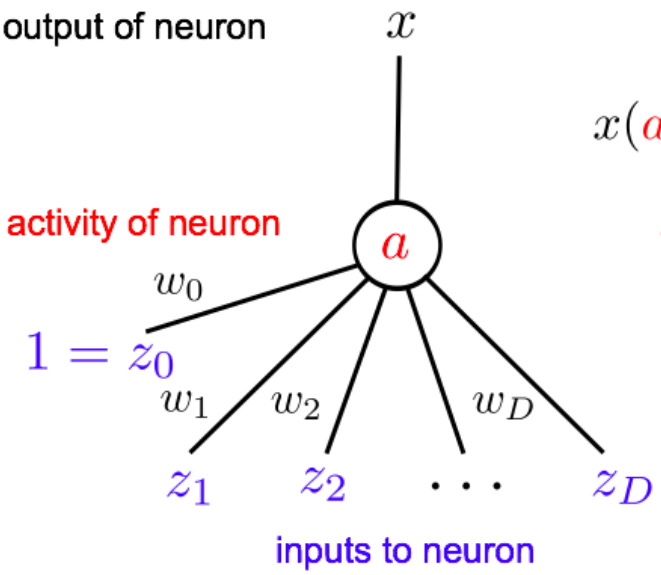
Lecture : Deep Learning
Neural Networks

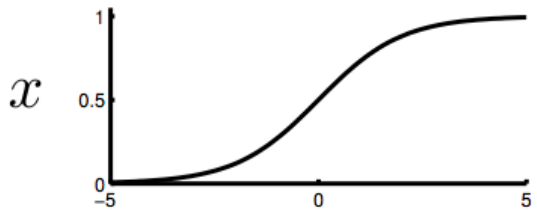
Riashat Islam

Reasoning and Learning Lab
McGill University

September 3, 2017

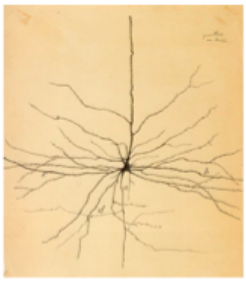
A Single Neuron



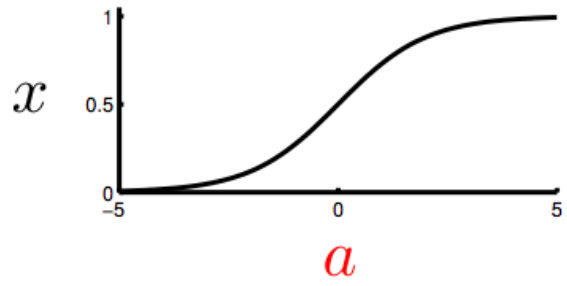


$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

$$a = \sum_{d=0}^{D} w_d z_d$$

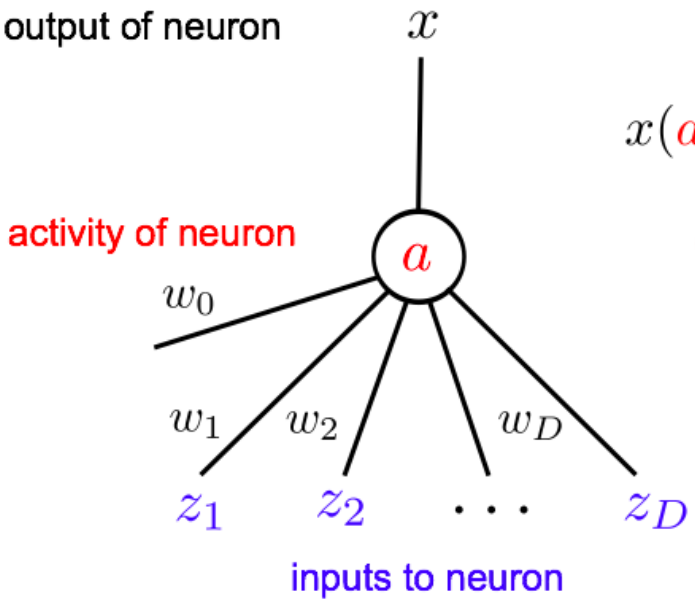


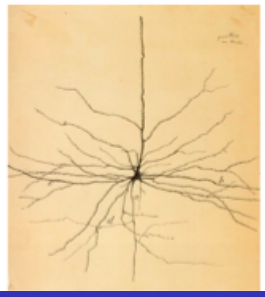
A Single Neuron



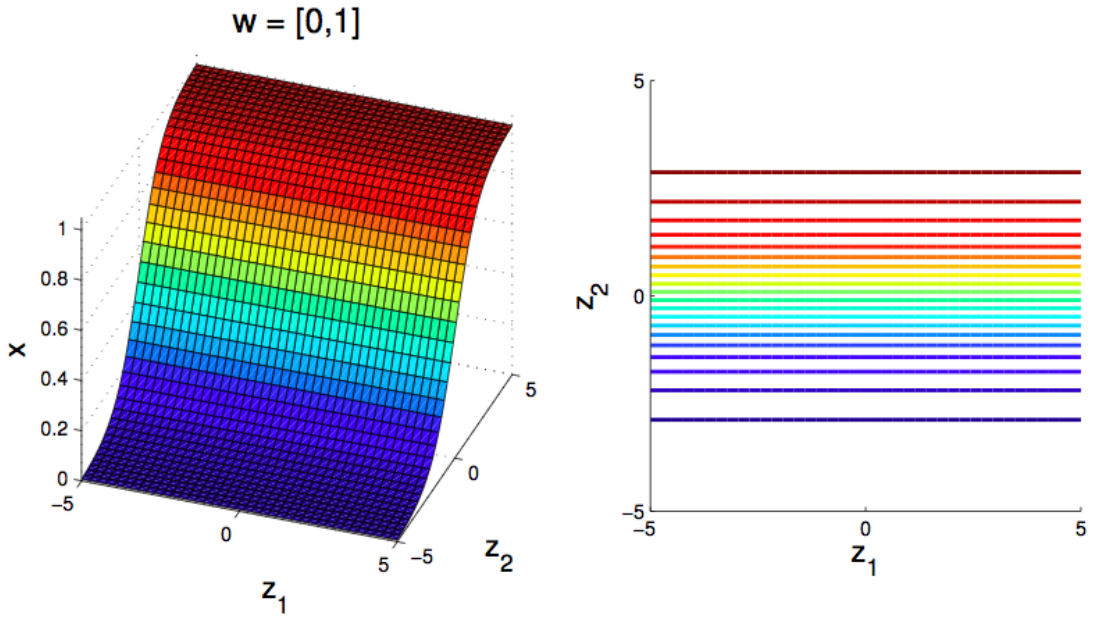
$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

$$a = w_0 + \sum_{d=1}^{D} w_d z_d$$



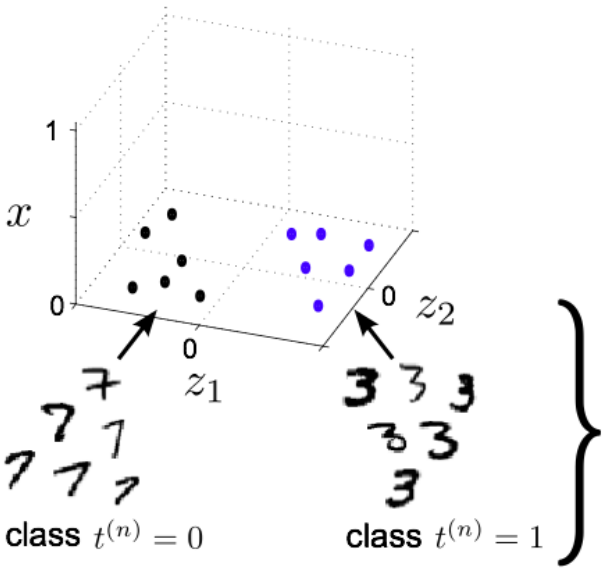


Input-Output of a Single Neuron

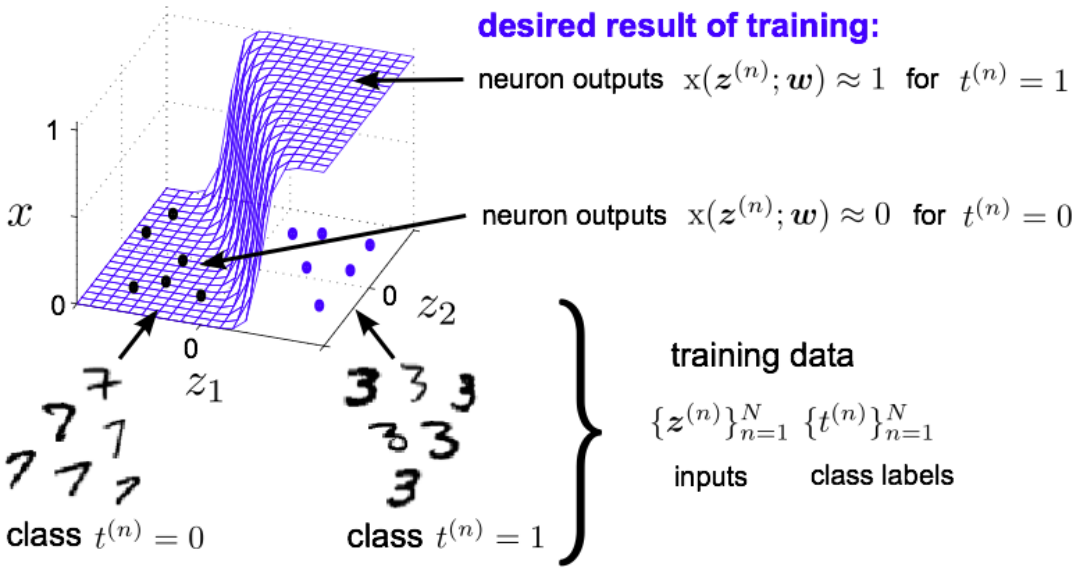


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

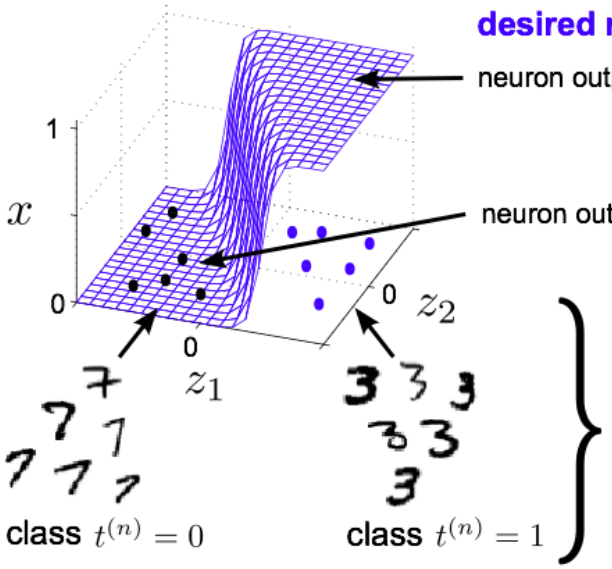
Training a Single Neuron



Training a Single Neuron



Training a Single Neuron



desired result of training:

