear discriminant

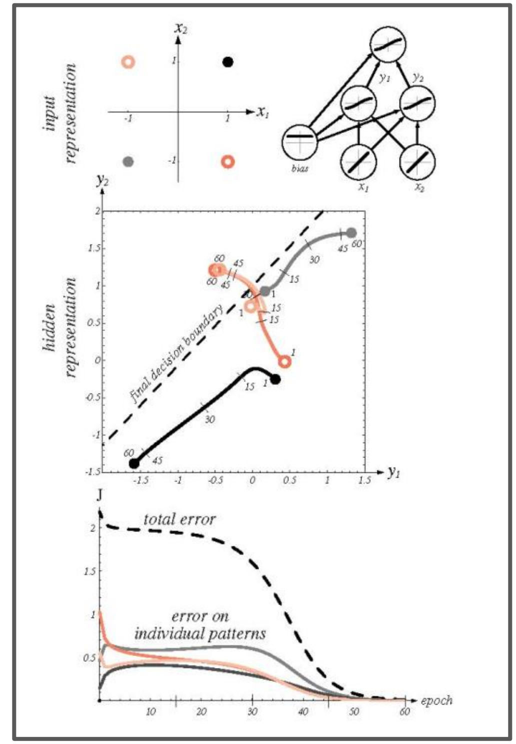


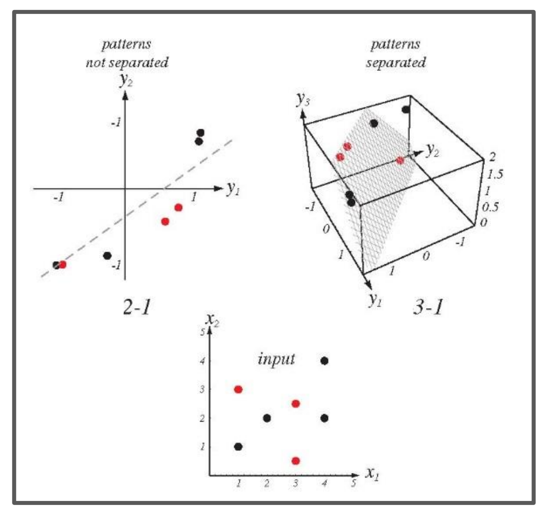
Feature Transformation

- feature transformation
  - searches for the space where the patterns become separable

- Q: is separability always possible?
- A: not really, depends on the number of units

- **in practice**
  - art–form
  - trial and error

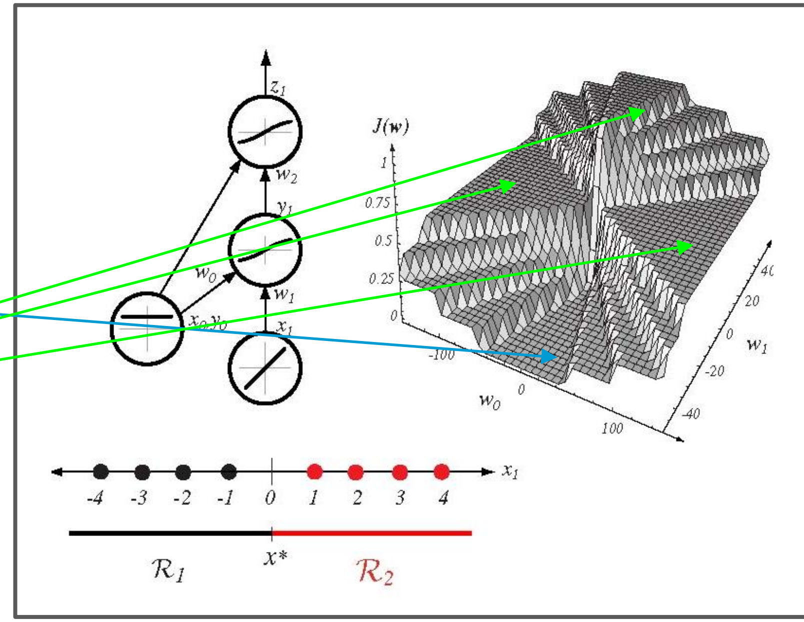




Other Problems

- the optimization surface (cost) can be quite <u>nasty</u>

- cost has <u>**many**</u> “plateaus”
  - global optimal solution has no error
  - but gradient frequently close to zero
  - <u>slow</u> progress



