Deep Learning

Yee Whye Teh (Oxford Statistics & DeepMind)
https://www.stats.ox.ac.uk/~teh

https://github.com/OxfordAIML/uniqplus-aiml-2022

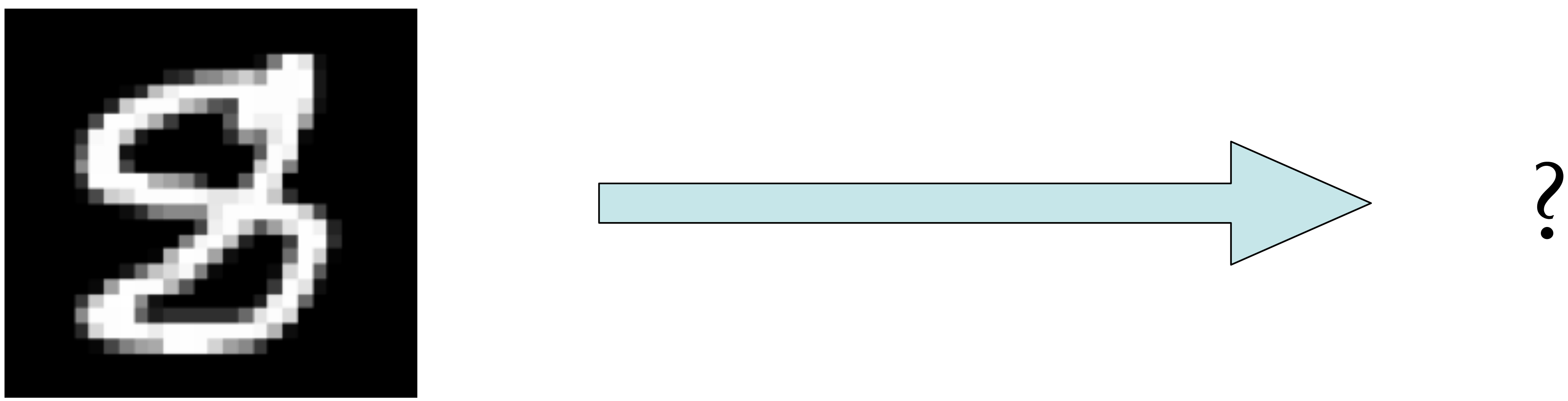
Some slides pinched from DeepMind UCL Deep Reinforcement Learning course and NIPS 2017 tutorial by Reed, Vinyals, de Freitas.

Artificial Intelligence

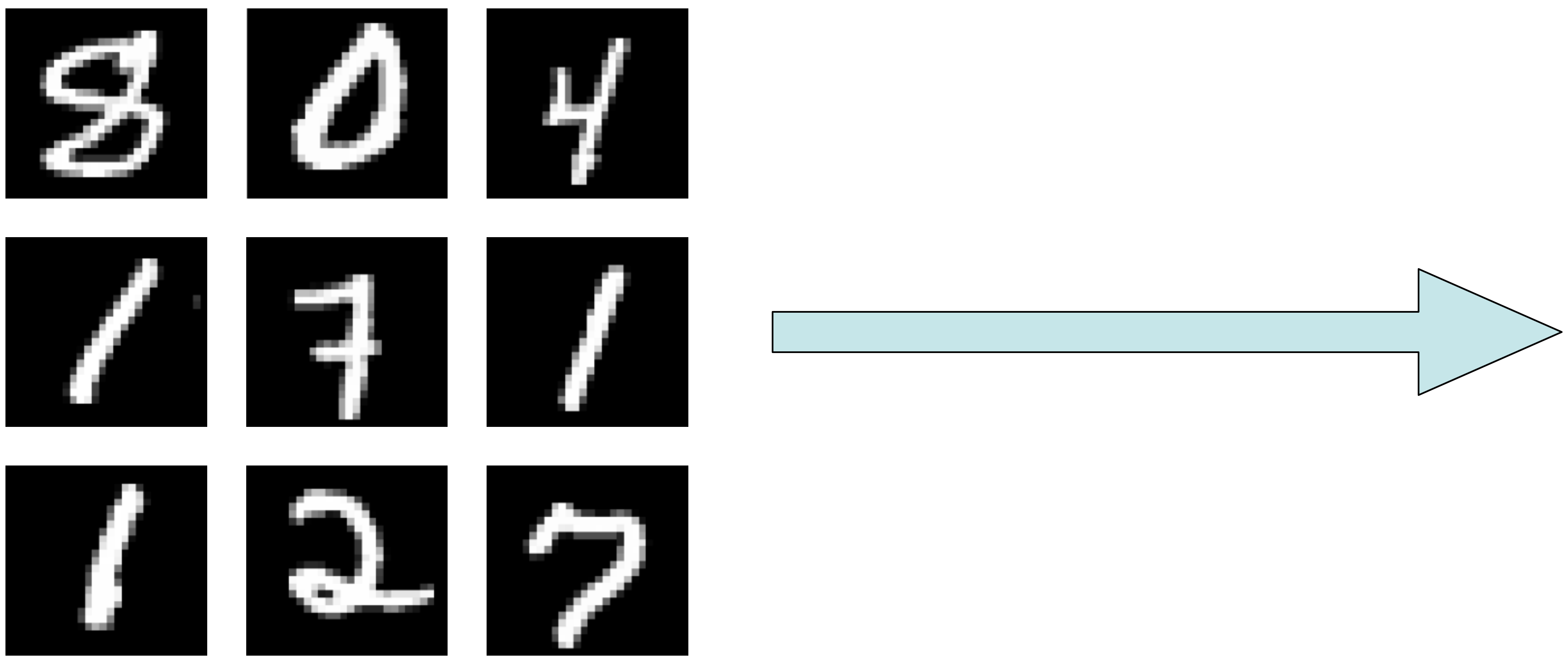
- AI problems are difficult and complex.
  - Impossible to manually programme explicit solutions.

- Modern deep learning approach developed to tackle this difficulty and complexity.
  - We “programme” a solution space by specifying **neural network architecture** and **objective** function.
  - The system then searches in solution space by optimizing (**learning**) on **large data sets**, taking advantage of **modern computing hardware**.