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

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

training data

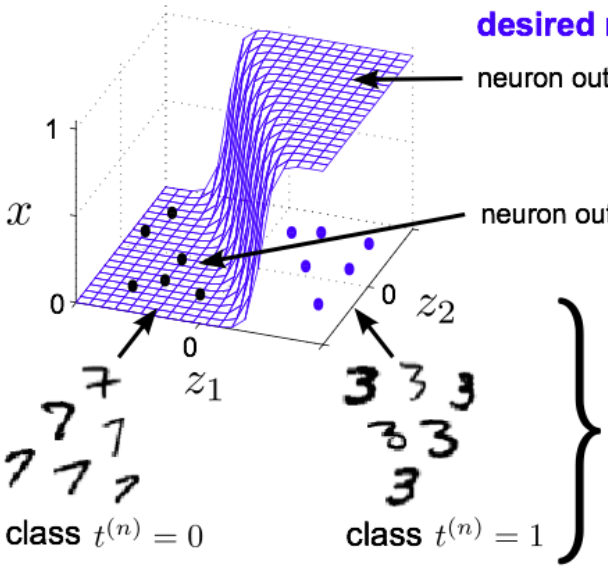
$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

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

Training a Single Neuron



desired result of training:

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

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

training data

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

inputs class labels

objective function:

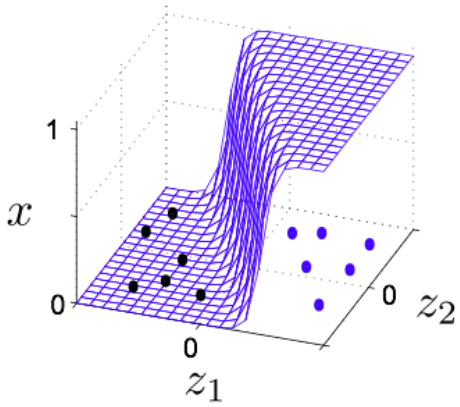
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

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

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

} encourages neuron output to match training data

Training a Single Neuron



training data

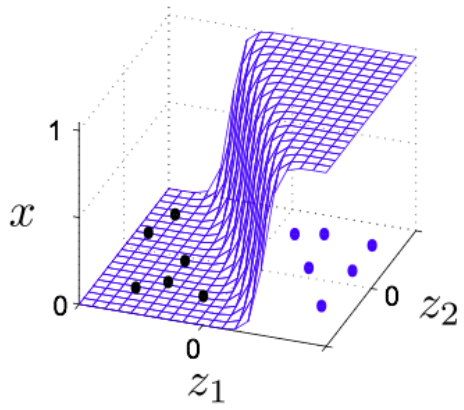
$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

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

Training a Single Neuron



training data

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

inputs class labels

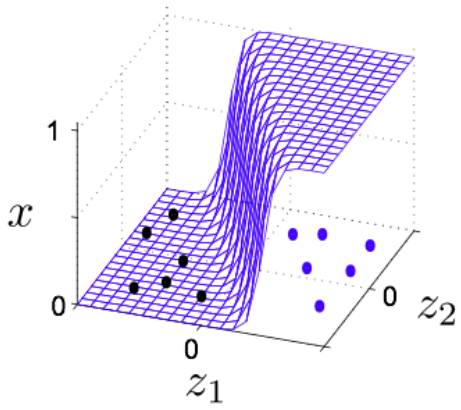
objective function:

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

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} G(\boldsymbol{w})$$

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

Training a Single Neuron



training data

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

inputs class labels

objective function:

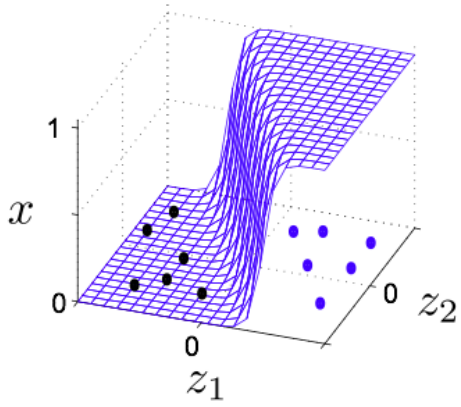
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} G(\boldsymbol{w})$$

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

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$ = prediction error x feature

Training a Single Neuron



training data

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

inputs class labels

objective function:

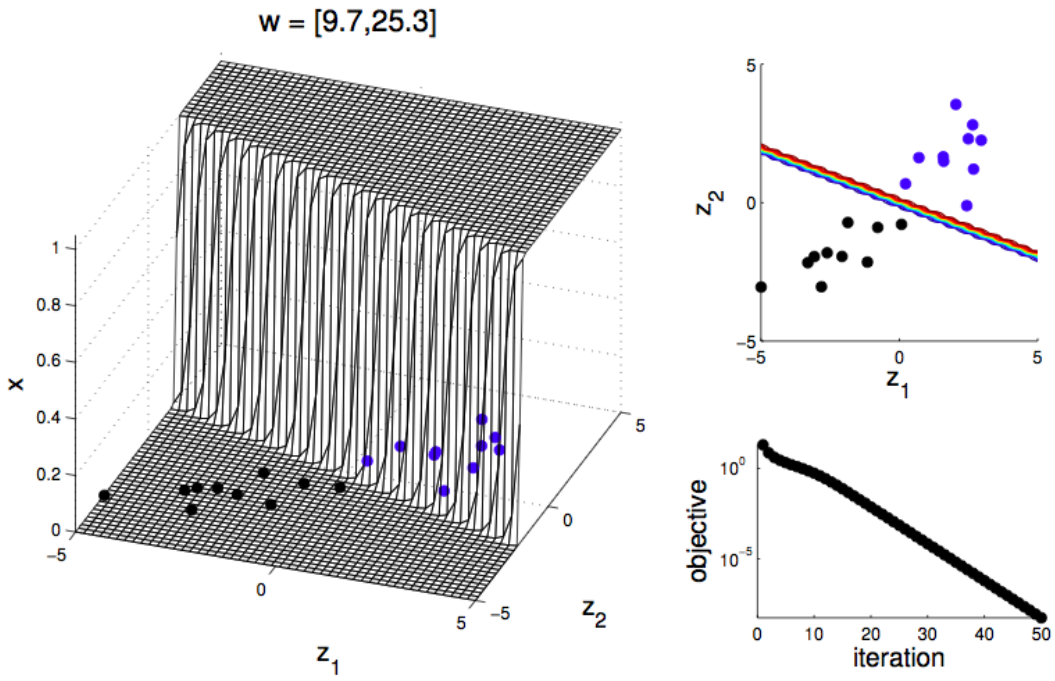
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} G(\boldsymbol{w})$ choose the weights that minimise the network's surprise about the training data

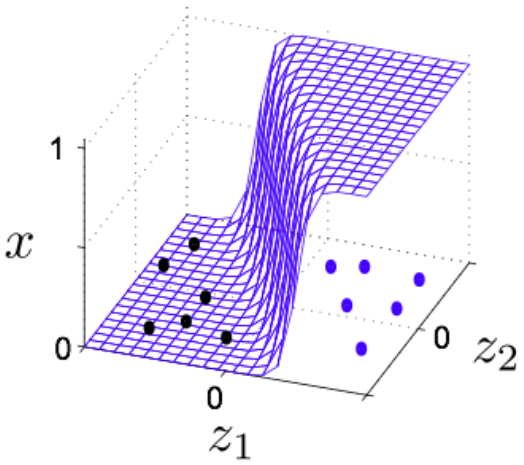
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$ = prediction error x feature

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w})$ iteratively step down the objective (gradient points up hill)

Training a Single Neuron



Over-Fitting and Weight Decay



training data

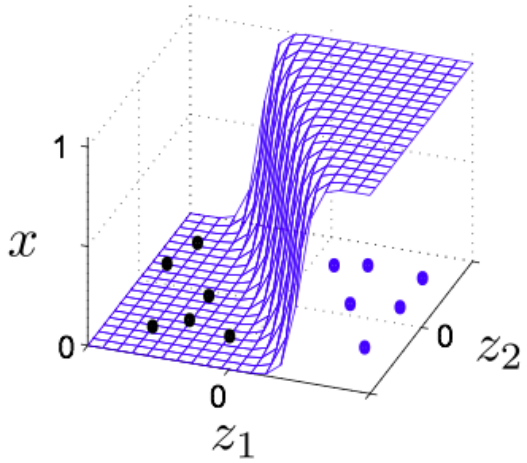
$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

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

Over-Fitting and Weight Decay



training data

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

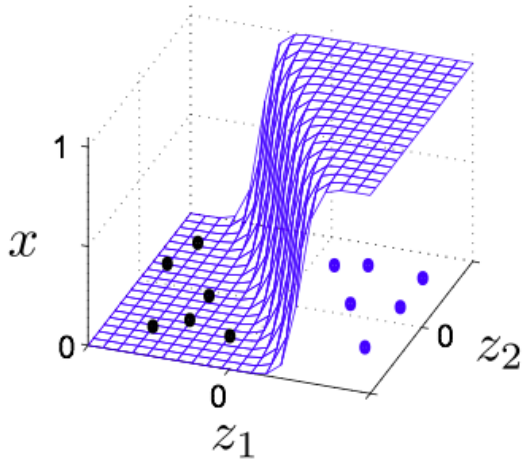
inputs class labels

objective function:

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

$E(\boldsymbol{w}) = \frac{1}{2} \sum_i w_i^2$ regulariser discourages the network using extreme weights

Over-Fitting and Weight Decay



training data

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

inputs class labels

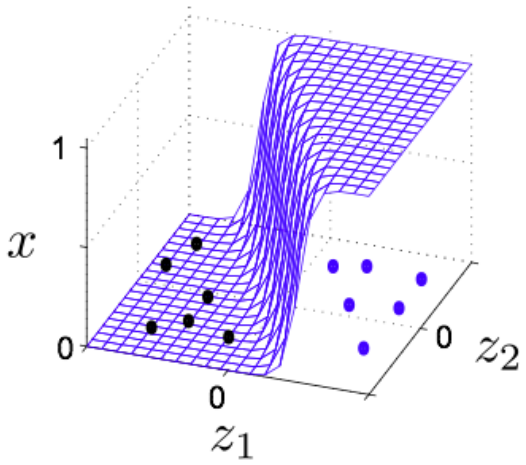
objective function:

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

$$E(\boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2$$ regulariser discourages the network using extreme weights

