

Neural networks

Training neural networks - empirical risk minimization

NEURAL NETWORK

Topics: multilayer neural network

- Could have L hidden layers:

- layer input activation for $k > 0$ ($\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$)

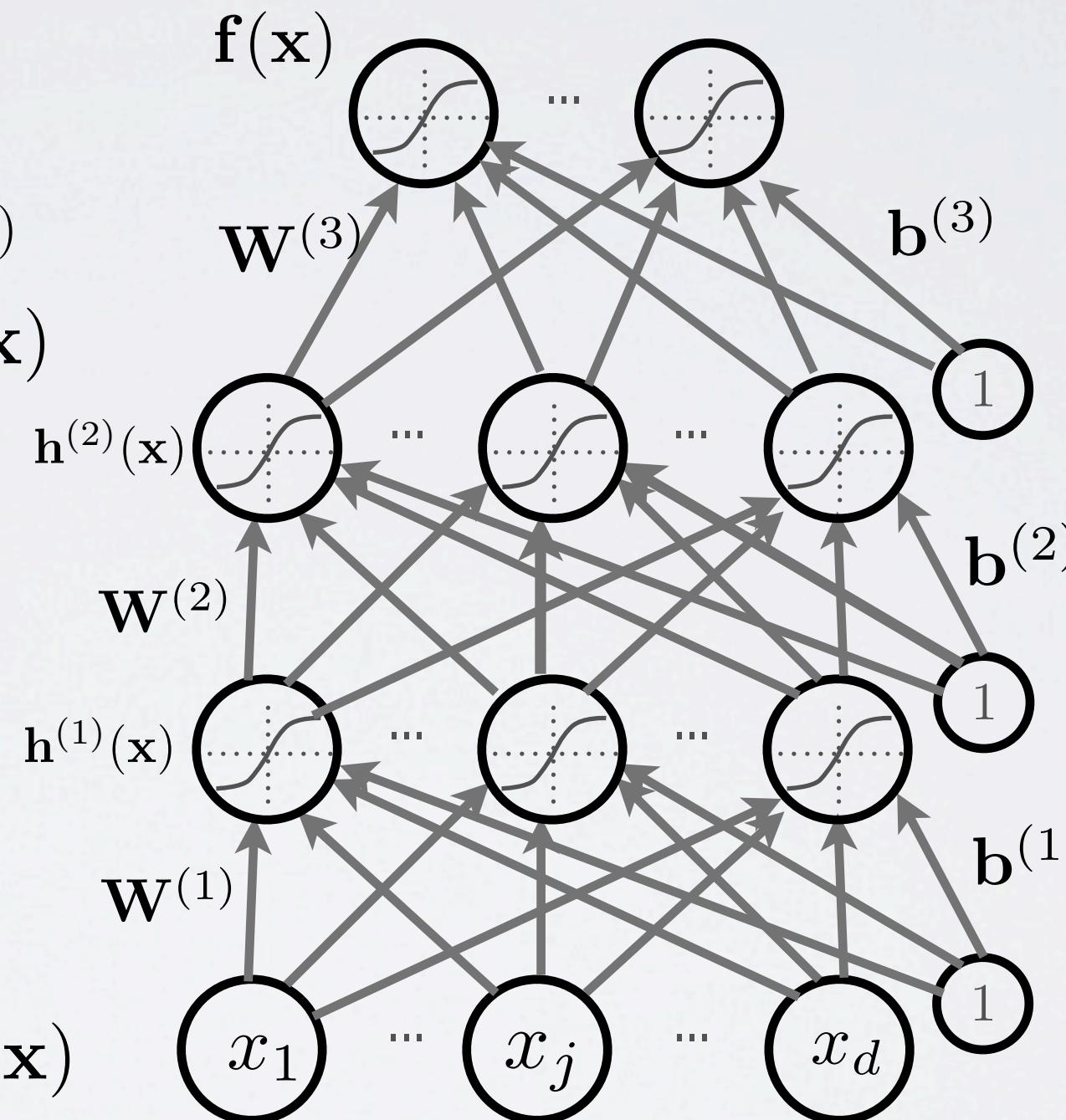
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- output layer activation ($k=L+1$):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



MACHINE LEARNING

Topics: empirical risk minimization, regularization

- Empirical risk minimization
 - ▶ framework to design learning algorithms

$$\arg \min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

- ▶ $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is a loss function
- ▶ $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- Learning is cast as optimization
 - ▶ ideally, we'd optimize classification error, but it's not smooth
 - ▶ loss function is a surrogate for what we truly should optimize (e.g. upper bound)

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

training epoch
 =
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

Neural networks

Training neural networks - loss function

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

LOSS FUNCTION

Topics: loss function for classification

- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
 - ▶ we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set
- To frame as minimization, we minimize the negative log-likelihood

$$l(\mathbf{f}(\mathbf{x}), y) = - \sum_c 1_{(y=c)} \log f(\mathbf{x})_c = - \log f(\mathbf{x})_y$$

natural log (ln)



- ▶ we take the log to simplify for numerical stability and math simplicity
- ▶ sometimes referred to as cross-entropy

Neural networks

Training neural networks - output layer gradient

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

GRADIENT COMPUTATION

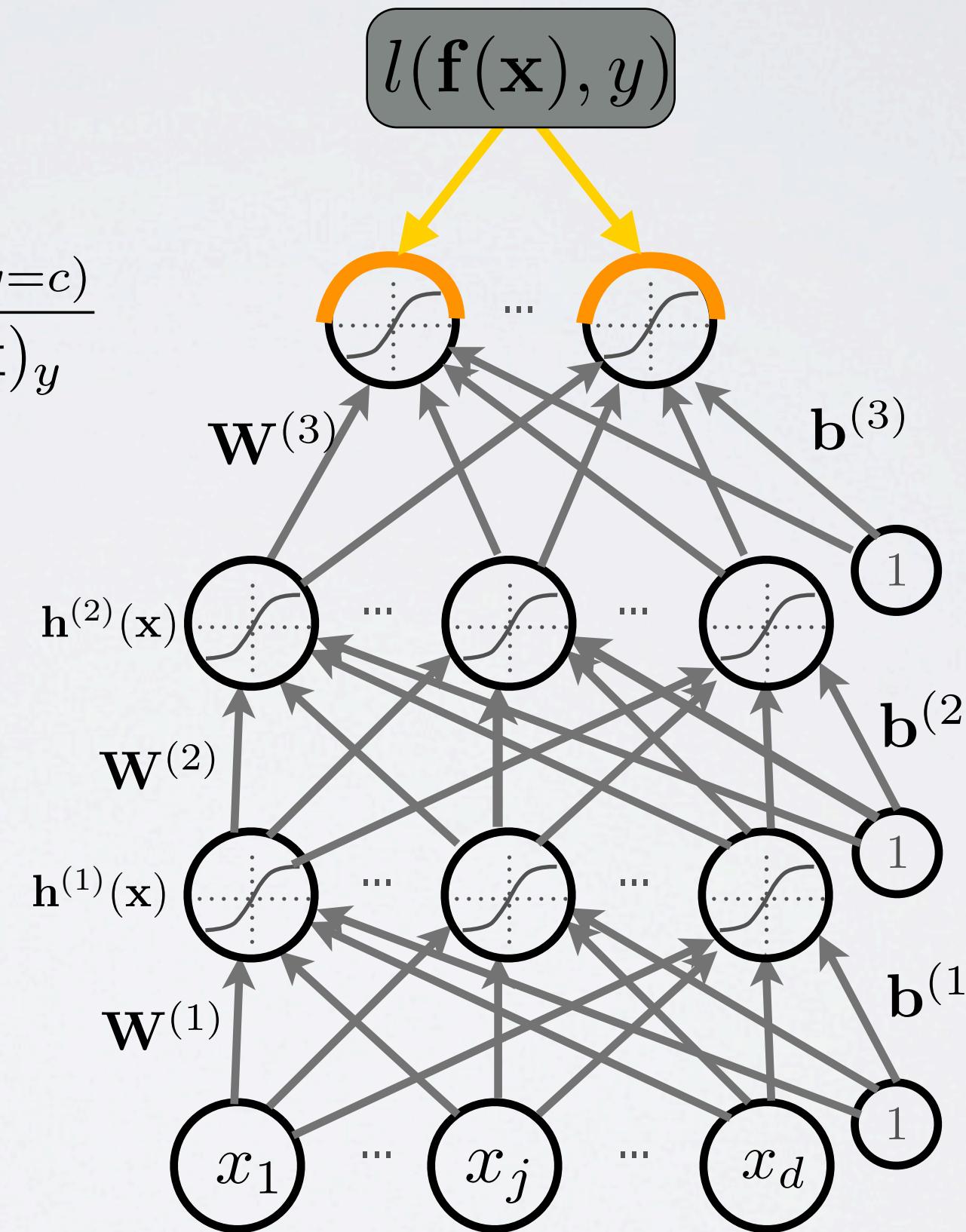
Topics: loss gradient at output

- Partial derivative:

$$\frac{\partial}{\partial f(\mathbf{x})_c} - \log f(\mathbf{x})_y = \frac{-1_{(y=c)}}{f(\mathbf{x})_y}$$

- Gradient:

$$\begin{aligned} \nabla_{f(\mathbf{x})} - \log f(\mathbf{x})_y \\ &= \frac{-1}{f(\mathbf{x})_y} \begin{bmatrix} 1_{(y=0)} \\ \vdots \\ 1_{(y=C-1)} \end{bmatrix} \\ &= \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y} \end{aligned}$$



GRADIENT COMPUTATION

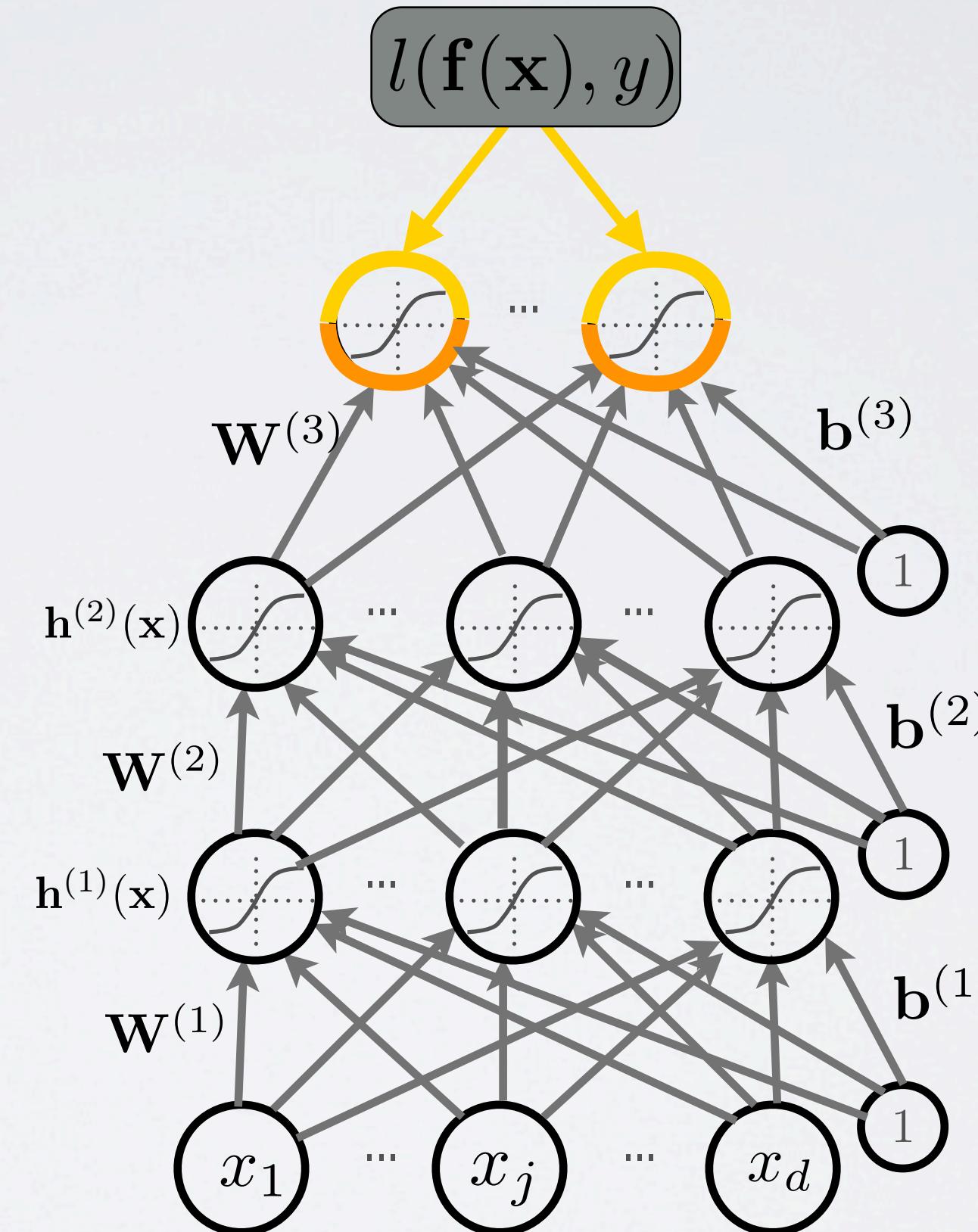
Topics: loss gradient at output
pre-activation

- Partial derivative:

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ &= - (1_{(y=c)} - f(\mathbf{x})_c) \end{aligned}$$

- Gradient:

$$\begin{aligned} & \nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ &= - (\mathbf{e}(y) - \mathbf{f}(\mathbf{x})) \end{aligned}$$



$$\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y$$

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \end{aligned}$$

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \end{aligned}$$

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})}
\end{aligned}$$

$$\boxed{
\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}
}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} f(\mathbf{x})_y - f(\mathbf{x})_y f(\mathbf{x})_c \right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right) \\
= & \frac{-1}{f(\mathbf{x})_y} (1_{(y=c)} f(\mathbf{x})_y - f(\mathbf{x})_y f(\mathbf{x})_c) \\
= & - (1_{(y=c)} - f(\mathbf{x})_c)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

Neural networks

Training neural networks - hidden layer gradient

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- for N iterations

$$\left. \begin{array}{l} \text{- for each training example } (\mathbf{x}^{(t)}, y^{(t)}) \\ \checkmark \Delta = -\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta}) \\ \checkmark \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta \end{array} \right\} \begin{array}{l} \text{training epoch} \\ = \\ \text{iteration over \textbf{all} examples} \end{array}$$

