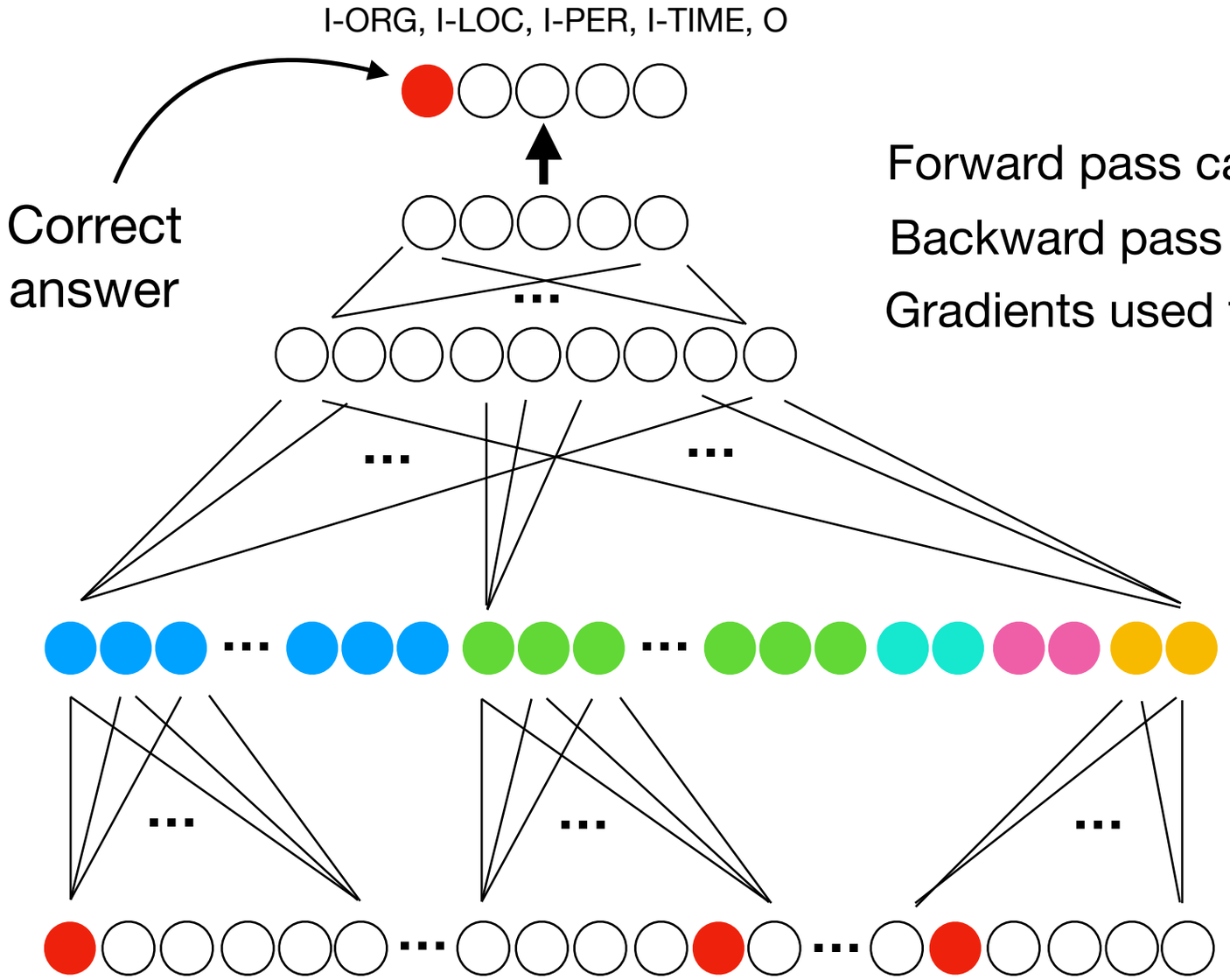
Deep Learning for Natural Language Processing

Stephen Clark
University of Cambridge and DeepMind

3. Training and Optimization

Stephen Clark
University of Cambridge and DeepMind

Supervised Training



Forward pass calculates loss

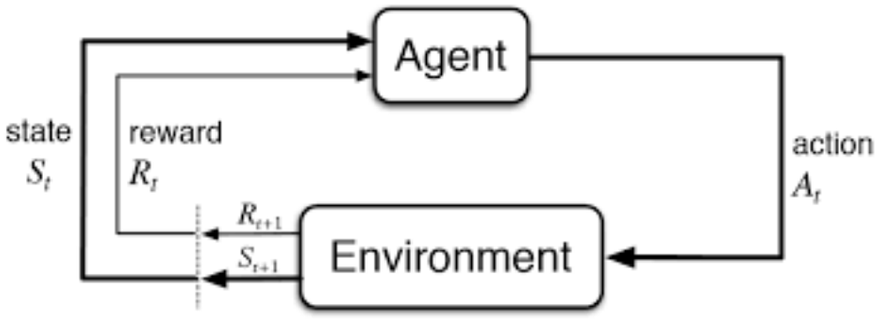
Backward pass (backprop) calculates gradients

Gradients used for parameter updates

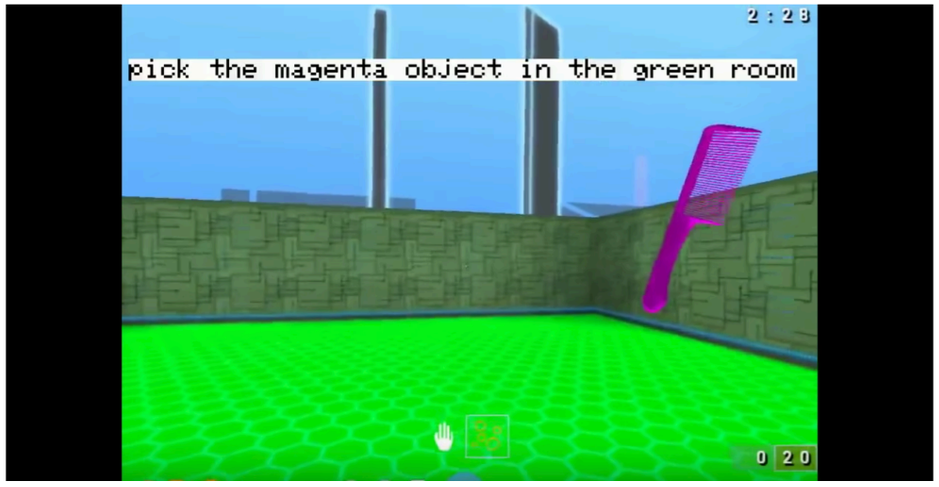
DeepMind

UNIVERSITY OF CAMBRIDGE

An Aside on Reinforcement Learning



- We're seeing more work on RL in NLP (e.g. dialog)

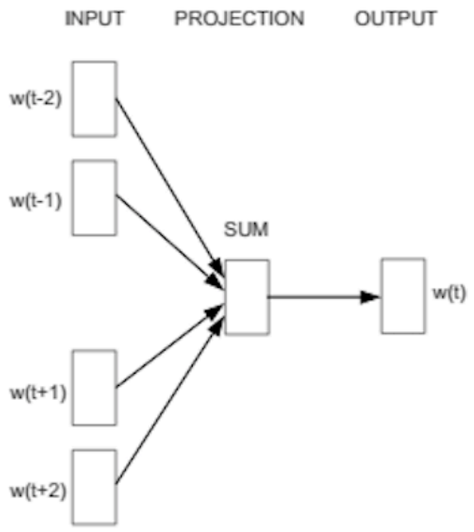


- RL problems typically have "trial-and-error search and delayed reward"

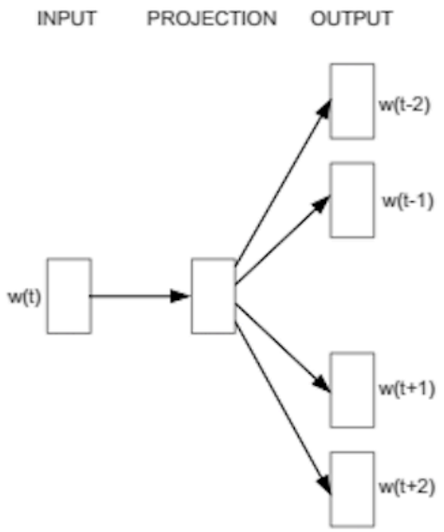
DeepMind

UNIVERSITY OF CAMBRIDGE

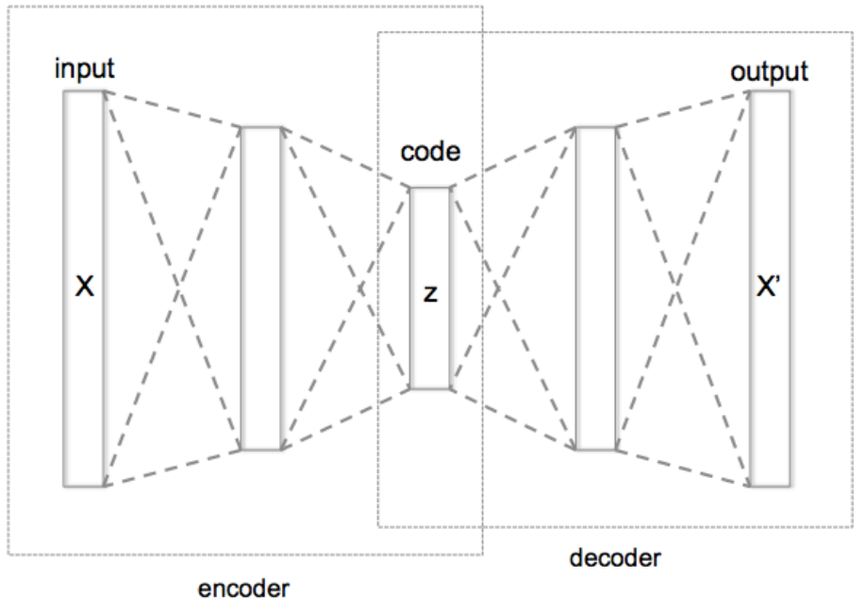
An Aside on Unsupervised Learning



CBOW



Skip-gram



- Important in deriving general-purpose representations

DeepMind

UNIVERSITY OF CAMBRIDGE

The Loss

- For language tasks, typically negative log-likelihood

Ian Botham *bowled* for Somerset	**O**
Botham bowled *for* Somerset cricket	**O**
bowled for *Somerset* cricket club	**I-ORG**
for Somerset *cricket* club in	**I-ORG**
Somerset cricket *club* in the	**I-ORG**
cricket club *in* the 1980s	**O**

