

Deep learning

3. Loss functions, optimization, regularization

A course @EDHEC
by Romain Tavenard (Prof. @Univ. Rennes 2)

Optimization in general

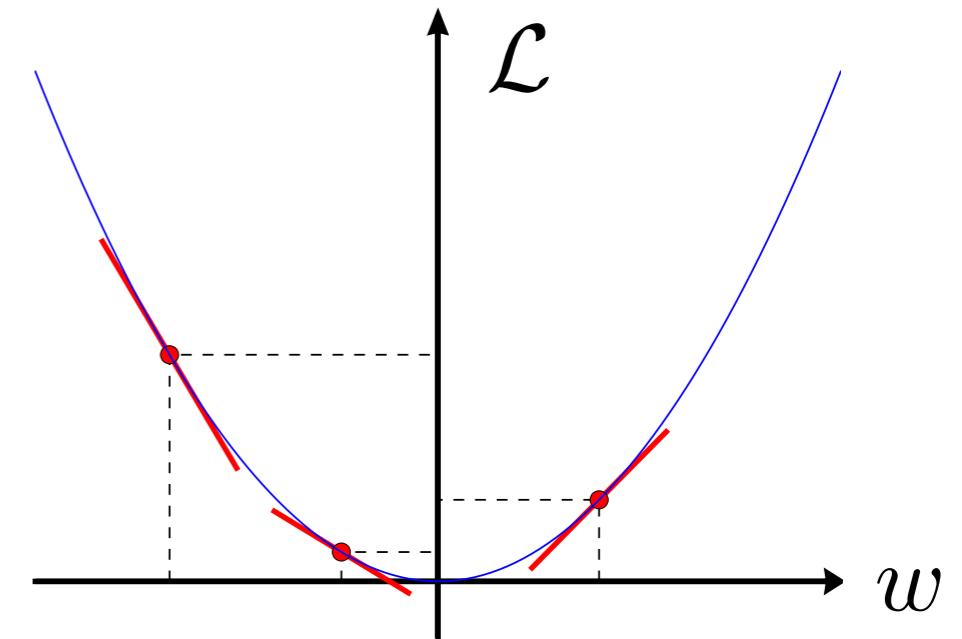
- **Goal:** Tune model parameters so as to minimize error
- Measuring error
 - Through a cost function / loss function
 - Typical example: Mean Squared Error in regression settings

Optimization

Gradient descent

1. Pick a (differentiable) loss function to be minimized

$$\begin{aligned}\text{eg. } \mathcal{L}(w, \{x_i, y_i\}) &= \frac{1}{n} \sum_{i=1}^n \mathcal{L}_i(w, x_i, y_i) \\ &= \frac{1}{n} \sum_{i=1}^n (\varphi(w^t x_i) - y_i)^2\end{aligned}$$



2. Use gradient descent

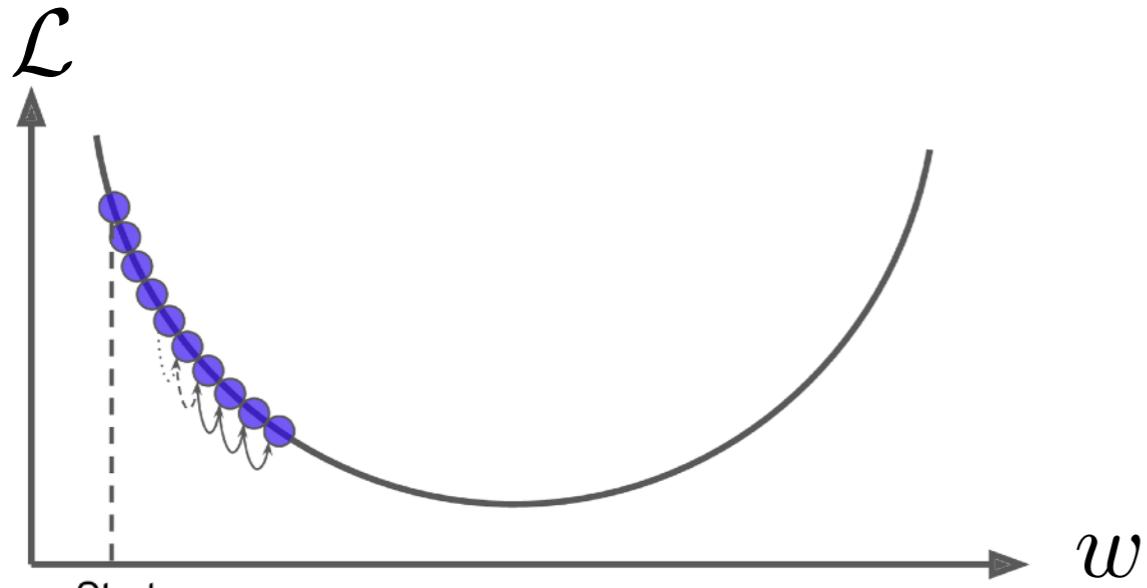
$$w^{(t+1)} \leftarrow w^{(t)} - \rho \nabla_w \mathcal{L}(w^{(t)})$$