- how do we set the learning rate η?
  - if too small or too big, we will need various iterations
  - could even diverge

Structural Risk Minimization

- what about complexity penalties, overfitting, and all that?
- solve

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} L\big(\mathbf{t}_i, \mathbf{z}(\mathbf{x}_i; \mathbf{W})\big) \quad \text{subject to} \quad \mathbf{W}^T\mathbf{W} < \lambda$$

- we will see that this is equivalent to

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \left\{ \sum_{i=1}^{n} L\big(\mathbf{t}_i, \mathbf{z}(\mathbf{x}_i; \mathbf{W})\big) + \frac{2\varepsilon}{\eta} \mathbf{W}^T\mathbf{W} \right\}$$

- re–working out backpropagation, this can be done by "shrinking"
  - after each weight update, do $\mathbf{W}^{new} = \mathbf{W}^{old}(1-\varepsilon), 0 < \varepsilon < 1$
  - this is known as "weight decay" and penalizes complex models

In Summary

- this works, but requires **tunning** ε
- the **cost surface** is **nasty**
- one needs to try **different architectures**
- hence, training can be **painfully slow**
  - "weeks" is quite common
  - a good neural network may take years to train
- however, when you are finished it **tends to work well**
- examples
  - the Rowley and Kanade face detector
  - the LeCunn digit recognizer (see http://yann.lecun.com/exdb/lenet/index.html)

The Last Five–Ten Years

- what we have seen so far was the state of the art in the 1990's
- over the last 5–10 years, neural networks have become dominant again
- you surely have heard all the talk about "deep learning" and how it is revolutionizing AI
- what has happened?

  well, a **combination** of

  - a few (small) advances in neural network architectures
  - large advances in
    - implementation
    - availability of data (what is now called "Big Data")

Architecture

- there have been some **advances in NN architectures** (although some of these were already used in the 80s)
- a **popular architecture** is the Convolutional NN (CNN)
  - commonly used in applications involving **signals** (images, speech, audio)
  - inspired by classic **linear systems** theory

- consider a **linear filter**
  - for simplicity, we consider a 1D signal $x[n]$
  - to which we apply a **filter** of impulse response $h[n]$ (e.g. to remove noise)
  - mathematically, the **filter output** is given by the **convolution** of x and h

$$\boxed{y[n] = \sum_k x[k]h[n-k]}$$

Architecture

- linear filter
  - the **convolution** of x and h

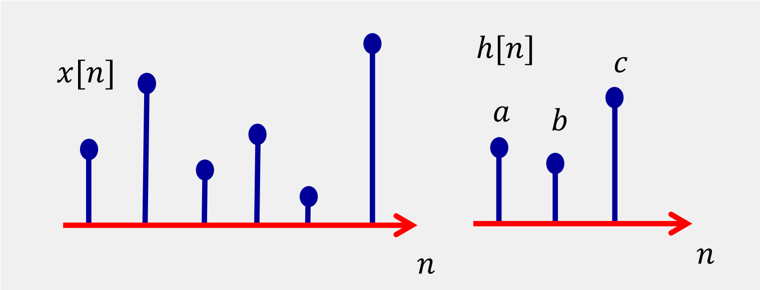
$$\boxed{y[n] = \sum_k x[k]h[n-k]}$$

can be written as

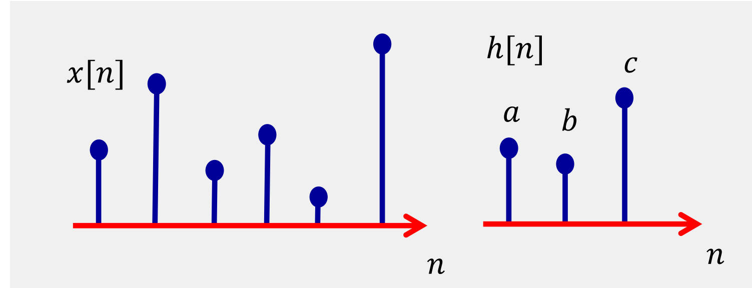
$$y[n] = \sum_k x[k]g_n[k] = \langle x[k], g_n[k] \rangle \qquad g_n[k] = h[n-k]$$