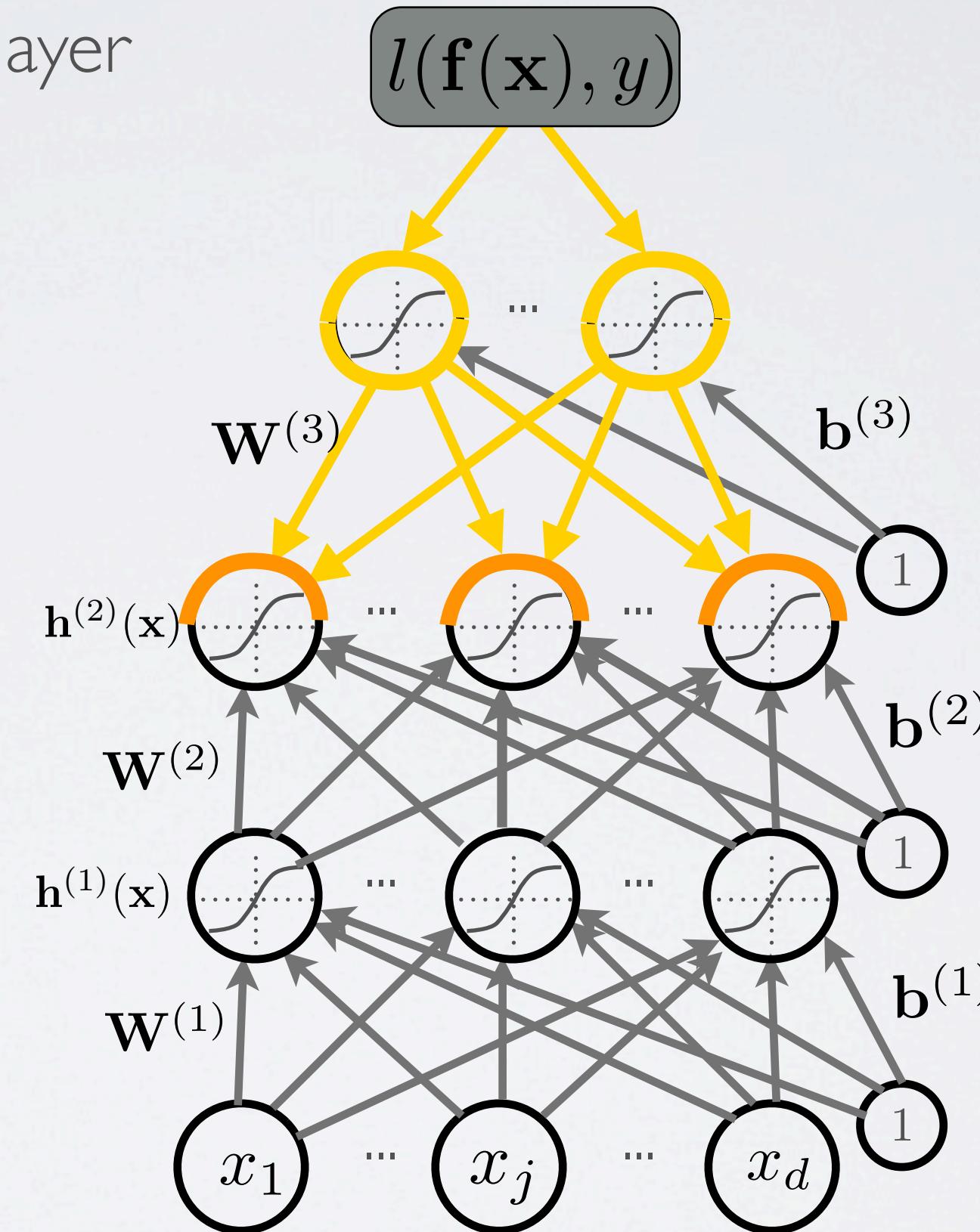
- To apply this algorithm to neural network training, we need

- the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- initialization method

GRADIENT COMPUTATION

Topics: loss gradient at hidden layer

- ... this is getting complicated!!



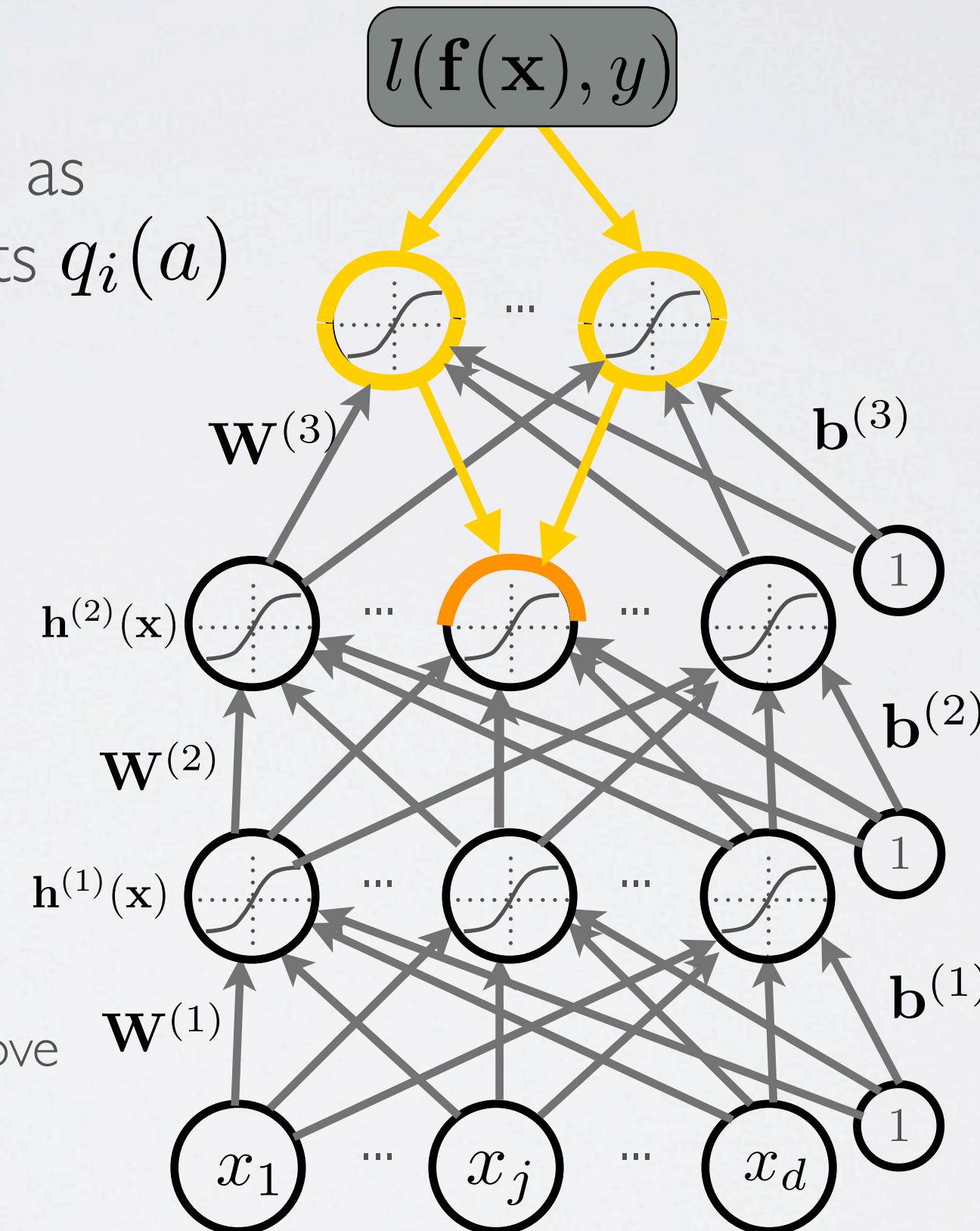
GRADIENT COMPUTATION

Topics: chain rule

- If a function $p(a)$ can be written as a function of intermediate results $q_i(a)$ then we have:

$$\frac{\partial p(a)}{\partial a} = \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a}$$

- We can invoke it by setting
 - a to a unit in layer
 - $q_i(a)$ to a pre-activation in the layer above
 - $p(a)$ is the loss function



GRADIENT COMPUTATION

Topics: loss gradient at hidden layers

- Partial derivative:

$$\frac{\partial}{\partial h^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y$$

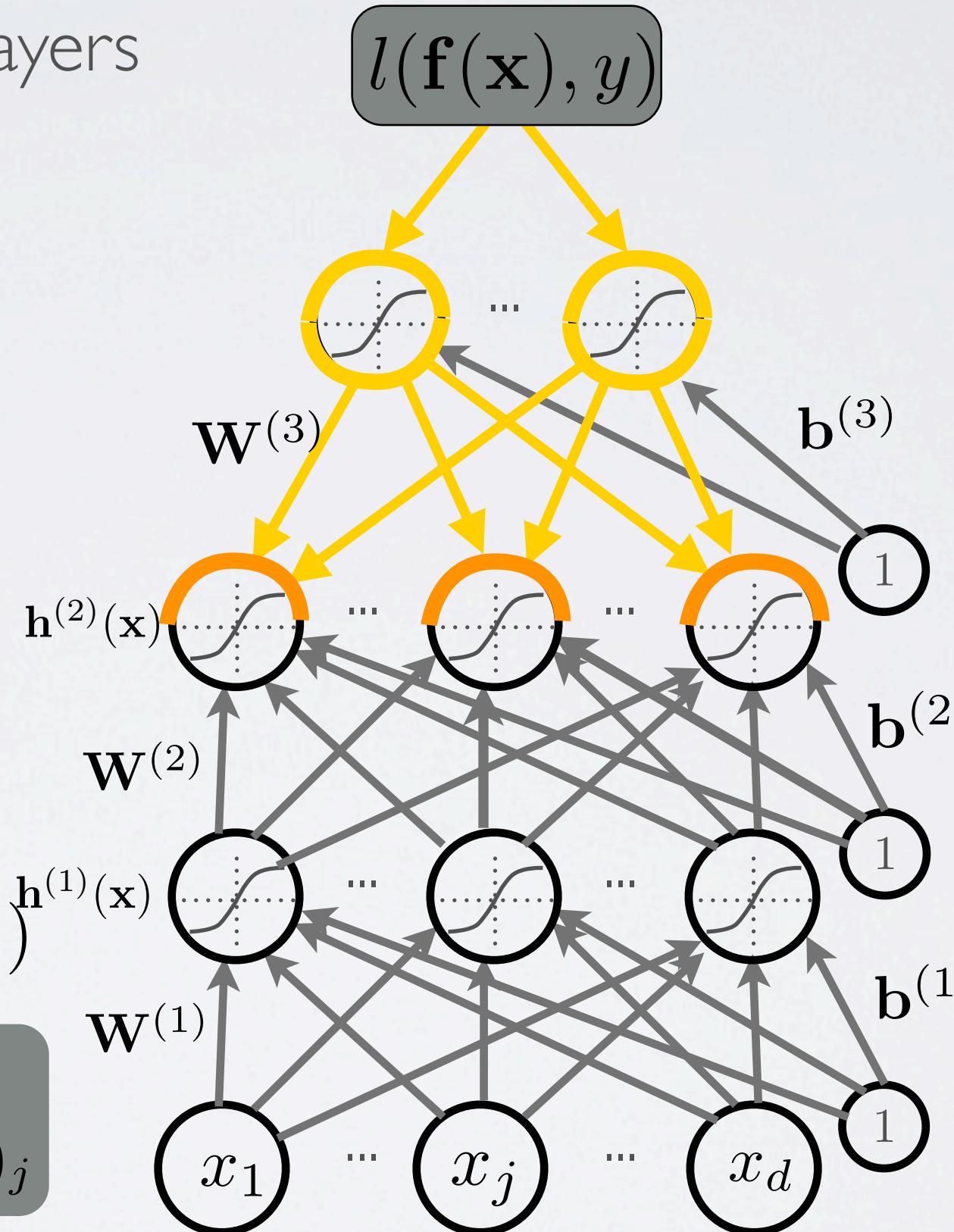
$$= \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} \frac{\partial a^{(k+1)}(\mathbf{x})_i}{\partial h^{(k)}(\mathbf{x})_j}$$

$$= \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} W_{i,j}^{(k+1)}$$

$$= (\mathbf{W}_{\cdot,j}^{k+1})^\top (\nabla_{\mathbf{a}^{k+1}(\mathbf{x})} - \log f(\mathbf{x})_y)$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

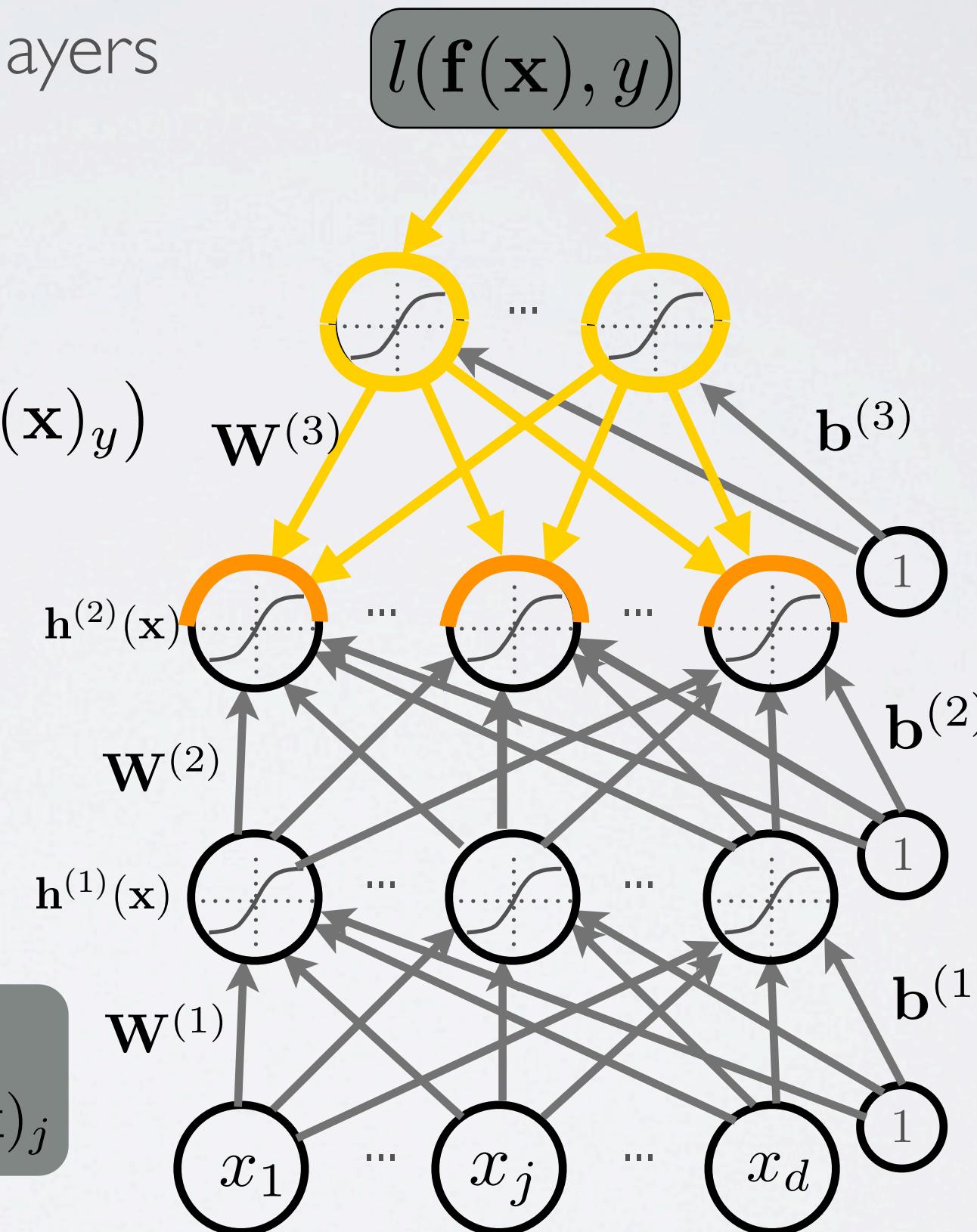


GRADIENT COMPUTATION

Topics: loss gradient at hidden layers

- Gradient:

$$\begin{aligned} & \nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = & \mathbf{W}^{(k+1)^\top} (\nabla_{\mathbf{a}^{(k+1)}(\mathbf{x})} - \log f(\mathbf{x})_y) \end{aligned}$$



REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

GRADIENT COMPUTATION

Topics: loss gradient at hidden layers
pre-activation

- Partial derivative:

