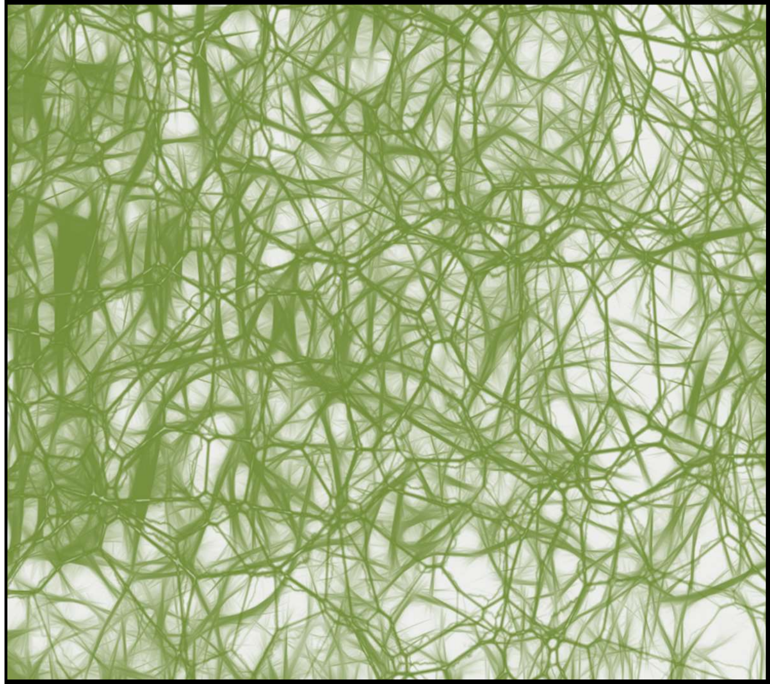
Department of Computer Science
University of Bristol

COMSM0045 – Applied Deep Learning 2020/21

comsm0045-applied-deep-learning.github.io



Lecture 05

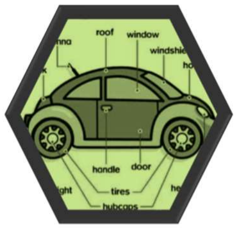
COST FUNCTIONS, REGULARISATION, DEPTH

Tilo Burghardt | tilo@cs.bris.ac.uk

29 Slides

Agenda Lecture 5

- Recap: Stochastic Gradient Descent
- Key Loss Functions
- L1 and L2 Weight Decay
- Dropout and Noise
- Data Augmentation
- Why deep is advantageous...
- Scalability Considerations...



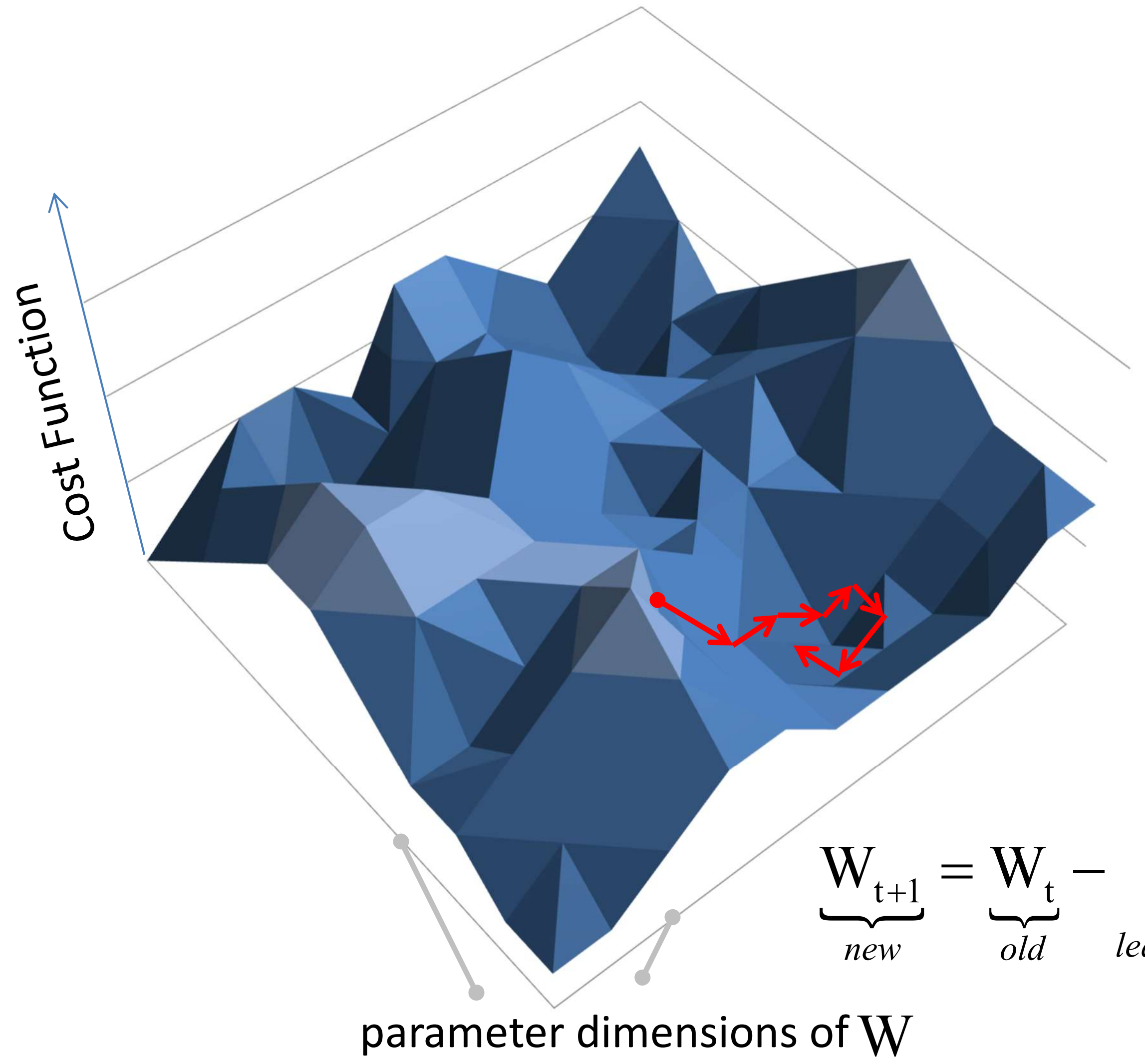
ReCap: SGD

(Online) Backpropagation so far: Notational Compaction

```
initialise all weights $w_{ij}^l$ randomly
for $t=0,\ 1,\ 2,\ \dots$ do
    pick next training sample $([f_1^0, f_2^0, \dots], [f_1^*, f_2^*, \dots])$
    FORWARD PASS: compute all $s_j^l = \sum_{i=1}^{d(l-1)} w_{ij}^l f_i^{l-1}$ and $f_j^l = g_j^l(s_j^l)$
    compute top deltas $\delta_j^N = g'^N_j(s_j^N) \cdot \partial J / \partial f_j^N$
    BACKWARD PASS: compute all $\delta_i^{l-1} = g'^{l-1}_i(s_i^{l-1}) \sum_{j=1}^{d(l)} w_{ij}^l \delta_j^l$
    update weights $w_{ij}^l \leftarrow w_{ij}^l - \eta\ f_i^{l-1} \delta_j^l$
    check if stopping criteria are met to break loop
return final weights $w_{ij}^l$
```

```
initialise all weights $\mathrm{W}$ randomly
for $t=0,\ 1,\ 2,\ \dots$ do
    pick next training sample $(\mathrm{x}, \mathrm{f}^*)$
    FORWARD-BACKWARD PASS: compute $\nabla J$
    update weights $\mathrm{W} \leftarrow \mathrm{W} - \eta \nabla J$
    check if stopping criteria are met to break loop
return final weights $\mathrm{W}$
```

Noisy Gradient Descent due to Online Sampling



$$\underbrace{W_{t+1}}_{\textit{new}} = \underbrace{W_t}_{\textit{old}} - \underbrace{\eta}_{\textit{learning rate}} \underbrace{\nabla J(x; W_t)}_{\textit{steepest gradient given } x}$$

