Lecture :
Neural Networks and Deep Learning

Riashat Islam

Reasoning and Learning Lab
McGill University

September 25, 2018

Deep Learning Hype

Forbes / Tech

MAR 29, 2016 @ 01:13 PM 2,357 VIEWS

How Deep Is Your Learning?

Artificial intelligence

Million-dollar babies

The Economist

As Silicon Valley fights for talent, universities struggle to hold on to their stars

Apr 2nd 2016 | SAN FRANCISCO | From the print edition

MACLEAN'S

The meaning of AlphaGo, the AI program that beat a Go champ

Geoffrey Hinton, the godfather of 'deep learning'—which helped Google's AlphaGo beat a grandmaster—on the past, present and future of AI

March 18, 2016

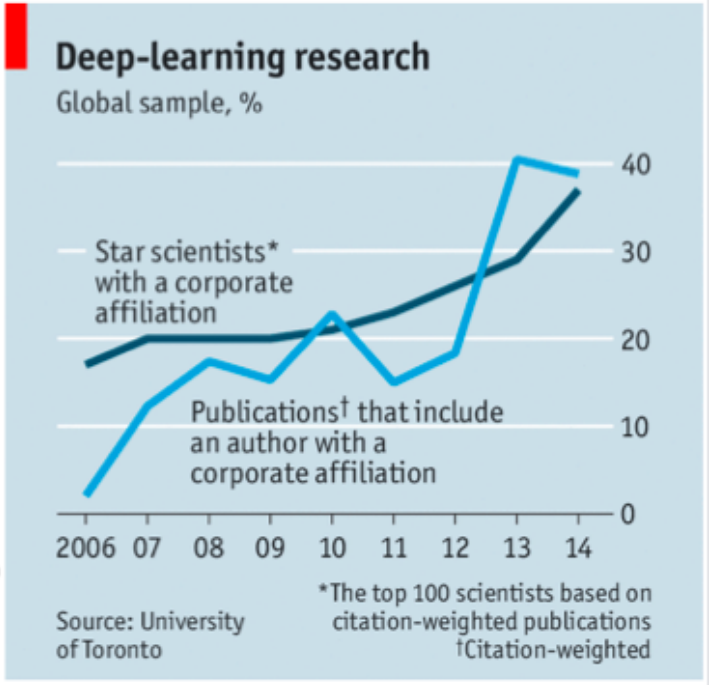
THE WALL STREET JOURNAL.

VENTURE CAPITAL DISPATCH

Hedge Fund Analysts Use Deep Learning To Diagnose Heart's Condition

Qi Liu and Tencia Lee, hedge fund analysts and self-described "quants," built the winning algorithm in a competition, which could find indicators of heart disease.

By CAT ZAKRZEWSKI
Mar 30, 2016 9:11 pm ET



What is Deep Learning

Traditional Machine Learning Flow

Deep Learning Flow

Deep Learning in Computer Vision

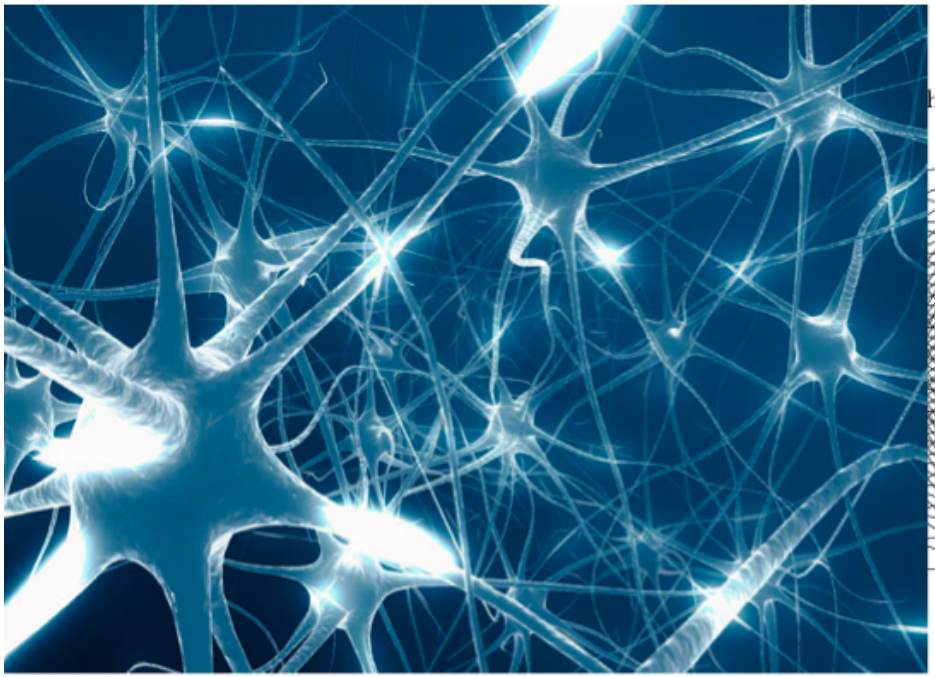
- Main Challenge : the information of interest is tangled with many nuisance variables in a high dimensional space
- Computer Vision is all about**data representation**
  - ie, extracting useful information with minimal loss while being variant to nuisance variables
  - What's useful or a nuisance depends on what we're doing - the representation is **task-dependent**

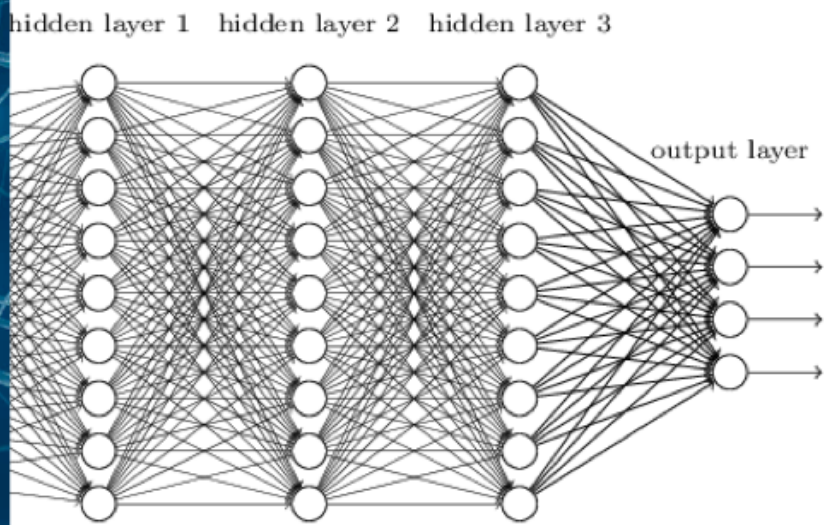
Why Deep Learning

Many strengths of Deep Learning :

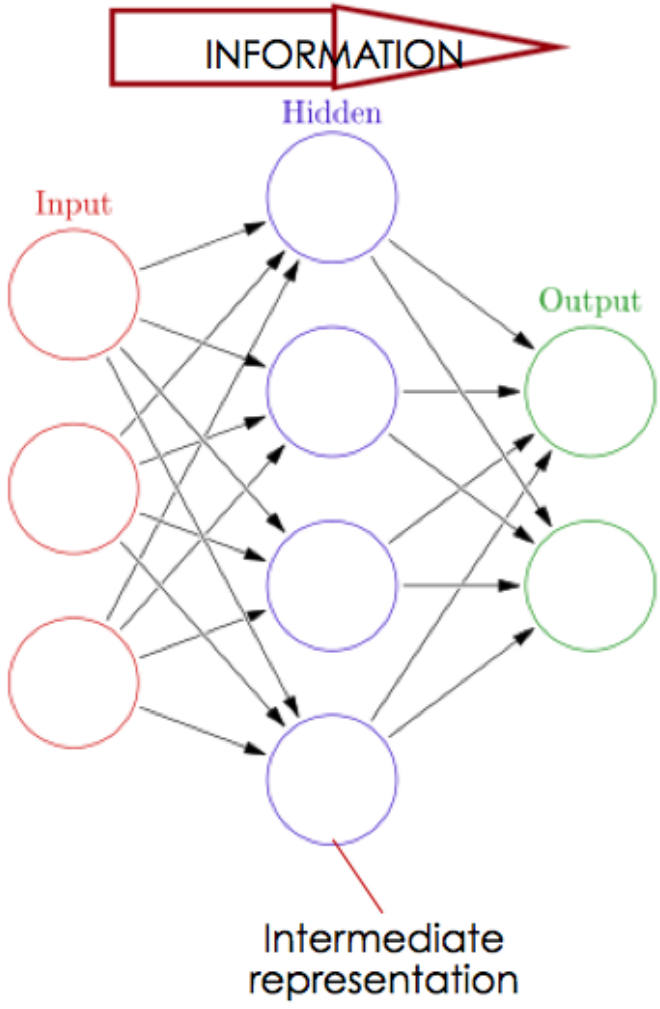
- Arbitrary number of intermediate representations
  - This is what makes it deep
  - Features are learned
  - Features increase in scale and/or abstraction with depth
- **End-to-end training**
  - Jointly optimize all intermediate representations for a given task

Neural Networks (Biological vs Artifical)





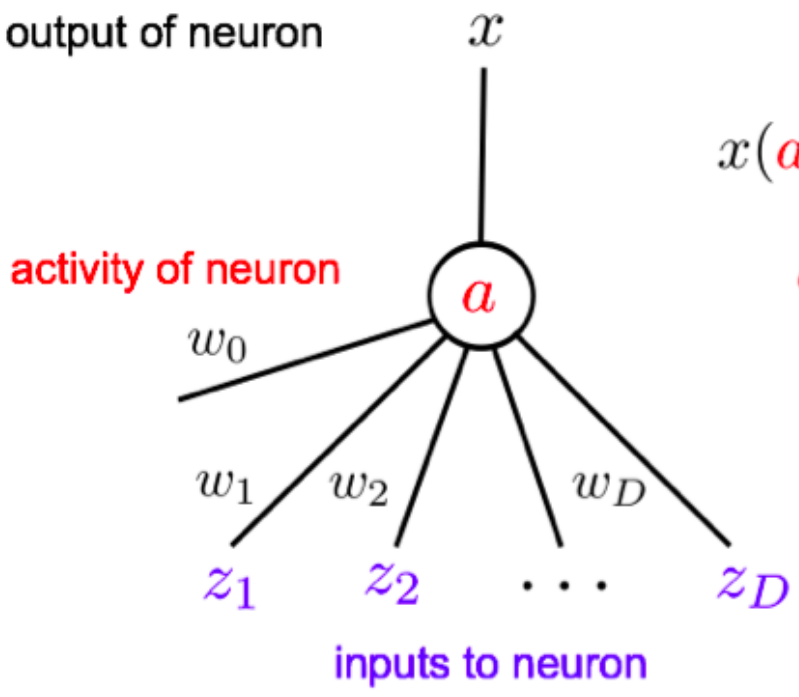
Neural Networks

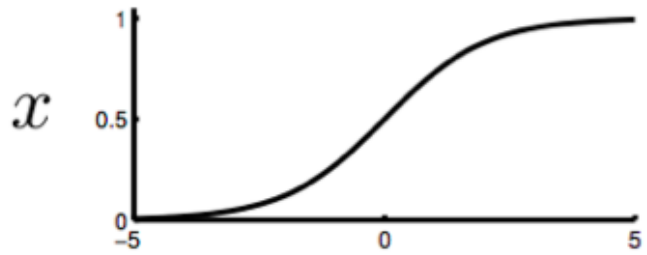


- Biologically-inspired highly-parallel model
- A network of nodes ("neurons") with weighted connections ("synapses")
- Each node outputs a non-linear activation function applied to a weighted sum of its inputs
- The network is trained by adjusting these weights

