Advanced Machine Learning
#L3
Neural Network

袁彩霞

yuancx@bupt.edu.cn

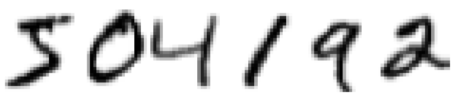
智能科学与技术中心

Topics for Today

- Motivating example: digit recognition
- Two important types of artificial neuron
  - the perceptron
  - the sigmoid neuron
- Neural networks
  - Architecture and representation
  - A simple network to classify handwritten digits
- Learning with gradient descent
- Backpropagation algorithm

Motivating example: digit recognition

- Consider the following sequence of handwritten digits:



- How can we humans recognize it?
  - We carry in our heads a supercomputer, and superbly adapted to understand the visual world

Motivating example: digit recognition

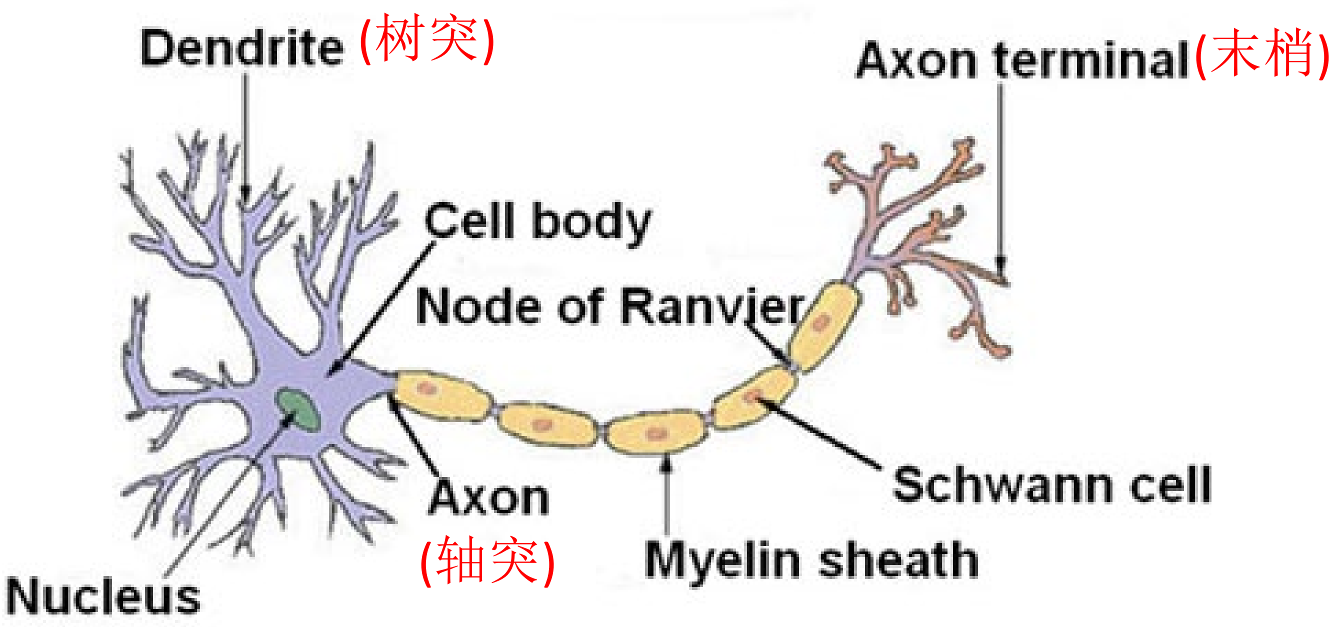
- How do we develop a system which can learn from those training examples?

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

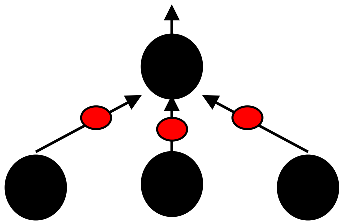
Neural Networks

- How the brain works?
  - Neuron in the brain



Neural Networks

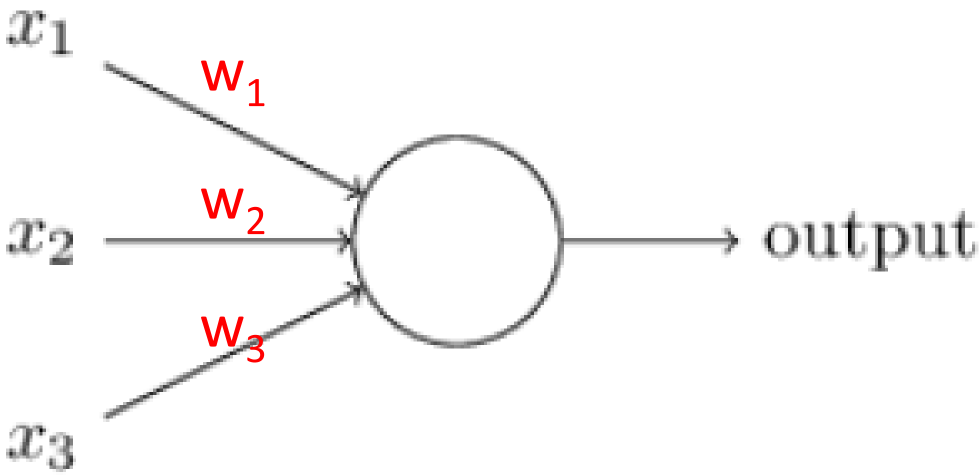
- How the brain works?



- Each neuron receives inputs from other neurons
  - A few neurons also connect to receptors.
  - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
  – The weights can be positive or negative.
- The synaptic weights adapt so that the whole network learns to perform useful computations
  – Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^{4} weights.
  – A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.

Neuron model: Perceptron

- A perceptron takes several binary inputs x_1, x_2, … and produces a single binary output:

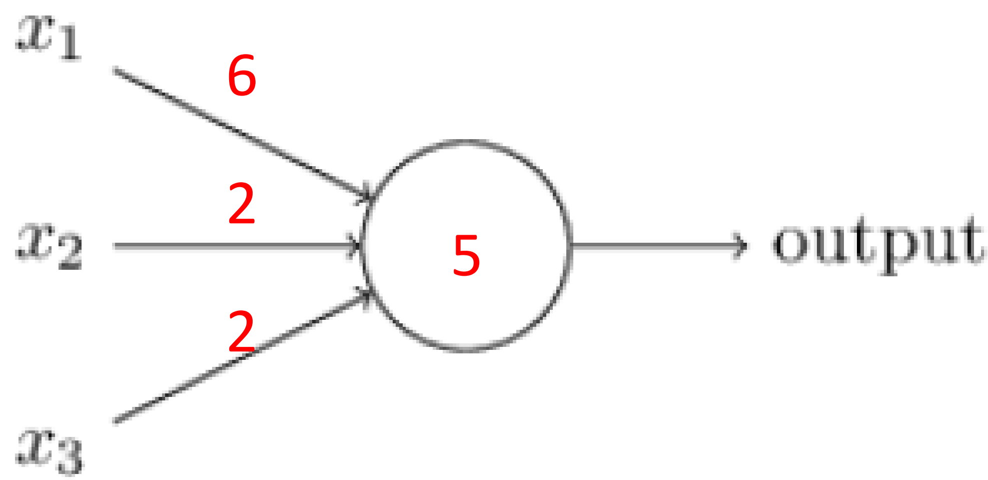


3 elements: Input   weight   output

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Neuron model: Perceptron

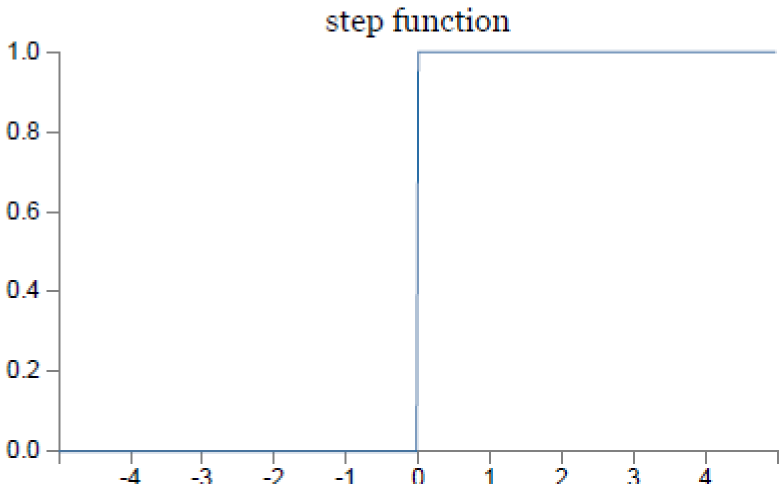
- It works like a device that makes decisions by weighing up evidence
- An example: whether or not to go to play tennis:
- Three factors:
  - 1. Is the weather goc
  - 2. Does your boyfrie
    you?
  - 3. Is the playground
    car)
- Choose a weight: w_1=
- Choose a threshold o
- (By varying the weights and the threshold, we can get different models of decision making.)