Some Key
Cost Functions

Adjusting the Output Layer for Classification

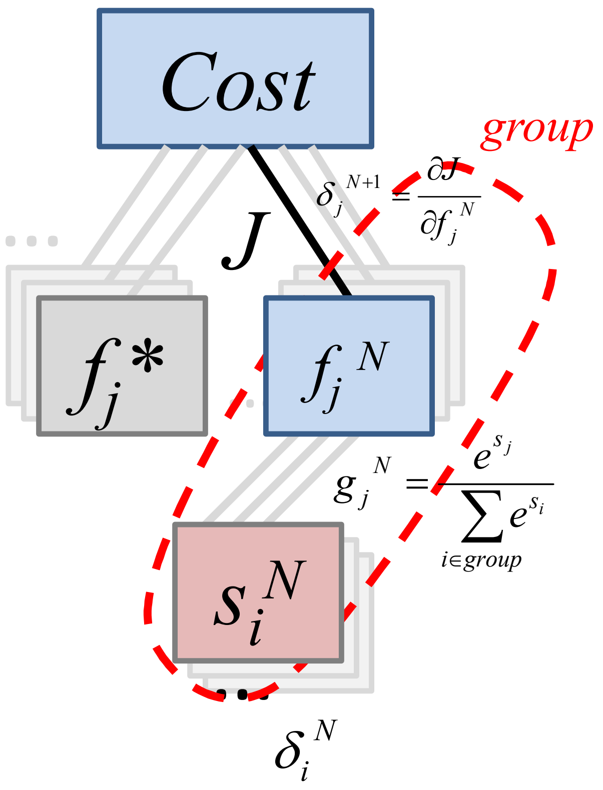
Practical Step 1: We want outputs to represent probabilities of class labels, thus outputs should be within $\{0,1\}$ and sum to 1! – Can we force outputs to reflect such a distribution by creating a layer-wide, normalising non-linearity?

→ yes, introduce a softmax neuron *group* in the last layer with:

$$g_j^N(s_j^N) = \frac{e^{s_j^N}}{\sum_{i \in group} e^{s_i^N}};$$

$$g'^N_j(s_j^N) = f_j^N(1 - f_j^N);\; g'^N_j(s_{i \neq j}^N) = -f_j^N f_i^N$$

- now all outputs f_i^N range between 0 and 1, while the *group* output sums to 1



Cost
group
J
$\delta_j^{N+1} = \frac{\partial J}{\partial f_j^N}$
f_j^*
f_j^N
$g_j^N = \frac{e^{s_j}}{\sum_{i \in group} e^{s_i}}$
s_i^N
δ_i^N

Adjusting Loss for Classification

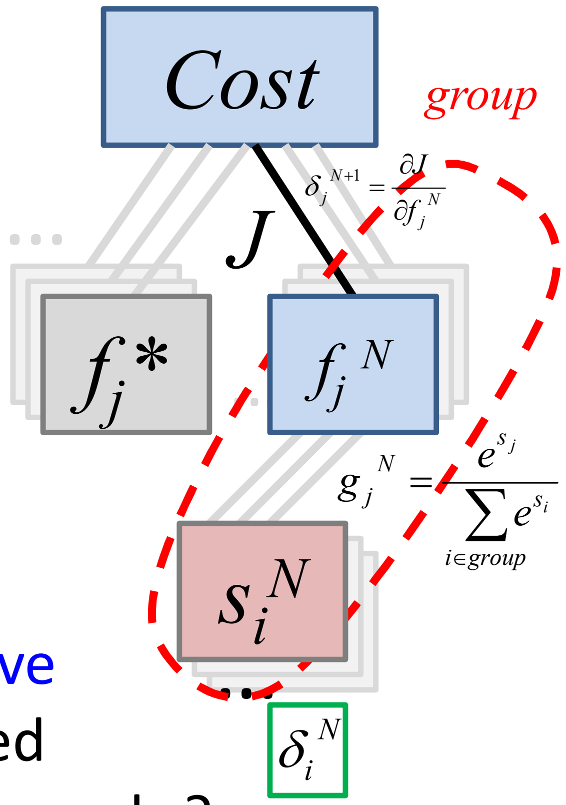
Now, so far we have changed the layout of the last layer to represent a classification setting. What is an appropriate cost function?

→ use cross-entropy

as the *group*'s cost function:

$$J = -\sum_{j \in group} f_j^{*} \log f_j^{N}$$

$$\boxed{\delta_i^N} = \sum_{j \in group} \boxed{\frac{\partial J}{\partial f_j^N}} \boxed{\frac{\partial f_j^N}{\partial s_i^N}} = f_i^N - f_i^{*}$$



$Cost$
$group$
$\delta_j^{N+1} = \frac{\partial J}{\partial f_j^N}$
J
f_j*
f_j^N
$g_j^N = \frac{e^{s_j}}{\sum_{i \in group} e^{s_i}}$
s_i^N
δ_i^N

- steepness of the cost function derivative now cancels the shallowness of the softmax derivative exactly, leading to an MSE-style delta propagated backwards from layer N – can we show why this works?

Counter-Action: Softmax exactly balances Cross-Entropy Loss

$$\delta_i^N = \frac{\partial J}{\partial s_i^N} = -\sum_{j\in group} f_j^* \underbrace{\frac{\partial \log f_j^N}{\partial s_i^N}}_{apply\ chain\ rule}$$

$$= -\sum_{j\in group} f_j^* \frac{1}{f_j^N}\frac{\partial f_j^N}{\partial s_i^N} \quad where \quad f_j^N = \frac{e^{s_j^N}}{\sum_{i\in group} e^{s_i^N}}$$

$$= -\sum_{j=i} f_j^* \frac{1}{f_j^N} \overbrace{f_j^N(1-f_j^N)}^{\partial f_j^N/\partial s_i^N \ for\ i=j} - \sum_{j\neq i} f_j^* \frac{1}{f_j^N} \underbrace{\left(-f_j^N f_i^N\right)}_{\partial f_j^N/\partial s_i^N \ for\ i\neq j}$$

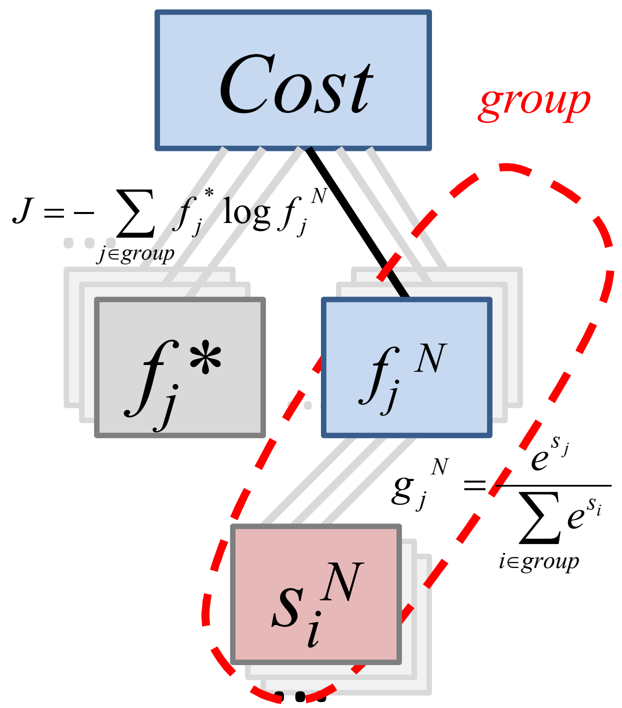
$$= -f_i^*(1-f_i^N) + \sum_{j\neq i} f_j^* \frac{f_j^N f_i^N}{f_j^N}$$

$$= -f_i^* + \underbrace{f_i^* f_i^N + \sum_{j\neq i} f_j^* f_i^N}_{=f_i^N \sum_{j\in group} f_j^*}$$