Let's understand Neural Networks

A Single Neuron

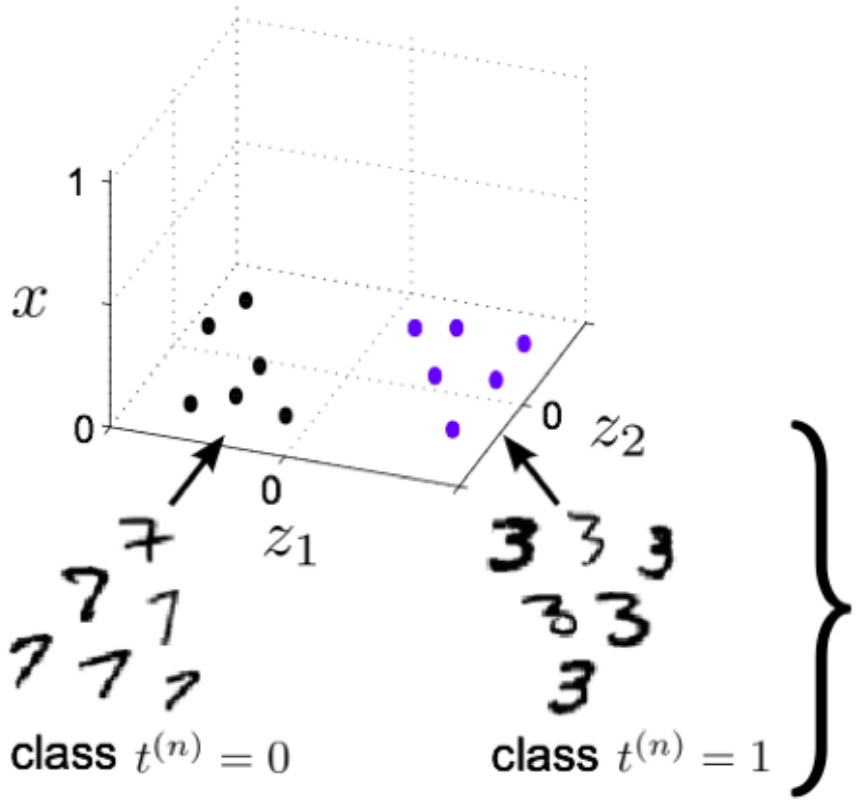




$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

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

Training a Single Neuron

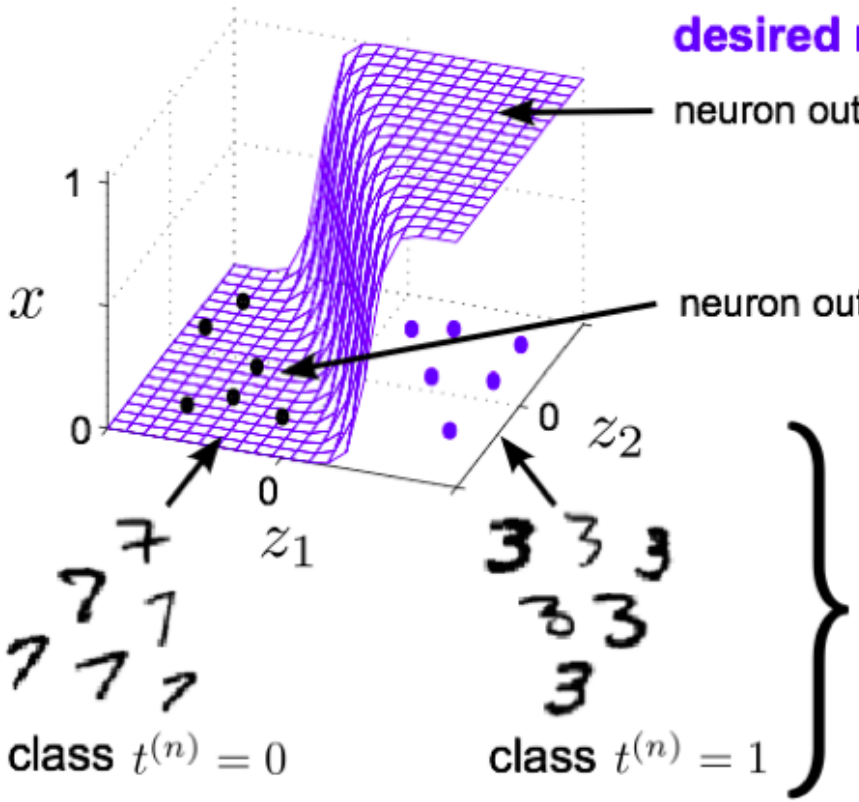


training data

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

inputs class labels

Training a Single Neuron



desired result of training:

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

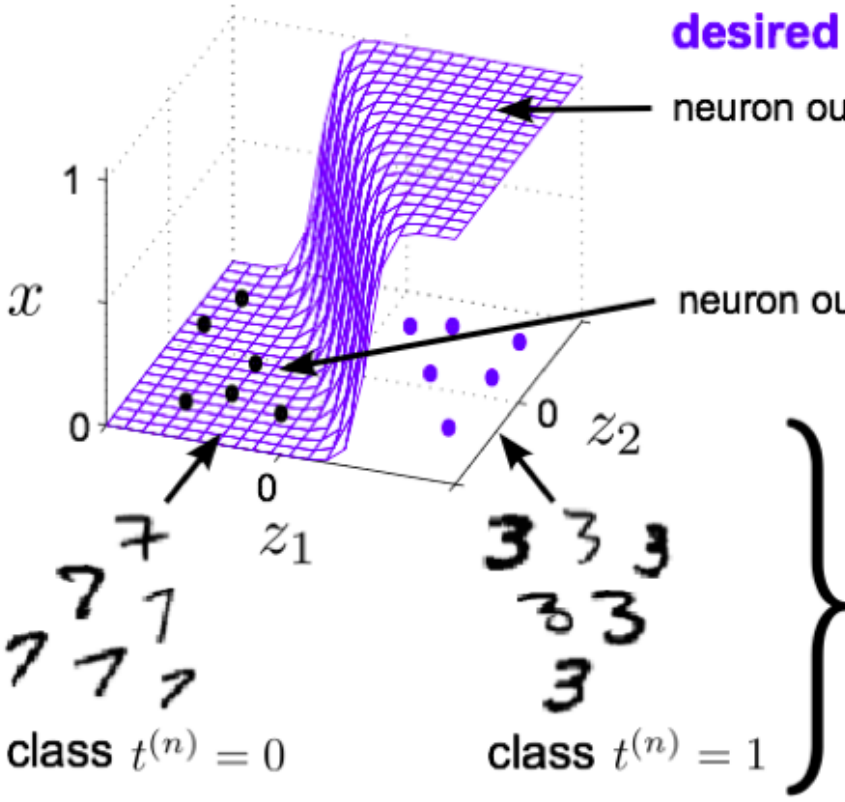
neuron outputs $\mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) \approx 0$ for $t^{(n)} = 0$

training data

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

inputs class labels

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

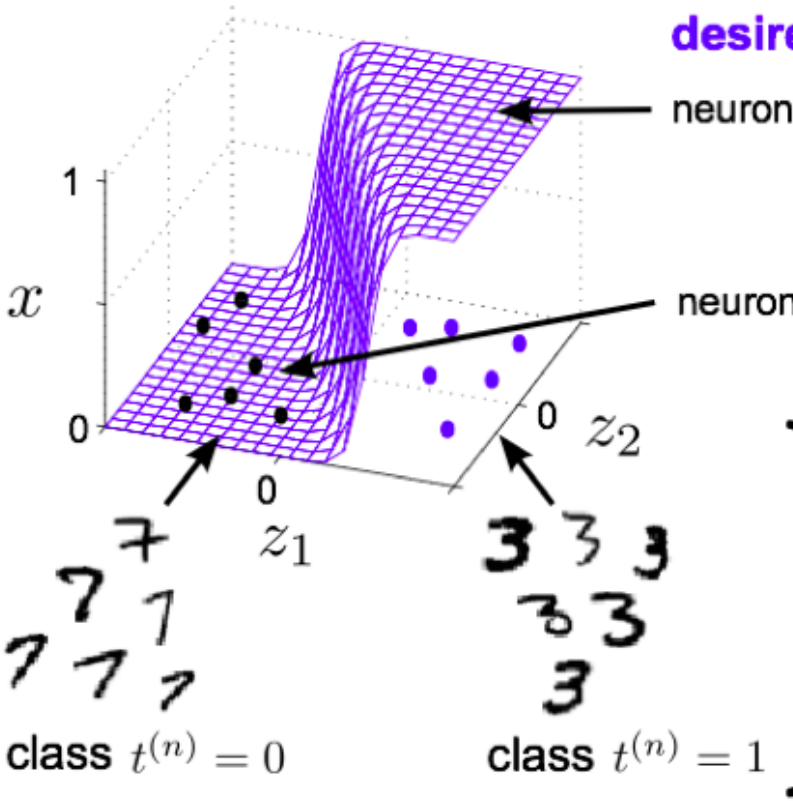
$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

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:

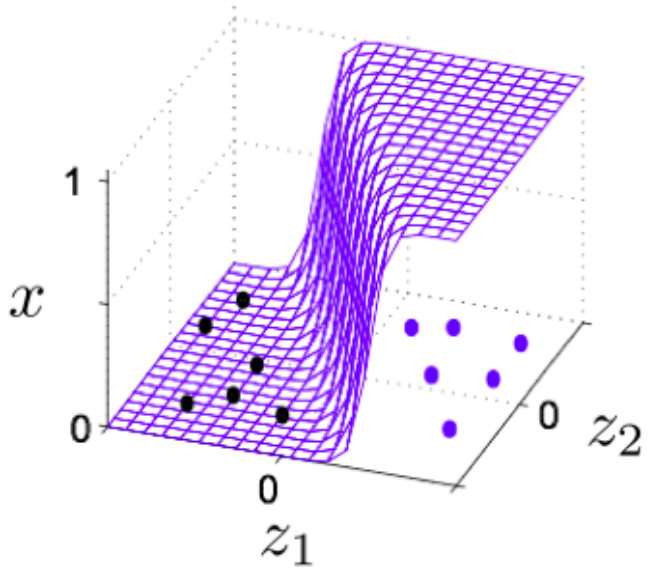
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

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

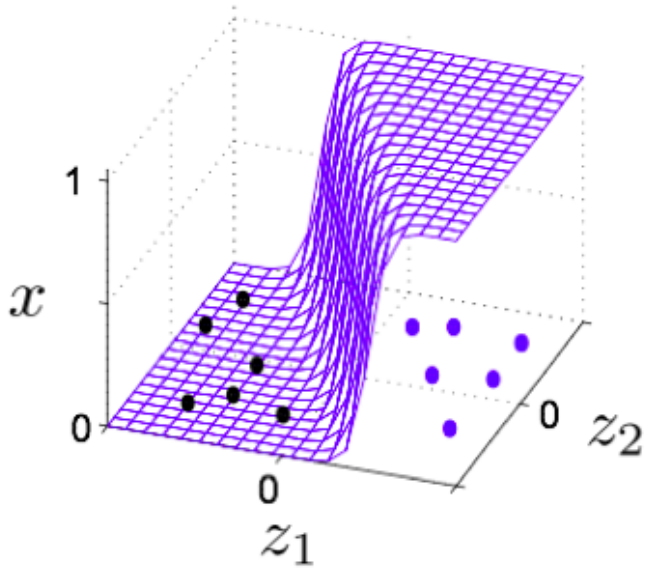
objective function:

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

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

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:

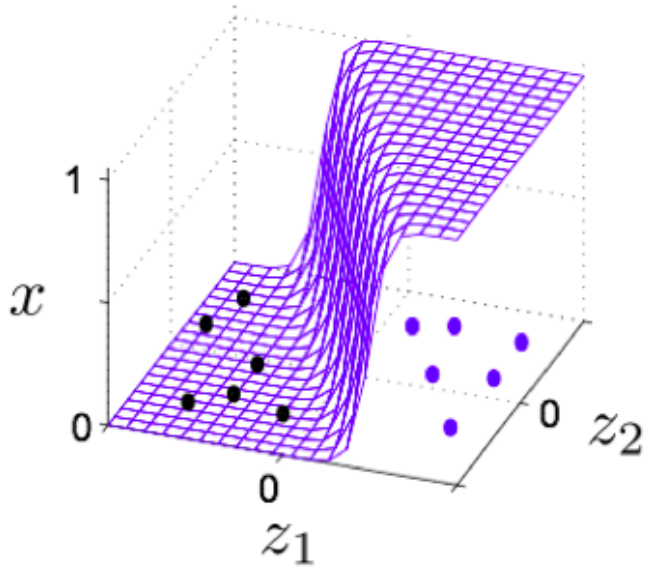
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right] \geq 0$$

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

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

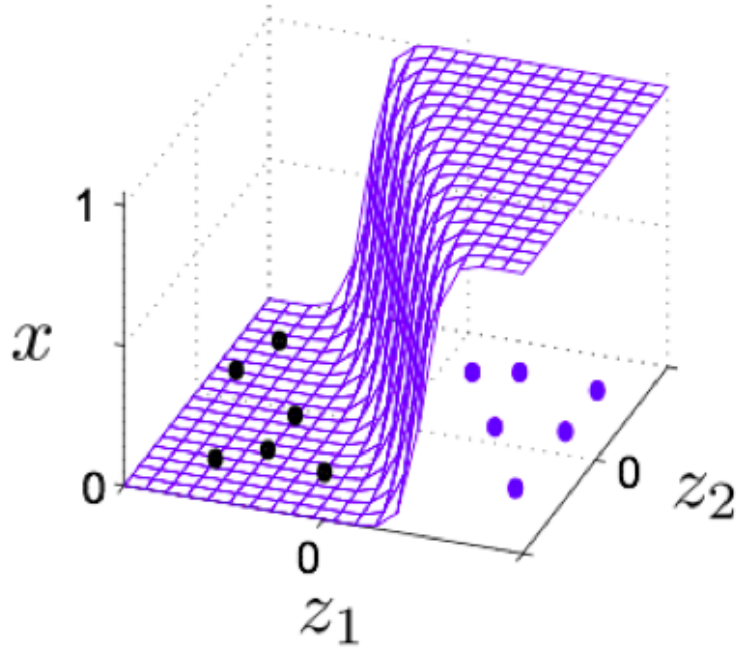
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} G(\boldsymbol{w}) = \sum_n \frac{\mathrm{d}G(\boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}\boldsymbol{w}} = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)}$$

= 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)

Overfitting and Weight Decay



training data

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

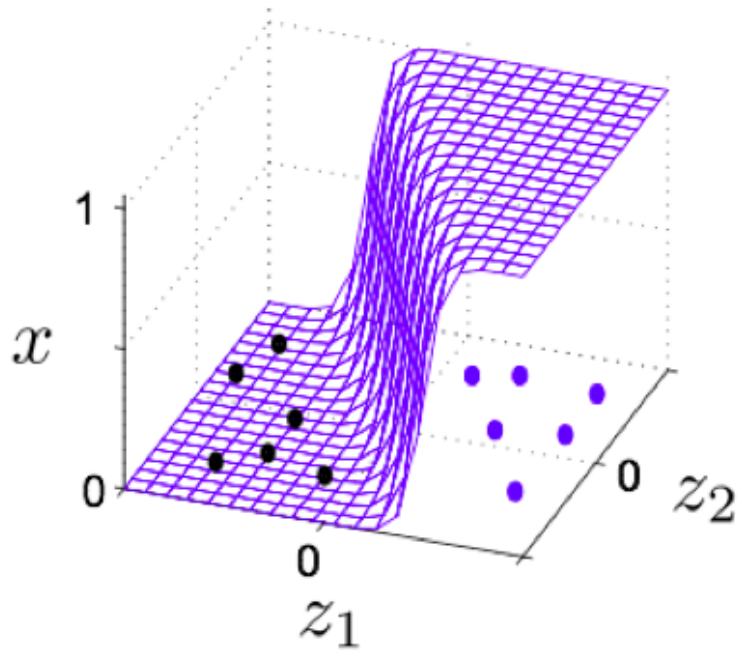
inputs class labels

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

Overfitting and Weight Decay



training data

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

inputs class labels

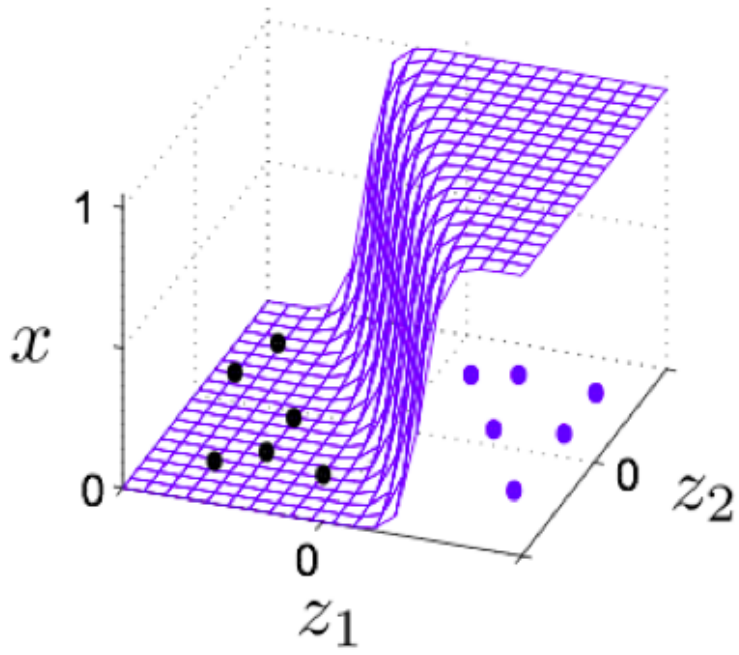
objective function:

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

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

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

Overfitting and Weight Decay



training data

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

inputs class labels

objective function:

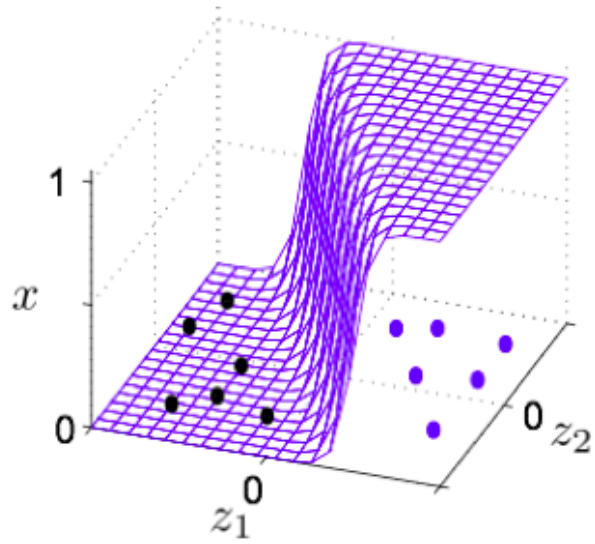
$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]$$

$$E(\boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2$$ 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 Single Neuron



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

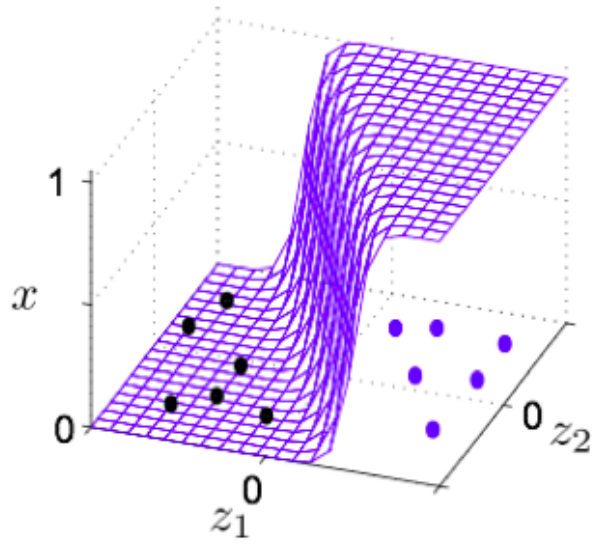
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

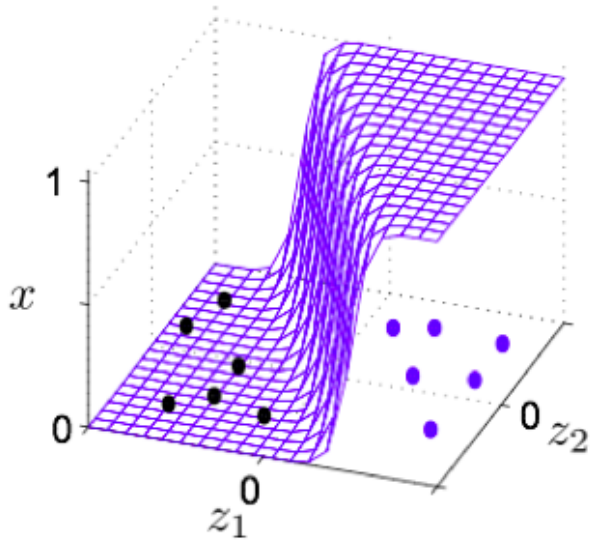
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

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

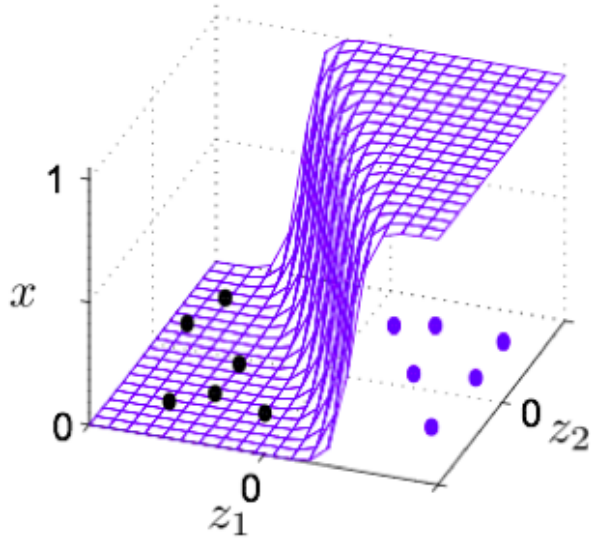
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