i.e., each entry of $y[n]$ is the **dot−product** of $x[n]$ with a shifted and inverted replica of the impulse response $h[n]$

  - e.g., let



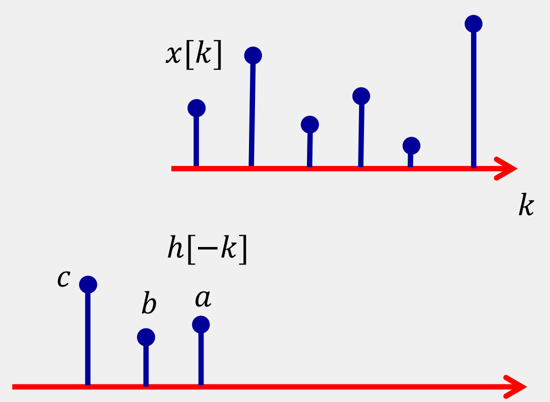
Architecture



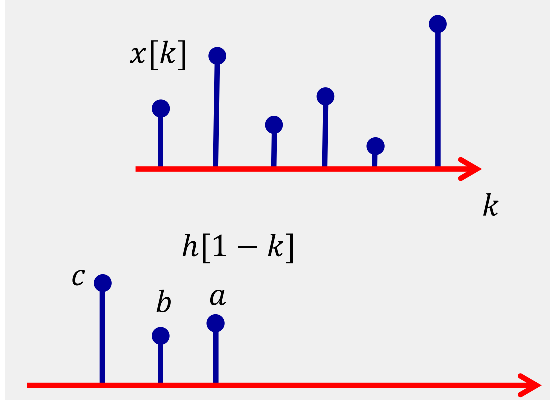
- convolution of x and h

$$y[n] = \sum_k x[k] g_n[k] = \langle x[k], g_n[k] \rangle \qquad g_n[k] = h[n-k]$$

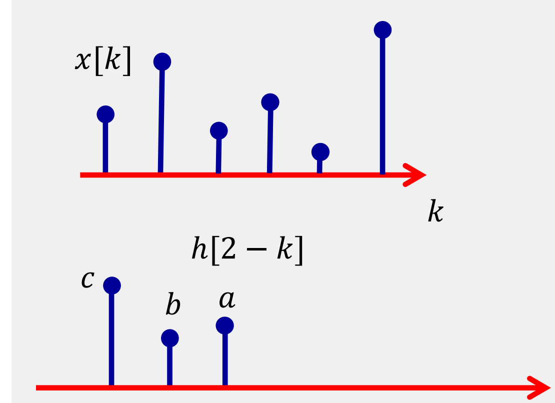
$y[0]$ is the dot–product of



$y[1]$ is the dot–product of



$y[2]$ is the dot–product of



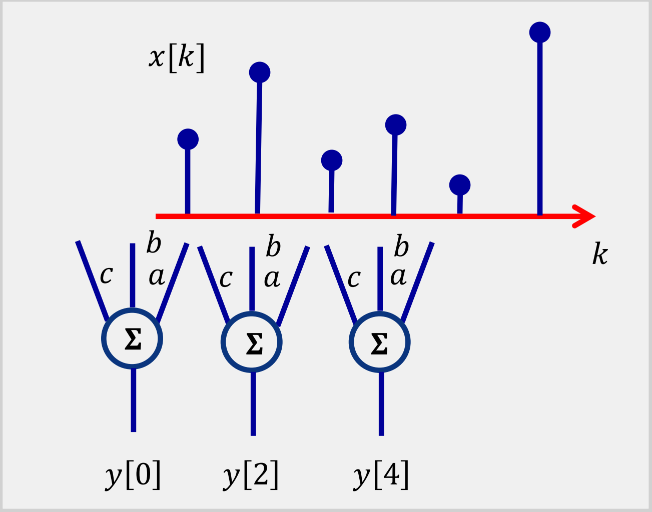
and so on…

Architecture

- convolution of x and h

$$y[n] = \sum_k x[k] g_n[k] = \langle x[k], g_n[k] \rangle \qquad g_n[k] = h[n-k]$$