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

Over-Fitting and Weight Decay



training data

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

inputs class labels

objective function:

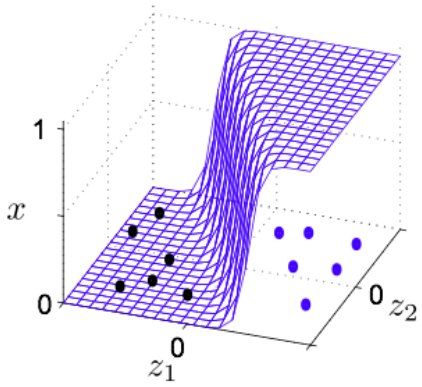
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]$$

$$E(\boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2$$ regulariser discourages the network using extreme weights

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} M(\boldsymbol{w}) = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)} + \alpha \boldsymbol{w}$$ weight decay - shrinks weights towards zero

Probabilistic Interpretation of the single neuron



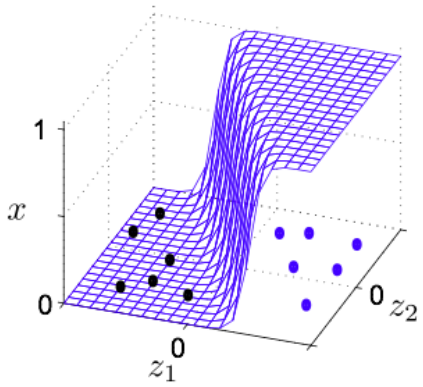
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N} \quad D = \{t^{(n)}\}_{n=1}^{N}$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log\left(1 - \mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z})=x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z})=1-x$$

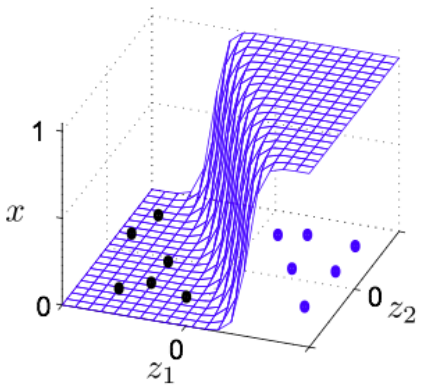
single neuron training:

$$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N} \qquad D=\{t^{(n)}\}_{n=1}^{N}$$

$$G(\boldsymbol{w})=-\sum_n \left[t^{(n)}\log \mathrm{x}^{(n)}+(1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w})=\tfrac{1}{2}\sum_i w_i^2$$

$$M(\boldsymbol{w})=G(\boldsymbol{w})+\alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^*=\arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z})=x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z})=1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z})=x^t(1-x)^{1-t}$$

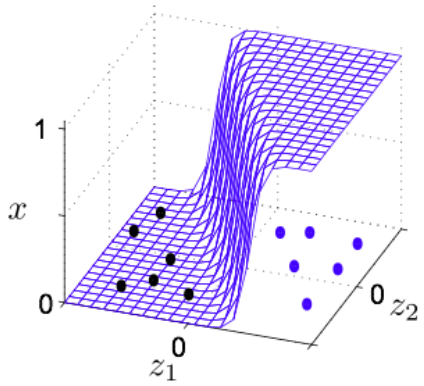
single neuron training:

$$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D=\{t^{(n)}\}_{n=1}^N$$

$$G(\boldsymbol{w})=-\sum_n\left[t^{(n)}\log \mathrm{x}^{(n)}+(1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w})=\tfrac{1}{2}\sum_i w_i^2$$

$$M(\boldsymbol{w})=G(\boldsymbol{w})+\alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^*=\arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

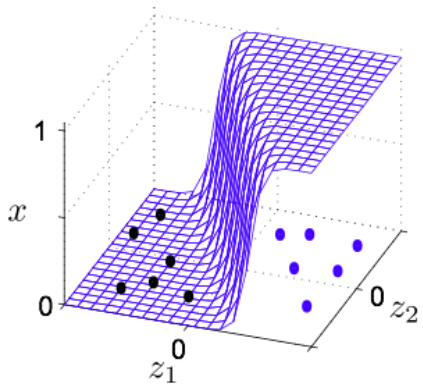
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

$$p(D|\boldsymbol{w},\boldsymbol{z}) = \exp(-G(\boldsymbol{w}))$$

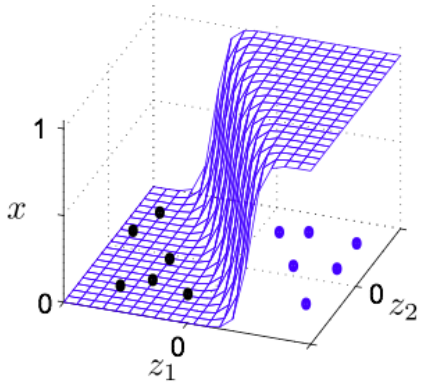
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \quad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

$$p(D|\boldsymbol{w},\boldsymbol{z}) = \exp(-G(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E(\boldsymbol{w}))$$

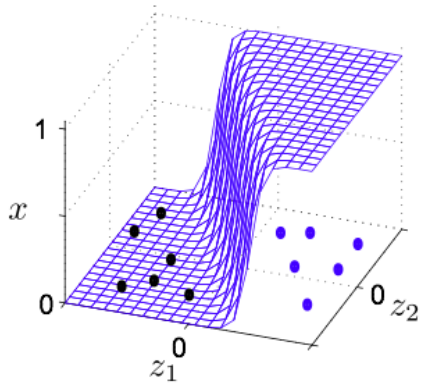
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)}\log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

$$p(D|\boldsymbol{w},\boldsymbol{z}) = \exp(-G(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|D,\alpha) = \frac{1}{p(D|\alpha)}p(D|\boldsymbol{w})p(\boldsymbol{w}|\alpha) \quad \text{Bayes' Rule}$$

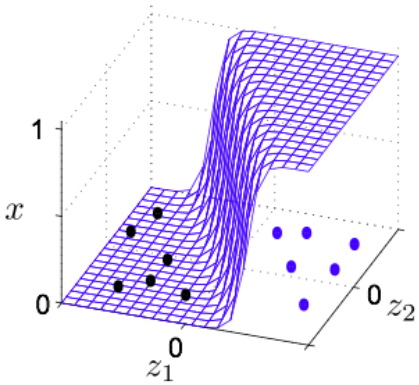
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \quad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)}\log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

$$p(D|\boldsymbol{w},\boldsymbol{z}) = \exp(-G(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|D,\alpha) = \frac{1}{p(D|\alpha)}p(D|\boldsymbol{w})p(\boldsymbol{w}|\alpha) = \frac{1}{Z_M}\exp(-G(\boldsymbol{w}) - \alpha E(\boldsymbol{w}))$$

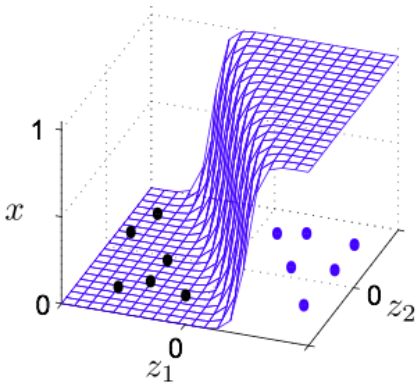
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)}\log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Probabilistic Interpretation of the single neuron



$$p(t=1|\boldsymbol{w},\boldsymbol{z}) = x \qquad p(t=0|\boldsymbol{w},\boldsymbol{z}) = 1-x$$

$$p(t|\boldsymbol{w},\boldsymbol{z}) = x^t(1-x)^{1-t}$$

$$= \exp(t\log x + (1-t)\log(1-x))$$

$$p(D|\boldsymbol{w},\boldsymbol{z}) = \exp(-G(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|\alpha) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E(\boldsymbol{w}))$$

$$p(\boldsymbol{w}|D,\alpha) = \frac{1}{p(D|\alpha)}p(D|\boldsymbol{w})p(\boldsymbol{w}|\alpha) = \frac{1}{Z_M}\exp(-G(\boldsymbol{w}) - \alpha E(\boldsymbol{w}))$$

$\Longrightarrow$ **training scheme finds the locally most probable weight vector given the data**

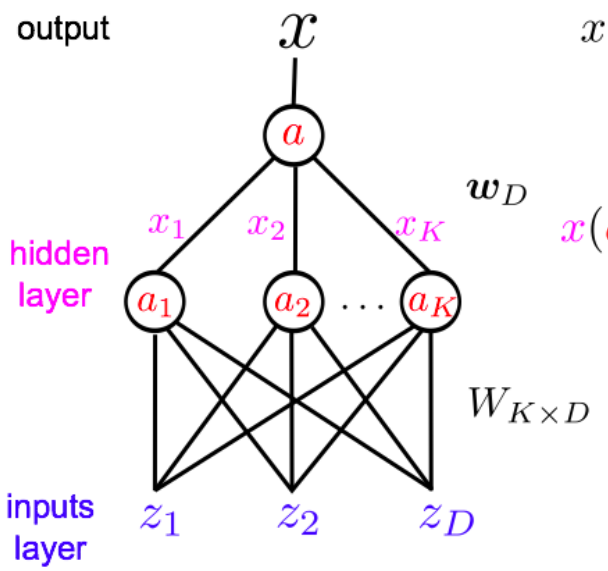
single neuron training:

$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D = \{t^{(n)}\}_{n=1}^N$

$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)}\log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right] \qquad E(\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2$

$M(\boldsymbol{w}) = G(\boldsymbol{w}) + \alpha E(\boldsymbol{w}) \qquad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w})$

Single Hidden Layer Neural Networks



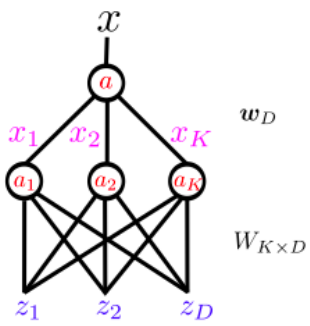
$$x(a) = \frac{1}{1+\exp(-a)}$$

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

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

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

Training of a Neural Network with Single Hidden Layer



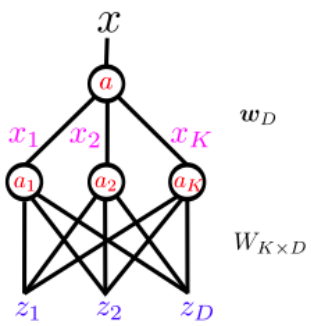
$$x(a) = \frac{1}{1+\exp(-a)}$$

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

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

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

Training of a Neural Network with Single Hidden Layer



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

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

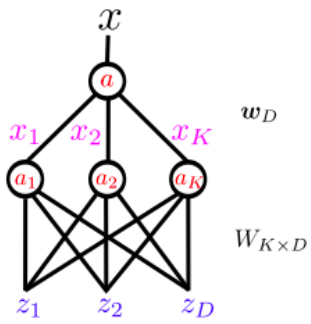
$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