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

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

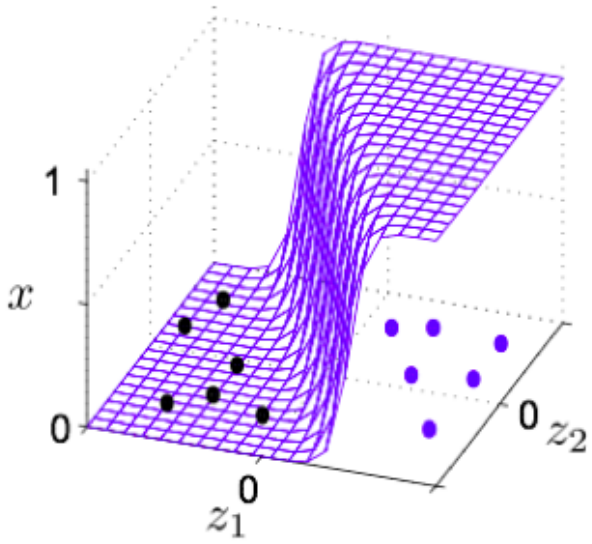
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

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

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

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

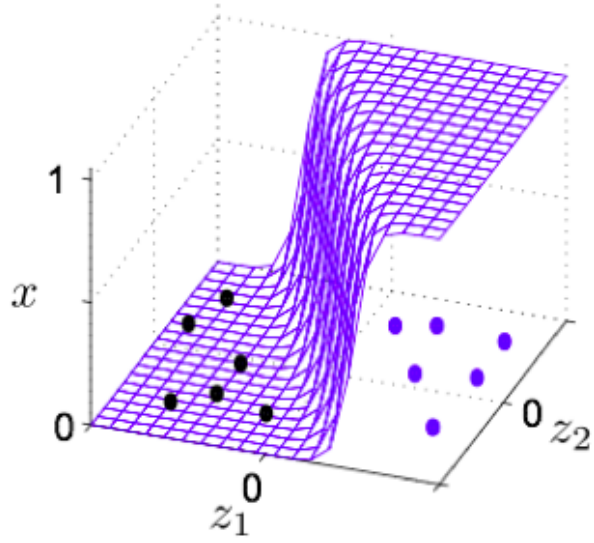
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

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

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

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

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

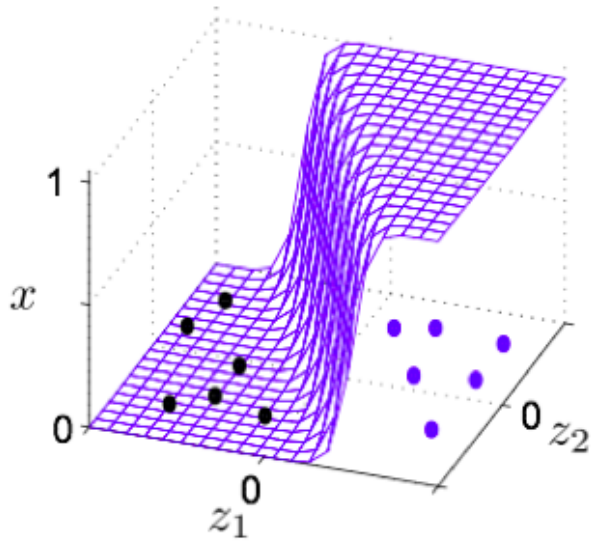
single neuron training:

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

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

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

Probabilistic Interpretation of Single Neuron



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

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

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

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

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

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

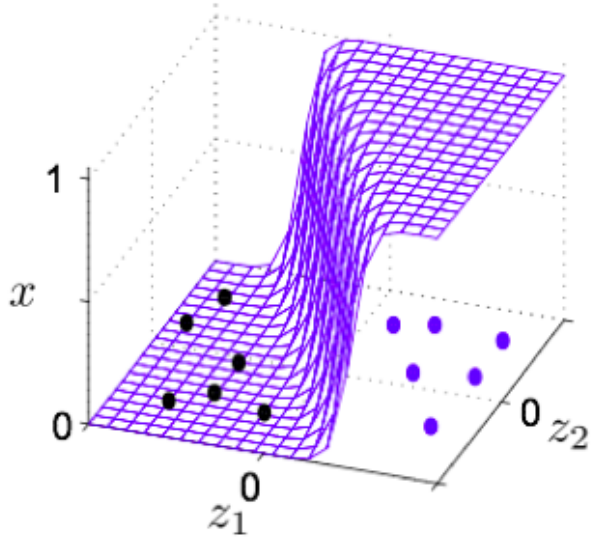
single neuron training:

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

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

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

Probabilistic Interpretation of 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**

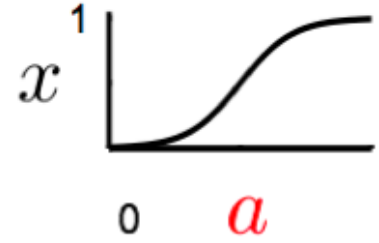
single neuron training:

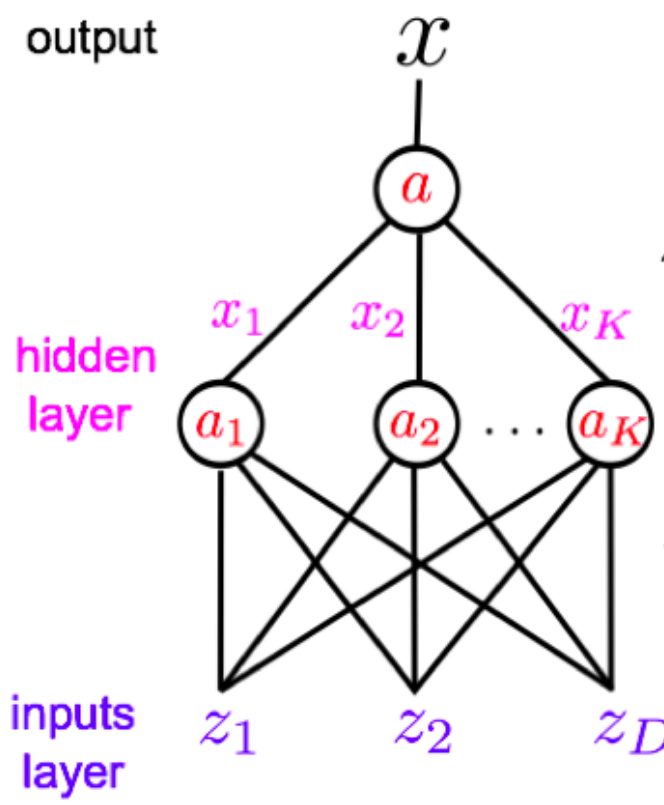
$\{\boldsymbol{z}^{(n)}\}_{n=1}^N \qquad D = \{t^{(n)}\}_{n=1}^N$

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

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

Single Hidden Layer Neural Networks





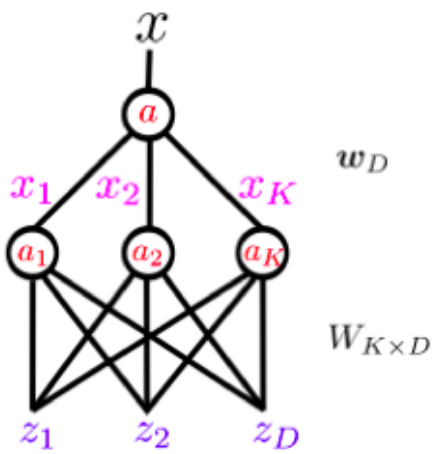
$$x(a) = \frac{1}{1+\exp(-a)}$$

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

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

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

Single Hidden Layer Neural Networks



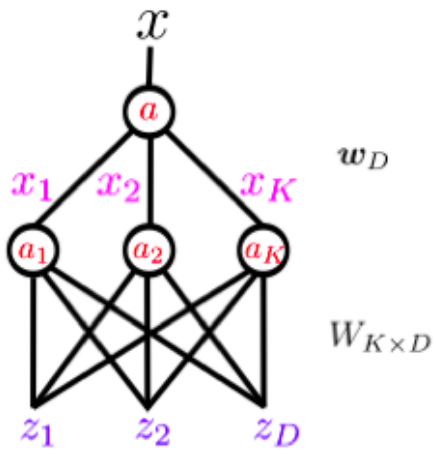
$$x(a) = \frac{1}{1+\exp(-a)}$$

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

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

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

Single Hidden Layer Neural Networks



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

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

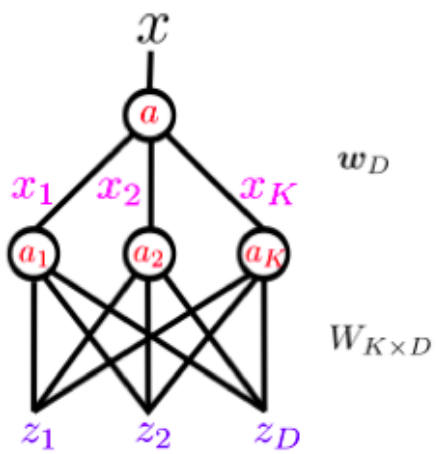
$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

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

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

Single Hidden Layer Neural Networks



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

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

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

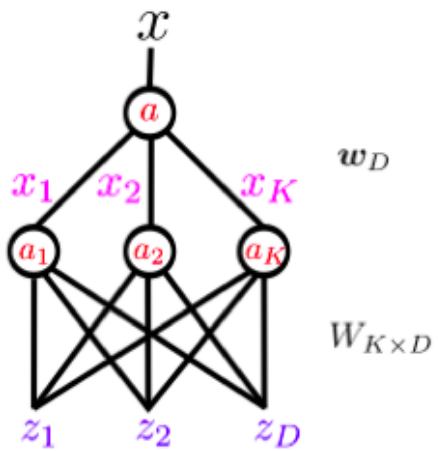
$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

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

Single Hidden Layer Neural Networks



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

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

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

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

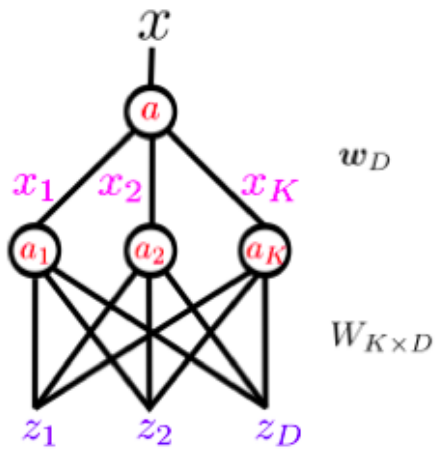
objective function:

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

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

Single Hidden Layer Neural Networks



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

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

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

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

objective function:

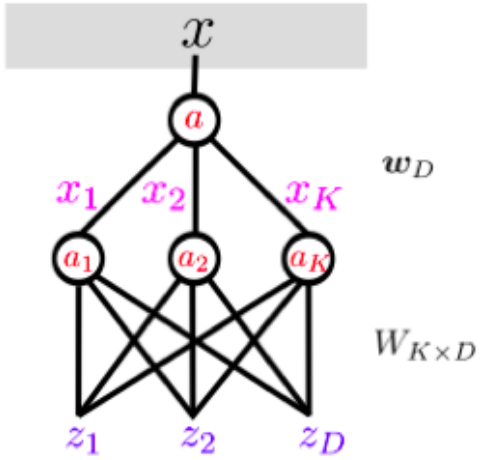
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

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

Single Hidden Layer Neural Networks



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

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

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

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

objective function:

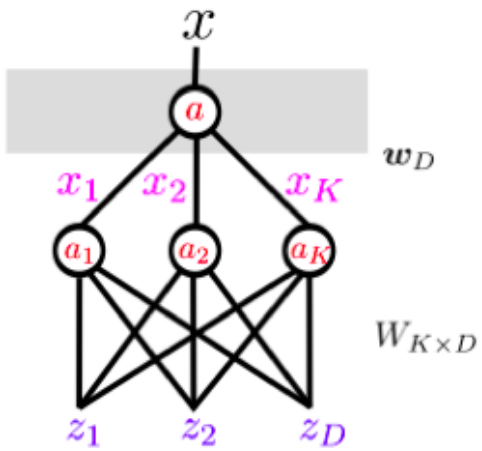
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1 - t^n) \log \left(1 - \mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

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

Single Hidden Layer Neural Networks



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

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

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

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

objective function:

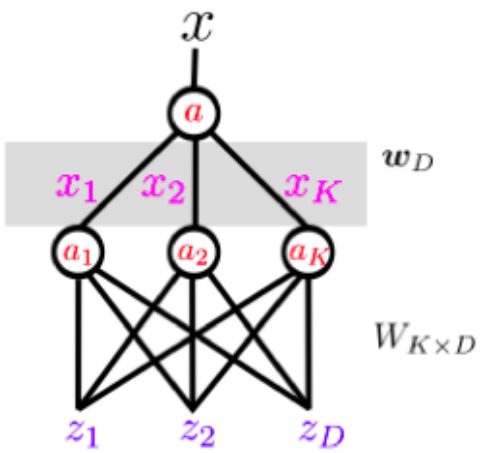
$$G(W, \boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1-t^n) \log\left(1-\mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

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

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

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

Single Hidden Layer Neural Networks



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

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

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

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

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

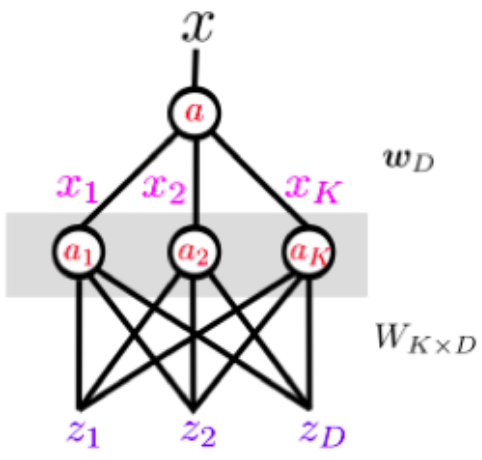
$$E(W, \boldsymbol{w}) = \tfrac{1}{2} \sum_i w_i^2 + \tfrac{1}{2} \sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

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

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

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

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

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

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

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

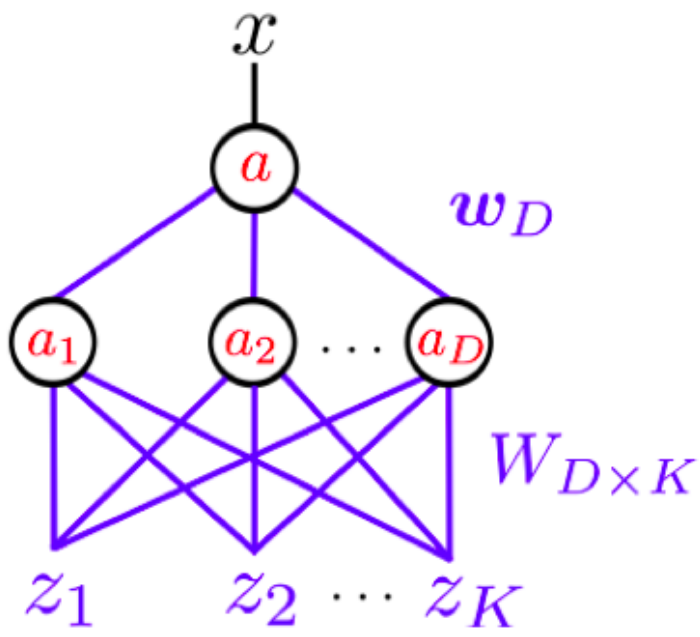
$$= \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}} \frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}} \frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W, \boldsymbol{w})}{\mathrm{d}x^{(n)}} \frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}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

Hierarchical Models with many hidden layers

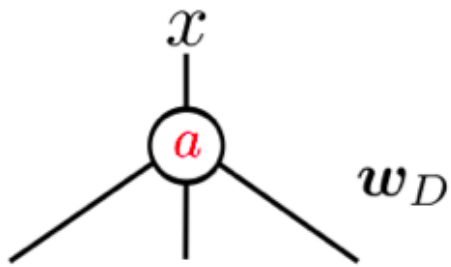
- **deep neural networks** have been a **longstanding goal of AI**
- initial attempts in the '90s and '80s fell into **poor local optima**
- resurgence of interest in neural networks: result of **better initialisation methods**
  - **method 1**: unsupervised pre-training (e.g. using a restricted Boltzmann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

Training multi-layer neural networks: Layerwise

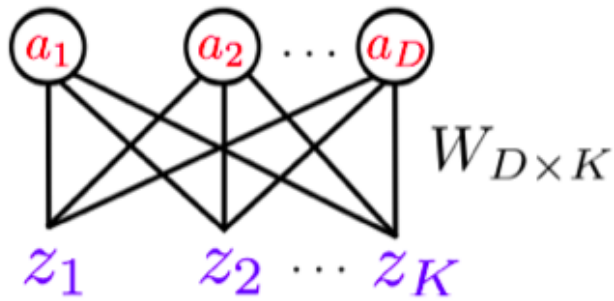
train input and output weights of two layer network using backprop



Training multi-layer neural networks : Layerwise

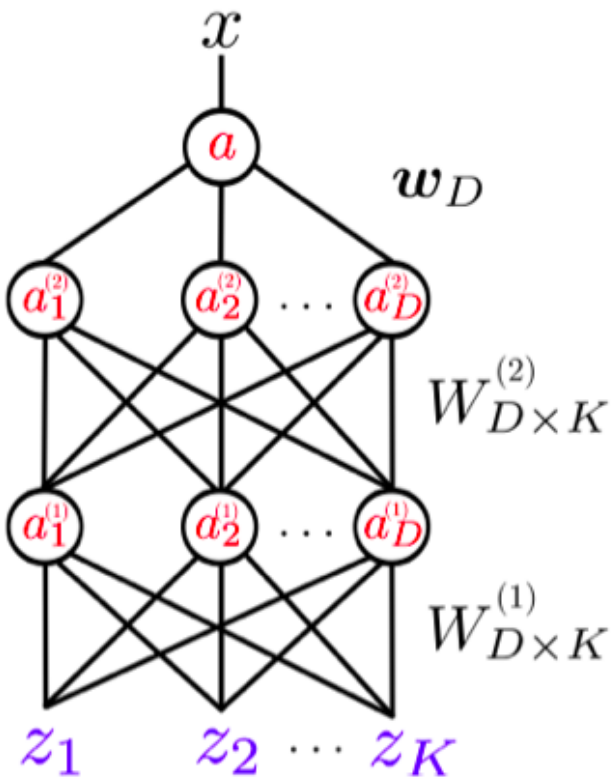


break apart network

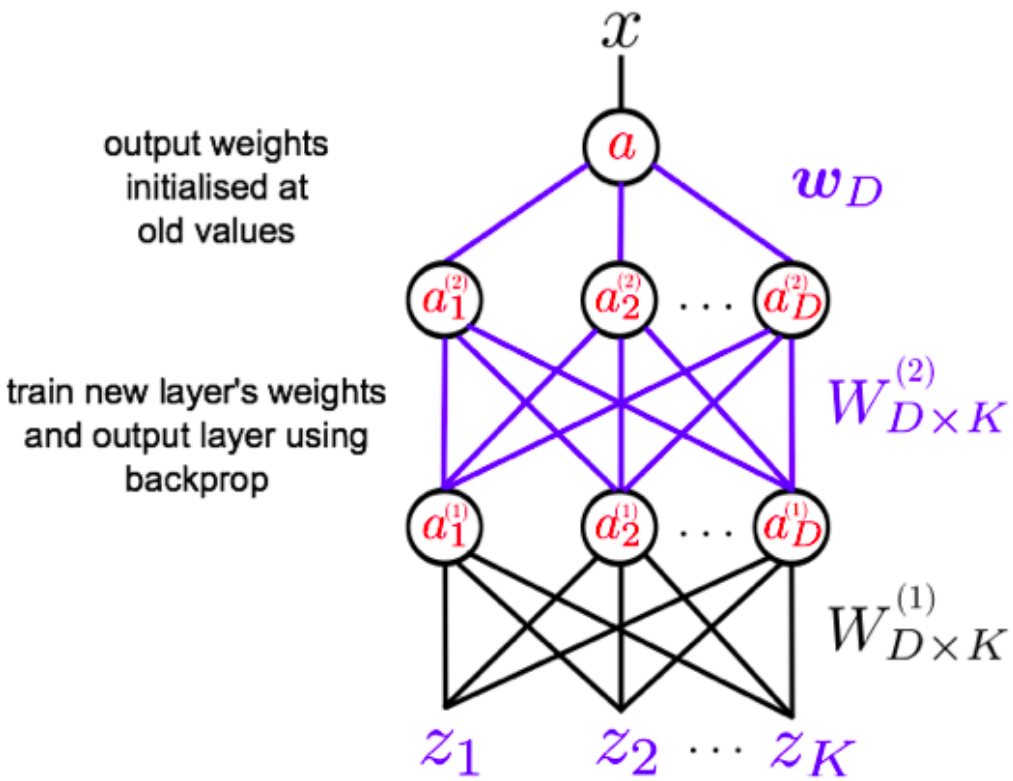


Training multi-layer neural networks : Layerwise

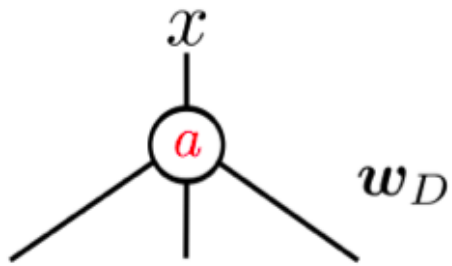
insert new layer into network



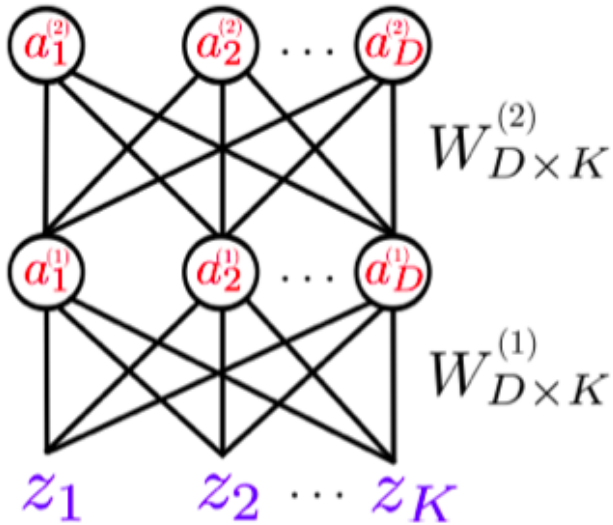
Training multi-layer neural networks : Layerwise



