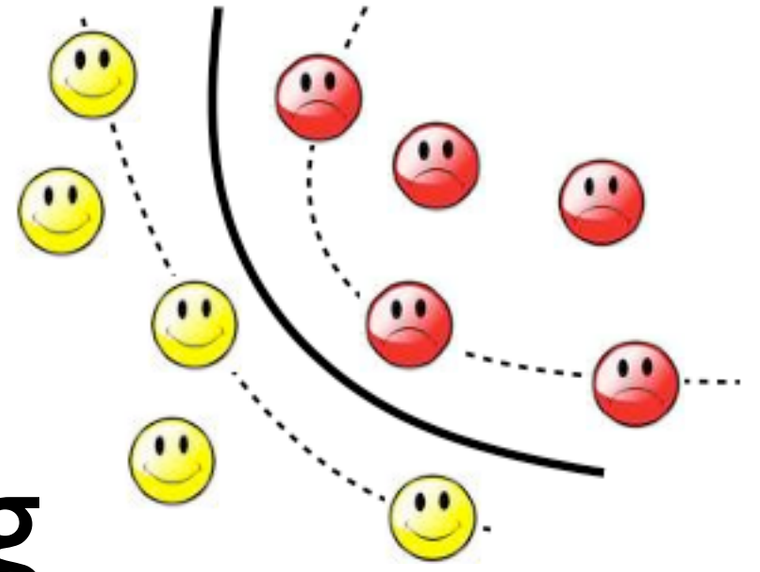
THE UNIVERSITY OF SYDNEY



Machine Learning and Data Mining
(COMP 5318)

Deep learning

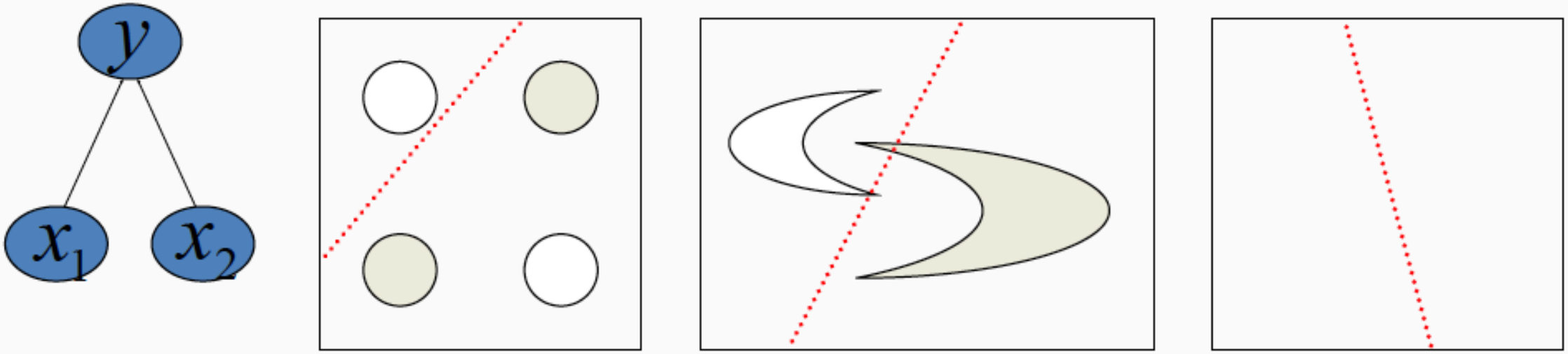
Nguyen Hoang Tran

Feed Forward Neural Networks

THE UNIVERSITY OF SYDNEY

Neural networks can be very expressive, especially as you add more layers (i.e., 'deep' neural networks)

0 hidden layers: linear classifier

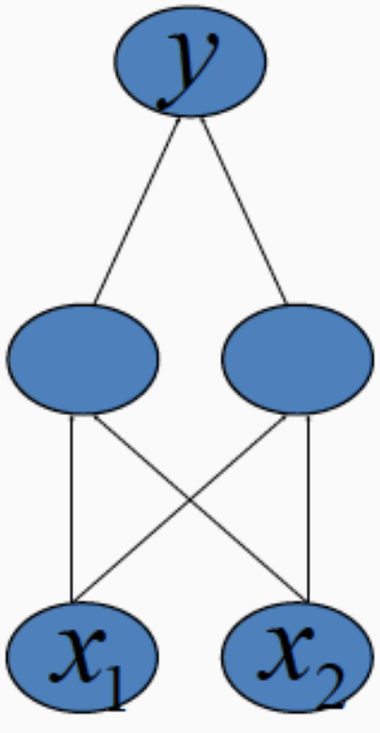


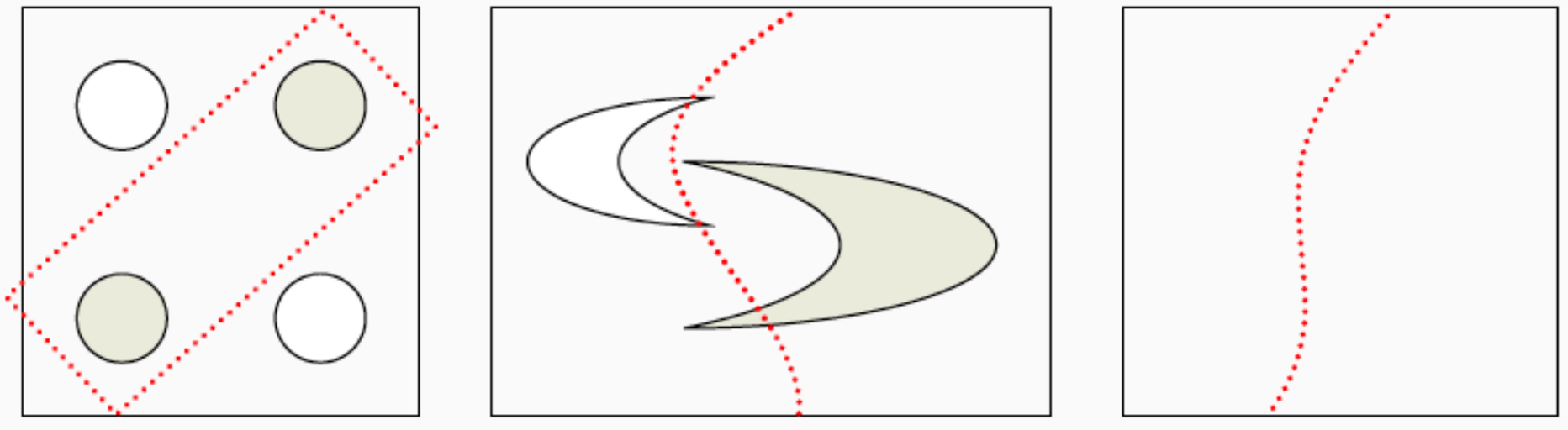
Feed Forward Neural Networks

THE UNIVERSITY OF SYDNEY

Neural networks can be very expressive, especially as you add more layers (i.e., 'deep' neural networks)

1 hidden layer: boundary of convex region (open or closed)

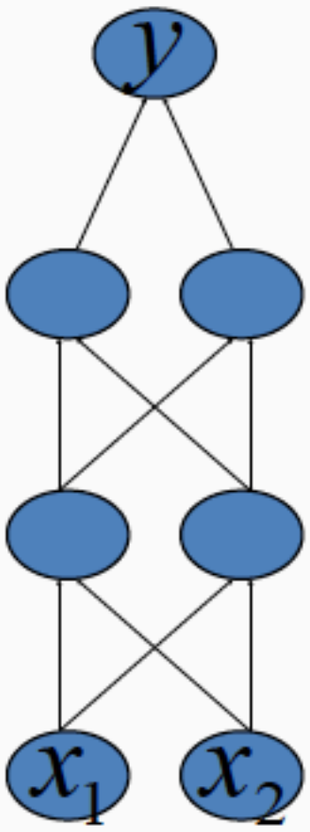




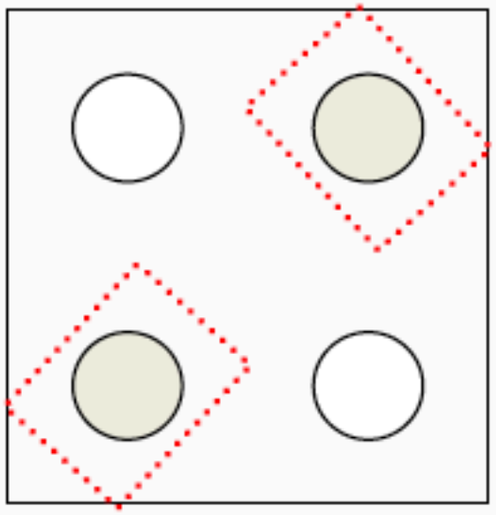
Feed Forward Neural Networks

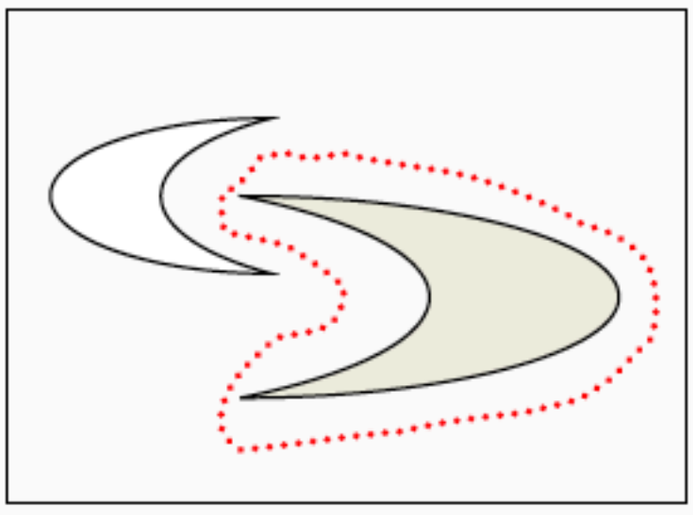
THE UNIVERSITY OF SYDNEY

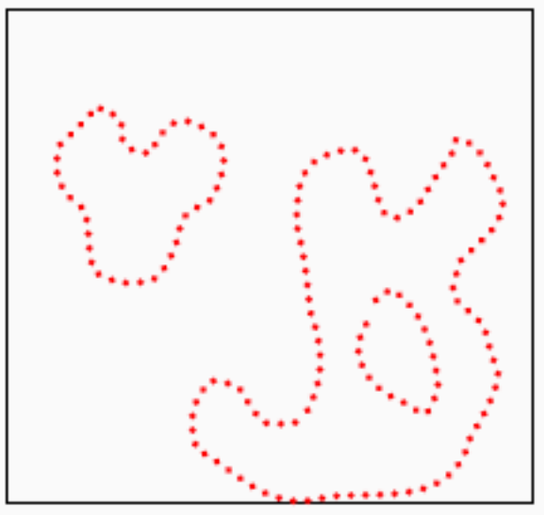
Neural networks can be very expressive, especially as you add more layers (i.e., 'deep' neural networks)



2 hidden layers: combinations of convex regions







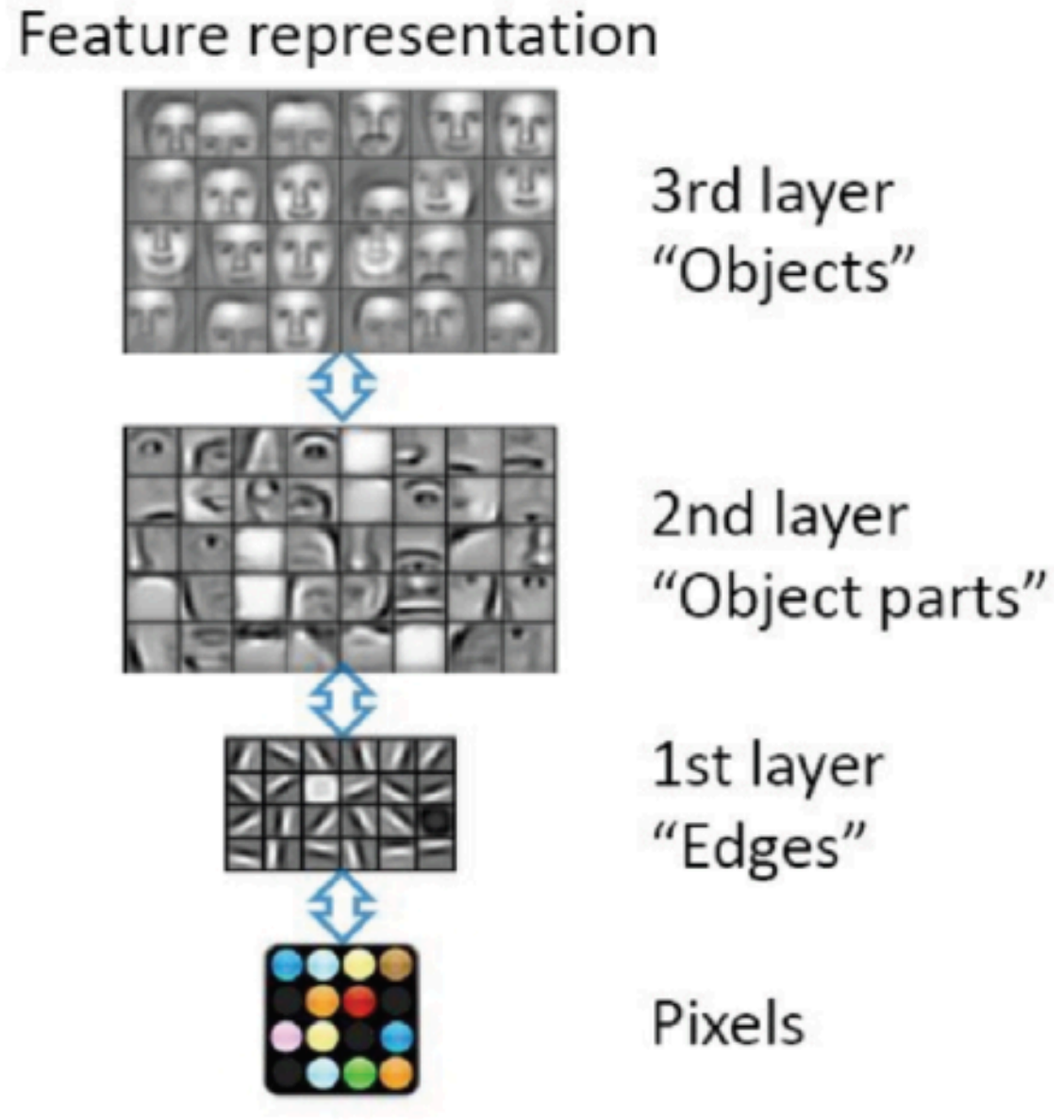
Feed Forward Neural Networks

THE UNIVERSITY OF SYDNEY

Neural networks can be very expressive, especially as you add more layers (i.e., 'deep' neural networks)

Pro: Can allow the model to learn a feature representation and the appropriate levels of abstraction from your data

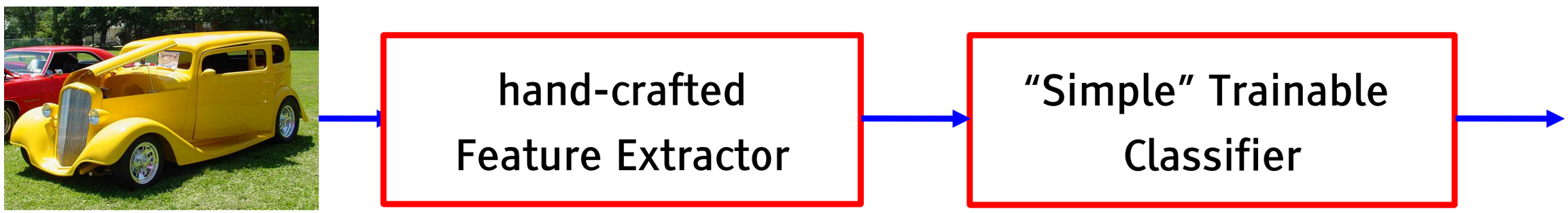
Challenge: this can make DNNs expensive …



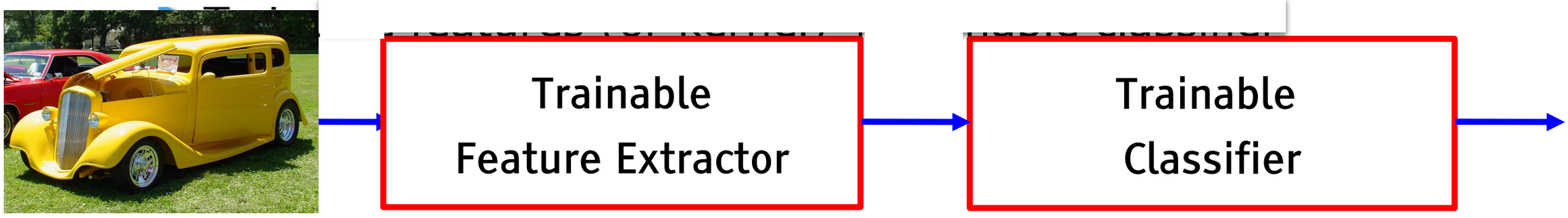
Deep Learning = Learning Features

THE UNIVERSITY OF SYDNEY

- **The traditional model of pattern recognition (since the late 50's)**
  - Fixed/engineered features (or fixed kernel) + trainable classifier