Deep Learning



Deep Learning

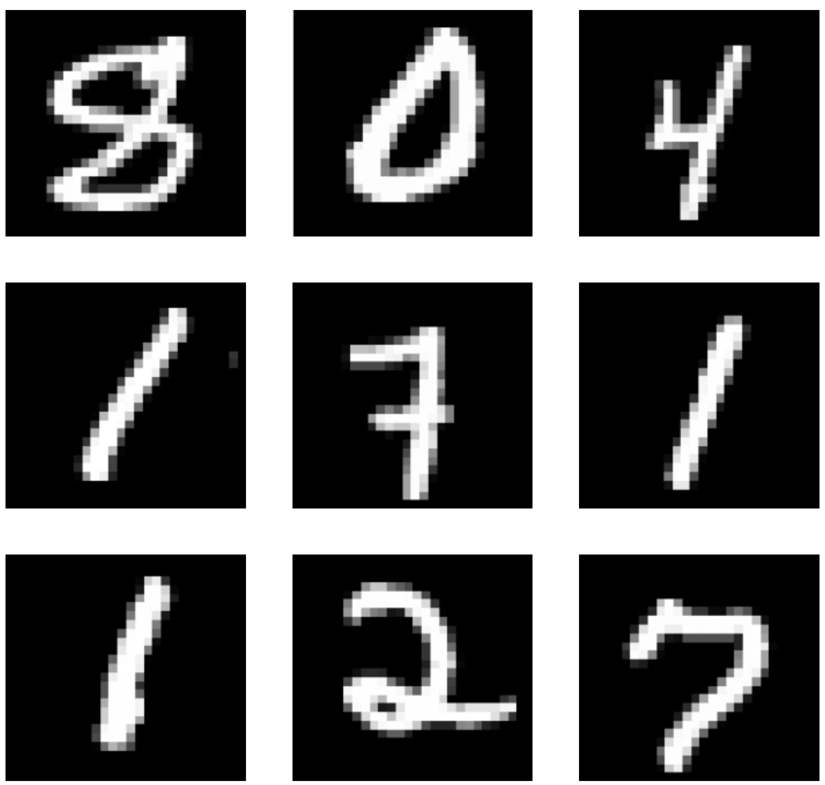


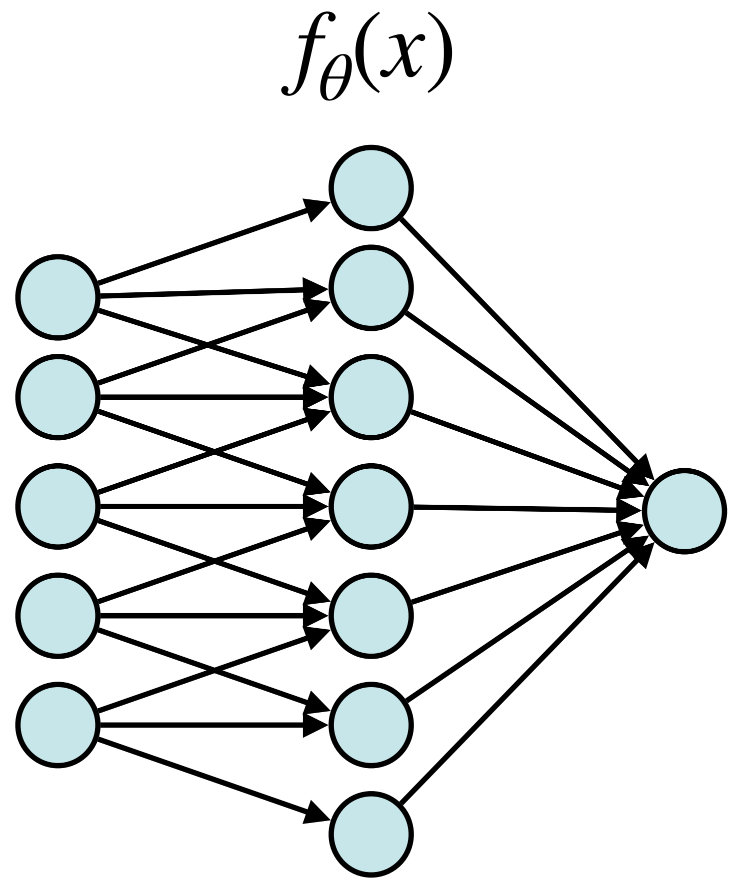
8 0 4

1 7 1

1 2 7

Deep Learning



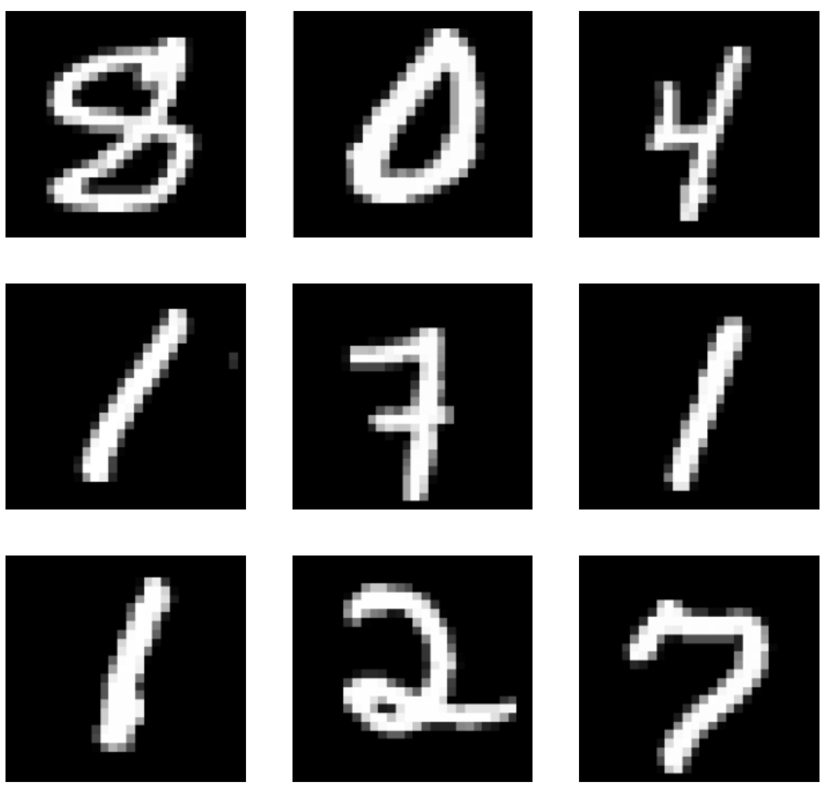


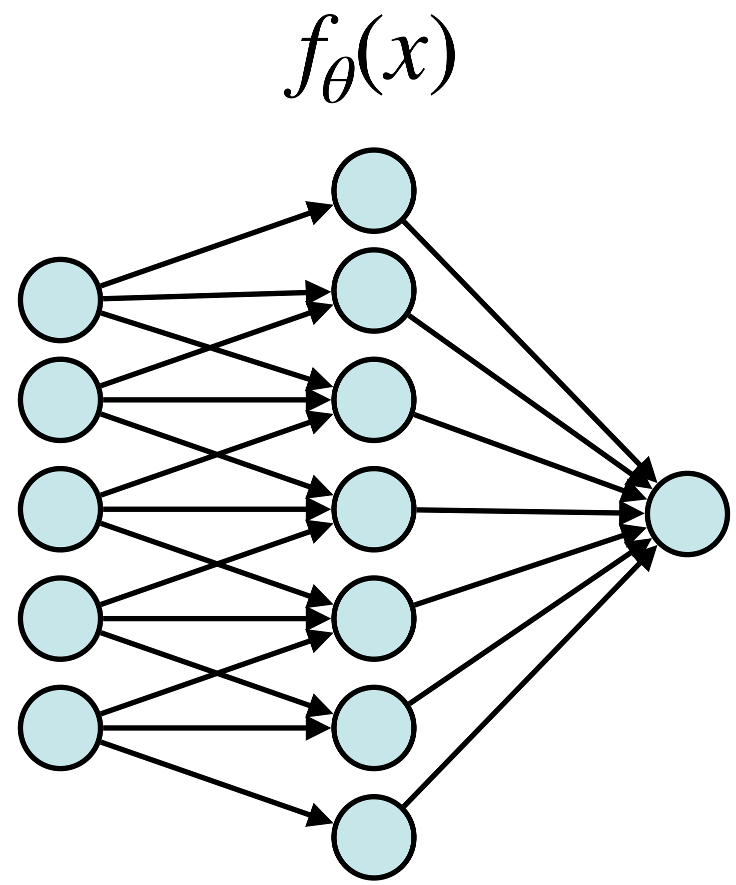
8 0 4

1 7 1

1 2 7

Deep Learning





8 0 4

1 7 1

1 2 7

Learning Neural Networks

- Objective function

$$\theta^* = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda\|\theta\|$$

  - Training dataset $\{x_i, y_i\}_{i=1\ldots n}$
  - Neural network f_θ with adjustable parameters θ.
  - Loss function $L(y, f_\theta(x))$ measures how good NN prediction is.
  - Regularisation $\lambda\|\theta\|$, e.g. weight decay, prevents overfitting.
  - Optimization of objective function using stochastic gradient descent or other more advanced optimizers.

- Mathematical framework is called **empirical risk minimisation**.

Learning Neural Networks

$$\theta^* = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda \|\theta\|$$

- Modern deep learning software frameworks allow for **construction** and **learning** of parameterised functions.
  - Consists of basic building blocks composed into **computation graphs** or "**neural architectures**".
  - Highly **expressive** and **flexible**.
  - **Modular**: reusable complex building blocks are themselves composed of simpler **building blocks**.
- Neural architectures expresses **domain knowledge**.
- Learning using stochastic gradient descent (on multiple CPUs, GPUs, clusters) is **automated** and **scalable**.

Research Infrastructure

- **Computational Infrastructure** critical to deep learning (and ML):
  - **software instructures** allow easy building of neural networks, automating away most low-level operations.
  - **hardware** allows fast training, and scalable productionisation.
  - **large datasets** and **difficult, shared, challenges** pushing frontier forward.
- Culture of sharing code via **open source**, findings via open access publication models.