  - this can be computed as



Note
for simplicity of the picture, $y[1]$, $y[3]$ are not represented

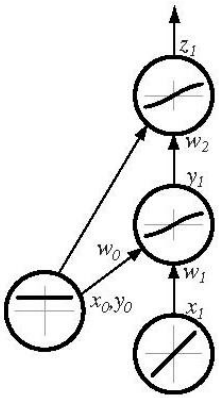
- note that this is a **neural network** with <u>two</u> properties
  - <u>sparse connectivity</u>: each unit only has a <u>few</u> **non−zero weights**
  - <u>shared weights</u>: the **weights of all units** are <u>equal</u>

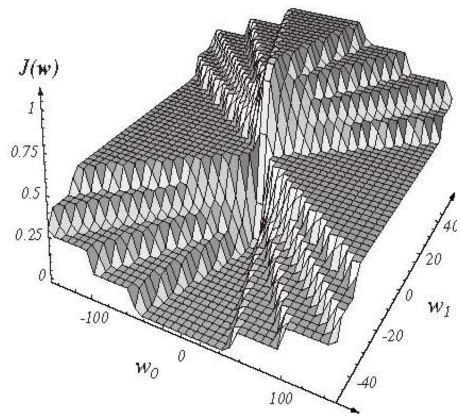
Architecture

- a network with such layers is called **convolutional**

- Convolutional NNs (CNNs) are very suitable for signal processing type of applications (vision, speech)

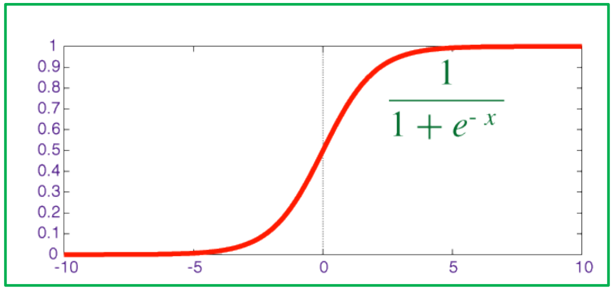
- besides this, it has been found that the training problems can be alleviated by using **different non–linearities**





- instead of the traditional hyperbolic tangent or the sigmoid

- modern networks frequently use [rectified linear units]



Architecture

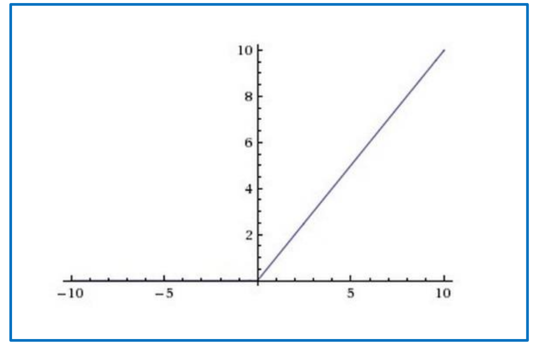
$$\frac{\partial L}{\partial w_{ji}} = -\delta_j \, y_i$$

$$\delta_j = \left[\sum_k \delta_k w_{kj}\right] s'[g_j] \quad \text{if } j \text{ is hidden}$$

