– SUS 2020–
Lecture 7: Neural Networks and Deep Learning

Hung Son Nguyen

MIM
University of Warsaw

Based on slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

Outline

Overview

- Last class we saw how basis expansion and kernels could be used to increase the capacity of linear models in a controlled way.
- **Question:** What is the primary weakness of this approach?
- In this lecture, we'll see how artificial neural networks can learn appropriate feature representations along with decision boundaries to maximize classification accuracy.
- We'll start with the inspiration for artificial neural networks: your brain.

Outline

Your Brain

Brain by the numbers

100: Number, in billions, of neurons in a human brain

100: Estimated number, in terabytes, of information it can store

1: Number, in terabytes, of information a typical desktop computer can store

2: Percentage of the body's weight represented by the brain

20: Percentage of the body's energy used by the brain

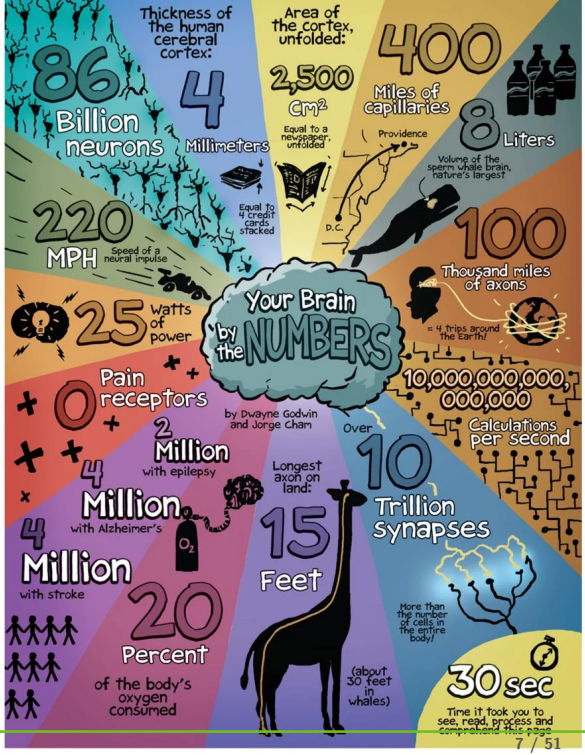
303: Highest number of random digits memorized at the 2012 USA Memory Championship, a record

10: Approximate percentage drop, in one study, in the accurate recall of random letters as a result of chewing gum

50: Percentage of times that human volunteers successfully recalled a sequence of five numbers presented briefly on a computer screen

80: Percentage of times that a chimpanzee named Ayumu succeeded at the same task

Somehow,
the brain is
greater than
the sum of its parts



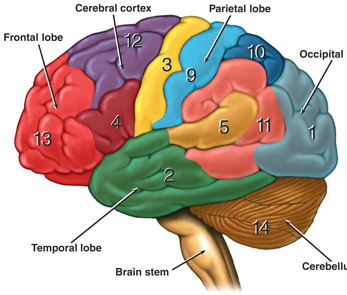
Anatomy and Functional Areas of the Brain

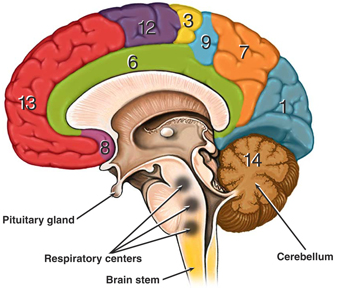
Functional Areas of the Cerebral Cortex

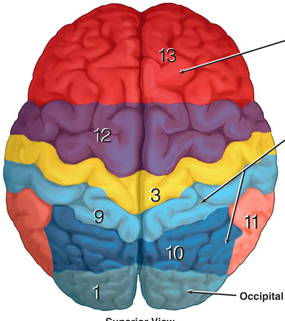
1. **Visual Area:**
   Sight
   Image recognition
   Image perception
2. **Association Area**
   Short-term memory
   Equilibrium
   Emotion
3. **Motor Function Area**
   Initiation of voluntary muscles
4. **Broca's Area**
   Muscles of speech
5. **Auditory Area**
   Hearing
6. **Emotional Area**
   Pain
   Hunger
   "Fight or flight" response
7. **Sensory Association Area**
8. **Olfactory Area**
   Smelling
9. **Sensory Area**
   Sensation from muscles and skin
10. **Somatosensory Association Area**
    Evaluation of weight, texture, temperature, etc. for object recognition
11. **Wernicke's Area**
    Written and spoken language comprehension
12. **Motor Function Area**
    Eye movement and orientation
13. **Higher Mental Functions**
    Concentration
    Planning
    Judgment
    Emotional expression
    Creativity
    Inhibition

