# ECE 5470 Lecture 16b
# Machine Learning
# (Neural Networks)

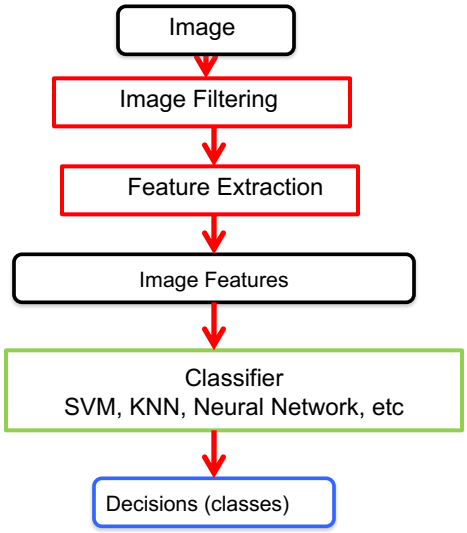
Anthony P. Reeves
Vision and Image Analysis Group
School of Electrical and Computer Engineering
Cornell University

Almost all slides are taken from presentations found on the web.
Attribution is given where possible



## Image Analysis

1. Image processing (filtering)
2. Segmentation
   (object, region of interest)
3. Feature Extraction
   or Measurement
4. Classification



## Topics: Pattern Recognition

- Pattern Recognition Examples
- Discriminant Functions
- Parametric Classifiers
- Non-Parametric Classifiers
- Neural Networks

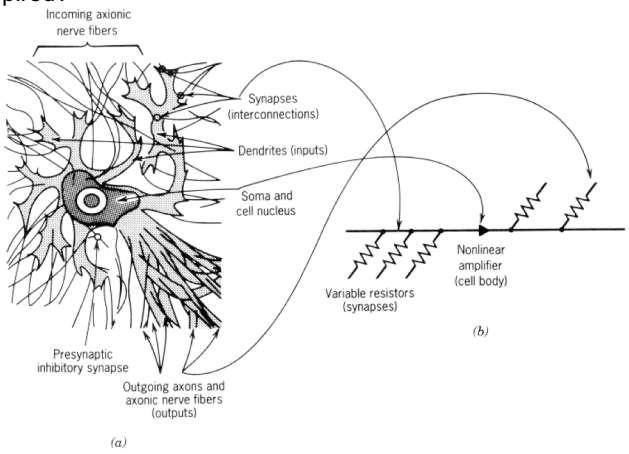


## Neural Networks

- Use elements modeled after biological neurons to form a learning network (classifier).
- Neural networks provide an alternative (adaptive, learning strategy) for performing computations and making decisions

# The Neuron

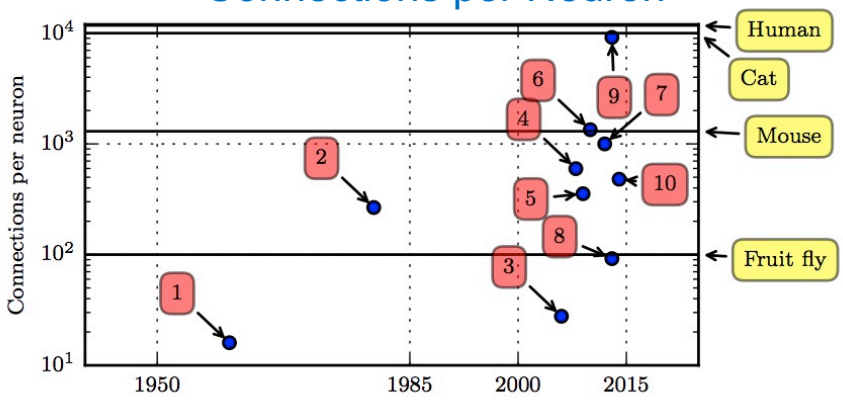
Biologically inspired?