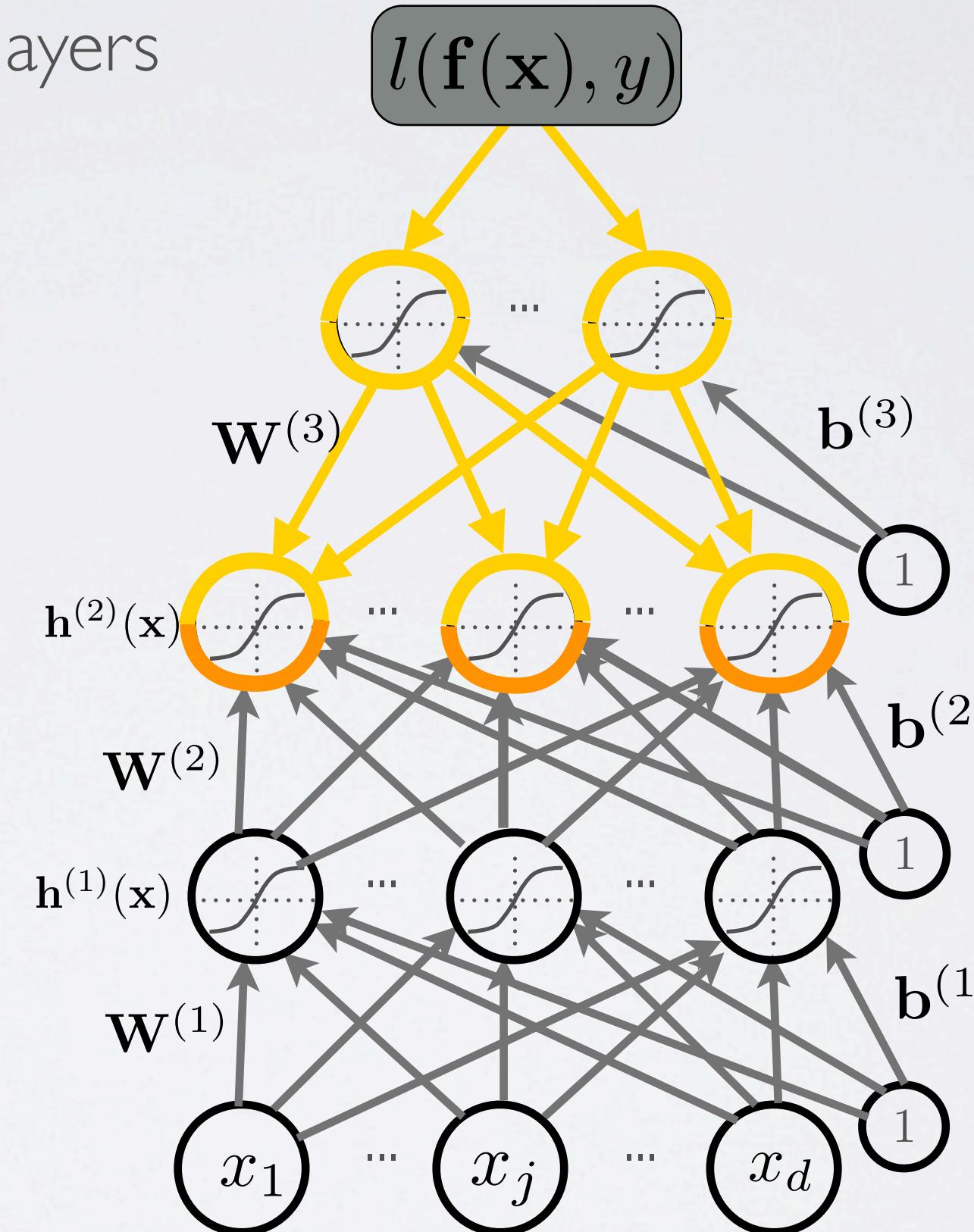
$$\frac{\partial}{\partial a^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y$$

$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} \frac{\partial h^{(k)}(\mathbf{x})_j}{\partial a^{(k)}(\mathbf{x})_j}$$

$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} g'(a^{(k)}(\mathbf{x})_j)$$

REMINDER

$$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$$



GRADIENT COMPUTATION

Topics: loss gradient at hidden layers
pre-activation

- Gradient:

$$\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

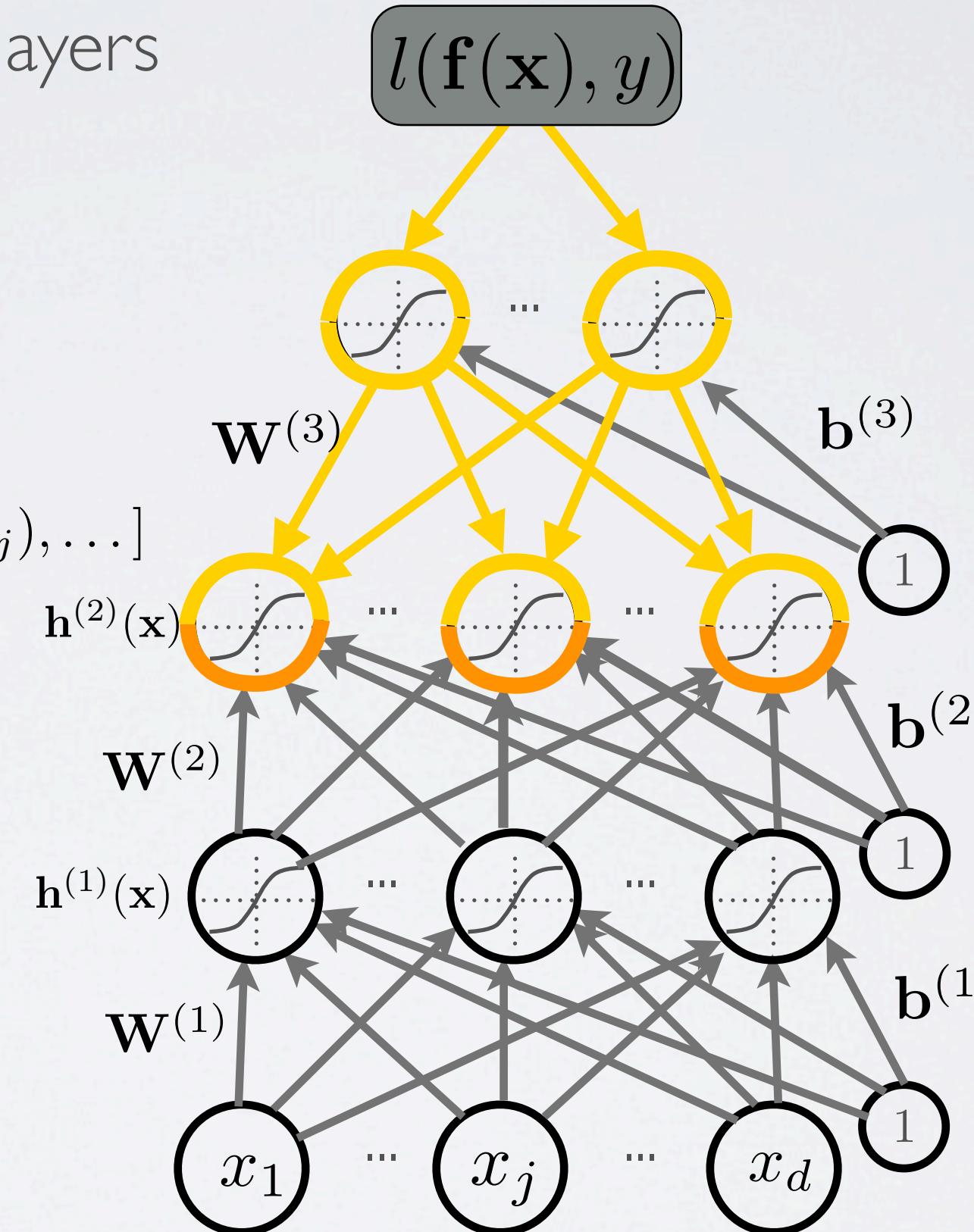
$$= (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y)^\top \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} \mathbf{h}^{(k)}(\mathbf{x})$$

$$= (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \odot [\dots, g'(a^{(k)}(\mathbf{x})_j), \dots]$$

↑
element-wise
product

REMINDER

$$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$$



Neural networks

Training neural networks - activation function derivative

GRADIENT COMPUTATION

Topics: loss gradient at hidden layers
pre-activation

- Gradient:

$$\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

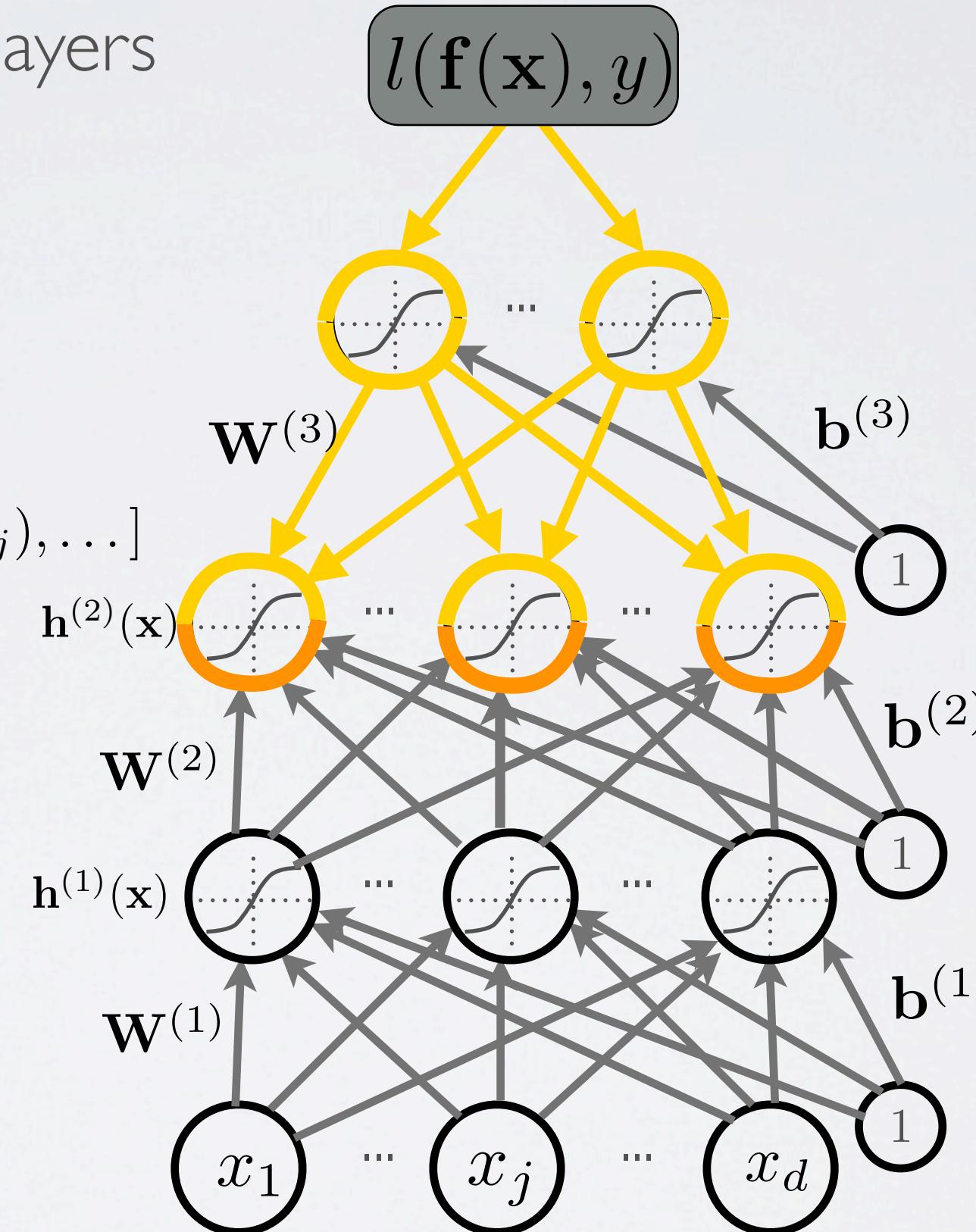
$$= (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y)^\top \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} \mathbf{h}^{(k)}(\mathbf{x})$$

$$= (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \odot [\dots, g'(a^{(k)}(\mathbf{x})_j), \dots]$$

↑
element-wise
product

REMINDER

$$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$$

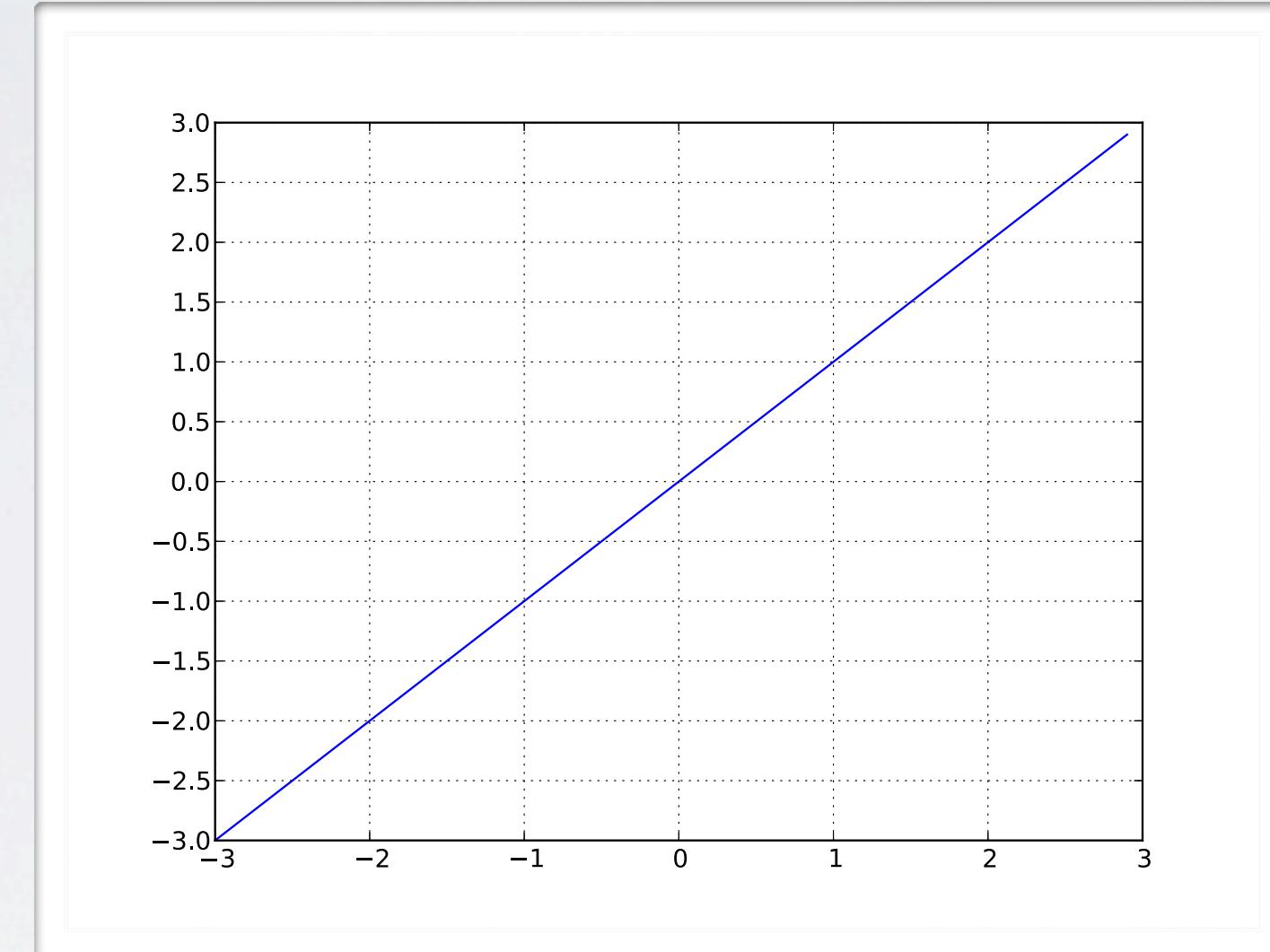


ACTIVATION FUNCTION

Topics: linear activation function gradient

- Partial derivative:

$$g'(a) = 1$$



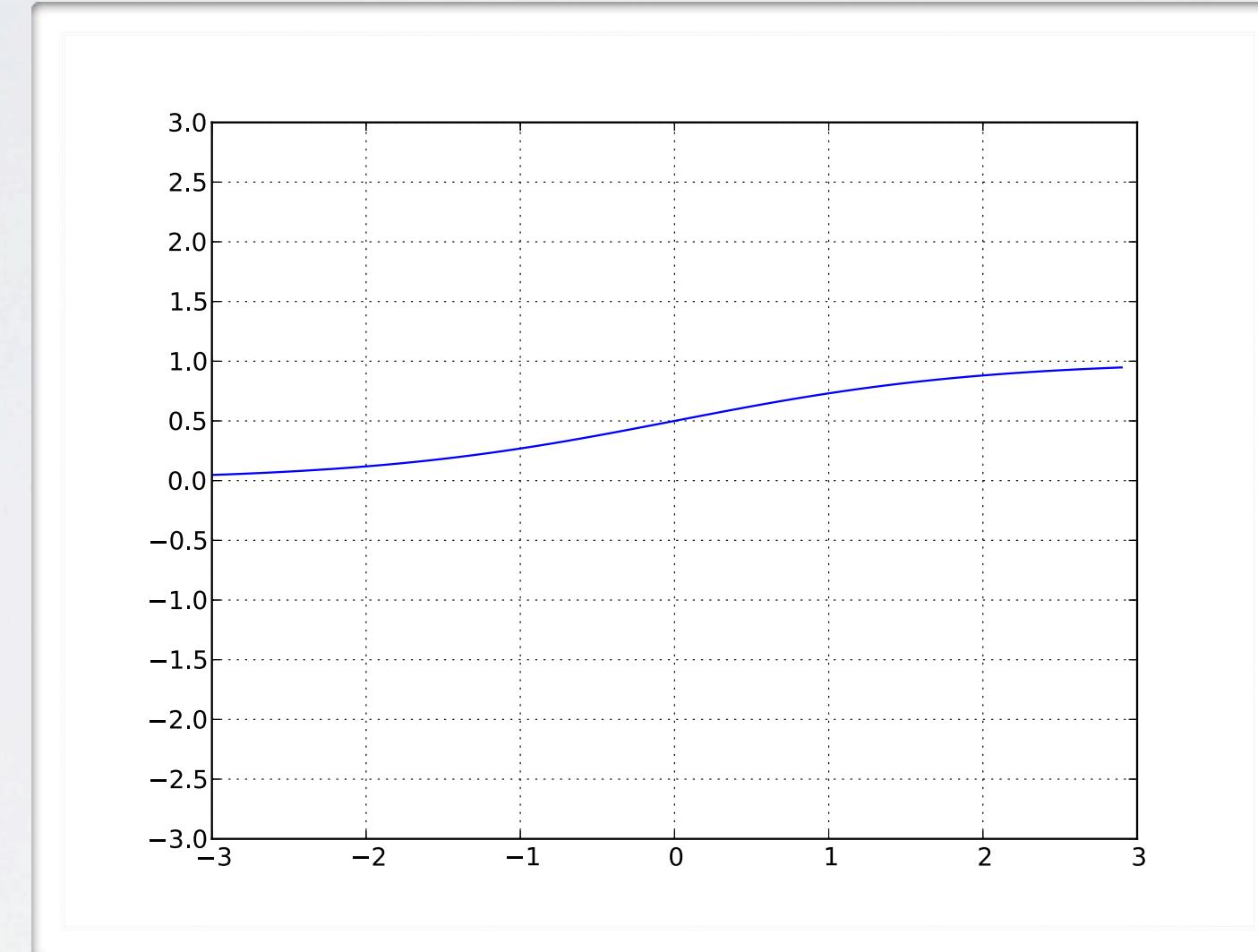
$$g(a) = a$$

ACTIVATION FUNCTION

Topics: sigmoid activation function gradient

- Partial derivative:

$$g'(a) = g(a)(1 - g(a))$$



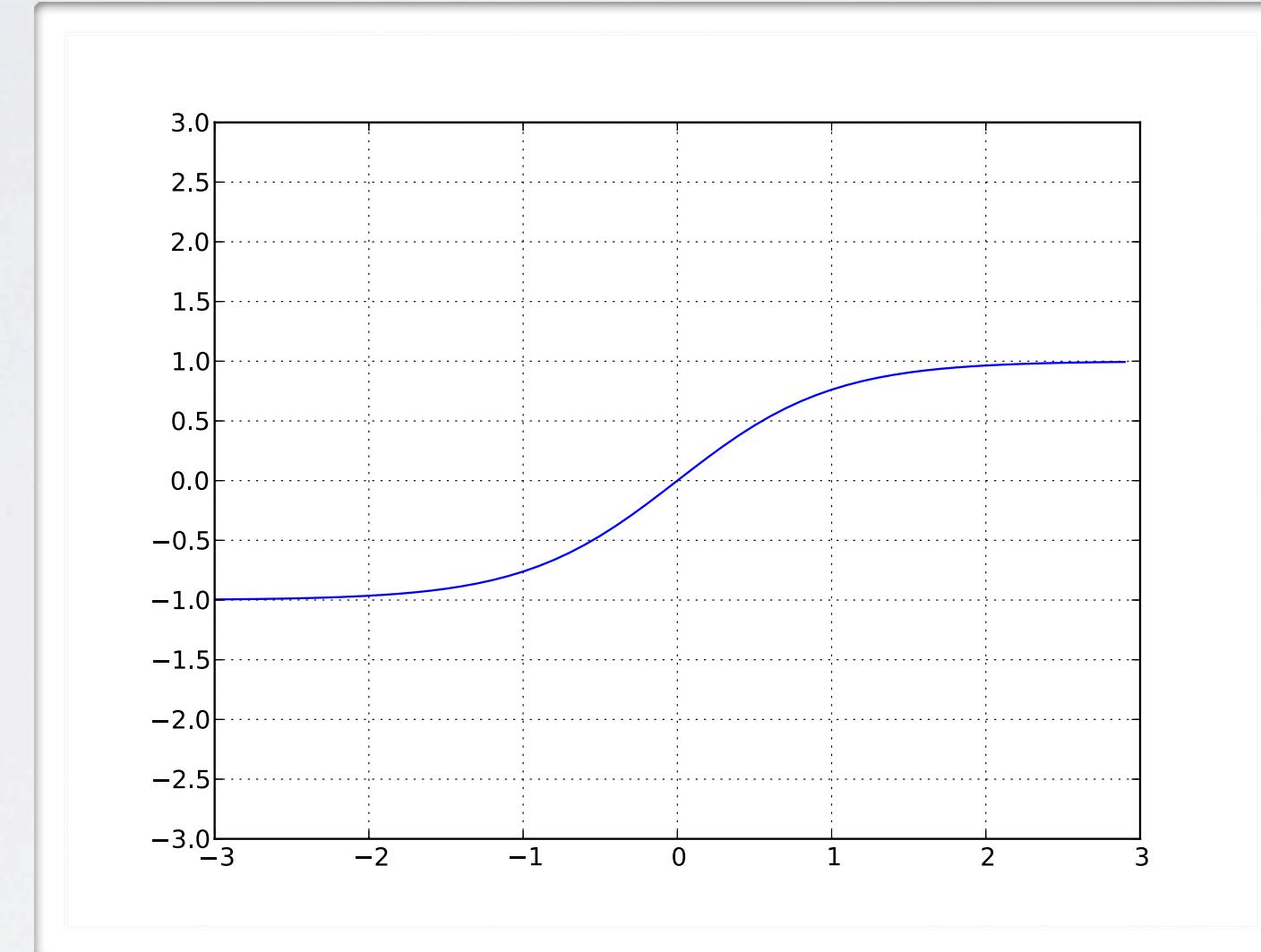
$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

ACTIVATION FUNCTION

Topics: tanh activation function gradient

- Partial derivative:

$$g'(a) = 1 - g(a)^2$$



$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

Neural networks

Training neural networks - parameter gradient

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

training epoch
 =
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

GRADIENT COMPUTATION

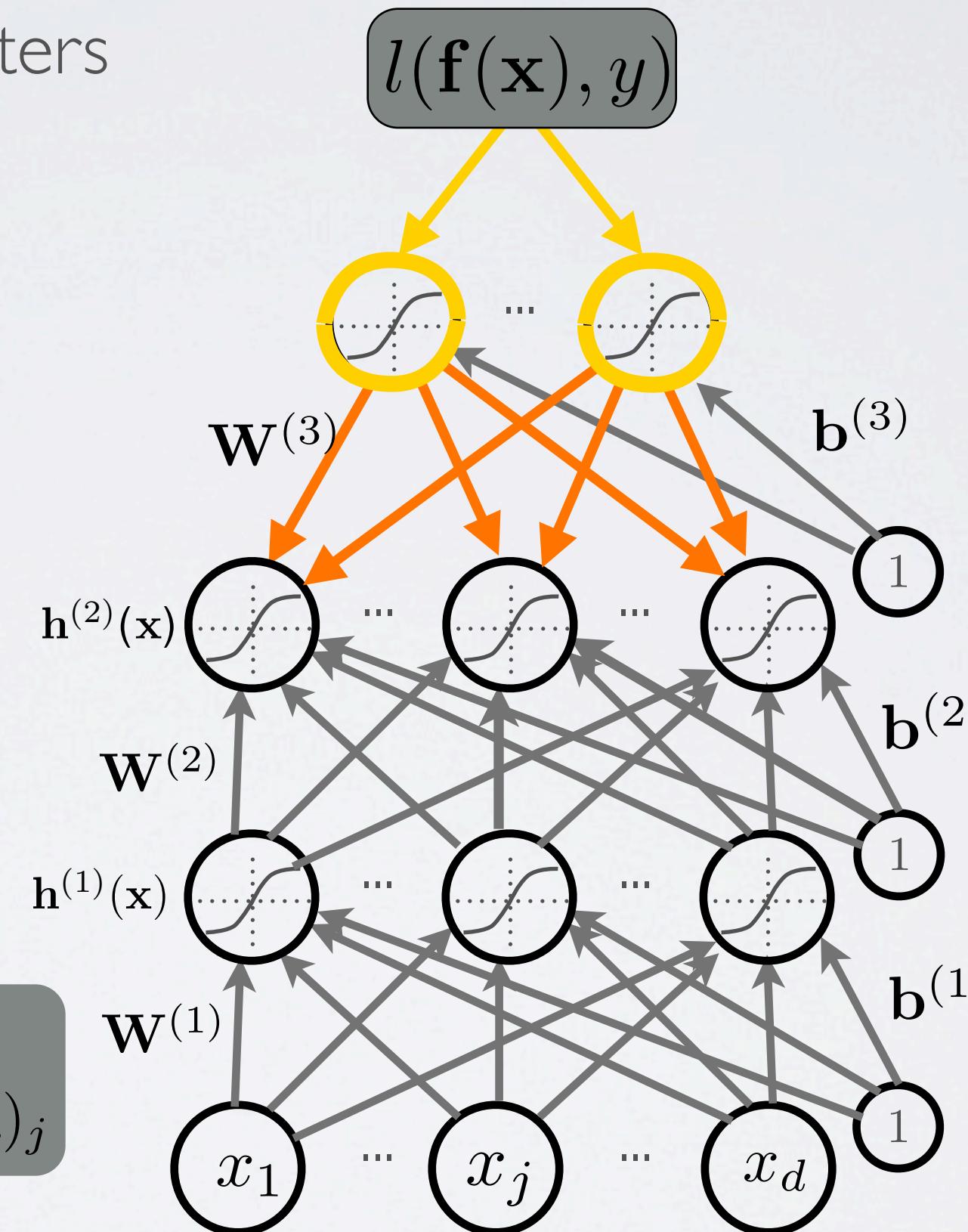
Topics: loss gradient of parameters

- Partial derivative (weights):

$$\begin{aligned} & \frac{\partial}{\partial W_{i,j}^{(k)}} - \log f(\mathbf{x})_y \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial W_{i,j}^{(k)}} \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} h_j^{(k-1)}(\mathbf{x}) \end{aligned}$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$



GRADIENT COMPUTATION

Topics: loss gradient of parameters

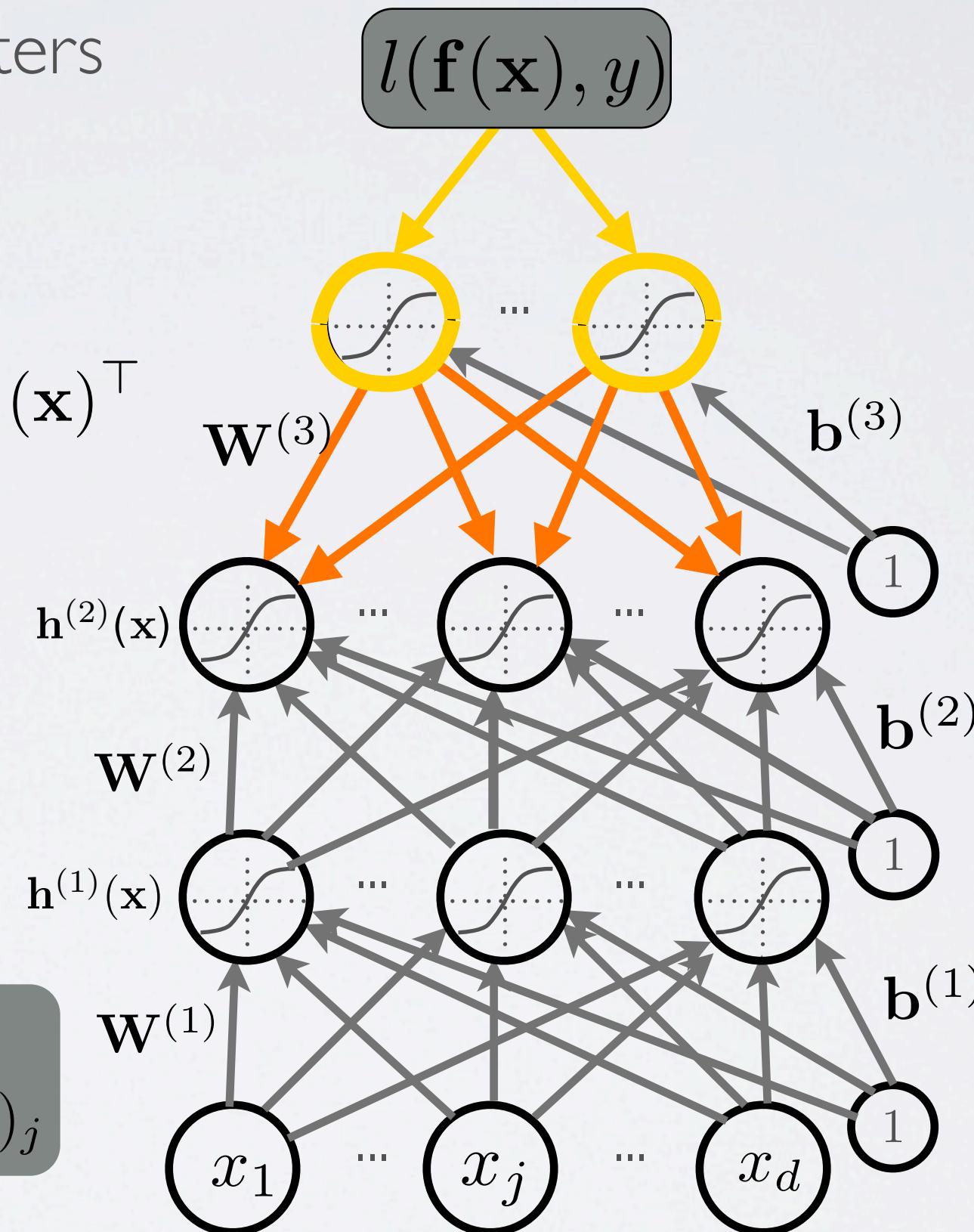
- Gradient (weights):