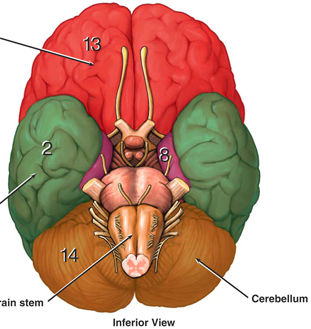
Functional Areas of the Cerebellum

14. **Motor Functions**
    Coordination of movement
    Balance and equilibrium
    Posture

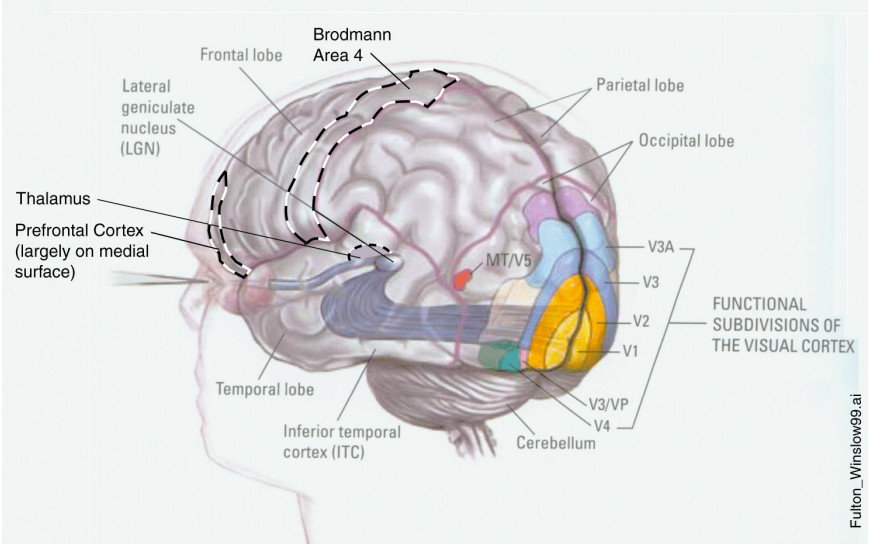






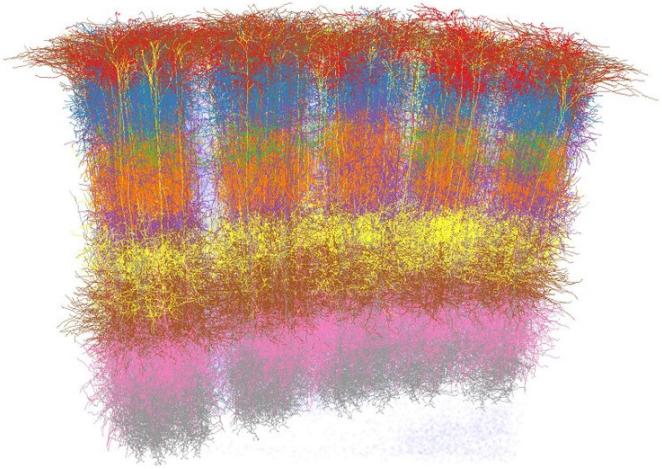


Visual Areas

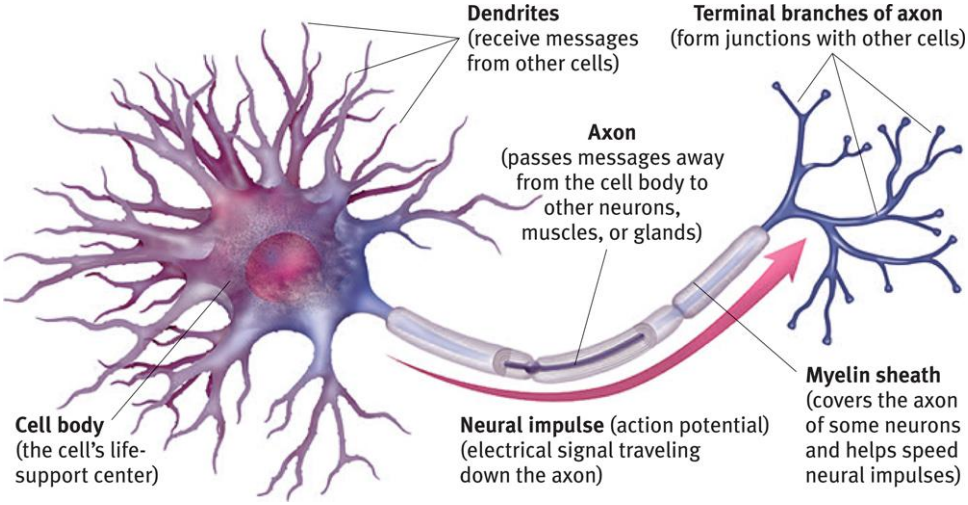


Optic nerve contains about 1 Million nerve fibers.

Cortical Columns

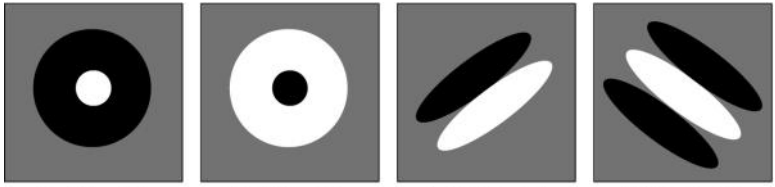


Neurons



Your brain contains about 100 Billions neurons.