Algorithm 1: Gradient Descent

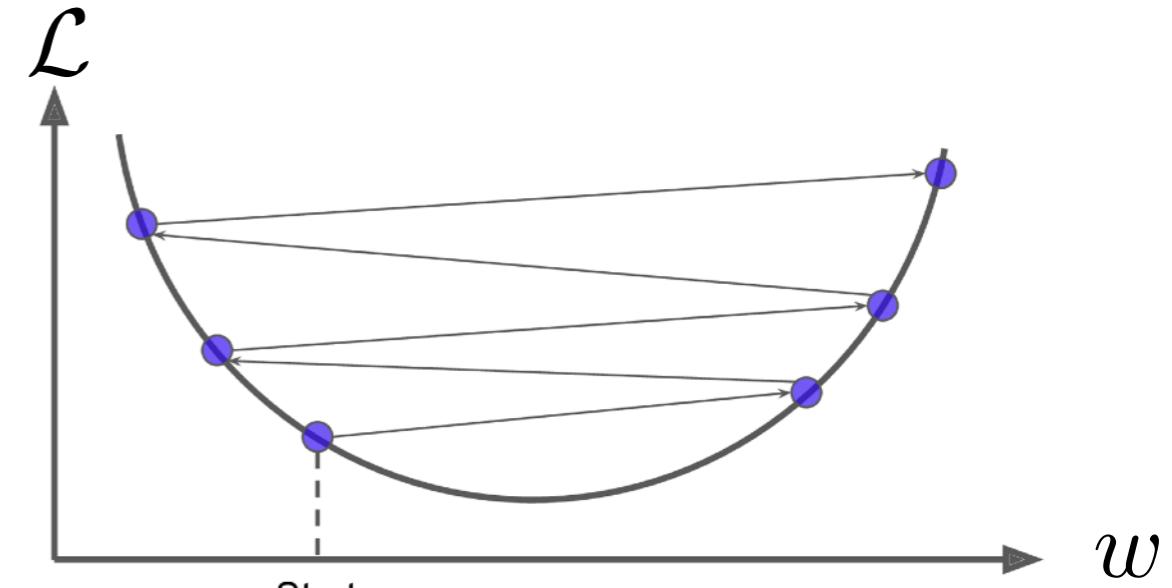
```
Data:  $\mathcal{D}$ : a dataset
Initialize weights
for  $e = 1..E$  do
    //  $e$  is called an epoch
    for  $(x_i, y_i) \in \mathcal{D}$  do
        Compute prediction  $\hat{y}_i = h(x_i)$ 
        Compute gradient  $\nabla_w \mathcal{L}_i$ 
    end
    Compute overall gradient  $\nabla_w \mathcal{L} = \frac{1}{n} \sum_i \nabla_w \mathcal{L}_i$ 
    Update parameter  $w$  using  $\nabla_w \mathcal{L}$ 
end
```