Biological neuron Electronic analogy



# Connections per Neuron



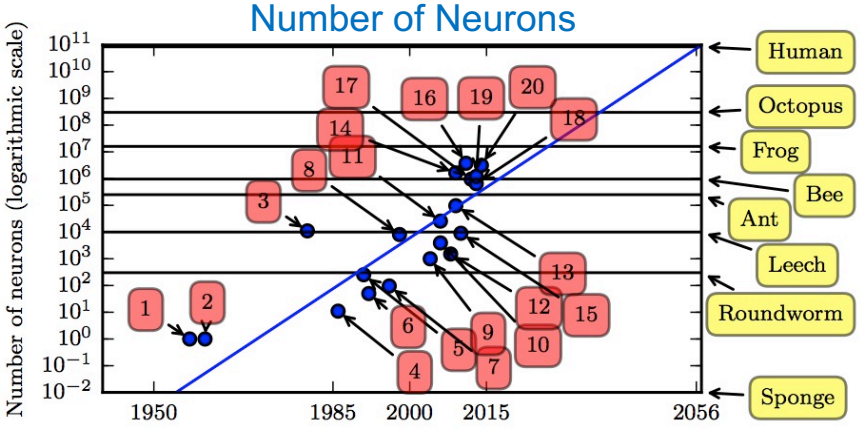

1. Adaptive linear element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)

. . .

8. Multi-GPU convolutional network (Krizhevsky et al., 2012)
9. COTS HPC unsupervised convolutional network (Coates et al., 2013)
10. GoogLeNet (Szegedy et al., 2014a)

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

## Number of Neurons



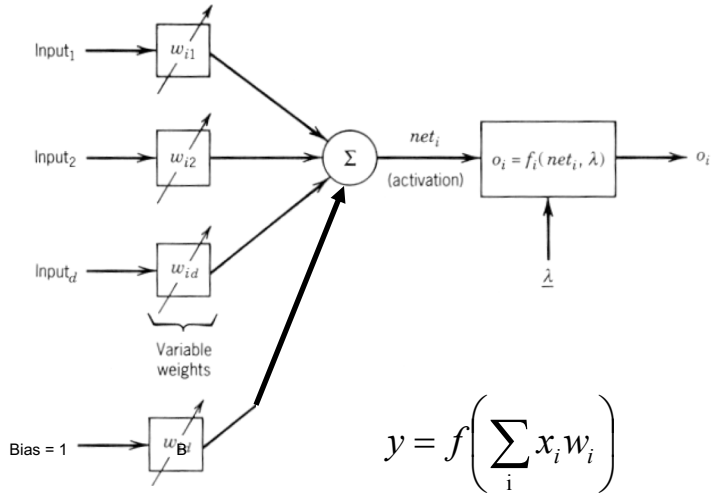

1. Perceptron (Rosenblatt, 1958, 1962)
2. 2. Adaptive linear element (Widrow and Hoff, 1960)
3. 3. Neocognitron (Fukushima, 1980)

. . .

18. Multi-GPU convolutional network (Krizhevsky et al., 2012)
19. COTS HPC unsupervised convolutional network (Coates et al., 2013)
20. GoogLeNet (Szegedy et al., 2014a)

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

## Artificial Neuron Model




$$y = f\left(\sum_{i} x_i w_i\right)$$

## Neural Network Issues

- Network Topology:
  - How to interconnect neurons
  - feedback? Structure? Multiple layers?
- The Neuron Model
- Strategy for learning or training
  - Present a set of inputs and make incremental changes until the desired behavior is achieved



## The Perceptron

- The output function is a threshold

Bias *(b)*

$$f(\underline{x}) = \begin{cases} 1 \text{ if } \sum_i x_i w_i > w_0 \\ 0 \text{ otherwise} \end{cases}$$

- This function partitions hyperspace with a hyperplane (c.f. linear discriminant functions)
- Simple topology, Convergent training algorithm
- Problem: limited functionality

R. Rosenblatt, *Principles of Neurodynamics* (1959)

## Perceptron Training

1. Initialize weights and threshold to small random values
2. Present new input *x* and the desired output
3. Calculate the actual output *y(t)*
4. Adapt weights

$$w_i(t+1) = w_i(t) + \eta(d(t) - y(t))x_i(t)$$

$$d(t) = \begin{cases} +1 \text{ if input from class A} \\ -1 \text{ if input from class B} \end{cases}$$