<SOS> **The**
<SOS> The **dog**
<SOS> The dog **sleeps**
<SOS> The dog sleeps **before**
<SOS> The dog sleeps before **chasing**
<SOS> The dog sleeps before chasing **the**
<SOS> The dog sleeps before chasing the **cat**

$$-\sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)})$$

$$-\sum_{i=1}^{m} \sum_{j=1}^{|\mathbf{y}^{(i)}|} \log P(\mathbf{y}_j^{(i)}|\mathbf{y}_{<j}^{(i)})$$

DeepMind

UNIVERSITY OF CAMBRIDGE

Empirical Risk Minimization

- What we'd ideally like to minimize:

$$J^*(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}} L(f(\mathbf{x};\theta), y)$$

- What we minimize in practice:

$$J(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}_{\text{data}}} L(f(\mathbf{x};\theta), y)$$
$$= \frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\theta), y^{(i)})$$

θ are the parameters, $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ the training data, L the loss

Regularization

- *Overfitting* is a major problem with powerful deep models (esp. when data is scarce)

- We'd like our models to perform well on unseen test data (possibly at the expense of training set performance)

- Requiring good *generalization* is what separates machine learning from pure optimization problems

It's All About the Gradients

$$J(\theta) = \mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}} L(\mathbf{x}, y, \theta) = \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{x}^{(i)}, y^{(i)}, \theta)$$

$$\text{where } L(\mathbf{x}, y, \theta) = -\log p(y|\mathbf{x};\theta)$$

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta)$$

DeepMind

UNIVERSITY OF CAMBRIDGE

It's All About the Gradients

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta)$$

$$\nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta) = \left\{ \frac{\partial}{\partial \theta_j} L(\mathbf{x}^{(i)}, y^{(i)}, \theta) \right\}_j$$

$$= \left\langle \frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \frac{\partial L}{\partial \theta_3}, \ldots, \frac{\partial L}{\partial \theta_{|\theta|}} \right\rangle$$

DeepMind

UNIVERSITY OF CAMBRIDGE

Gradient-based Optimization

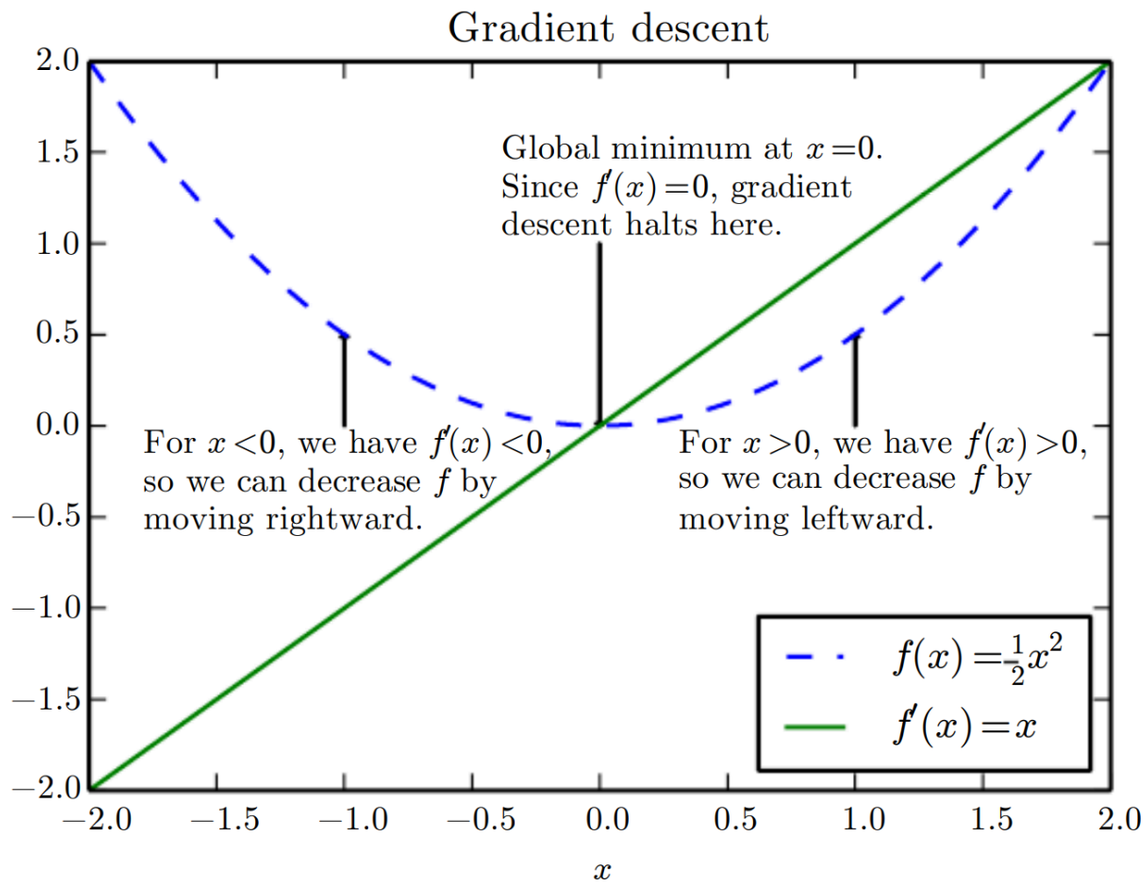


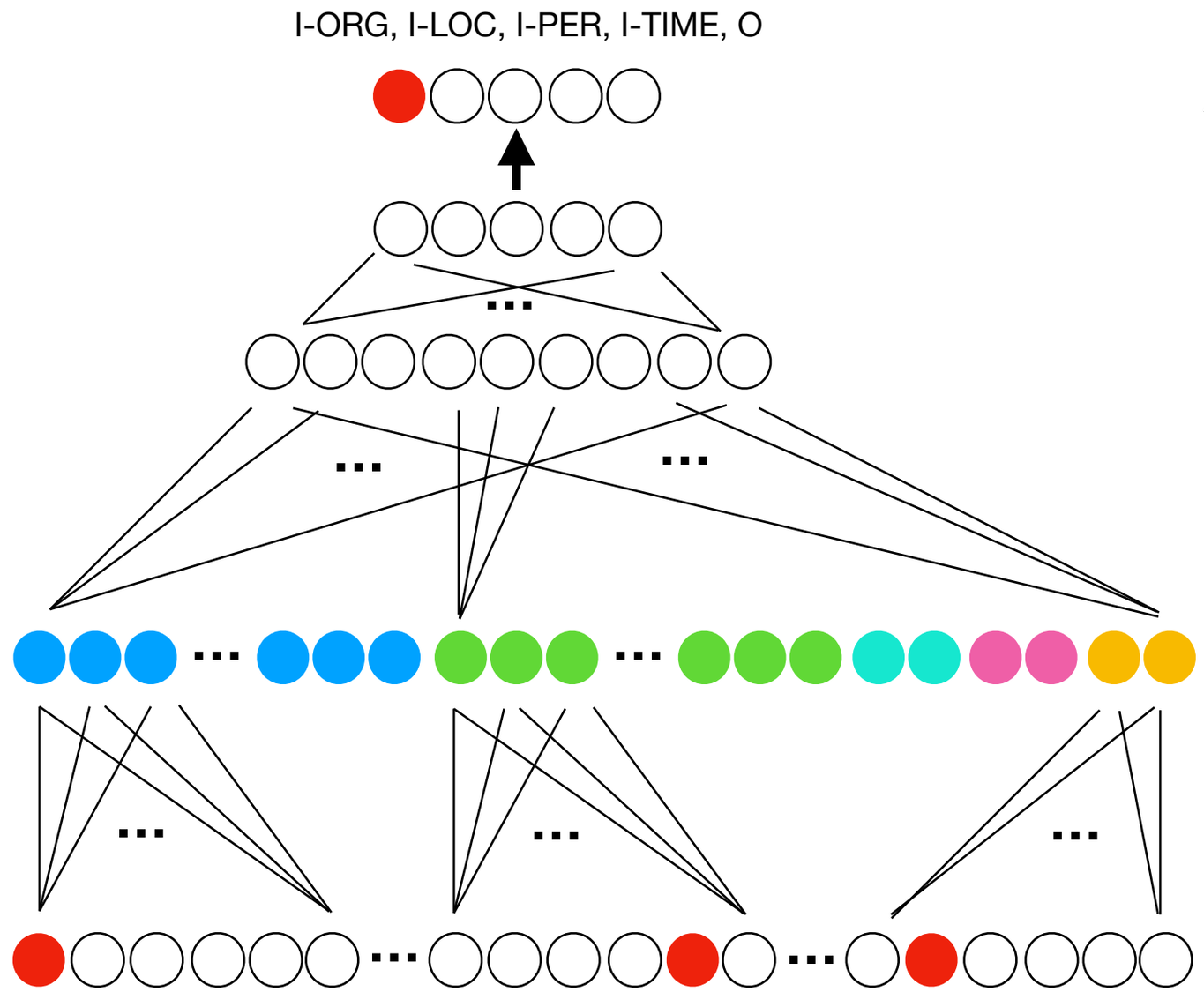
Figure 4.1: An illustration of how the derivatives of a function can be used to follow the function downhill to a minimum. This technique is called *gradient descent*.