Optimization

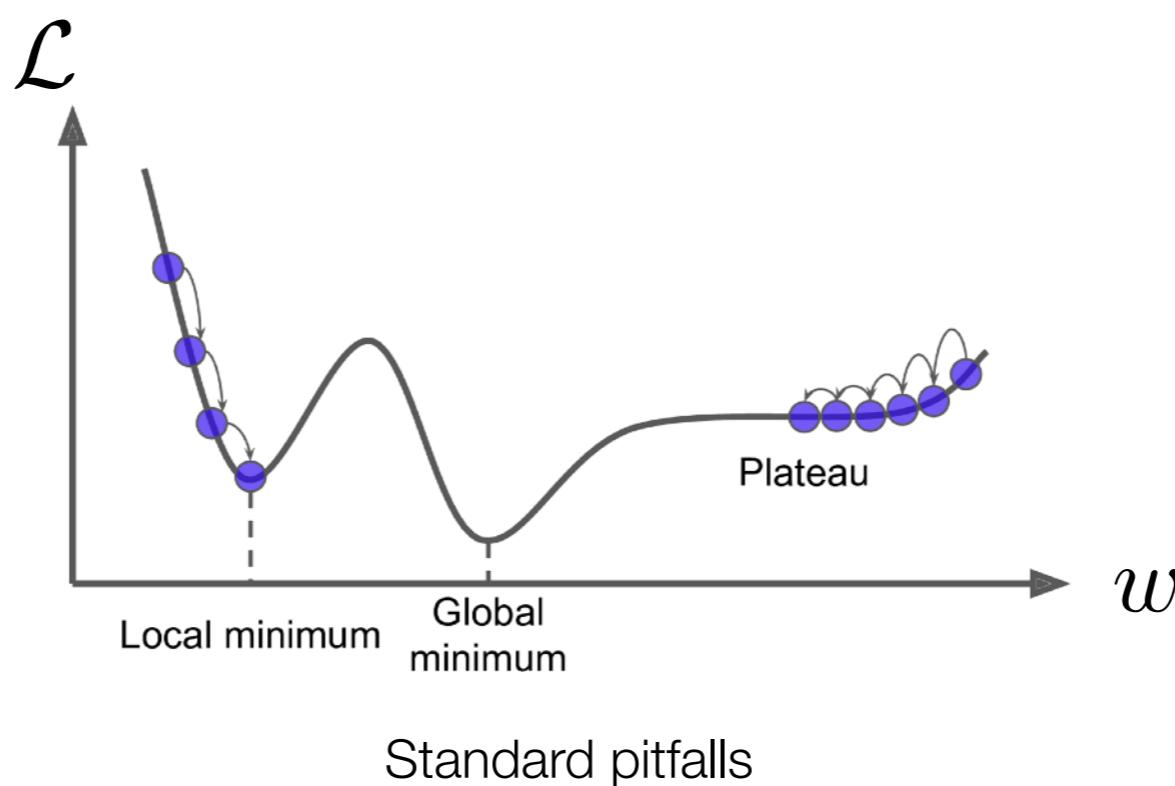
Gradient descent in Real Life



Learning rate is too small



Learning rate is too large



Source: "Hands-On Machine Learning with Scikit-Learn and TensorFlow", A. Géron

Optimization

Stochastic Gradient Descent

Algorithm 1: Gradient Descent

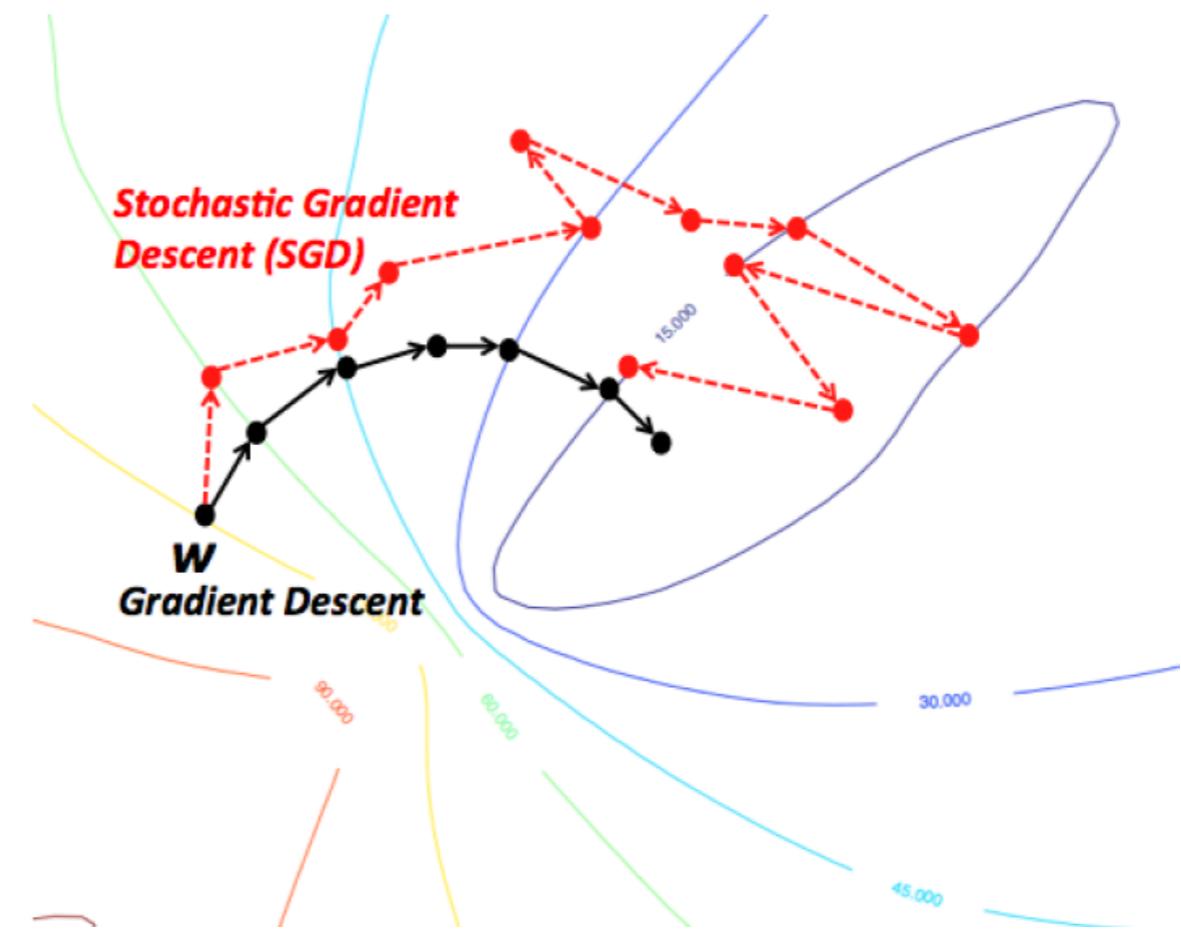
Data: \mathcal{D} : a dataset
Initialize weights
for $e = 1..E$ **do**
 // e is called an epoch
 for $(x_i, y_i) \in \mathcal{D}$ **do**
 Compute prediction $\hat{y}_i = h(x_i)$
 Compute gradient $\nabla_w \mathcal{L}_i$
 end
 Compute overall gradient $\nabla_w \mathcal{L} = \frac{1}{n} \sum_i \nabla_w \mathcal{L}_i$
 Update parameter w using $\nabla_w \mathcal{L}$
end

Algorithm 2: Mini-Batch Stochastic Gradient Descent

Data: \mathcal{D} : a dataset
Initialize weights
for $e = 1..E$ **do**
 // e is called an epoch
 for $t = 1..n_b$ **do**
 // t is called an iteration
 for $i = 1..m$ **do**
 Draw (x_i, y_i) without replacement from t -th minibatch of \mathcal{D}
 Compute prediction $\hat{y}_i = h(x_i)$
 Compute gradient $\nabla_w \mathcal{L}_i$
 end
 Compute gradient for the t -th minibatch $\nabla_w \mathcal{L}_{(t)} = \frac{1}{m} \sum_i \nabla_w \mathcal{L}_i$
 Update parameter w using $\nabla_w \mathcal{L}_{(t)}$
 end
end

Optimization: Gradient Descent vs Stochastic Gradient Descent

- Cons
 - Subject to high variance
- Pros
 - Faster weight update (each sample, or each mini batch)
 - Escape local minima in non-convex settings