$$= f_i^N \left(\underbrace{\sum_{j\in group} f_j^*}_{=1} \right) - f_i^*$$

$$= f_i^N - f_i^*$$

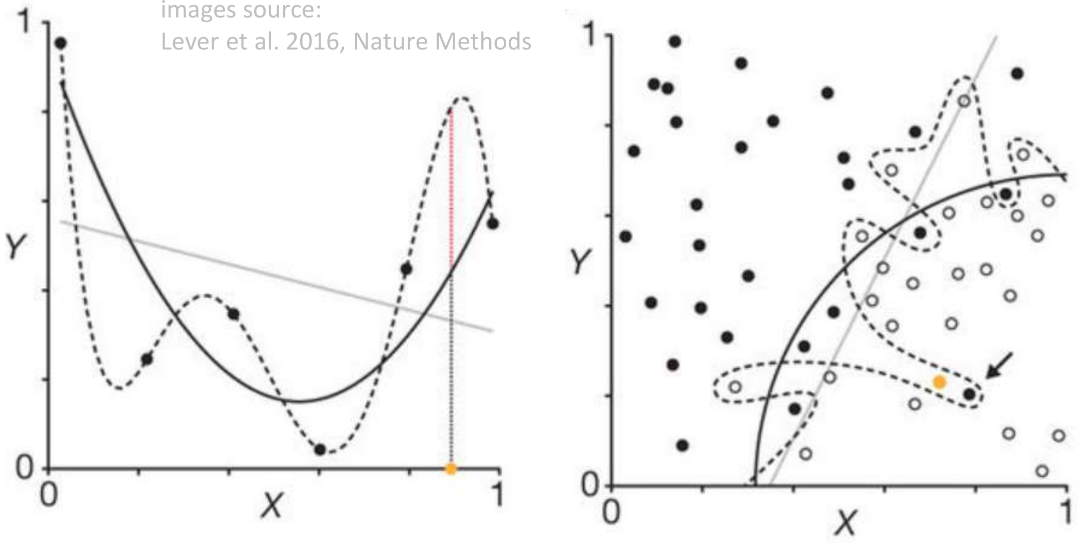
this is propagated backwards as the delta – basically, just the discrepancy for a particular output as one would intuitively expect...


Cost
group
$J = -\sum_{j\in group} f_j^* \log f_j^N$
f_j^*
f_j^N
$g_j^N = \frac{e^{s_j}}{\sum_{i\in group} e^{s_i}}$
s_i^N

Problem with High Parameter Spaces: Overfitting

• consider ImageNet *(Krizhevsky et al. 2012)*, which is a highly successful, by now classic network with (only) 8 layers; yet, it has already approx. 650,000 neurons, 60,000,000 parameters and 630 million connections!
→ models with such a high degrees of freedom are particularly prone to overfitting

images source:
Lever et al. 2016, Nature Methods



overfitting (dotted) and underfitting (grey) examples for regression (left) and classification (right) task

Loss across Training: 'Pocket' Algorithm and Early Stopping

Keep best N results at any stage ('in your pocket') based on validation set and decide based on further performance whether to revert back to it. One may also decide to terminate learning altogether (early stopping) while the validation performance is better.

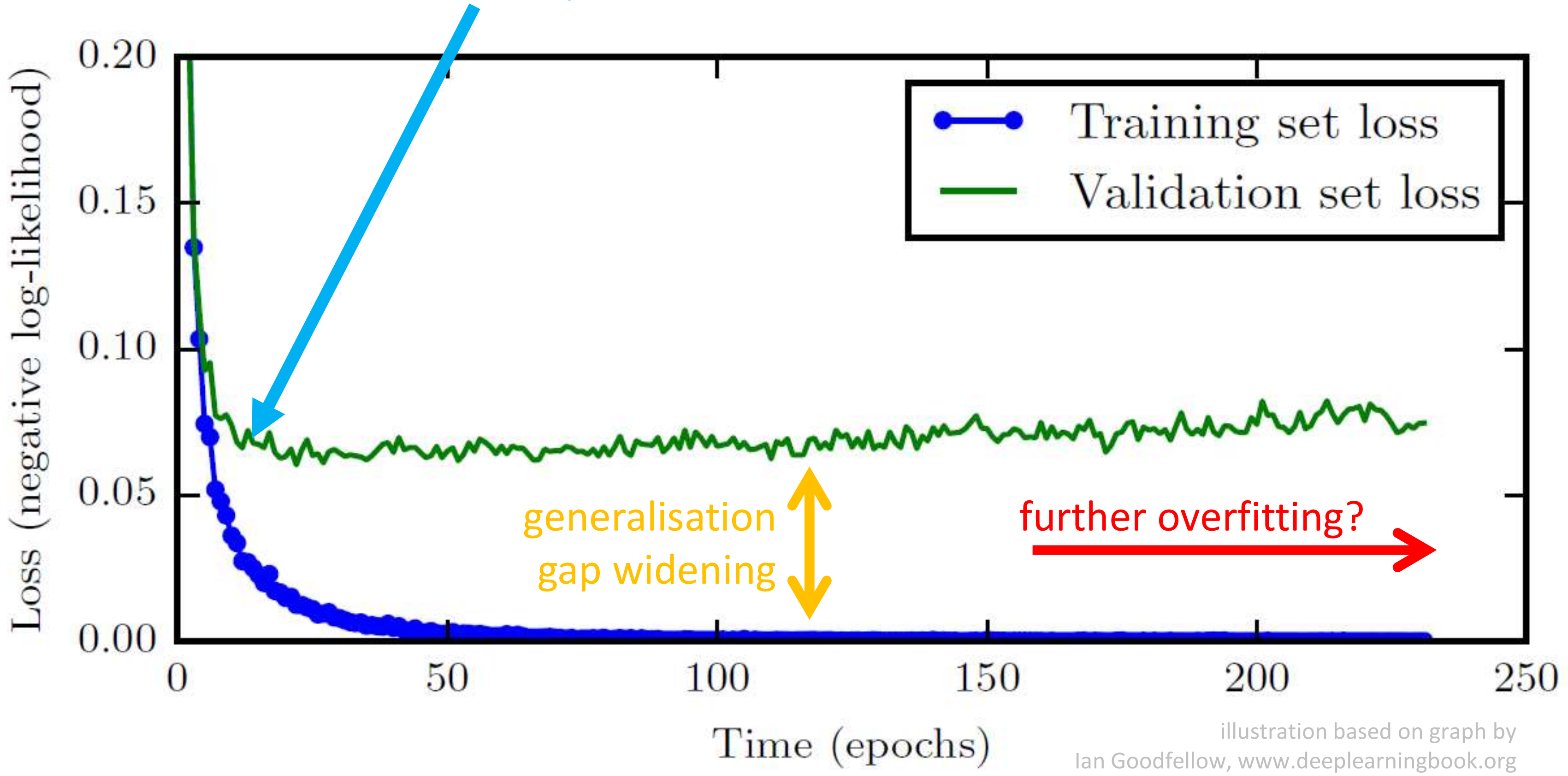


illustration based on graph by Ian Goodfellow, www.deeplearningbook.org

Regularisation

Regularisation

"Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error, but not its training error."

From: Ian Goodfellow 2015

→ One may view many recent developments in deep learning as attempts to **reduce the complexity** of neural network training models – including
L-regularization (constraining weight space),
dropout (stochastic network thinning),
depth (composition over concatenation), as well as parameter (CNNs) and network (RNNs) **sharing**.

L2-Regularisation

- **Assumption:** we target a local minimum with *small-magnitude weights* to combat overfitting
- **Idea:** introduce a penalty for every weight based on its squared value directly in the cost function J:

$$J(\mathrm{X};\mathrm{W}) = \boxed{\frac{1}{|\mathrm{X}|}\sum_{\mathrm{x}\in\mathrm{X}} L(f(\mathrm{x};\mathrm{W}), f^{*}(\mathrm{x}))} \boxed{+\frac{1}{2}\sum_{w\in\mathrm{W}}\lambda w^{2}}$$

- practically, this is easiest implemented as scaled weight decay towards zero during training:

$$\mathrm{W}_{\mathrm{t}+1} = \mathrm{W}_{\mathrm{t}} - \eta \boxed{(\nabla J_{plain}(\mathrm{X};\mathrm{W}_{\mathrm{t}})} \boxed{+\lambda \mathrm{W}_{\mathrm{t}})}$$