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

objective function:

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

Training of a Neural Network with Single Hidden Layer



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

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

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

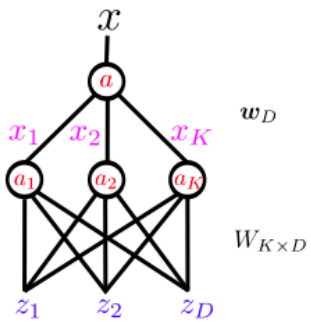
$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

objective function:

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

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

Training of a Neural Network with Single Hidden Layer



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

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

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

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

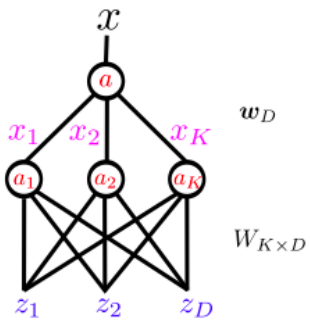
objective function:

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

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

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

Training of a Neural Network with 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:

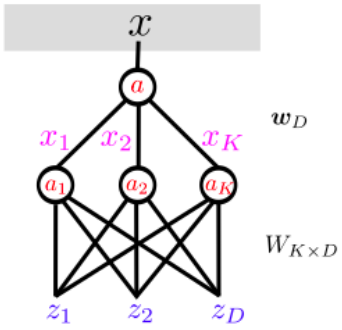
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

$$E(W, \boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2 + \tfrac{1}{2} \sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

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

$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}}$$

Training of a Neural Network with 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:

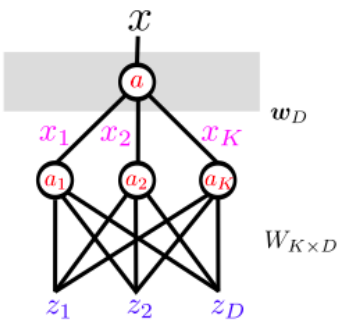
$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$ likelihood same as before

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

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

$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}}$$

Training of a Neural Network with 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:

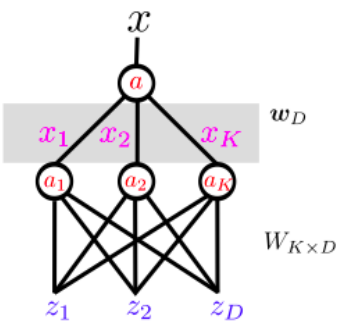
$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1-t^n) \log\left(1-\mathrm{x}^{(n)}\right)\right]$ likelihood same as before

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

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

$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

Training of a Neural Network with 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:

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

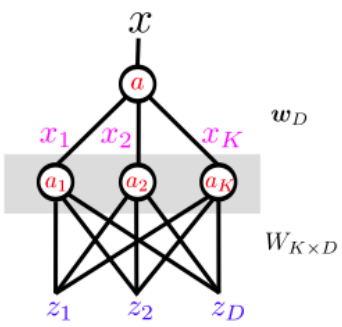
$E(W, \boldsymbol{w}) = \frac{1}{2} \sum_i w_i^2 + \frac{1}{2} \sum_{ij} W_{ij}^2$ regulariser discourages extreme weights

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

$$\frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

$$= \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}}$$

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:

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

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

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

$$\frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

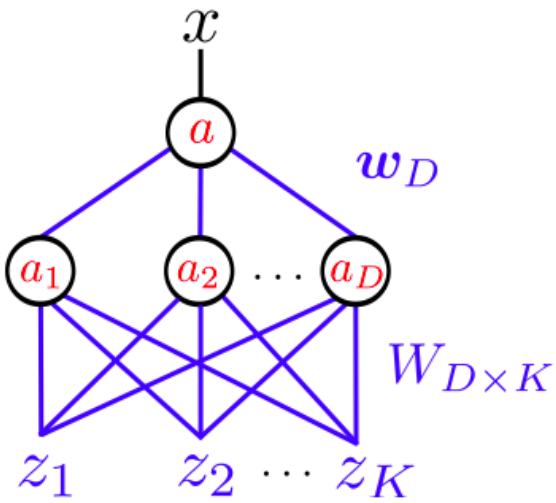
$$= \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}a_i^{(n)}} \frac{\mathrm{d}a_i^{(n)}}{\mathrm{d}W_{ij}}$$

Hierarchical Models with many hidden layers

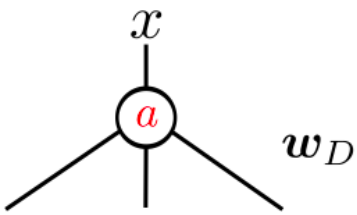
- **deep neural networks** have been a **longstanding goal of AI**
- initial attempts in the '90s and '80s fell into **poor local optima**
- resurgence of interest in neural networks: result of **better initialisation methods**
  - **method 1**: unsupervised pre-training (e.g. using a restricted Boltzmann machine)
  - **method 2**: recursively apply backprop. (take care to initialise scales of weights carefully)

Training multi-layer neural networks : One Layer at a time

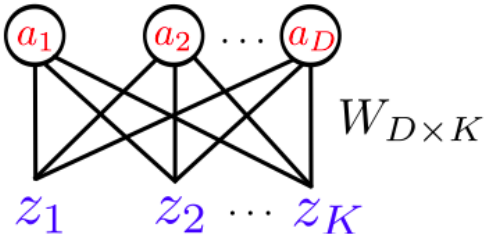
train input and output weights of two layer network using backprop



Training multi-layer neural networks : One Layer at a time

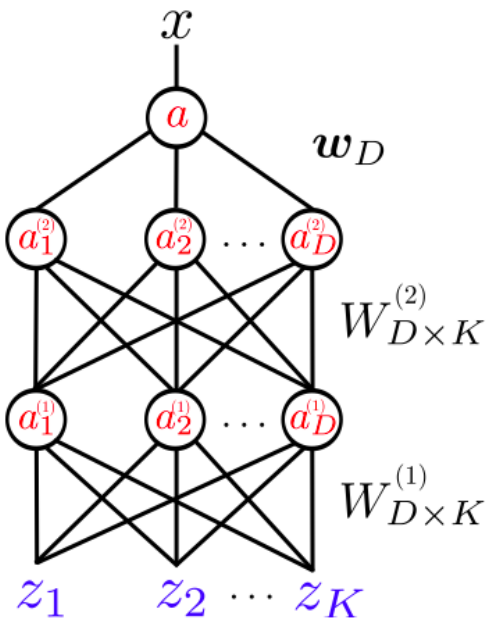


break apart network

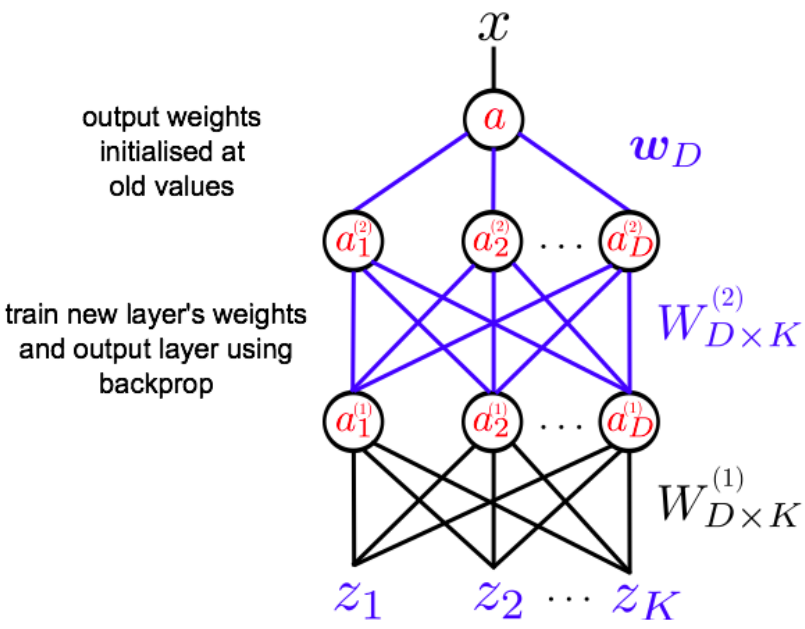


Training multi-layer neural networks : One Layer at a time

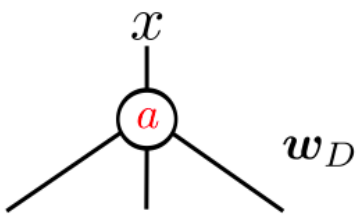
insert new layer into network



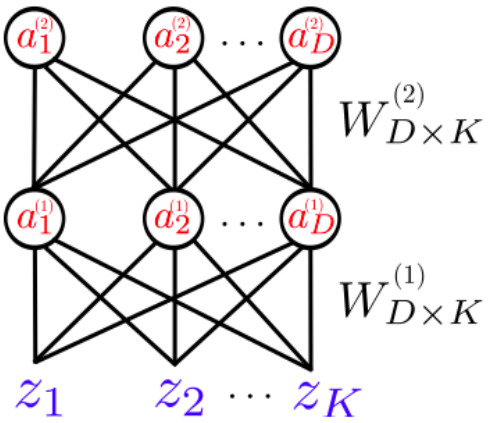
Training multi-layer neural networks : One Layer at a time