Visual Receptive Fields

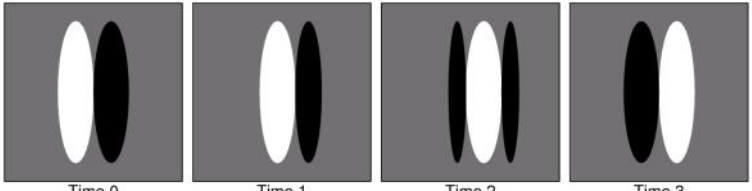


(*a*) ON cell in retina or LGN

(*b*) OFF cell in retina or LGN

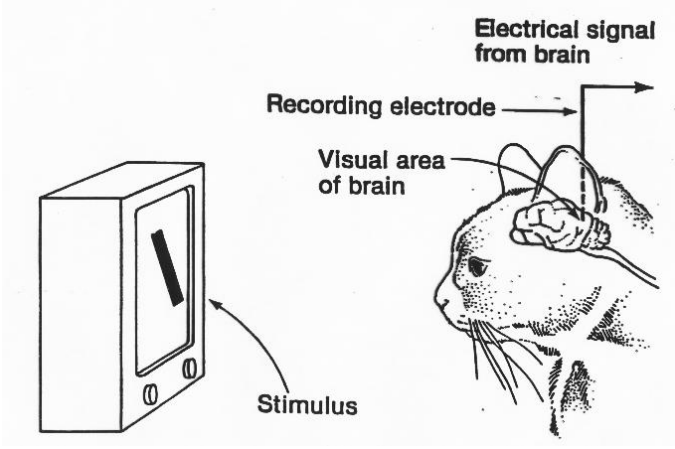
(*c*) 2-lobe V1 simple cell

(*d*) 3-lobe V1 simple cell



(*e*) Spatiotemporal RF of a V1 cell

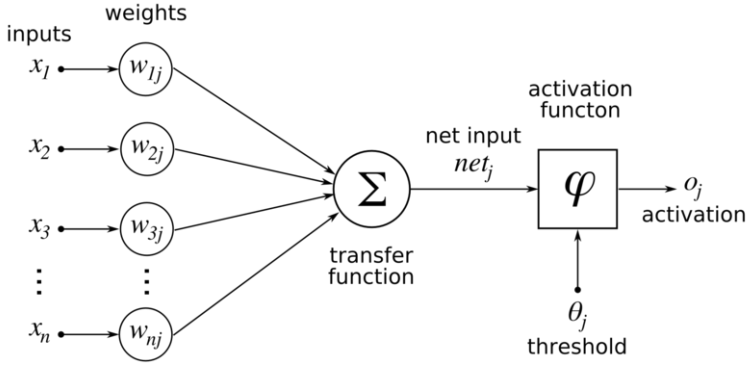
Mapping Receptive Fields in Live Subjects



Hubel and Wiesel earned the Nobel Prize in Physiology or Medicine in 1981 for this work.