L1-Regularisation

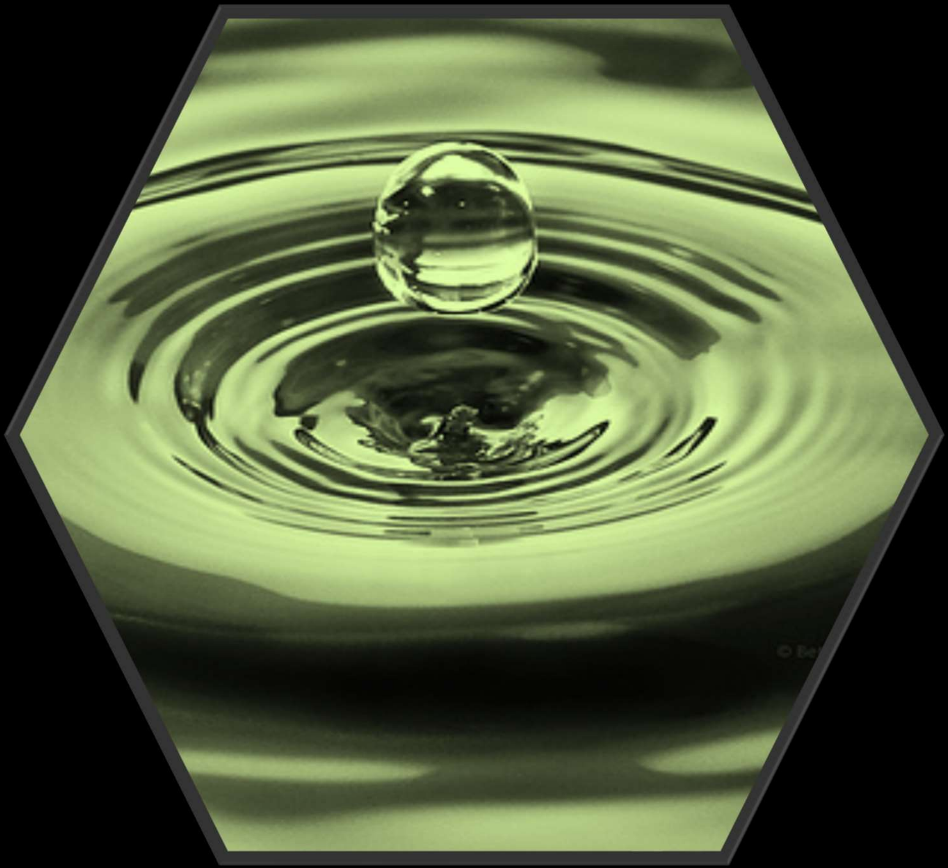
- **Assumption:** we target a local minimum with *sparse weights* against overfitting (Occam's Razor)
- **Idea:** introduce a penalty for every weight based on its magnitude directly in the cost function J:

$$J(\mathrm{X};\mathrm{W}) = \boxed{\frac{1}{|\mathrm{X}|}\sum_{\mathrm{x}\in\mathrm{X}} L(f(\mathrm{x};\mathrm{W}), f^{*}(\mathrm{x}))}\ \boxed{+\sum_{w\in\mathrm{W}} \mu\,|w|}$$

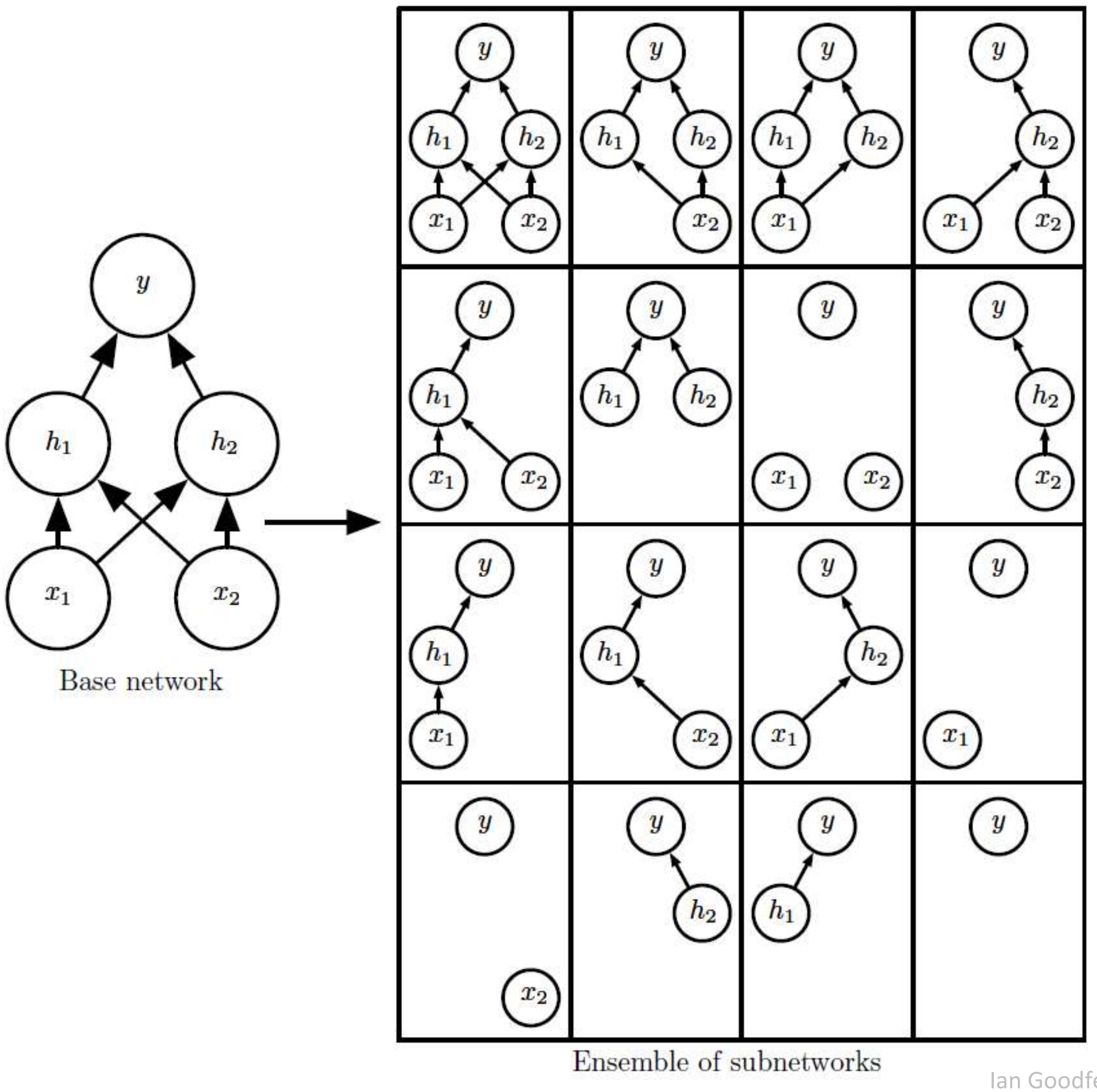
- practically, this is easiest implemented as absolute weight decay towards zero during training:

$$\mathrm{W}_{\mathrm{t+1}} = \mathrm{W}_{\mathrm{t}} - \eta\,(\boxed{\nabla J_{plain}(\mathrm{X};\mathrm{W}_{\mathrm{t}})}\ \boxed{+\mu\,\mathrm{sgn}(\mathrm{W}_{\mathrm{t}}))}$$

Dropout



Neural Networks as Ensemble of Sub-networks

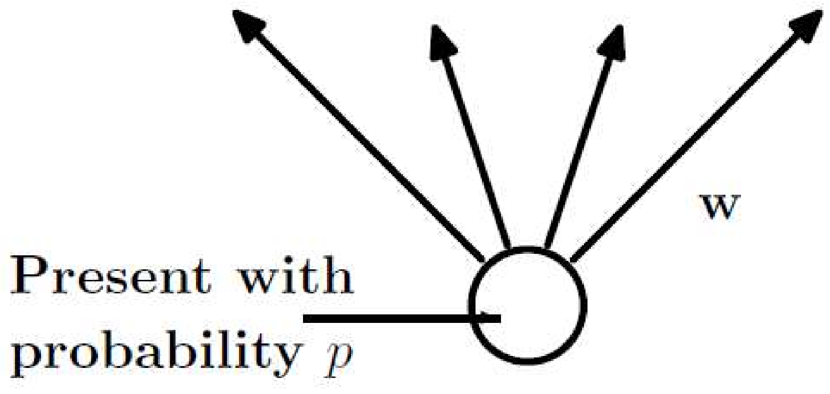


adjusted based on graph by
Ian Goodfellow, www.deeplearningbook.org

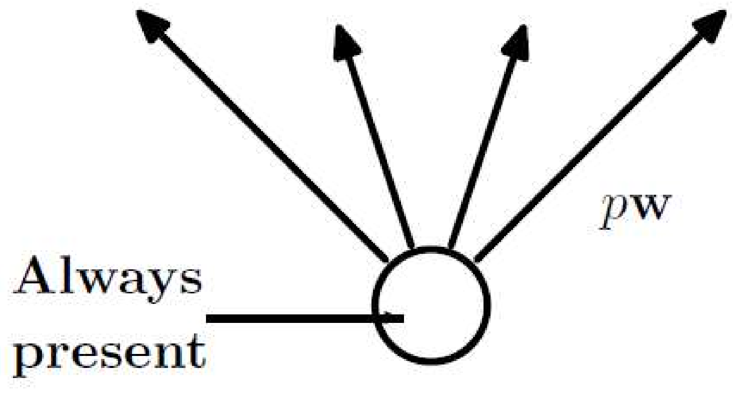
Dropout for Regularisation *(Srivastava et al. 2014)*