Source: wikidocs.net/3413

Optimization

SGD variants: a focus on Adam

- Adam uses ideas from
 - Momentum [[link to distill](#)]
 - AdaGrad

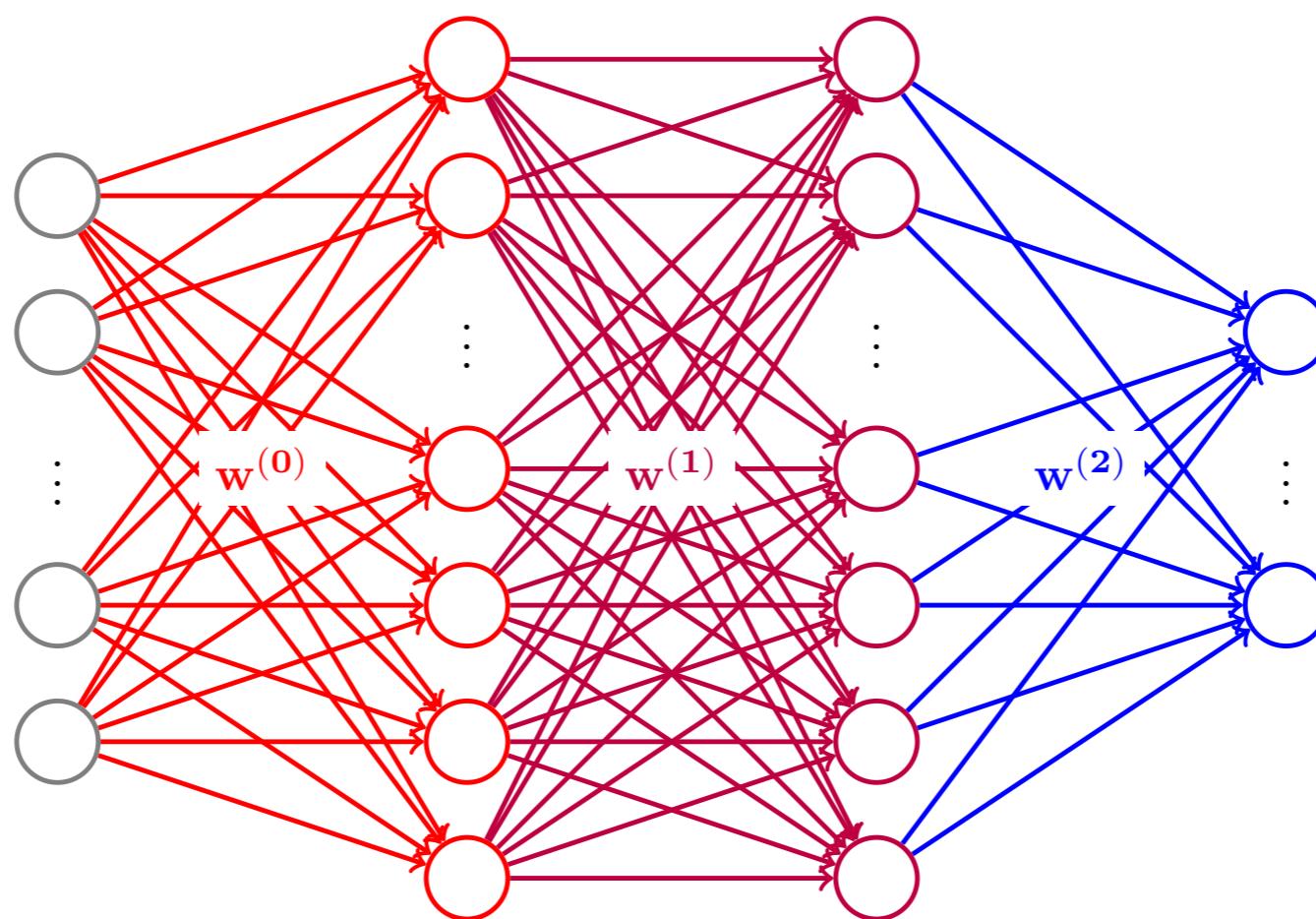
$$\mathbf{m}^{(t+1)} \propto \beta_1 \mathbf{m}^{(t)} + (1 - \beta_1) \nabla_w \mathcal{L}$$

$$\mathbf{s}^{(t+1)} \propto \beta_2 \mathbf{s}^{(t)} + (1 - \beta_2) \nabla_w \mathcal{L} \otimes \nabla_w \mathcal{L}$$

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \rho \mathbf{m}^{(t+1)} \oslash \sqrt{\mathbf{s}^{(t+1)} + \epsilon}$$