η is a positive gain fraction < 1 Learning Rate

5. Repeat by going to step 2



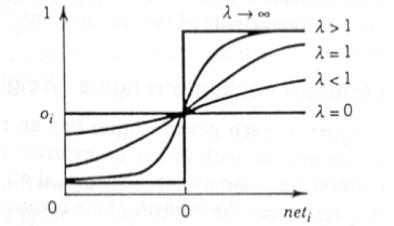
## The Perceptron

- The perceptron convergence procedure may oscillate continuously if the inputs are not separable.
  - One solution, replace threshold output function with a smoother (multilevel) function. Weights are corrected based on the difference between the desired output and the actual output
- The Perceptron cannot solve the exclusive OR problem

M. Minskey and S. Papert *Perceptrons an Introduction to Computational Geometry* (1969)

# The Sigmoid output function

$$y = \frac{1}{1 + e^{-\lambda x}}$$



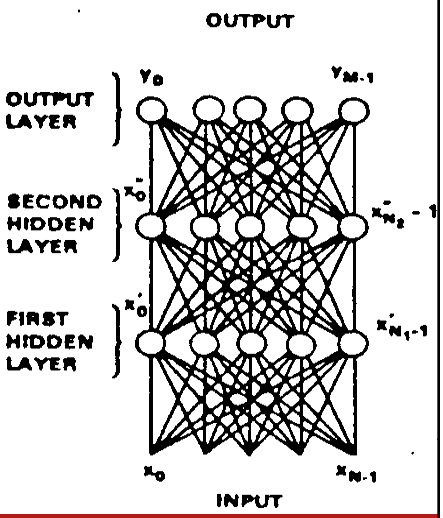

- $\lambda$ is an adjustable gain parameter
- Typically $\lambda = 1$
- $\lambda = \infty$ implies a threshold function



# Multi-layer Multi-output Neural Network

## Multi-Layer Perceptron (MLP)

- Rumelhart, Hinton and Williams *Learning Internal Representations by Error Propagation* (1986)
- Back-propagation learning algorithm
- Multi-output networks: typically use one output for each class (1 out of N code)



## Back-Propagation Training

1. Initialize weights and threshold to small random values
2. Present new input *x* and the desired outputs *d*
3. Calculate the actual output
4. Adapt weights, start at the output nodes

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x'_i$$

$$\delta_j = y_j(1-y_j)(d_j - y_j) \quad \text{for output node}$$

$$\delta_j = x'_j(1-x'_j)\sum_k \delta_k W_{jk} \quad \text{for internal node}$$

$\eta$ is a positive gain fraction < 1

5. Repeat by going to step 2



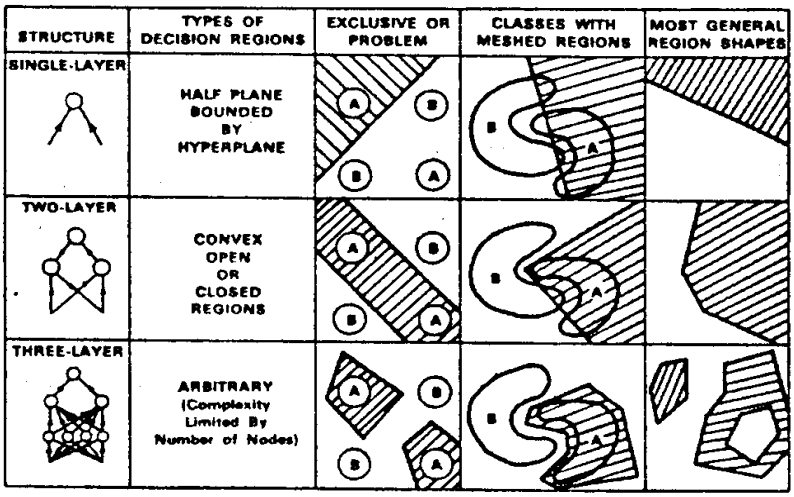
## Issues with back-propagation learning