- **Idea:** during training, for each loop of weight updates `immobilise' a random subset of neurons with probability *(1-p)* by forcing their output to zero (particularly effective when applied to final fully-connected layers of networks)

→ these neurons now do not contribute to the network output nor can they be used for adjusting weights or passing gradients through them; p is often set to a value around 0.5 before tuning on validation data



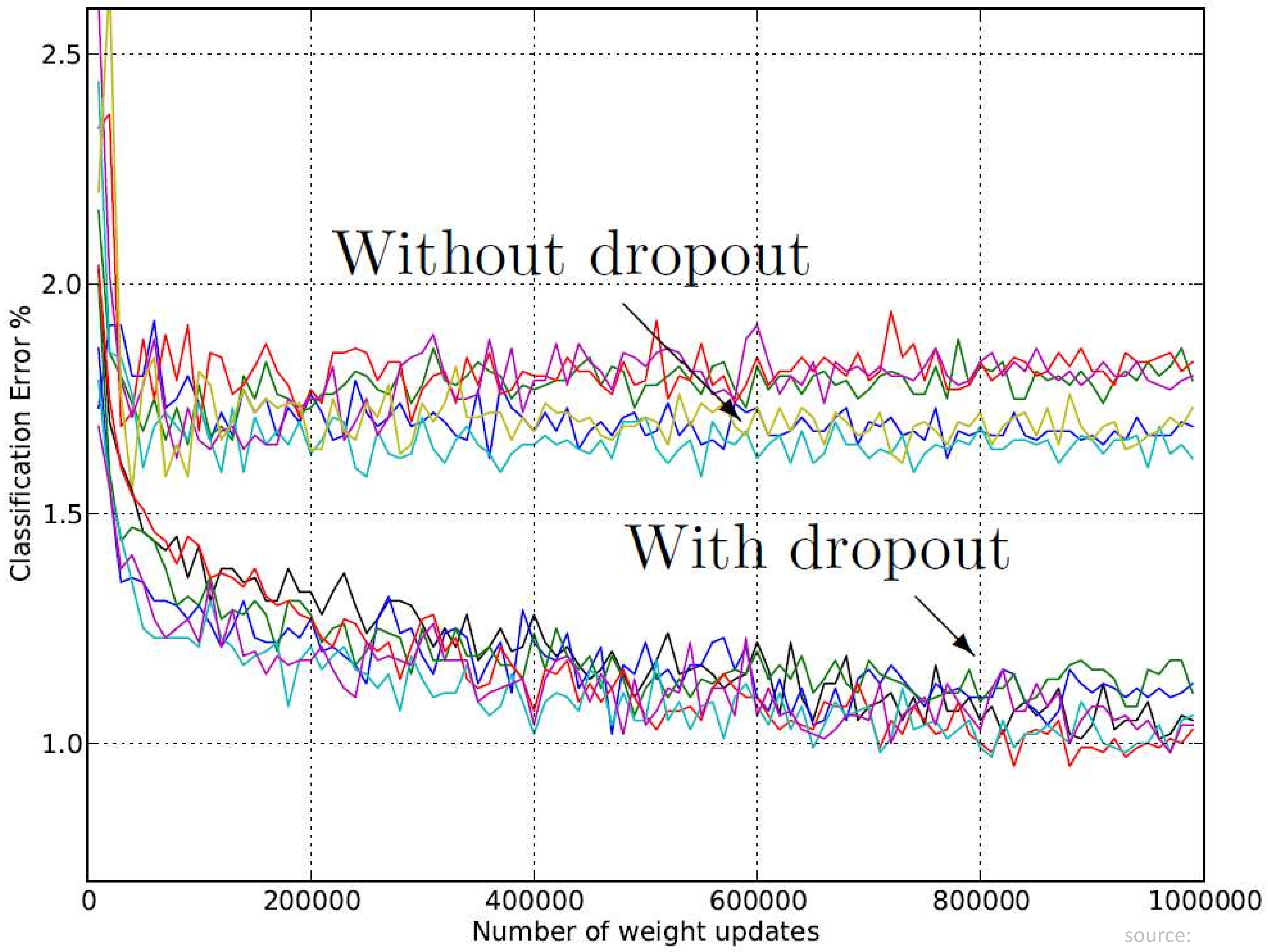
Training Time



Testing Time

image source: Srivastava et al. 2014

MNIST Test Performance Impact of Dropout on Test Data



source:
Srivastava et al. 2014

DropConnect *(Wan et al. 2013)*

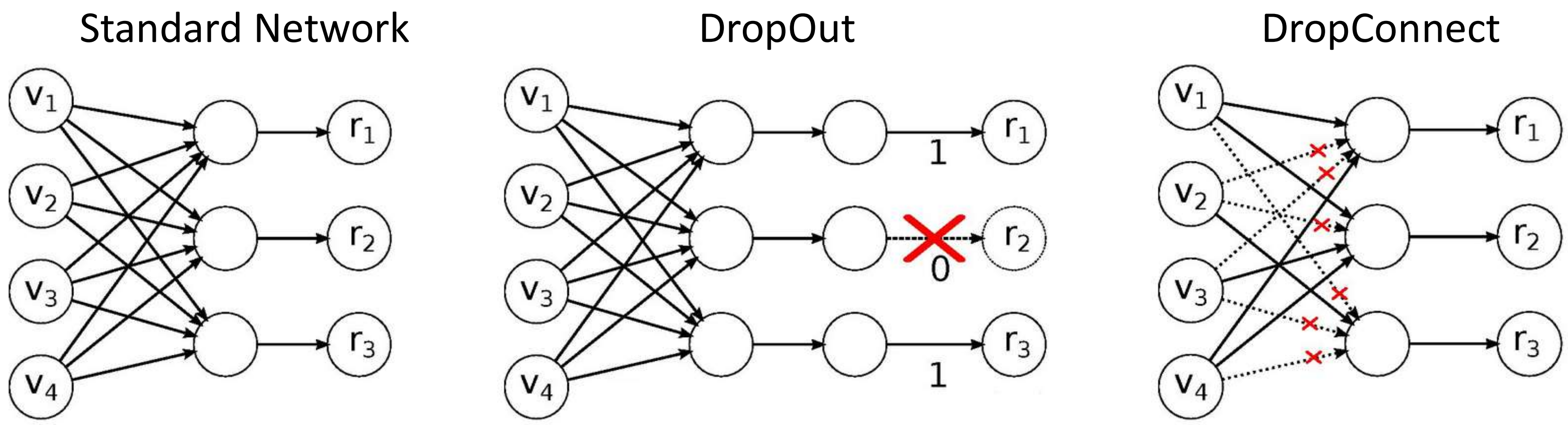


image source:
Wan Li, http://cs.nyu.edu/~wanli/dropc

- **Idea:** instead of dropping neurons, DropConnect sets weights to zero with a probability *(1-p)* providing a more fine-grained mechanism for regularisation based on the same underlying concept.