p.83 of the Deep Learning textbook

DeepMind

UNIVERSITY OF CAMBRIDGE

Lots of Parameters



All need to be learned (or updated)

$$\begin{pmatrix} 0.1063 & -0.0214 & -0.4013 & 0.9999 & -0.1109 \\ 0.0211 & 0.0086 & 0.0102 & 0.2311 & 0.0876 \\ & & \dots & & \\ -0.1176 & 1.0012 & 1.2150 & 0.7610 & 1.1211 \end{pmatrix}$$

$$\begin{pmatrix} 0.1043 & -0.0135 & -0.1113 & \dots & 0.1209 \end{pmatrix}$$

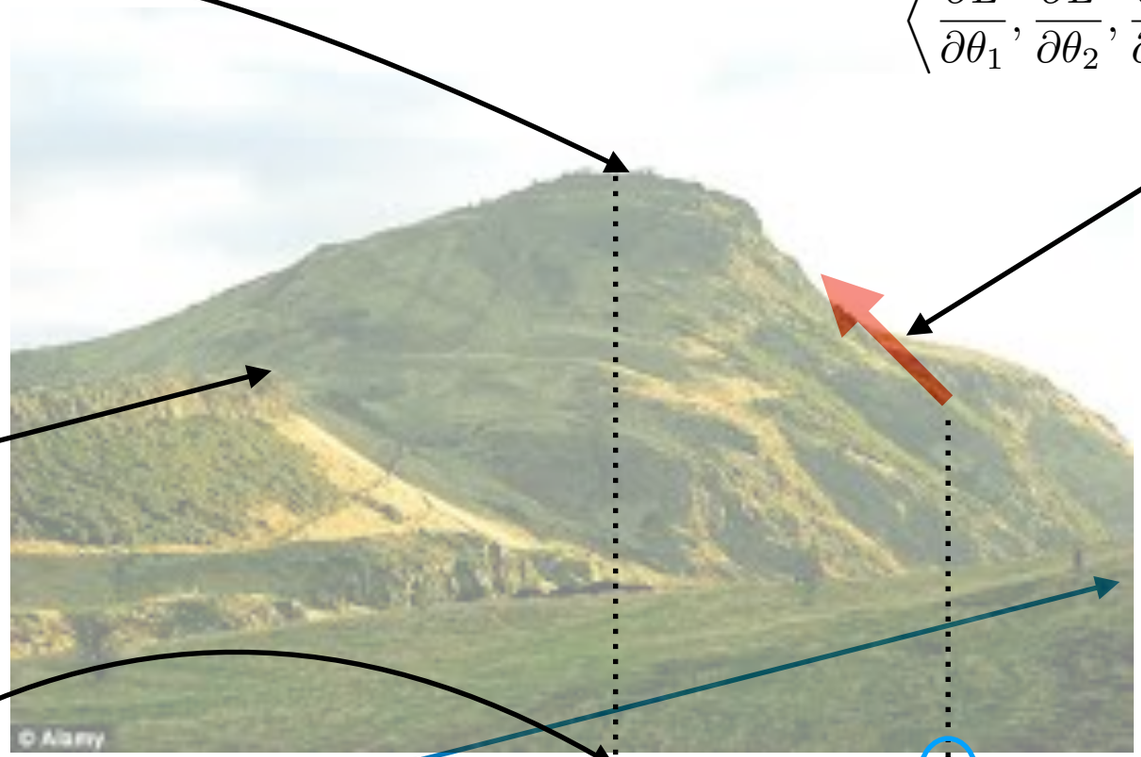
$$\begin{pmatrix} -0.1063 & -0.0234 & -0.4013 & \dots & -0.1109 \\ 0.0011 & 0.1986 & -0.0122 & \dots & 0.0876 \\ & & \dots & & \\ -0.9876 & 1.0092 & -1.9650 & \dots & 1.1200 \end{pmatrix}$$

$$\begin{pmatrix} -0.0537 & 0.0263 & -0.0003 & \dots & 0.0099 \\ 0.0000 & 0.0986 & -0.1023 & \dots & 0.0654 \\ & & \dots & & \\ 0.1020 & -0.8750 & 0.0000 & \dots & -0.1200 \end{pmatrix}$$

DeepMind

UNIVERSITY OF CAMBRIDGE

Gradient-based Optimization



DeepMind

UNIVERSITY OF CAMBRIDGE

Gradient-based Optimization is Hard

- Lots of local maxima for deep networks
- In particular lots of saddle points
- Exploding and vanishing gradients

DeepMind

UNIVERSITY OF CAMBRIDGE

Approximate Minimization

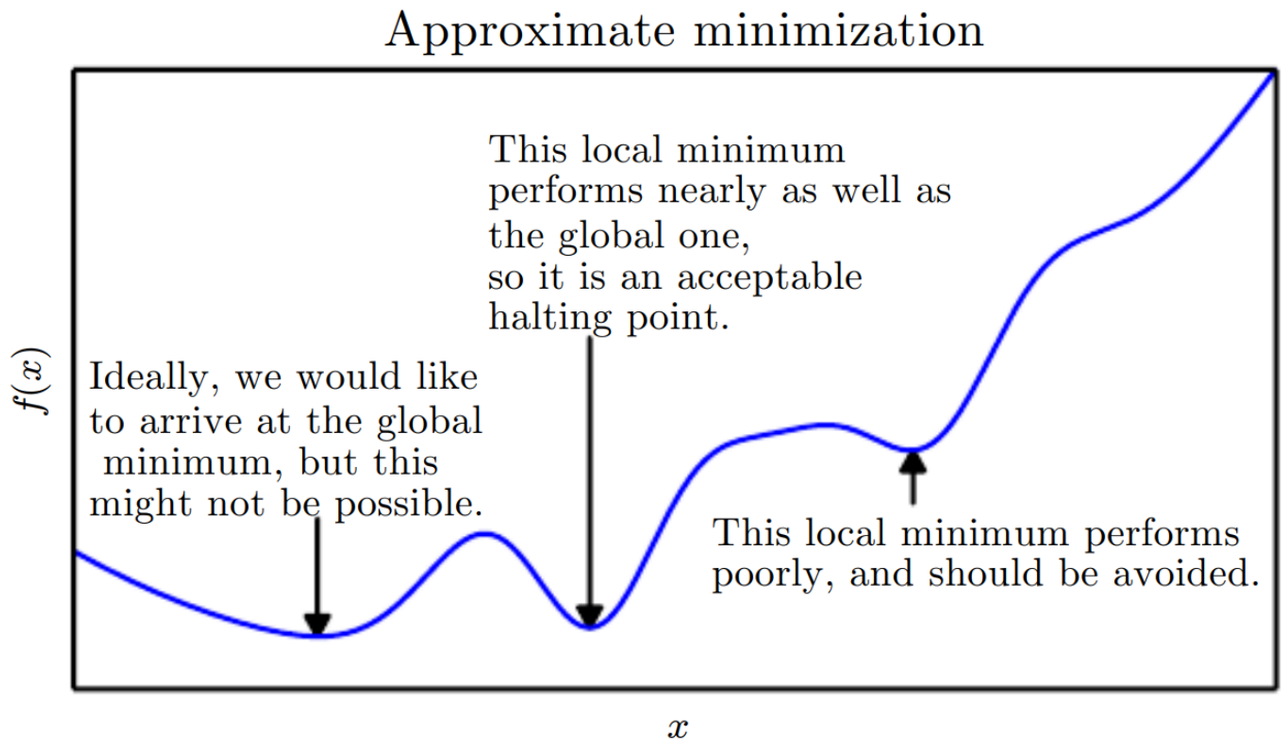


Figure 4.3: Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. In the context of deep learning, we generally accept such solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

p.85 of the Deep Learning textbook

DeepMind

UNIVERSITY OF CAMBRIDGE

Calculating Gradients

- Back-propagation is an efficient algorithm for computing gradients in a neural network: "*Back-propagation is an algorithm that computes the chain rule [of calculus], with a specific order of operations that is highly efficient.*"