- Slow convergence
  - **Momentum** term may be added to speed convergence

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x'_i + \alpha(w_{ij}(t) - w_{ij}(t-1))$$

  - where $0 < \alpha < 1$
- Inability to deal with contradictory training information
- Overfitting

# Capabilities of multi-layer Neural networks






# Kolmogorov's Mapping Neural Network Existence Theorem (1957)

- Any continuous function in *d* inputs and *c* outputs can be implemented by a three layer network
  - layer 1 (*d*-elements) holds the input
  - layer 2 (*2d+ 1* elements) nonlinear continuously increasing functions
  - layer 3 (*c* elements) produces the outputs
  - layers 2 and 3 elements contain nonlinear continuously increasing functions
- The theorem does not indicate how weights or nonlinearities should be selected

## Recent Neural Network Developments

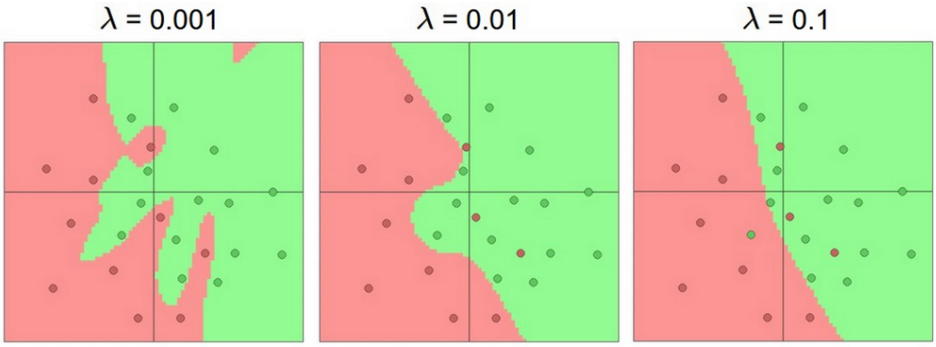
- RELU Nonlinearity
- Train in mini-batches (Stochastic gradient descent)
- Softmax Loss
- Learning rate schedule
- Dropout
- Regularization
- Data Augmentation

## Overfitting

In statistics and machine learning, **overfitting** occurs when a statistical model describes random error or noise instead of the underlying relationship.

Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data.

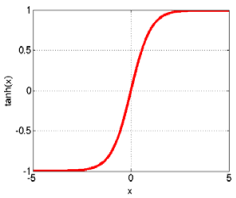
# Overfitting

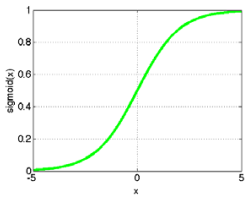


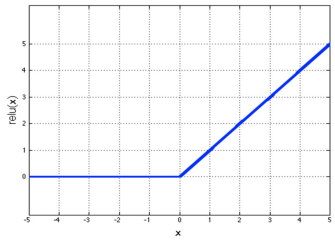



# Non-Linearity

- Per-element (independent)
- Options:
  - Tanh (1-exp(-x))/(1+exp(-x))
  - Sigmoid: 1/(1+exp(-x))
  - Rectified Linear Unit (ReLU)
    - max(0, x)
    - Preferred option for CNN
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues




# Output Functions

- **Two class problem**
  - Train for a high response to class A and a low response for class B
  - For testing no non-linearity required
  - For training can use sigmoid function for **score** to limit impact of outliers
- **Multiclass Task**
  - Use a one-out-of-N code with N outputs
  - For testing select class with highest output
  - For training Use softmax function for score

  (normalize response so that it is scaled between 0 – 1)

$$x_i \rightarrow \cdots \rightarrow f_i \rightarrow \boxed{\text{Softmax}} \rightarrow p_i$$

*(input)* *(scores)* *(probabilities)*

$$p_{i,j} = \frac{e^{f_{i,j}}}{\sum_{k}^{N} e^{f_{i,k}}}$$

- **Regression Task**
  - To predict a real value no non-linear function is required



# Loss Functions

Training: Minimize the Loss function **L** based on the **score.**

Use different loss functions for different tasks:

- **Softmax loss** $L_i$ is used for predicting a single class of K mutually exclusive classes.