Training multi-layer neural networks : Layerwise

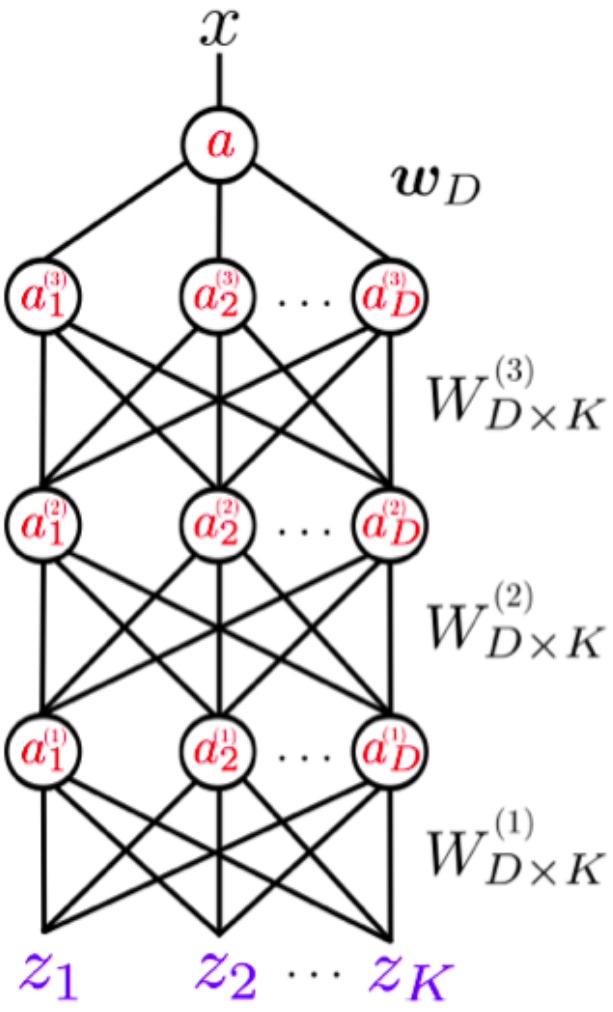


break apart network

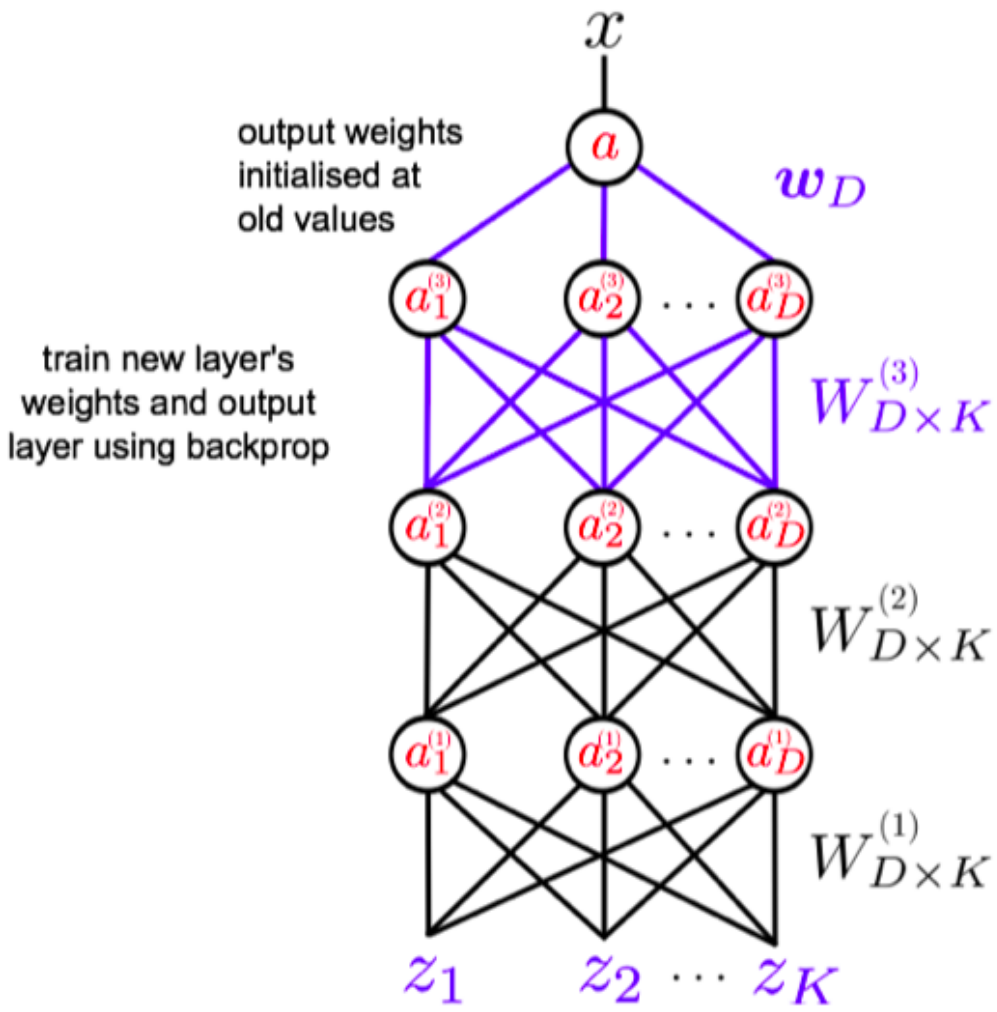


Training multi-layer neural networks : Layerwise

insert new layer into network

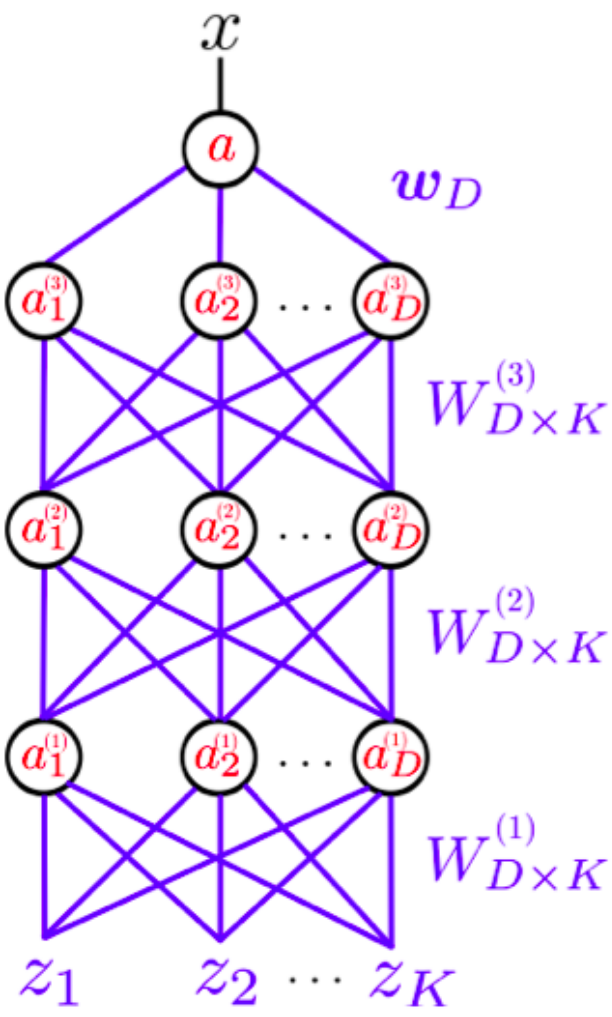


Training multi-layer neural networks : Layerwise



Training multi-layer neural networks : Layerwise

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

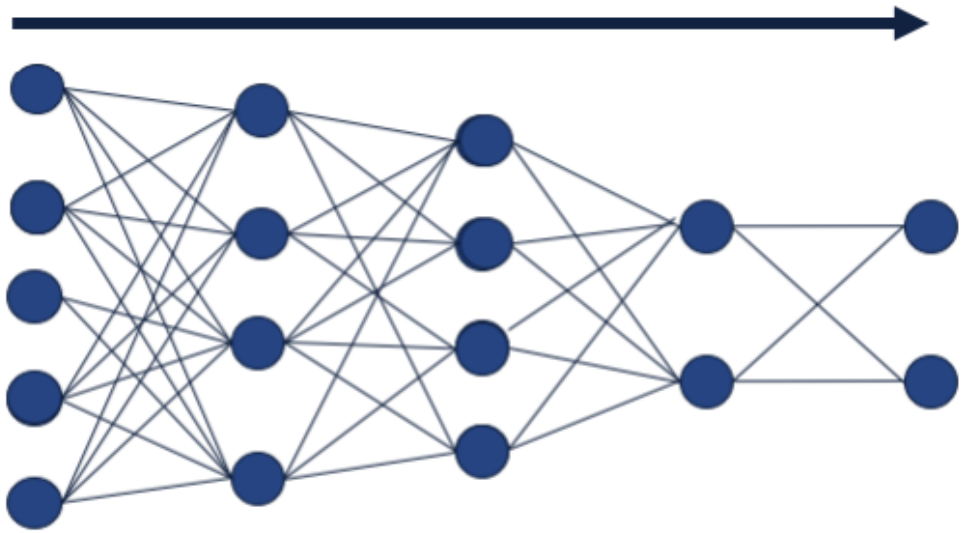


Summary

- 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/`

More Hidden Layers



input h_1 h_2 h_3 output

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

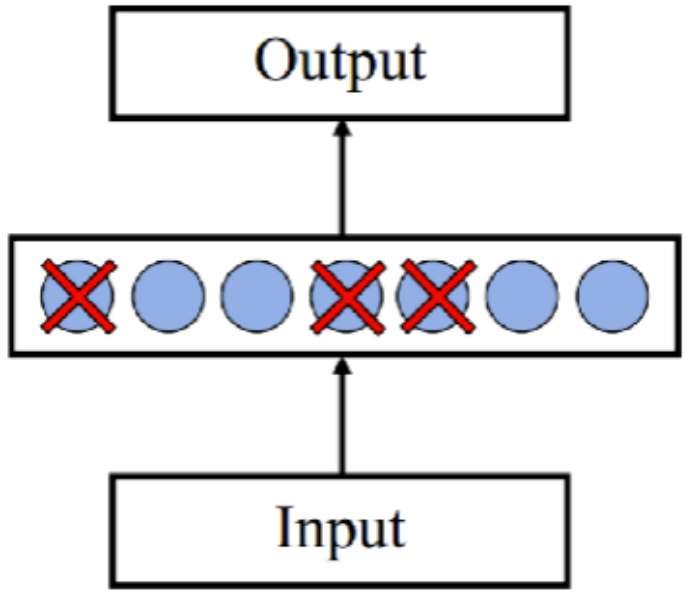
- No guarantee of convergence, may oscillate or reach a local minima
- In practice, many large networks can be trained on large amounts of data for realistic problems
- Termination criteria : Number of epochs; Threshold on training set error; No decrease in error; Increase error on a validation set
- 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.