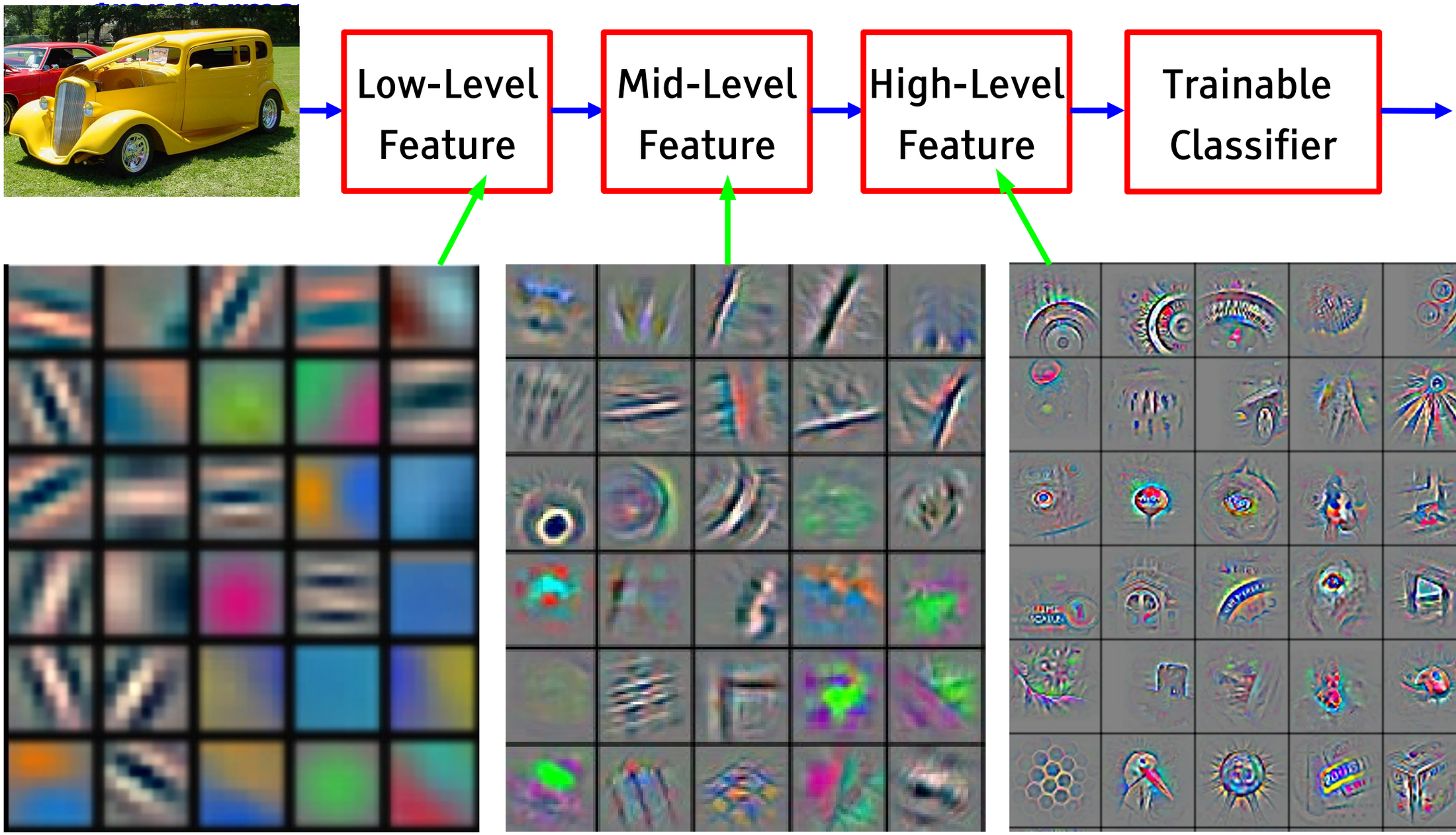
- **End-to-end learning / Feature learning / Deep learning**
  - Trainable features (or kernel) + classifier



Deep Learning = Learning Hierarchies

THE UNIVERSITY OF SYDNEY

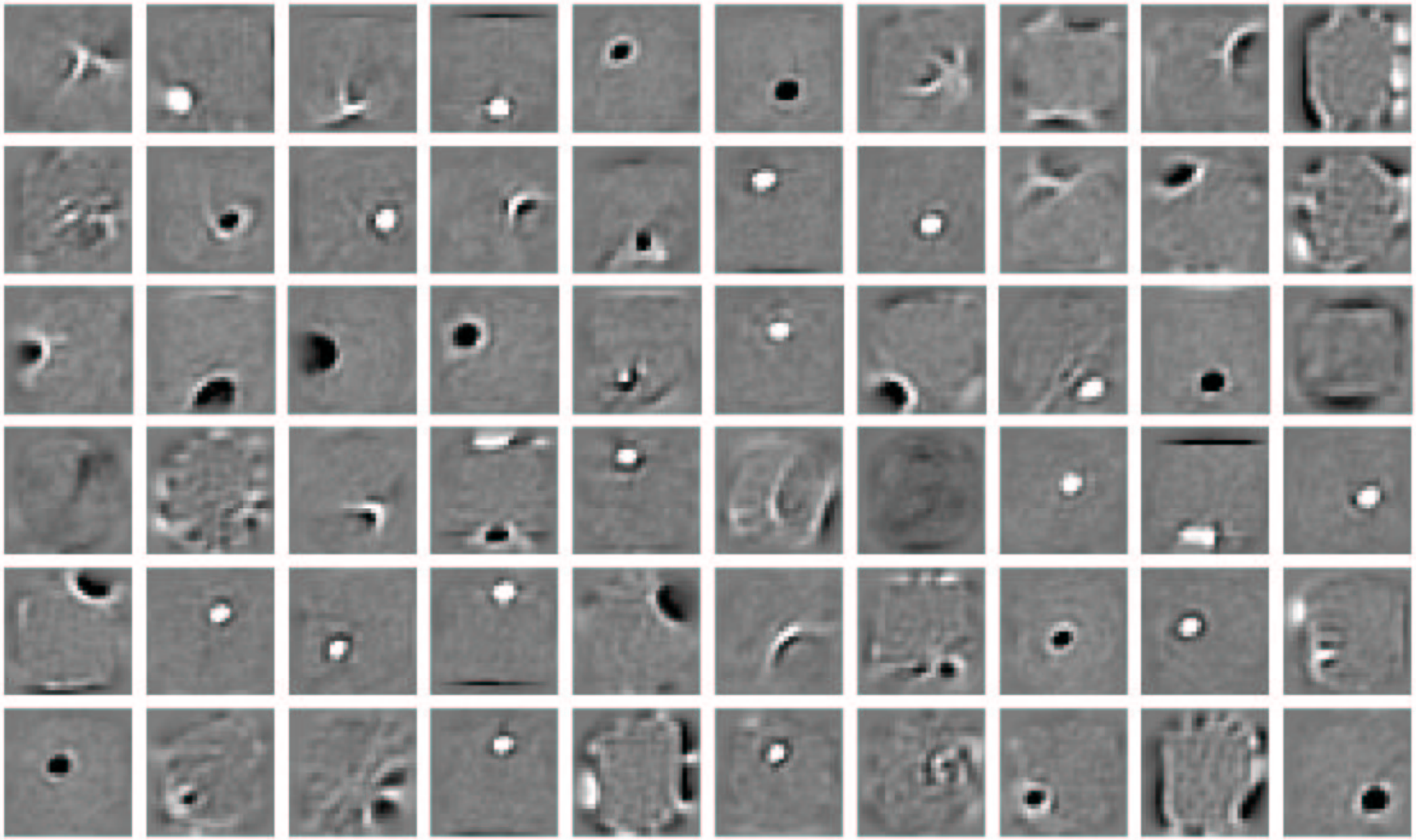
- It's deep if it has more than one stage of nonlinear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