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y$$

$$= (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$



GRADIENT COMPUTATION

Topics: loss gradient of parameters

- Partial derivative (biases):

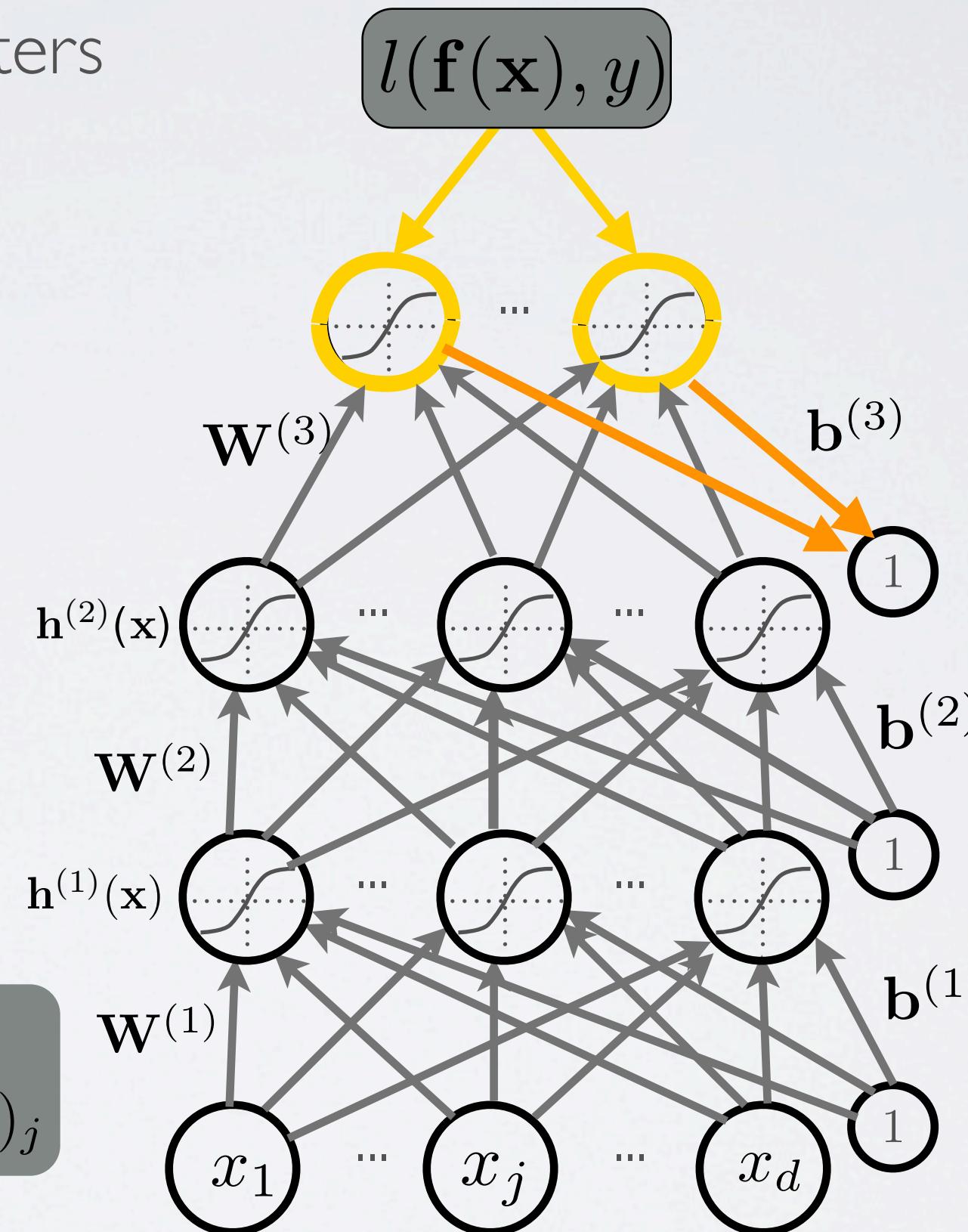
$$\frac{\partial}{\partial b_i^{(k)}} - \log f(\mathbf{x})_y$$

$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial b_i^{(k)}}$$

$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i}$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$



GRADIENT COMPUTATION

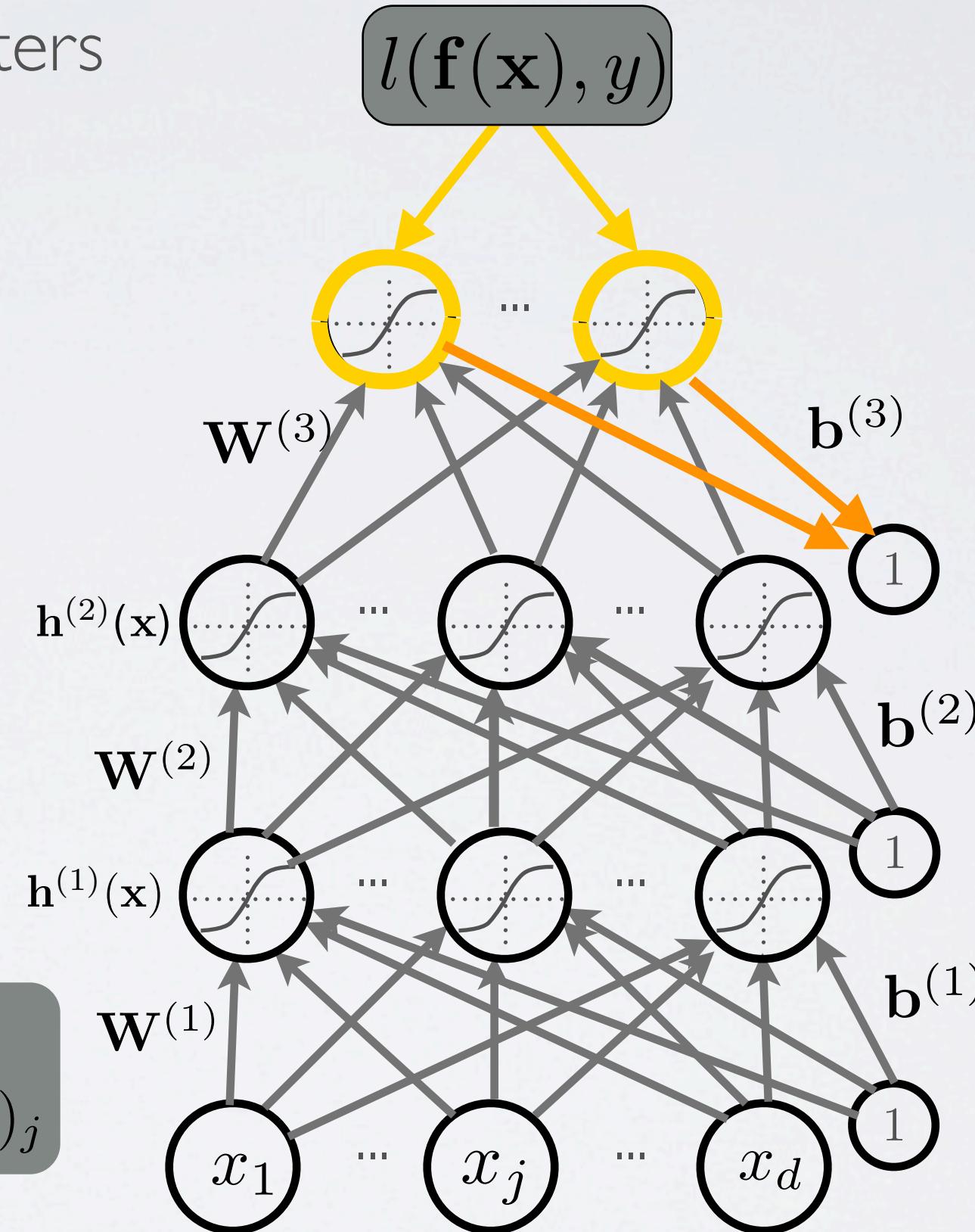
Topics: loss gradient of parameters

- Gradient (biases):

$$\begin{aligned} \nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \\ = \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y \end{aligned}$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$



Neural networks

Training neural networks - backpropagation algorithm

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

BACKPROPAGATION

Topics: backpropagation algorithm

- This assumes a forward propagation has been made before

- ▶ compute output gradient (before activation)

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

- ▶ for k from $L+1$ to 1

- compute gradients of hidden layer parameter

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \iff (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \iff \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

- compute gradient of hidden layer below

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \mathbf{W}^{(k)^\top} (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y)$$

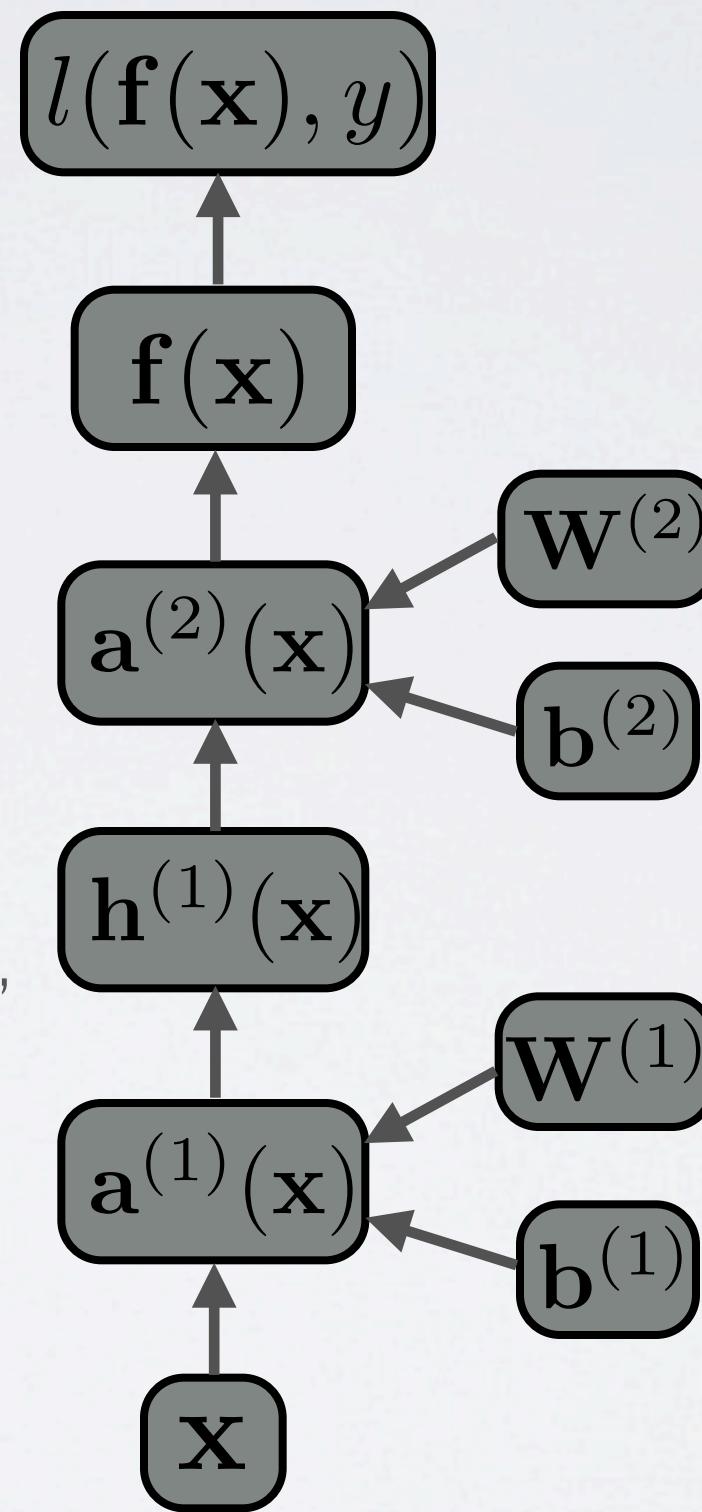
- compute gradient of hidden layer below (before activation)

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff (\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y) \odot [\dots, g'(a^{(k-1)}(\mathbf{x})_j), \dots]$$

FLOW GRAPH

Topics: flow graph

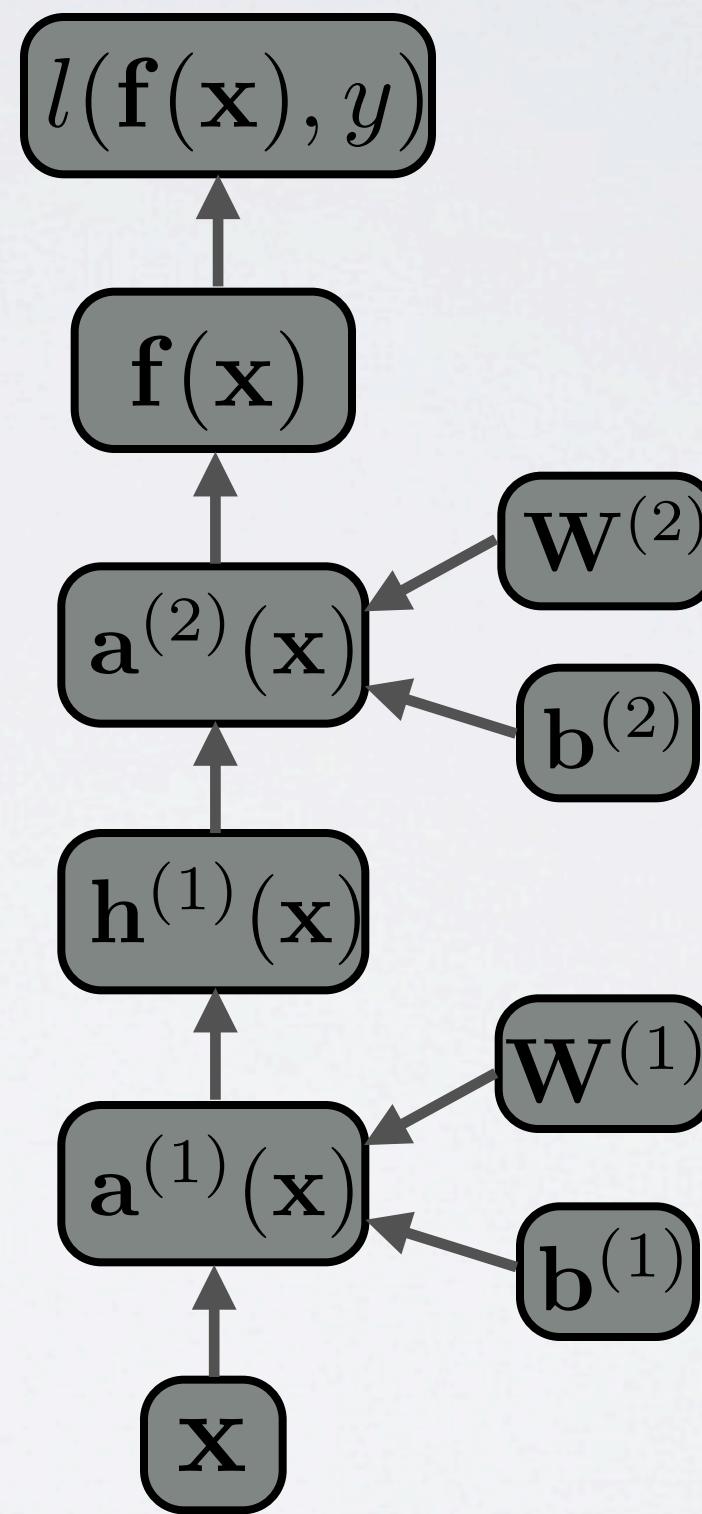
- Forward propagation can be represented as an acyclic flow graph
- It's a nice way of implementing forward propagation in a modular way
 - ▶ each box could be an object with an fprop method, that computes the value of the box given its children
 - ▶ calling the fprop method of each box in the right order yield forward propagation