Neuron model: Perceptron

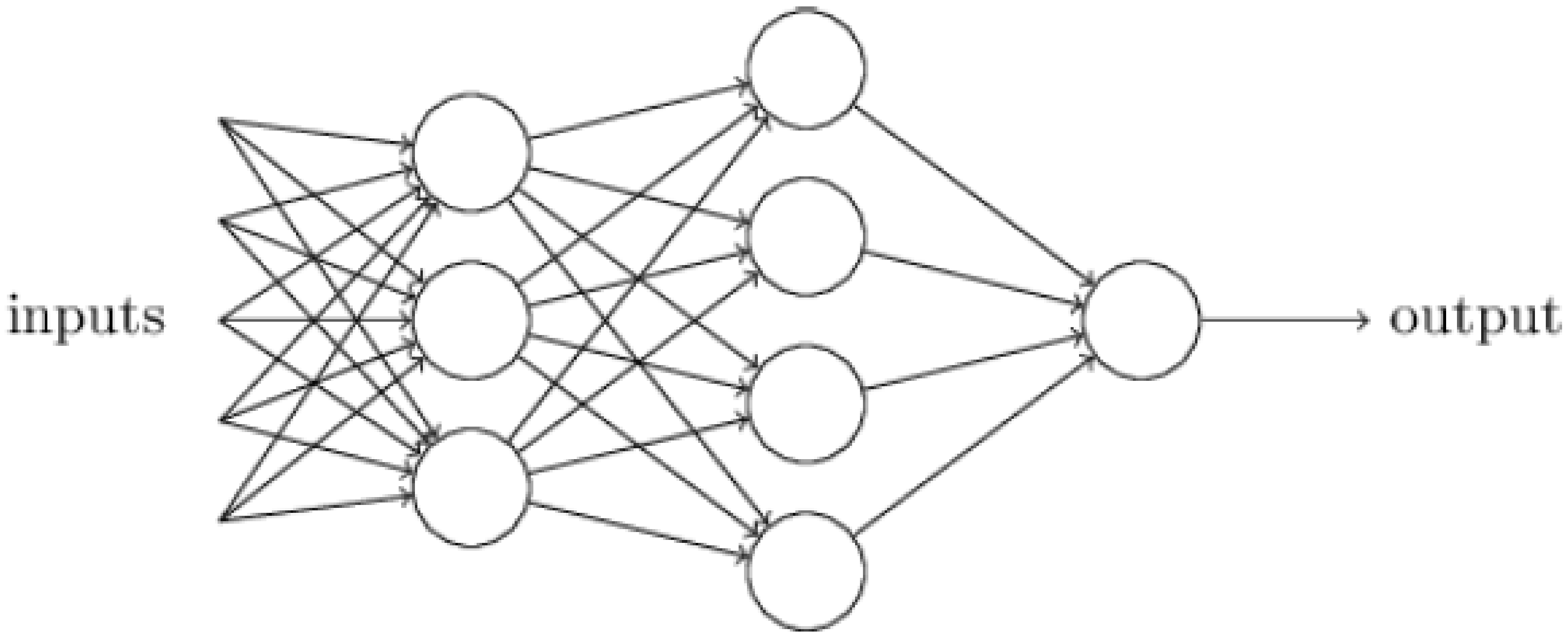
- To simplify:
  - Denote: $w \cdot x \equiv \sum_j w_j x_j$,
  - And bias b=threshold
- The perceptron rule can be rewritten:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



Neuron model: Perceptron

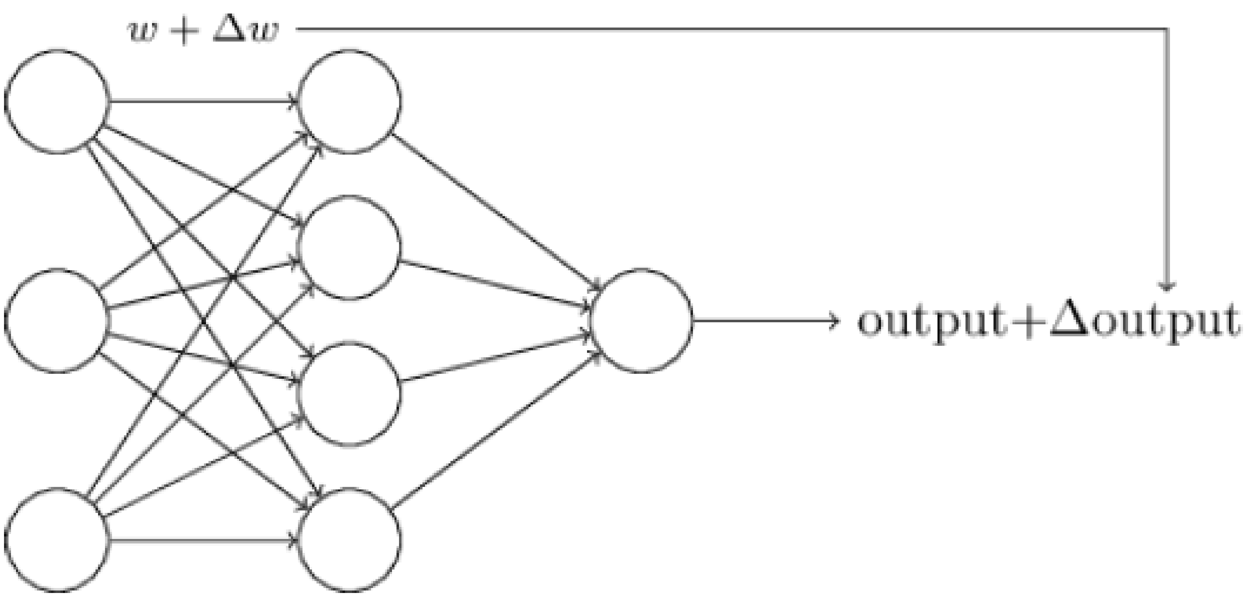
- A complex network of perceptrons for quite subtle decisions:



Neuron model: Sigmoid Neuron

- We need some learning algorithm that can learn weights and biases, e.g., to correctly classifies the digit

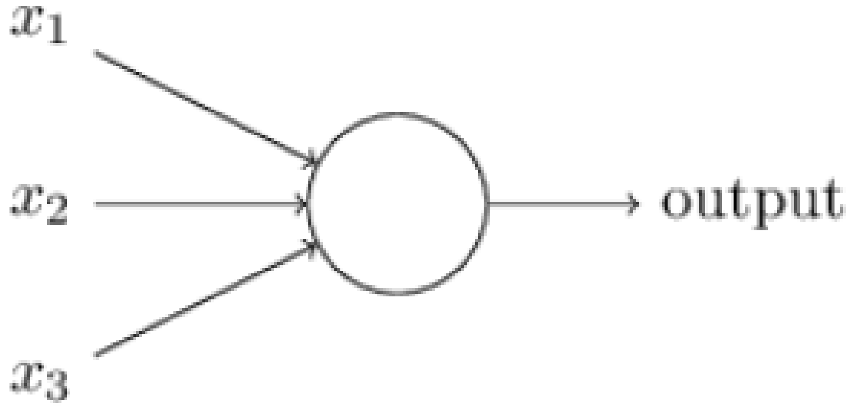
A small change in any weight (or bias) causes a small change in the output



However, it sometimes cause the output of that perceptron to completely flip, say from 1 to 0