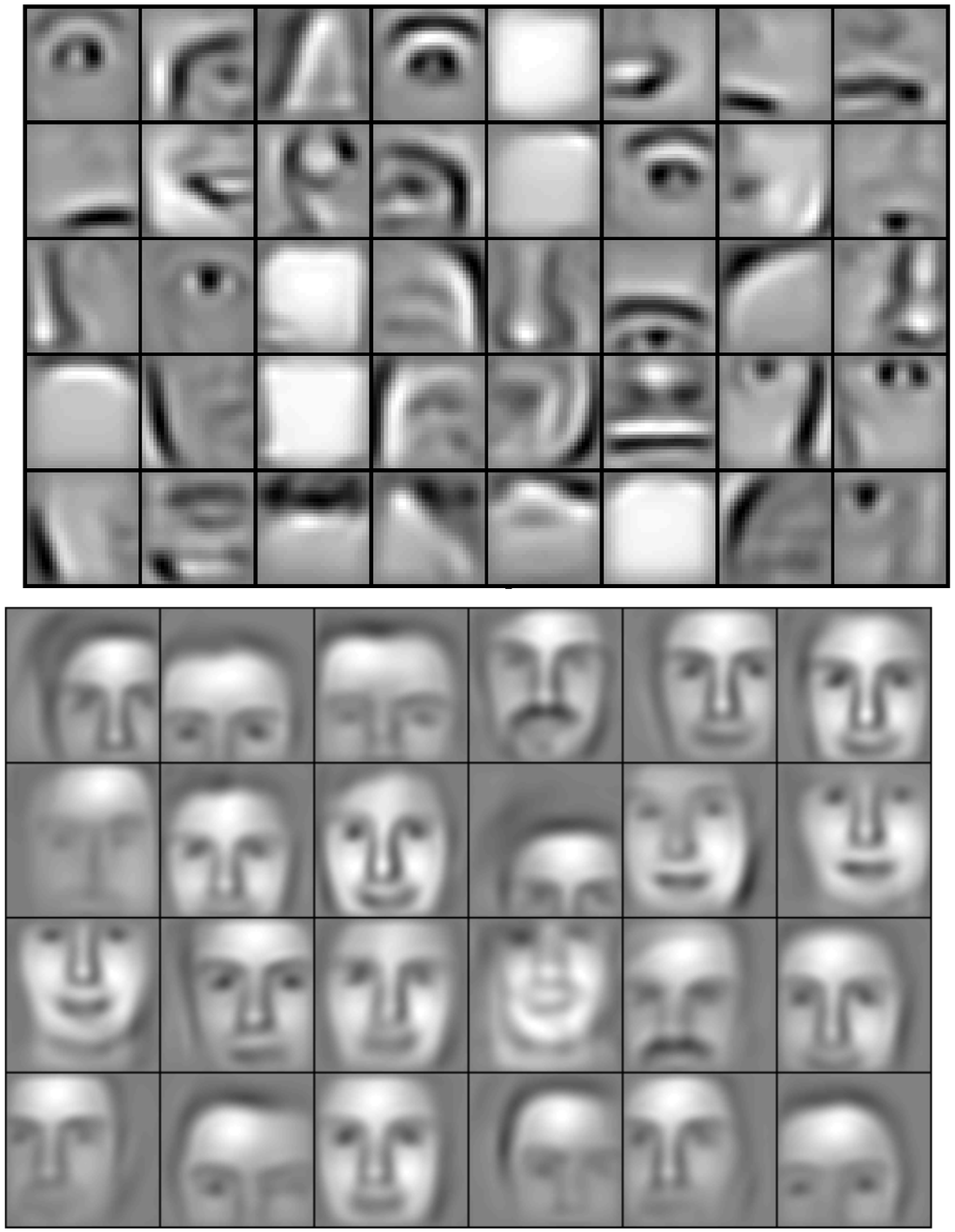
First Layer

THE UNIVERSITY OF SYDNEY

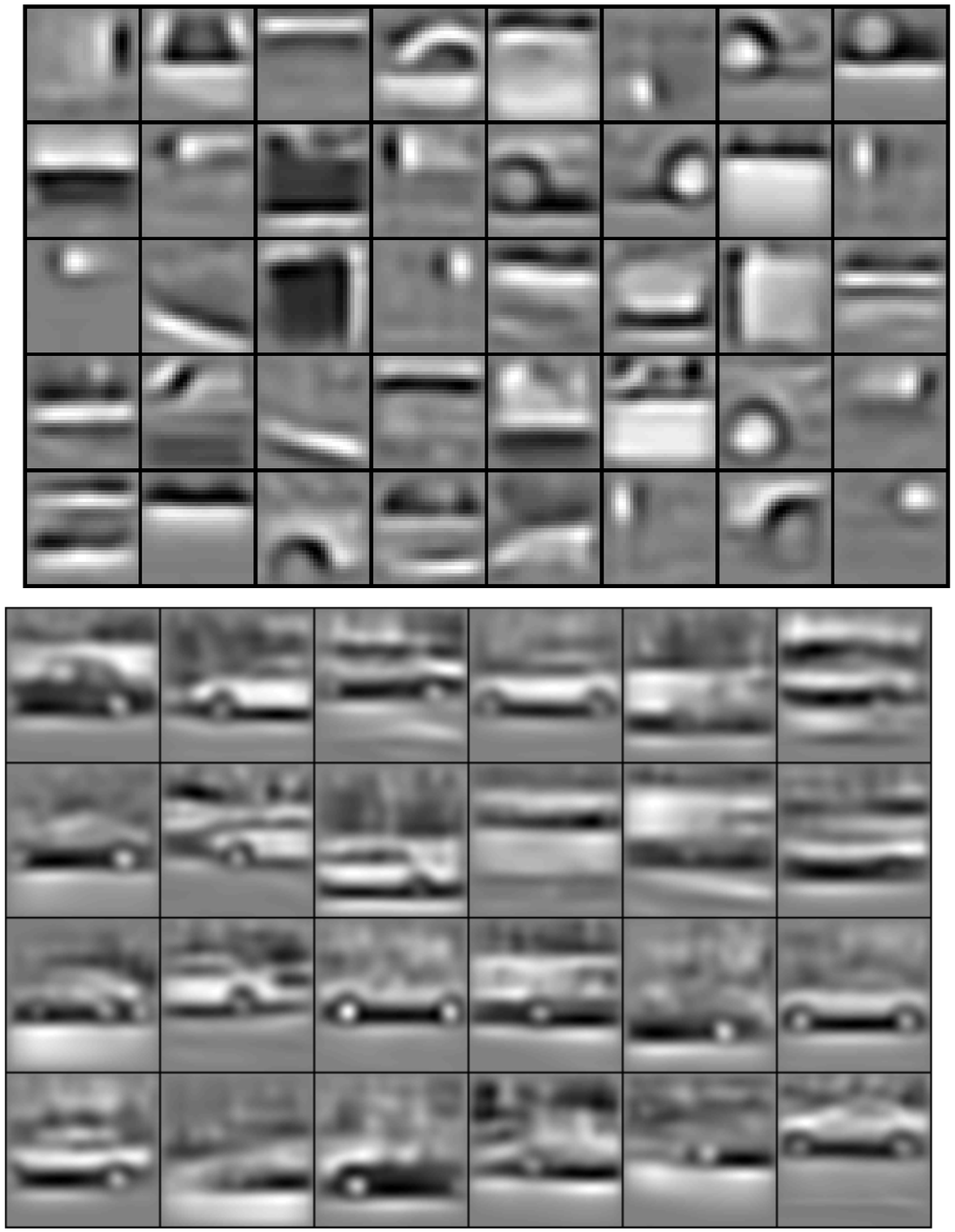


Other Layers

faces



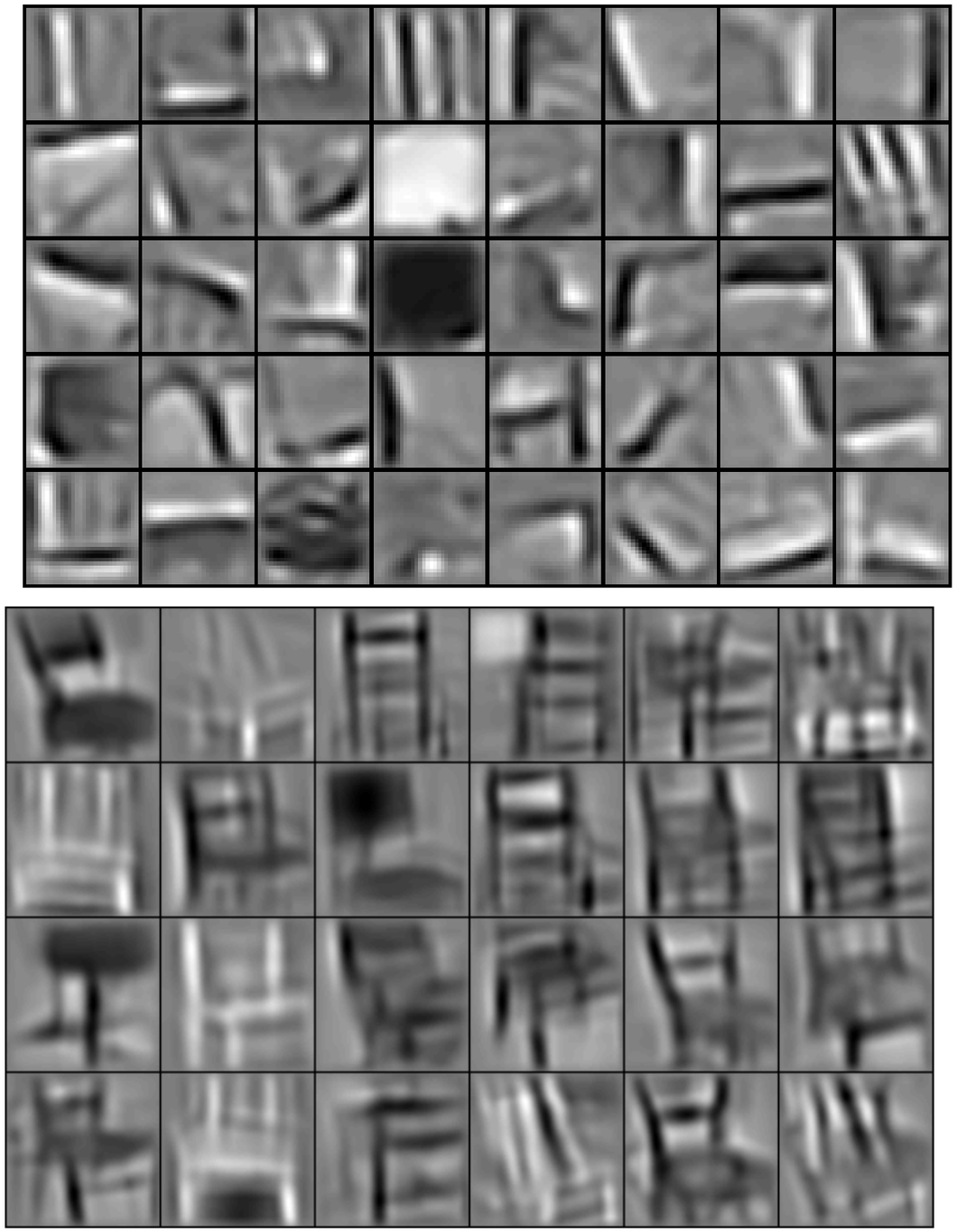
cars



Other Layers

elephants

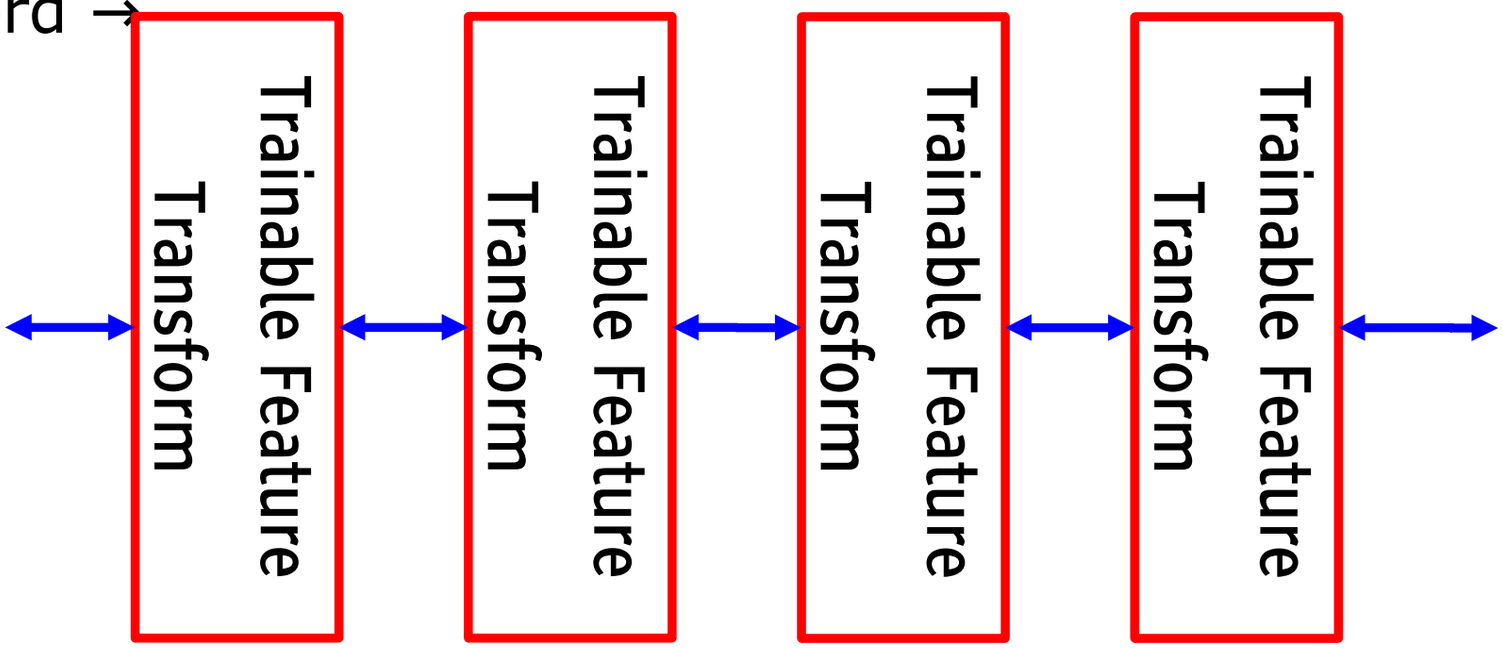
chairs



Trainable Feature Hierarchy

THE UNIVERSITY OF SYDNEY

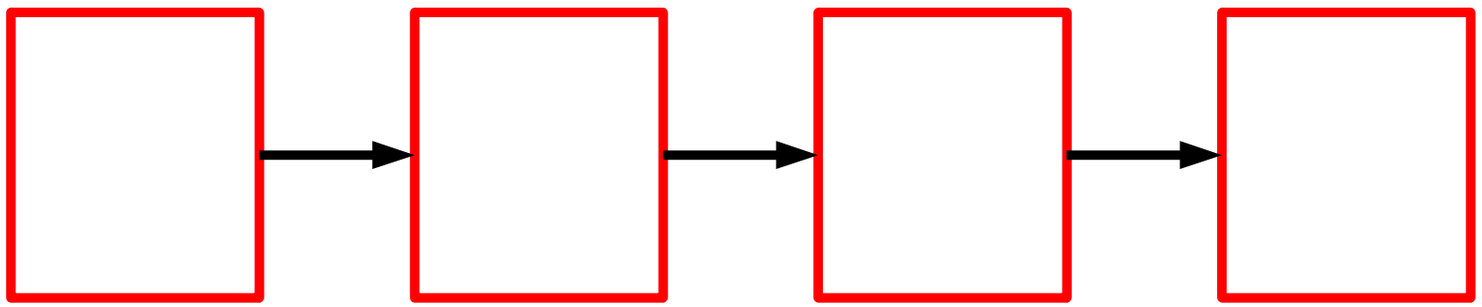
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word →



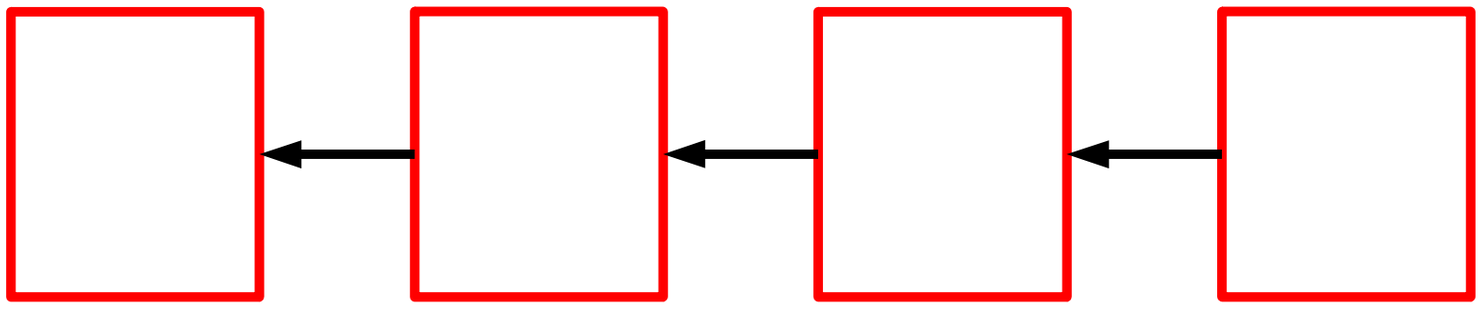
Deep architectures

THE UNIVERSITY OF SYDNEY

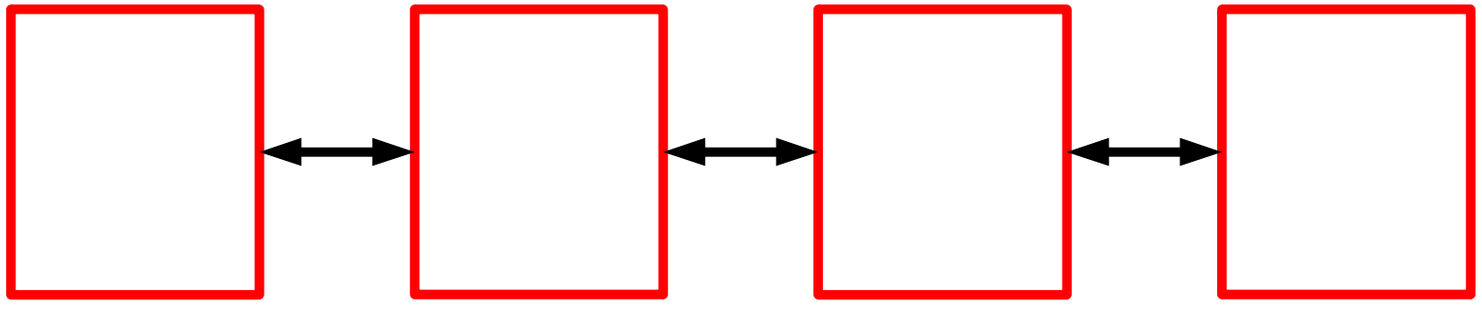
- Feed-Forward: multilayer neural nets, convolutional nets



- Feed-Back: Stacked Sparse Coding, Deconvolutional Nets



- Bi-Drectional: Deep Boltzmann Machines, Stacked Auto-Encoders



Let's try DL in playground

THE UNIVERSITY OF SYDNEY

Iterations 000,000 | Learning rate 0.03 | Activation ReLU | Regularization L2 | Regularization rate 0.001 | Problem type Regression

DATA

Which dataset do you want to use?

Ratio of training to test data: 40%

Noise: 20

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1, X_2, X_1^2, X_2^2, X_1X_2, $\sin(X_1)$, $\sin(X_2)$

3 HIDDEN LAYERS

4 neurons | 4 neurons | 2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.121

Training loss 0.134

Colors shows data, neuron and weight values. -1 0 1

Show test data | Discretize output

Source: http://playground.tensorflow.org/

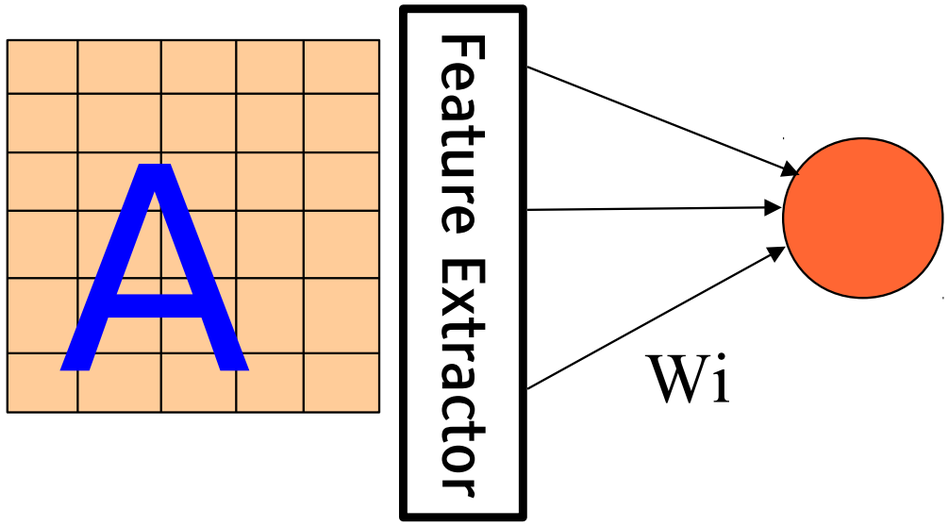
Training protocols

THE UNIVERSITY OF SYDNEY

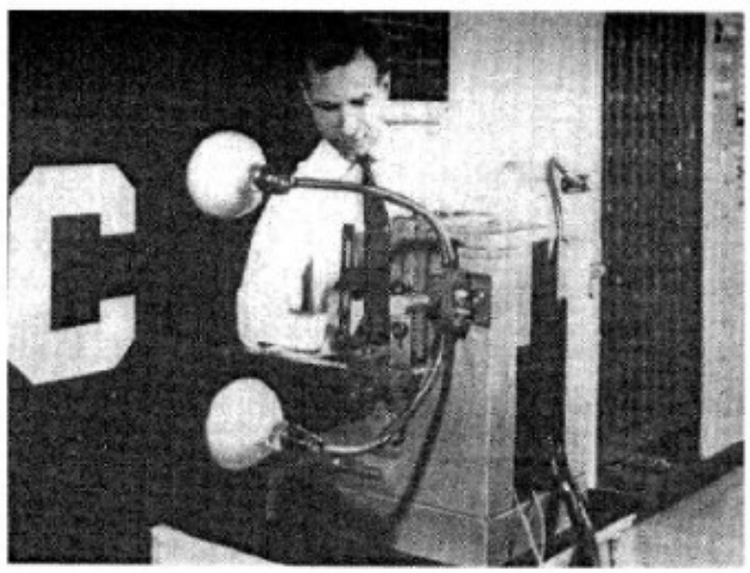
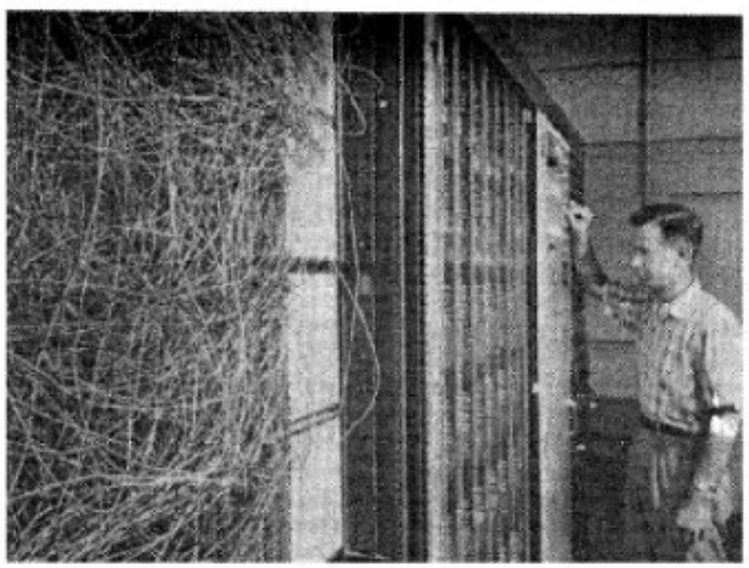
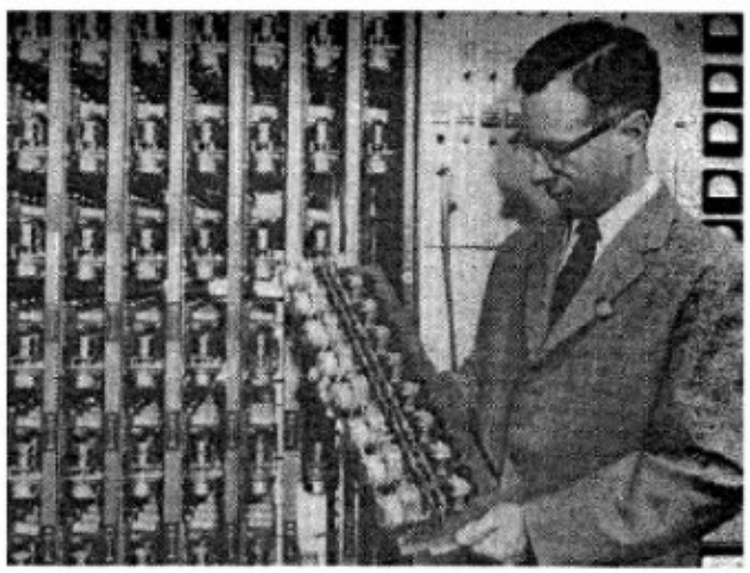
- **Purely Supervised**
  - Initialize parameters randomly
  - Train in supervised mode
    - typically with SGD, using backprop to compute gradients
  - Used in most practical systems for speech and image recognition
- **Unsupervised, layerwise + supervised classifier on top**
  - Train each layer unsupervised, one after the other
  - Train a supervised classifier on top, keeping the other layers fixed
  - Good when very few labeled samples are available
- **Unsupervised, layerwise + global supervised fine-tuning**
  - Train each layer unsupervised, one after the other
  - Add a classifier layer, and retrain the whole thing supervised
  - Good when label set is poor (e.g. pedestrian detection)

Basic element

- The first learning machine: the Perceptron
  - Built at Cornell in 1960
- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



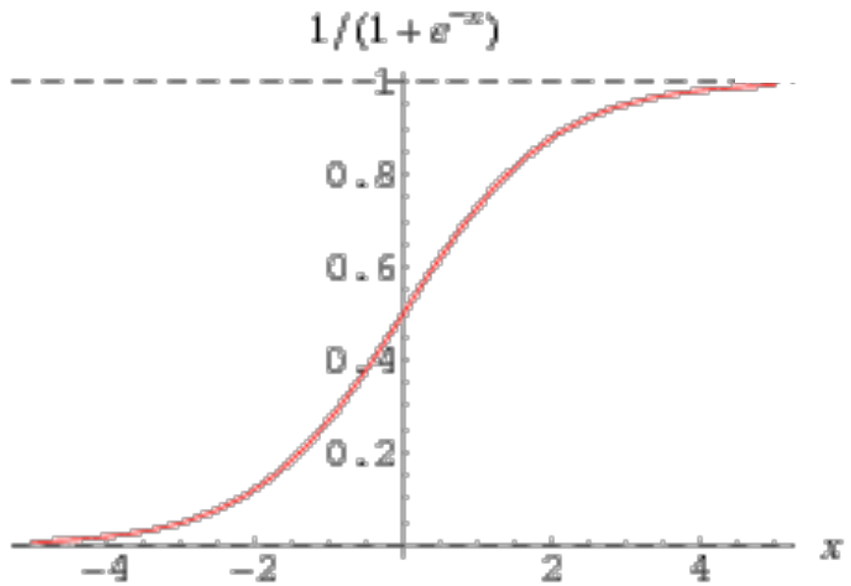
$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