Outline

McCulloch and Pitts Neuron (1943)



Assuming $\varphi() = \text{sign}()$, what model is this?

The Perceptron (1950)

The Perceptron is a simple online algorithm for adapting the weights in a McCulloch/Pitts neuron. It was developed in the 1950s by Rosenblatt at Cornell.

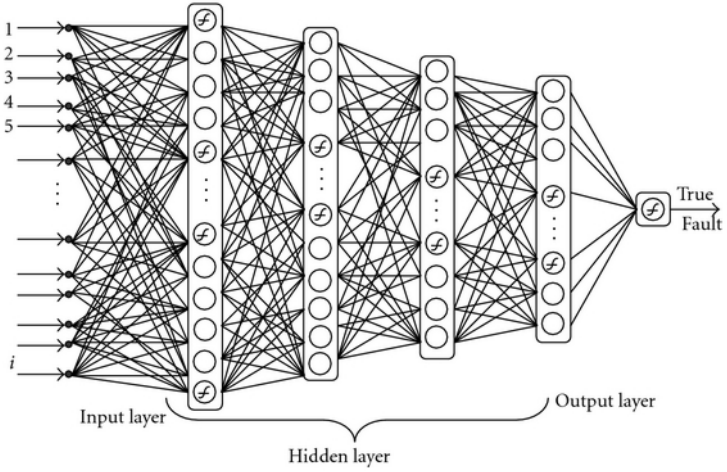
Algorithm PERCEPTRONTRAIN(**D**, *MaxIter*)

```
w_d ← 0, for all d = 1 ... D                          // initialize weights
b ← 0                                                 // initialize bias
for iter = 1 ... MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b                   // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + y x_d, for all d = 1 ... D    // update weights
            b ← b + y                                 // update bias
        end if
    end for
end for
return w_0, w_1, ..., w_D, b
```

Limitations of Single Layer Perceptrons

- The representational limitations of the single-layer perceptron were not well understood at first in the AI community.
- In 1969, Minsky and Papert at MIT popularized a set of arguments showing that the single-layer perceptron could not learn certain classes of functions (including XOR).
- They also showed that more complex functions could be represented using a *multi-layer perceptron* or MLP, but no one knew how to learn them from examples.
- This led to a shift away from mathematical/statistical models in AI toward logical/symbolic models.

Multi-Layer Perceptron



Sigmoid Neural Networks

The solution to MLP learning turned out to be:

1. Make the hidden layer non-linearities smooth (sigmoid/logistic) functions:

$$h_k^{(1)} = \frac{1}{1 + \exp(-(\sum_d w_{dk} x_d + b_{dk}))} \tag{1}$$

$$h_k^{(i)} = \frac{1}{1 + \exp(-(\sum_l w_{lk} h_l^{(i-1)} + b_{lk}))} \tag{2}$$

2. Make the output layer non-linearity a smooth (sigmoid/logistic/softmax) function.

3. Use standard numerical optimization methods (gradient descent) to learn the parameters. The algorithm is known as Backpropagation and was popularized by Rumelhart, Hinton and Williams in the 1980s.

Sigmoid Neural Network Properties

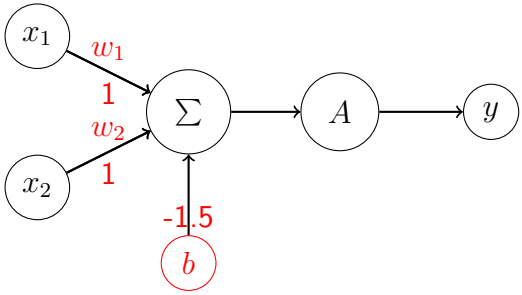
- It can be shown that a sigmoid network with one hidden layer (of unbounded size) is a universal function approximator.
- In terms of classification, this means neural networks with one hidden layer (of unbounded size) can represent any decision boundary and thus have infinite capacity.
- It was also shown that deep networks can be exponentially more efficient at representing certain types of functions than shallow networks.
- Demo!