Neuron model: Sigmoid Neuron

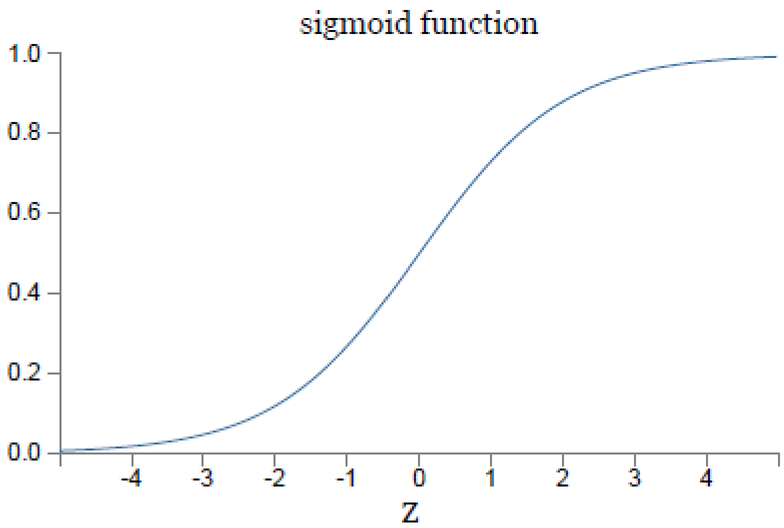
- Sigmoid neuron



- Input: x_1, x_2, … can take on any values between 0 and 1 (not just 0 or 1)
- Weight: weights for each input w_1, w_2, …, and bias b
- Output: between 0 and 1(not just 0 or 1)

Neuron model: Sigmoid Neuron

- The output of a sigmoid neuron with inputs x_1, x_2, …, weights w_1, w_2, …, and bias b:



$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$ is a sigmoid function

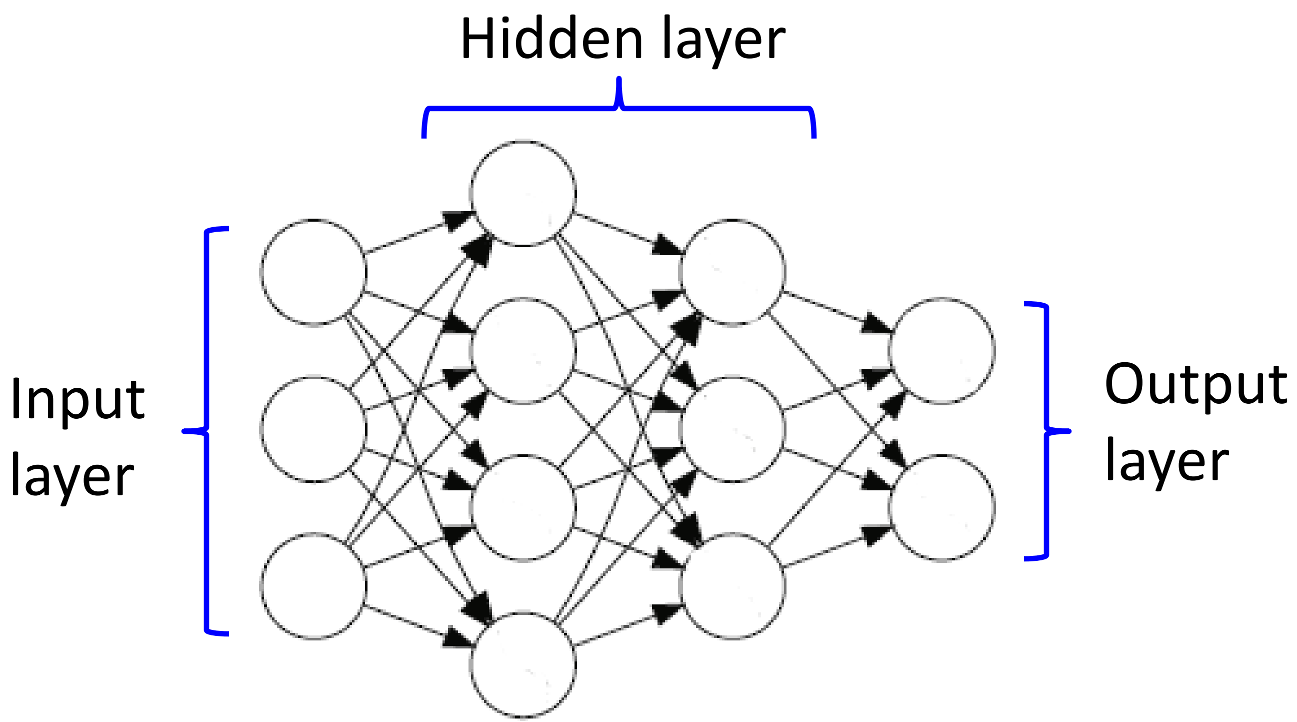
Neuron model: Sigmoid Neuron

- Sigmoid neuron closely approximates a smoothed out perceptron
- $\triangle$output is a *linear function* of the changes $\triangle$w and $\triangle$b in the weights and bias.

$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b$$

Neural networks: architecture

Feedforward neural networks: the output from one layer is used as input to the next layer (no loops)



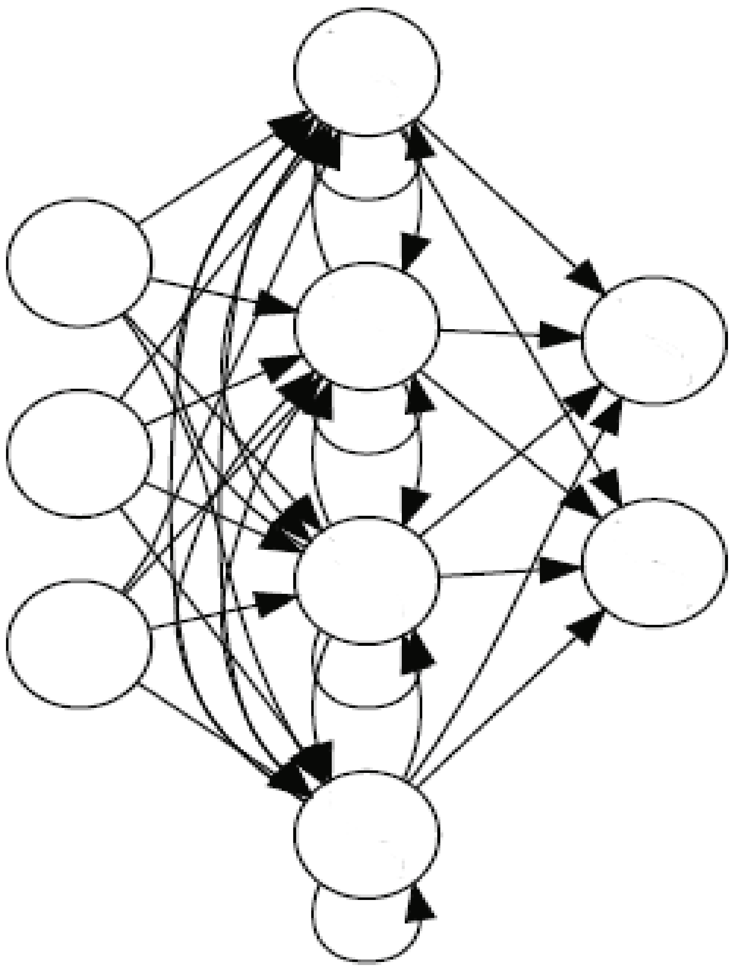
Neural networks: architecture

- Well known examples of FNNs include:
  - *Perceptrons* (Rosenblatt, 1958)
  - *Radial basis function networks (Broomhead and Lowe,* 1988)
  - *Kohonen maps (Self-Organizing Map)(Kohonen, 1989)*
  - *Hopfield nets (Hopfield, 1982)*
  - The most widely used form of FNN is the *multilayer perceptron (MLP; Rumelhart et al., 1986; Werbos, 1988;* Bishop, 1995).

* Sometimes called *multilayer perceptrons or MLPs,* despite being made up of sigmoid neurons, not perceptrons.