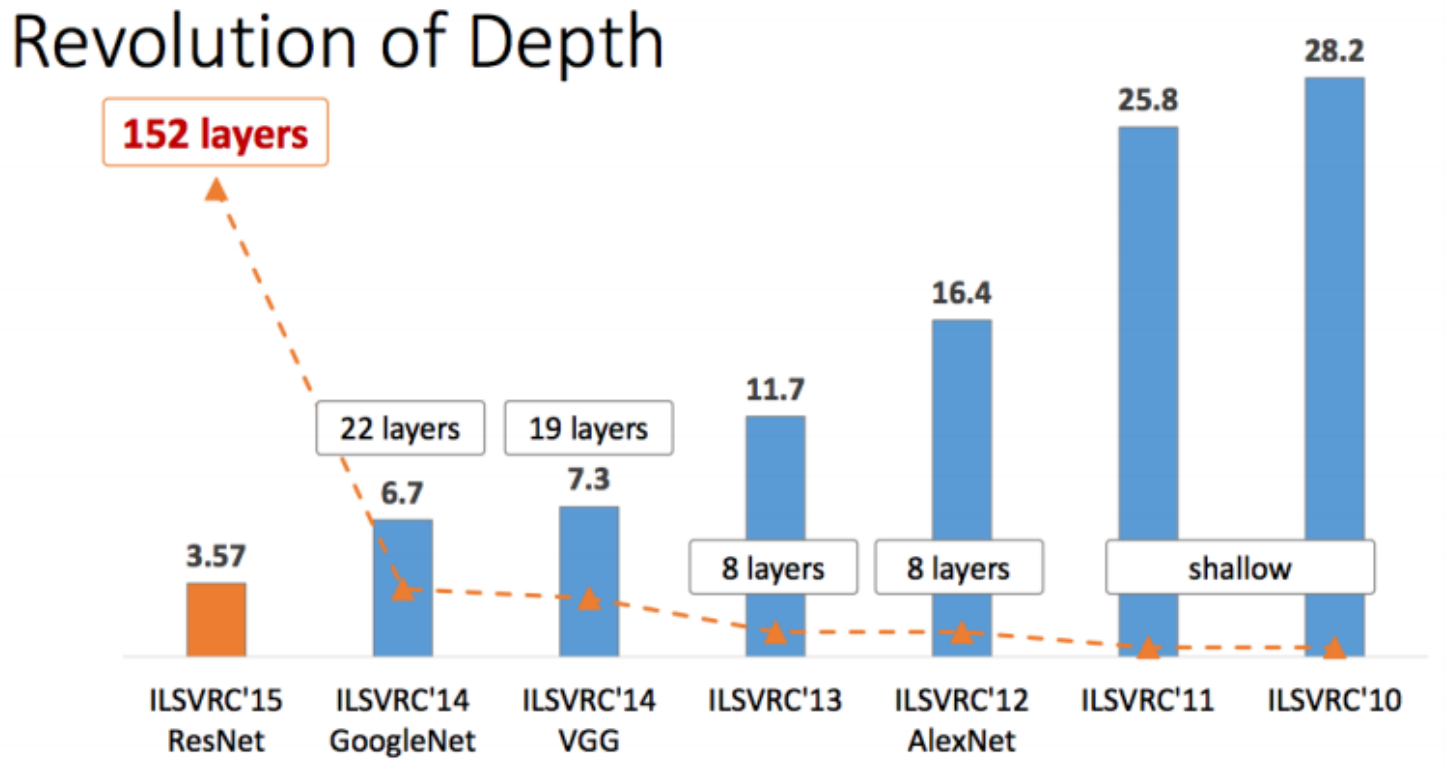
Dropout Training

Proposed by Hinton et al 2012



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

Deep Learning

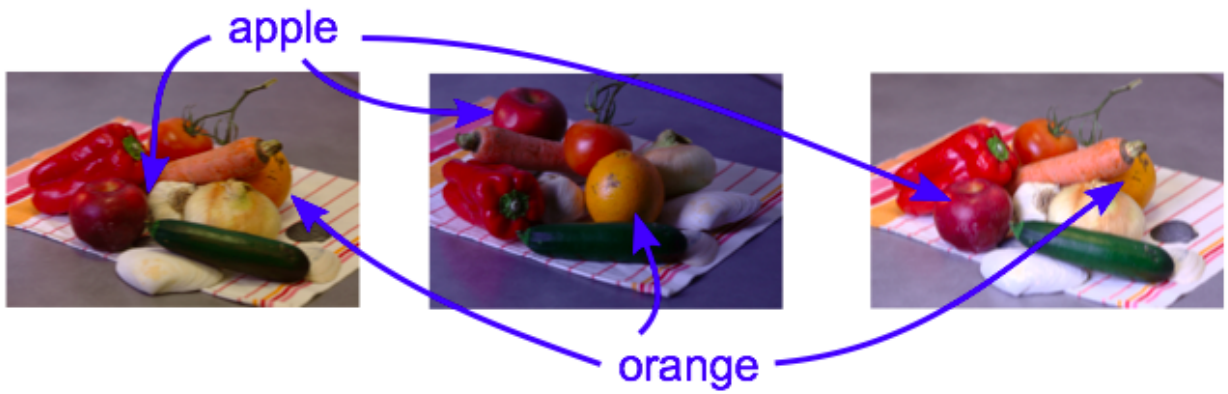


Object Recognition Performance on Imagenet

Convolutional Neural Networks

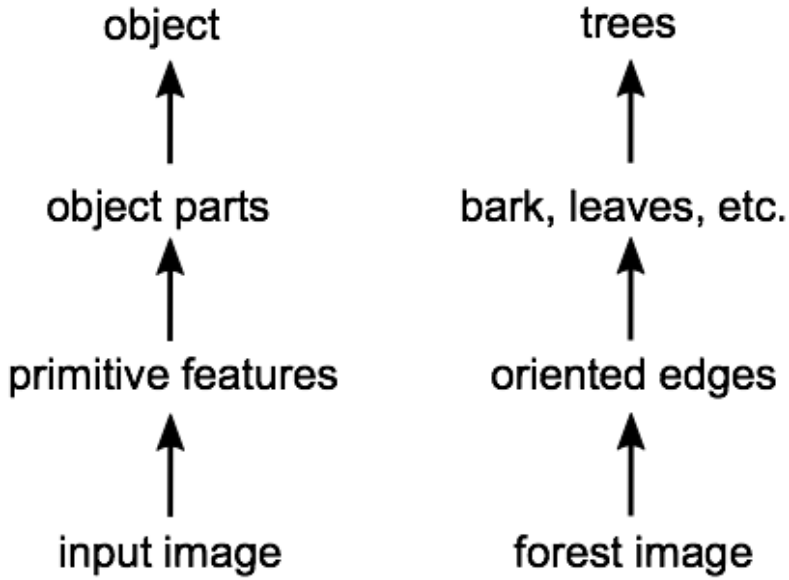
Big Picture

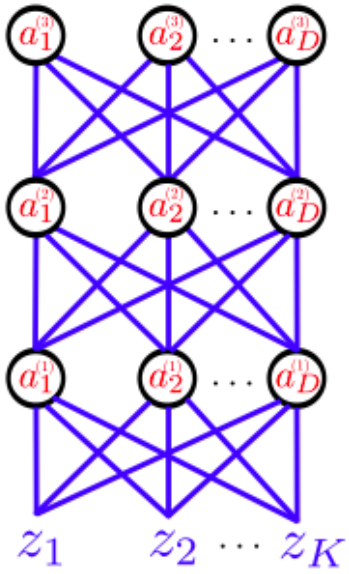
- **Goal**: how to produce good internal representations of the visual world to support recognition...
  - detect and classify objects into categories, independently of pose, scale, illumination, conformation, occlusion and clutter
- how could an artificial vision system learn appropriate internal representations 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

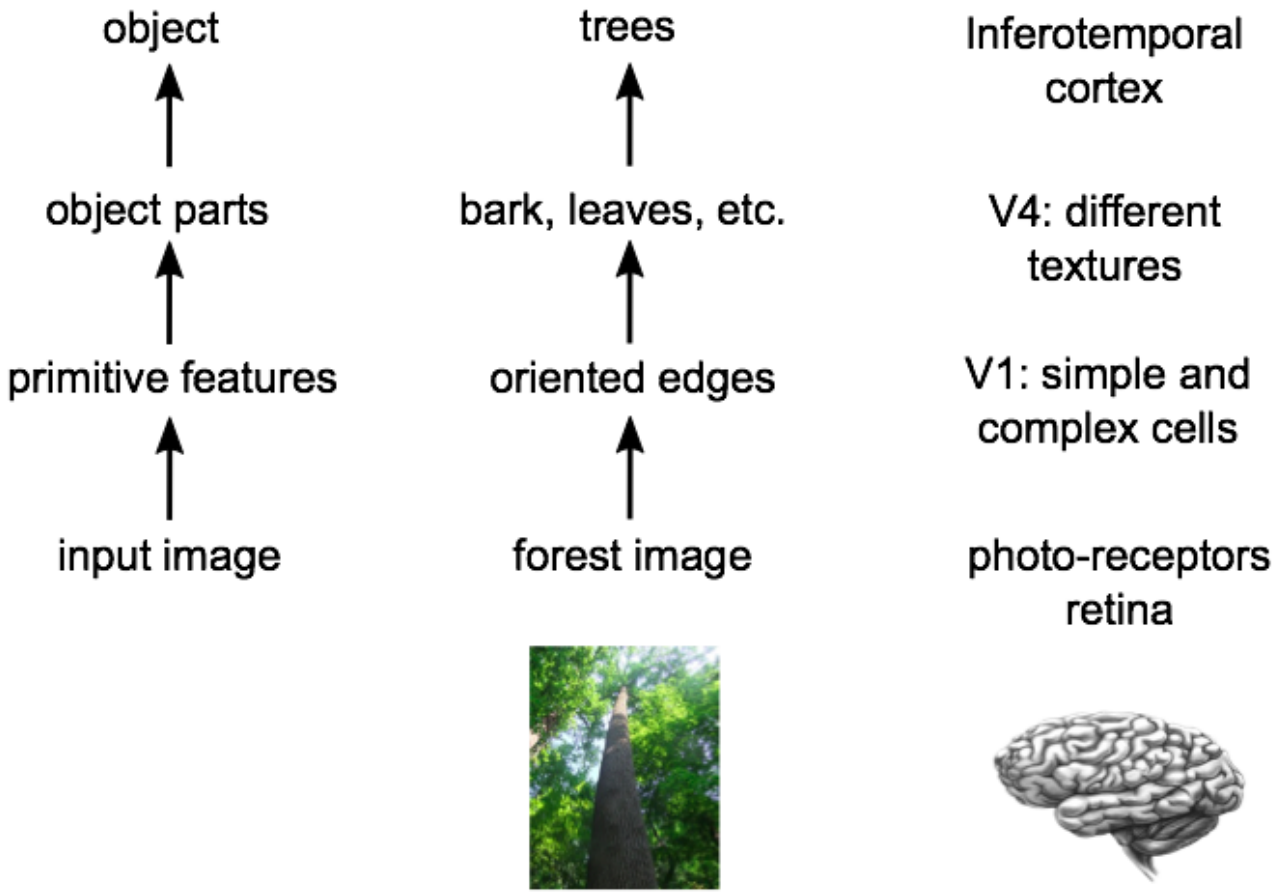
Argument 1: visual scenes are hierachically organised





Why use Hierarchical Multi-Layered Models

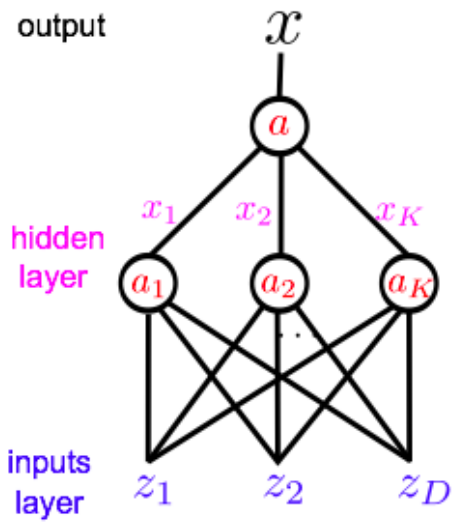
Argument 2: biological vision is hierachically organised