More Failures

- Through the 1990s is became clear that while these models could represent arbitrarily complex functions and that deep networks should work better than shallow networks, no one could effectively train deep networks.
- This led to a shift away from neural networks towards probabilistic/statistical models and SVMs through the late 1990s and 2000s.
- Only a few groups continued to work on neural network models during this time period (Hinton, LeCun, Bengio, Ng).

Computation example

Binary AND gate



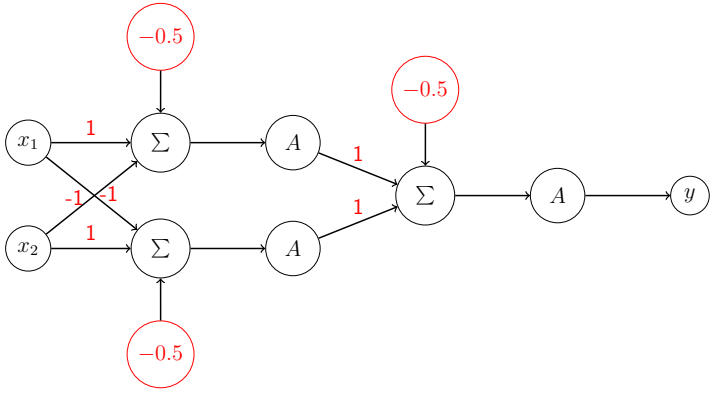
We want to set w_1, w_2 and b such that:

$$A(w_1 x_1 + w_2 x_2 + b) = x_1 \text{ AND } x_2$$

$$x_0 = 0, x_1 = 1. \quad y = A(0 + 1 - 1.5) = A(-0.5) = 0$$

Computation example

Binary XOR gate



Model complexity

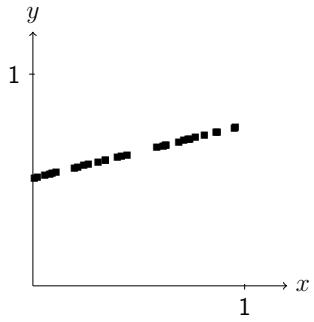
One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters
- dogs vs cats pictures (VGG16 network): 138,357,544 parameters

We should probably look for a way to tune these parameters automatically otherwise it is going to get *really* boring *really* quickly.

Parameter tuning - Line approximation

We have a few sample points and we want to find a line that approximate these points.



Our model is just a line

$$y_{pred} = ax + b$$

We want to find a and b to best match our samples.

Parameter tuning - Gradient descent

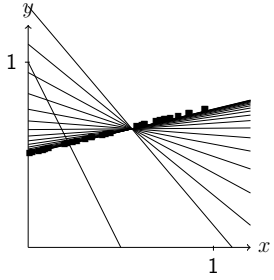
First we have a define a loss that measures how good our predictions are.

$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

and now, we compute how the loss is affected by small changes of a and b:

$$\frac{dl}{da} = 2x(ax + b - y) \qquad \frac{dl}{db} = 2(ax + b - y)$$

We start with random values: $a = -2$ and $b = 1$



Then, we compute the average of the gradient along all (x, y) couples

And update a and b by substracting a small proportion of their gradient.

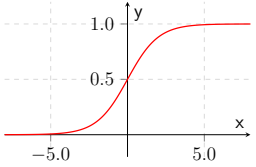
Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

Problem: The function computed by our neural network is not differentiable because of the activation function.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

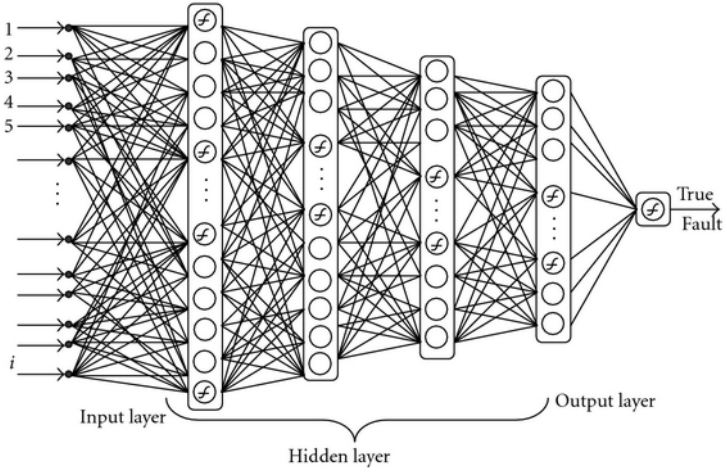
Solution: We replace it by a differentiable function that does the same job.

$$A(x) = \frac{1}{1+e^{-x}}$$



Outline

Multi-Layer Perceptron



Neural Network Regression

To convert an MLP from classification to regression, we only need to change the output activation function from logistic to linear.