Optimizing multi-layer perceptron parameters

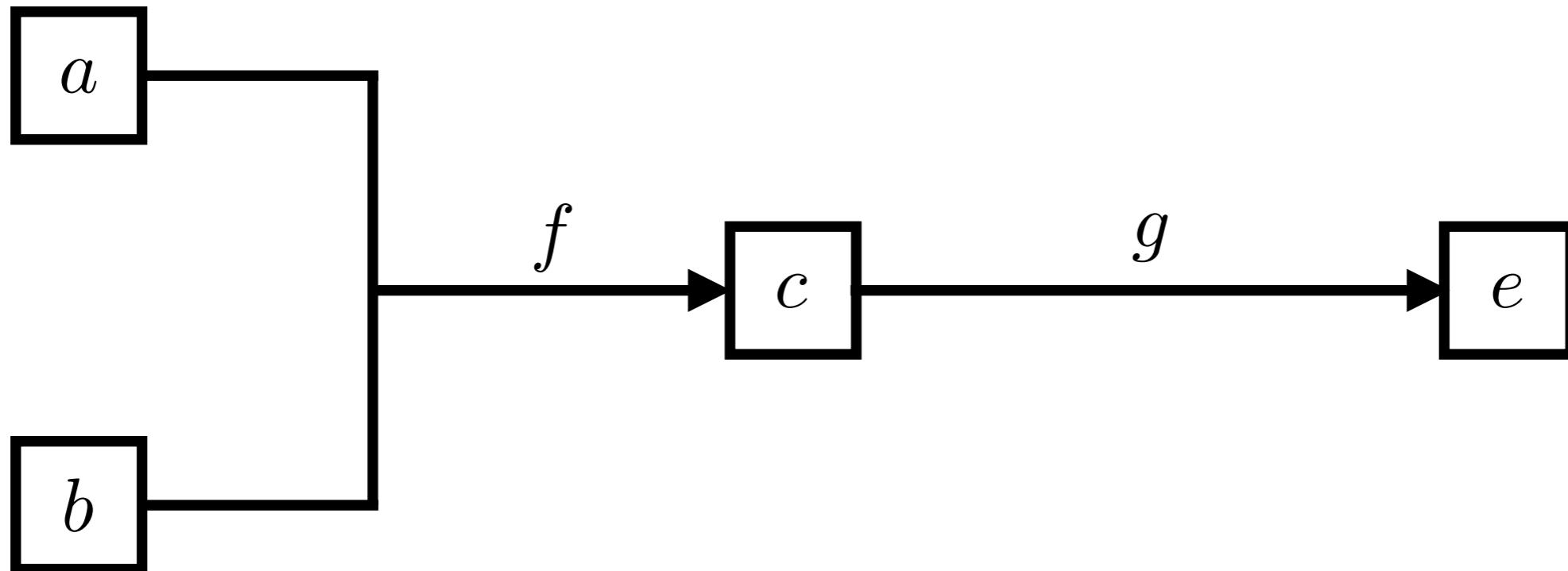
- Who wants to compute gradients by hand for such networks (and deeper ones)?



$$\hat{\mathbf{y}} = \varphi \left[\mathbf{w}^{(2)} \varphi \left(\mathbf{w}^{(1)} \varphi \left(\mathbf{w}^{(0)} \mathbf{x} + b^{(0)} \right) + b^{(1)} \right) + b^{(2)} \right]$$

Optimizing multi-layer perceptron parameters

Automatic differentiation to the rescue!

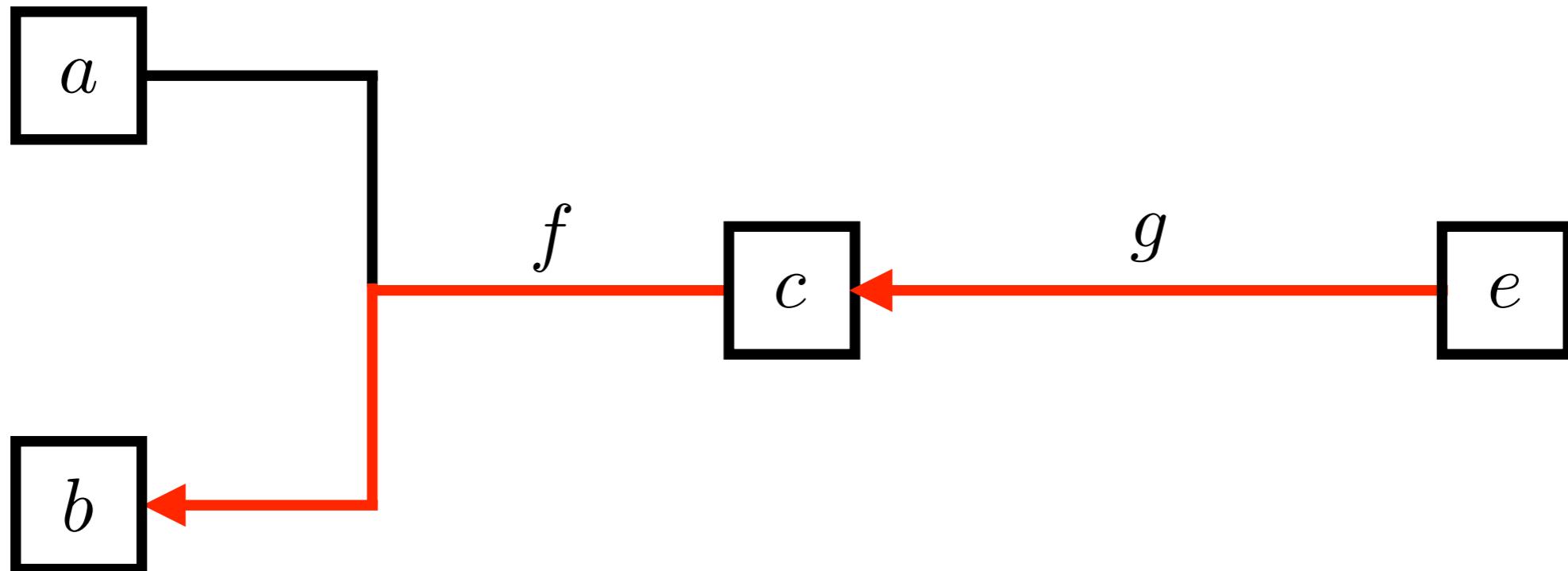


$$c = f(a, b)$$

$$e = g(c)$$

Optimizing multi-layer perceptron parameters

Automatic differentiation to the rescue!



