Lecture 16: Neural networks for machine vision

Dr. Richard E. Turner (ret26@cam.ac.uk)

December 3, 2013
Scientists See Promise in Deep-Learning Programs

Using an artificial intelligence technique inspired by theories about how the brain recognizes patterns, technology companies are reporting startling gains in fields as diverse as computer vision, speech recognition and the identification of promising new molecules for designing drugs.
A single neuron

\[
x(a) = \frac{1}{1 + \exp(-a)} \quad x \in (0, 1)
\]

\[
a = w_0 + \sum_{d=1}^{D} w_d z_d
\]
A single neuron

\[ x(a) = \frac{1}{1 + \exp(-a)} \quad x \in (0, 1) \]

\[ a = \sum_{d=0}^{D} w_d z_d \]
Input-output function of a single neuron

\[ w = [0,1] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0.2, 1] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0.3, 0.9] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1z_1 - w_2z_2)} \]
Input-output function of a single neuron

\[ w = [0.5, 0.9] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0.6, 0.8] \]

\[
x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}
\]
Input-output function of a single neuron

\[ w = [0.8, 0.6] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1z_1 - w_2z_2)} \]
Input-output function of a single neuron

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]

\[ w = [0.9, 0.5] \]
Input-output function of a single neuron

\[ w = [0.9, 0.3] \]

\[
x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}
\]
Input-output function of a single neuron

\[ w = [1, 0.2] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [1, 0] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0,1] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0,2] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0, 3] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0, 4] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

\[ w = [0, 5] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]
Input-output function of a single neuron

$w = [0, 1]$

$x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)}$

contours $w_1 z_1 + w_2 z_2 = c = w^T z$
Input-output function of a single neuron

\[ w = [0, 1] \]

\[ x(z_1, z_2) = \frac{1}{1 + \exp(-w_1 z_1 - w_2 z_2)} \]

contours \( w_1 z_1 + w_2 z_2 = c = w^T z \)

\( \frac{w}{|w|} \) sets direction of boundary

\( |w| \) sets steepness of boundary
Weight space of a single neuron

\[ W = [2, 2] \]

The diagram illustrates the weight space of a single neuron with weight vectors \( W_1 \) and \( W_2 \). The output \( z \) is a linear combination of the inputs \( x \) weighted by the weight vectors. The figure shows the effect of different weight vectors on the weight space.
Training a single neuron
Training a single neuron

desired result of training:
neuron outputs $x(z^{(n)}; w) \approx 1$ for $t^{(n)} = 1$
neuron outputs $x(z^{(n)}; w) \approx 0$ for $t^{(n)} = 0$

training data
$\{z^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$
inputs class labels
class $t^{(n)} = 0$
class $t^{(n)} = 1$
Training a single neuron

desired result of training:
neuron outputs $x(z^{(n)}; \mathbf{w}) \approx 1$ for $t^{(n)} = 1$

neuron outputs $x(z^{(n)}; \mathbf{w}) \approx 0$ for $t^{(n)} = 0$

training data
\[
\{z^{(n)}\}_{n=1}^{N} \quad \{t^{(n)}\}_{n=1}^{N}
\]
inputs           class labels

given by class $t^{(n)} = 0$
class $t^{(n)} = 1$

objective function:
\[
G(\mathbf{w}) = - \sum_{n} [t^{(n)} \log x(z^{(n)}; \mathbf{w}) + (1 - t^{n}) \log (1 - x(z^{(n)}; \mathbf{w}))] \geq 0
\]
Training a single neuron

desired result of training:

neuron outputs \( x(z^{(n)}; w) \approx 1 \) for \( t^{(n)} = 1 \)

neuron outputs \( x(z^{(n)}; w) \approx 0 \) for \( t^{(n)} = 0 \)

objective function:

\[
G(w) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; w) + (1 - t^{(n)}) \log \left(1 - x(z^{(n)}; w)\right)\right] \geq 0
\]

surprise \(- \log p(\text{outcome})\) when observing \( t^{(n)} \)

relative entropy between \( x(z^{(n)}; w) \) and \( t^{(n)} \)

encourages neuron output to match training data
Training a single neuron

training data
\[ \{z^{(n)}\}_{n=1}^N \quad \{t^{(n)}\}_{n=1}^N \]
inputs class labels

objective function:

\[
G(w) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; w) + (1 - t^n) \log (1 - x(z^{(n)}; w)) \right] \geq 0
\]
Training a single neuron

objective function:

\[ G(w) = -\sum_n [t^{(n)} \log x(z^{(n)}; w) + (1 - t^n) \log (1 - x(z^{(n)}; w))] \geq 0 \]

\[ w^* = \arg\min_w G(w) \]

choose the weights that minimise the network's surprise about the training data

\[ \{z^{(n)}\}_{n=1}^N \quad \{t^{(n)}\}_{n=1}^N \]

inputs \quad class labels
Training a single neuron

objective function:

\[ G(w) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; w) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; w)) \right] \geq 0 \]

\[ w^* = \arg \min_w G(w) \quad \text{choose the weights that minimise the network's surprise about the training data} \]

\[ \frac{d}{dw} G(w) = \sum_n \frac{dG(w)}{dx^{(n)}} \frac{dx^{(n)}}{dw} = - \sum_n (t^{(n)} - x^{(n)}) z^{(n)} \]

training data

\( \{ z^{(n)} \}_{n=1}^N \{ t^{(n)} \}_{n=1}^N \)

inputs class labels

\( x \)

\( z_1 \)

\( z_2 \)
Training a single neuron

**objective function:**

\[
G(w) = -\sum_n [t^{(n)} \log x(z^{(n)}; w) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; w))] \geq 0
\]

\[
w^* = \arg \min_w G(w)
\]

choose the weights that minimise the network's surprise about the training data

\[
\frac{d}{dw} G(w) = \sum_n \frac{dG(w)}{dx^{(n)}} \frac{dx^{(n)}}{dw} = -\sum_n (t^{(n)} - x^{(n)}) z^{(n)} = \text{prediction error} \times \text{feature}
\]

\[
w \leftarrow w - \eta \frac{d}{dw} G(w)
\]

iteratively step down the objective (gradient points up hill)
Training a single neuron

\[ w = [0, -1] \]
Training a single neuron

\[ w = [0.4, -0.7] \]
Training a single neuron

\[ w = [0.9, -0.2] \]

The diagram illustrates the training process of a single neuron, showing the input data points, the neuron's activation function, and the objective function over iterations.
Training a single neuron

\[ w = [1.1, 0.1] \]
Training a single neuron

\[ w = [1.4, 0.4] \]
Training a single neuron

$w = [1.6, 0.7]$
Training a single neuron

\[ w = [1.8, 1.1] \]
Training a single neuron

\[ w = [2, 1.6] \]
Training a single neuron

\[ w = [2.2, 2.3] \]
Training a single neuron

\[ w = [2.4, 3.1] \]
Training a single neuron

\[ w = [2.6, 4] \]
Training a single neuron

$w = [2.8, 5]$
Training a single neuron

\[ w = [3, 5.9] \]
Training a single neuron

\[ w = [3.3, 6.9] \]
Training a single neuron

\( w = [3.6, 7.8] \)
Training a single neuron

\( w = [3.9, 8.8] \)
Training a single neuron

\[ w = [4.2, 9.8] \]
Training a single neuron

\[ w = [4.6, 10.7] \]
Training a single neuron

\[ w = [4.9, 11.7] \]
Training a single neuron

\[ w = [5.2, 12.6] \]
Training a single neuron

\[ w = [9.7, 25.3] \]
Over-fitting and weight decay

objective function:

\[ G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; \mathbf{w})) \right] \]
Over-fitting and weight decay

objective function:

\[
G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log \left(1 - x(z^{(n)}; \mathbf{w})\right) \right]
\]

\[
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \quad \text{regulariser discourages the network using extreme weights}
\]
Over-fitting and weight decay

Training data
\[
\{ z^{(n)} \}_{n=1}^N \quad \{ t^{(n)} \}_{n=1}^N
\]
inputs class labels