Platforms



Microsoft Azure

aws

Frameworks

scikit learn

Caffe

Keras

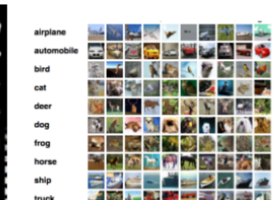
Microsoft CNTK

mxnet

theano

PYTORCH

Datasets

Caltech 101

IMAGENET

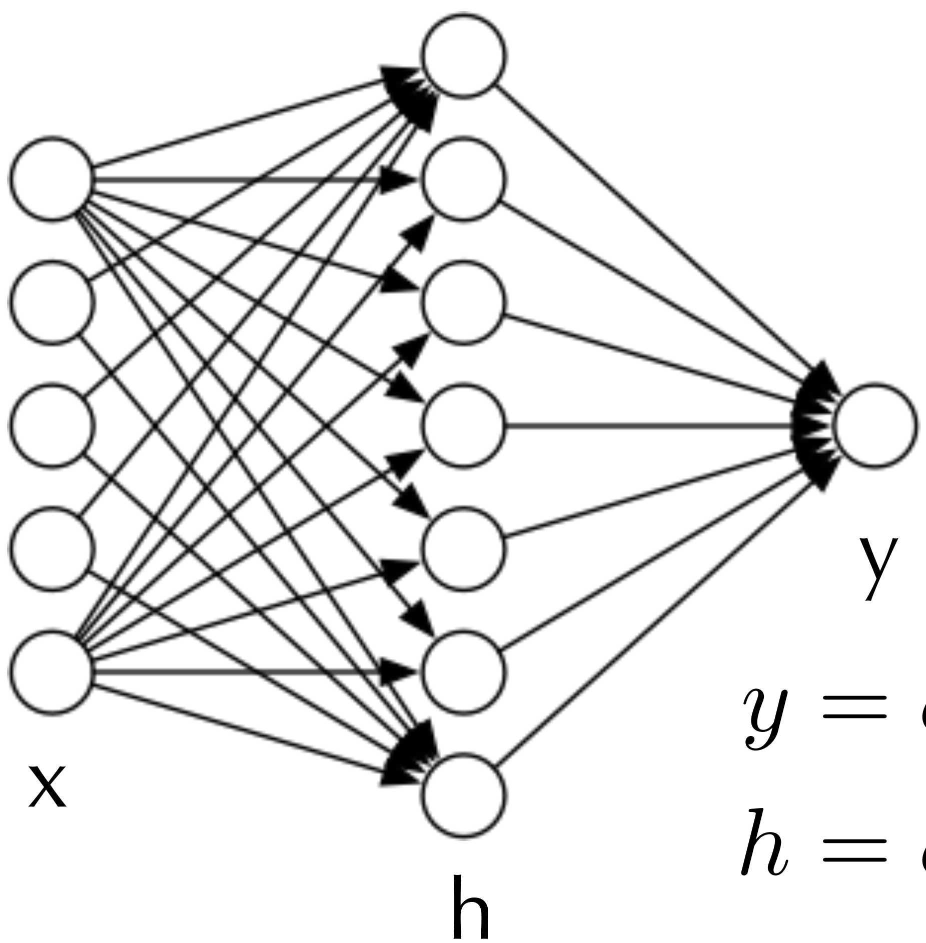
LDC Linguistic Data Consortium

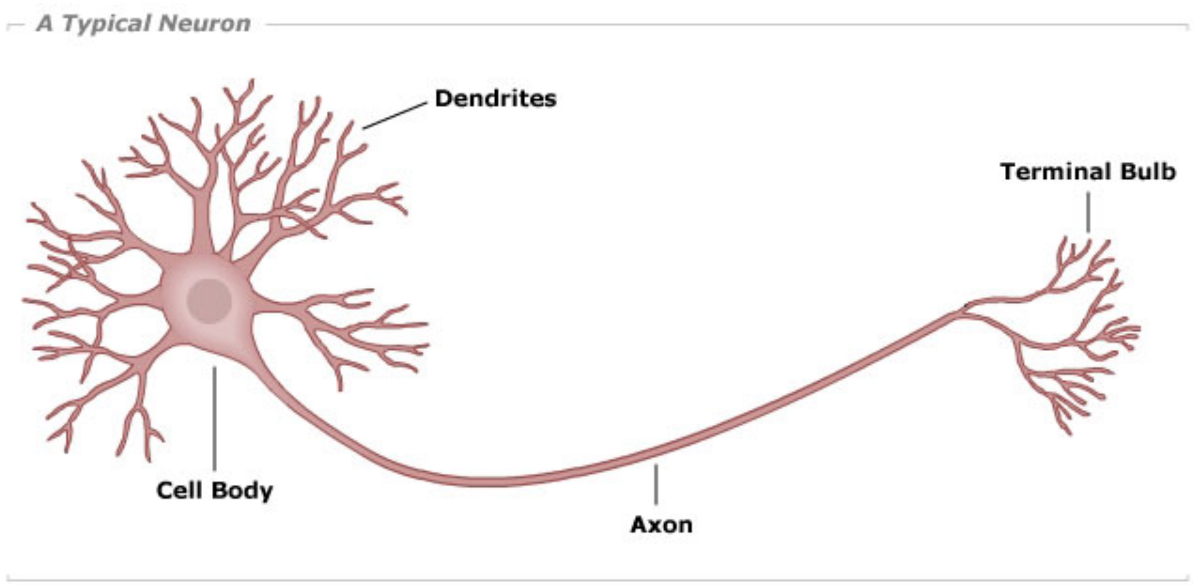
UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