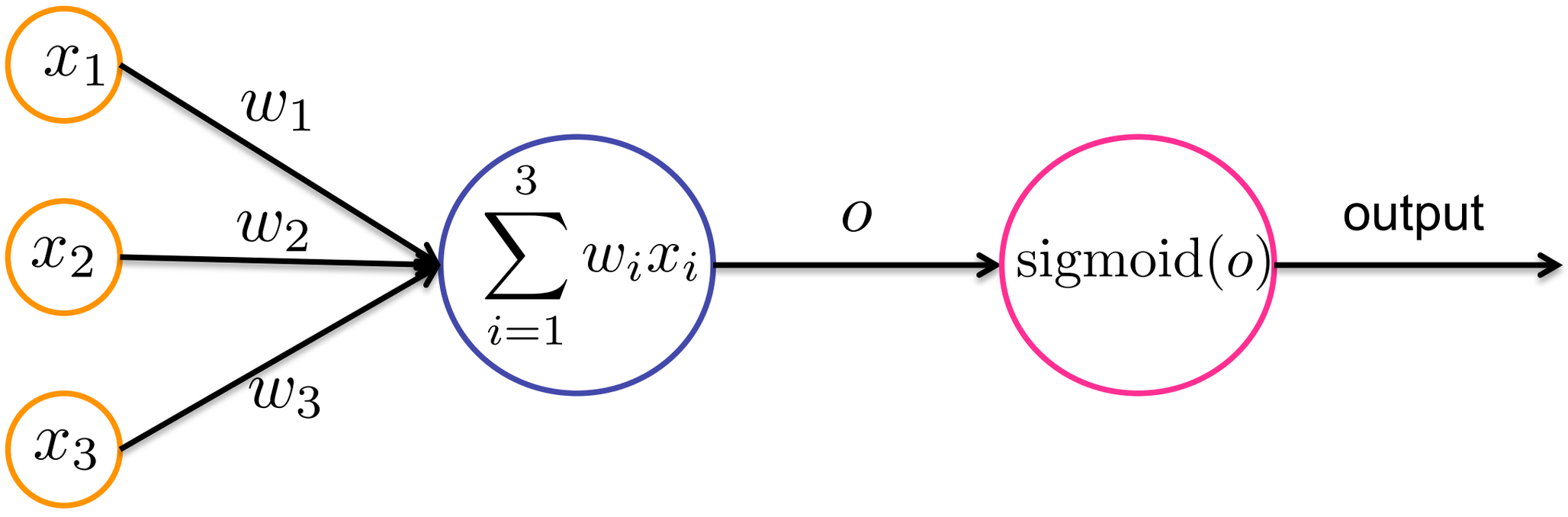




Basic element

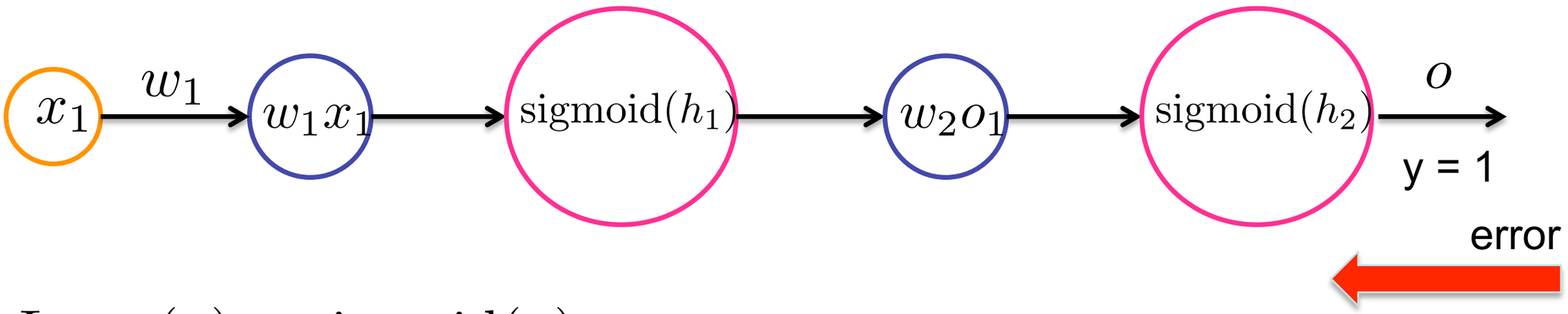
- Sigmoid function

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





Training



Let $\sigma(x) = \text{sigmoid}(x)$

Forward propagation : $o = \sigma(w_2 \sigma(w_1 x_1))$

Loss : $L(w_1, w_2) = -\log o$

$$\frac{\partial}{\partial w_2} L(w_1, w_2) = \boxed{\frac{\partial L}{\partial o} \frac{\partial o}{\partial h_2} \frac{\partial h_2}{\partial w_2}}$$

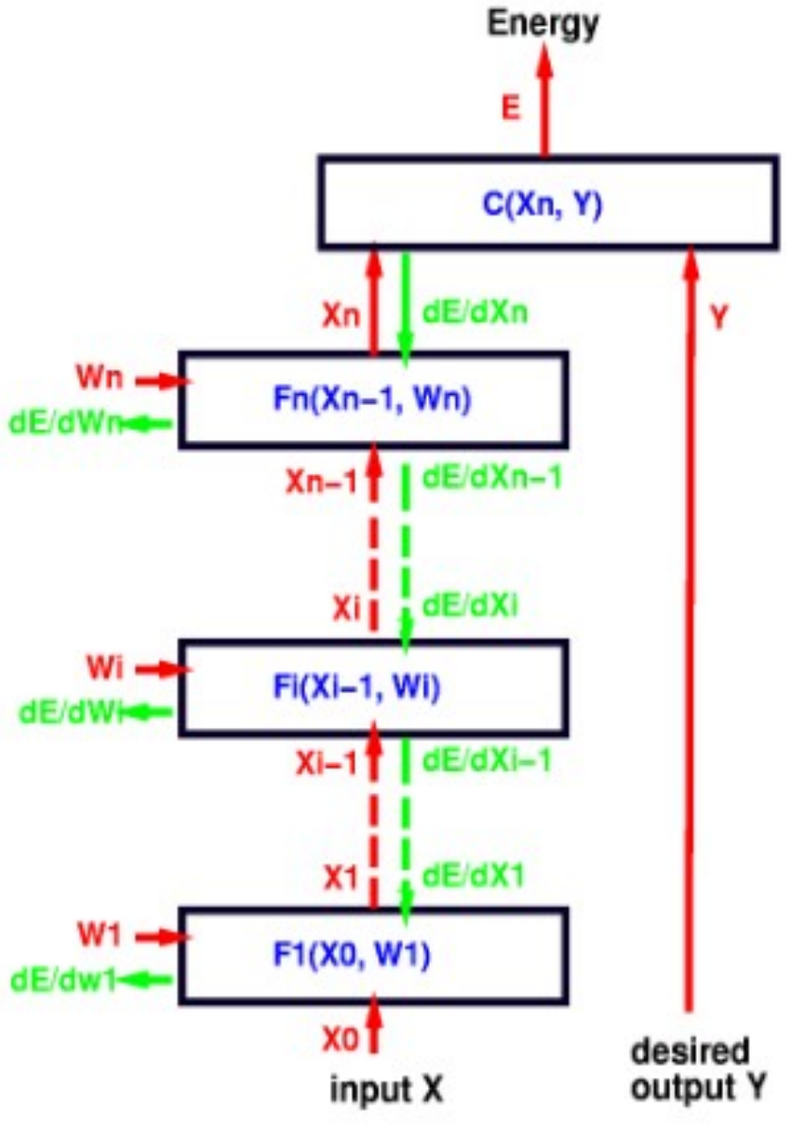
$$\frac{\partial}{\partial w_1} L(w_1, w_2) = \boxed{\frac{\partial L}{\partial o} \frac{\partial o}{\partial h_2} \frac{\partial h_2}{\partial o_1}} \frac{\partial o_1}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Chain Rule

Training with back-propagation

THE UNIVERSITY OF SYDNEY

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

Back-prop in practice

THE UNIVERSITY OF SYDNEY

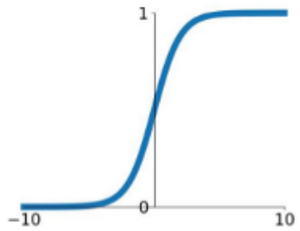
- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use "dropout" for regularization
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- Lots more in [LeCun et al. "Efficient Backprop" 1998]
- Lots, lots more in "Neural Networks, Tricks of the Trade" (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)

Rectified Linear Units

Activation Functions

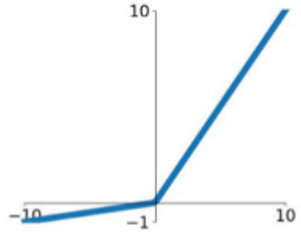
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



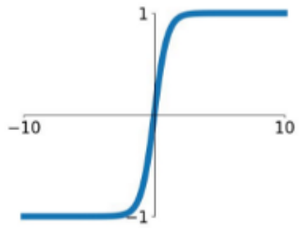
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

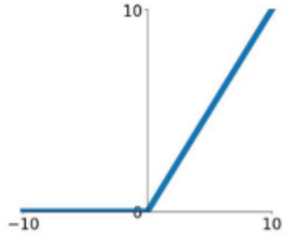


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

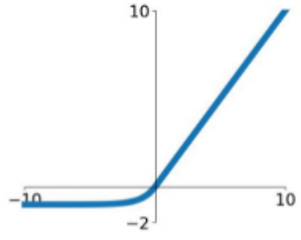
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- *f*(*x*) = Max(0,*x*) More efficient gradient propagation, derivative is 0 or constant, just fold into learning rate
  - Helps *f* '(*net*) issue, but still left with other unstable gradient issues
- More efficient computation: Only comparison, addition and multiplication.
  - Leaky ReLU *f*(*x*) = *x* if *x* > 0 else *ax,* where 0 ≤ *a* <= 1, so that derivate is not 0 and can do some learning for net < 0 (does not “die”).
  - Lots of other variations
- Sparse activation: For example, in a randomly initialised networks, only about 50% of hidden units are activated (having a non-zero output)
- Learning in linear range easier for most learning models

Problems where DL is the best

- Handwriting recognition MNIST (many), Arabic HWX (IDSIA)
- OCR in the Wild [2011]: StreetView House Numbers (NYU and others)
- Traffic sign recognition [2011] GTSRB competition (IDSIA, NYU)
- Pedestrian Detection [2013]: INRIA datasets and others (NYU)
- Volumetric brain image segmentation [2009] connectomics (IDSIA, MIT)
- Human Action Recognition [2011] Hollywood II dataset (Stanford)
- Object Recognition [2012] ImageNet competition
- Scene Parsing [2012] Stanford bgd, SiftFlow, Barcelona (NYU)
- Scene parsing from depth images [2013] NYU RGB-D dataset (NYU)
- Speech Recognition [2012] Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection [2011] MITOS (IDSIA)

Why Deep?

THE UNIVERSITY OF SYDNEY

- **Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"**

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

  - kernel machines (and 2-layer neural nets) are "universal".

- **Deep learning machines**

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - they can represent more complex functions with less "hardware"
- **We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)**

Is there any theory?

THE UNIVERSITY OF SYDNEY

- **Deep Learning involves non-convex loss functions**
  - With non-convex losses, all bets are off
  - Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).

- **But to some of us all "interesting" learning is non convex**
  - Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
  - Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.

Is there any theory?

THE UNIVERSITY OF SYDNEY

intuitively: generalization error = |training error - test error|

- **No generalization bounds?**
  - Actually, the usual VC bounds apply: most deep learning systems have an infinite VC dimension.
  - We don't have tighter bounds than that.
  - But then again, how many bounds are tight enough to be useful for model selection?

- **It's hard to prove anything about deep learning systems**
  - Then again, if we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today.

Some theory emerging

THE UNIVERSITY OF SYDNEY

- **Deep Learning is about representing high-dimensional data**
  - There has to be interesting theoretical questions there
  - What is the geometry of natural signals?
  - Is there an equivalent of statistical learning theory for unsupervised learning?
  - What are good criteria on which to base unsupervised learning?

- **Deep Learning Systems are a form of latent variable factor graph**
  - Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
  - The most interesting deep belief nets have intractable loss functions: how do we get around that problem?

- **Lots of theory at the 2012 IPAM summer school on deep learning**
  - Wright's parallel SGD methods, Mallat's "scattering transform", Osher's "split Bregman" methods for sparse modeling, Morton's "algebraic geometry of DBN",....

An intuitive understanding

THE UNIVERSITY OF SYDNEY

Recall that: intuitively, Generalisation error = |training error - test error|

Intuitively, a learning algorithm is said to be stable if slight perturbations in the training data result in small changes in the output of the algorithm, and these changes vanish as the data set grows bigger and bigger.

Good algorithmic stability implies small generalisation error. (we can gradually change the training examples to the test ones while the output hypotheses will not change much.)

An intuitive understanding

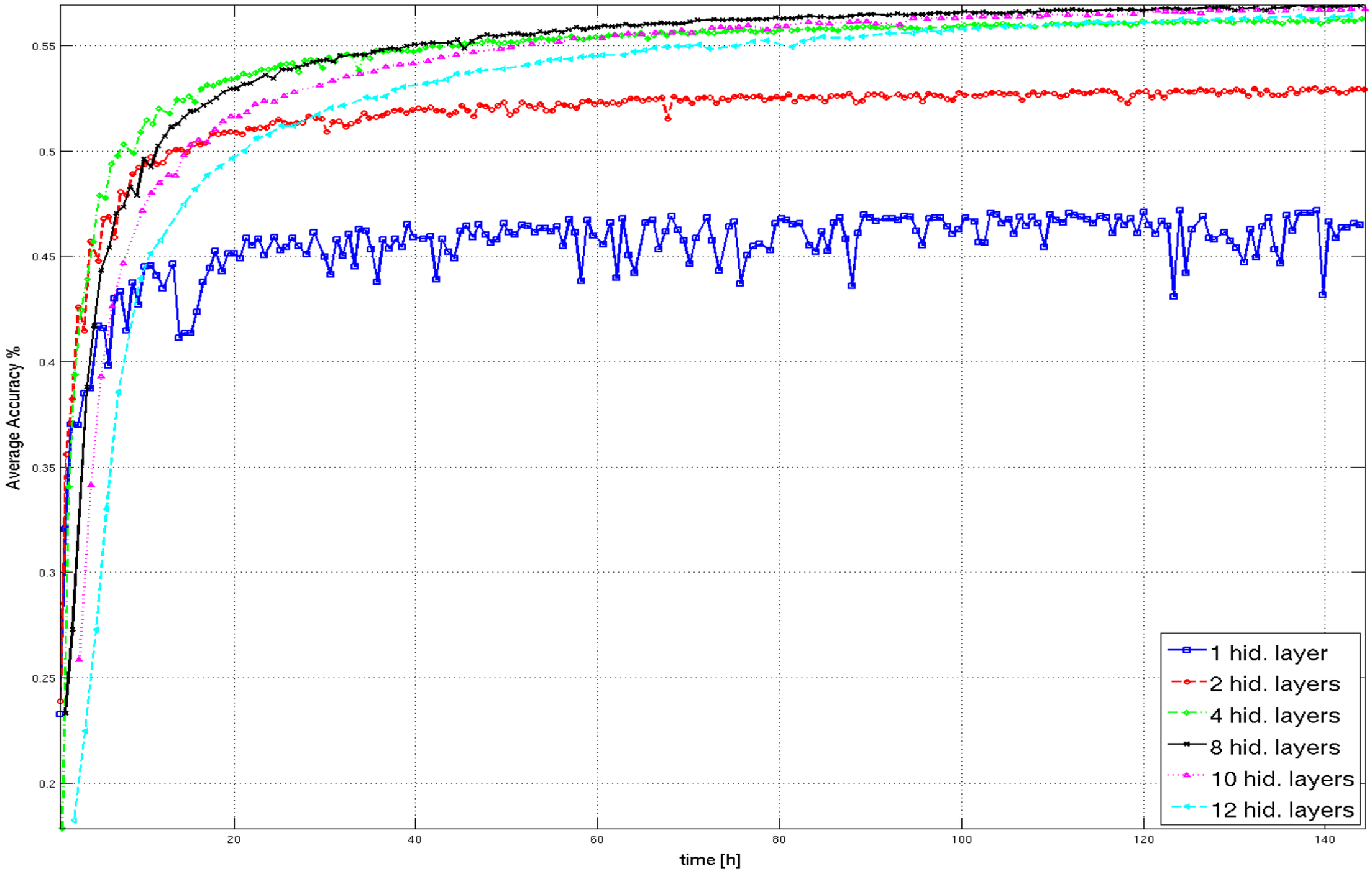
THE UNIVERSITY OF SYDNEY

Deep learning: going deeper, generalising better!