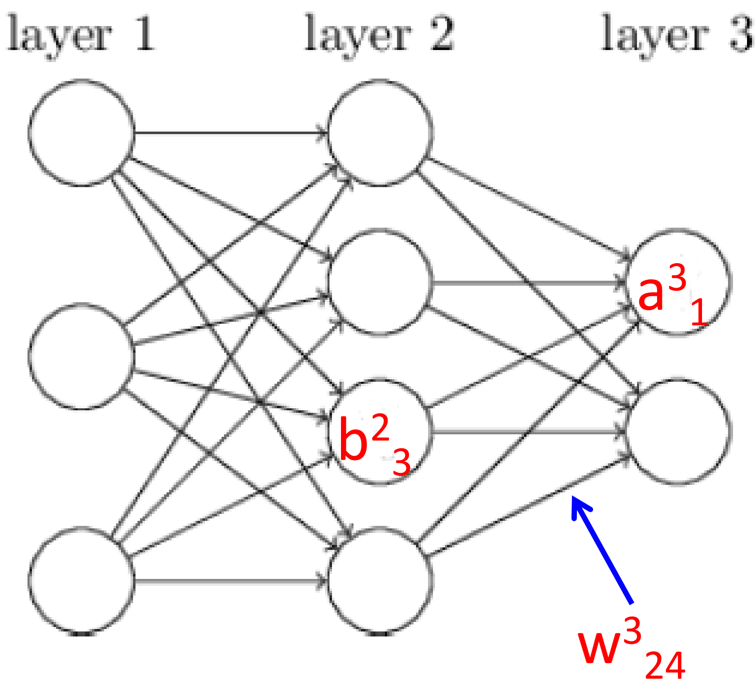
Neural networks: architecture

Recurrent neural networks: allow cyclical connections between layers (e.g., a single, self connected hidden layer)



- Many varieties of RNN:
  - Elman networks (Elman, 1990)
  - Jordan networks (Jordan, 1990)
  - time delay neural networks (Lang et al., 1990)
  - Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997)
  - echo state networks (Jaeger, 2001)
  - Gated Recurrent Units (Chung et al., 2014)
  - ……

Neural networks: representation



- w^l_{jk}: the weight for the connection from the *kth* neuron in the *(l−1)th* layer to the *jth* neuron in the *lth* layer
- b^l_j: the bias of the *jth* neuron in the *lth* layer
- a^l_j: the activation of the *jth* neuron in the *lth* layer

Neural networks: representation

- a^l_j: the activation of the *jth* neuron in the *lth* layer

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

  – the sum is over all neurons *k* in the *(l–1)th* layer

Neural networks: representation

- w^l: a weight matrix for each layer, l
  - the entry in the jth row and kth column is w^l_{jk}
- b^l: a bias vector for each layer, l
  - the entry in the jth row is b^l_j
- a^l: an activation vector
  - with components the activations a^l_j
- σ: function vectorization
- Then,

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

  - how the activations in one layer relate to activations in the previous layer

Neural networks: representation

- z^l: the weighted input to the neurons in layer l

$$z^l \equiv w^l a^{l-1} + b^l$$

  - with components the weighted input to the activation function for neuron j in layer l

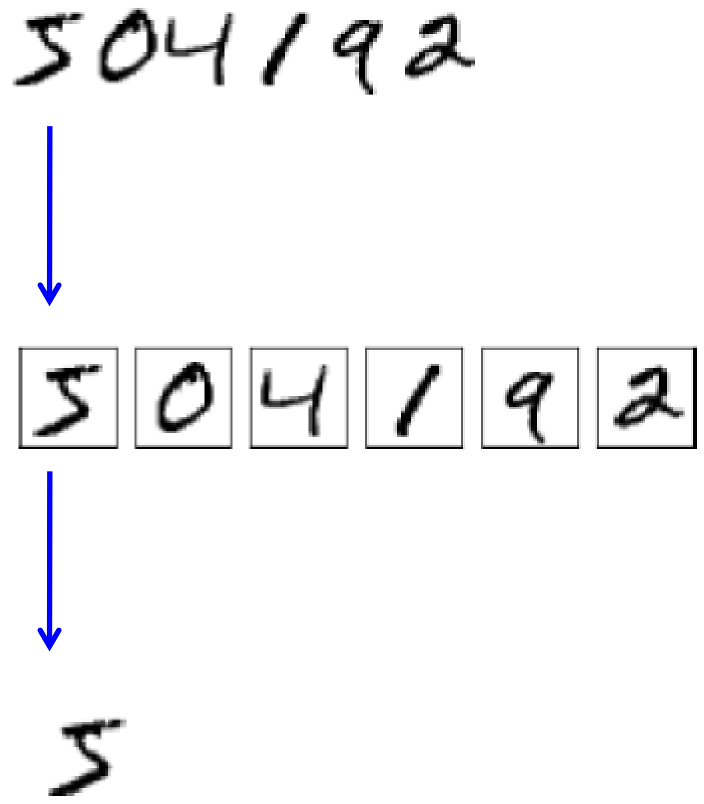
$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

- Finally,

$$a^l = \sigma(z^l).$$

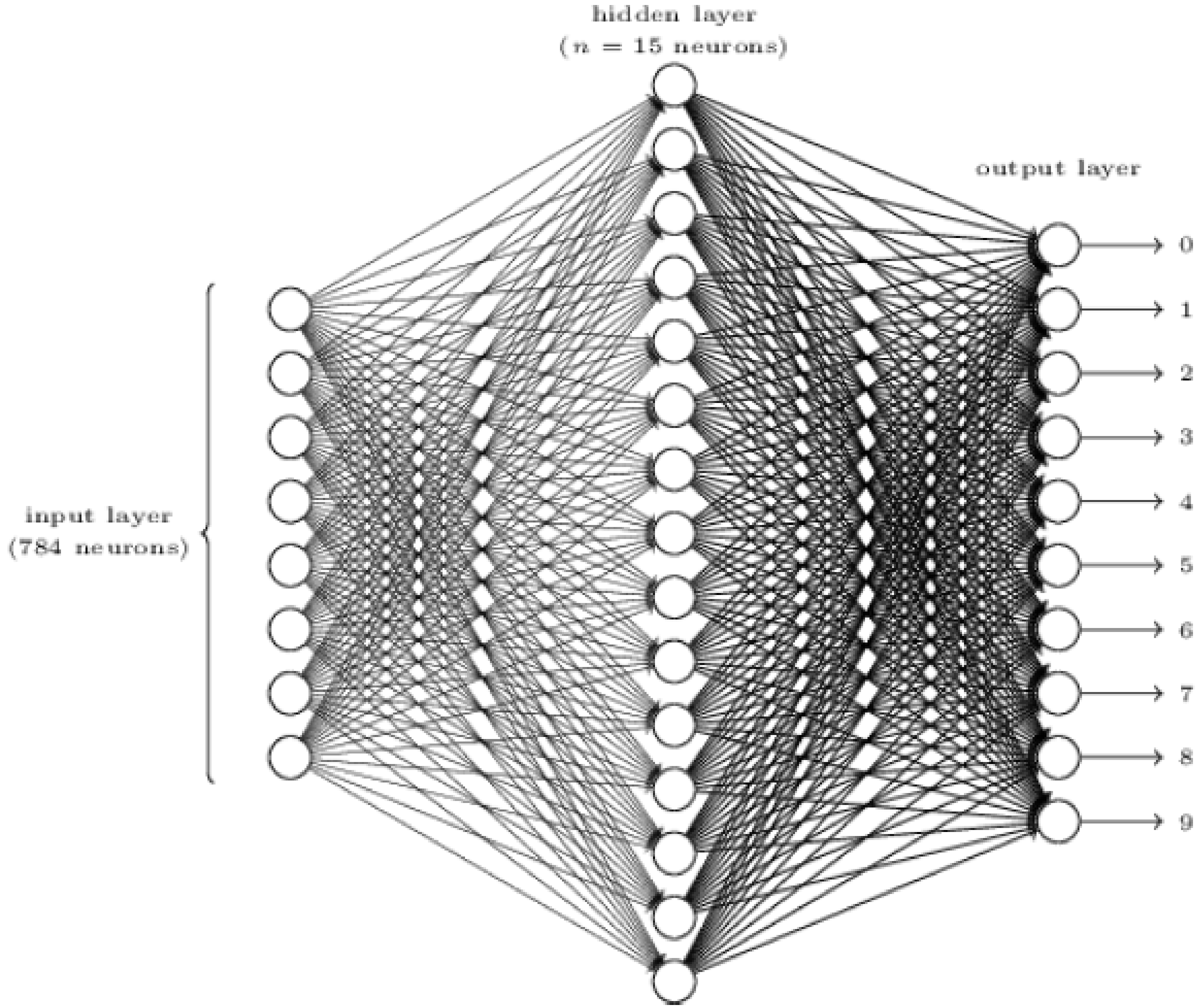
Neural networks: Example

- A simple network to classify handwritten digits
  - Segment the image
  - Classify individual digits