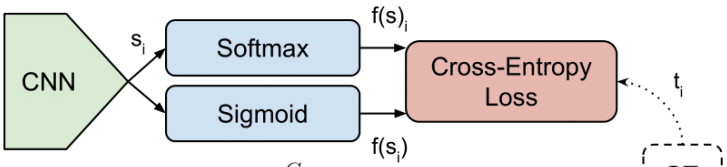
$$p_{i,j} = \frac{e^{f_{i,j}}}{\sum_{k} e^{f_{i,k}}} \quad \text{(Softmax)}$$

$$L_i = -\log p_{i,y_i} \quad \text{(Cross-entropy)}$$

Probability for correct output

- **Euclidean loss** is used for regressing to real-valued labels [-inf,inf]
- (**Sigmoid cross-entropy loss** is used for predicting K independent probability values in [0,1]. )

# Cross-Entropy Loss




$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$C'=2$

Where $t_i$ and $s_i$ are the ground truth and the CNN score for each class i in C

In a **binary classification problem**, where C′=2

$$CE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) + (1 - t_1) log(1 - s_1)$$

For the general (not 1-hot multilabel case) $$CE = -\sum_{i}^{C} t_i log(s_i)$$

# Cross-Entropy Loss

In the specific (and usual) case of **Multi-Class classification** the labels are one-hot, so only the positive class Cp keeps its term in the loss. There is only one element of the Target vector t which is not zero $t_i=t_p$. So discarding the elements of the summation which are zero due to target labels, we can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Where **Sp** is the CNN score for the positive class.

The derivative with respect to the positive class is:

$$\frac{\partial}{\partial s_p}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \left(\frac{e^{s_p}}{\sum_j^C e^{s_j}} - 1\right)$$

The derivative with respect to the other (negative) classes is:

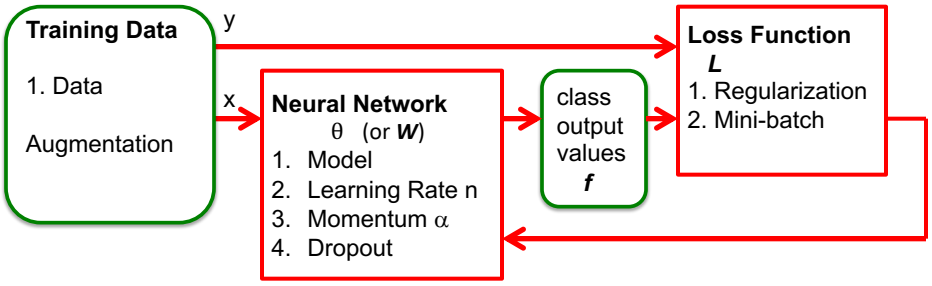
$$\frac{\partial}{\partial s_n}\left(-log\left(\frac{e^{s_n}}{\sum_j^C e^{s_j}}\right)\right) = \left(\frac{e^{s_n}}{\sum_j^C e^{s_j}}\right)$$

Where $s_n$ is the score of any negative class in C different from $C_p$.

**Logistic Loss** and **Multinomial Logistic Loss** are other names for **Cross-Entropy loss**.

https://gombru.github.io/2018/05/23/cross_entropy_loss/

## Neural Network Training




x Input vector
y Label (true value)
θ Network parameters (also ***W***)
f Score
L Loss function

**Network Operations**

Forward propagation $f = f_{infer}(x, \theta)$

Backward propagation $\theta = f_{learn}(L, \theta)$



## Neural Network Training

**Stochastic Gradient Descent**

(Stochastic: => train in mini-batches rather than individually)

- Learning rate η
- Learning rate schedule (decrease η with time)
- Momentum α
- Mini-batch size