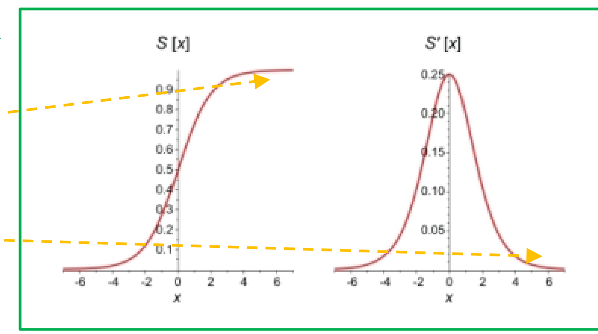
$$\delta_j = (t_j - z_j)\, s'[u_j] \quad \text{if } j \text{ is output}$$

- Rectified Linear Unit (ReLU)

$$\boxed{s(x) = \max(x, 0)}$$



- recall: the gradient of backpropagation depends **only** on the derivative of the non−linearity
  - for sigmoid or hyperbolic tan, this is mostly zero
  - a large response creates **zero** derivative
  - for networks that have **many layers**, the **gradient vanishes**
  - since the ReLU has **constant derivative**, it reduces this problem
  - **learning tends to converge much faster**



Architecture

- another factor that can significantly speed up the training of neural networks is the cost function

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} J(\mathbf{W})$$

$$J(\mathbf{W}) = \sum_{i=1}^{n} L\big(\mathbf{t}_i, \mathbf{z}(\mathbf{x}_i; \mathbf{W})\big)$$

$$L(\mathbf{t}, \mathbf{z}) = \frac{1}{2} \sum_{k} [t_k - z_k]^2$$

- above, we have used the **square of the error**

$$[t - z(\mathbf{x}; \mathbf{W})]^2$$

- a more popular choice nowadays is the cross–entropy loss

$$\boxed{-t \log z(\mathbf{x}; \mathbf{W}) + (1 - t) \log(1 - z(\mathbf{x}; \mathbf{W}))}$$

- for networks with sigmoid units, this again eliminates the dependence of the gradient on the derivative of the sigmoid

Computing the Gradient of L

$$\frac{\partial L}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\sum_k t_k \log z_k + (1-t_k)\log(1-z_k)\right]$$

$$= -\sum_k \left(\frac{t_k}{z_k} - \frac{1-t_k}{1-z_k}\right)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_k \frac{t_k - z_k}{z_k(1-z_k)}\,\frac{\partial z_k}{\partial u_k}\,\frac{\partial u_k}{\partial y_j}$$

$$= -\sum_k \frac{t_k - z_k}{z_k(1-z_k)}\, s'[u_k]\, w_{kj}$$

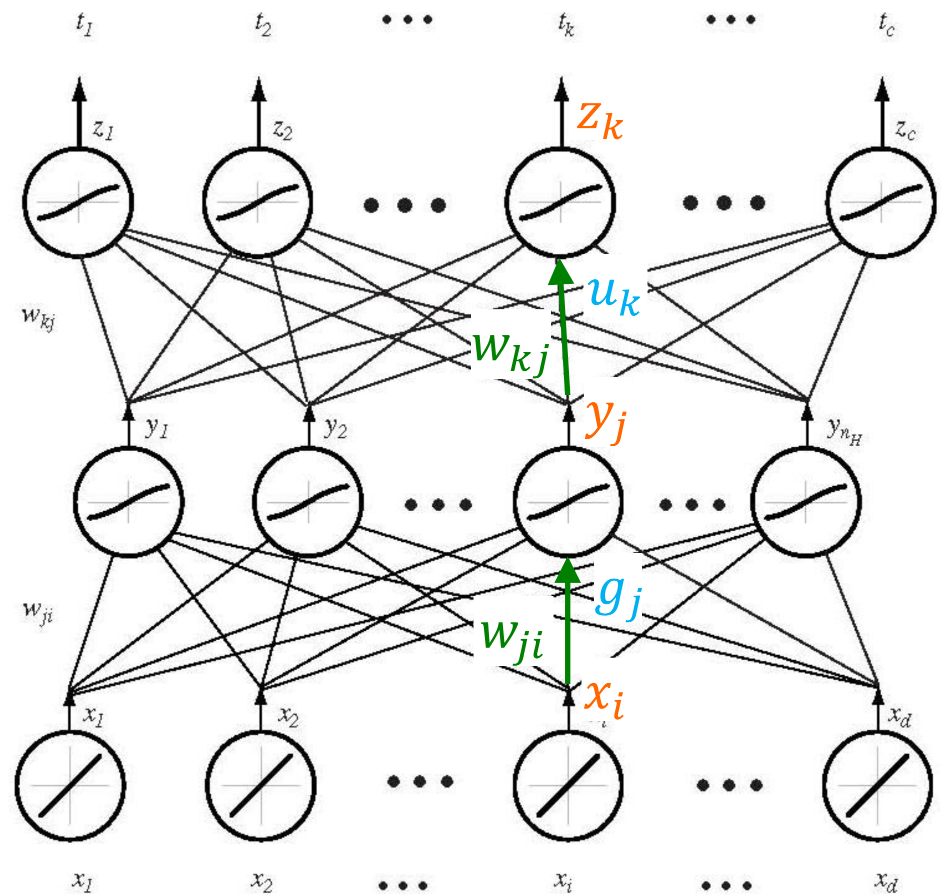
- for the sigmoid non–linearity, it can be shown that