Objective function:
\[
G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log \pi(z^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log (1 - \pi(z^{(n)}; \mathbf{w})) \right]
\]
\[
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \quad \text{regulariser discourages the network using extreme weights}
\]
\[
\mathbf{w}^* = \arg\min_{\mathbf{w}} M(\mathbf{w}) = \arg\min_{\mathbf{w}} [G(\mathbf{w}) + \alpha E(\mathbf{w})]
\]
Over-fitting and weight decay

objective function:

\[ G(w) = - \sum_n \left[ t^{(n)} \log x(z^{(n)}; w) + (1 - t^n) \log (1 - x(z^{(n)}; w)) \right] \]

\[ E(w) = \frac{1}{2} \sum_i w_i^2 \quad \text{regulariser discourages the network using extreme weights} \]

\[ w^* = \arg \min_w M(w) = \arg \min_w [G(w) + \alpha E(w)] \]

\[ \frac{d}{dw} M(w) = - \sum_n (t^{(n)} - x^{(n)})z^{(n)} + \alpha w \quad \text{weight decay - shrinks weights towards zero} \]
Training a single neuron

\[ w_{\text{reg}} = [0, -1] \]

\[ w = [0, -1] \]

- Diagram showing the changes in the objective and iteration for both original and regularised training.
- The diagram compares the performance of the neuron with and without regularisation.
Training a single neuron

\( w_{\text{reg}} = [0.4, -0.7] \)

\( w = [0.4, -0.7] \)

objective

iteration

original

regularised
Training a single neuron

\[ w_{\text{reg}} = [0.6, -0.4] \]

\[ w = [0.6, -0.4] \]
Training a single neuron

\[ w_{\text{reg}} = [0.8, -0.2] \]

\[ w = [0.8, -0.3] \]
Training a single neuron

\[ w_{\text{reg}} = [0.9, -0.1] \]

\[ w = [0.9, -0.1] \]
Training a single neuron

\[ w_{\text{reg}} = [0.9, 0] \]

\[ w = [1, 0] \]
Training a single neuron

$w_{\text{reg}} = [1, 0.1]$  

$w = [1.1, 0]$
Training a single neuron

$w_{\text{reg}} = [1,0.1]$

$w = [1.1,0.1]$

---

objective

iteration
Training a single neuron

$\mathbf{w}_{\text{reg}} = [1.1, 0.2]$

$\mathbf{w} = [1.2, 0.2]$

- Objective
- Iteration
Training a single neuron

\[ w_{\text{reg}} = [1.1, 0.3] \]

\[ w = [1.2, 0.2] \]
Training a single neuron

\[ w_{\text{reg}} = [1.2, 0.5] \]

\[ w = [1.4, 0.5] \]
Training a single neuron

$w_{\text{reg}} = [1.1, 0.8]$  

$w = [1.6, 0.9]$
Training a single neuron

$w_{\text{reg}} = [1, 1.1]$  

$w = [1.9, 1.7]$
Training a single neuron

\[ w_{\text{reg}} = [1, 1.1] \]

\[ w = [2.2, 2.7] \]
Training a single neuron

$w_{\text{reg}} = [1, 1.1]$  

$w = [2.5, 4]$
Probabilistic interpretation of the single neuron

single neuron training:

\[ \{z^{(n)}\}_{n=1}^N, \quad D = \{t^{(n)}\}_{n=1}^N \]

\[ G(w) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^n) \log (1 - x^{(n)}) \right] \]

\[ M(w) = G(w) + \alpha E(w) \quad w^* = \arg \min_w M(w) \]

\[ E(w) = \frac{1}{2} \sum_i w_i^2 \]
Probabilistic interpretation of the single neuron

\[ p(t = 1 | \mathbf{w}, z) = x \quad p(t = 0 | \mathbf{w}, z) = 1 - x \]

**single neuron training:**

\[
\{z^{(n)}\}_{n=1}^{N} \quad D = \{t^{(n)}\}_{n=1}^{N} \\
G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \\
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \\
M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \\
w^* = \arg\min_{\mathbf{w}} M(\mathbf{w})
\]
Probabilistic interpretation of the single neuron

\[ p(t = 1|\mathbf{w}, z) = x \quad p(t = 0|\mathbf{w}, z) = 1 - x \]

\[ p(t|\mathbf{w}, z) = x^t(1 - x)^{1-t} \]