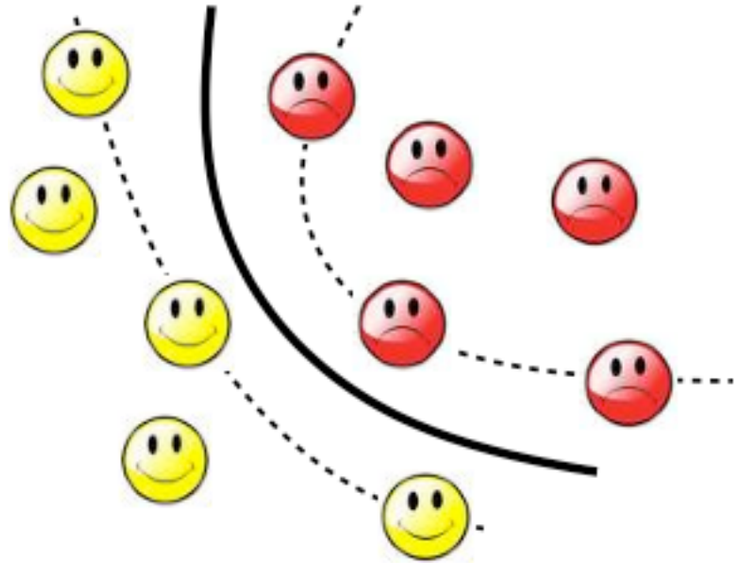
"Deeper" implies
"smaller input and output mutual information" implies
"better" algorithmic stability" implies
"smaller generalisation error".

Speech Recog. Example

THE UNIVERSITY OF SYDNEY



THE UNIVERSITY OF SYDNEY



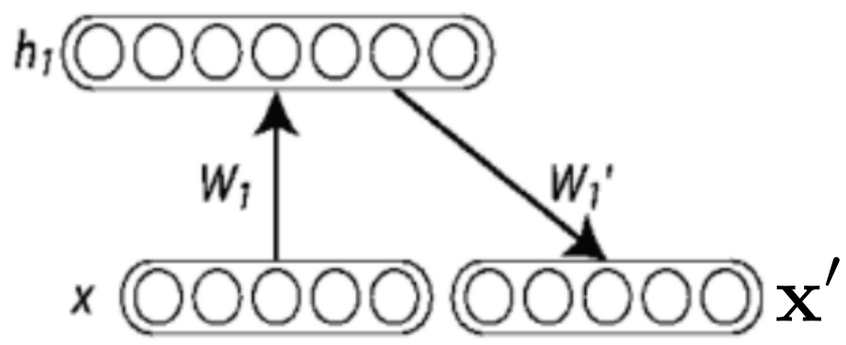
Auto-encoders

Auto-encoder

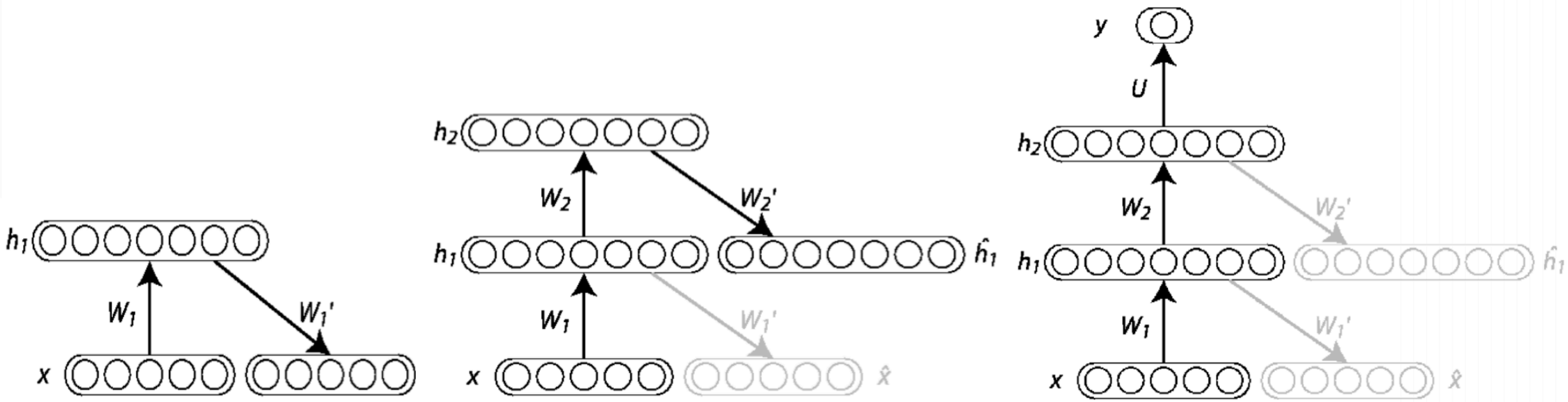
THE UNIVERSITY OF SYDNEY

- Simple Auto-encoder
  - Encoder maps input into hidden representation.
  - Decoder reconstructs input from hidden representation.
- Minimize reconstruction errors.
  - reconstruction = decoder(encoder (input)).
  - Minimize difference between reconstruction and input.
  - E.g.

$$\mathbf{h} = \tanh(\mathbf{b} + \mathbf{W}_1\mathbf{x})$$
$$\text{reconstruction } \mathbf{x}' = \tanh(\mathbf{b}' + \mathbf{W}_1'\mathbf{h})$$
$$\text{Loss } L(\mathbf{x}, \mathbf{x}') = 0.5\|\mathbf{x}' - \mathbf{x}\|^2$$



Stacked auto-encoders

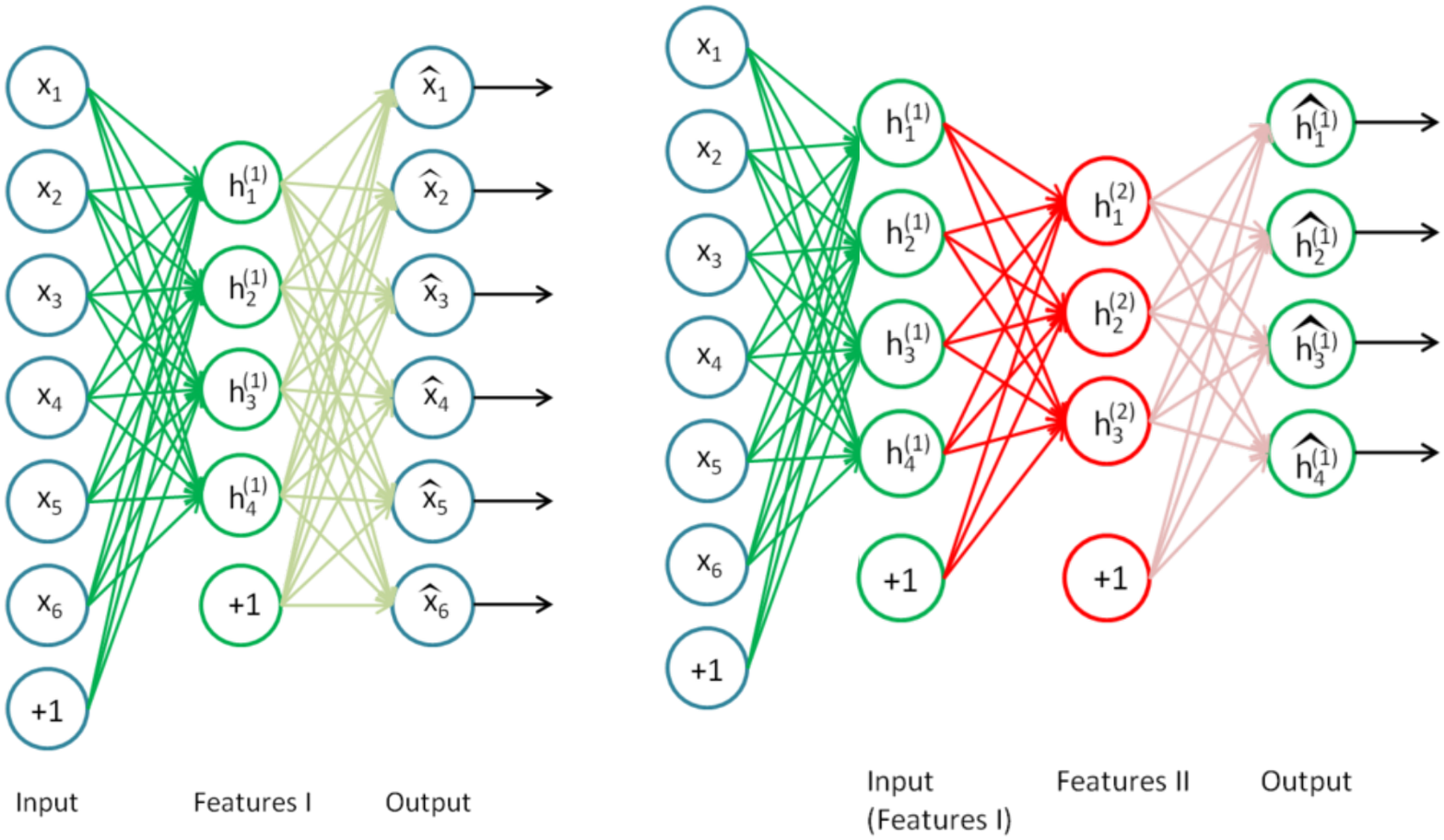


Simple auto-encoders or denoising auto-encoders can be stacked to form deep architectures. The parameters are initialized with unsupervised pre-training. (Bengio et al 2006, Vincent et al 2008)

Stacked auto-encoders

THE UNIVERSITY OF SYDNEY

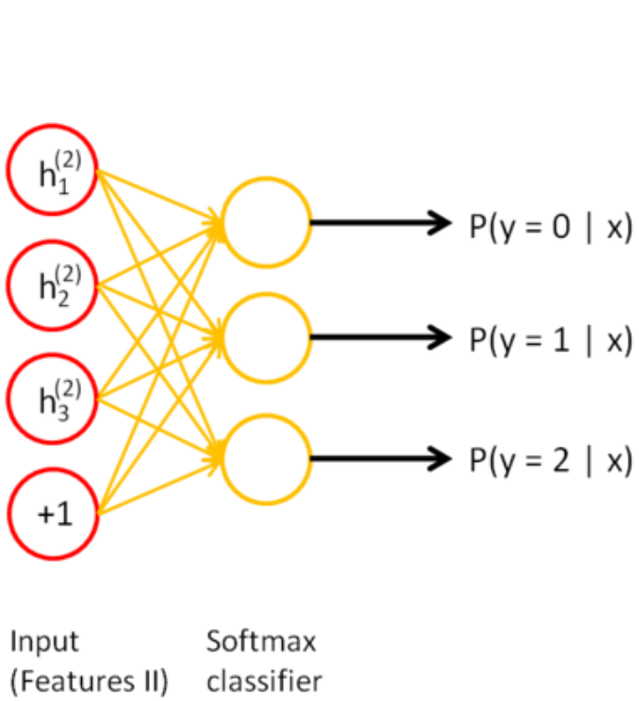
- Bengio (2007) – After Deep Belief Networks (2006)
- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training

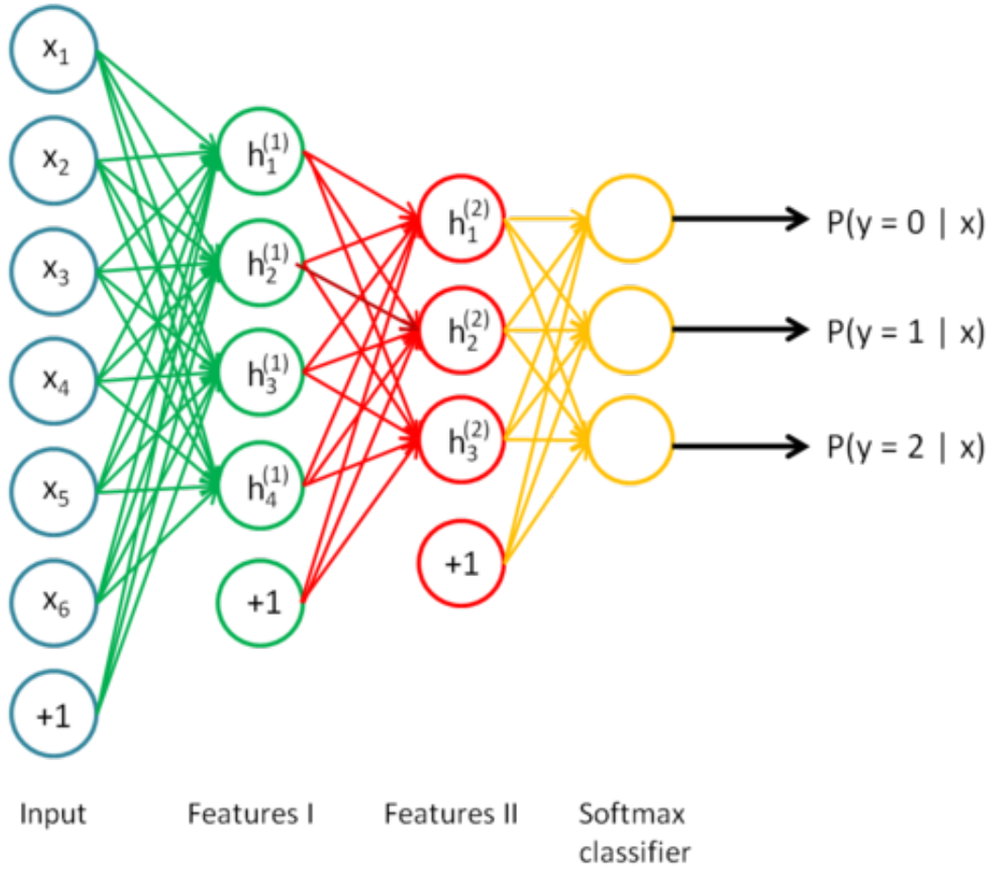


Stacked auto-encoders

THE UNIVERSITY OF SYDNEY

- Do supervised training (can now only used labeled examples) on the last layer using final features
- Then do supervised training on the entire network to fine- tune all weights



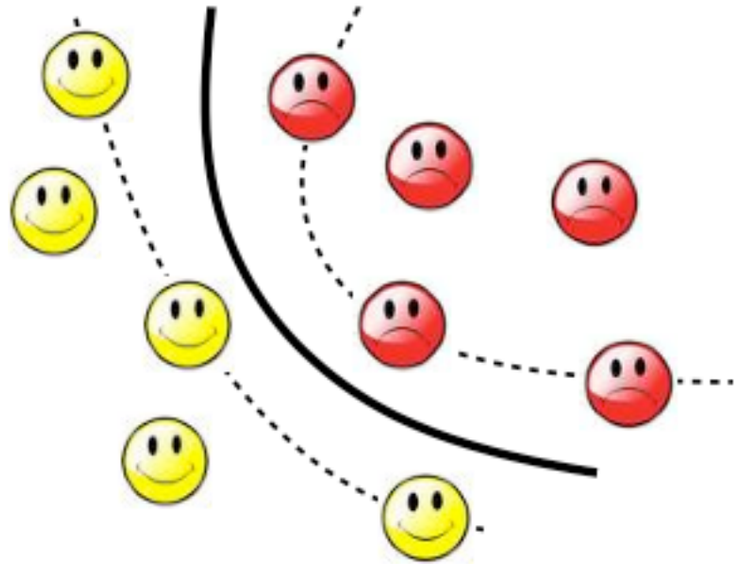


Auto-encoder in practice

- Use more hidden nodes in the encoder

- Use regularisation techniques which encourage sparseness (e.g. a significant portion of nodes have 0 output for any given input)
  - Penalty in the learning function for non-zero nodes
  - Weight decay
  - etc.