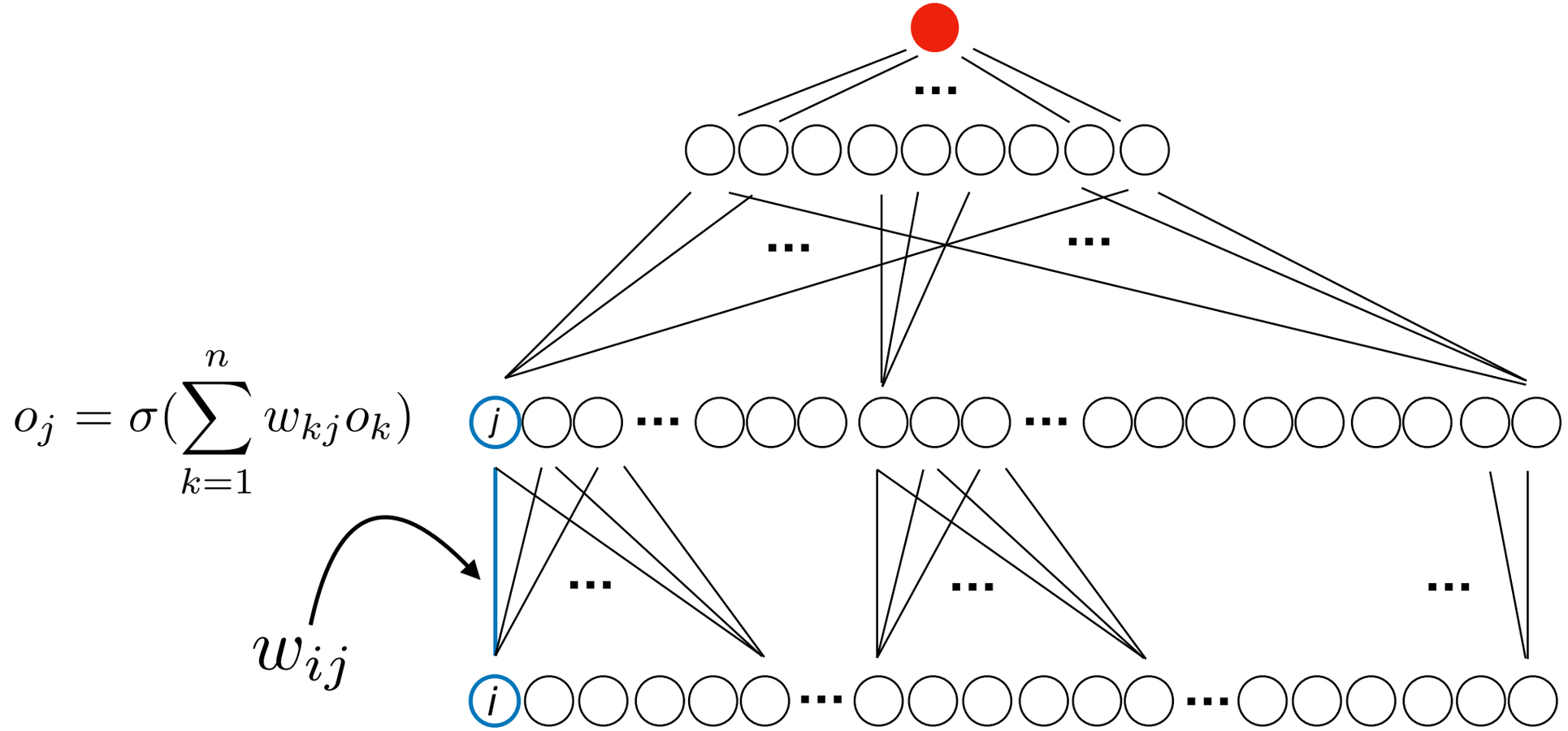
- Another algorithm, such as SGD, uses those gradients to perform learning (optimization)

- Back-propagation is not specific to MLPs, but in principle can compute derivatives of any function

p.198 of the Deep Learning textbook

Backprop for MLPs

$L = \frac{1}{2}(t - y)^2$ where t is the target output, y is the actual output



DeepMind

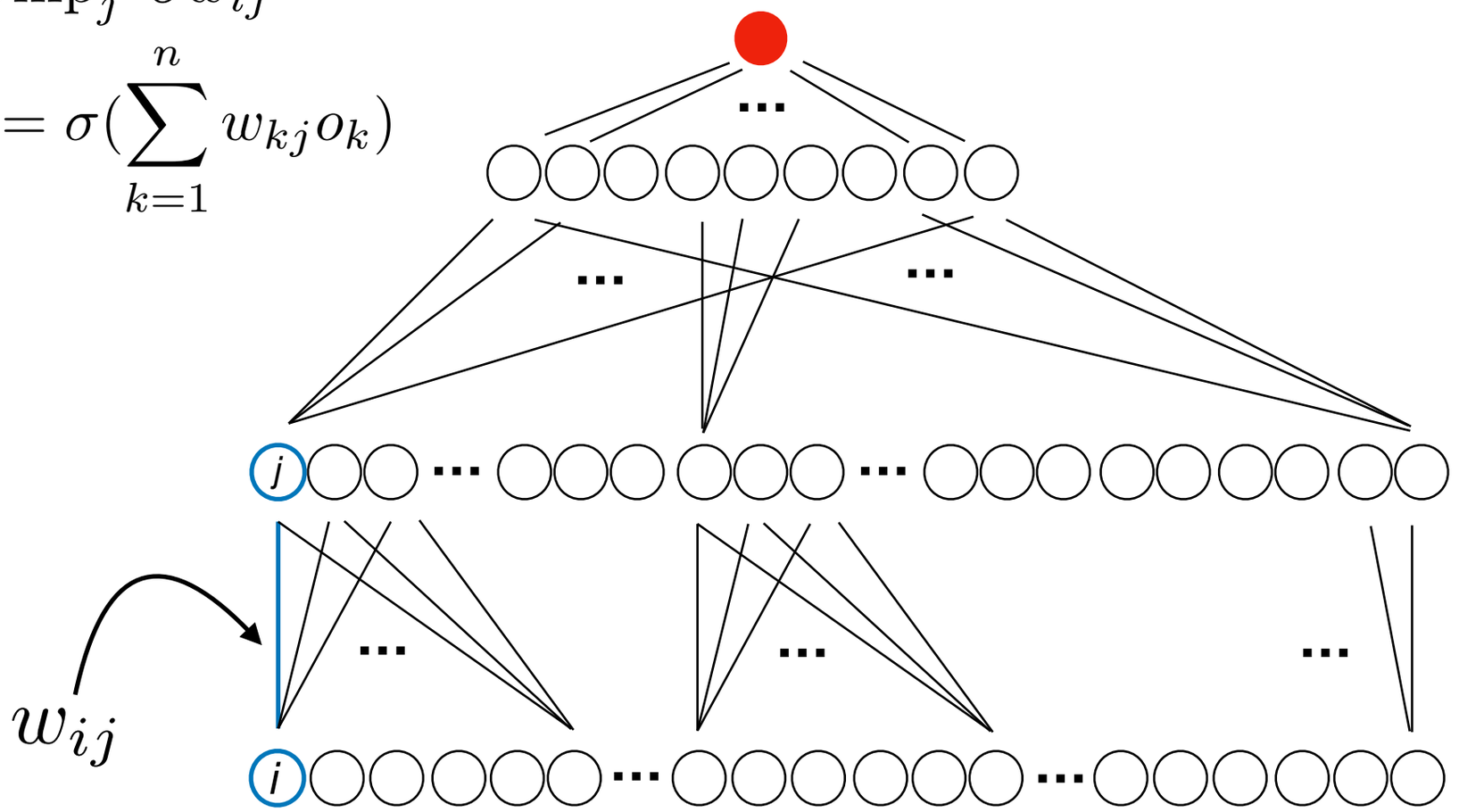
UNIVERSITY OF CAMBRIDGE

Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial o_j} \frac{\partial o_j}{\partial \text{inp}_j} \frac{\partial \text{inp}_j}{\partial w_{ij}}$$

$$o_j = \sigma(\text{inp}_j) = \sigma(\sum_{k=1}^{n} w_{kj} o_k)$$

$$L = \frac{1}{2}(t - y)^2$$



DeepMind

UNIVERSITY OF CAMBRIDGE

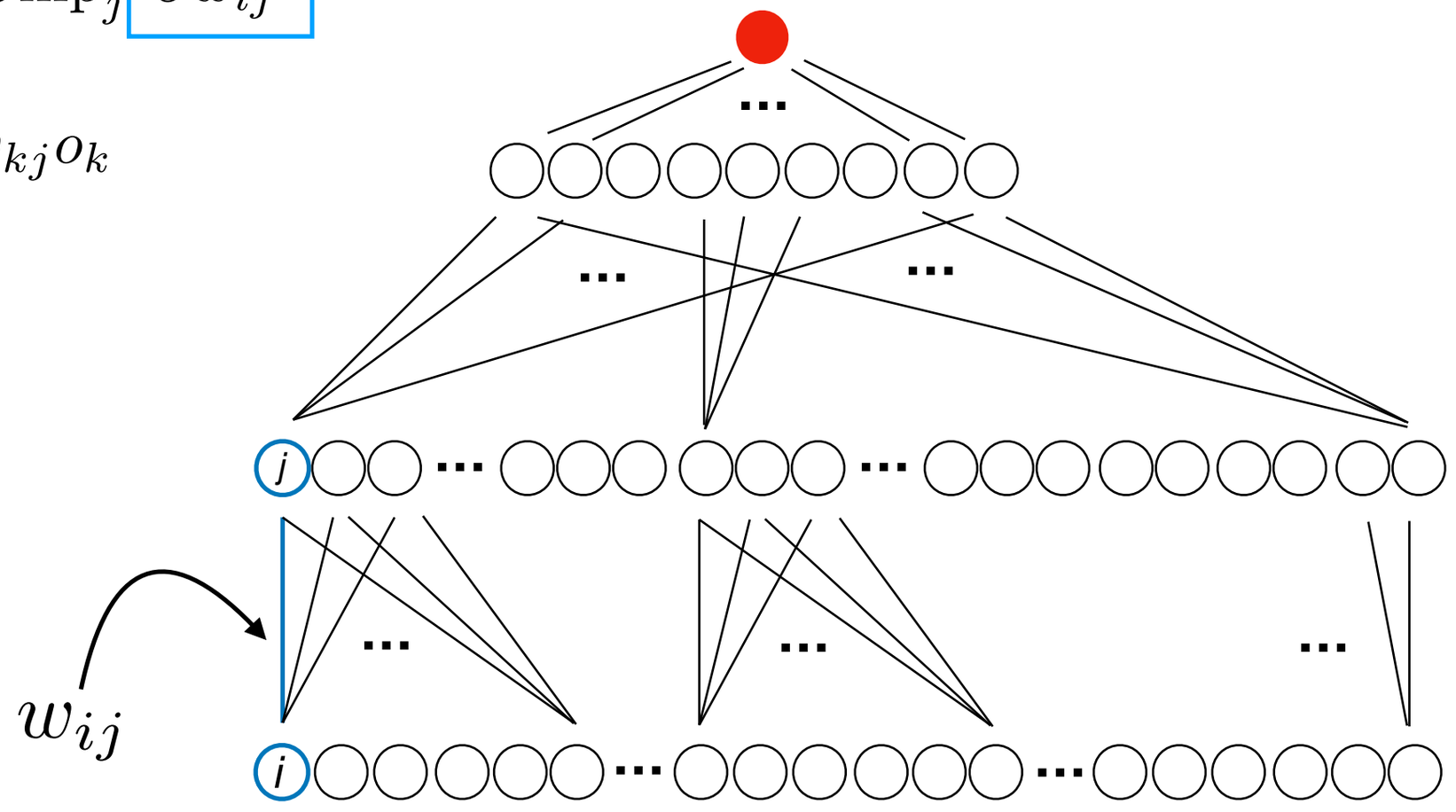
Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial \text{inp}_j}\boxed{\frac{\partial \text{inp}_j}{\partial w_{ij}}}$$