- The hidden layer non-linearities are smooth functions:

$$h_k^1 = \frac{1}{1 + \exp(-(\sum_d w_{dk}^1 x_d + b_{dk}^1))}$$
$$h_k^i = \frac{1}{1 + \exp(-(\sum_l w_{lk}^i h_l^{(i-1)} + b_{lk}^i))} \text{ for } i = 2, ..., L$$

- The output layer activation function is a linear function:

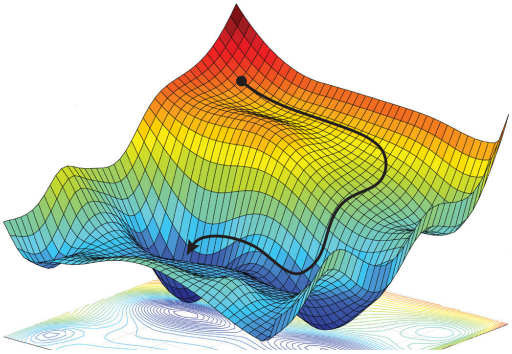
$$\hat{y} = \sum_l w_l^o h_l^L + b^o$$

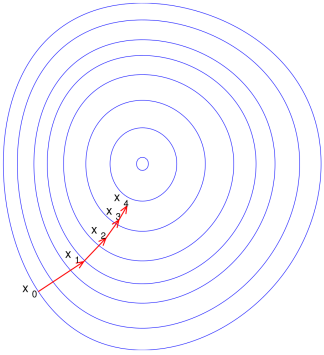
Learning

Let θ be the complete collection of parameters defining a neural network model. Our goal is to find the value of θ that minimizes the MSE on the training data set $\mathcal{D} = \{y_i, \mathbf{x}_i\}_{i=1:N}$

$$\mathcal{L}_{MSE}(\mathcal{D}|\theta) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

Gradient descent





We start with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \ n \geq 0.$$

Gradients

We need the gradient with respect to each of the parameters. Let's begin with w_l^o:

$$\frac{\partial \mathcal{L}_{MSE}(\mathcal{D}|\theta)}{\partial w_l^o} = \frac{\partial}{\partial w_l^o} \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 = 0 \tag{3}$$

$$= \frac{2}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n) \frac{\partial \hat{y}_n}{\partial w_l^o} \tag{4}$$

$$= \frac{2}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n) h_l^L \tag{5}$$

Gradients

It's also useful to define the derivatives wrt the hidden units for a single data case:

$$\varepsilon_k^L = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^L} = \frac{\partial}{\partial h_k^L}(y - \hat{y})^2 \tag{6}$$

$$= 2(y - \hat{y})\frac{\partial \hat{y}}{\partial h_k^L} \tag{7}$$

$$= 2(y - \hat{y})w_k^o \tag{8}$$

In general, we can define: $\varepsilon_k^j = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^j}$

Gradients

Suppose we're trying to compute the derivative with respect to the weight w_{kl}^j for some layer j and assume we have ε_l^j computed for all hidden units l in layer j.

$$\frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial w_{kl}^j} = \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_l^j} \frac{\partial h_l^j}{\partial w_{kl}^j} \tag{9}$$

$$= \varepsilon_l^j h_l^j (1 - h_l^j) h_k^{j-1} \tag{10}$$

The total derivative is then given by:

$$\frac{\partial \mathcal{L}_{MSE}(\mathcal{D}|\theta)}{\partial w_{kl}^j} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}_{MSE}(y_n, \mathbf{x}_n|\theta)}{\partial w_{kl}^j}$$

Gradients

Suppose we're trying to compute the error with respect to hidden unit k in layer $j-1$ and assume we have ε_l^j computed for all hidden units l in layer j.

$$\frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_k^{j-1}} = \sum_l \frac{\partial \mathcal{L}_{MSE}(y, \mathbf{x}|\theta)}{\partial h_l^j} \frac{\partial h_l^j}{\partial h_k^{j-1}} \tag{11}$$