FLOW GRAPH

Topics: automatic differentiation

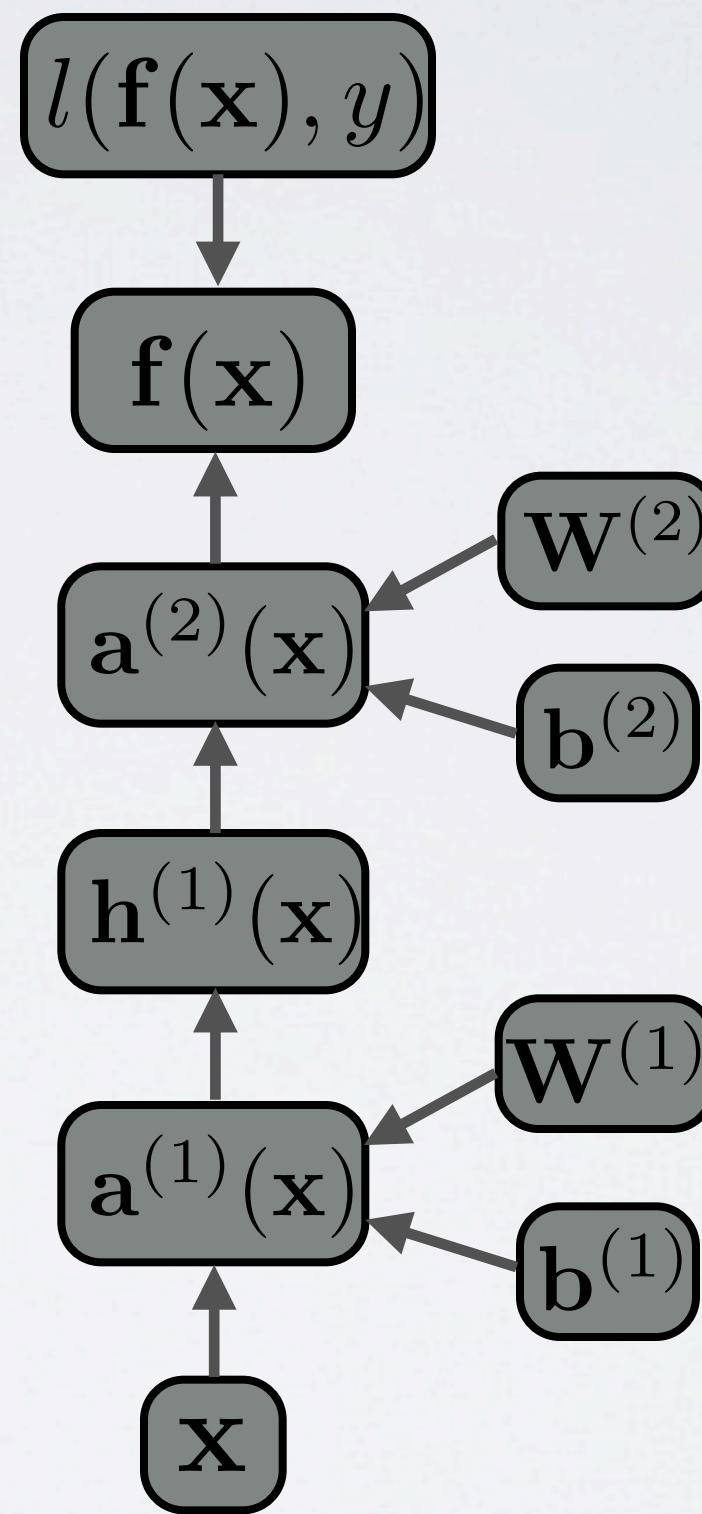
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

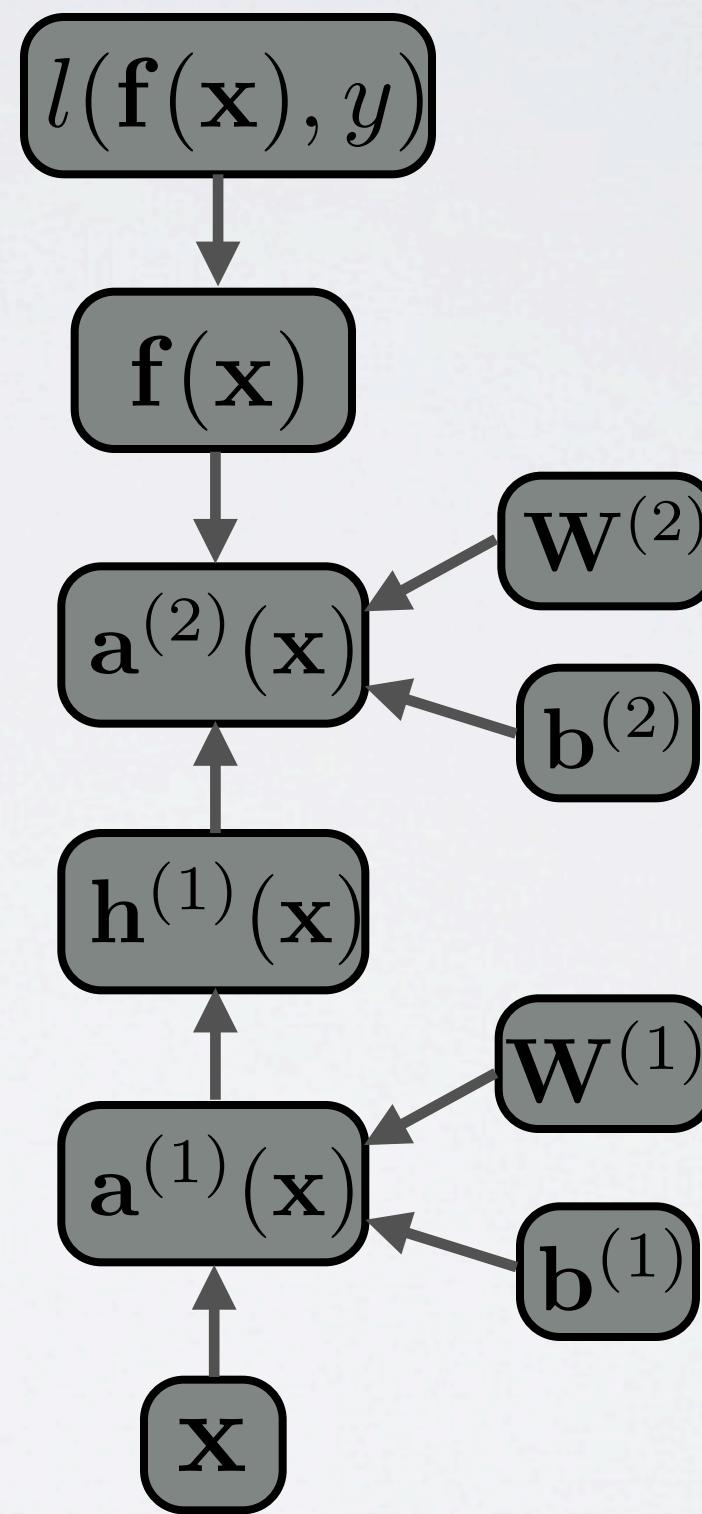
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

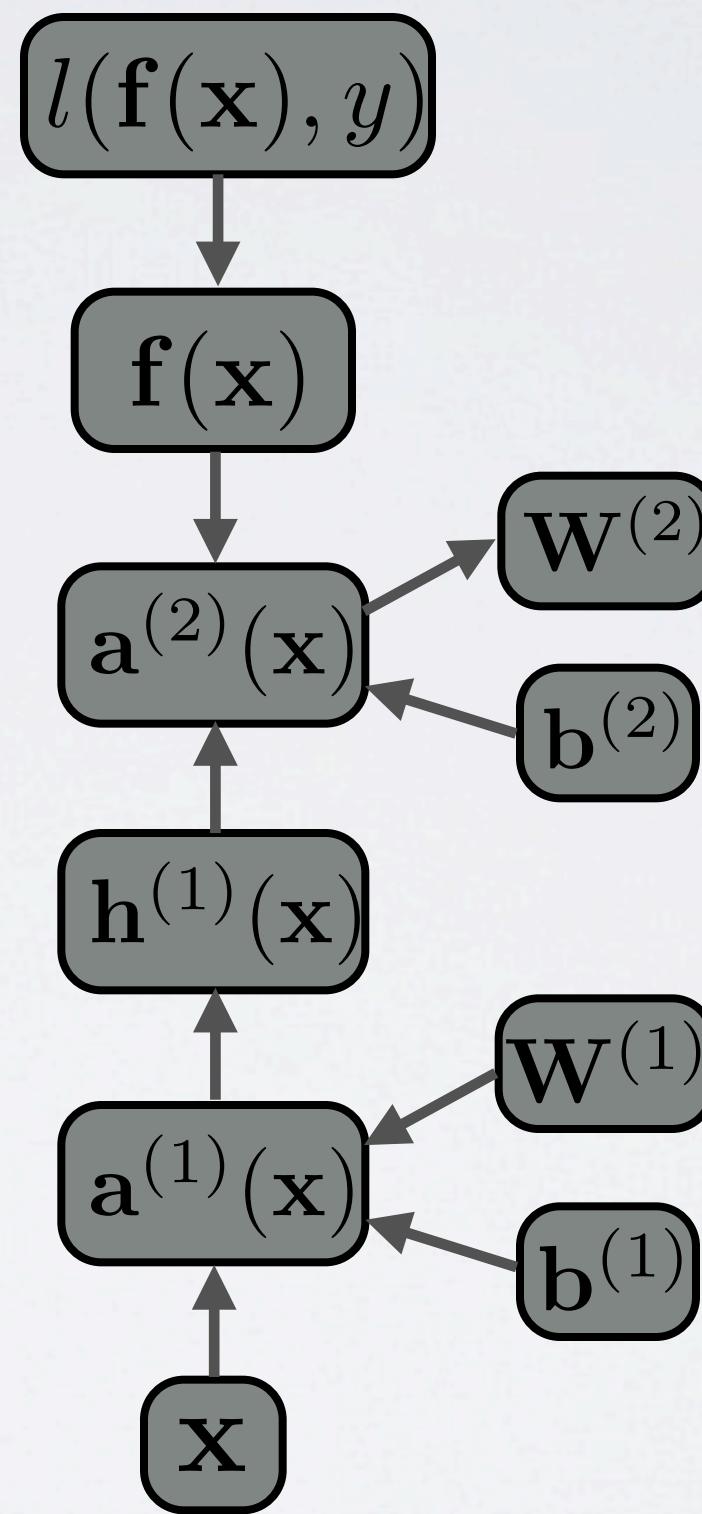
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

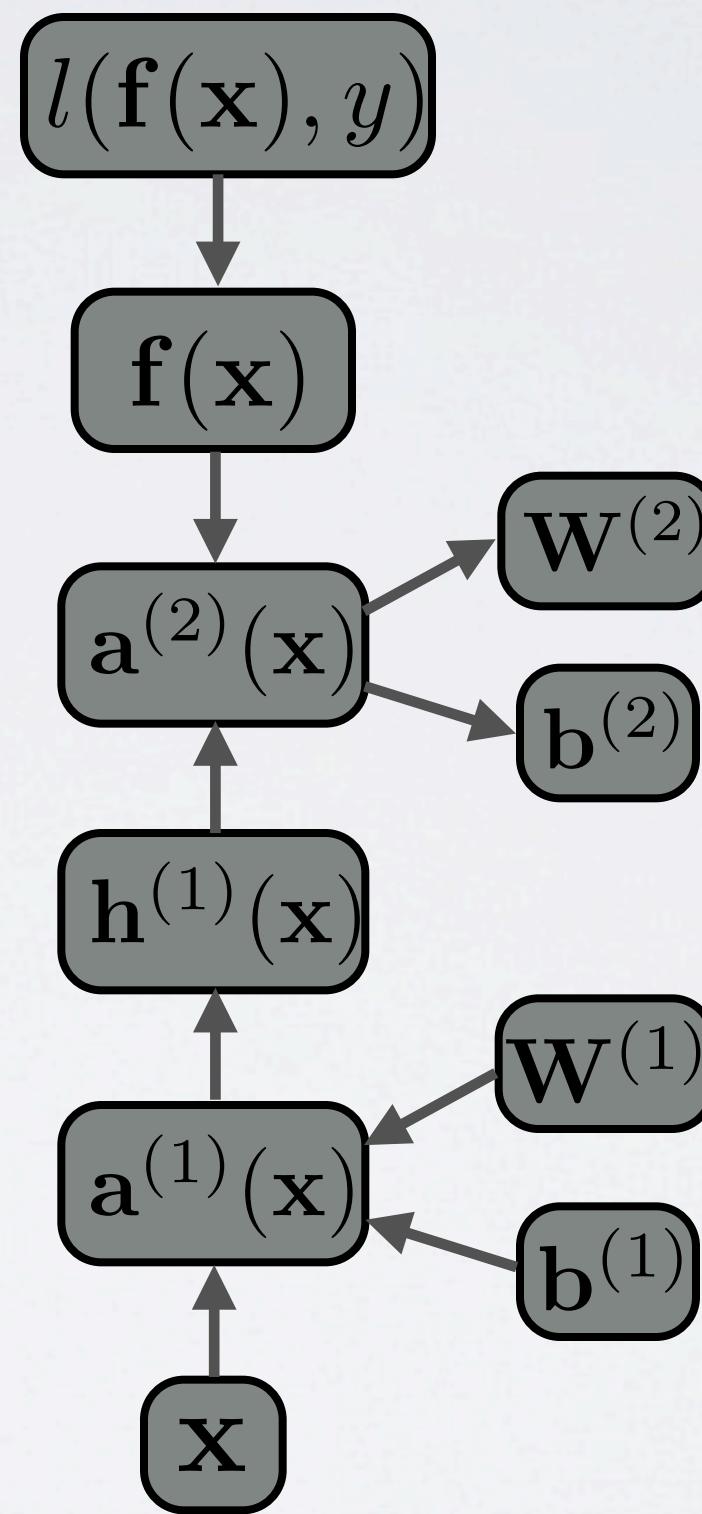
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