Neural networks: Example

- One possible network architecture:



Neural networks: Example

- Input layer: 784 neurons
  - 28*28 pixel
  - greyscale, with a value of 1 representing white, a value of 0 representing black, and in between values representing gradually darkening shades of grey.
- Hidden layer: 15 neurons
  - Quite an art to the design of the hidden layers
  - Experiment with different values
- Output layer: 10 neurons
  - If the first neuron fires, i.e., has an output≈1, then that will indicate that the network thinks the digit is a 0. The second fires for digit 1. And so on.

Neural networks: Learning

- Training data: e.g., the MNIST data set, with n training data
  - x: denotes a training data, a 28*28=784 dimensional vector
  - y: desired output, y=y(x), a 10 dimensional vector
  - E.g., if a particular training image, x, depicts a 6, then $y=(0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$
- Network:
  - Parameters: *w*, all the weights, *b*, all the bias
  - Output: a=a(x), the vector of outputs from the network when x is input
- Goal: find weights *w* and biases *b* so that the output *a*(*x*) from the network approximates y(x) for all training inputs x

Neural networks: Learning

- Cost function: e.g., the *mean squared error* (or just *MSE*)

$$C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2$$

  - L: the number of layers in the network
- A gradient method!
  - to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network
  - Ultimately, this means computing the partial derivatives $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$

Neural networks: Learning

- C_x: Cost functions for individual training examples, *x*

$$C_x = \frac{1}{2} \| y - a^L \|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2$$

- *Then,*

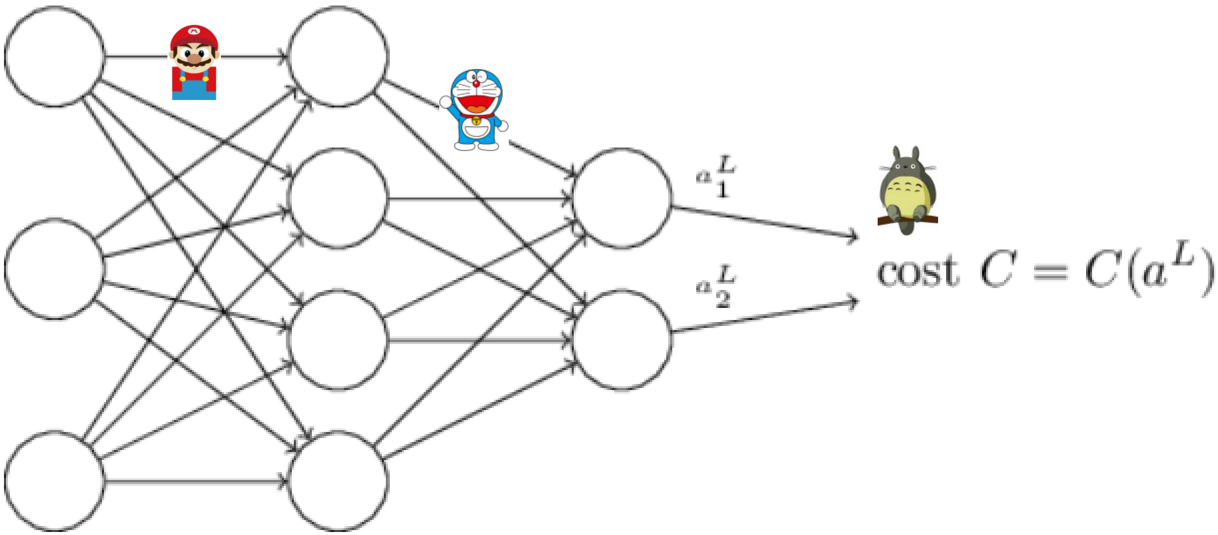
$$C = \frac{1}{n} \sum_x C_x$$

  - C is a function of the output activations:

$$C = C(a^L)$$

Ultimately, this means computing the partial derivatives **$\partial C / \partial w^l_{jk}$ and $\partial C / \partial b^l_j$**

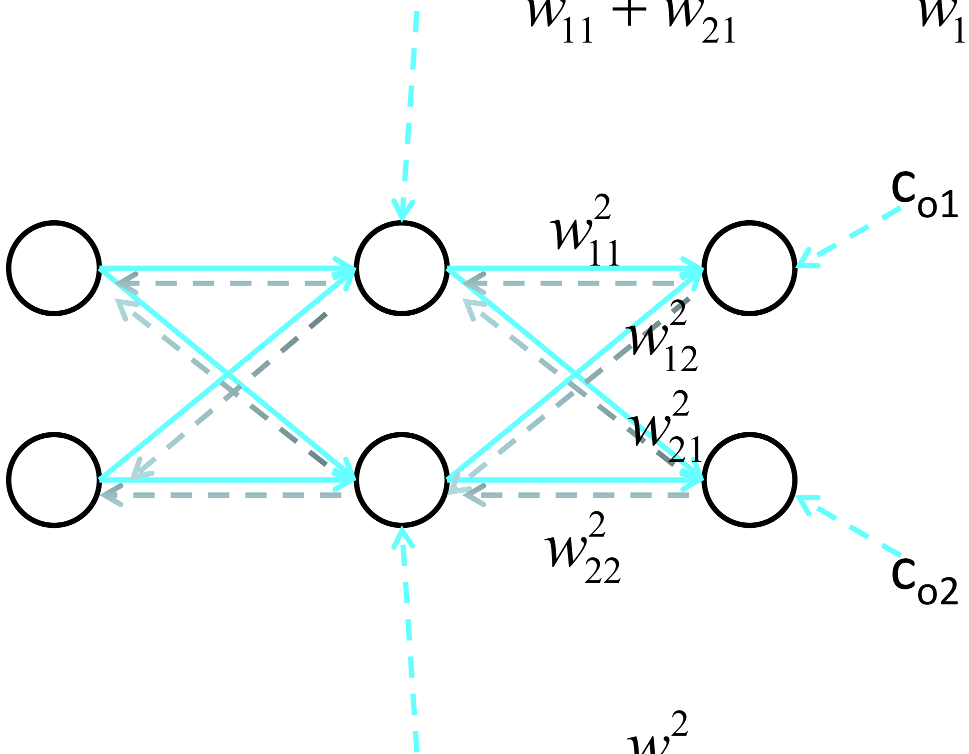
Neural networks: Learning



- We are trying to make the cost smaller.
- A little change Δz^l_j to the neuron's weighted input will cause the outputting $\sigma(z^l_j)$ change to $\sigma(z^l_j+\Delta z^l_j)$
- This change propagates through later layers in the network, finally causing the overall cost to change by an amount $\partial C/\partial z^l_j$

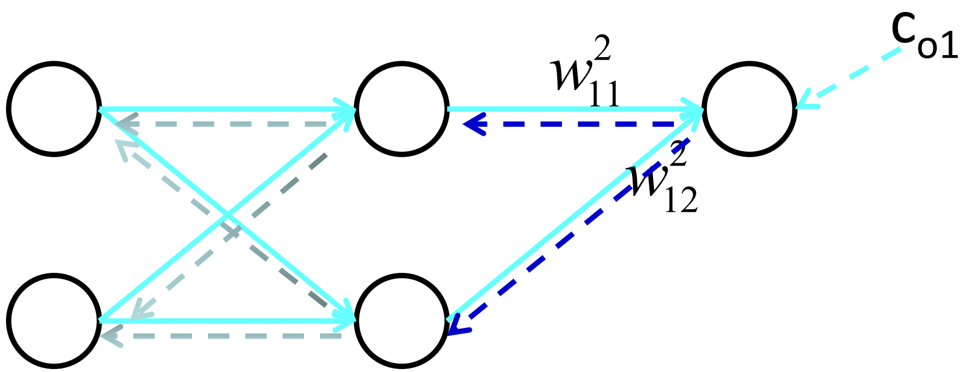
Neural networks: Learning

$$c_{h1} = \frac{w_{11}^2}{w_{11}^2 + w_{21}^2} \cdot c_{o1} + \frac{w_{12}^2}{w_{12}^2 + w_{22}^2} \cdot c_{o2}$$



$$c_{h2} = \frac{w_{21}^2}{w_{11}^2 + w_{21}^2} \cdot c_{o1} + \frac{w_{22}^2}{w_{12}^2 + w_{22}^2} \cdot c_{o2}$$