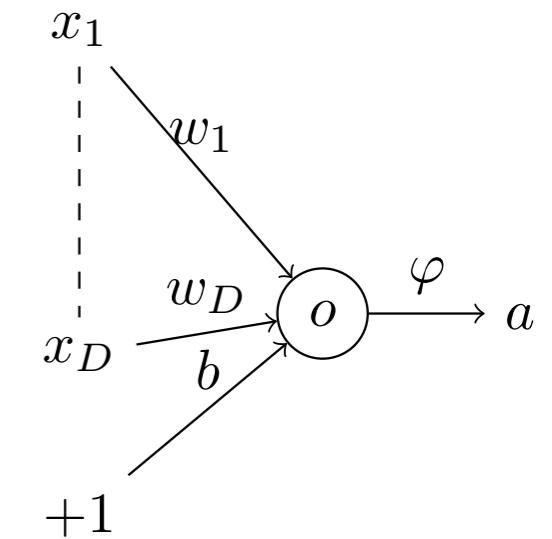
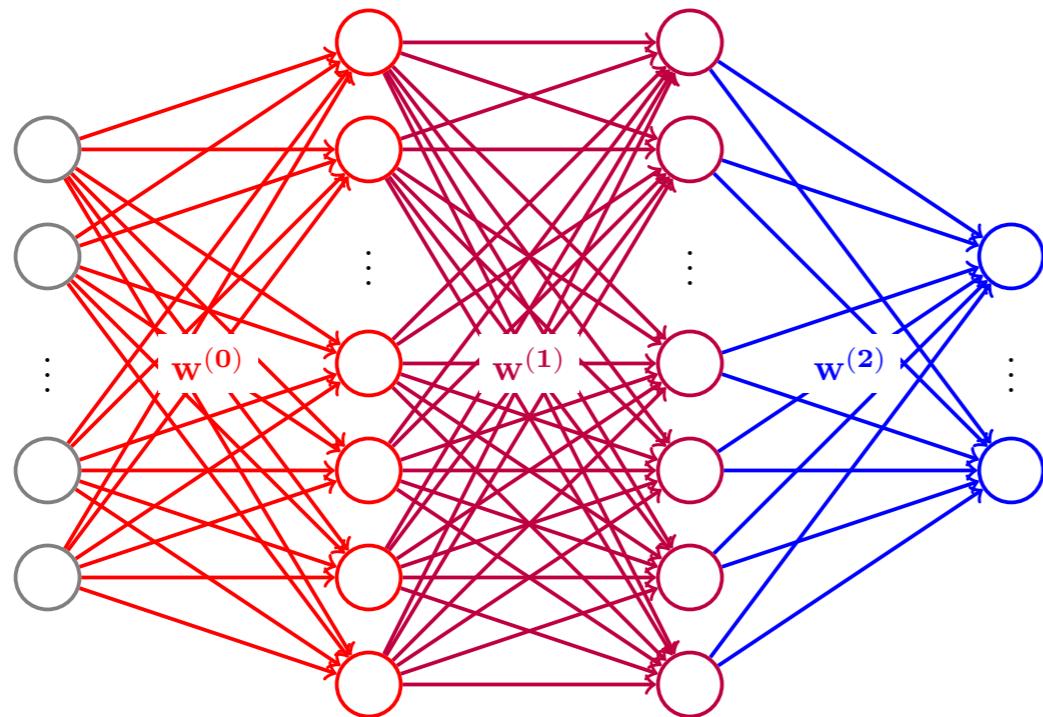
$$c = f(a, b)$$

$$e = g(c)$$

$$\frac{\partial e}{\partial b} = \underbrace{\frac{\partial e}{\partial c} \Big|_{c=c_0}}_{g'(c_0)} \cdot \frac{\partial c}{\partial b} \Big|_{b=b_0}$$

Optimization

Neural networks and back-propagation



$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial w^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial w^{(1)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$

$$\frac{\partial a^{(l)}}{\partial o^{(l)}} = \varphi'(o^{(l)})$$

$$\frac{\partial o^{(l)}}{\partial a^{(l-1)}} = w^{(l-1)}$$

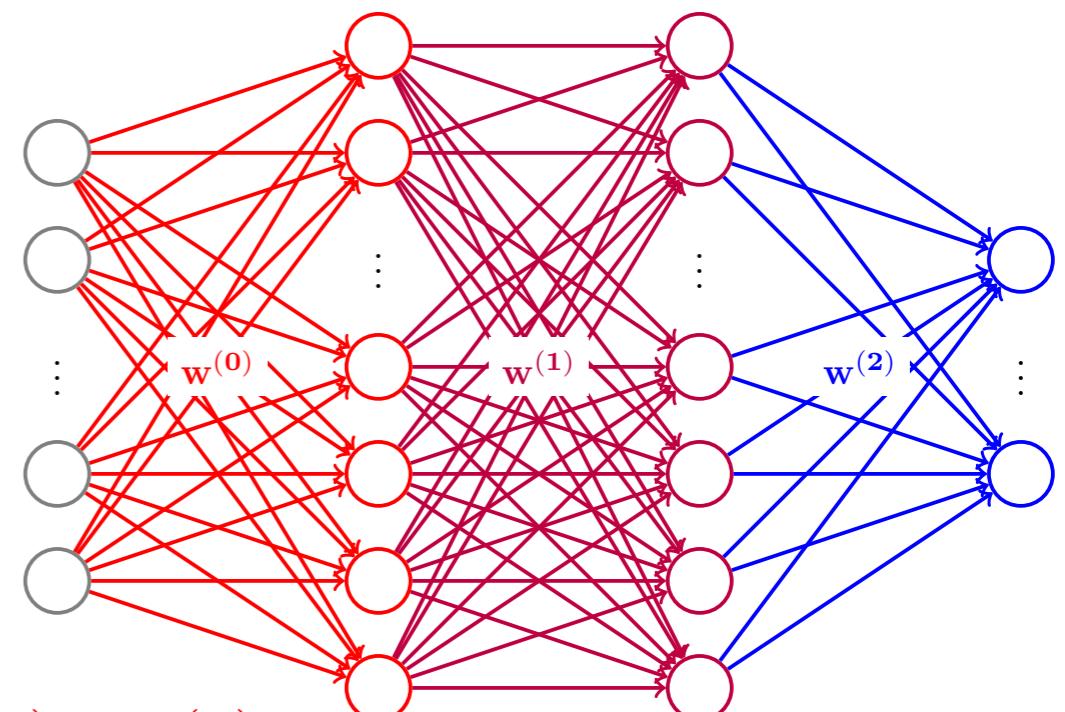
A history of Convolutional neural networks (CNN)