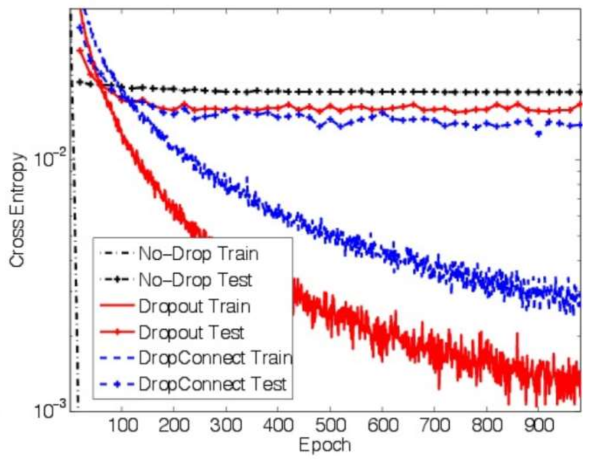


image source:
Wan Li, http://cs.nyu.edu/~wanli/dropc

Data against Overfitting

- so far, we have addressed overfitting by limiting the representational capacity of our networks by introduction of model constraints and by interpretation of the networks as ensembles
- however, overfitting can always be addressed by more data, more representative data and/or strategically sampled data

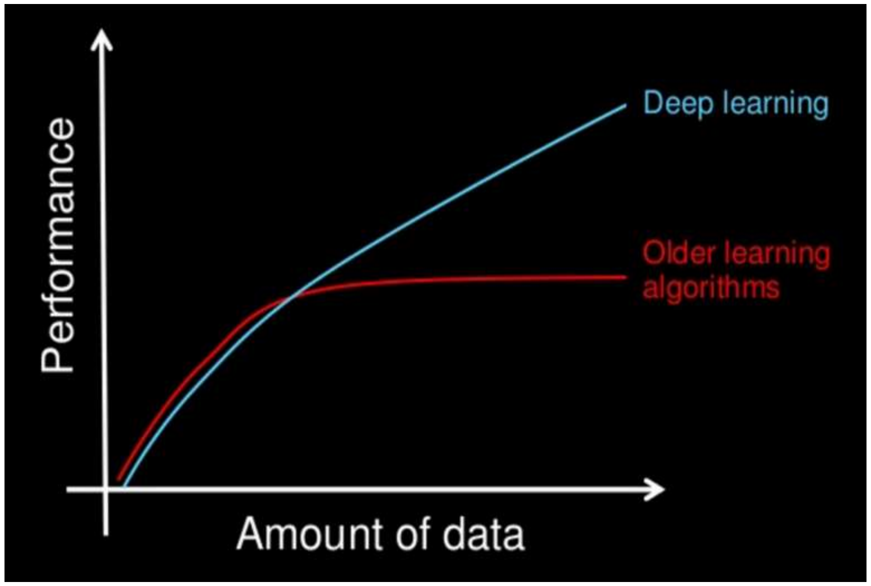
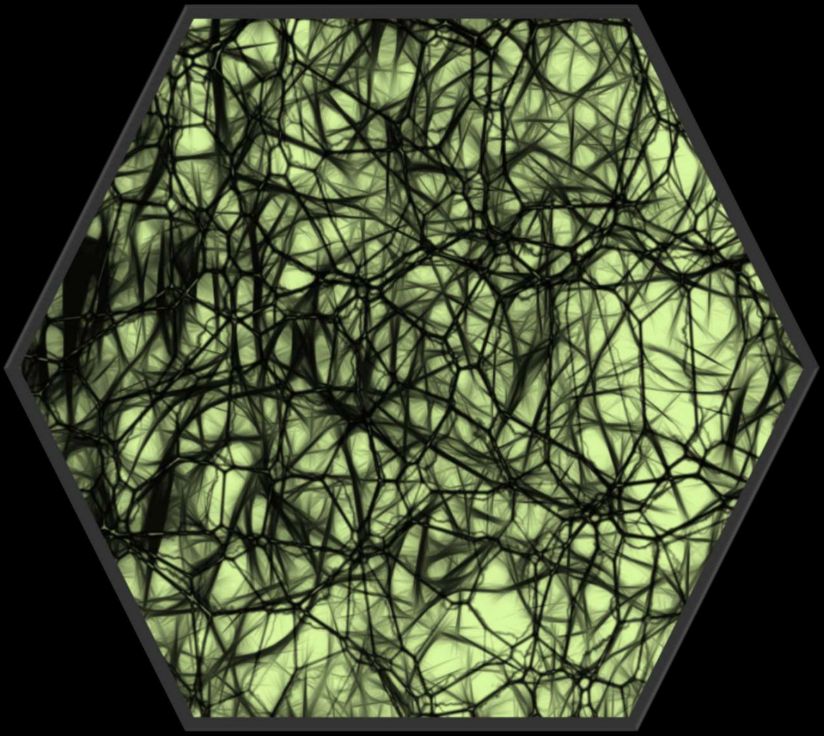


image source: Andrew Ng

- given the high dimensionality of deep net parameter spaces, taking advantage of more data is often possible without reaching the representational limits of the network