$$\text{inp}_j = \sum_{k=1}^{n} w_{kj} o_k$$

$$\frac{\partial \text{inp}_j}{\partial w_{ij}} = o_i$$

$$L = \frac{1}{2}(t-y)^2$$



DeepMind

UNIVERSITY OF CAMBRIDGE

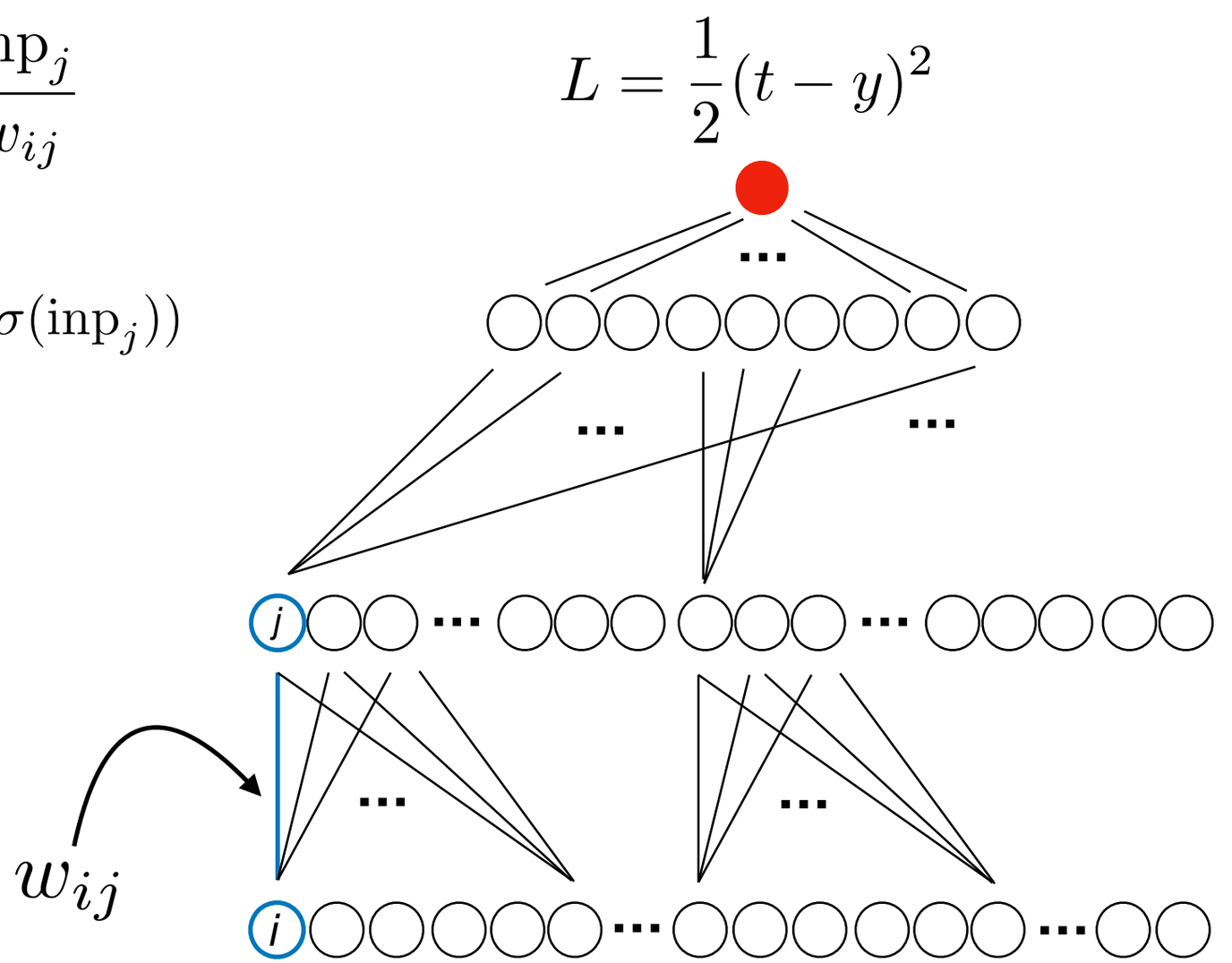
Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial \text{inp}_j}\frac{\partial \text{inp}_j}{\partial w_{ij}}$$

$$\frac{\partial \sigma(\text{inp}_j)}{\partial \text{inp}_j} = \sigma(\text{inp}_j)(1-\sigma(\text{inp}_j))$$
$$= o_j(1-o_j)$$

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

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$



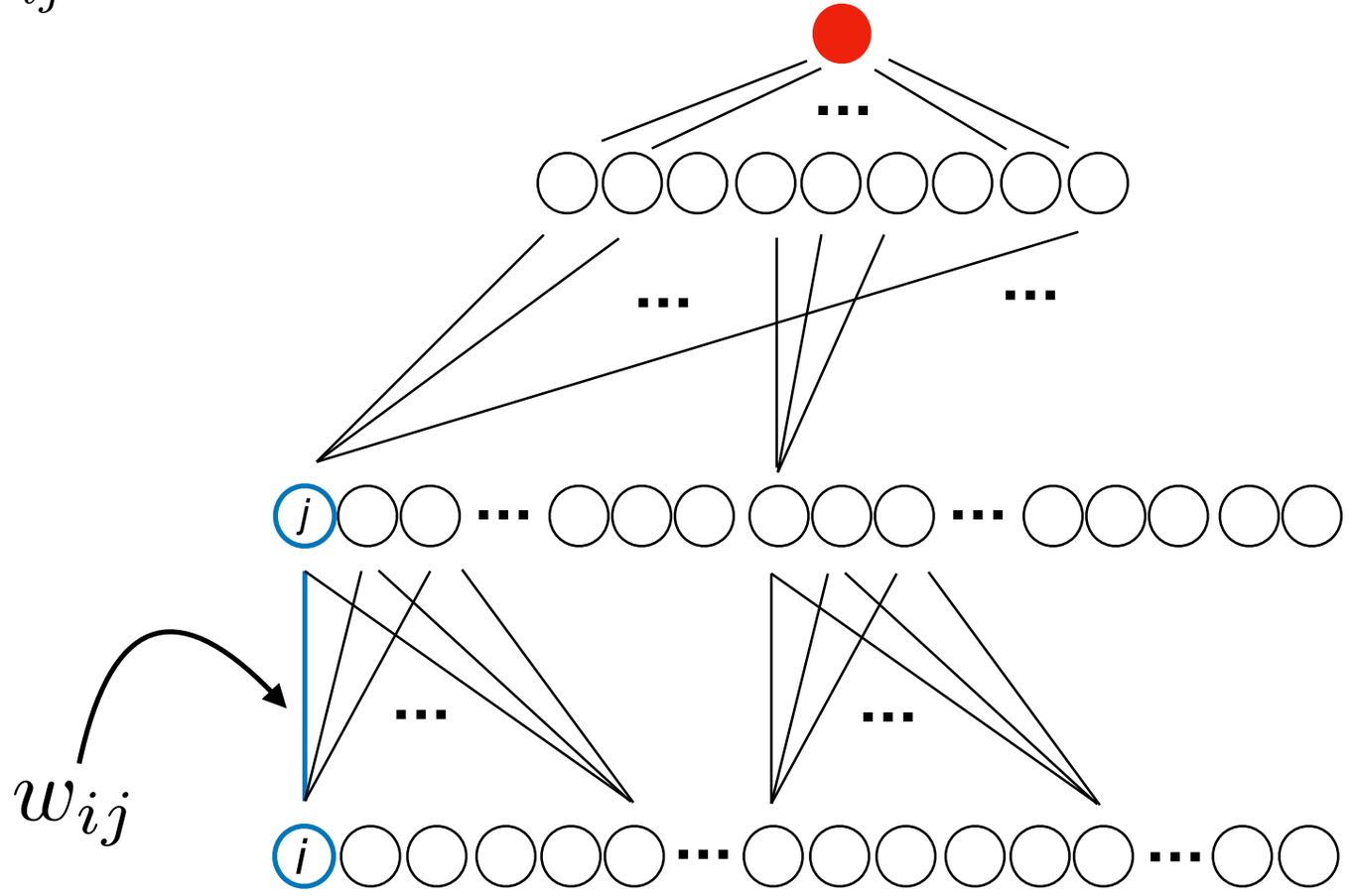
DeepMind

UNIVERSITY OF CAMBRIDGE

Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \boxed{\frac{\partial L}{\partial o_j}} \frac{\partial o_j}{\partial \text{inp}_j} \frac{\partial \text{inp}_j}{\partial w_{ij}}$$