**single neuron training:**

\[
\{z^{(n)}\}_{n=1}^{N} \quad D = \{t^{(n)}\}_{n=1}^{N}
\]

\[
G(\mathbf{w}) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right]
\]

\[
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2
\]

\[
M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \quad \mathbf{w}^* = \arg \min_{\mathbf{w}} M(\mathbf{w})
\]
Probabilistic interpretation of the single neuron training:

\[ p(t = 1|\mathbf{w}, \mathbf{z}) = x \quad p(t = 0|\mathbf{w}, \mathbf{z}) = 1 - x \]

\[ p(t|\mathbf{w}, \mathbf{z}) = x^t(1 - x)^{1-t} \]

\[ = \exp(t \log x + (1 - t) \log(1 - x)) \]

**single neuron training:**

\[
\begin{align*}
\{z^{(n)}\}_{n=1}^N & \quad D = \{t^{(n)}\}_{n=1}^N \\
G(\mathbf{w}) &= -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \\
M(\mathbf{w}) &= G(\mathbf{w}) + \alpha E(\mathbf{w}) \\
\mathbf{w}^* &= \arg\min_{\mathbf{w}} M(\mathbf{w}) \\
E(\mathbf{w}) &= \frac{1}{2} \sum_i w_i^2
\end{align*}
\]
Probabilistic interpretation of the single neuron

\[
p(t = 1 | \mathbf{w}, \mathbf{z}) = x \quad p(t = 0 | \mathbf{w}, \mathbf{z}) = 1 - x
\]

\[
p(t | \mathbf{w}, \mathbf{z}) = x^t (1 - x)^{1 - t}
\]

\[
= \exp(t \log x + (1 - t) \log (1 - x))
\]

\[
p(D | \mathbf{w}, \mathbf{z}) = \exp(-G(\mathbf{w}))
\]

**single neuron training:**

\[
\{ \mathbf{z}^{(n)} \}_{n=1}^N \quad D = \{ t^{(n)} \}_{n=1}^N
\]

\[
G(\mathbf{w}) = - \sum_n [t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)})]
\]

\[
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2
\]

\[
M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \quad \mathbf{w}^* = \arg \min \limits_\mathbf{w} M(\mathbf{w})
\]
Probabilistic interpretation of the single neuron

\[
p(t = 1|\mathbf{w}, z) = x \quad p(t = 0|\mathbf{w}, z) = 1 - x
\]
\[
p(t|\mathbf{w}, z) = x^t (1 - x)^{1-t}
\]
\[
= \exp(t \log x + (1 - t) \log(1 - x))
\]
\[
p(D|\mathbf{w}, z) = \exp(-G(\mathbf{w}))
\]
\[
p(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E(\mathbf{w}))
\]

**single neuron training:**

\[
\left\{ z^{(n)} \right\}_{n=1}^{N} \quad D = \left\{ t^{(n)} \right\}_{n=1}^{N}
\]
\[
G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right]
\]
\[
E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2
\]
\[
M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \quad \mathbf{w}^* = \arg \min_{\mathbf{w}} M(\mathbf{w})
\]
Probabilistic interpretation of the single neuron training:

\[
p(t = 1 \mid w, z) = x \quad p(t = 0 \mid w, z) = 1 - x
\]
\[
p(t \mid w, z) = x^t (1 - x)^{1-t} = \exp(t \log x + (1 - t) \log(1 - x))
\]
\[
p(D \mid w, z) = \exp(-G(w))
\]
\[
p(w \mid \alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E(w))
\]
\[
p(w \mid D, \alpha) = \frac{1}{p(D \mid \alpha)} p(D \mid w) p(w \mid \alpha) \quad \text{Bayes' Rule}
\]

\[
\{z^{(n)}\}_{n=1}^N \quad D = \{t^{(n)}\}_{n=1}^N
\]
\[
G(w) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \quad E(w) = \frac{1}{2} \sum_i w_i^2
\]
\[
M(w) = G(w) + \alpha E(w) \quad w^* = \arg \min_w M(w)
\]
Probabilistic interpretation of the single neuron

\[ p(t = 1|\mathbf{w}, \mathbf{z}) = x \quad p(t = 0|\mathbf{w}, \mathbf{z}) = 1 - x \]
\[ p(t|\mathbf{w}, \mathbf{z}) = x^t(1 - x)^{1-t} \]
\[ = \exp(t \log x + (1 - t) \log(1 - x)) \]
\[ p(D|\mathbf{w}, \mathbf{z}) = \exp(-G(\mathbf{w})) \]
\[ p(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E(\mathbf{w})) \]
\[ p(\mathbf{w}|D, \alpha) = \frac{1}{p(D|\alpha)} p(D|\mathbf{w})p(\mathbf{w}|\alpha) = \frac{1}{Z_M} \exp(-G(\mathbf{w}) - \alpha E(\mathbf{w})) \]