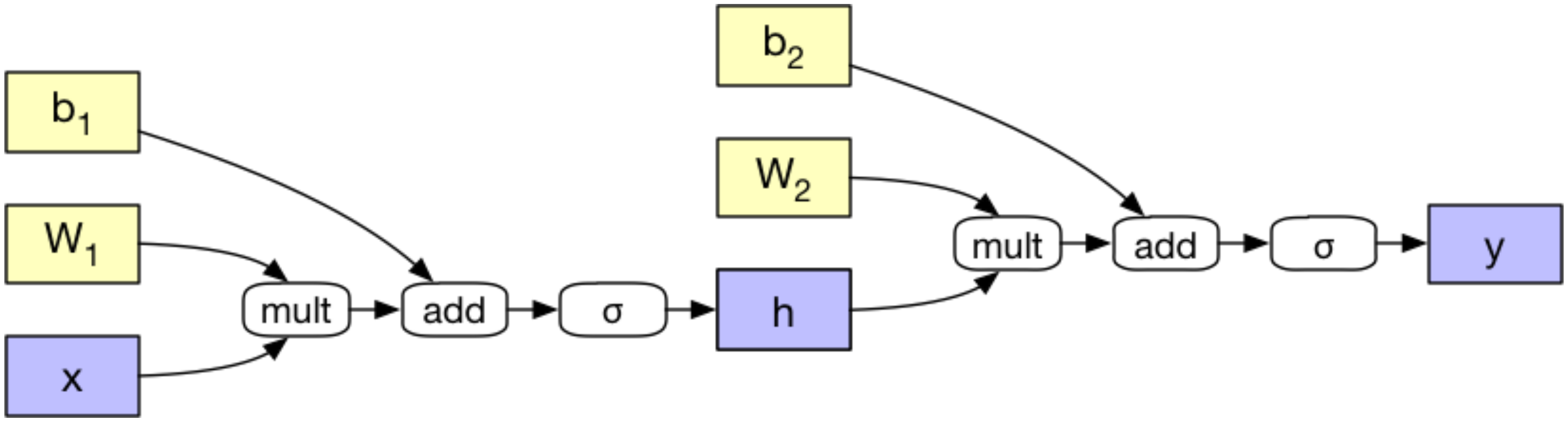
WMT Workshop 2014

Neural Networks





$$y = \sigma(W_2 h + b_2)$$
$$h = \sigma(W_1 x + b_1)$$



Popular NN Modules

- sigmoid

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

- tanh

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

- relu

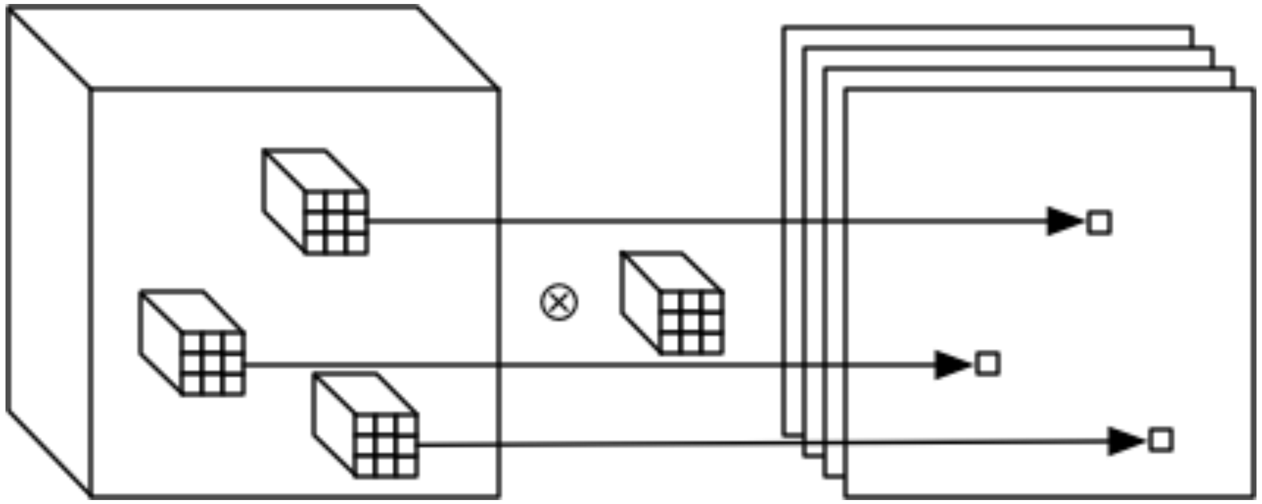
$$\mathrm{relu}(x) = \max(0, x)$$

- softmax

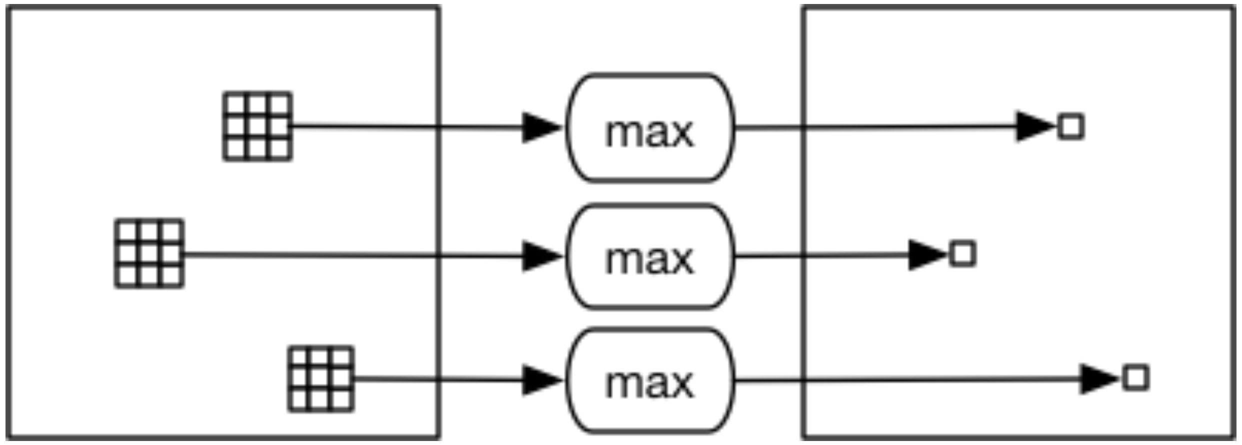
$$\mathrm{softmax}(x_1, \ldots, x_n)$$

$$= \left(\frac{\exp(x_1)}{\sum_i \exp(x_i)}, \ldots, \frac{\exp(x_n)}{\sum_i \exp(x_i)} \right)$$

- Convolution

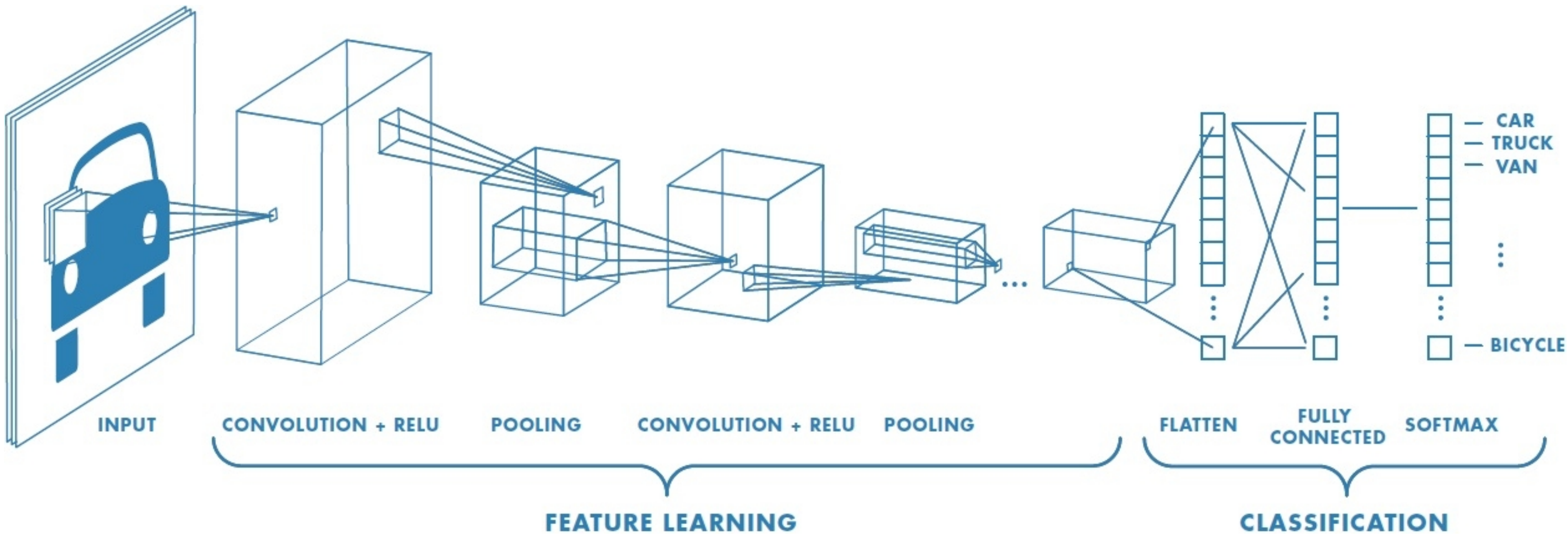


- max pooling



- ....

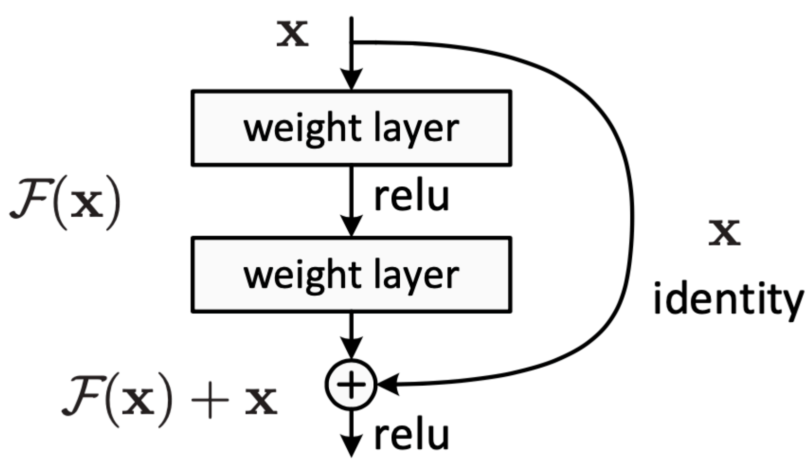
Popular NN Architectures



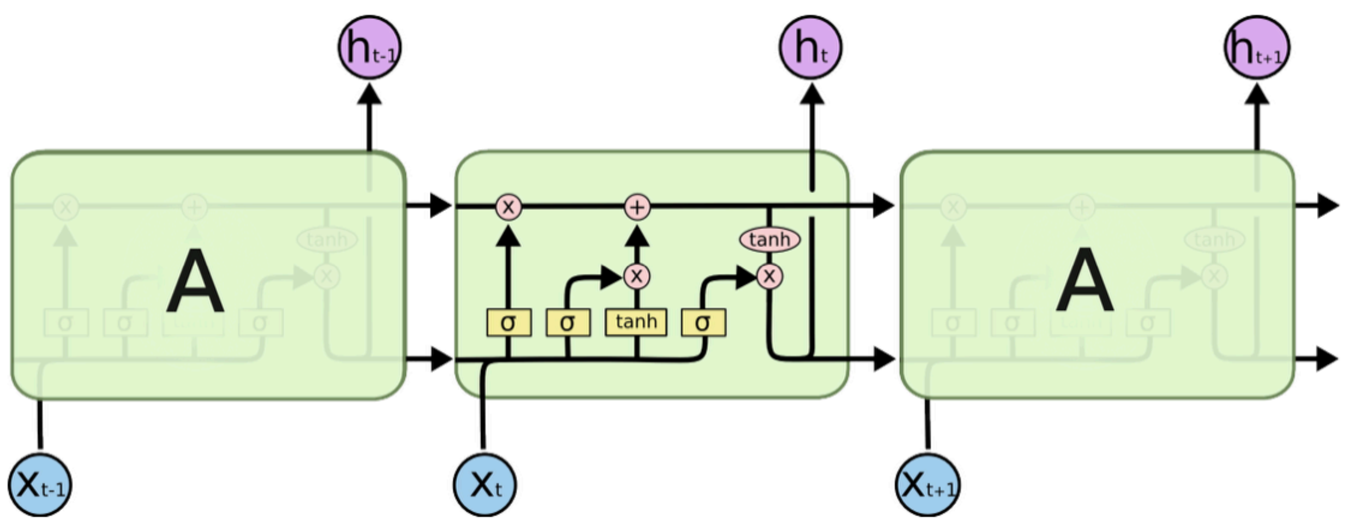
- ConvNets
- Both filter banks and layers are 4D tensors.