Data Augmentation



Realistic Variance Matters: Augmentation for Images

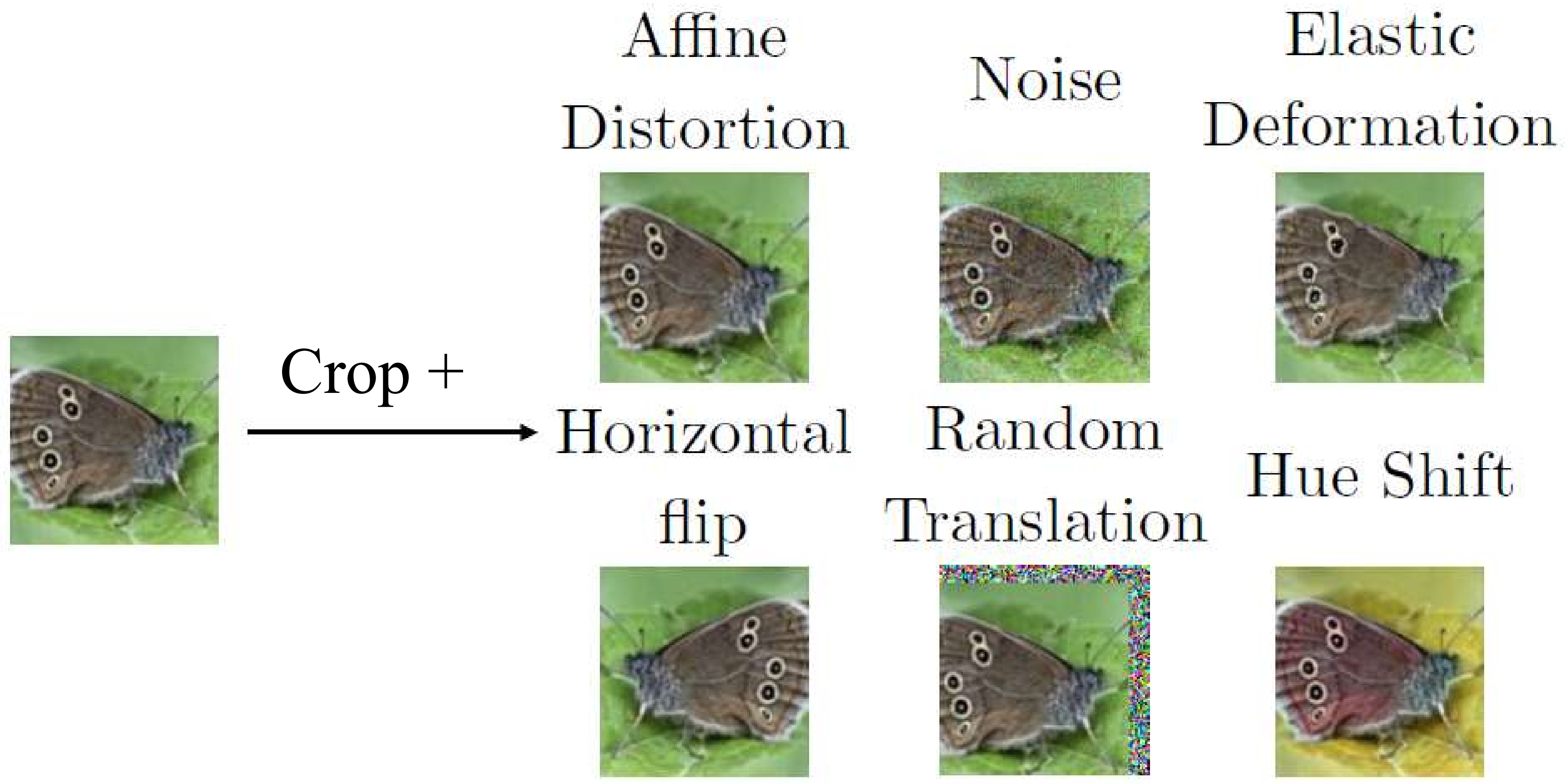


image source: Ian Goodfellow

Example: Augmentation for Images

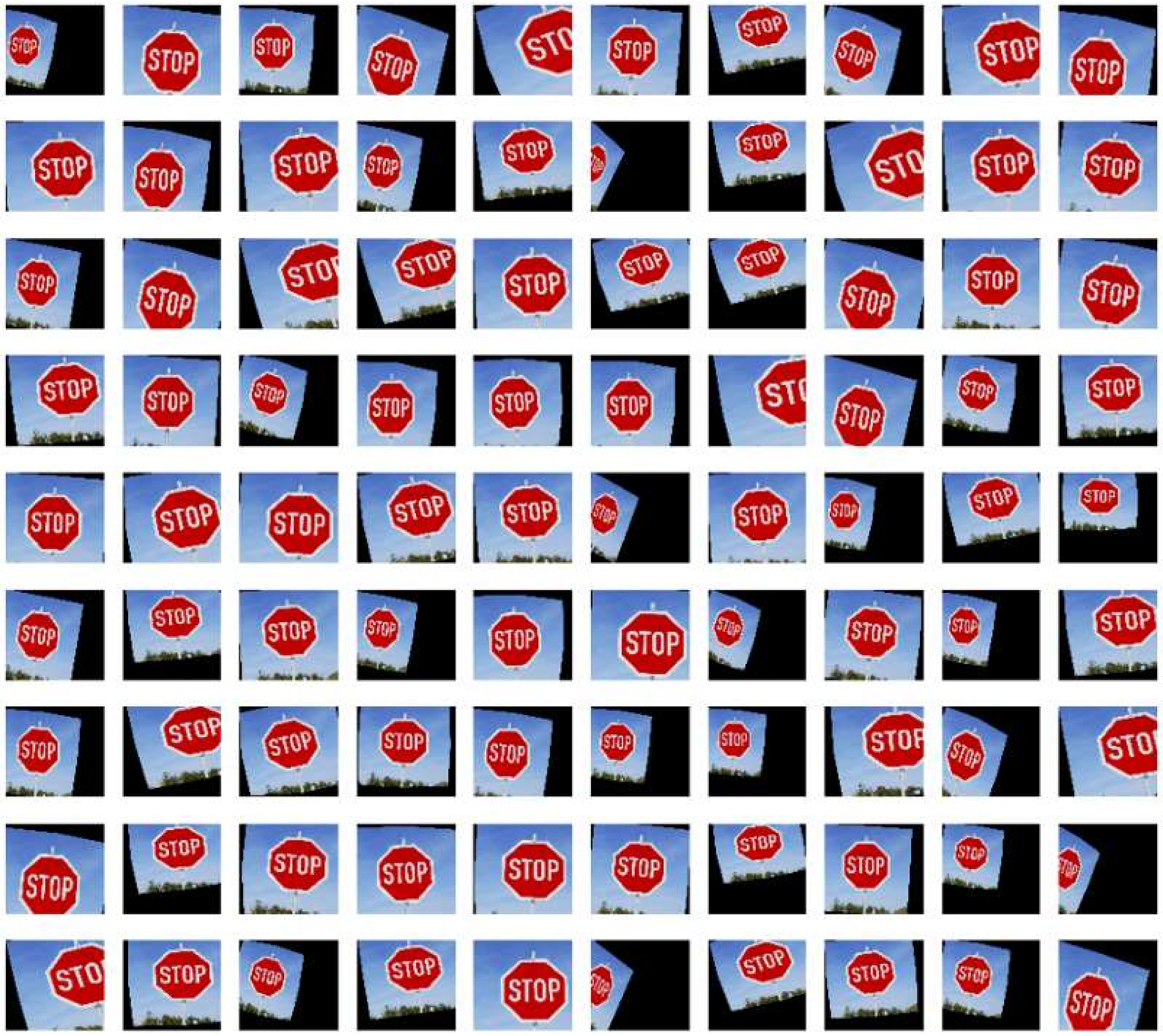
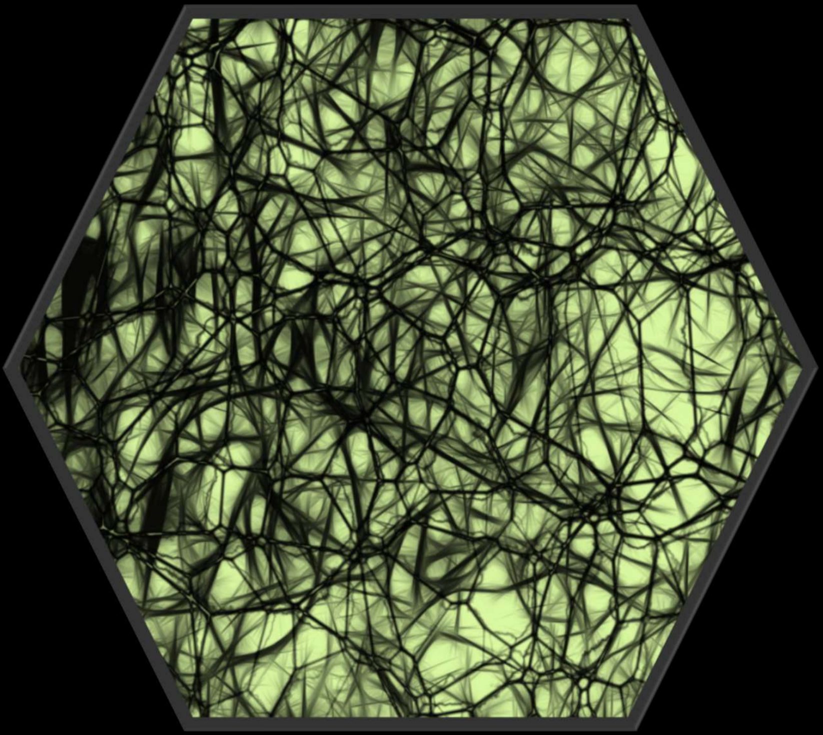


image source: V Yadav, https://github.com/vxy10

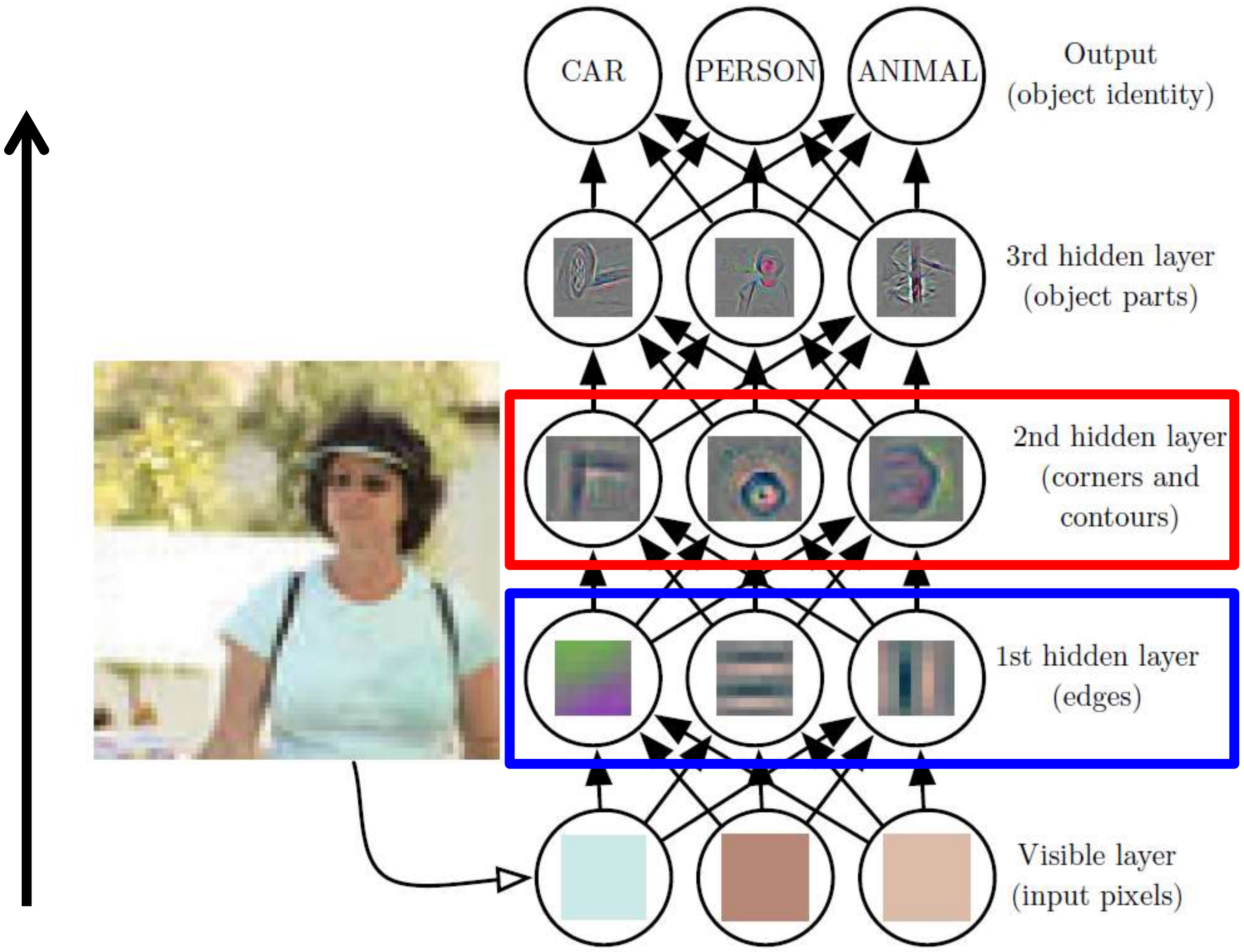
Why Deep and Scalability

(open topic)