Deeper and deeper networks

- Deeper networks = higher-level understanding
- Main limitation: vanishing gradients

$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial w^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$



$$\frac{\partial a^{(l)}}{\partial o^{(l)}} = \varphi'(o^{(l)})$$

ReLU
as default
activation function

Neural networks and back-propagation: Losses

$$\frac{\partial \mathcal{L}}{\partial w^{(0)}} = \frac{\partial \mathcal{L}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial o^{(2)}} \frac{\partial o^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial w^{(0)}}$$

- Requirement
 - \mathcal{L} should be differentiable wrt. to the net's output
- Standard losses
 - Mean Squared Error (MSE) for regression
$$\mathcal{L}(x_i, y_i; \theta) = (m(x_i; \theta) - y_i)^2$$
 - Cross-entropy for classification
$$\mathcal{L}(x_i, y_i; \theta) = -\log P_\theta(y = y_i | x_i)$$

Optimization

Over-parametrization in deep learning

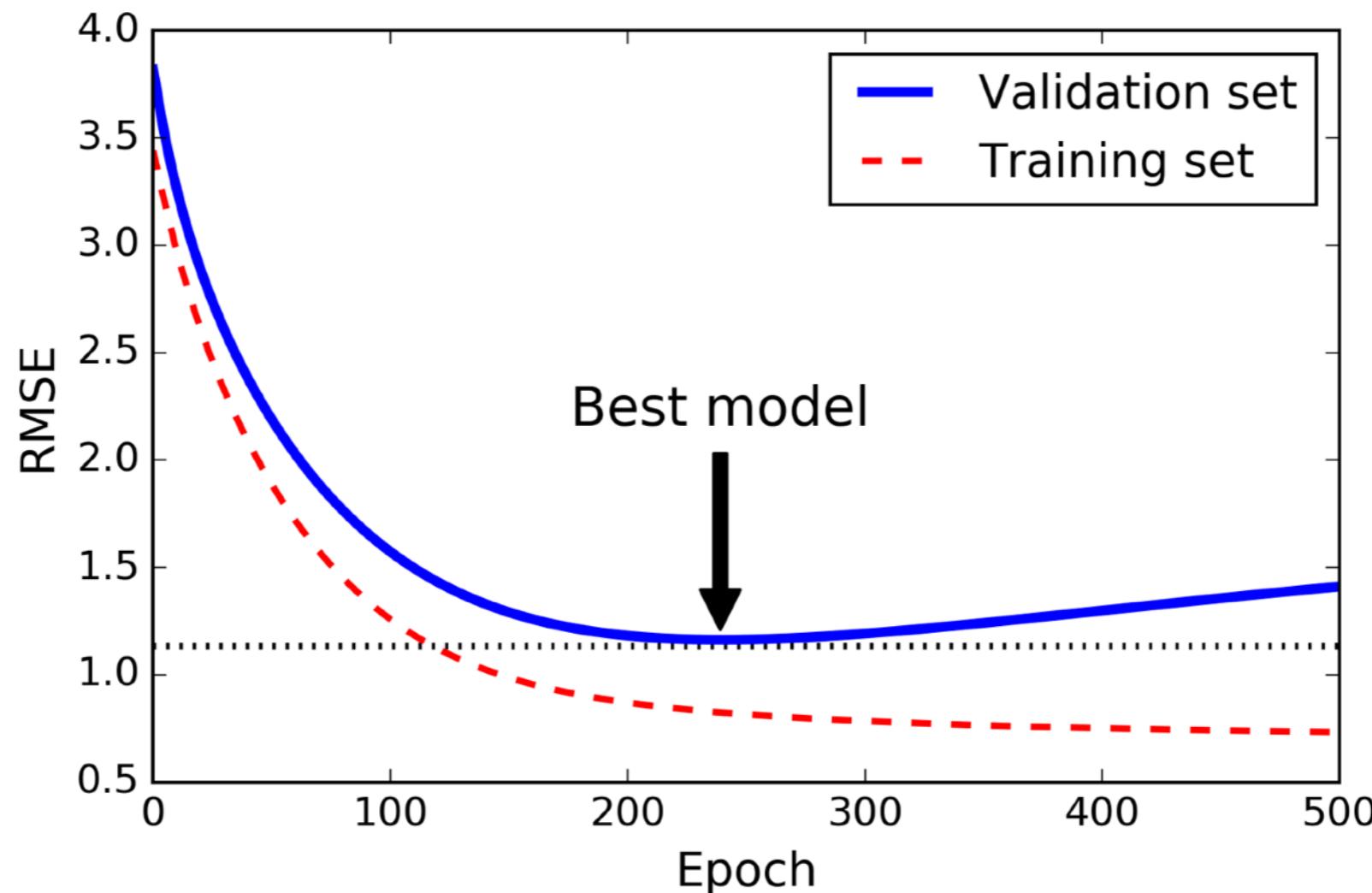
- Optimization (SGD) to minimize a loss function
 - Larger & deeper nets improve (training) performance
 - Risks over-fitting

$$\arg \min_{\theta} \sum_{(x_i, y_i) \in \mathcal{D}_t} \mathcal{L}(x_i, y_i; \theta) \neq \arg \min_{\theta} \mathbb{E}_{x, y \sim \mathcal{D}} \mathcal{L}(x, y; \theta)$$