Training multi-layer neural networks : One Layer at a time

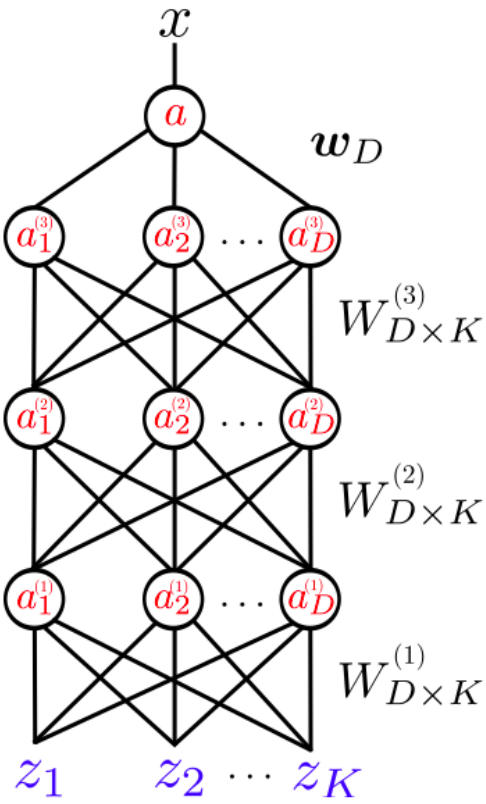


break apart network

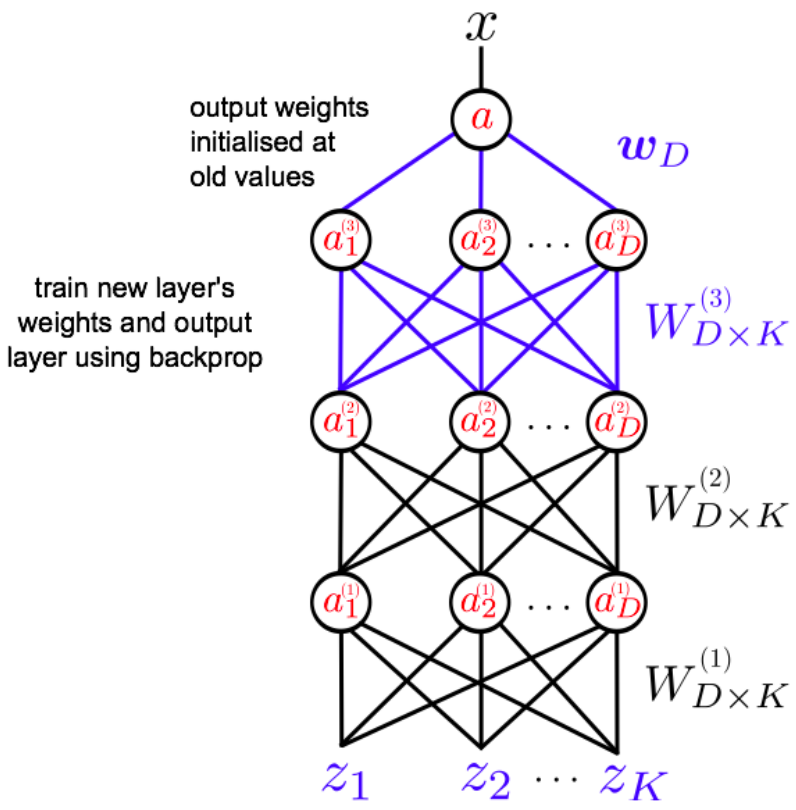


Training multi-layer neural networks : One Layer at a time

insert new layer into network

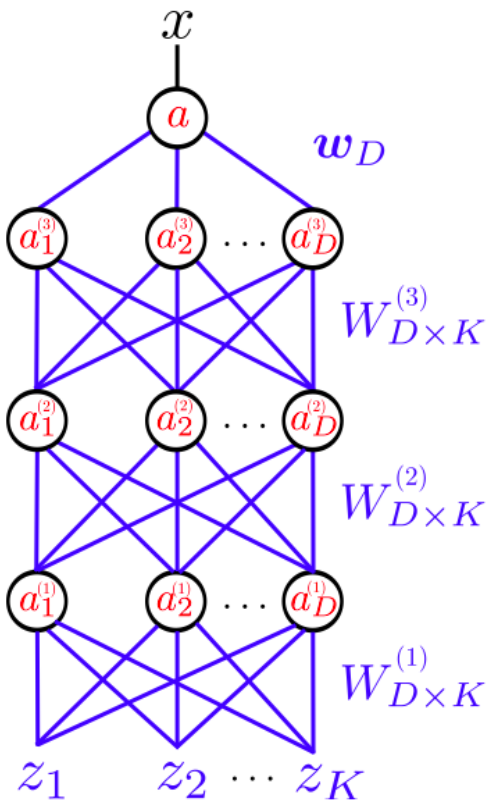


Training multi-layer neural networks : One Layer at a time



Training multi-layer neural networks : One Layer at a time

fine-tune all layer's weights simultaneously initialising from trained values



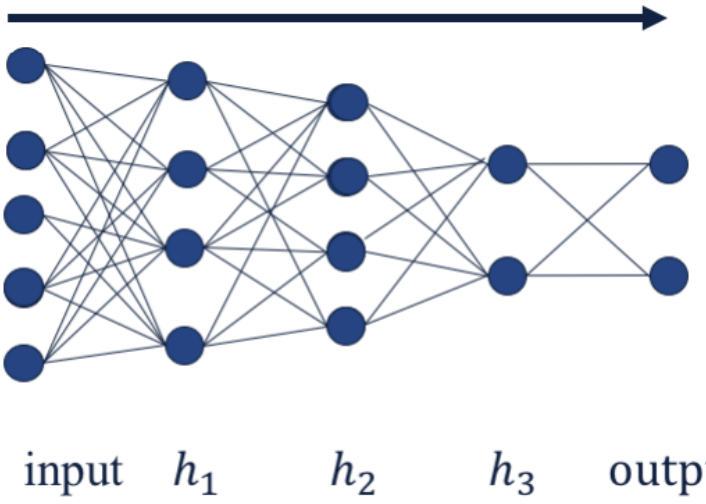
Summary

- 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/`

Deep Learning

More Hidden Layers



Same algorithm holds for more hidden layers

Deep Learning Libraries

Tensorflow

Torch

Theano

Pytorch

Dynet

They allow automatic differentiation. You can just write the network, get the gradients for free.

Comments on Training

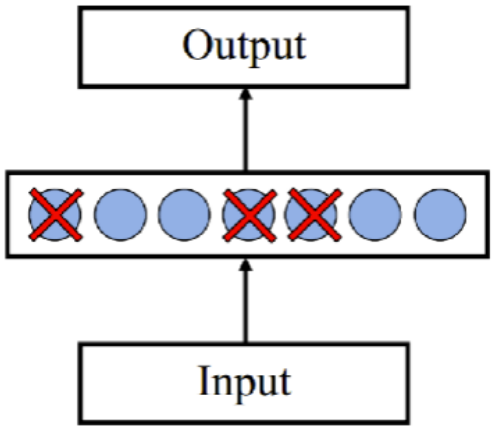
- No guarantee of convergence, may oscillate or reach a local minima
- In practice, many large networks can be trained on large amounts of data for realistic problems
- Termination criteria : Number of epochs; Threshold on training set error; No decrease in error; Increase error on a validation set
- To avoid local minima : several trials with different random initial weights with majority or voting techniques

Overfitting and Perception

- Running too many epochs may over-train the network and result in overfitting.
- Keep a held out validation set and test accuracy after each epoch, maintain weights for the best performing network on the validation set, and return it when performance decreases significantly beyond that.
- Too few hidden units can prevent the data from adequately fitting the data, too many hidden units lead to overfitting. You can tune to get the best number of hidden units for your task.
- Another approach to prevent overfitting : Change error function to include a term for the sum of squares of the weights in the network.

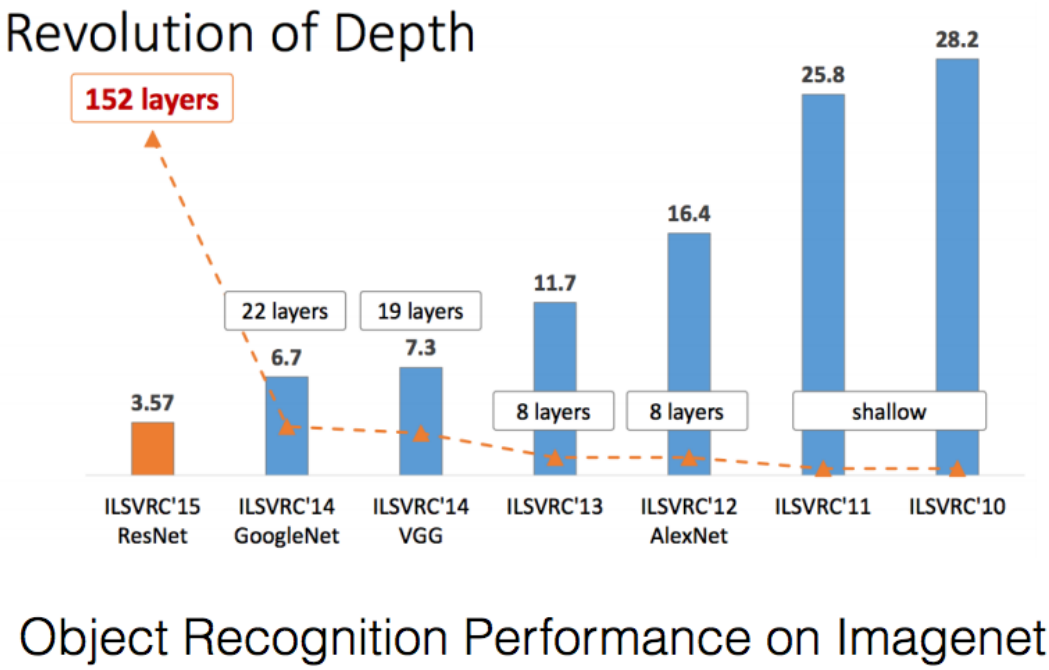
Dropout Training

Proposed by Hinton et al 2012



Each time, decide on whether to delete a hidden unit with some probability **p**

Deep Learning



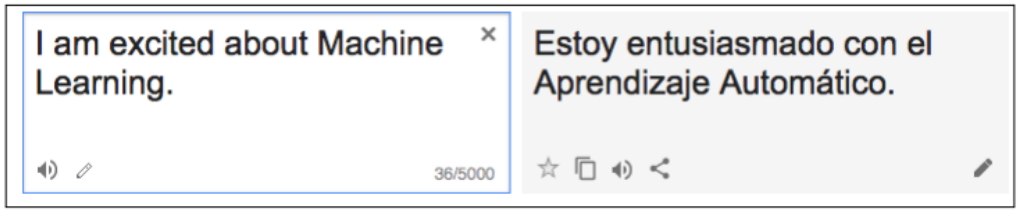
Object Recognition Performance on Imagenet

Recurrent Neural Networks

Motivation

Not all problems can be converted into one with fixed length inputs and outputs

Machine Translation : translating English sentence to other languag



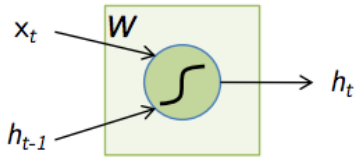
Speech Recognition : translating spoken sentence to written sentenc

Hard or impossible to choose a large enough window, there can always be a new sentence longer than anything seen

Recurrent Neural Networks

- Recurrent Neural Networks take as input each element of a sequence one by one. (Think of this as taking input $\mathbf{x_1}$ at time step 1, $\mathbf{x_2}$ at time step 2, etc)
- Recurrent Neural Networks take the previous output or hidden states as inputs.
- The composite input at time t has some historical information about the happenings at time T < t
- RNNs are useful as their intermediate values (state) can store information about past inputs for a time that is not fixed a priori

Recurrent Neural Networks



Input at time t is $\mathbf{x_t}$

State at the end of time (t-1) is $\mathbf{h_{t-1}}$

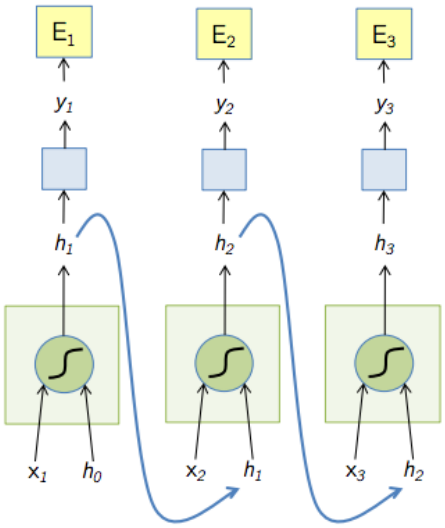
State at the end of time t is $\mathbf{h_t}$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

You can obviously replace **tanh** by your favorite non-linear differentiable function

How is it different from a standard neuron ??