$$\boxed{s'[u_k] = z_k(1-z_k)}$$

and

$$\frac{\partial L}{\partial y_j} = -\sum_k (t_k - z_k)\, w_{kj}$$

<u>no</u> longer depends on $s'[u_k]$

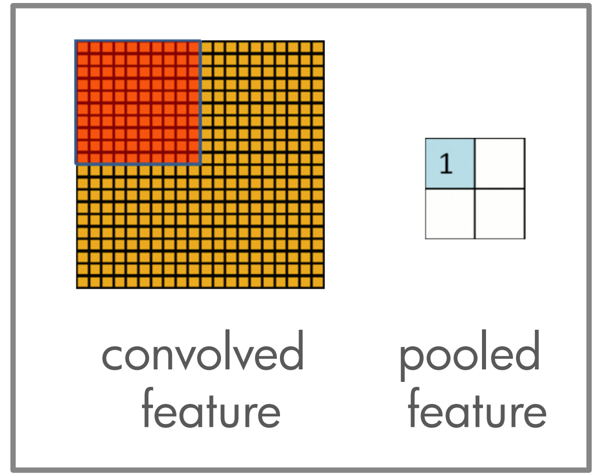


$$\boxed{z_k = s[u_k]}$$

$$\boxed{u_k = \sum_j w_{kj} y_j}$$

Architecture

- CNN usually also perform pooling
  - this consists of mapping a region of the convolution output (after the non−linearity) into a single response
  - in the example, 10×10 units are mapped into one value



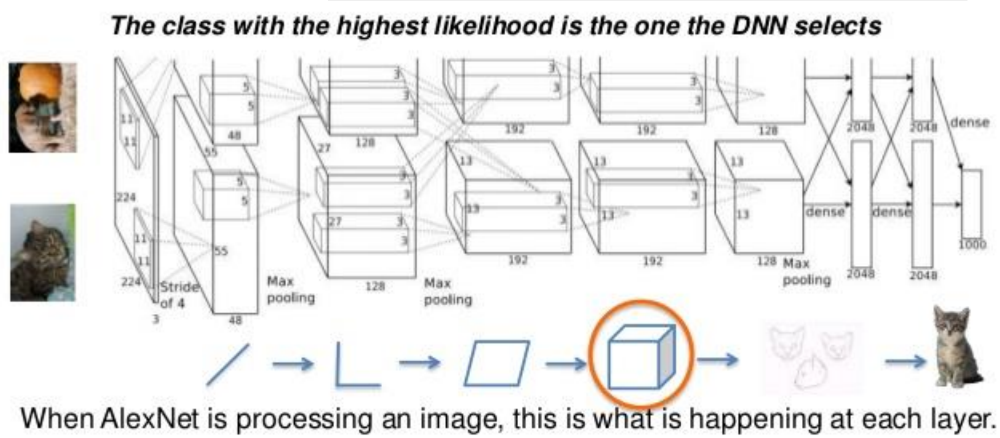
  - pooling operators are typically the average or the maximum
    - consider maximum pooling: as long as the largest response among the orange units stays the same, the pooled output is the same
  - this has two benefits
    - dimensionality reduction: from 40×40 units to 4
    - invariance: the pooled response is invariant to small shifts or changes of scale (size) of the input