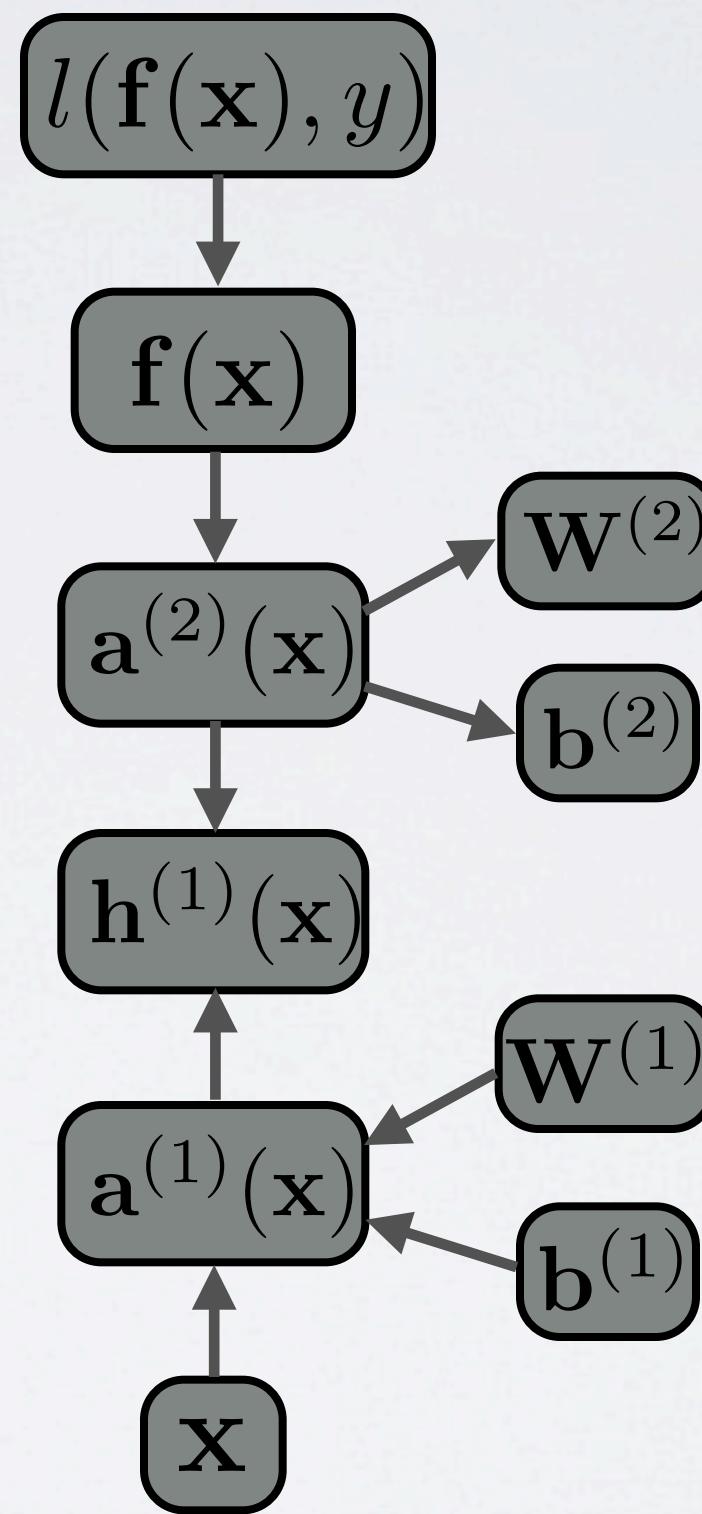
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

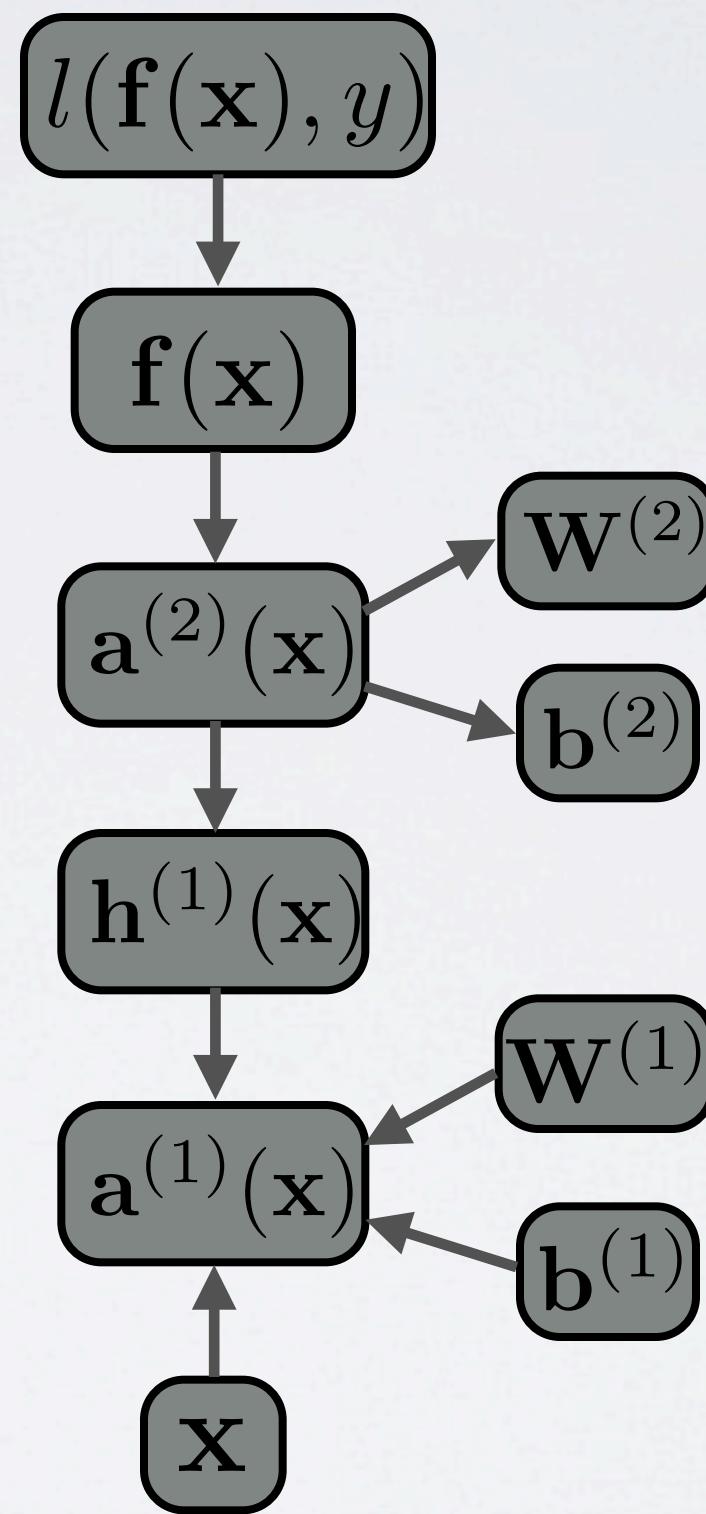
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

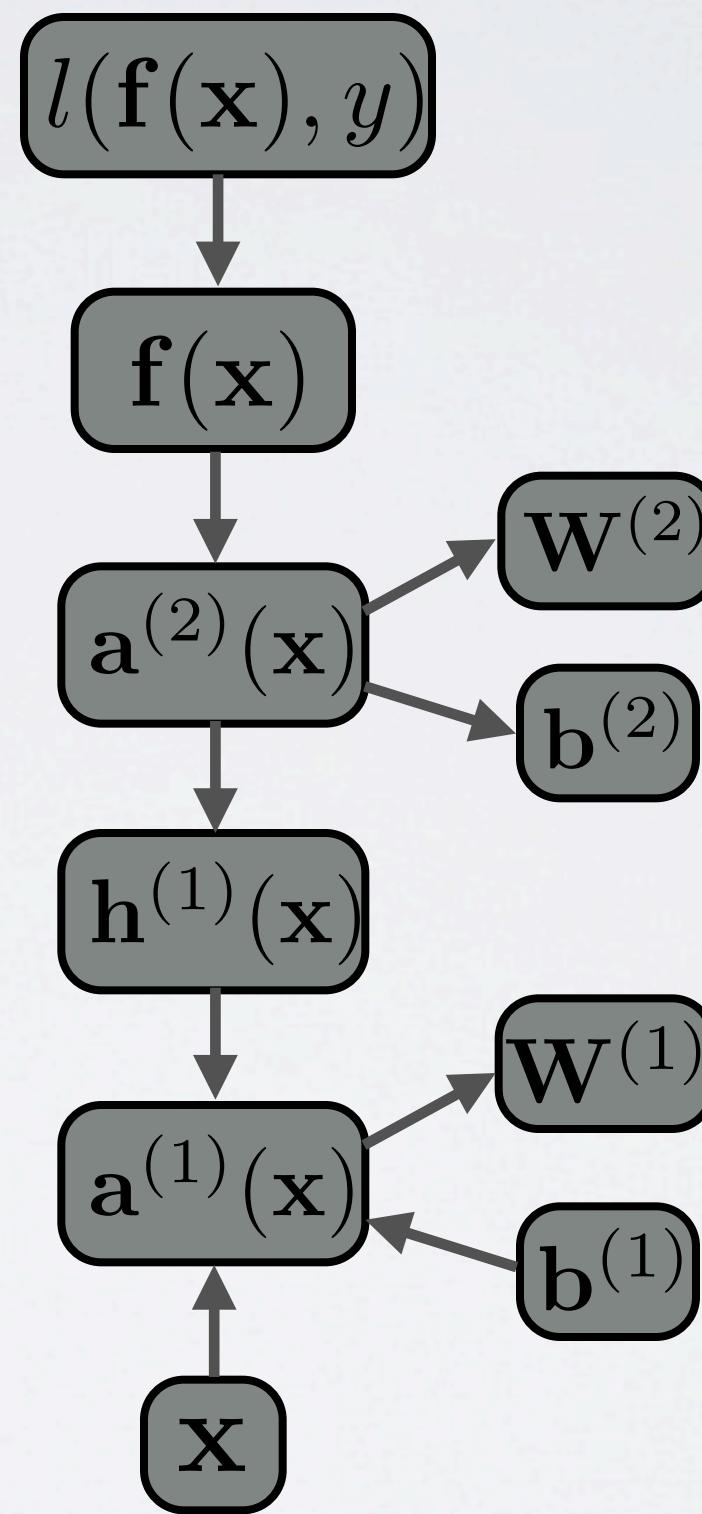
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

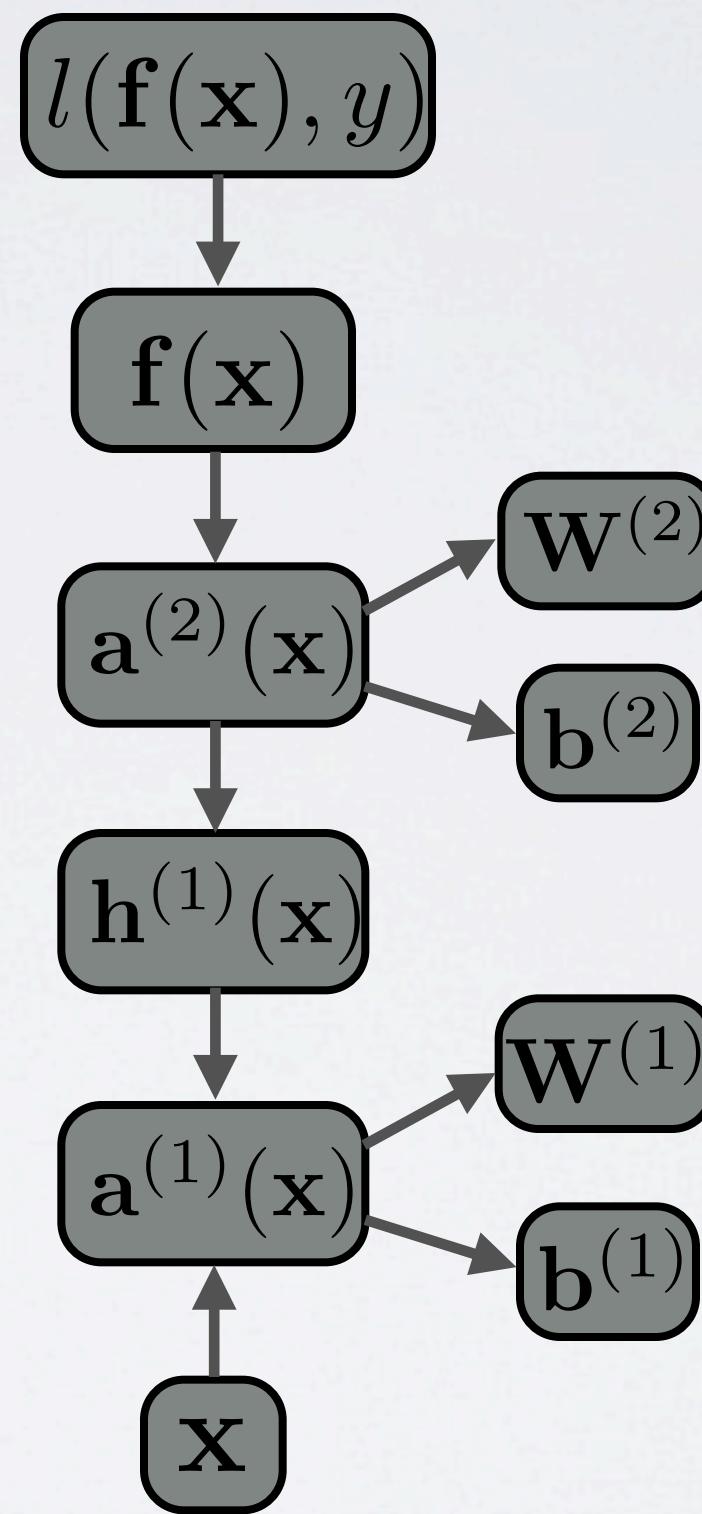
- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



FLOW GRAPH

Topics: automatic differentiation

- Each object also has a bprop method
 - ▶ it computes the gradient of the loss with respect to each children
 - ▶ fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
 - ▶ only need to reach the parameters



GRADIENT CHECKING

Topics: finite difference approximation

- To debug your implementation of fprop/bprop, you can compare with a finite-difference approximation of the gradient

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

- $f(x)$ would be the loss
- x would be a parameter
- $f(x + \epsilon)$ would be the loss if you add ϵ to the parameter
- $f(x - \epsilon)$ would be the loss if you subtract ϵ to the parameter

Neural networks

Training neural networks - regularization

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch
 iteration over **all** examples

- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

REGULARIZATION

Topics: L2 regularization

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)} \right)^2 = \sum_k \|\mathbf{W}^{(k)}\|_F^2$$

- Gradient: $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\mathbf{W}^{(k)}$
- Only applied on weights, not on biases (weight decay)
- Can be interpreted as having a Gaussian prior over the weights

REGULARIZATION

Topics: L1 regularization

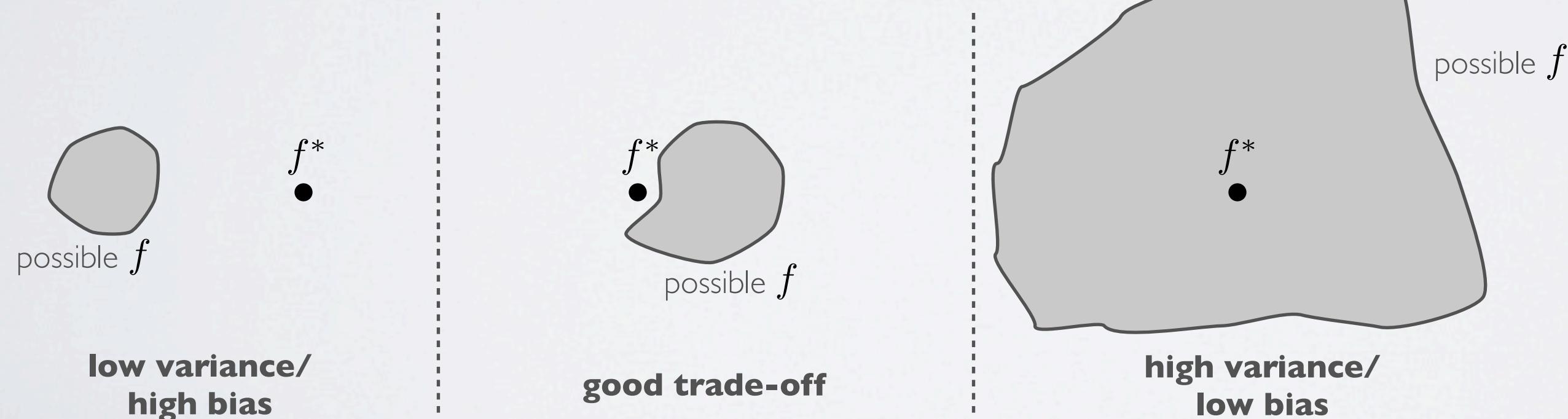
$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