Popular NN Architectures

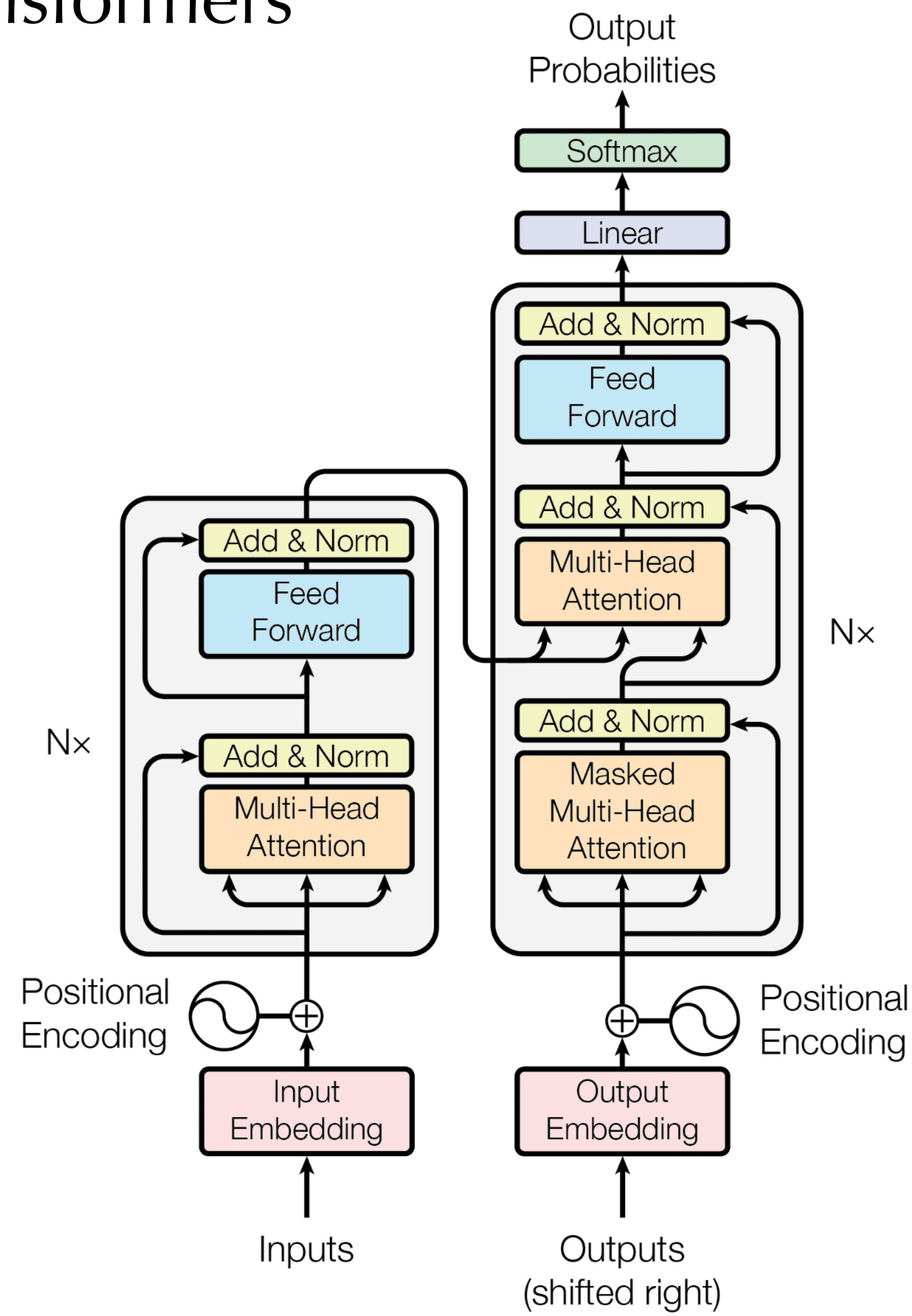
- ResNets



- LSTMs



- Transformers



Losses

- Cross entropy

$$\mathrm{CrossEntropy}(t, y) = \sum_i t_i \log y_i$$

- Square loss

$$\mathrm{Square}(t, y) = \|t - y\|_2^2$$

- Hinge loss

$$\mathrm{Hinge}(t, y) = \max(0, 1 - t \cdot y)$$

Exercise: Why Sigmoid?

- Typical final-layer nonlinearity is the sigmoid:

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

- Typical loss for binary classification is cross-entropy:

$$\mathrm{CrossEntropy}(t, y) = \sum_i t_i \log y_i$$

- If x is the pre-activation of final NN layer (the so-called **logits**), the loss for predicting label t given inputs is:

$$L(t, x) = \mathrm{CrossEntropy}(t, \sigma(x))$$

- What is the derivative $\frac{dL(t,x)}{dx}$?

Advanced Exercises: Why softmax?

- Generalise the derivation on the previous slide from binary to multiclass classification.

- In popular DL frameworks like pytorch, there are modules for binary cross entropy nn.BCELoss and nn.Sigmoid. But it is recommended to not use these for training, and rather use nn.BCEWithLogitsLoss, which just composes the two. Why?

Optimising Neural Networks

Gradient Descent

$$\theta^* = \arg\min_\theta \frac{1}{n}\sum_{i=1}^n L(y_i, f_\theta(x_i))] + \lambda D(\theta\|\theta_0)$$

- Iterative procedure:

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon_t \left(\frac{1}{n}\sum_{i=1}^n \nabla L(y_i, f_{\theta^{(t)}}(x_i)) + \lambda \nabla D(\theta^{(t)}\|\theta_0)\right)$$

- Five issues:
  - how to deal with high dimensionality?
  - scalability to large data sets?
  - how to compute derivatives?
  - how to ensure gradient updates are neither too small nor too large?
  - how to deal with non-linearities?

Stochastic Gradient Descent

- Estimate gradient of loss using "minibatches" of data:

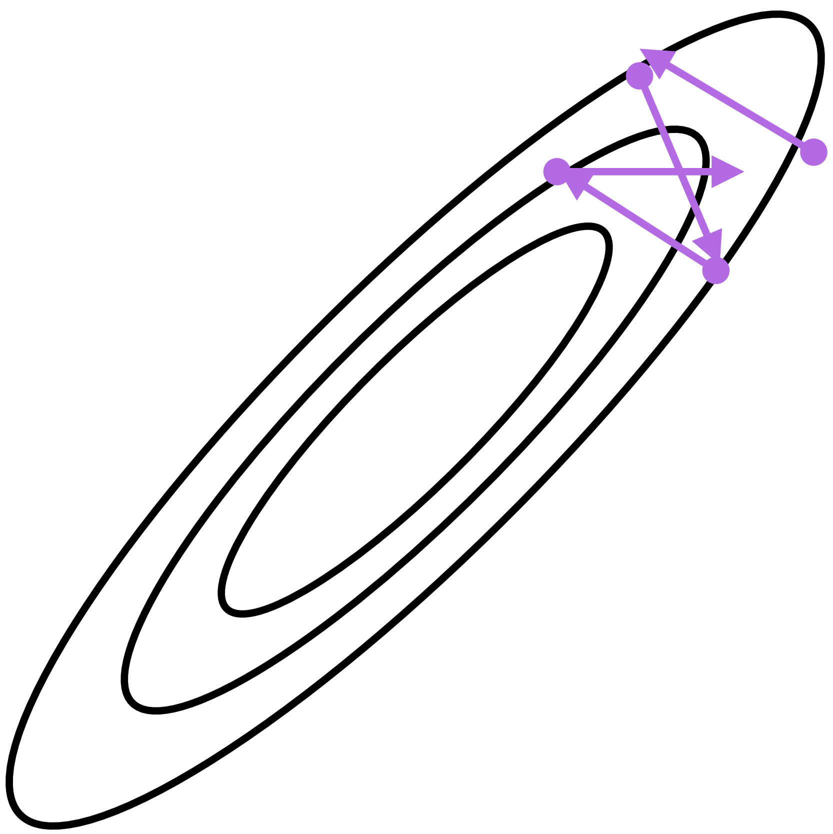
$$\theta^{(t+1)} = \theta^{(t)} - \epsilon_t \left(\frac{1}{|B_t|} \sum_{i \in B_t} \nabla L(y_i, f_{\theta^{(t)}}(x_i)) + \lambda \nabla D(\theta^{(t)} \| \theta_0) \right)$$

- Reduce computation cost from O(n) to O($|B_t|$).
  - More data is always better, as long as you have the compute to handle it.
- Stochastic gradients are unbiased estimates ⇒ convergence theory.
- Stochasticity can help regularise and alleviate over-fitting
  - Hardt et al 2016, Barrett & Dherin 2020

Other Popular Optimizers

- SGD+momentum

$$\Delta^{(t+1)} = \alpha\Delta^{(t)} + (1-\alpha)\nabla\mathcal{L}_t$$
$$\theta^{(t+1)} = \theta^{(t)} - \epsilon\Delta^{(t+1)}$$