Architecture

- deep learning
  - deep learning networks are networks with <u>many</u> layers
  - modern deep learning networks, usually involve a **combination** of convolutional and fully–connected layers, ReLUs, and pooling
  - this started with the AlexNet
    - 5 convolutional layers
    - some with max pooling
    - 3 fully–connected (conventional) layers
    - ReLU non–linearities
    - around 650,000 units, 60 million parameters (weights)
    - created a big buzz by significantly <u>outperforming</u> the previous best results in the problem of object recognition

AlexNet

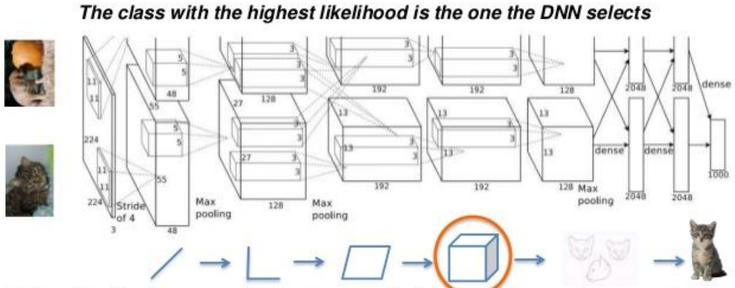
Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey; ImageNet Classification with Deep Convolutional Neural Networks. *NIPS 25*, 1097—1105, 2012.



NOTE:
Today, there are also **GoogleNet, ResNet** (**Microsoft**) - 152 layers, **VGG** (**Oxford**), etc.

Implementation

AlexNet

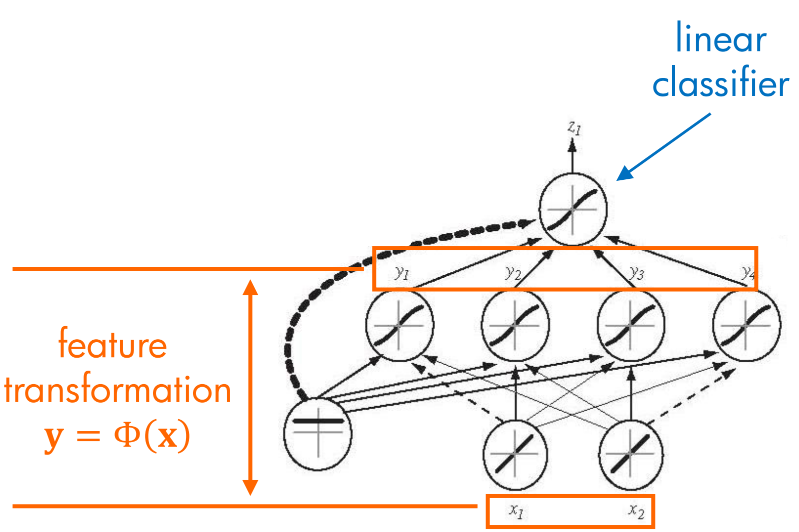


When AlexNet is processing an image, this is what is happening at each layer.

- complexity
  - as you might imagine, a network like this is **<u>not</u> easy to train or use**
  - an important development has been the **introduction** of GPUs (Graphical Processing Units)
    - these are the processors that come with your graphics board
    - they are **specialized processors** that can **perform convolutions <u>much faster</u>** than traditional CPUs
    - all modern CNNs are implemented on GPUs, using publically available software packages
    - typically, you do <u>not train a CNN</u> but <u>adapt</u> an existing one (e.g., AlexNet) to your problem
      - this consists of **initializing** your network with the existing one and running a <u>few</u> **iterations** of backpropagation on your data
      - this is called <u>fine–tuning</u>