$$L = \frac{1}{2}(t - y)^2$$



DeepMind

UNIVERSITY OF CAMBRIDGE

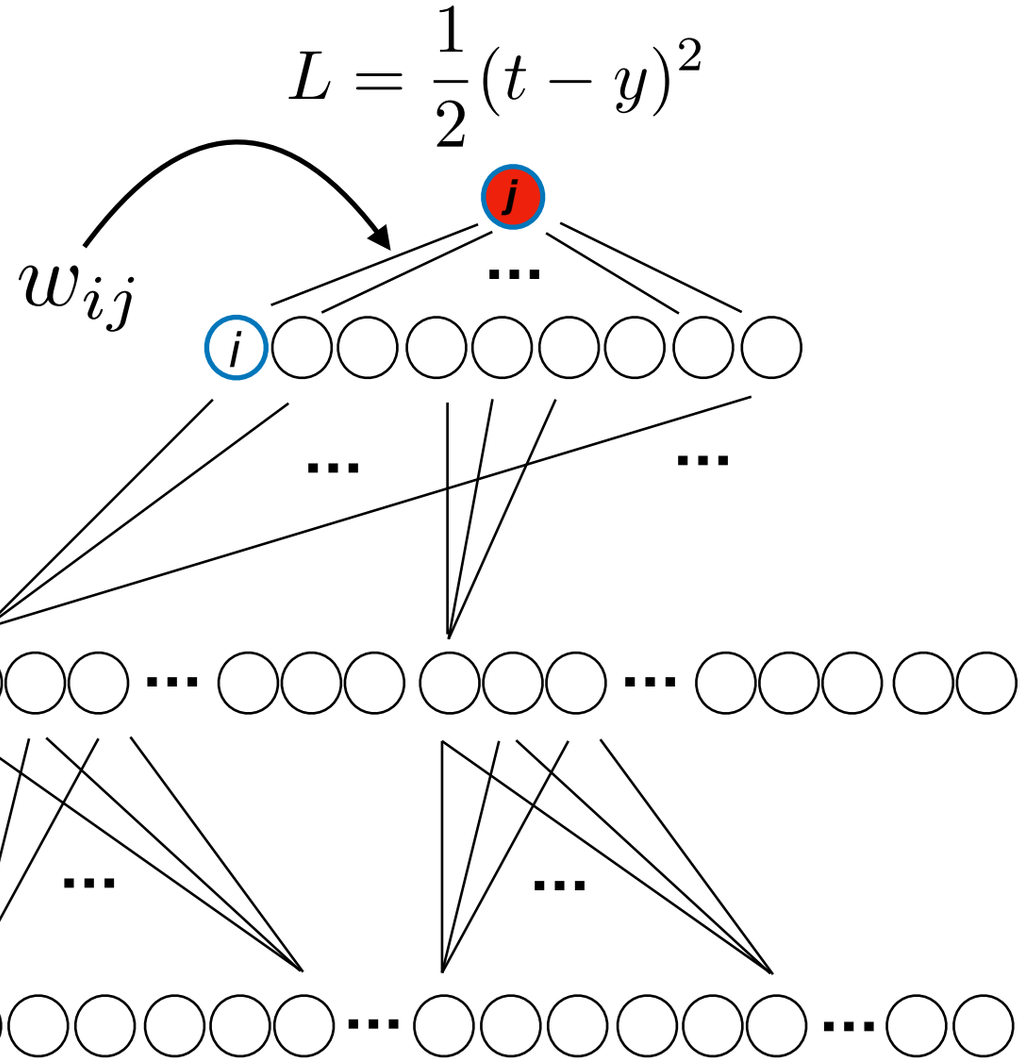
Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \boxed{\frac{\partial L}{\partial o_j}} \frac{\partial o_j}{\partial \text{inp}_j} \frac{\partial \text{inp}_j}{\partial w_{ij}}$$

$$o_j = y$$

$$\frac{\partial L}{\partial y} = y - t$$

$$L = \frac{1}{2}(t-y)^2$$



DeepMind

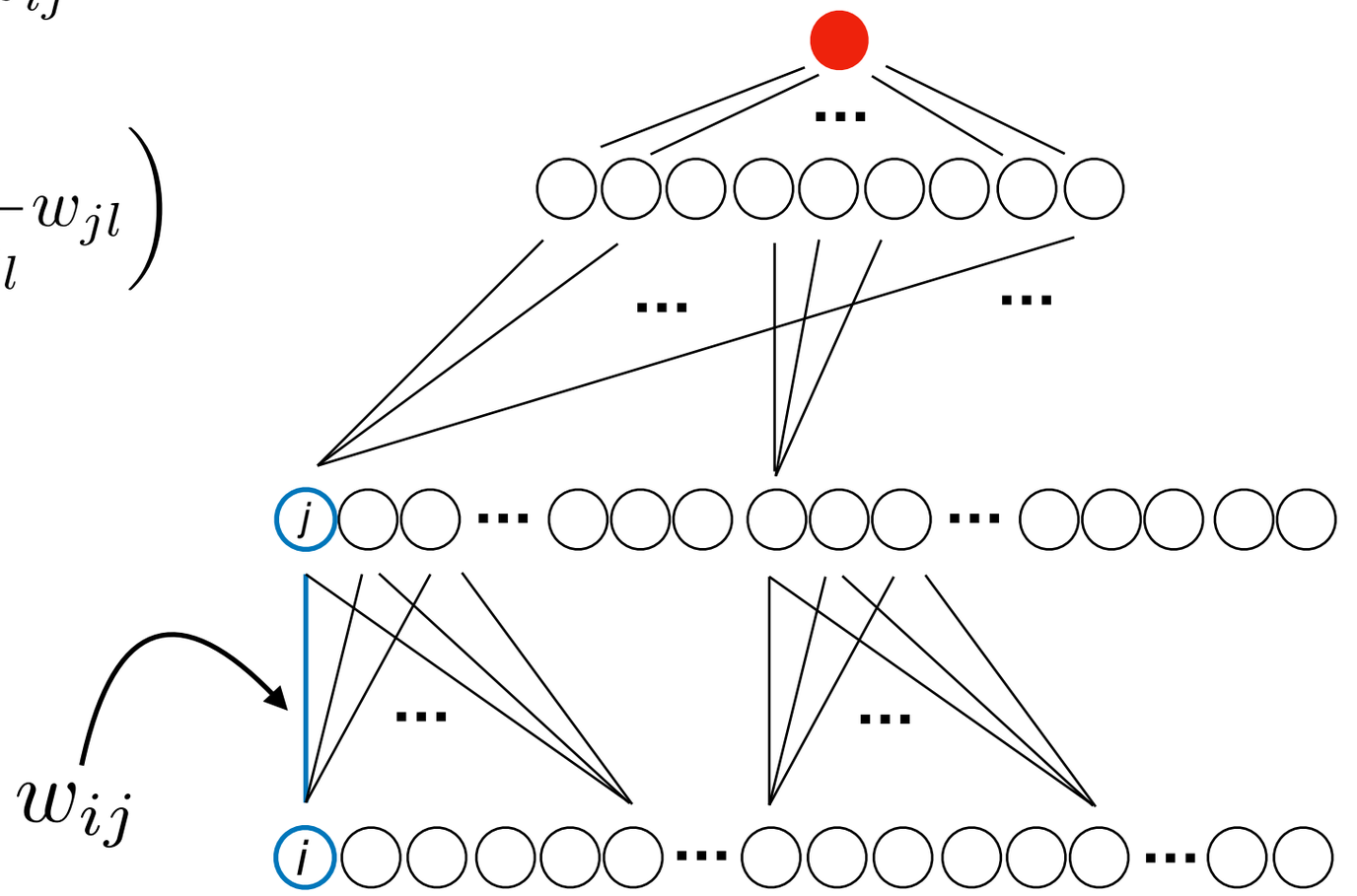
UNIVERSITY OF CAMBRIDGE

Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = \boxed{\frac{\partial L}{\partial o_j}} \frac{\partial o_j}{\partial \text{inp}_j} \frac{\partial \text{inp}_j}{\partial w_{ij}}$$

$$\frac{\partial L}{\partial o_j} = \sum_l \left(\frac{\partial L}{\partial o_l} \frac{\partial o_l}{\partial \text{inp}_l} w_{jl} \right)$$