Feeding Sequence to RNNs



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

You can output $\mathbf{y_t}$ at each time step (based on your problem), based on $\mathbf{h_t}$

$$y_t = F(h_t)$$

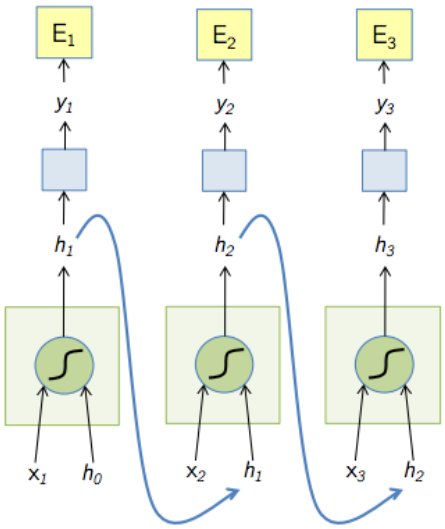
often linear function of $\mathbf{h_t}$

Finally define error $\mathbf{E_t}$ based on $\mathbf{y_t}$ and the target output at time **t**

$$E_t = Loss(y_t, GT_t)$$

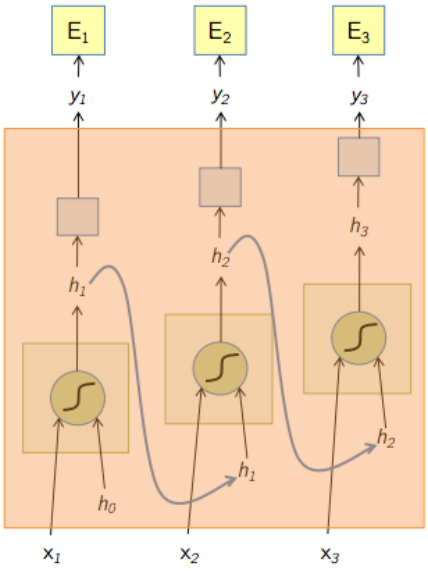
$\mathbf{GT_t}$ is target at time **t**

RNNs



- Note that the weights are shared over time
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

Backpropagation Through Time



- Treat the unfolded network as one big feed-forward network
- This unfolded network accepts the whole time series as input
- The weight update is computed for each copy in the unfolded network, then summed (or averaged) and then applied to RNN weights

An Issue with RNNs

Recall, back propagation requires computation of gradients with respect to each variable. For RNNs, consider the gradient of $\mathbf{E_t}$ with respect to $\mathbf{h_1}$

$$\frac{\delta E_t}{h_1} = \left(\frac{\delta E_t}{\delta y_t}\right)\left(\frac{\delta y_t}{\delta h_1}\right)$$
$$= \left(\frac{\delta E_t}{\delta y_t}\right)\left(\frac{\delta y_t}{\delta h_t}\right)\left(\frac{\delta h_t}{\delta h_{t-1}}\right)\cdots\left(\frac{\delta h_2}{\delta h_1}\right)$$

Product of a lot of terms can shrink to zero (**vanishing gradient**) or explode to infinity (**exploding gradient**). Exploding gradients are often controlled by clipping the values to a max value. One way to handle vanishing gradient problem is to try to have some relationship between $\mathbf{h_t}$ and $\mathbf{h_{t-1}}$ such that each $\left(\frac{\delta h_t}{\delta h_{t-1}}\right) = 1$

Long Short Term Memory (LSTM) try to do that, but vanishing gradients still a problem. As a result, capturing long range dependencies is still challenging.

Summary

- RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps
- Various input / output scenarios possible (Single / Multiple)
- Exploding gradients are handled by gradient clipping, LSTMs are a way towards handling vanishing gradients.

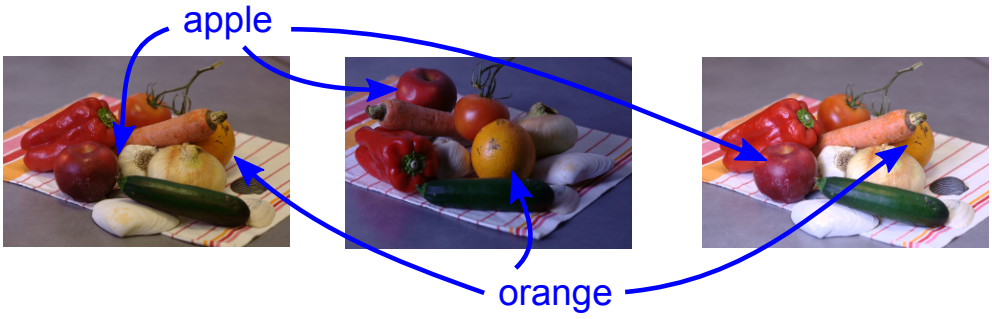
You can read more about LSTMs at http://arunmallya.github.io/ . A major part of RNN slides were taken from there.

Questions ??

Convolutional Neural Networks

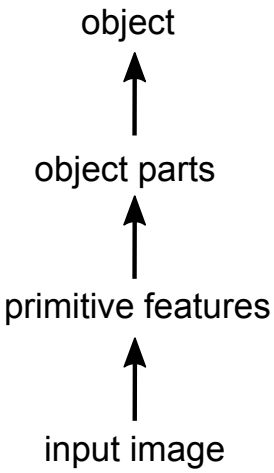
Big picture

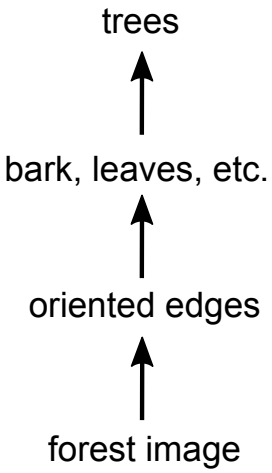
- **Goal**: how to produce good internal representations of the visual world to support recognition...
  - detect and classify objects into categories, independently of pose, scale, illumination, conformation, occlusion and clutter
- how could an artificial vision system learn appropriate internal representations automatically, the way humans seem to by simply looking at the world?
- **previously in CV and the course**: hand-crafted feature extractor
- **now in CV and the course**: learn suitable representations of images

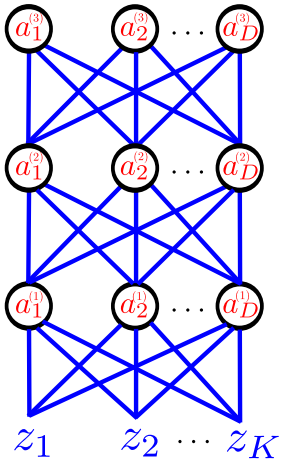


Why use hierarchical multi-layered models?

Argument 1: visual scenes are hierachically organised

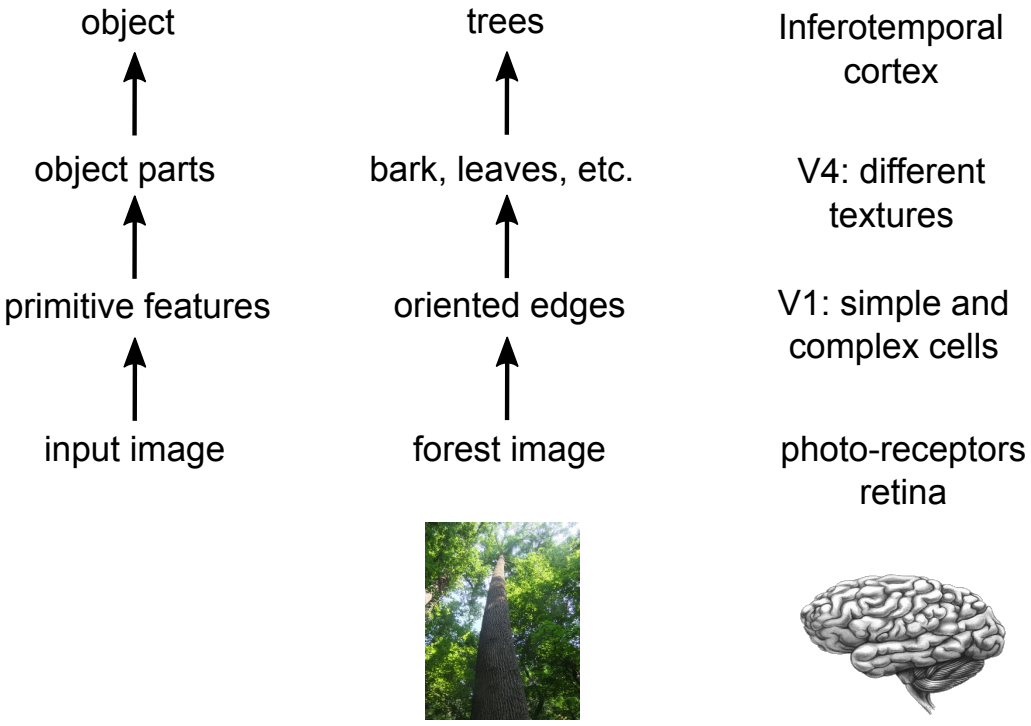






Why use hierarchical multi-layered models?

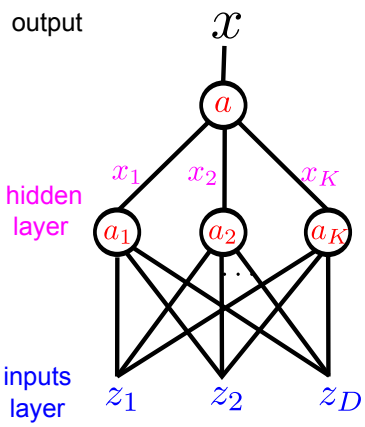
Argument 2: biological vision is hierachically organised



Why use hierarchical multi-layered models?