Fine–Tuning

- MLP can be seen as: non–linear feature transformation + linear classifier
- Fine–tuning **basic idea** is
  - keep the feature transformation
  - replace the linear classifier by one **suitable** for the new problem
- **Procedure:**
  - replace the last (output) layer by one with as **many output units** as the **number of classes** in your problem
  - keep the rest of the CNN exactly the **same**
  - **run backpropagation** on the new network
  - **limit the number of iterations** so that it does not overfit (if your training set is small)



Big Data

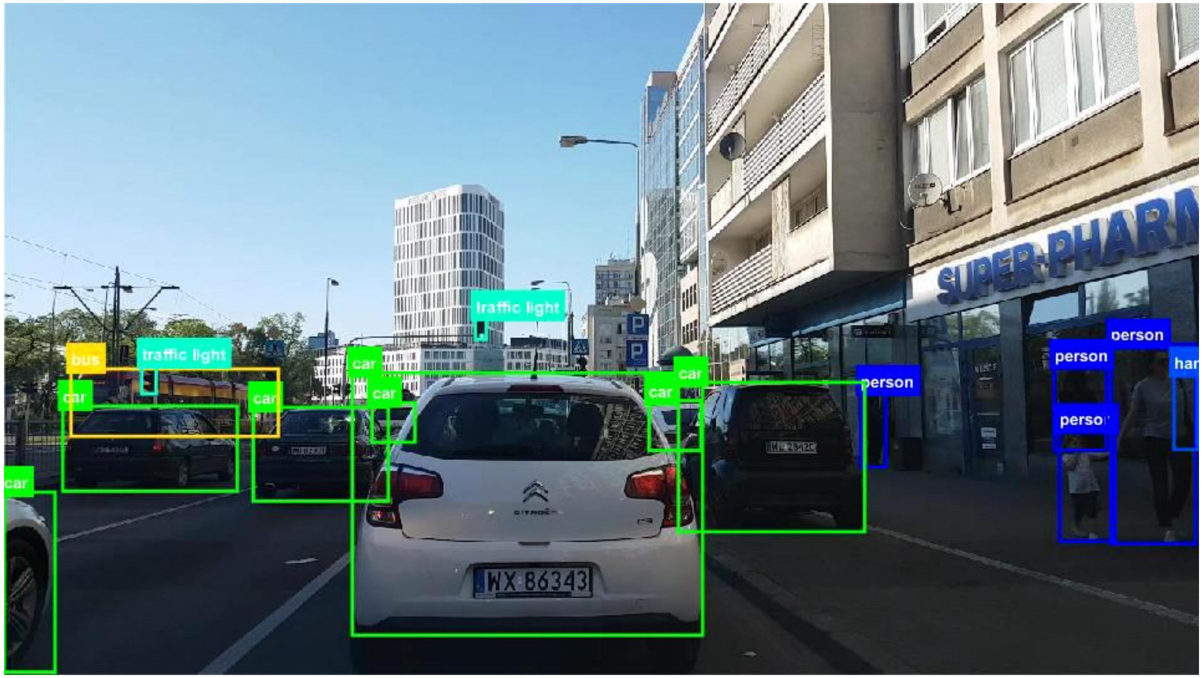
- complexity also affects learning
  - how many examples do you need to learn 60 million parameters?
  - training of these networks has only become possible with the advent of large datasets
  - the most popular one is ImageNet, which (currently) contains 14 million images of thousands of classes

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L.
ImageNet: A Large-Scale Hierarchical Image Database. *CVPR*, 2009.

  - again, you typically do **not** collect a dataset of this magnitude, but **fine–tune** the network trained on ImageNet using your (smaller) training set

Examples

- the joint results of **architecture advances**, **deep learning**, **GPUs**, and **big data** have been staggering
  - today, we have vision and speech systems that could not be imagined 5 years ago
  - tasks like object recognition now seem to be solvable



Cai, Zhaowei; Vasconcelos, Nuno; Cascade R-CNN: Delving into High Quality Object Detection. *CVPR 2018.*

https://www.youtube.com/watch?v=I9kNhXfNnHs

  - in a decade, deep learning will be in your car, phone, TV, etc.