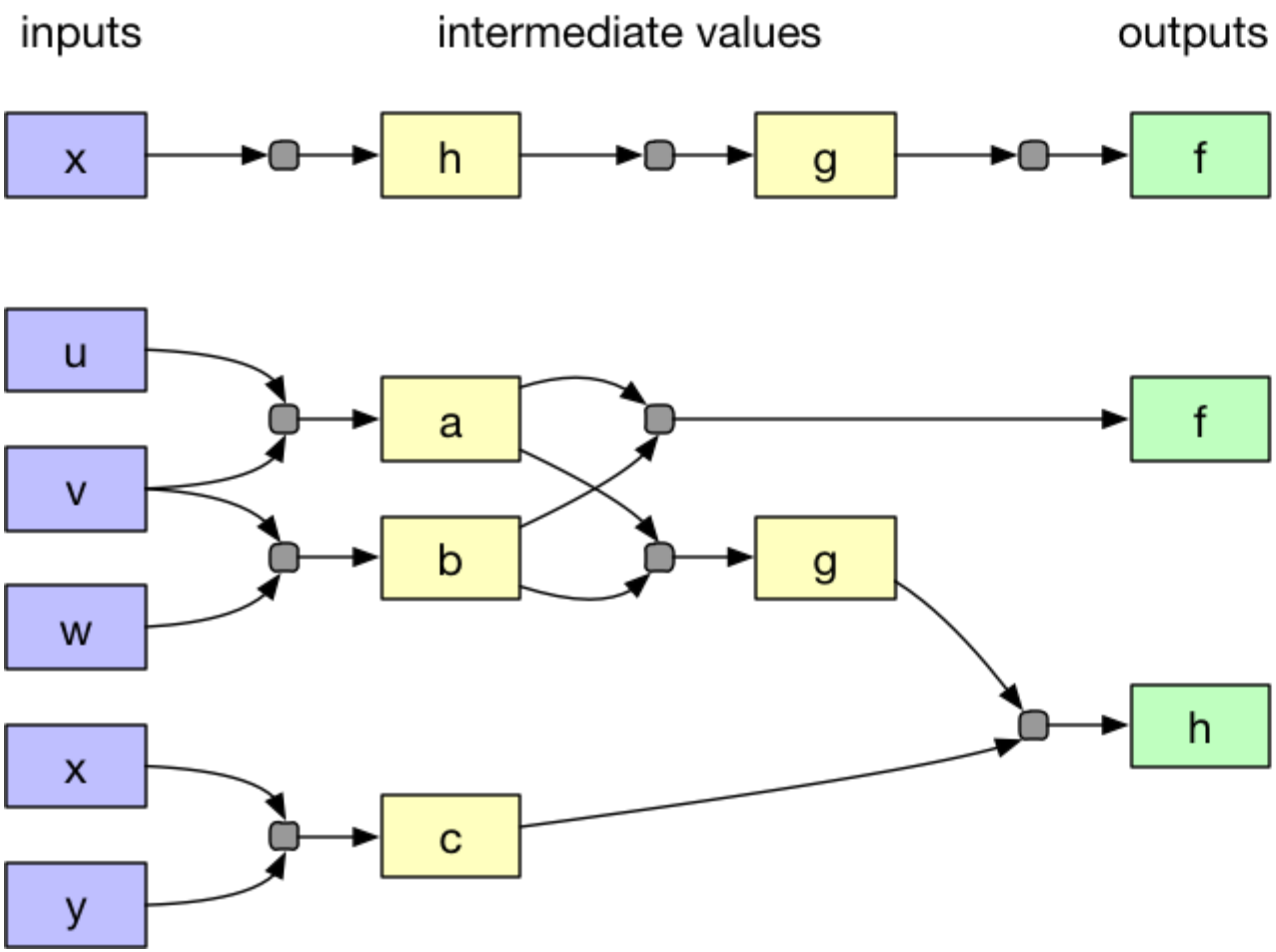
- Adam

$$\Delta^{(t+1)} = \beta_1\Delta^{(t)} + (1-\beta_1)\nabla\mathcal{L}_t$$
$$V^{(t+1)} = \beta_2 V^{(t)} + (1-\beta_2)(\nabla\mathcal{L}_t)^2$$
$$\theta^{(t+1)} = \theta^{(t)} - \epsilon\frac{\Delta^{(t+1)}/(1-\beta_1^{t+1})}{\sqrt{V^{(t+1)}/(1-\beta_2^{t+1})}+\delta}$$

- Other tricks:
  - gradient clipping
  - batchnorm
  - decreasing step sizes
  - increasing step sizes
  - oscillating step sizes
  - second order optimisation
- Other issues:
  - vanishing gradients, exploding gradients, initialisation schemes

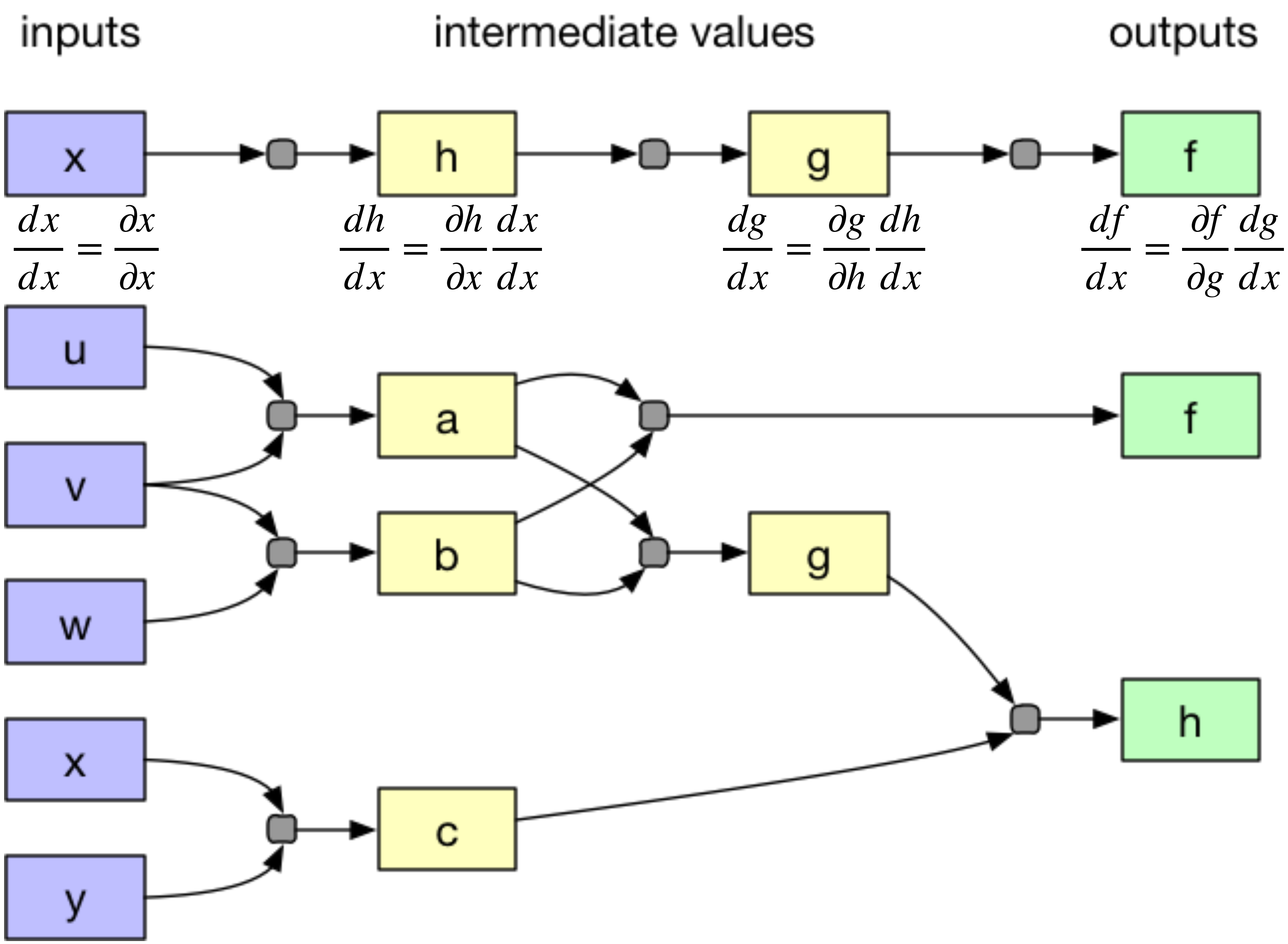
Automatic Differentiation

- Two major approaches: forward mode, and reverse mode AD.