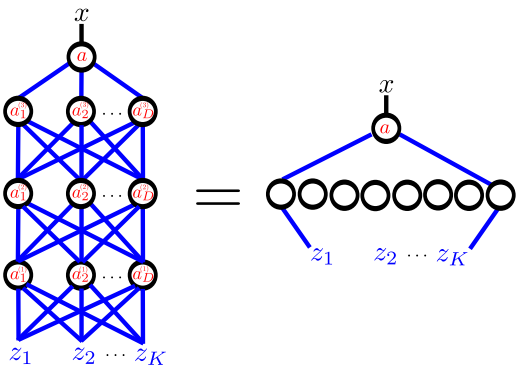
Argument 3: shallow architectures are inefficient at representing deep functions

single layer neural network
implements: $x = f_\theta(\mathbf{z})$



networks we met last lecture
with large enough single hidden layer
can implement **any** function
'**universal approximator**'

shallow networks can be
computationally inefficient



however, if the function is 'deep'
a very large hidden layer may
be required

What's wrong with standard neural networks?

How many parameters does this neural network have?

$$|\theta| = 3D^2 + D$$

For a small 32 by 32 image:

$$|\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6$$

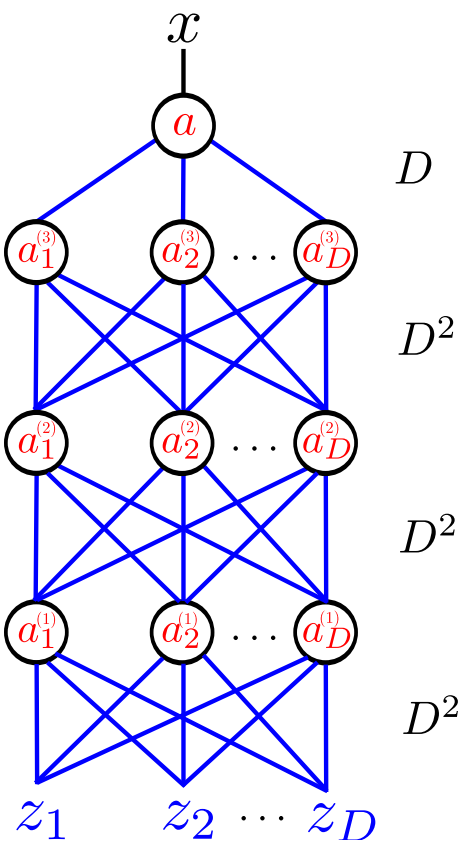
Hard to train

over-fitting and local optima

Need to initialise carefully

layer wise training
unsupervised schemes

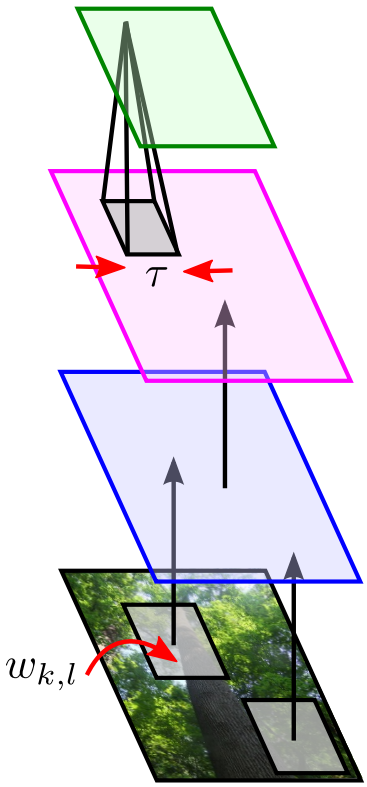
Convolutional nets reduce the number of parameters



The key ideas behind convolutional neural networks

- **image statistics are translation invariant** (objects and viewpoint translates)
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters
- **expect learned low-level features to be local** (e.g. edge detector)
  - build this into the model by allowing only local connectivity
  - reduces the numbers of parameters further
- **expect high-level features learned to be coarser** (c.f. biology)
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again

Building block of a convolutional neural network



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

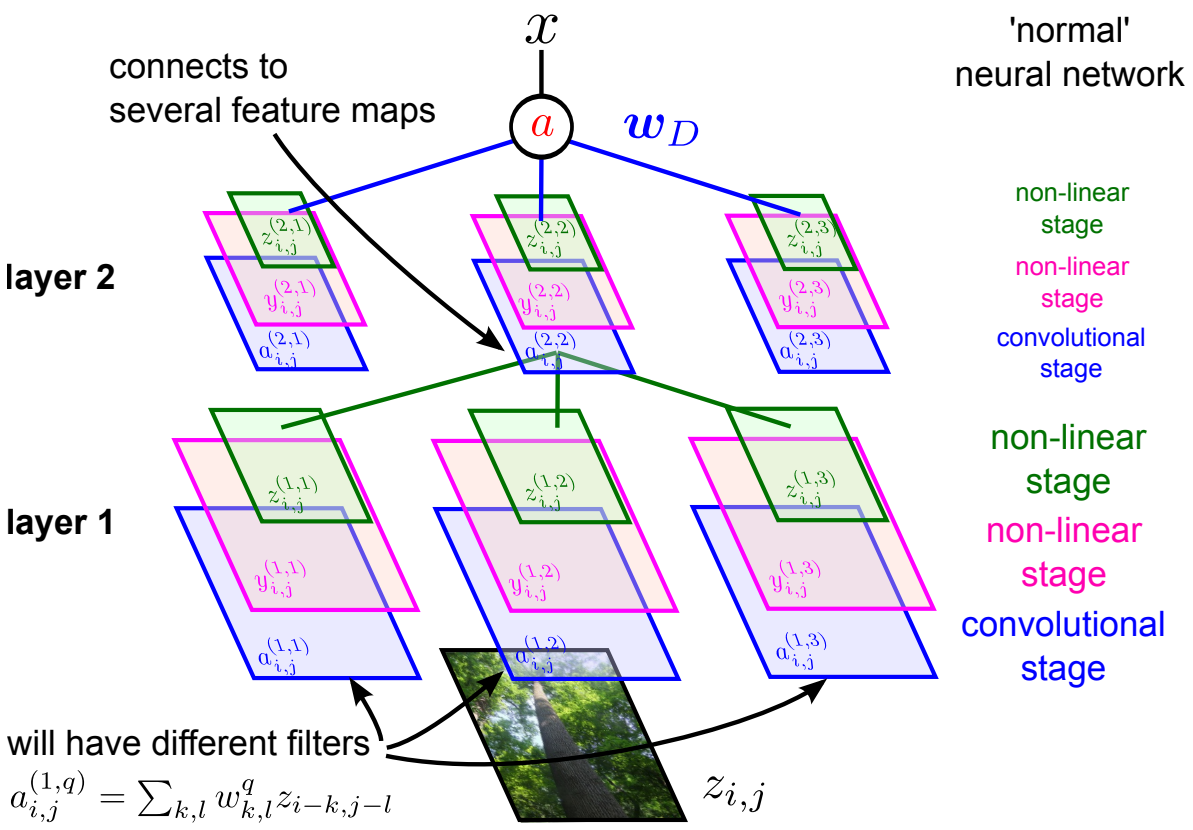
only parameters

convolutional stage

$z_{i,j}$

input image

Full convolutional neural network



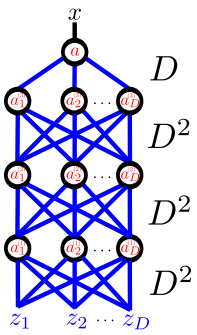
How many parameters does a convolutional network have?

How many parameters does this neural network have?

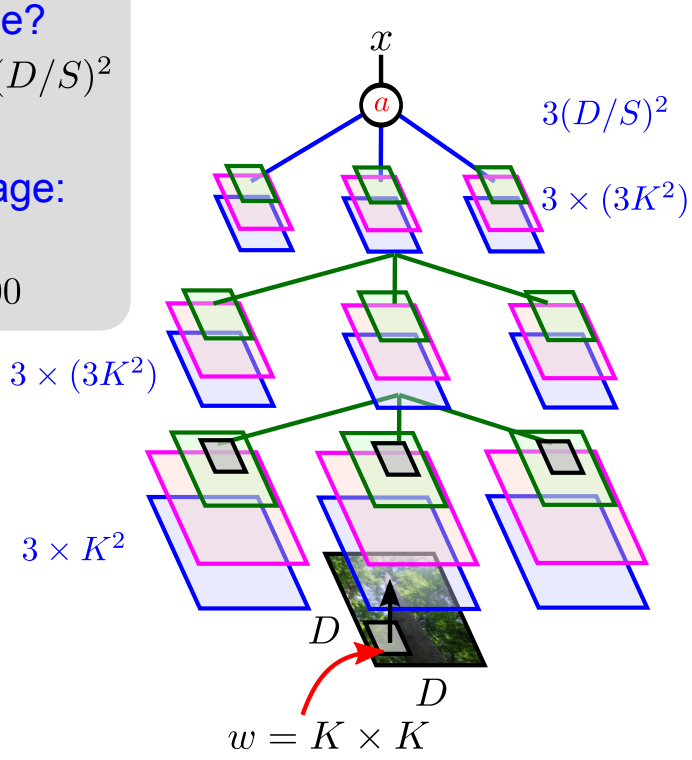
$$|\theta| = 3K^2 + 9K^2 + 9K^2 + 3(D/S)^2$$
$$= 21K^2 + D$$

For a small 32 by 32 image:

$$K = 5 \qquad S = 2$$
$$|\theta| = 21 \times 5^2 + 4^2 \approx 600$$



$$|\theta| = 3D^2 + D \approx 3 \times 10^6$$



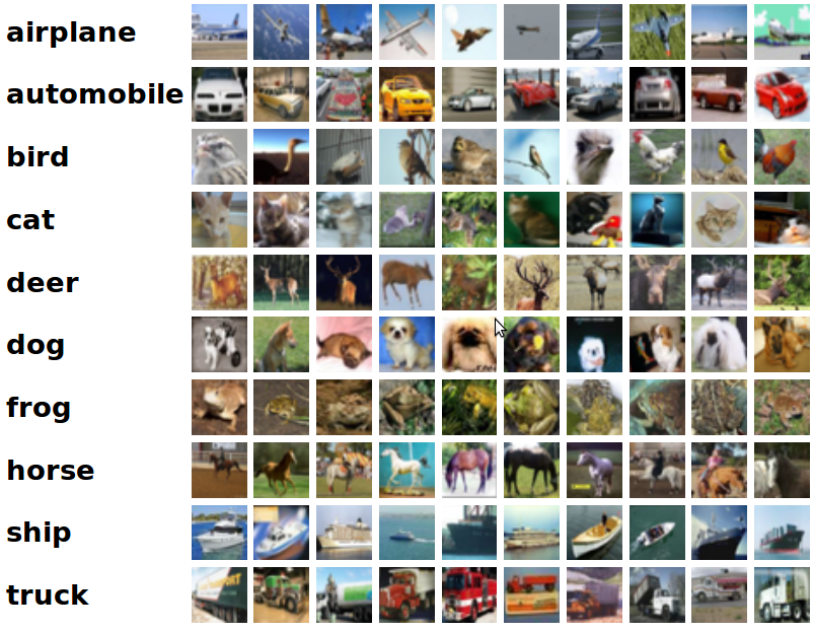
Training

- **back-propagation for training**: stochastic gradient ascent
  - like last lecture output interpreted as a class label probability, $x = p(t = 1|\boldsymbol{z})$
  - now x is a more complex function of the inputs $\boldsymbol{z}$
  - can optimise same objective function computed over a mini-batch of datapoints