**single neuron training:**

\[ \{\mathbf{z}^{(n)}\}_{n=1}^N \quad D = \{t^{(n)}\}_{n=1}^N \]
\[ G(\mathbf{w}) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \]
\[ E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \]
\[ M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \quad \mathbf{w}^* = \arg\min_{\mathbf{w}} M(\mathbf{w}) \]
Probabilistic interpretation of the single neuron

\[ p(t = 1|\mathbf{w}, z) = x \quad p(t = 0|\mathbf{w}, z) = 1 - x \]
\[ p(t|\mathbf{w}, z) = x^t (1 - x)^{1-t} \]
\[ = \exp(t \log x + (1 - t) \log(1 - x)) \]
\[ p(D|\mathbf{w}, z) = \exp(-G(\mathbf{w})) \]
\[ p(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E(\mathbf{w})) \]
\[ p(\mathbf{w}|D, \alpha) = \frac{1}{p(D|\alpha)} p(D|\mathbf{w}) p(\mathbf{w}|\alpha) = \frac{1}{Z_M} \exp(-G(\mathbf{w}) - \alpha E(\mathbf{w})) \]

\[ \text{training scheme finds the locally most probable weight vector given the data} \]

**single neuron training:**

\[ \{z^{(n)}\}_{n=1}^N \quad D = \{t^{(n)}\}_{n=1}^N \]
\[ G(\mathbf{w}) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \]
\[ E(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 \]
\[ M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}) \quad \mathbf{w}^* = \arg \min_{\mathbf{w}} M(\mathbf{w}) \]
Single hidden layer neural networks

\[ x(a) = \frac{1}{1 + \exp(-a)} \]

\[ a = \sum_{k=1}^{K} w_k x_k \]

\[ x(a_k) = \frac{1}{1 + \exp(-a_k)} \]

\[ a_k = \sum_{d=1}^{D} W_{k,d} z_d \]
Sampling random neural network classifiers
Sampling random neural network classifiers

![Diagram showing sampling random neural network classifiers](image)
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers

\[ x, z_1, z_2 \]

\[ w_K, a_1, a_2, \ldots, a_K, Z_1, Z_2, W_{K \times 2} \]
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Sampling random neural network classifiers
Training a neural network with a single hidden layer

\[ x(a) = \frac{1}{1 + \exp(-a)} \]

\[ a = \sum_{k=1}^{K} w_k x_k \]

\[ x(a_k) = \frac{1}{1 + \exp(-a_k)} \]

\[ a_k = \sum_{d=1}^{D} W_{k,d} z_d \]
Training a neural network with a single hidden layer

Objective function:

\[ G(W, w) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \]

likelihood same as before
Training a neural network with a single hidden layer

**objective function:**

\[
G(W, \mathbf{w}) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right]
\]

\[
E(W, \mathbf{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2
\]

\[
x(a) = \frac{1}{1 + \exp(-a)}
\]

\[
a = \sum_{k=1}^K w_k x_k
\]

\[
x(a_k) = \frac{1}{1 + \exp(-a_k)}
\]

\[
a_k = \sum_{d=1}^D W_{k,d} z_d
\]

likelihood same as before

regulariser discourages extreme weights
Training a neural network with a single hidden layer

Objective function:

\[ G(W, \mathbf{w}) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \] likelihood same as before

\[ E(W, \mathbf{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2 \] regulariser discourages extreme weights

\[ \{W, \mathbf{w}^*\} = \arg \min_{W, \mathbf{w}} M(W, \mathbf{w}) = \arg \min_{W, \mathbf{w}} [G(W, \mathbf{w}) + \alpha E(W, \mathbf{w})] \]
Training a neural network with a single hidden layer

\[ x(a) = \frac{1}{1 + \exp(-a)} \]

\[ a = \sum_{k=1}^{K} w_k x_k \]

\[ x(a_k) = \frac{1}{1 + \exp(-a_k)} \]

\[ a_k = \sum_{d=1}^{D} W_{k,d} z_d \]