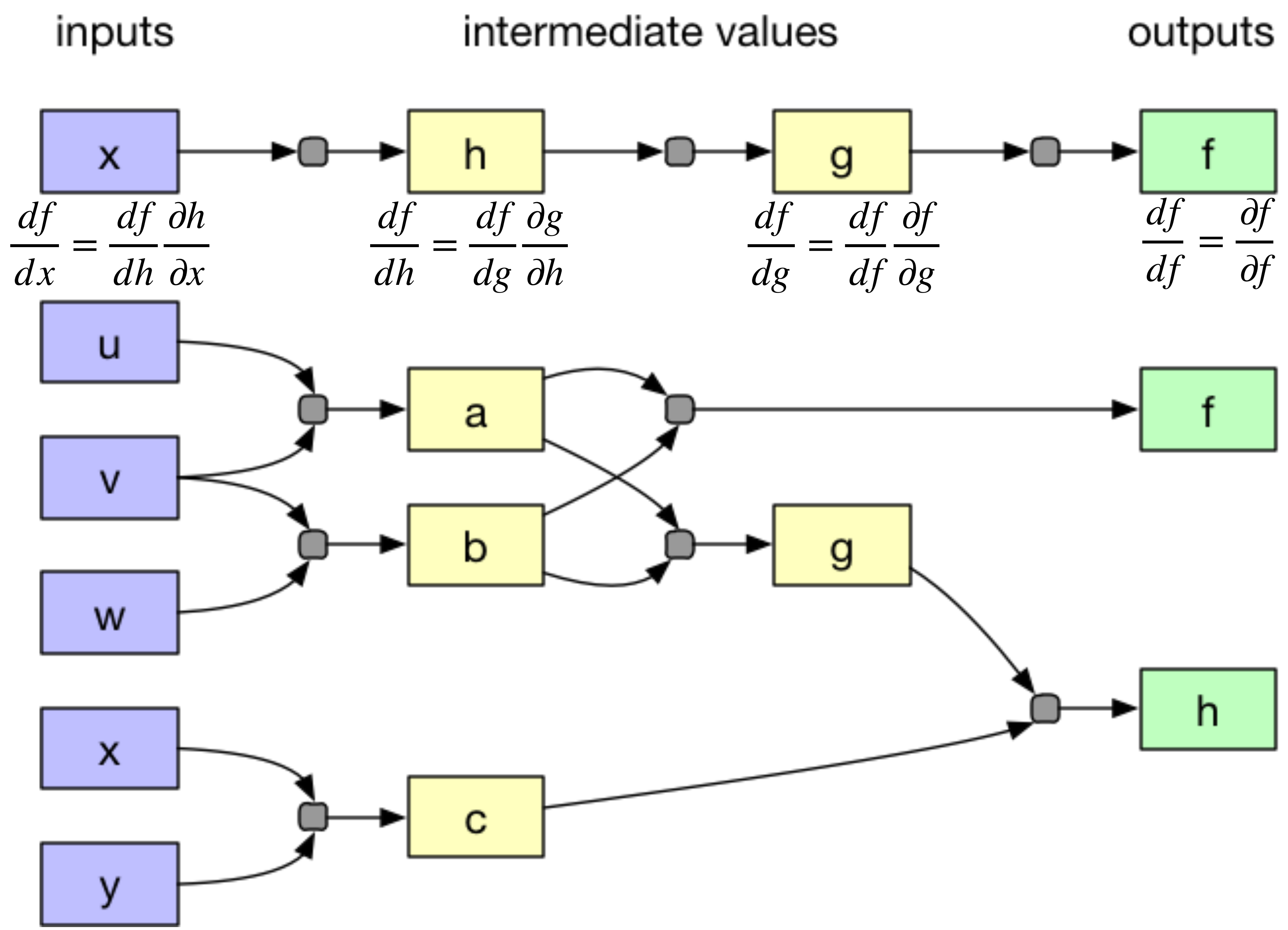
Automatic Differentiation

- Two major approaches: **forward** mode, and reverse mode AD.



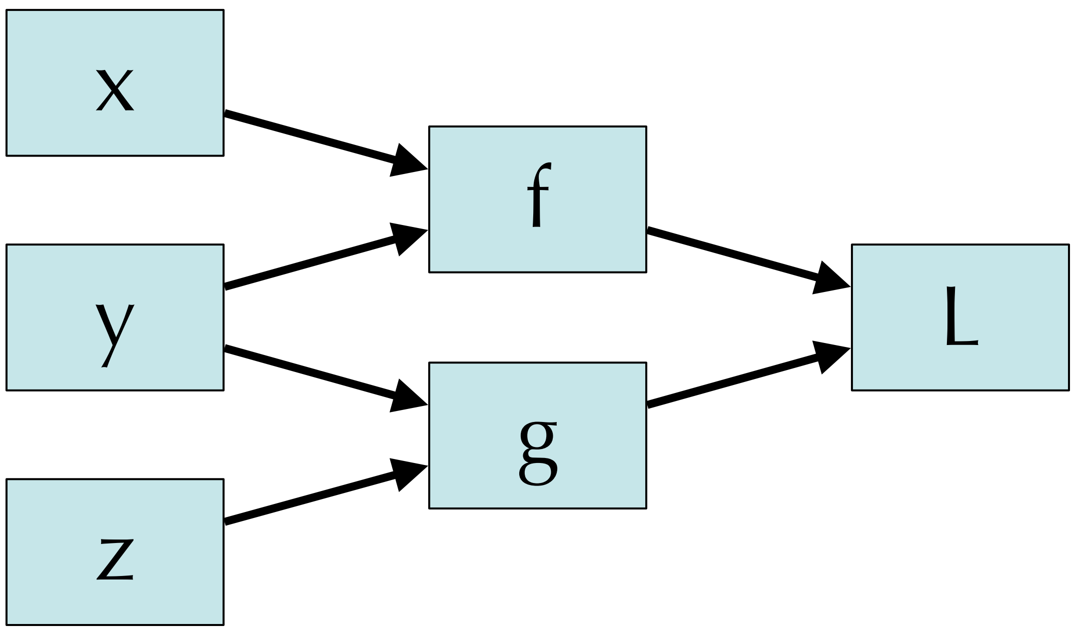
Automatic Differentiation

- Two major approaches: forward mode, and **reverse** mode AD.



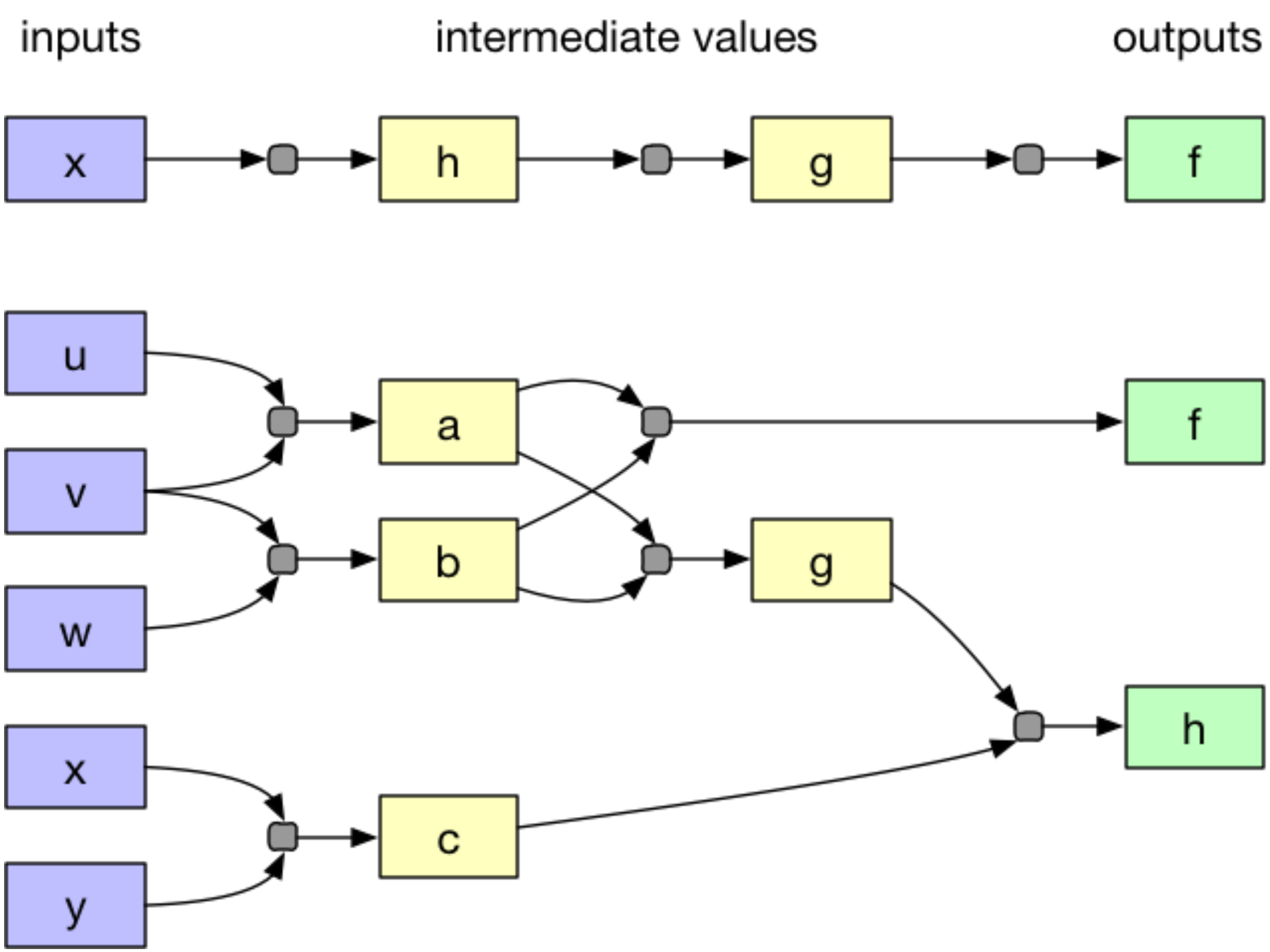
Exercise: Computational Complexity of AD