Point 1: Hierarchical Automatic Modularisation

increasing module specificity and complexity


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

availability of previous layer may <u>reduce</u> data requirement for training this layer

'module' <u>shared</u> by following layer

image adapted from Ian Goodfellow, www.deeplearningbook.org

Point 2: Practical Performance (Example)

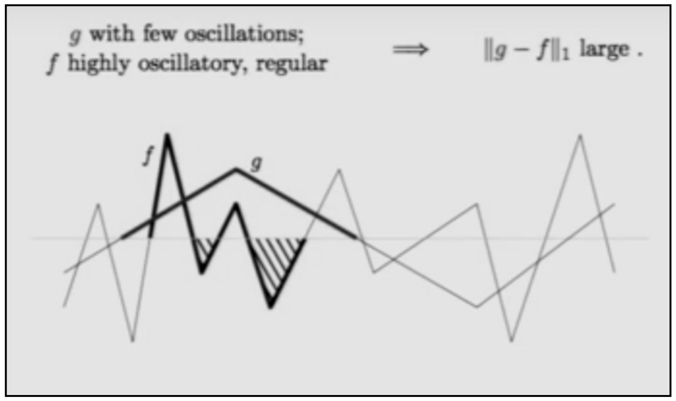
- **Example:** Conversational Speech Transcription Using Context-Dependent Deep Neural Networks *(Seide et al. 2011)*
- direct comparison of different layer depth shows performance advantage for deeper structures across many domains
- however, required training times increase rapidly given large datasets and deep/large network architectures

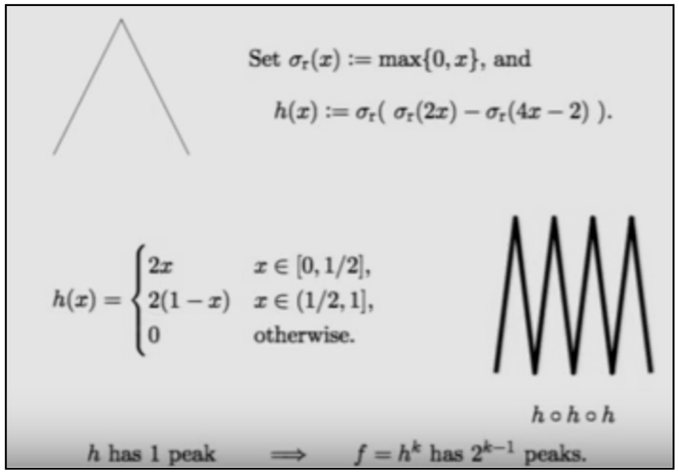
Hidden Layers *x* Size	**Error Rate (%) (word-error)**
1 x 2k	*24.2*
2 x 2k	*20.4*
3 x 2k	*18.4*
4 x 2k	*17.8*
5 x 2k	*17.2*
7 x 2k	*17.1*
1 x 3.8k	*22.5*
1 x 4.6k	*22.6*
1 x 16k	*22.1*

data source:
Seide et al 2011

Point 3: Sketch of Oscillation Argument *(Telgarsky 2016)*

- **Key Claim:** There are functions f that can be represented by a deep ReLU net with a polynomial number of neurons, where a shallow network would require exponentially many units.
- Telgarsky shows that k ReLU-like layers h can generate in the order of 2^{k-1} oscillations to approximate functions, while shallow networks have exponentially less peaks.
- **Open Question:** Are these functions f the type of `natural’ functions of interest to practical problems?


g with few oscillations; f highly oscillatory, regular $\Longrightarrow$ $\|g - f\|_1$ large .


Set $\sigma_r(x) := \max\{0, x\}$, and

$$h(x) := \sigma_r(\ \sigma_r(2x) - \sigma_r(4x-2)\ ).$$

$$h(x) = \begin{cases} 2x & x \in [0, 1/2], \\ 2(1-x) & x \in (1/2, 1], \\ 0 & \text{otherwise.} \end{cases}$$

h has 1 peak $\Longrightarrow$ $f = h^k$ has 2^{k-1} peaks.

image source: Telgarsky 2016, watch https://www.youtube.com/watch?v=ssaXJqG9Dz4

Some Current Ideas related to Scale and Depth

- **Idea of Mixtures of Experts:** extend network capacity and yet keep parameters per training run at bay via learning to switch on/off parts of networks dependent on their appropriateness per example *(Shazeer et al. 2017)*

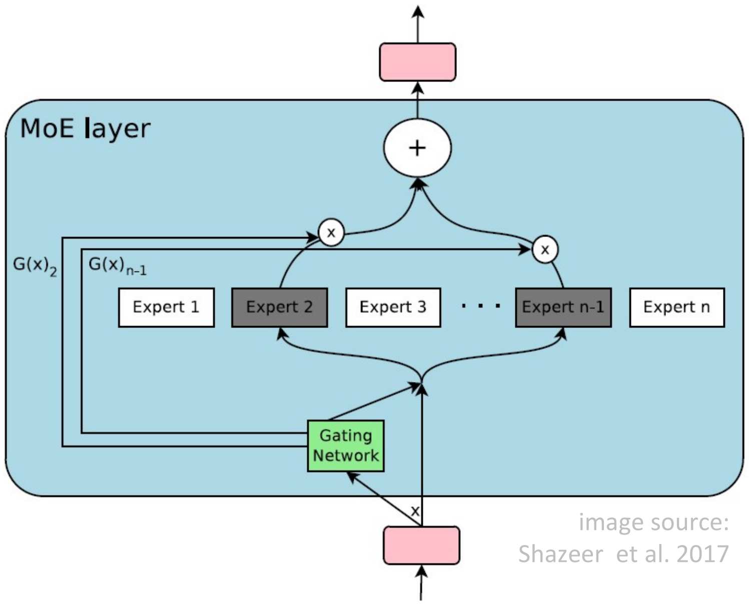
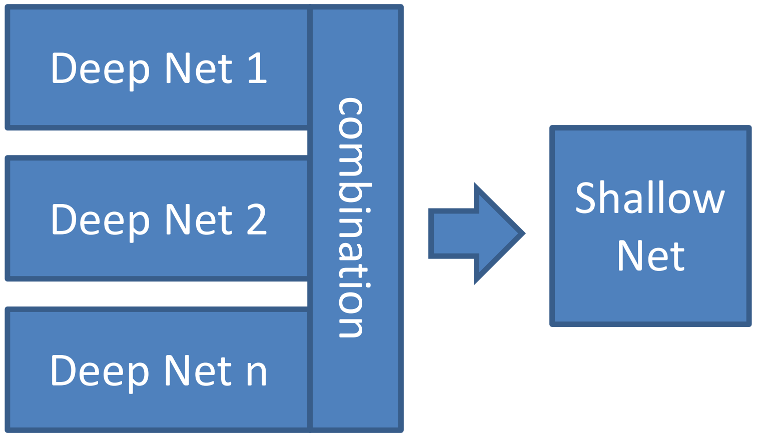


image source: Shazeer et al. 2017

- **Idea of Network Distillation:** learn deep ensembles, but then 'compress' deep networks into shallow or more efficient compactions *(Ba and Caruana 2014, Hinton 2017)*


Deep Net 1
Deep Net 2
Deep Net n
combination
Shallow Net