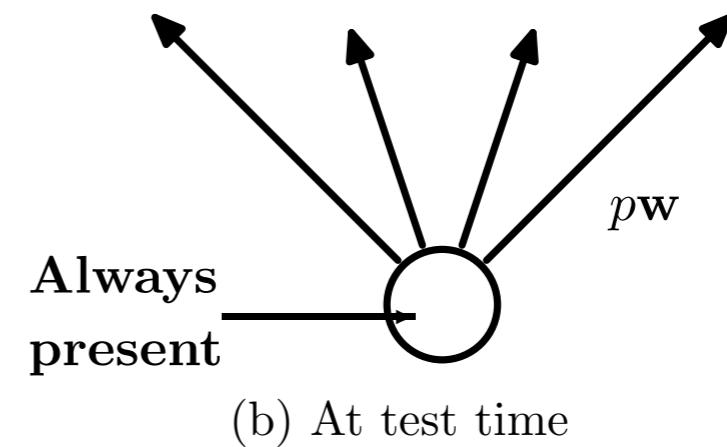
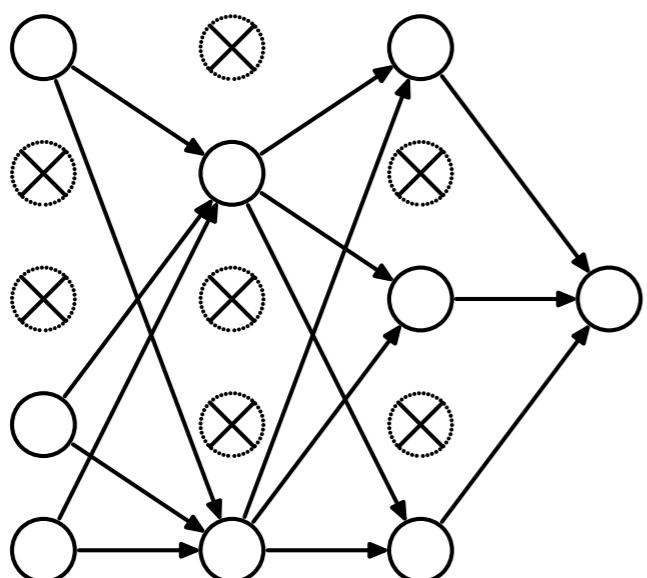
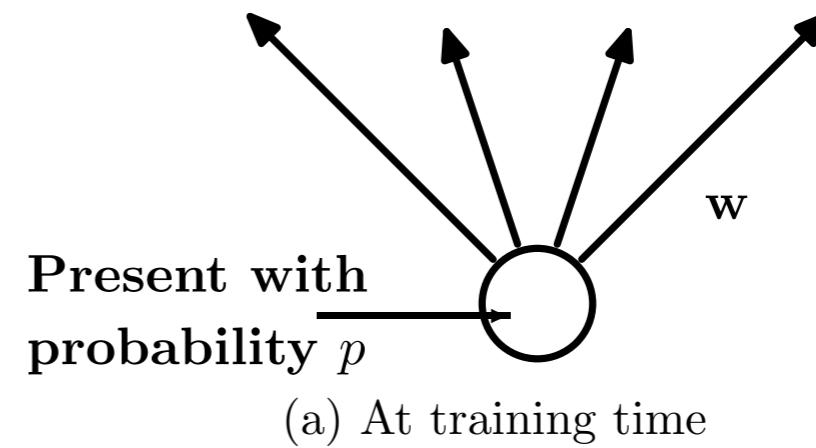
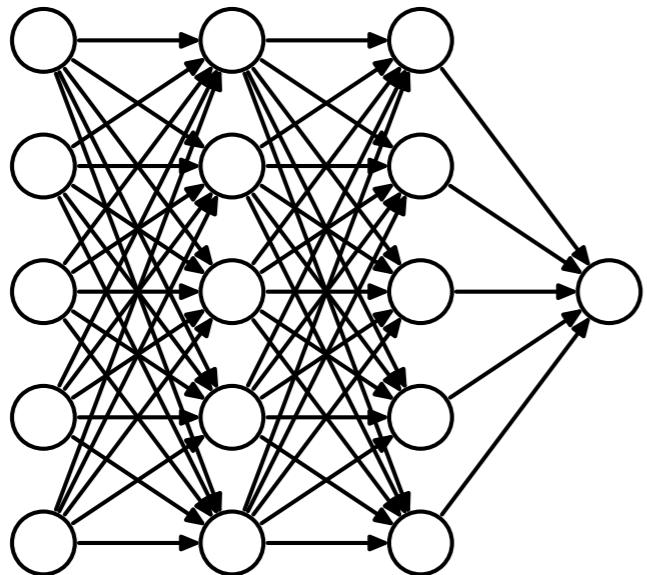
- Regularization tricks
 - L2 penalty on weights (cf. Ridge regression)
 - Early stopping (cf. Gradient boosting)
 - Dropout (relates to Random Forests)

Optimization

Regularization: Early Stopping



Optimization Regularization: Dropout



Images from [Srivastava *et al.*, 2014]

Conclusion

- Stochastic Gradient Descent
 - Gradient estimates computed on mini-batches
 - More frequent updates
 - Adam is an extremely powerful variant
- Loss functions
 - Mean Squared Error for regression
 - Logistic loss for classification
- Neural networks are usually used in over-parametrized mode
 - Need regularization
 - Need to check performance on hold-out data (validation set)