- **data-augmentation**: always improves performance substantially (include shifted, rotations, mirroring, locally distorted versions of the training data)

- **typical numbers**:
  - 5 convolutional layers, 3 layers in top neural network
  - 500,000 neurons
  - 50,000,000 parameters
  - 1 week to train (GPUs)

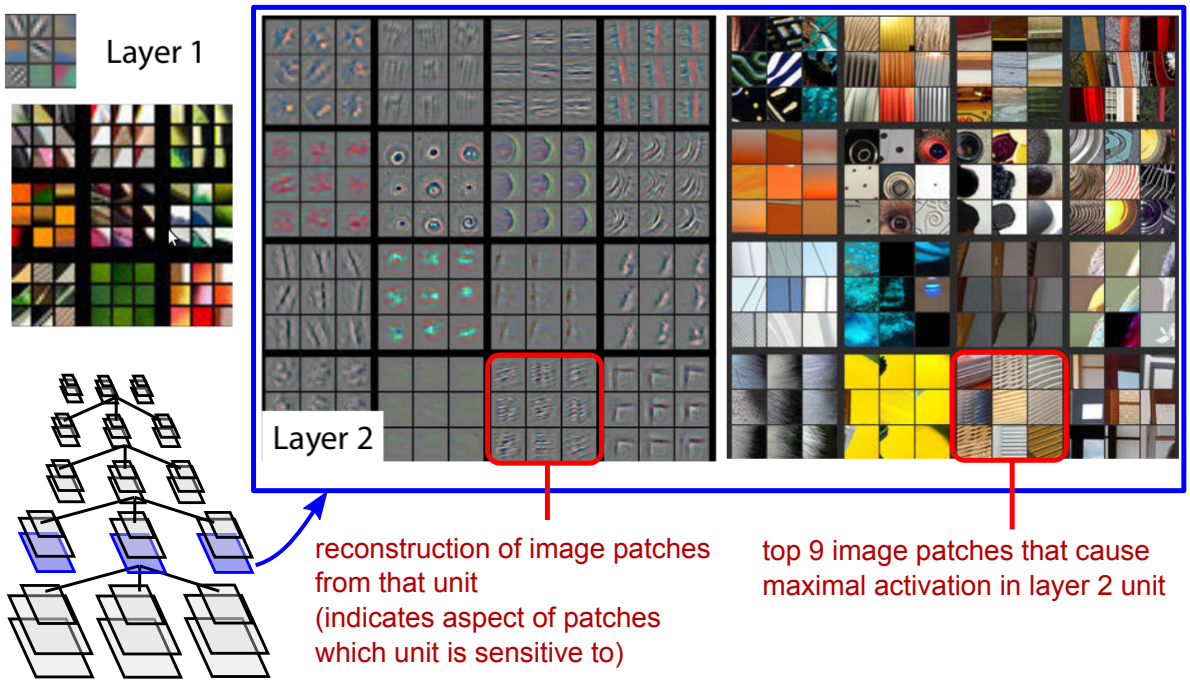
Demo



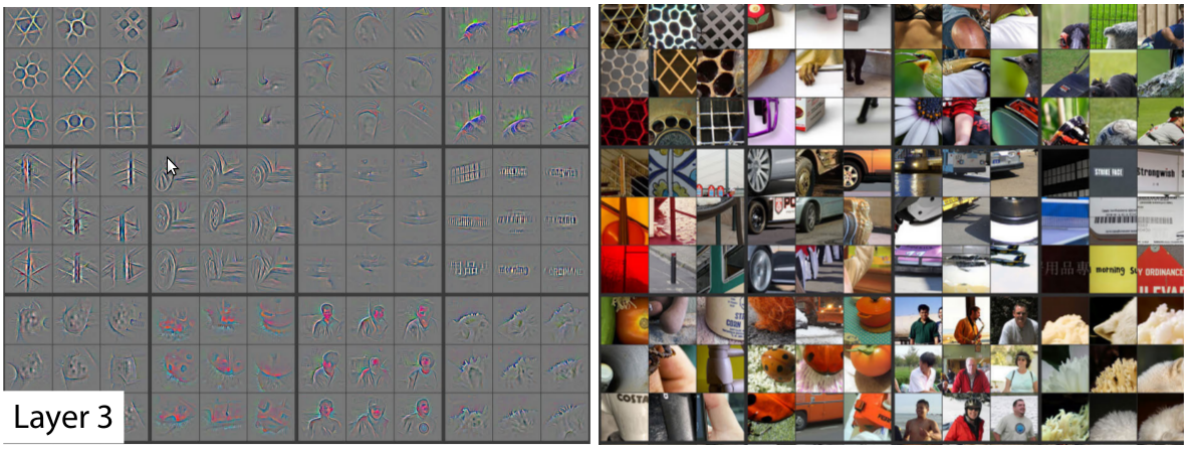
CIFAR 10 dataset: 50,000 training images, 10,000 test images

`http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`

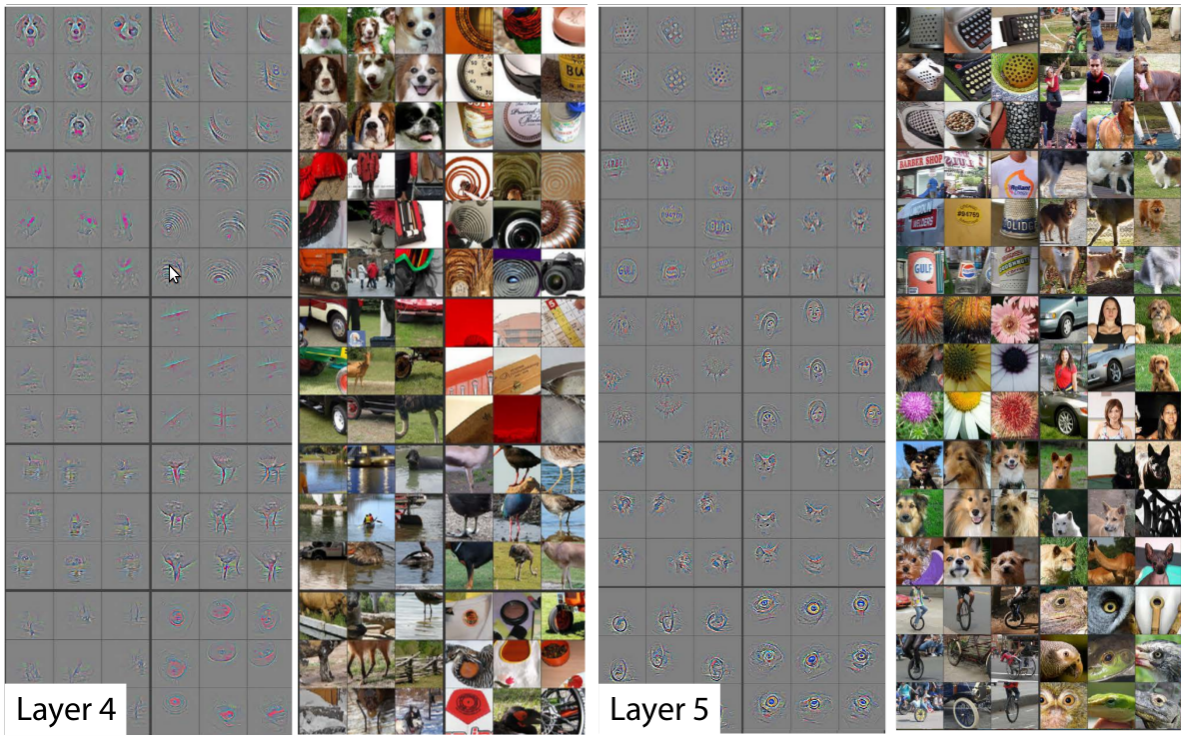
Looking into a convolutional neural network's brain



Looking into a convolutional neural network's brain



Looking into a convolutional neural network's brain



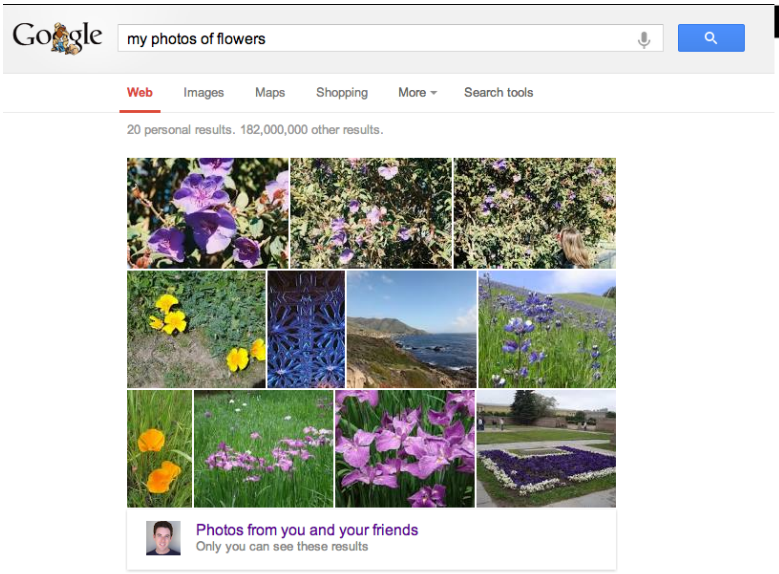
Summary

- higher level layers encode more **abstract features**
- higher level layers show more **invariance to instantiation parameters**
  - translation
  - rotation
  - lighting changes
- a method for **learning feature detectors**
  - first layer learns edge detectors
  - subsequent layers more complex
  - integrates training of the classifier with training of the featural representation

Convolutional neural networks in the news

- convolutional neural networks are the go-to model for many computer vision classification problems

- form of neural network with an architecture suited to vision problems



Finally some cautionary words

- hierarchical modelling is a **very old idea** and not new
- the ‘deep learning’ revolution has come about mainly due to new methods for initialising learning of neural networks
- current methods aim at invariance, but this is far from all there is to computer and biological vision: **e.g. instantiation parameters should also be represented**
- **classification can only go so far**: "tell us a story about what happened in this picture"