- De-noising Auto-Encoder
  - Stochastically corrupt training instance each time, but still train auto-encoder to decode the uncorrupted instance, forcing it to de-noise and learn conditional dependencies within the instance
  - Improved empirical results, handles missing values well

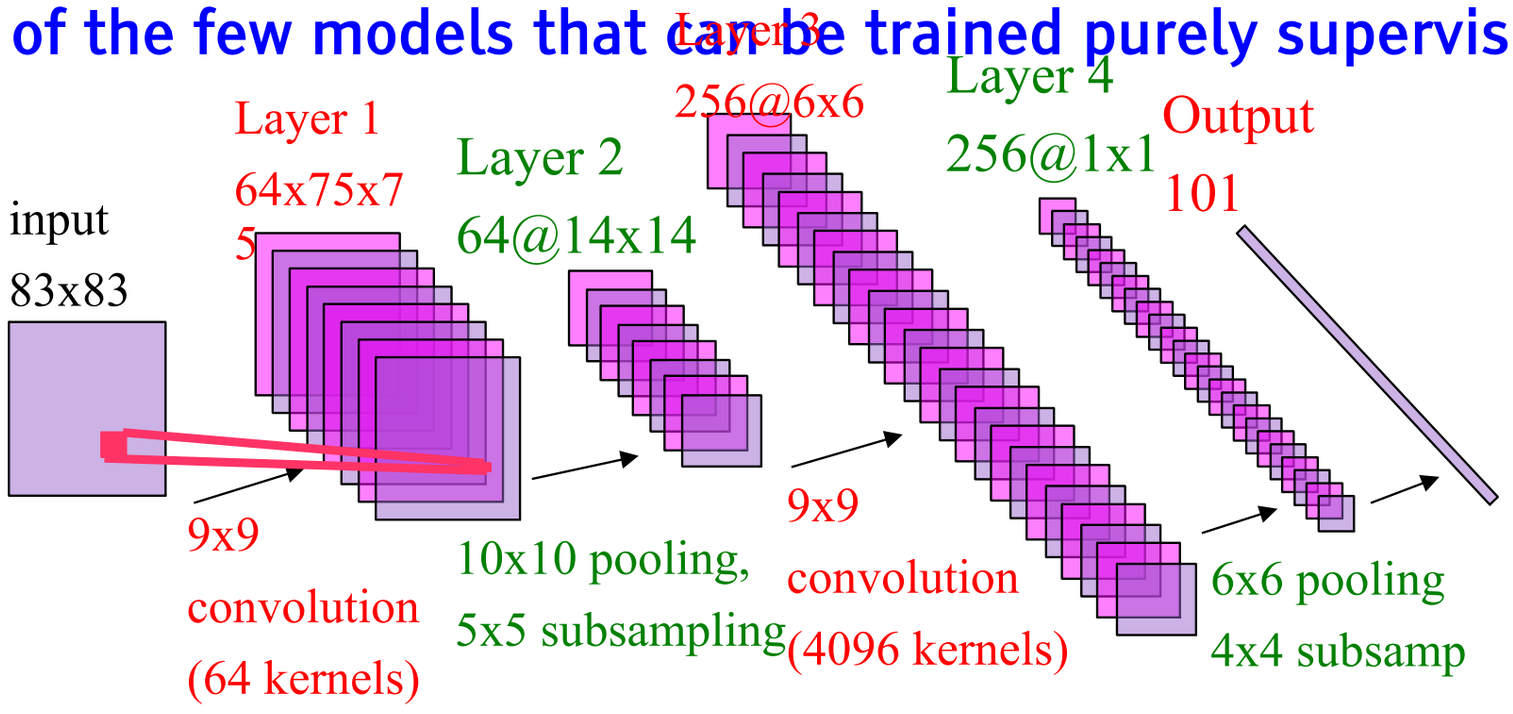


Convolutional Neural Nets

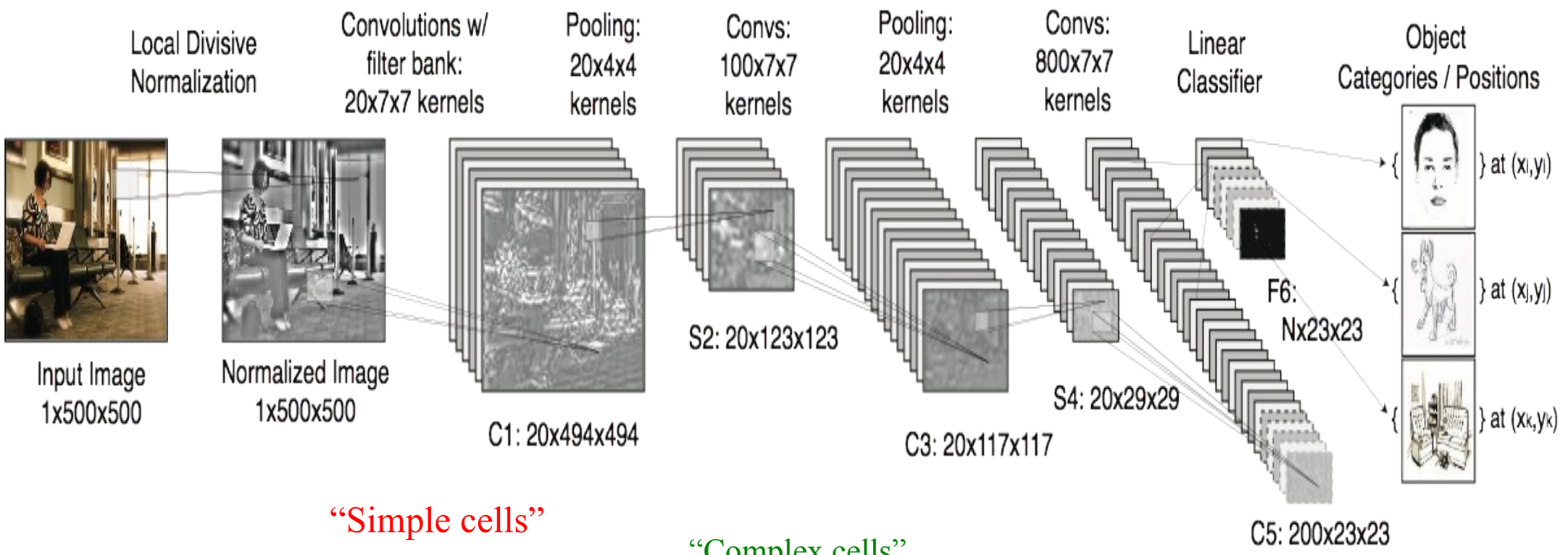
Convolutional Networks

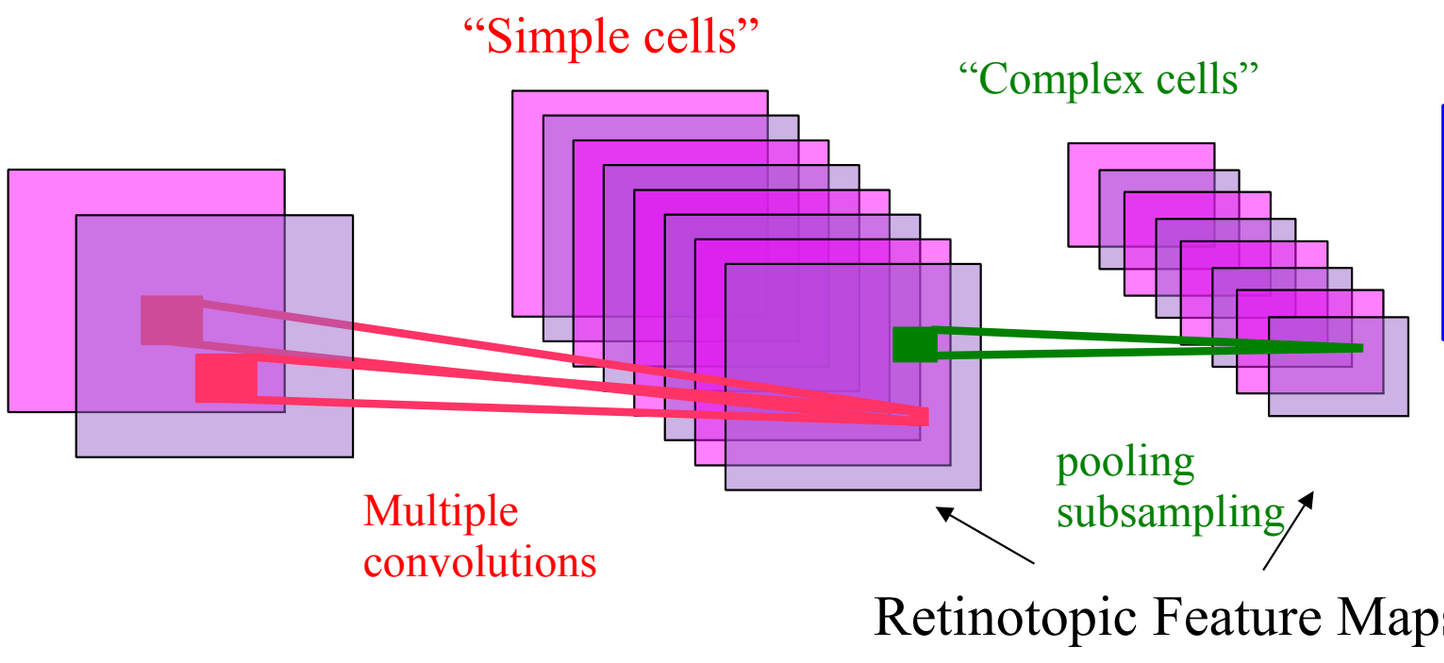
THE UNIVERSITY OF SYDNEY

- **Are deployed in many practical applications**
  - Image recognition, speech recognition, Google's and Baidu's photo taggers
- **Have won several competitions**
  - ImageNet, Kaggle Facial Expression, Kaggle Multimodal Learning, German Traffic Signs, Connectomics, Handwriting....
- **Are applicable to array data where nearby values are correlated**
  - Images, sound, time-frequency representations, video, volumetric images, RGB-Depth images,.....
- **One of the few models that can be trained purely supervised**



CNN Structure





- Training is supervised
- With stochastic gradient descent

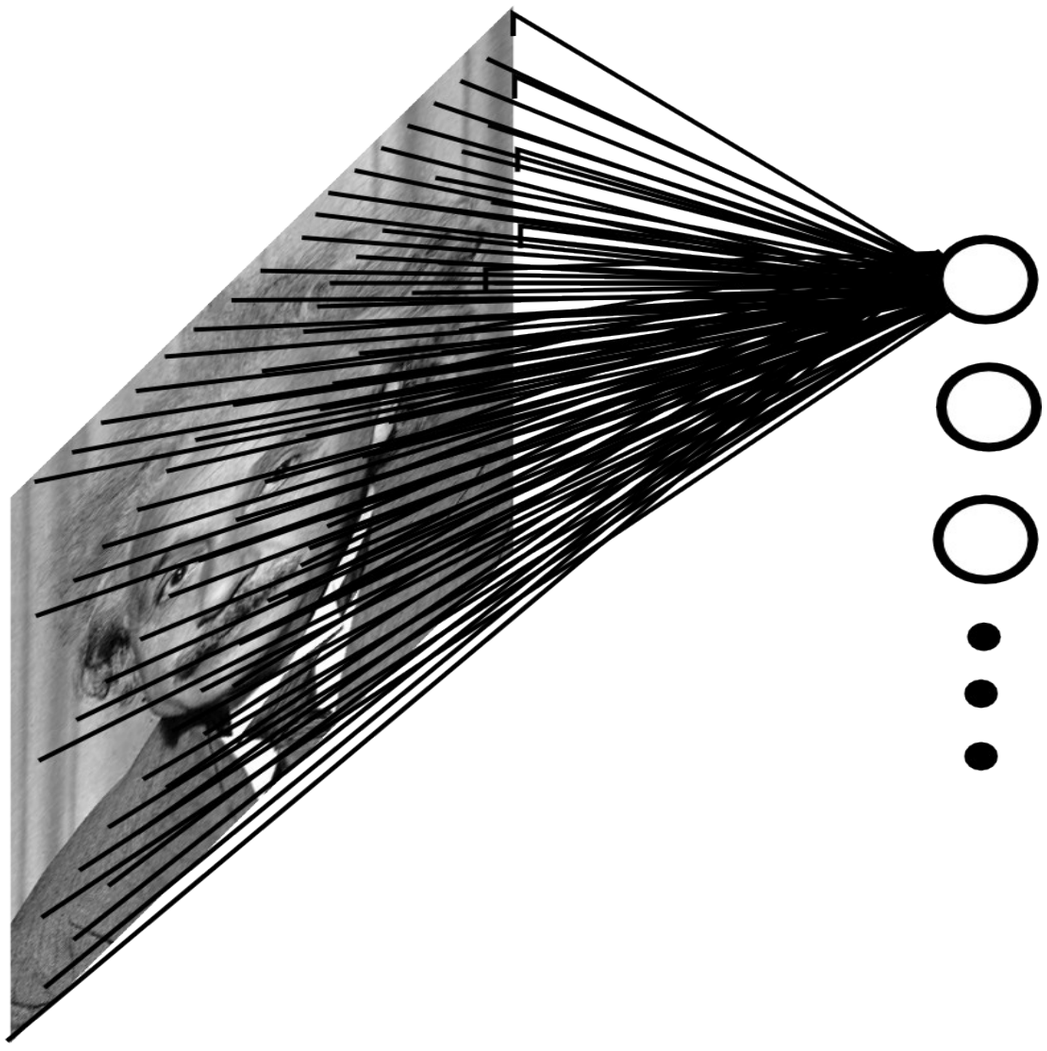
[LeCun et al. 89]

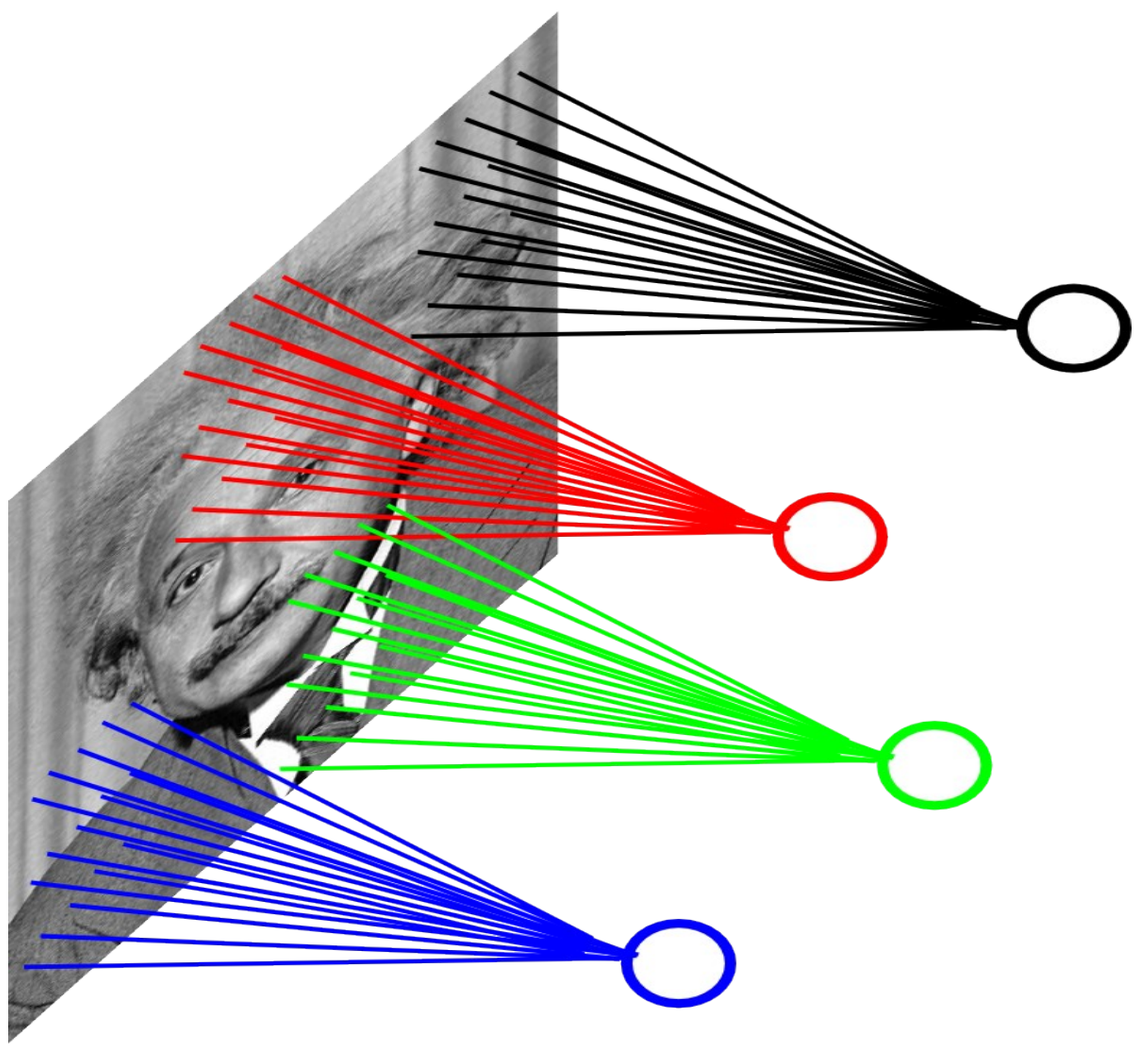
[LeCun et al. 98]

Fully connected

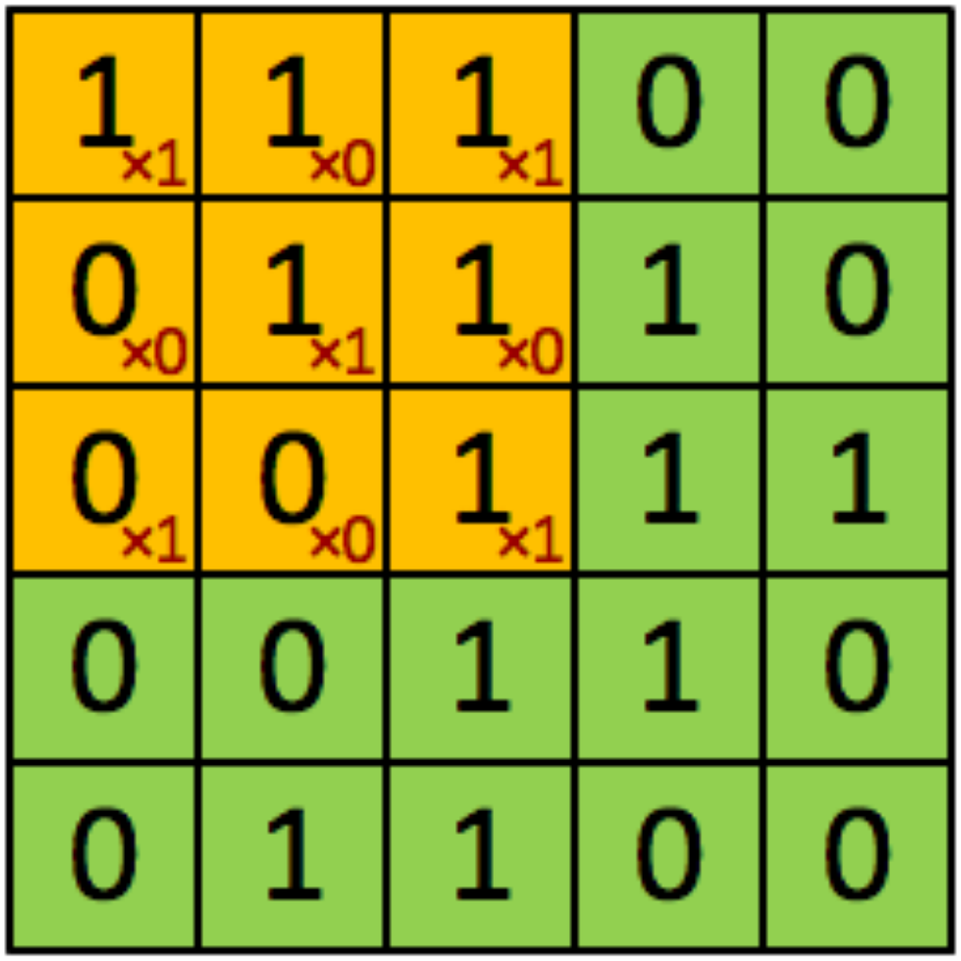
THE UNIVERSITY OF SYDNEY

- Example: 200x200 image
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies





Convolution example



Image

1	0	1
0	1	0
1	0	1

Convolved Feature

Shared Weights

THE UNIVERSITY OF SYDNEY

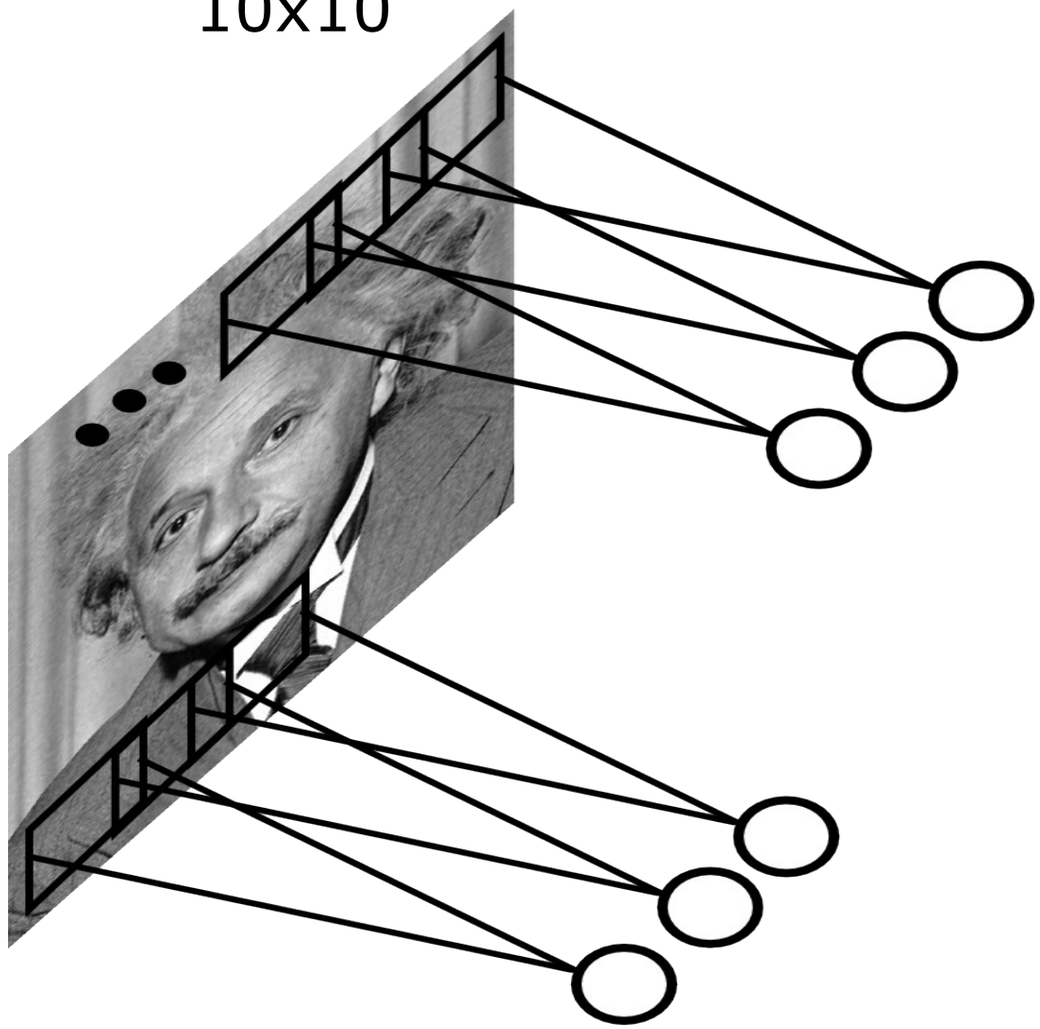
- Features that are useful on one part of the image and probably useful elsewhere.
- All units share the same set of weights
- Shift equivariant processing:
  - When the input shifts, the output also shifts but stays otherwise unchanged.
- Convolution
  - with a learned kernel (or filter)
  - Non-linearity: ReLU (rectified linear)

$$A_{ij} = \sum_{kl} W_{kl} X_{i+j.\,k+l}$$

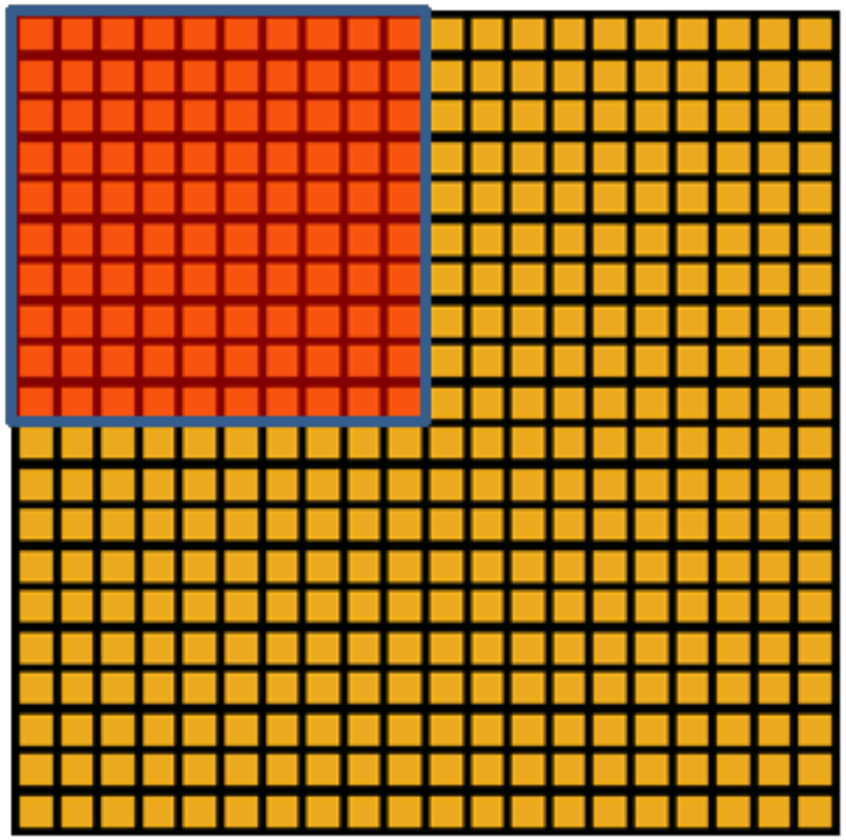
- The filtered "image" Z is called a feature map

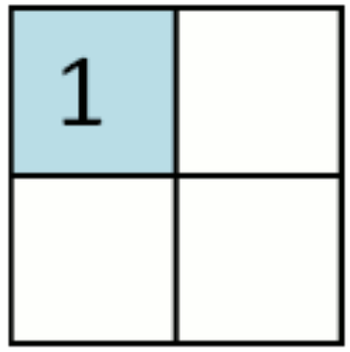
$$Z_{ij} = max(0, A_{ij})$$

- Example: 200x200 image
  - 400,000 hidden units with 10x10 fields = 1000 params
  - 10 feature maps of size 200x200, 10 filters of size 10x10



Pooling example





Convolved feature

Pooled feature

Examples

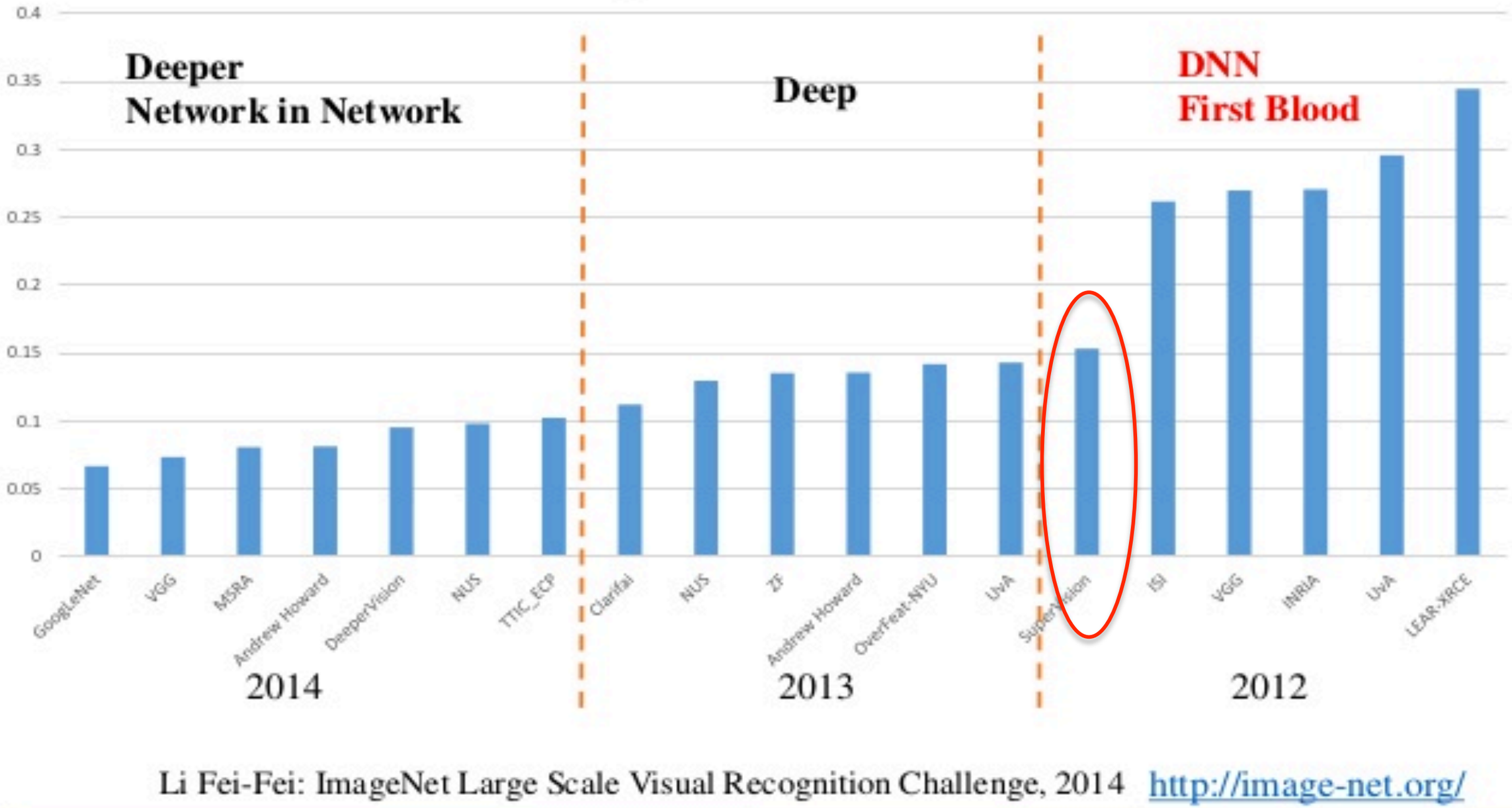
THE UNIVERSITY OF SYDNEY

- **Traffic Sign Recognition (GTSRB)**
  - German Traffic Sign Reco Bench
  - 99.2% accuracy

- **House Number Recognition (Google)**
  - Street View House Numbers
  - 94.3 % accuracy

ILSVRC

ImageNet Classification error throughout years and groups

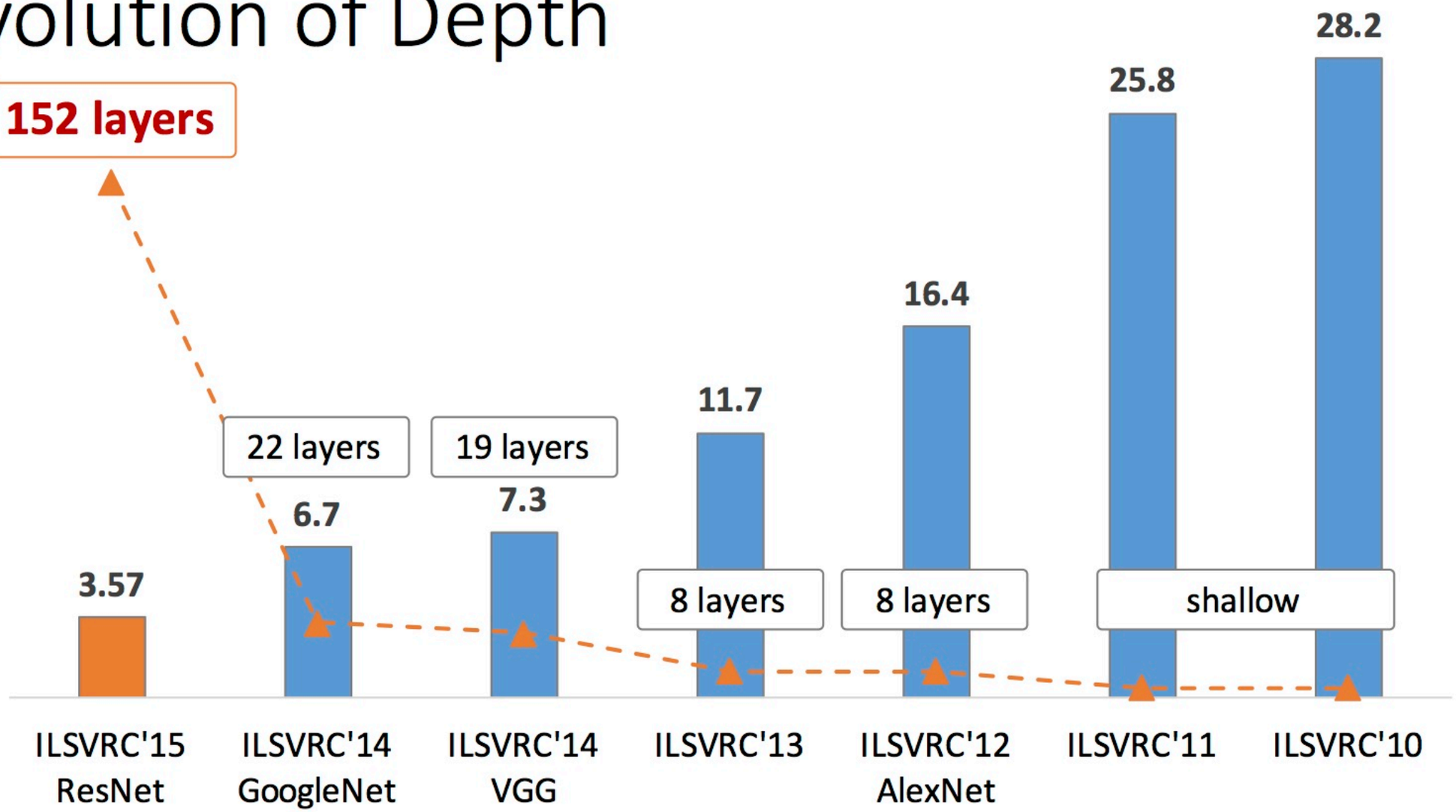


Li Fei-Fei: ImageNet Large Scale Visual Recognition Challenge, 2014 http://image-net.org/

Depth vs Error

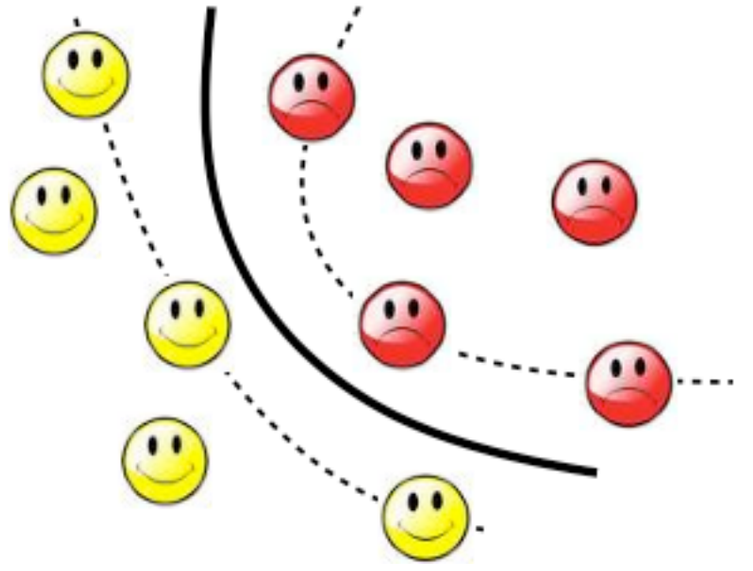
THE UNIVERSITY OF SYDNEY

Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR

THE UNIVERSITY OF SYDNEY



Making DL work

Vanishing gradients

THE UNIVERSITY OF SYDNEY

- Analyzing the norms of the Jacobians, yields:

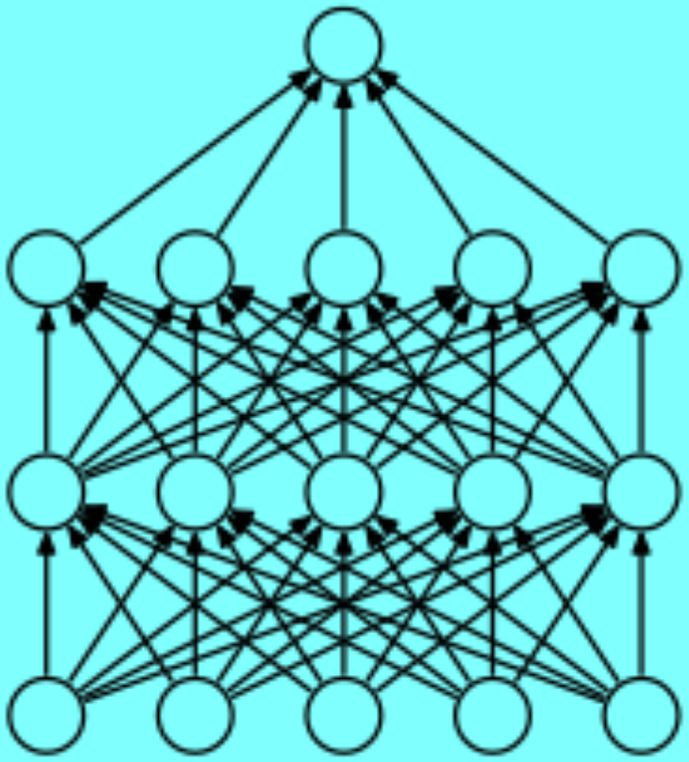
$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\text{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

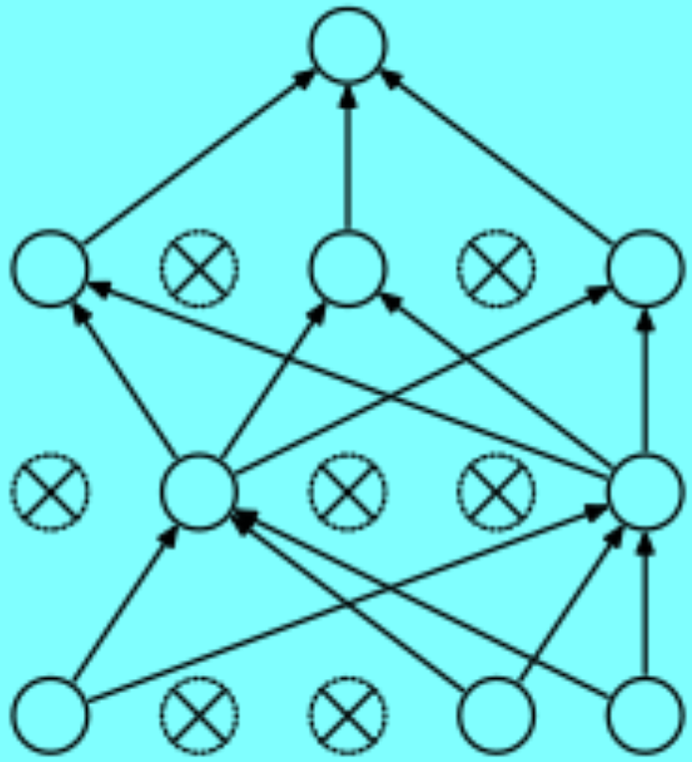
$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- For each instance drop a node (hidden or input) and its connections with probability *p* and train
- Final net just has all averaged weights (actually scaled by 1-*p*)
- As if ensembling 2^n different network substructures

Batch normalisation (1)

- Rather than just guess initial parameters to maintain learning balance, why not renormalise activations at each layer to maintain balance
- Since we like 0-mean and unit variance inputs (standard 1st layer normalisation), we can just re-normalise the activation/net values at each input dimension *k* at each layer

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Want mean and variance of that activation for the entire data set. Approximate the empirical mean and variance over a mini-batch of instances and then normalise the activation
- Then scale and shift the normalised activation with 2 learnable weights per input, γ and β, to attain the final batch normalisation for that activation

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Batch normalisation (2)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[Ioffe and Szegedy, 2015]

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

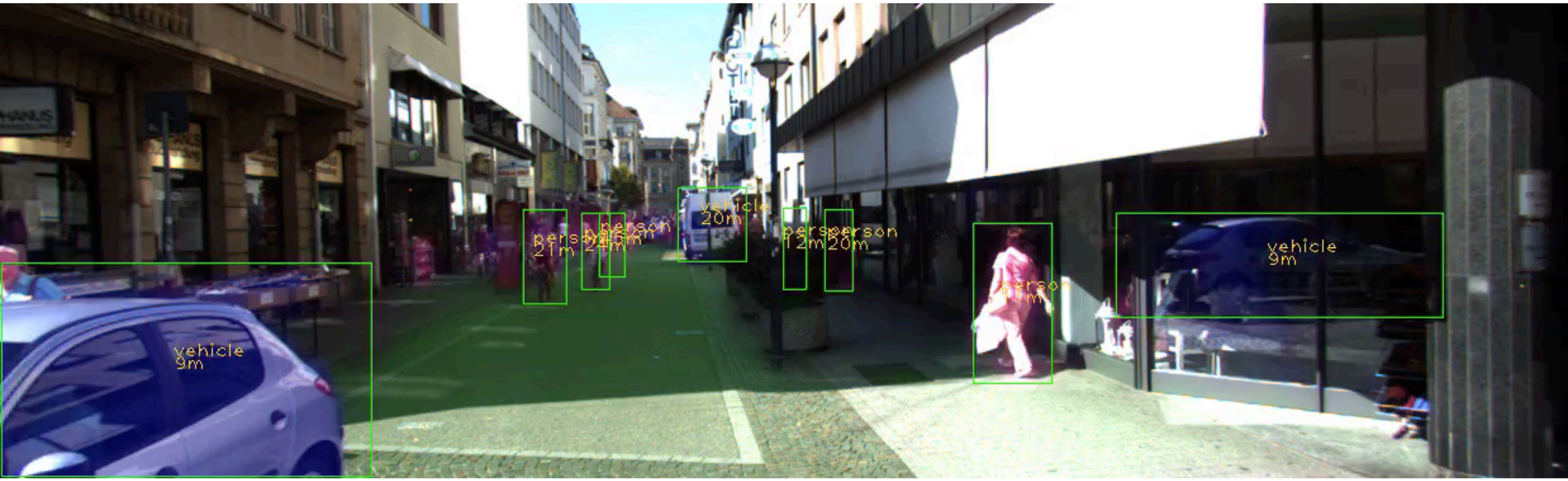
Allows larger LR, improves gradient flow and reduces dependence on initialisation

When to try deep learning

- There is lots of data
- The data is high dimensional
- The data has an underlying structure
- You have good GPUs

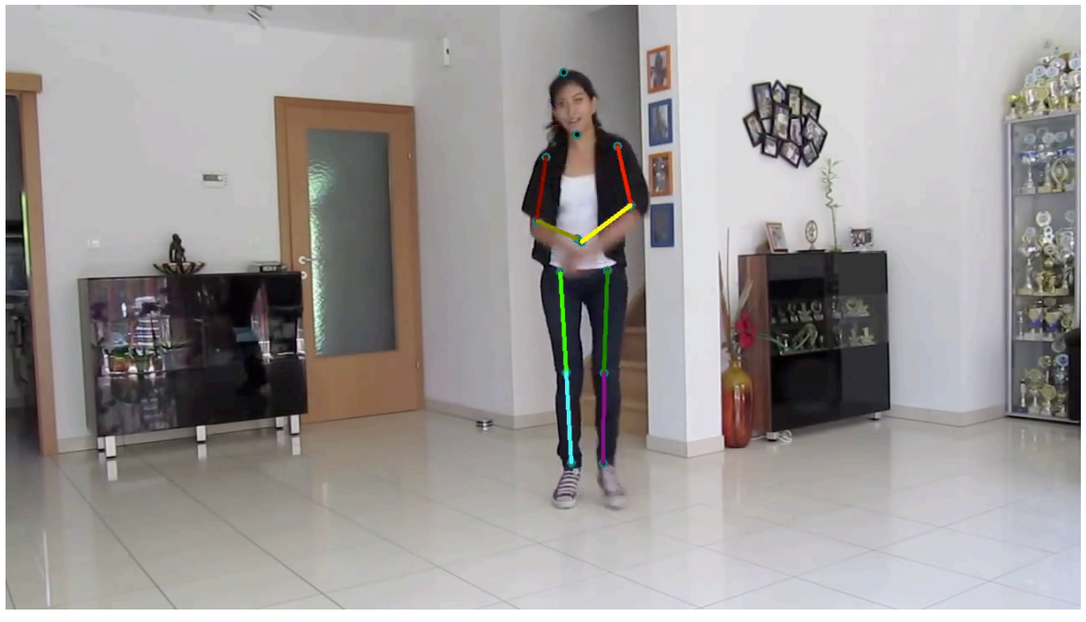
Real-time object recognition

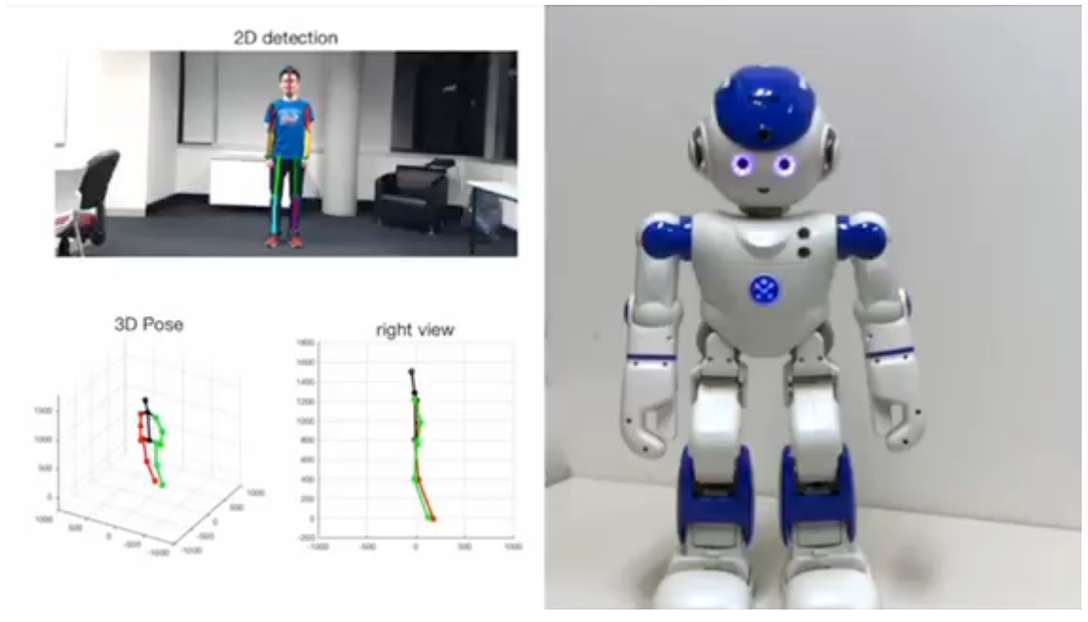
THE UNIVERSITY OF SYDNEY



Deep pose estimation

THE UNIVERSITY OF SYDNEY





Beyond Autoencoders and CNNs

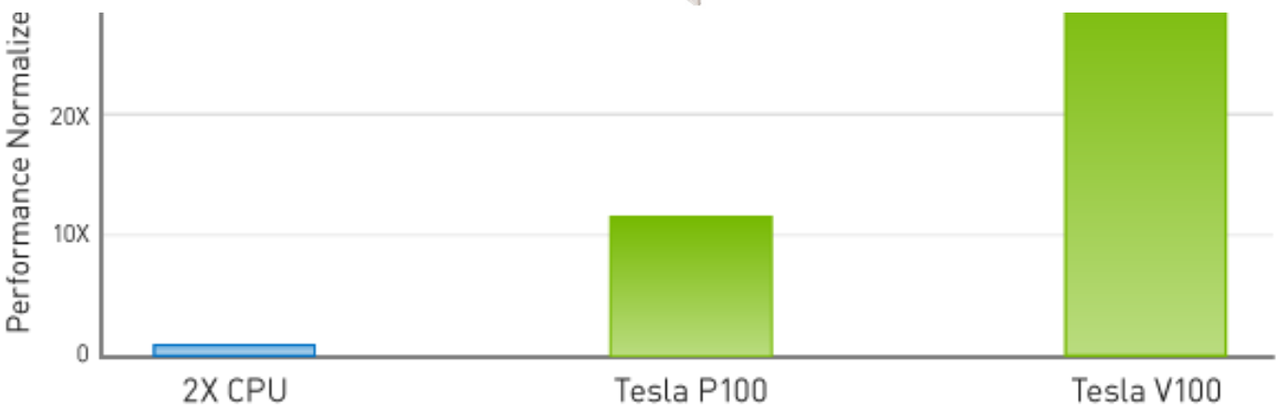
- Deep Belief Networks
- Restricted Boltzman Machines
- Variational Bayesian Autoencoders
- Convolutional Autoencoders
- Recurrent Neural Nets
- Variational Bayesian CNNs
- Convolutional Kernel Networks
- Residual networks (ResNets)
- Inception networks
- Generative adversarial networks (GANs)

Very powerful GPUs

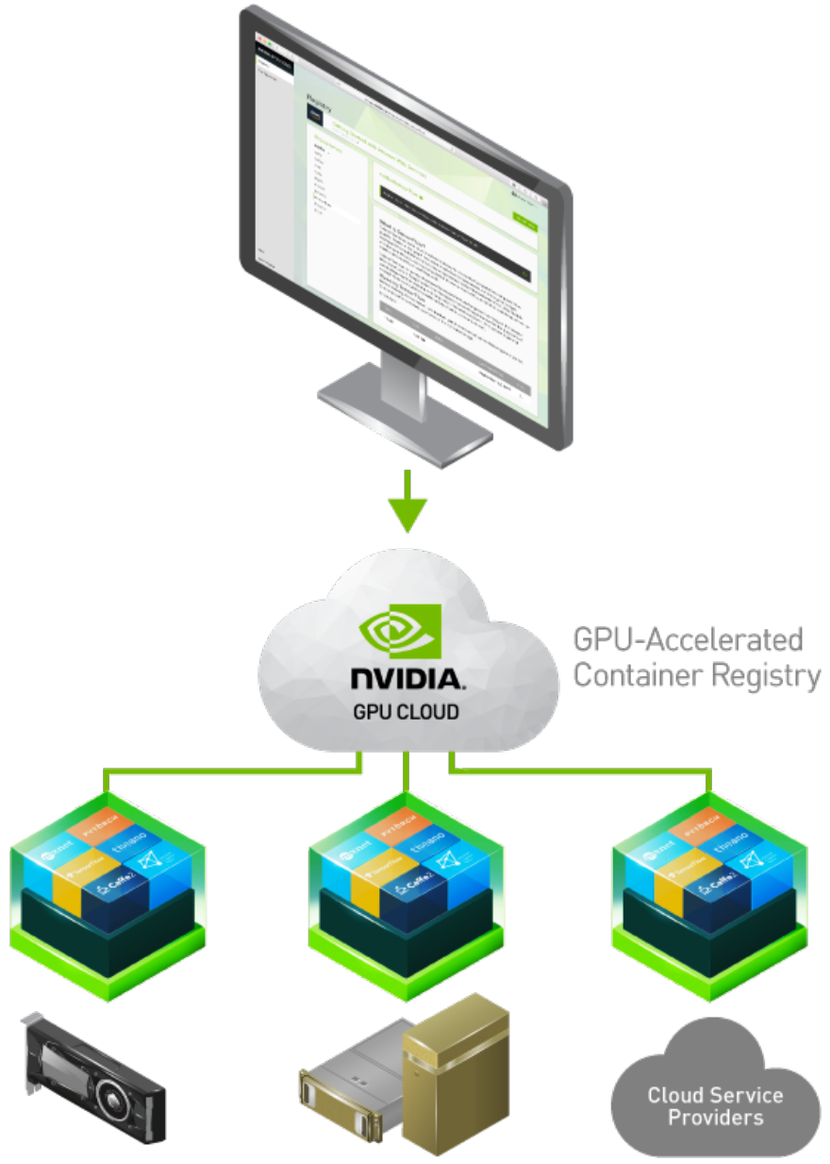
NVIDIA TESLA V100 FOR NVLINK

Ultimate performance for deep learning.

THE UNIVERSITY OF SYDNEY



Workload: ResNet-50 | CPU: 2X Xeon E5-2660 v4, 2GHz | GPU: add 1X Tesla P100 or V100 at 150W | V100 measured on pre-production hardware.



Software

THE UNIVERSITY OF SYDNEY

- TensorFlow
- PyTorch
- Theano
- Keras
- Caffe
- Lasagne

http://deeplearning.net/software_links/

Resources

THE UNIVERSITY OF SYDNEY

- Deep learning website: http://deeplearning.net
- Deep learning book: http://www.deeplearningbook.org/
- Deep learning for NLP tutorial: http://www.socher.org/index.php/DeepLearningTutorial/DeepLearningTutorial
- Neural networks tutorials: http://www.coursera.org/course/neuralnets