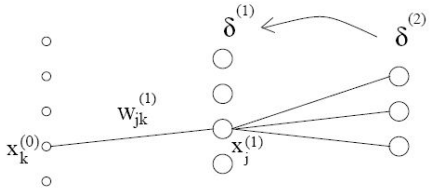
$$= \sum_l \varepsilon_l^j h_l^j (1 - h_l^j) w_{kl}^{j-1} \tag{12}$$

Backpropagation

- The Backpropagation algorithm works by making a forward pass through the network for each data case and storing all the hidden unit values.
- The algorithm then computes the error at the output and makes a backward pass through the network computing the derivatives with respect to the parameters as well as the contribution of each hidden unit to the error. These are the ε_k^j values.
- The complete computation is just an application of the chain rule with caching of intermediate terms in the neural network graph structure.

Until satisfied, Do

- **For each training example, Do**
  1. **Input the training example to the network and compute the network outputs**



2. **For each output unit** k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. **For each hidden unit** h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

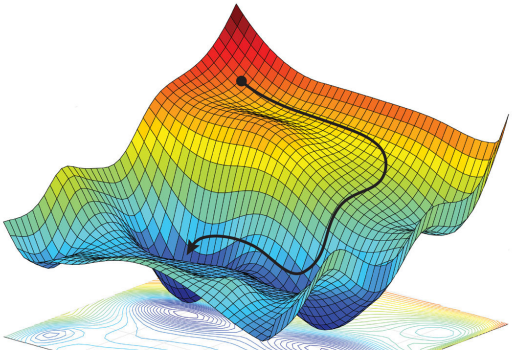
4. **Update each network weight** $w_{i,j}$

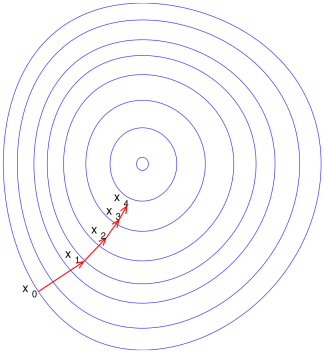
$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Gradient descent





We start with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \ n \geq 0.$$

Optimisation Algorithms

Optimisation Algoritms are used to update weights and biases i.e. the internal parameters of a model to reduce the cost function (error). They can be divided into three categories:

1. Gradient descent variants
   - batch gradient descent
   - stochastic gradient descent
   - mini-batch gradient descent
2. Momentum
   - Nesterov accelerated gradient (Nesterov momentum)
3. Algorithms with adaptive learning rates
   - AdaGrad
   - AdaDelta
   - RMSprop
   - Adam
   - AdaMax
   - Nadam
   - AMSGrad

Learning modes

Gradient descent step:

$$\theta^{next} = \theta^{cur} - \eta \nabla_\theta MSE(\theta^{cur})$$

In order to compute gradient descent, we need to compute the gradient $\nabla_\theta \mathcal{L}_{MSE}(\theta)$ of the cost function with regards to each model parameter θ_j.

- **Batch Gradient Descent:** The standard formula involves calculations over full training set X, at each gradient decent step. Still gradient descent is faster for many features than normal equation.
- **Stochastic Gradient Descent:** *picks a random instance in the training set at every step and computes the gradients based only on thatinstance.*
  - *it is much faster than Batch version*
  - *random nature $\implies$ it is much less regular than Batch Gradient Descent*
  - *good advantage: when the cost function is irregular, the randomness can help jump out of local minima.*
- **Mini-Batch Gradient Descent:** *at each step, we compute the gradients on small random sets of instances called mini-batches. It will end up walking a bit closer to minimum compared to SGD, but it may be harder for him to escape local minima*

Trade-Offs

- Neural network regression has low bias, but high variance.
- The objective function has local optima and requires iterative numerical optimization using backpropagation to compute the gradients, which can be slow.
- Making predictions with trained models can be very fast.
- Capacity control in these models can be crucial. The capacity parameters are the depth of the network and the size of each layer.
- These models can also be trained using ℓ_2 or ℓ_1 regularization or the more recent dropout scheme as an alternative to controlling network structure.