**objective function:**

\[
G(W, w) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right]
\]

\[
E(W, w) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{i,j} W_{i,j}^2
\]

\[
\{ W, w^* \} = \arg \min_{W, w} M(W, w) = \arg \min_{W, w} [G(W, w) + \alpha E(W, w)]
\]

\[
\frac{dG(W, w)}{dW_{ij}}
\]
Training a neural network with a single hidden layer

\[ x(\alpha) = \frac{1}{1 + \exp(-\alpha)} \]

\[ \alpha = \sum_{k=1}^{K} w_k x_k \]

\[ x(\alpha_k) = \frac{1}{1 + \exp(-\alpha_k)} \]

\[ a_k = \sum_{d=1}^{D} W_{k,d} z_d \]

**objective function:**

\[ G(W, w) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right] \] likelihood same as before

\[ E(W, w) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2 \] regulariser discourages extreme weights

\[ \{W, w^*\} = \arg\min_{W, w} M(W, w) = \arg\min_{W, w} [G(W, w) + \alpha E(W, w)] \]

\[ \frac{dG(W, w)}{dW_{ij}} = \sum_n \frac{dx^{(n)}}{dW_{ij}} \frac{dx^{(n)}}{dx^{(n)}} \frac{dx^{(n)}}{dW_{ij}} \]
Training a neural network with a single hidden layer

- Regularizer discourages extreme weights
- Likelihood same as before

**Objective function:**

\[
G(W, w) = - \sum_n [t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)})]
\]

Likelihood same as before

\[
E(W, w) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2
\]

Regularizer discourages extreme weights

\[
\{W, w^*\} = \underset{W, w}{\arg \min} M(W, w) = \underset{W, w}{\arg \min} [G(W, w) + \alpha E(W, w)]
\]

\[
\frac{dG(W, w)}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dW_{ij}}
\]
Training a neural network with a single hidden layer

\[
x(a) = \frac{1}{1 + \exp(-a)}
\]
\[
a = \sum_{k=1}^{K} w_k x_k
\]
\[
x(a_k) = \frac{1}{1 + \exp(-a_k)}
\]
\[
a_k = \sum_{d=1}^{D} W_{kd, d} z_d
\]

**Objective function:**

\[
G(W, w) = - \sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^{(n)}) \log (1 - x^{(n)}) \right]
\]

Likelihood same as before

\[
E(W, w) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2
\]

Regulariser discourages extreme weights

\[
\{ W, w^* \} = \arg \min_{W, w} M(W, w) = \arg \min_{W, w} [G(W, w) + \alpha E(W, w)]
\]

\[
\frac{dG(W, w)}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dW_{ij}}
\]

\[
= \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dx_i^{(n)}} \frac{dx_i^{(n)}}{dW_{ij}}
\]
Backpropagation

\[ x(a) = \frac{1}{1+\exp(-a)} \]

\[ a = \sum_{k=1}^{K} w_k x_k \]

\[ x(a_k) = \frac{1}{1+\exp(-a_k)} \]

\[ a_k = \sum_{d=1}^{D} W_{k,d} z_d \]

**objective function:**

\[ G(W, w) = -\sum_n \left[ t^{(n)} \log x^{(n)} + (1 - t^n) \log (1 - x^{(n)}) \right] \] likelihood same as before

\[ E(W, w) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2 \] regulariser discourages extreme weights

\[ \{W, w^*\} = \arg \min_{W, w} M(W, w) = \arg \min_{W, w} [G(W, w) + \alpha E(W, w)] \]

\[ \frac{dG(W, w)}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{dW_{ij}} = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dW_{ij}} \]

\[ = \sum_n \frac{dG(W, w)}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{dx^{(n)}}{da_i^{(n)}} \frac{da_i^{(n)}}{dW_{ij}} \]
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer

\( x \)

\( z_1 \)

\( z_2 \)

\( w_K \)

\( a_1, a_2, \ldots, a_K \)

\( W_K \times 2 \)
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Training a neural network with a single hidden layer
Hierarchical models with many hidden layers

\[ \begin{align*}
&\text{inputs layer} \\
&z_1 \quad z_2 \quad \ldots \quad z_D \\
&\text{hidden layer} \\
&a_1 \quad a_2 \quad \ldots \quad a_K
\end{align*} \]

\[ \begin{align*}
&\text{output} \\
&\mathcal{X} \quad w_K \\
&\text{hidden layer} \\
&a \quad a^{(2)} \quad \ldots \quad a^{(K)}
\end{align*} \]

\[ \begin{align*}
&\text{inputs layer} \\
&z_1 \quad z_2 \quad \ldots \quad z_D \\
&\text{hidden layer} \\
&a_1^{(1)} \quad a_2^{(1)} \quad \ldots \quad a_K^{(1)}
\end{align*} \]
Hierarchical models with many hidden layers