- Gradient: $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = \text{sign}(\mathbf{W}^{(k)})$
 - ▶ where $\text{sign}(\mathbf{W}^{(k)})_{i,j} = 1_{\mathbf{W}_{i,j}^{(k)} > 0} - 1_{\mathbf{W}_{i,j}^{(k)} < 0}$
- Also only applied on weights
- Unlike L2, L1 will push certain weights to be exactly 0
- Can be interpreted as having a Laplacian prior over the weights

MACHINE LEARNING

Topics: bias-variance trade-off

- Variance of trained model: does it vary a lot if the training set changes
- Bias of trained model: is the average model close to the true solution
- Generalization error can be seen as the sum of the (squared) bias and the variance



Neural networks

Training neural networks - parameter initialization

MACHINE LEARNING

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- ▶ initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
- ▶ for N iterations
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
 - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
 - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch
 iteration over **all** examples

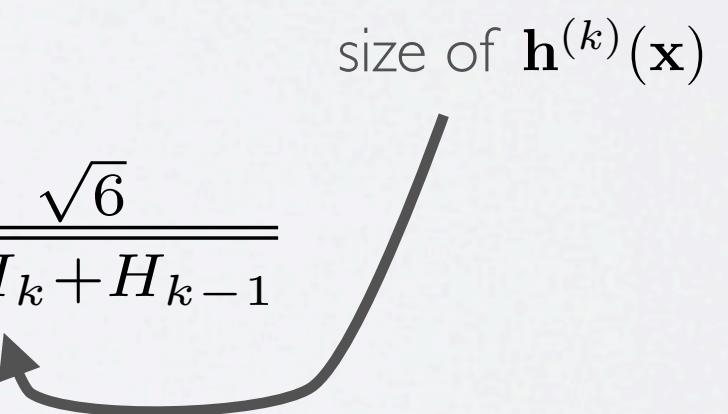
- To apply this algorithm to neural network training, we need

- ▶ the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- ▶ the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
- ▶ initialization method

INITIALIZATION

Topics: initialization

- For biases
 - ▶ initialize all to 0
- For weights
 - ▶ Can't initialize weights to 0 with tanh activation
 - we can show that all gradients would then be 0 (saddle point)
 - ▶ Can't initialize all weights to the same value
 - we can show that all hidden units in a layer will always behave the same
 - need to break symmetry
 - ▶ Recipe: sample $\mathbf{W}_{i,j}^{(k)}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
 - the idea is to sample around 0 but break symmetry
 - other values of b could work well (not an exact science) (see Glorot & Bengio, 2010)



Neural networks

Training neural networks - model selection

MACHINE LEARNING

Topics: training, validation and test sets, generalization

- Training set $\mathcal{D}^{\text{train}}$ serves to train a model
- Validation set $\mathcal{D}^{\text{valid}}$ serves to select hyper-parameters
- Test set $\mathcal{D}^{\text{test}}$ serves to estimate the generalization performance (error)
- Generalization is the behavior of the model on **unseen examples**
 - ▶ this is what we care about in machine learning!

MODEL SELECTION

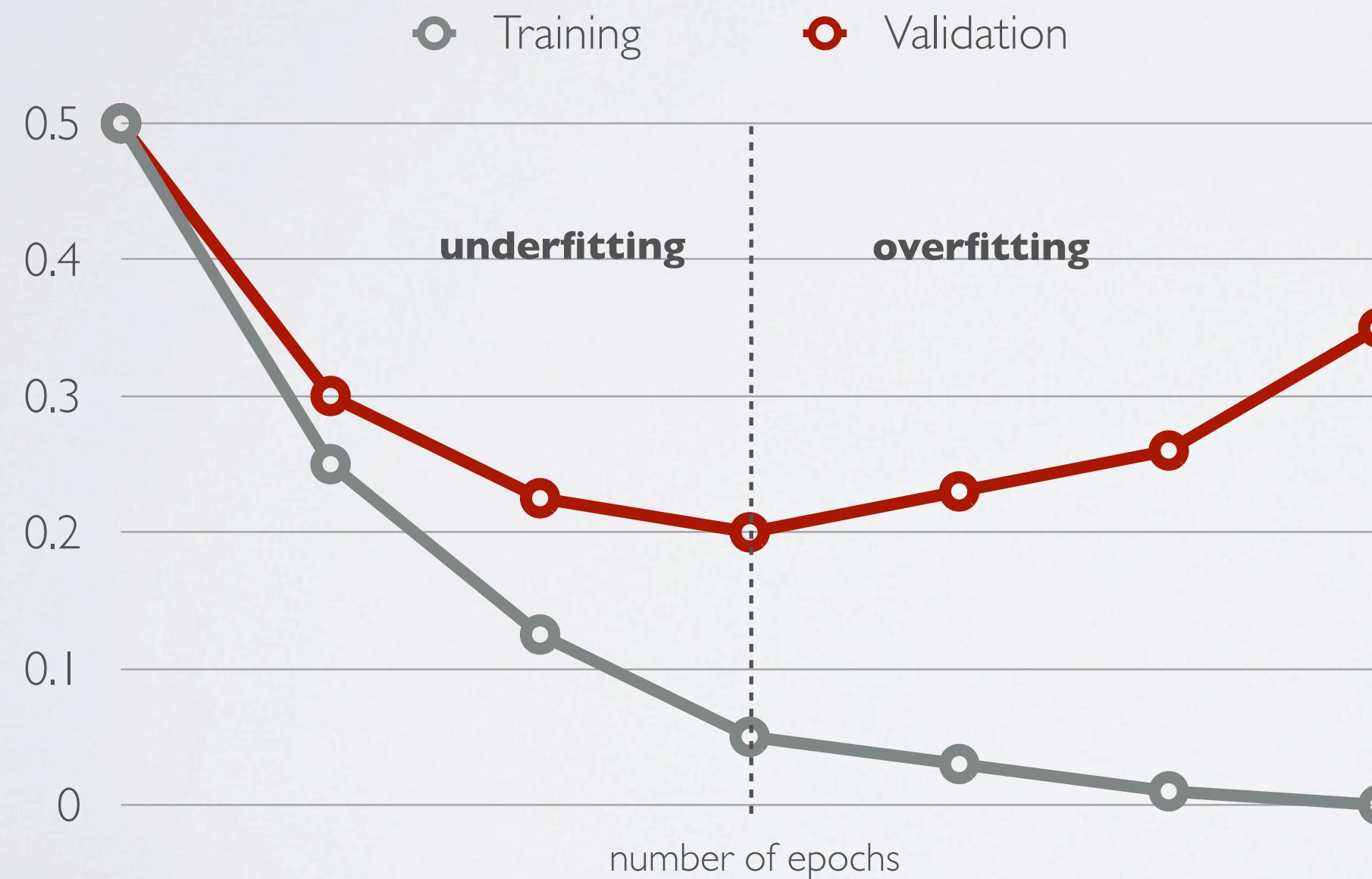
Topics: grid search

- To search for the best configuration of the hyper-parameters:
 - ▶ you can perform a grid search
 - specify a set of values you want to test for each hyper-parameter
 - try all possible configurations of these values
 - ▶ you can perform a random search
 - specify a distribution over the values of each hyper-parameters (e.g. uniform in some range)
 - sample independently each hyper-parameter to get a configuration, and repeat as many times as wanted
- Use a validation set performance to select the best configuration
- You can go back and refine the grid/distributions if needed

KNOWING WHEN TO STOP

Topics: early stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead)



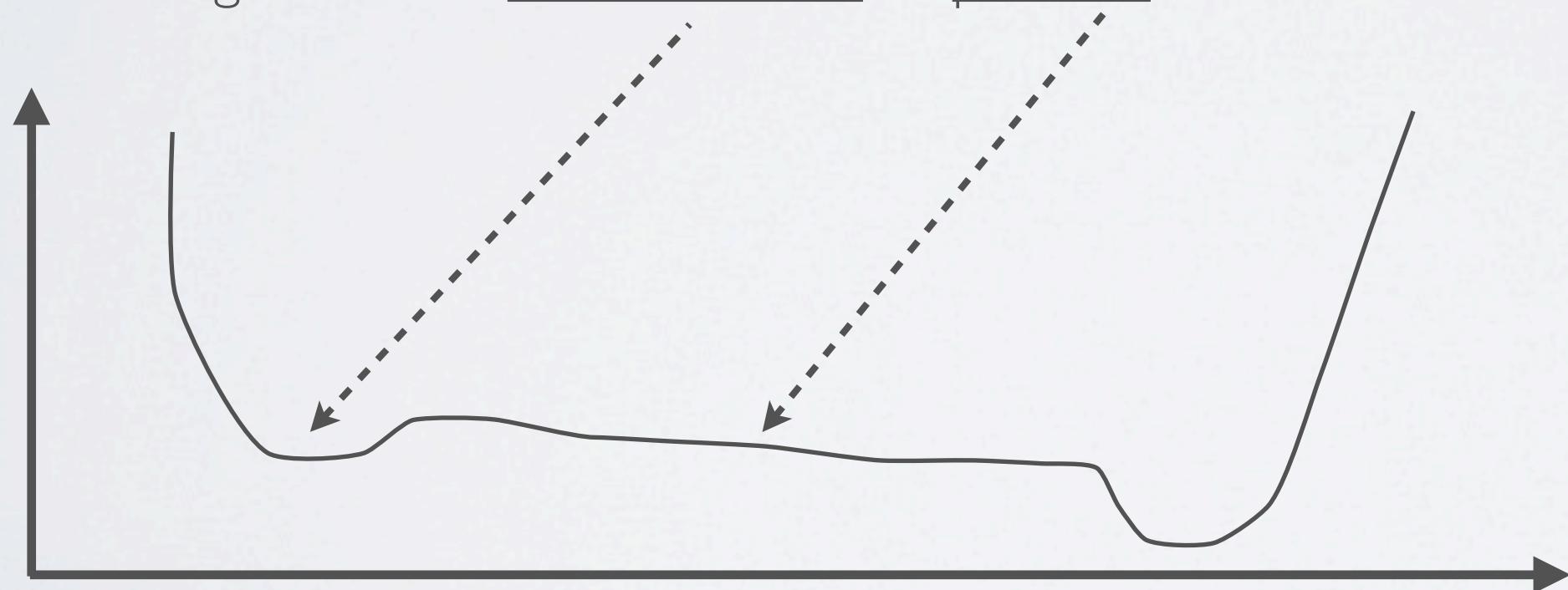
Neural networks

Training neural networks - optimization

OPTIMIZATION

Topics: local optimum, global optimum, plateau

- Notes on the optimization problem
 - ▶ there isn't a single global optimum (non-convex optimization)
 - we can permute the hidden units (with their connections) and get the same function
 - we say that the hidden unit parameters are not identifiable
 - ▶ Optimization can get stuck in local minimum or plateaus



OPTIMIZATION

Topics: local optimum, global optimum, plateau

Neural network training demo
(by Andrej Karpathy)

<http://cs.stanford.edu/~karpathy/svmjs/demo/demonn.html>

GRADIENT DESCENT

Topics: convergence conditions, decrease constant

- Stochastic gradient descent will converge if

- $\sum_{t=1}^{\infty} \alpha_t = \infty$

- $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

where α_t is the learning rate of the t^{th} update

- Decreasing strategies: (δ is the decrease constant)

- $\alpha_t = \frac{\alpha}{1+\delta t}$

- $\alpha_t = \frac{\alpha}{t^\delta}$ (où $0.5 < \delta \leq 1$)

- Better to use a fixed learning rate for the first few updates

GRADIENT DESCENT

Topics: mini-batch, momentum

- Can update based on a mini-batch of example (instead of 1 example):
 - ▶ the gradient is the average regularized loss for that mini-batch
 - ▶ can give a more accurate estimate of the risk gradient
 - ▶ can leverage matrix/matrix operations, which are more efficient
- Can use an exponential average of previous gradients:

$$\bar{\nabla}_{\theta}^{(t)} = \nabla_{\theta} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \bar{\nabla}_{\theta}^{(t-1)}$$

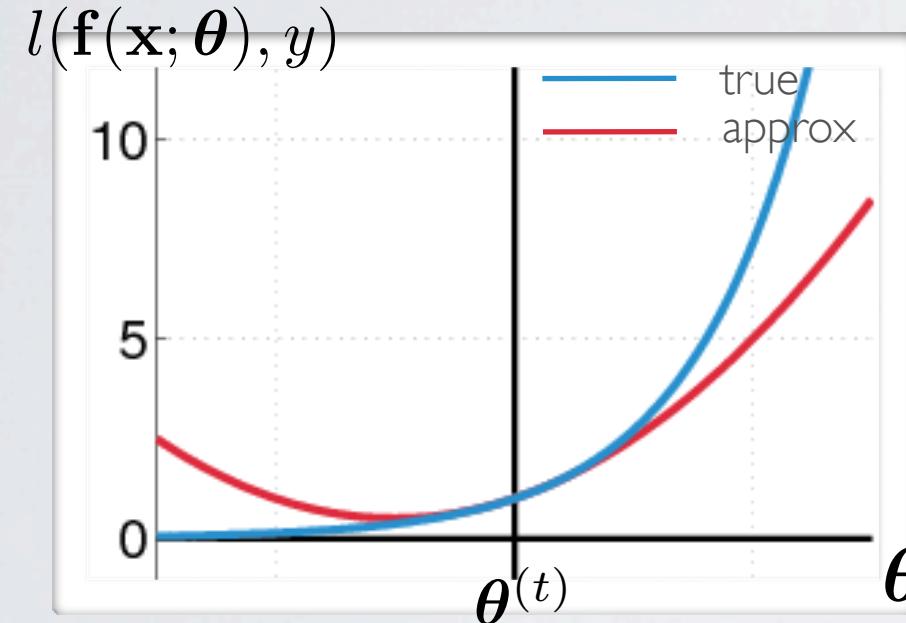
- ▶ can get through plateaus more quickly, by “gaining momentum”

GRADIENT DESCENT

Topics: Newton's method

- If we locally approximate the loss through Taylor expansion:

$$\begin{aligned} l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), y) &\approx l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y) + \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y)^{\top} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) \\ &\quad + 0.5 (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^{\top} \underbrace{\left(\nabla_{\boldsymbol{\theta}}^2 l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y) \right)}_{\text{Hessian}} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) \end{aligned}$$



- We could minimize that approximation, by solving:

$$0 = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y) + \left(\nabla_{\boldsymbol{\theta}}^2 l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y) \right) (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})$$

GRADIENT DESCENT

Topics: Newton's method

- We can show that the minimum is:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - (\nabla_{\boldsymbol{\theta}}^2 l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y))^{-1} (\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}^{(t)}), y))$$

- Only practical if:
 - ▶ few parameters (so we can invert Hessian)
 - ▶ locally convex (so the Hessian is invertible)
- See recommended readings for more on optimization of neural networks