Outline

Deep Learning

The solution to the deep learning problem (as of right now) appears to be:

1. Have access to lots of labeled data (ie: millions of examples).
2. Make the non-linearities non-smooth again (rectified linear units, ReLU):

$$h_k^{(1)} = \max(0, \sum_d w_{dk} x_d + b_{dk})) \tag{13}$$

$$h_k^{(i)} = \max(0, \sum_l w_{lk} h_l^{(i-1)} + b_{lk})) \tag{14}$$

Deep Learning

The solution to the deep learning problem (as of right now) appears to be:

1. Have access to lots of labeled data (ie: millions of examples).
2. Make the non-linearities non-smooth again (rectified linear units, ReLU):

$$h_k^{(1)} = \max(0, \sum_d w_{dk} x_d + b_{dk})) \tag{15}$$

$$h_k^{(i)} = \max(0, \sum_l w_{lk} h_l^{(i-1)} + b_{lk})) \tag{16}$$

Deep Learning

The solution to the deep learning problem (as of right now) appears to be:

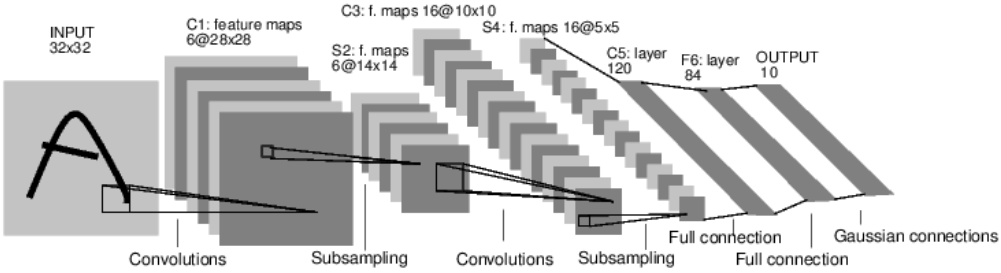
3. Use somewhat improved optimization methods for non-smooth functions (accelerated stochastic sub-gradient descent) to learn the parameters.
4. Use new regularization schemes based on randomly zeroing-out nodes in the network.
5. Do the computing on a GPU with 1000s of cores and 10s of GB of RAM for a massive 20-30X speedup (go SIMD!). Model training takes 10 days for large vision problems instead of 1 year.

Deep Learning Applications

- Deep learning models using these basic ingredients have become dominant in AI over the last few years starting with computer vision and moving on to other areas including speech recognition.
- Many companies use deep learning-based vision systems to analyze photos and images.
- Google and others switched to deep-learning based speech recognition systems after it was shown to lead to substantial reductions in the word error rate.

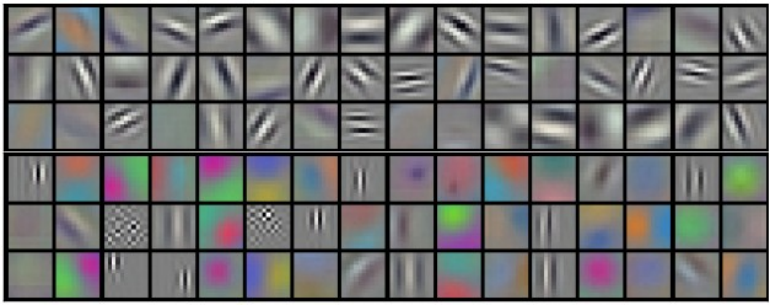
Deep Learning For Vision

Deep learning for vision uses a modified deep architecture called a *convolutional network* or convnet that learns small patches of weights (or filters) and replicates (convolves) them over an image.

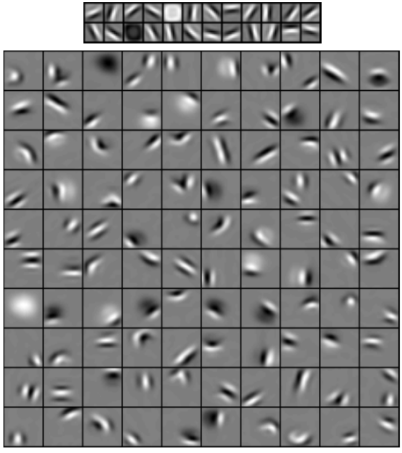


This architecture is also from the 1980s and was directly inspired by the structure of the visual areas of the brain.

Learned Receptive Fields



Learned Receptive Fields



faces, cars, airplanes, motorbikes