Why use Hierarchical Multi-Layered Models

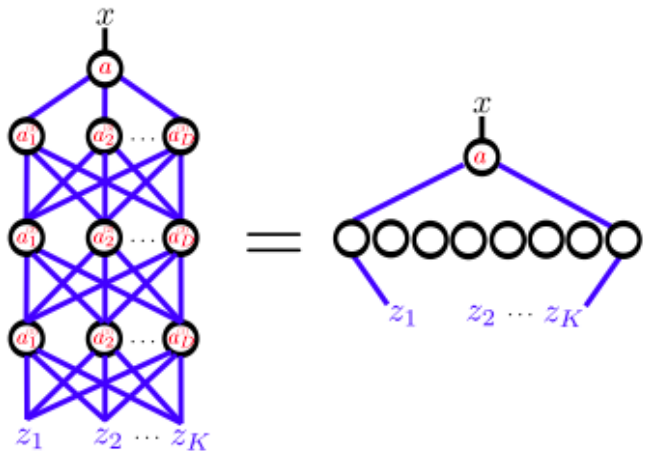
Argument 3: shallow architectures are inefficient at representing deep functions

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

Why not 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

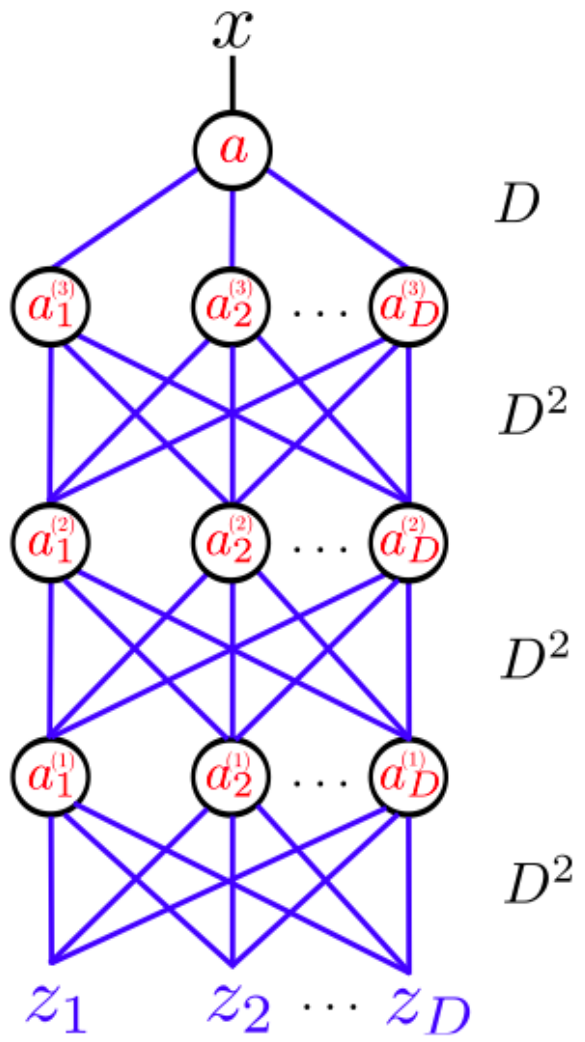
over-fitting and local optima

Need to initialise carefully

layer wise training

unsupervised schemes

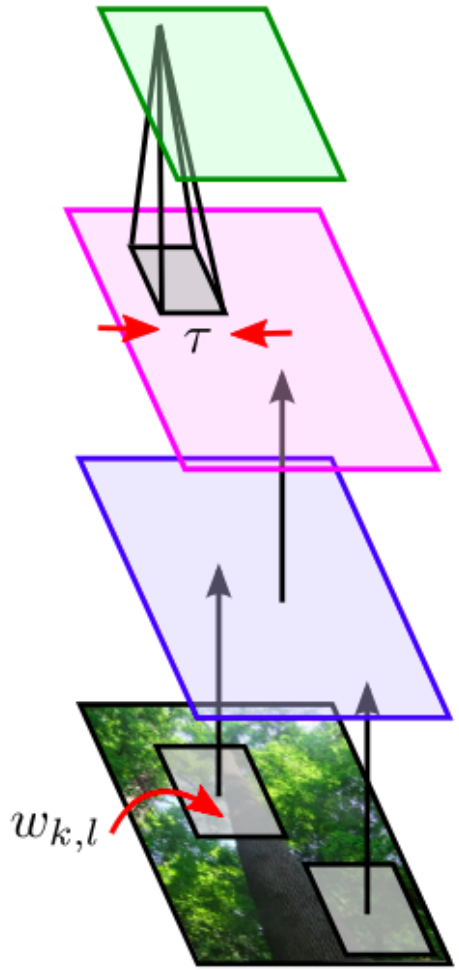
Convolutional nets reduce the number of parameters



Key Ideas behind CNNs

- **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

CNN Building Blocks



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

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

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

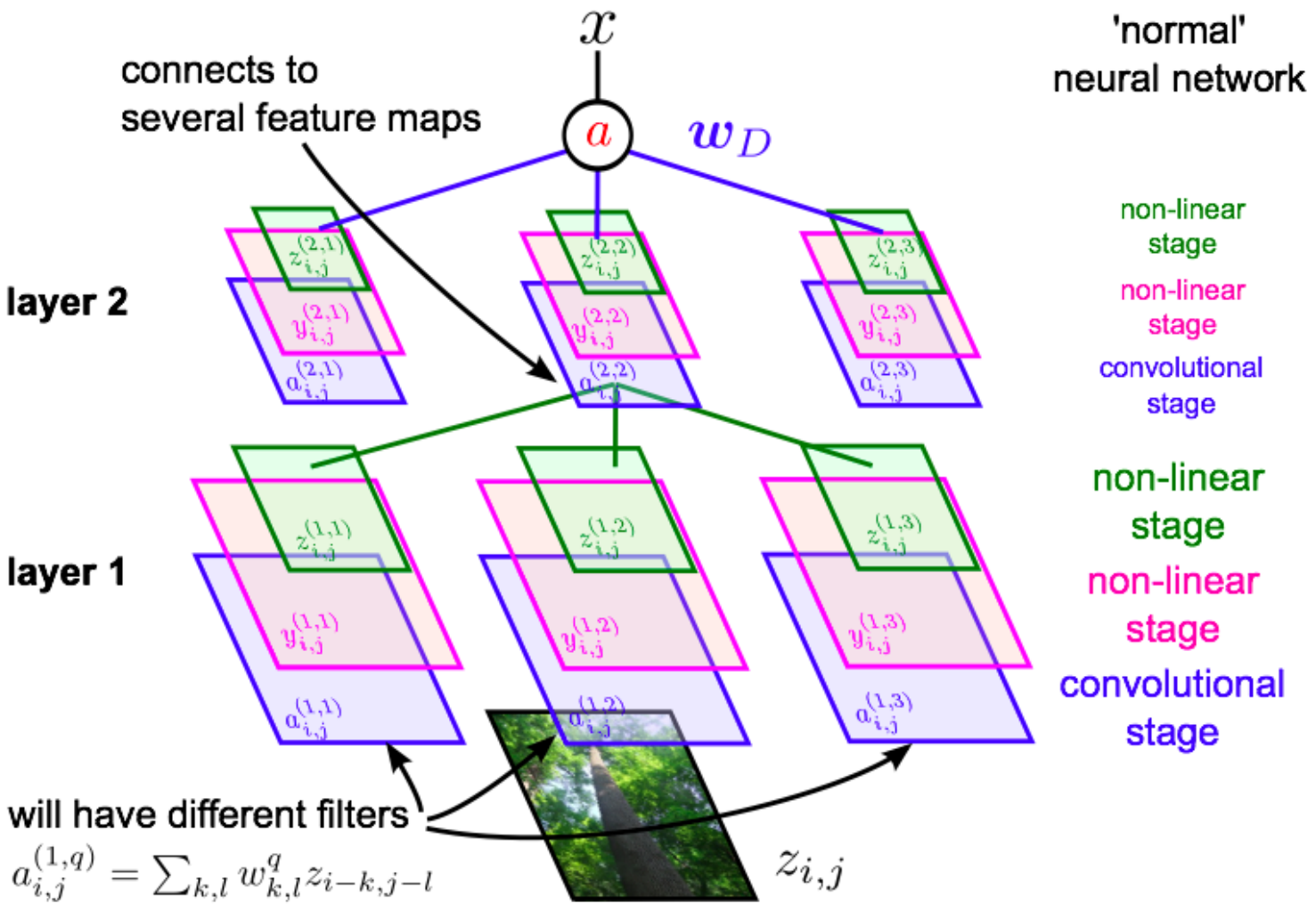
only parameters

convolutional stage

$z_{i,j}$

input image

Full Convolutional Neural Networks



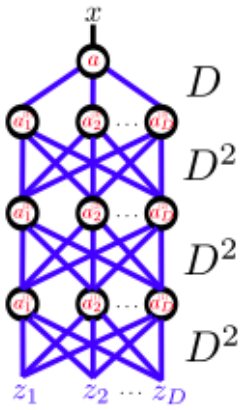
How many parameters does CNN have

How many parameters does this neural network have?

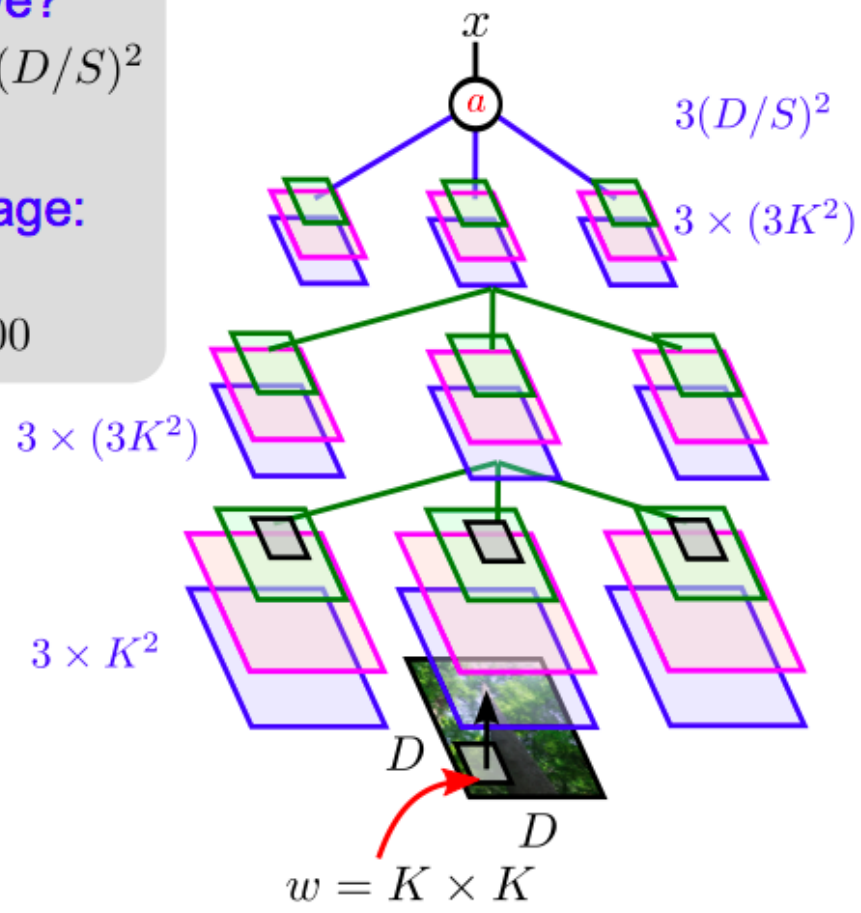
$$|\theta| = 3K^2 + 9K^2 + 9K^2 + 3(D/S)^2$$
$$= 21K^2 + D$$

For a small 32 by 32 image:

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



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



CNN 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)

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"