Neural networks: Learning



$$c_{o1} = \frac{1}{2}(a_1^3 - y_1)^2$$

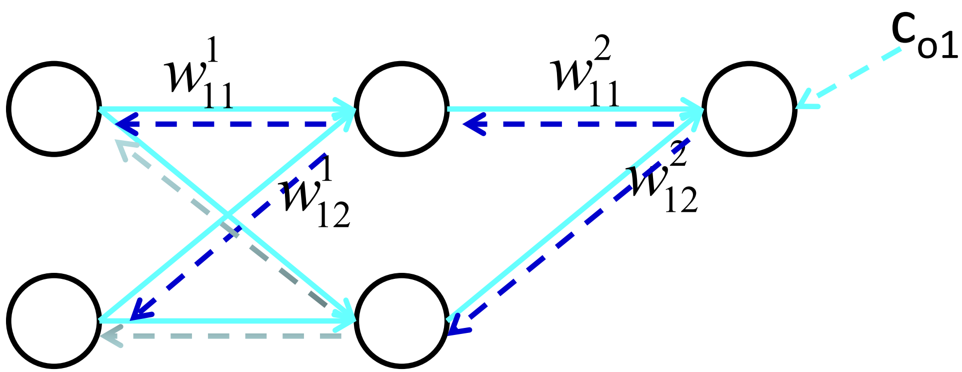
$$a_1^3 = \sigma(z_3^1)$$

$$z_3^1 = w_{11}^2 \cdot a_1^2 + w_{12}^2 \cdot a_2^2 + b_1^3$$

$$\frac{\partial c_{o1}}{\partial w_{11}^2} = \frac{\partial c_{o1}}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial w_{11}^2}$$

$$\frac{\partial c_{o1}}{\partial w_{12}^2} = \frac{\partial c_{o1}}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial w_{12}^2}$$

Neural networks: Learning



$$c_{o1} = \frac{1}{2}(a_1^3 - y_1)^2$$

$$a_1^3 = \sigma(z_3^1)$$

$$z_3^1 = w_{11}^2 \cdot a_1^2 + w_{12}^2 \cdot a_2^2 + b_1^3$$

$$a_1^2 = \sigma(z_1^2)$$

$$z_1^2 = w_{11}^1 \cdot a_1^1 + w_{12}^1 \cdot a_2^1 + b_1^1$$

$$\frac{\partial c_{o1}}{\partial w_{11}^1} = \frac{\partial c_{o1}}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial z_1^2} \cdot \frac{\partial z_1^2}{\partial w_{11}^1}$$

weight update:

$$w_{11}^1 = w_{11}^1 - \eta \frac{\partial c_{o1}}{\partial w_{11}^1}$$

Neural networks: Learning

- *Define* δ^l_j error of neuron *j in layer l*:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$$

- δ^l: *the vector of* errors associated with layer *l*
- δ^L: the error in the output layer

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

BP1

- Proof:

$$\delta^L_j = \frac{\partial C}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j} = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

Neural networks: Learning

- In a matrix-based form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

  - $\odot$: elementwise multiplication (or Hadamard product)
  - As an example,

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Neural networks: Learning

- In the case of the quadratic cost,

$$C = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

- So

$$\partial C / \partial a_j^L = (a_j^L - y_j)$$

- And

$$\nabla_a C = (a^L - y)$$

- So the fully matrix-based form of δ is

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Neural networks: Learning

- *Error backward*:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

BP2

– Proof:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \Rightarrow \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

$$\Rightarrow \delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Neural networks: Learning

- The rate of change the cost with respect to any bias is: (exactly equal to the error δ^l_j)

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$\boxed{\text{BP3}}$

  - δ^l_j is being evaluated at the same neuron as the bias b

- Proof:

$$\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial b^l_j} = \delta^l_j \cdot \frac{\partial (w^l_{jk} a^{l-1}_k + b^l_j)}{\partial b^l_j} = \delta^l_j$$

Neural networks: Learning

- The rate of change of the cost with respect to any weight:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$\boxed{\text{BP4}}$

- Proof:

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial w^l_{jk}} = \delta^l_j \cdot \frac{\partial (w^l_{jk} a^{l-1}_k + b^l_j)}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

→weights output from low activation neurons learn slowly

Neural networks: Learning

- Summary: the four fundamental equations

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

The backpropagation algorithm

- The backpropagation equations provide us with a way of computing the gradient of the cost function

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute

$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l)$$

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

The backpropagation algorithm

- Stochastic gradient descent with a mini batch:

1. **Input a set of training examples**

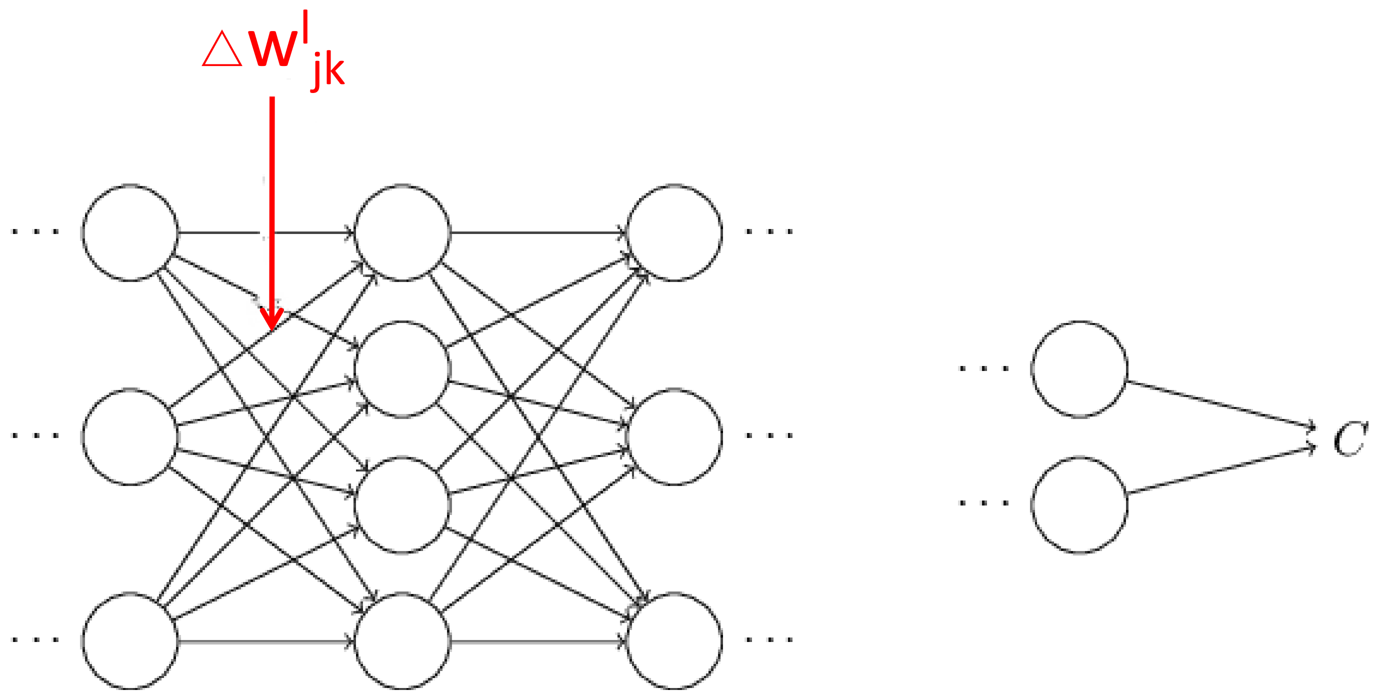
2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:

   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

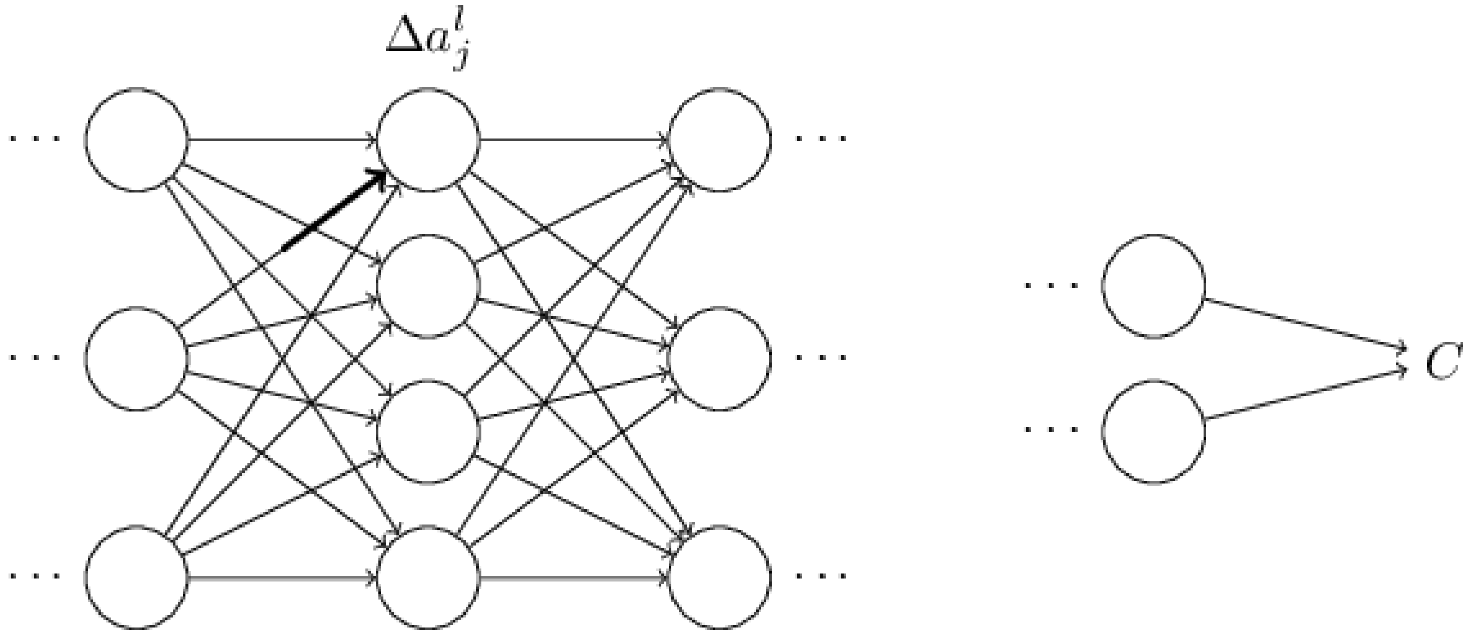
Backpropagation: the big picture

- Suppose we make a small change $\triangle w^l_{jk}$ to some weight in the network, w^l_{jk}



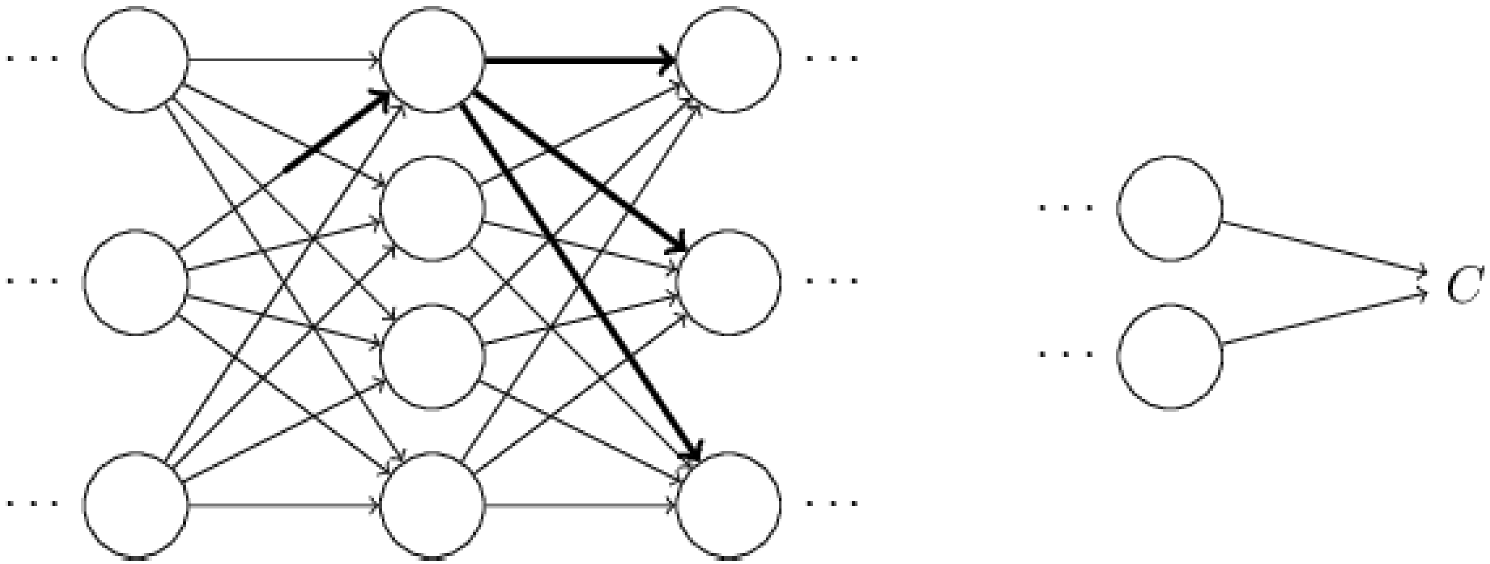
Backpropagation: the big picture

- That change in weight will cause a change in the output activation from the corresponding neuron:



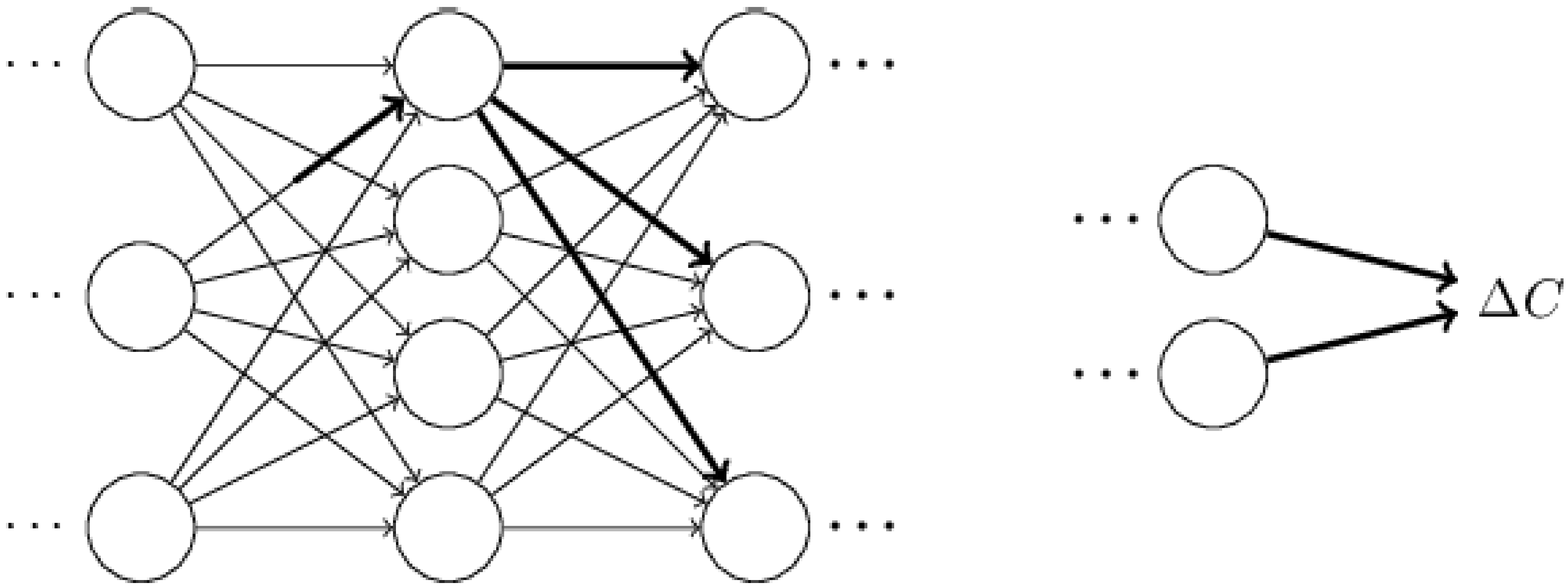
Backpropagation: the big picture

- That, in turn, will cause a change in *all* the activations in the next layer:



Backpropagation: the big picture

- And finally will cause a change in the final layer, and then in the cost function:



Backpropagation: the big picture

- The change in the cost $\triangle C$ is related to the change in the weight $\triangle w^l_{jk}$ by the equation:

$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk}$$

- BUT how such change propagates to the cost C?

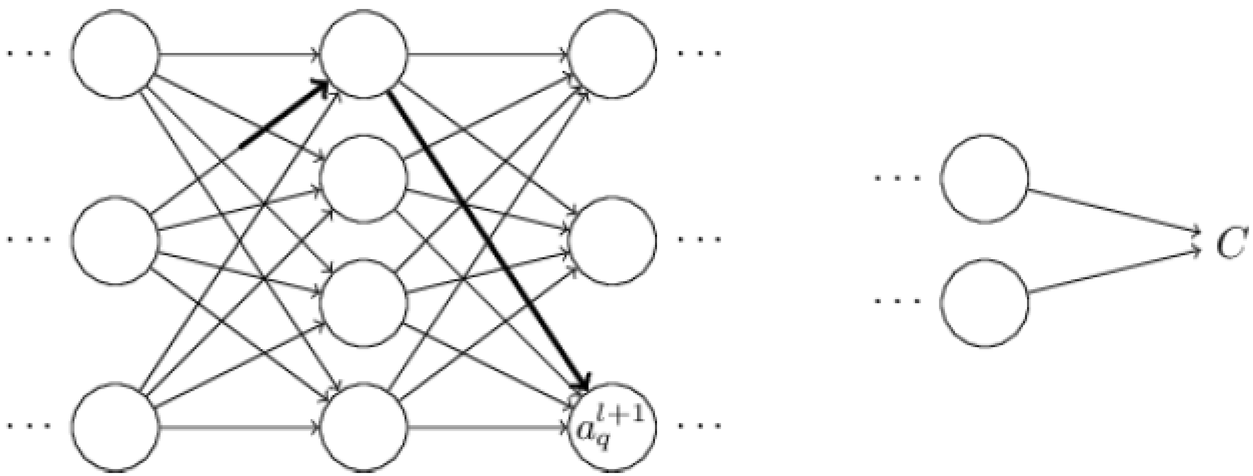
Backpropagation: the big picture

- The change $\triangle w^l_{jk}$ causes a small change $\triangle a^l_j$ in the activation of the j^{th} neuron in the i^{th} layer

$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

Backpropagation: the big picture

- The change $\triangle a^l_j$ in activation will cause changes in *all the* activations in the next layer, *l*+1 layer
- Consider a single one of the activation a^{l+1}_q



$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

Backpropagation: the big picture

- The change $\triangle a^{l+1}_q$ will, in turn, cause changes in the activations in the next layer.
- If the path goes through activations a^l_j, a^{l+1}_q, ..., a^{L-1}_n, a^L_m, then the change of cost caused in this particular path:

$$\Delta C \approx \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}$$

Backpropagation: the big picture

- To compute the total change in C it is plausible that we should sum

$$\Delta C \approx \sum_{mnp\ldots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

- Comparing

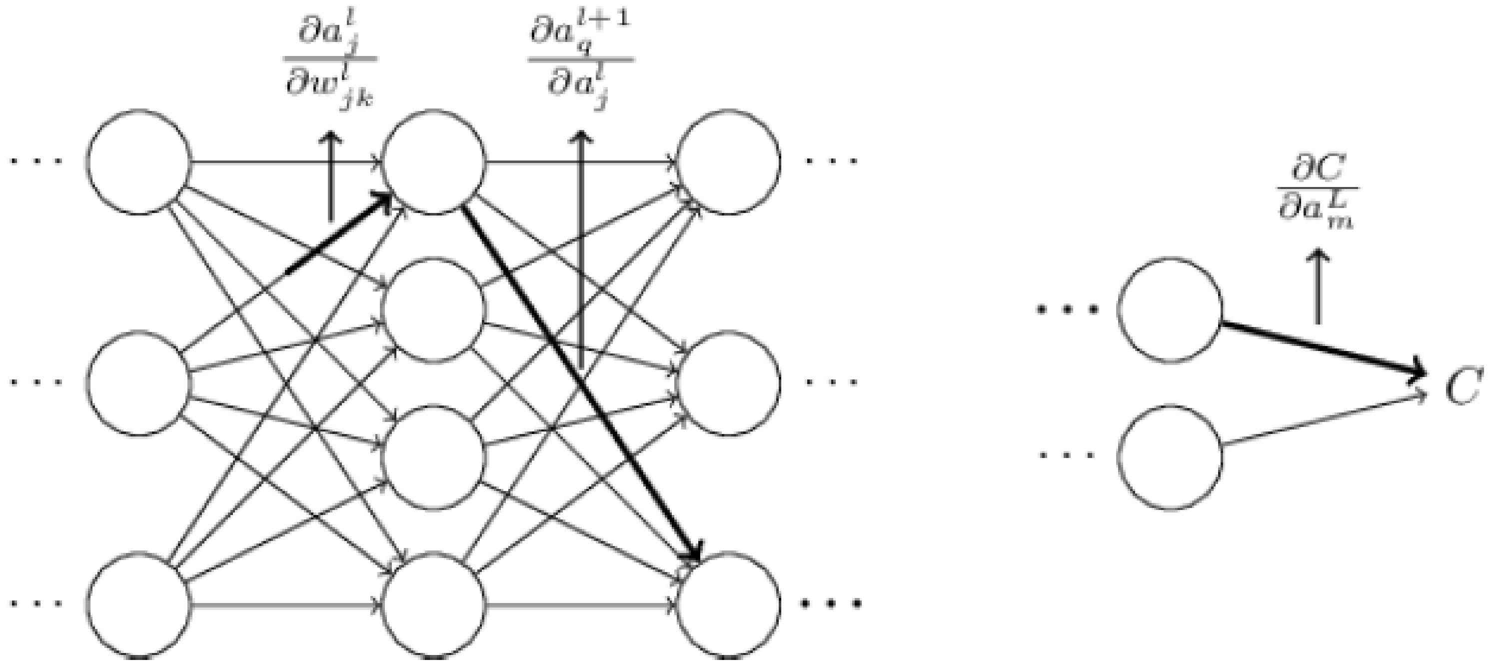
What will happen?

- We get

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\ldots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$

Backpropagation: the big picture

- The rate of change of C with respect to a weight w^l_{jk} in the network



Backpropagation: the big picture

- Every edge between two neurons in the network is associated with a rate factor which is just the partial derivative of one neuron’s activation with respect to the other neuron's activation.
- The backpropagation algorithm provides a way of computing the sum over the rate factor for all these paths

思考题

- 1. How perceptrons can be used is to compute the elementary logical functions such as AND, OR, and NAND?
- 2. In what sense is backpropagation a fast algorithm?
- 3. Backpropagation with linear neurons: Suppose we replace the usual nonlinear function σ with $\sigma(z)=z$ throughout the network. Rewrite the backpropagation algorithm for this case.

- Next lecture:
  - Deep neural networks and advanced ways neural networks learn