$$L = \frac{1}{2}(t - y)^2$$



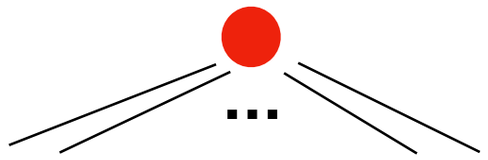
DeepMind

UNIVERSITY OF CAMBRIDGE

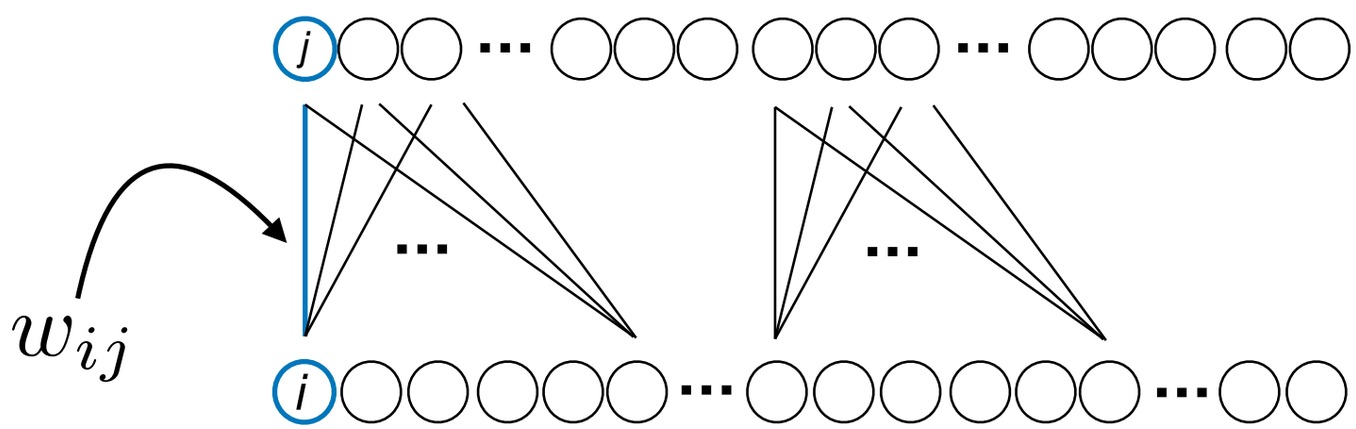
Backprop for MLPs

$$\frac{\partial L}{\partial w_{ij}} = o_i \delta_j$$

$$L = \frac{1}{2}(t - y)^2$$



$$\delta_j = \begin{cases} (o_j - t) o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_l \delta_l w_{jl}\right) o_j (1 - o_j) & \text{otherwise} \end{cases}$$



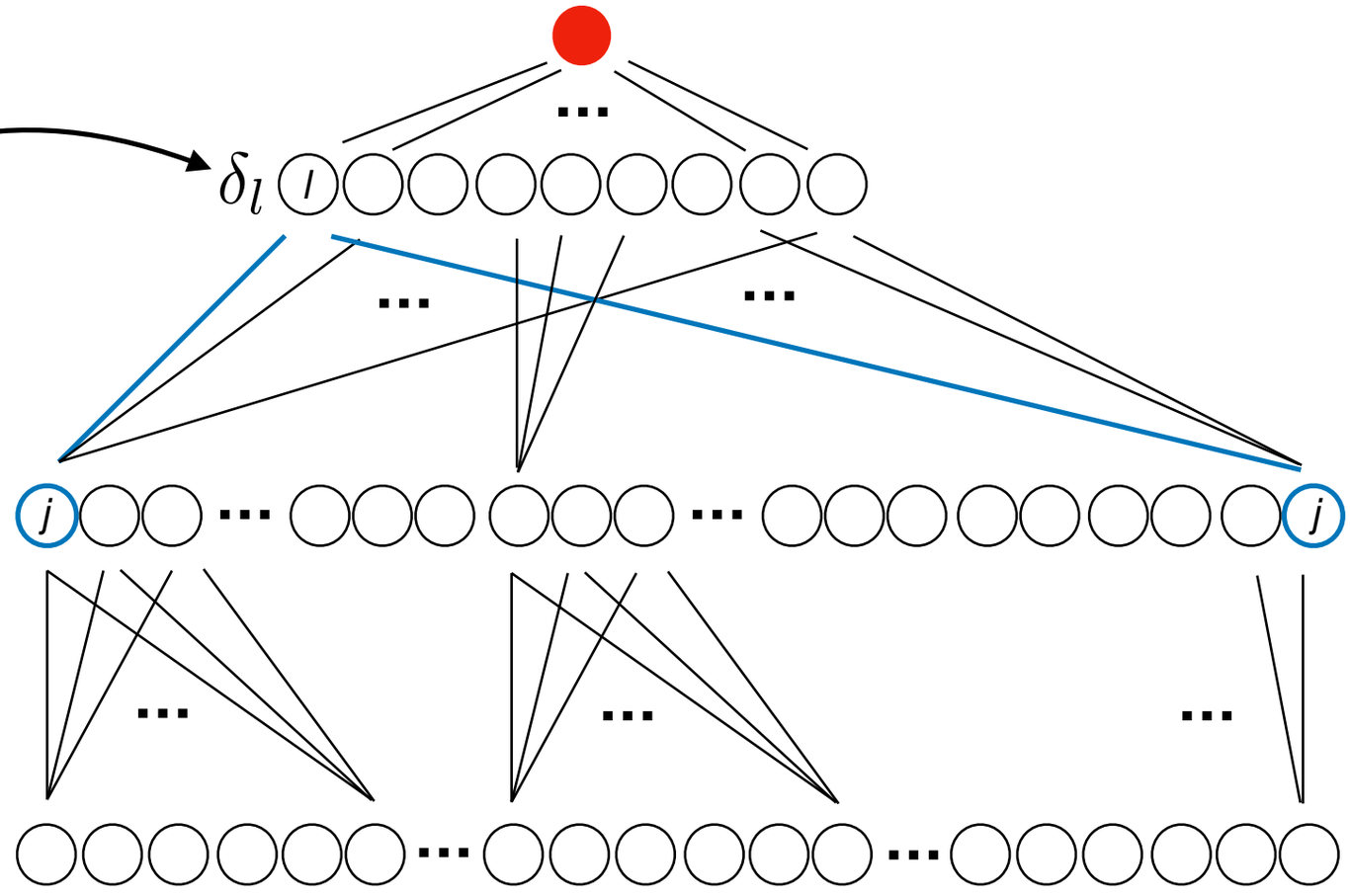
DeepMind

UNIVERSITY OF CAMBRIDGE

Backprop for MLPs

$$\delta_j = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_l \delta_l w_{jl}\right) o_j (1 - o_j) & \text{otherwise} \end{cases}$$

Only compute this once and store in a DP table



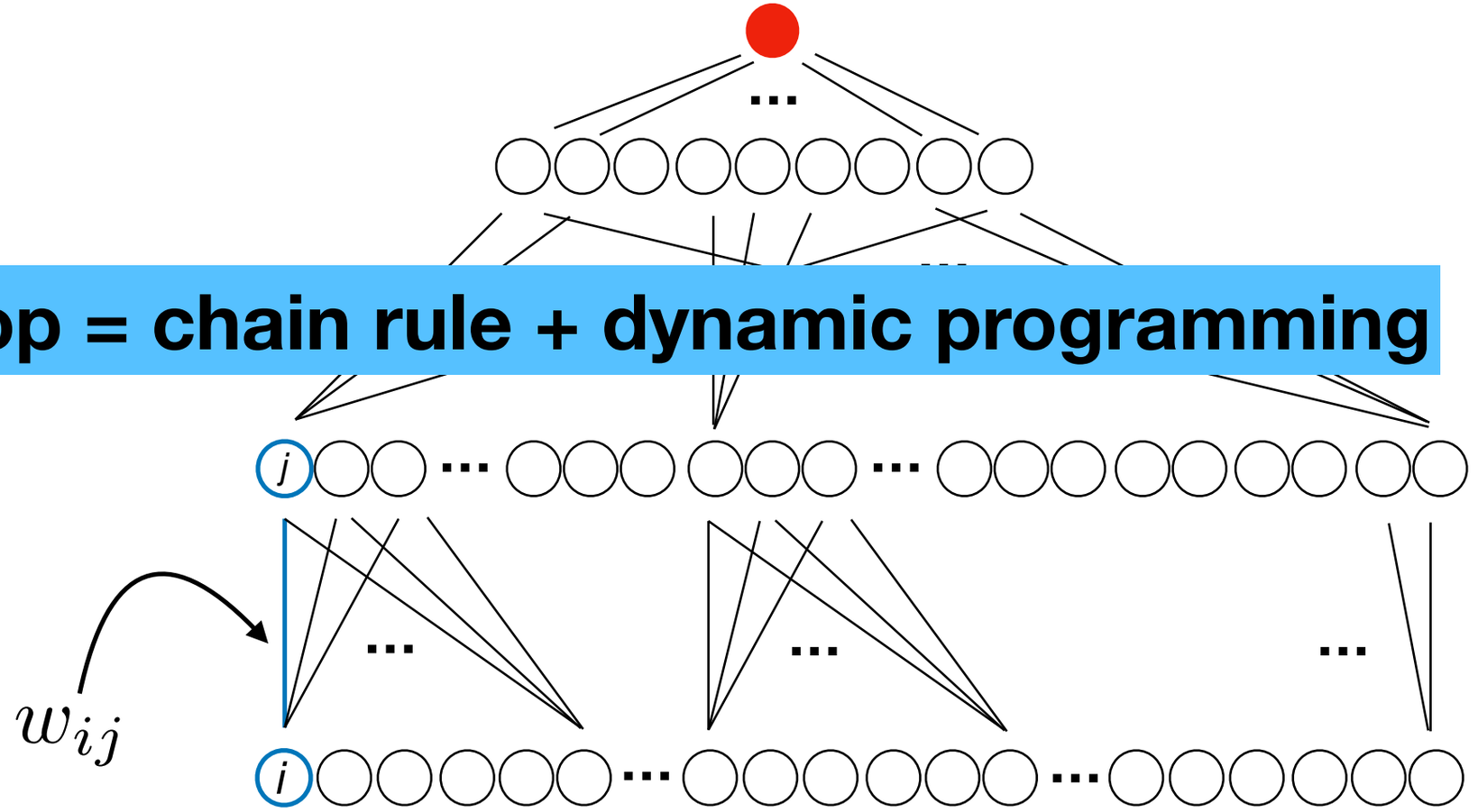
DeepMind

UNIVERSITY OF CAMBRIDGE

Backprop for MLPs

$$\delta_j = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_l \delta_l w_{jl}\right) o_j (1 - o_j) & \text{otherwise} \end{cases}$$