  Compute responses for a small set of inputs and train at a time
- Regularization
- Dropout
- Number of cycles of training (epochs)

Input conditioning techniques:
- Data Augmentation

Initialization
- Weight initialization

# Learning Rate (η) Schedule

**How do we change the learning rate over time?**

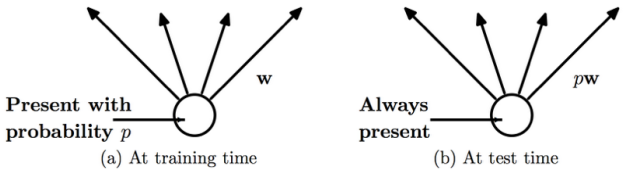
**Various choices:**

- Step down by a factor of 0.1 every 50,000 mini-batches (used by SuperVision [Krizhevsky 2012])
- Decrease by a factor of 0.97 every epoch (used by GoogLeNet [Szegedy 2014])
- Scale by sqrt(1-t/max_t) (used by BVLC to re-implement GoogLeNet)
- Scale by 1/t
- Scale by exp(-t)



# Dropout

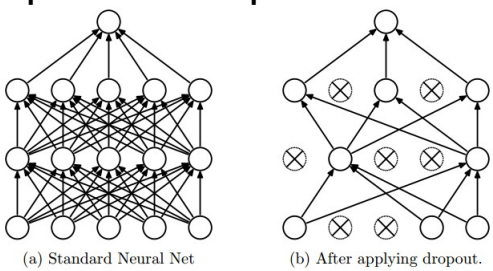
**Simple but powerful technique to reduce overfitting:**




[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

# Dropout

**Simple but powerful technique to reduce overfitting:**



(a) Standard Neural Net (b) After applying dropout.

**Note:** Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

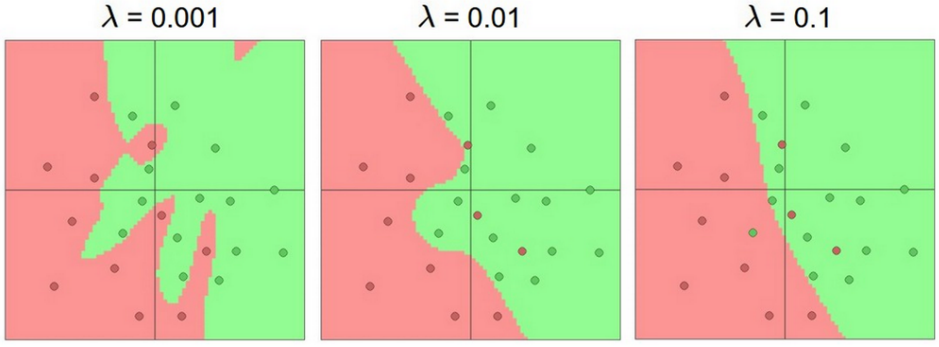
[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]



# Regularization

**Regularization reduces overfitting:**

$$L = L_{\text{data}} + L_{\text{reg}} \qquad L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2$$




[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

# Regularization

**L2 regularization** $L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2$

(L2 regularization encourages small weights)

**L1 regularization** $L_{\text{reg}} = \lambda ||W||_1 = \lambda \sum_{ij} |W_{ij}|$

(L1 regularization encourages sparse weights:
weights are encouraged to reduce to exactly zero)

**"Elastic net"** $L_{\text{reg}} = \lambda_1 ||W||_1 + \lambda_2 ||W||_2^2$

(combine L1 and L2 regularization)

**Max norm**

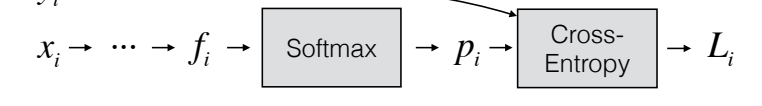
Clamp weights to some max norm $||W||_2^2 \leq c$

# Example: Softmax (for N inputs)

Let's assume we are using Softmax and Cross-entropy loss
(together this is often called "Softmax loss")