- What are the compute costs of computing the three derivatives dL/dx, dL/dy and dL/dz by forward and reverse mode AD?
- Speculate as to the computational complexity of AD for general computation graphs.



Automatic Differentiation

- Two major approaches: forward mode, and reverse mode AD.



- Forward: O(#inputs*#nodes). Reverse: O(#outputs*#nodes).

Learning Neural Networks

Optimisation!

NN architecture!

$$\theta^* = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda\|\theta\|$$

Data!

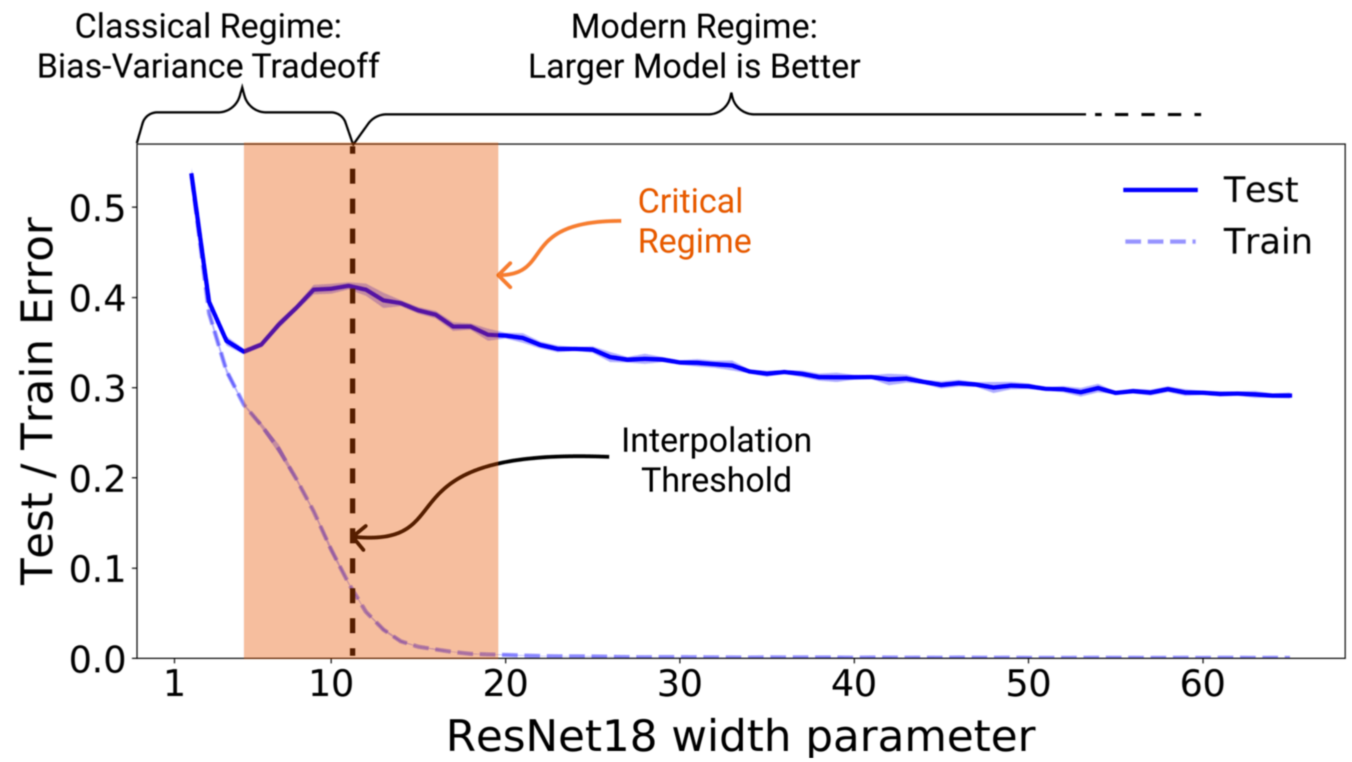
$R(\theta^*) - R_{\mathrm{emp}}(\theta^*)$

Generalisation!

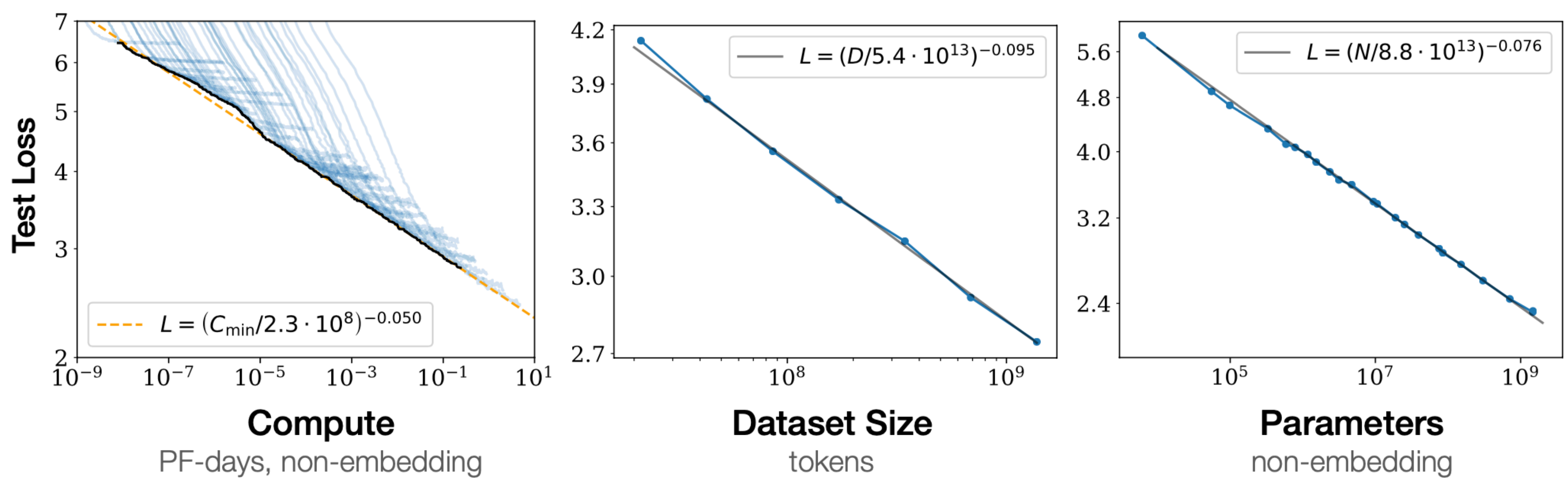
Scale! Data, NN, compute

Current Issues

- Double descent



- Scaling laws



- Robustness, distribution shifts, out-of-distribution detection
- Costs of scaling up deep learning
- Exacerbation of social biases, AI safety

More Resources

- Tutorials and courses:
  - https://www.deepmind.com/learning-resources
  - http://www.gatsby.ucl.ac.uk/teaching/courses/ml1/
  - https://www.coursera.org/learn/machine-learning
  - http://videolectures.net/deeplearning2015_salakhutdinov_deep_learning/
  - Andrew Ng's NeurIPS 2016 tutorial "Nuts and bolts of building AI applications using Deep Learning"
- Summer schools: MLSS, DLSS, RLSS, CIFAR summer school
- Conferences: NeurIPS (formerly NIPS), ICLR, ICML, UAI, AISTATS
- Journals: JMLR, TPAMI, Neural Computation
- ArXiv
- Blogs, distill.pub, thegradient.pub