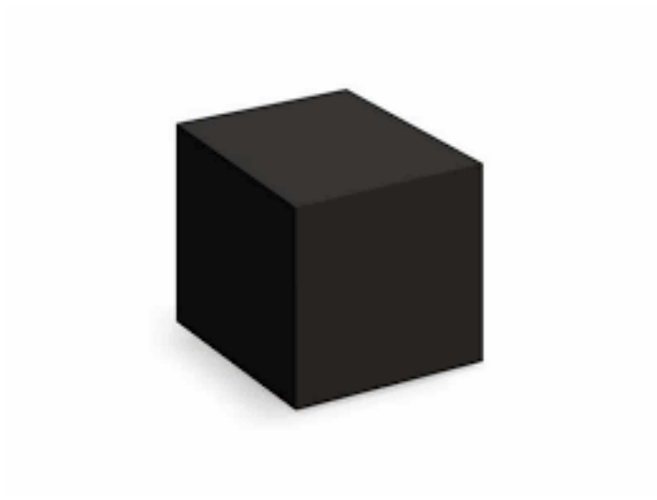
Backprop = chain rule + dynamic programming



DeepMind

UNIVERSITY OF CAMBRIDGE

Backprop in Tensorflow



DeepMind

UNIVERSITY OF CAMBRIDGE

Estimating Gradients

$$g = \nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta)$$ Batch learning

$$\hat{g} = \frac{1}{|B|} \sum_{(\mathbf{x}^{(j)}, y^{(j)}) \in B} \nabla_\theta L(\mathbf{x}^{(j)}, y^{(j)}, \theta)$$ Mini-batch learning

In “pure” SGD, $|B| = 1$ (!)

DeepMind

UNIVERSITY OF CAMBRIDGE

Selecting the Mini-batch

- Larger batches give more accurate estimate of the gradient, but with less than linear returns (p.271 DL book)

- Multicore architectures are usually underutilized by extremely small batch sizes (p.272 DL book)

- Small batches can offer a regularizing effect, perhaps due to the noise they add to the learning process (p.272)

- In practice, shuffle the training data, then cycle through the same mini-batches on each epoch

Stochastic Gradient Descent (SGD)

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \max(k/\tau, 1)$$

p.286 of the Deep Learning textbook

DeepMind

UNIVERSITY OF CAMBRIDGE

Lots of Other Alternatives

- Gradient descent optimization algorithms
  - Momentum
  - Nesterov accelerated gradient
  - Adagrad
  - Adadelta
  - RMSprop
  - Adam
  - AdaMax
  - Nadam
  - .....

DeepMind

UNIVERSITY OF CAMBRIDGE

Optimization in Tensorflow

```
self._lr = tf.Variable(0.0, trainable=False)
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(self._cost, tvars),
                                  config.max_grad_norm)
optimizer = tf.train.GradientDescentOptimizer(self._lr)
self._train_op = optimizer.apply_gradients(
    zip(grads, tvars),
    global_step=tf.train.get_or_create_global_step())

self._new_lr = tf.placeholder(
    tf.float32, shape=[], name="new_learning_rate")
self._lr_update = tf.assign(self._lr, self._new_lr)
```

from the Tensorflow RNN LM tutorial

DeepMind

UNIVERSITY OF CAMBRIDGE

Parameter Initialization (matters)

Imagine you're trying to get to the top of Arthur's Seat in the fog (*haar*); where you start will be crucial to success

DeepMind

UNIVERSITY OF CAMBRIDGE

Parameter Initialization (matters)

- Typically the weights are initialized randomly, biases are set to heuristically chosen constants
- Random weights are drawn from Gaussian or Uniform
  - e.g. $U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$ where m is the number of inputs

p.292 of the Deep Learning textbook

DeepMind

UNIVERSITY OF CAMBRIDGE

Regularization

$$\tilde{J}(\theta; \mathbf{x}, y) = J(\theta; \mathbf{x}, y) + \alpha\Omega(\theta)$$

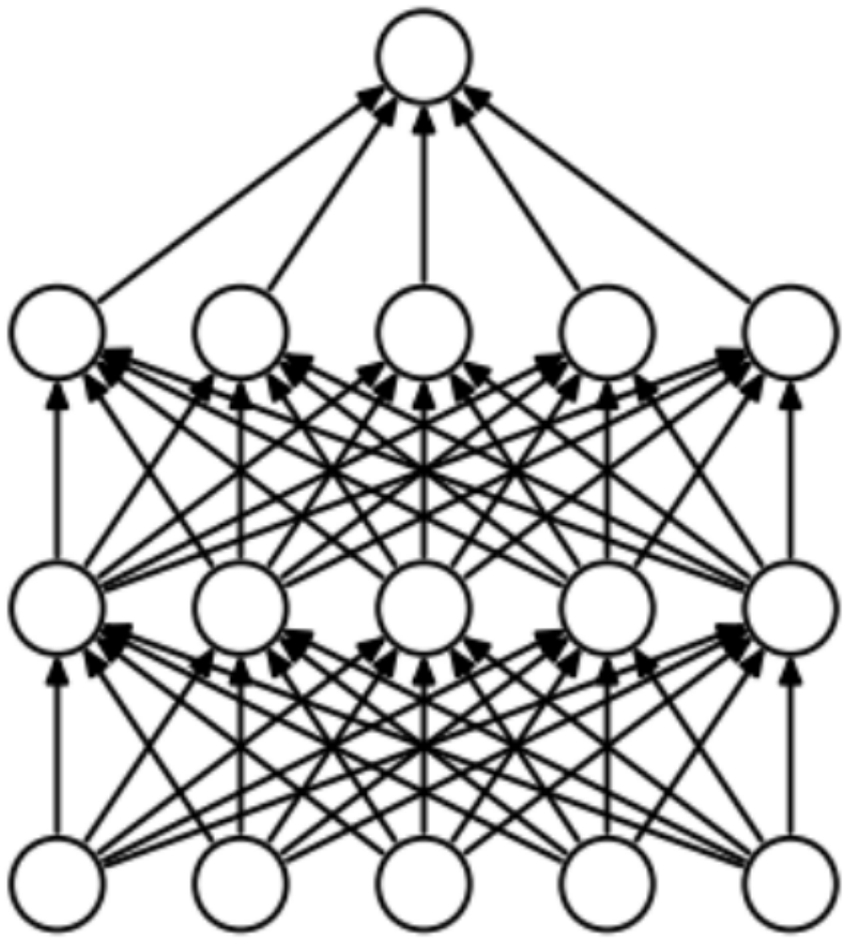
$$\Omega(\theta) = \frac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\mathbf{w}^\top\mathbf{w}$$

L^2 parameter norm penalty (weight decay)

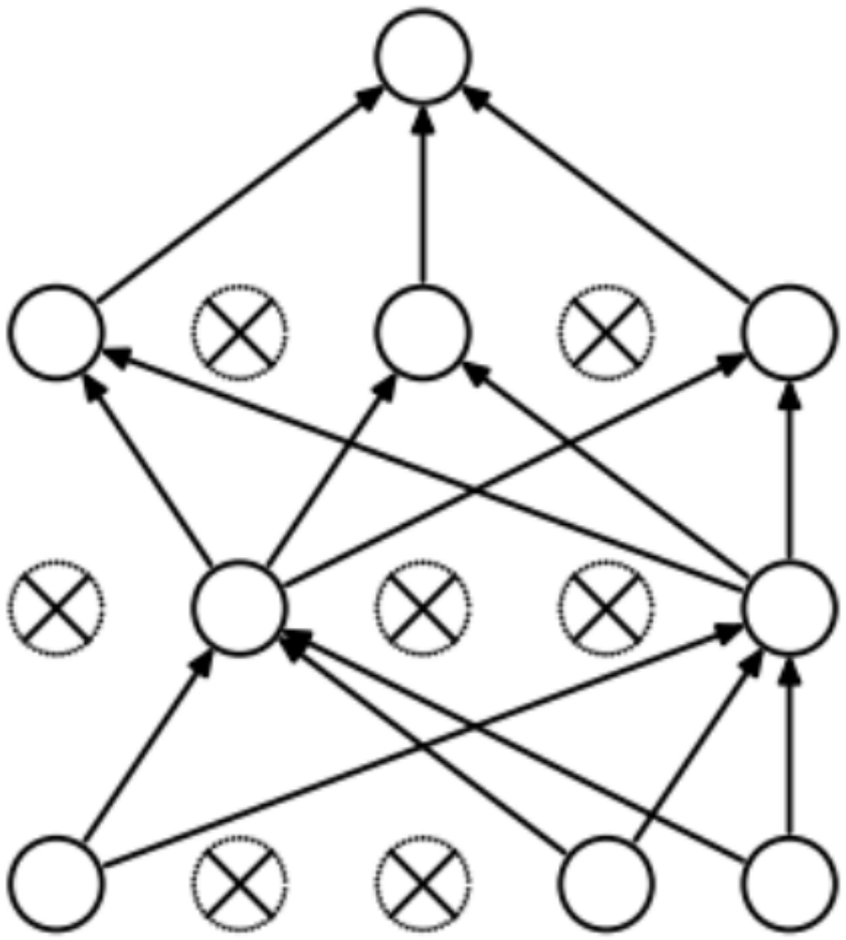
DeepMind

UNIVERSITY OF CAMBRIDGE

Dropout



(a) Standard Neural Net



(b) After applying dropout.

DeepMind

UNIVERSITY OF CAMBRIDGE

References

- *How the backpropagation algorithm works*, Michael Nielsen

  http://neuralnetworksanddeeplearning.com/chap2.html

- colah's blog

  http://colah.github.io/posts/2015-08-Backprop/

DeepMind

UNIVERSITY OF CAMBRIDGE