*(ground truth labels)* $y_i$

(here, "i" are different examples)



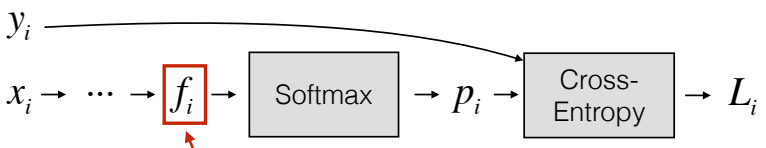

*(input)* *(scores)* *(probabilities)* *(loss)*

$$p_{i,j} = \frac{e^{f_{i,j}}}{\sum_k e^{f_{i,k}}}$$ (Softmax)

$$L_i = -\log p_{i,y_i}$$ (Cross-entropy)

$$L = \frac{1}{N} \sum_i L_i$$ (Avg. over examples)

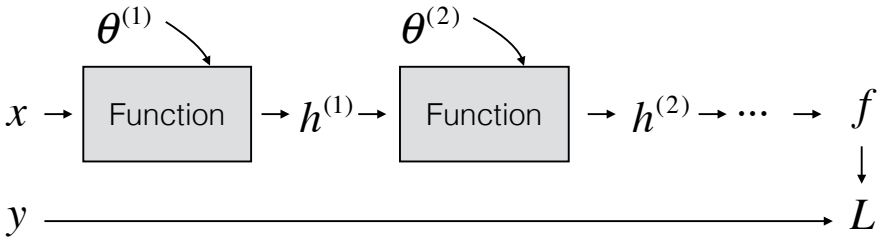
# Example: Softmax (for N inputs)




Derivative: $\frac{\partial L}{\partial f_{i,j}} = \frac{p_{i,j} - t_{i,j}}{N}$ where $t_i = [0 \ \ldots 1 \ \ldots 0]$ (Entry $y_i$ set to 1)

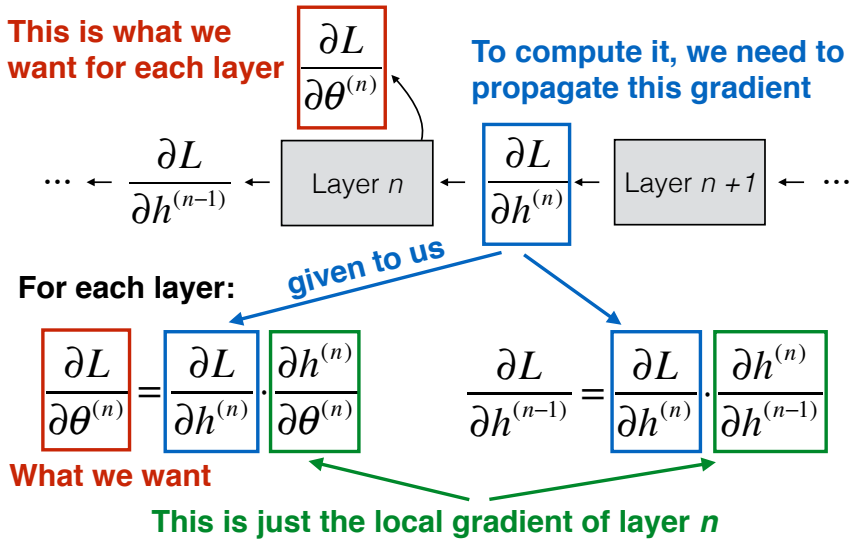
softmax loss

# Example: General Model




- **Goal:** Find a value for parameters ($\theta^{(1)}$, $\theta^{(2)}$, …), so that the loss (L) is small

## Backpropagation using the Chain Rule




## Neural Networks Summary

- Perceptrons
  - similar properties to linear discriminant functions.
  - Training may require a lot of time and result may not be optimal
- Multi-layer feed-forward networks (3 layers)
  - may require a very long time to train
  - can implement arbitrary complexity (depending upon number of elements in hidden layer)
  - may be faster and simpler than non-parametric classifiers (but require retraining for new data)