- **deep neural networks** have been a **longstanding goal of AI**

- initial attempts in the ’90s and ’80s fell into **poor local optima**

- resurgence of interest in neural networks: result of **better initialisation methods**
  - **method 1**: unsupervised pre-training (e.g. using a restricted Boltzmann machine)
  - **method 2**: recursively apply backprop. (take care to initialise scales of weights carefully)
Training multi-layer neural networks: one layer at a time

Train input and output weights of two-layer network using backprop.
Training multi-layer neural networks: one layer at a time
Training multi-layer neural networks: one layer at a time

The figure shows a neural network with a new layer inserted into it. The network has an input layer $x$, a hidden layer $a_1, a_2, \ldots, a_D$, and an output layer $z_1, z_2, \ldots, z_K$. The weights between layers are denoted as $W_D^{(2)}$ and $W_D^{(1)}$. The process of inserting a new layer into the network is indicated by the text "insert new layer into network."
Training multi-layer neural networks: one layer at a time

- Train new layer's weights and output layer using backprop.
- Output weights initialised at old values.

Diagram:
- $x$ connected to $\mathbf{a}$.
- $\mathbf{a}_1^{(2)}, \mathbf{a}_2^{(2)}, \ldots, \mathbf{a}_D^{(2)}$ connected to $\mathbf{w}_D$.
- $\mathbf{a}_1^{(1)}, \mathbf{a}_2^{(1)}, \ldots, \mathbf{a}_D^{(1)}$ connected to $W_D^{(1)}$.
- $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K$ connected to $W_D^{(1)}$.

- Output weights initialised at old values.
- Train new layer's weights and output layer using backprop.
Training multi-layer neural networks: one layer at a time

break apart network

\[ x \]

\[ a \]

\[ w_D \]

break apart network

\[ a_1^{(2)} \]
\[ a_2^{(2)} \]
\[ \ldots \]
\[ a_D^{(2)} \]

\[ W_{D\times K}^{(2)} \]

\[ a_1^{(1)} \]
\[ a_2^{(1)} \]
\[ \ldots \]
\[ a_D^{(1)} \]

\[ W_{D\times K}^{(1)} \]

\[ z_1 \]
\[ z_2 \]
\[ \ldots \]
\[ z_K \]
Training multi-layer neural networks: one layer at a time

\[ x \rightarrow a \rightarrow a_1^{(3)} \rightarrow a_2^{(3)} \rightarrow \ldots \rightarrow a_D^{(3)} \rightarrow w_D \rightarrow W_D^{(3)} \rightarrow W_D^{(2)} \rightarrow W_D^{(1)} \rightarrow z_1 \rightarrow z_2 \rightarrow \ldots \rightarrow z_K \]

insert new layer into network
Training multi-layer neural networks: one layer at a time

output weights initialised at old values

train new layer's weights and output layer using backprop
Training multi-layer neural networks: one layer at a time

fine-tune all layer's weights simultaneously
initialising from trained values
Hierarchical neural networks at Google

- Google infrastructure being developed to train neural networks (Jeff Dean)
- Numbers. **model**: 1 billion connections. **Dataset**: 10 million 200x200 pixel images. **training**: 1,000 machines (16,000 cores)
- 15.8% accuracy in recognizing 22,000 object categories from ImageNet
- leap of 70% relative improvement to previous state-of-the-art

![Image of 32 images in 4x8 grid with a blurred image on the right]
Summary

- Supervised artificial neural networks fit a non-linear function that maps from input features \((z)\) to output targets \((t)\)

- Networks with hidden layers can be fit using gradient descent using an algorithm called backpropagation

- Finds the maximum a posteriori setting of the network parameters, given the training data

- Regularisation is required to stop over-fitting

- Smart initialisation is required to stop the algorithm from falling into local optima

Demo: http://yann.lecun.com/exdb/lenet/