Train A MLP Softmax Classifier

CPT_S 434/534 Neural network design and application

In today's class

- Backpropagation: an optimization algorithm to train NNs
- An example of training a Softmax classifier

Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n}\sum_{i=1}^{n} x_i' w x_i - y_i x_i \rightarrow 0 \quad \Rightarrow \quad XX'w^* - XY = 0 \quad \Rightarrow \quad w^* = (XX')^{-1}XY$$

- Matrix multiplication:

XX': d × n × d XY: d × n $(XX')^{-1}XY: d \times d \times n$ $\rightarrow O(d^2 n)$

- Inverse of a matrix:

$$(XX')^{-1}:\ O(d^{2.373})$$

- Total complexity

$$O(d^2 n + d^{2.373})$$

Optimization for machine learning

- First-order algorithms (commonly used and researched in machine learning)
  - Gradient descent
  - Momentum methods
  - Stochastic variants
  - Hessian vector products
  - ……

Optimization for machine learning

- First-order algorithms (commonly used and researched in machine learning)
  - Gradient descent
  - Momentum methods
  - Stochastic variants
  - Hessian vector products
  - ……

First-order → need to compute gradients

Optimization for machine learning

- First-order algorithms (commonly used and researched in machine learning)
  - Gradient descent
  - Momentum methods
  - Stochastic variants
  - Hessian vector products
  - ……

$$h_S^{\mathrm{ERM}} = \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_S(h).$$

First-order → need to compute gradients

Optimization for machine learning

- First-order algorithms (commonly used and researched in machine learning)
  - Gradient descent
  - Momentum methods
  - Stochastic variants
  - Hessian vector products
  - ……

$$h_S^{\mathrm{ERM}} = \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_S(h).$$

$$\min_{w_1,\ldots,w_K} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left(- \sum_{k=1}^{K} y_{ik} \cdot \log(f(w_k; x_i)) \right)}_{\triangleq L_i(W) \text{ (see Section 2)}},$$

First-order → need to compute gradients

Optimization for machine learning

- First-order algorithms (commonly used and researched in machine learning)
  - Gradient descent
  - Momentum methods
  - Stochastic variants
  - Hessian vector products
  - ……

First-order → need to compute gradients

$$h_S^{\mathrm{ERM}} = \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_S(h).$$

$$\min_{w_1,\ldots,w_K} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left(-\sum_{k=1}^{K} y_{ik} \cdot \log(f(w_k; x_i)) \right)}_{\triangleq L_i(W) \text{ (see Section 2)}},$$

$$\frac{\partial F}{\partial w_k} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L_i}{w_k} + \lambda w_k.$$

Gradients for updating

Determining model parameters

Analytical solution $O(d^2 n + d^{2.373})$

- Stochastic gradient descent (SGD)

Randomly sample b data

$b>=1$

$$\nabla_w f(w) = \frac{1}{b}\sum_{i=1}^{b} x_i' w x_i - y_i x_i \rightarrow O(db)$$

$dn \gg b/\epsilon$

$\epsilon = O(1/\sqrt{n})$

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d)$$ → An iterative algorithm

$O(db\frac{1}{\epsilon})$

Theorem 5 *Set the parameters $T_1 = 4$ and $\eta_1 = \frac{1}{\lambda}$ in the* Epoch-GD *algorithm. The final point $\mathbf{x}_1^k$ returned by the algorithm has the property that*

$$\mathbb{E}[F(\mathbf{x}_1^k)] - F(\mathbf{x}^\star) \leq \frac{16G^2}{\lambda T}.$$

$= \epsilon(\mathrm{T}) \rightarrow T = O(\frac{1}{\epsilon})$

The total number of gradient updates is at most T.

$O(dbT)$

Hazan, Elad, and Satyen Kale. "Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization." *The Journal of Machine Learning Research* 15, no. 1 (2014): 2489-2512.

Determining model parameters

Analytical solution $O(d^2 n + d^{2.373})$

$dn \gg b/\epsilon$

$\epsilon = O(1/\sqrt{n})$

- Stochastic gradient descent (SGD)

Randomly sample b data

$b>=1$

$$\nabla_w f(w) = \frac{1}{b}\sum_{i=1}^{b} x_i' w x_i - y_i x_i \rightarrow O(db)$$

gradients

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d)$$

gradient descent

An iterative algorithm

$O(db\frac{1}{\epsilon})$

Theorem 5 *Set the parameters $T_1 = 4$ and $\eta_1 = \frac{1}{\lambda}$ in the* EPOCH-GD *algorithm. The final point $\mathbf{x}_1^k$ returned by the algorithm has the property that*

$$\mathbb{E}[F(\mathbf{x}_1^k)] - F(\mathbf{x}^\star) \leq \frac{16G^2}{\lambda T}.$$

$= \epsilon(T) \rightarrow T = O(\frac{1}{\epsilon})$

The total number of gradient updates is at most T.

$O(dbT)$

Hazan, Elad, and Satyen Kale. "Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization." *The Journal of Machine Learning Research* 15, no. 1 (2014): 2489-2512.

How to compute gradient?

$$f(x) \rightarrow ?$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

What if we have composition structure?
f and g both have their own parameters
x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

How to compute gradient?

What if we have composition structure?
f and g both have their own parameters
x is the parameter of function g

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$f(g(x)) \rightarrow ?$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$h^{(1)} = g^{(1)}\left(W^{(1)\top} x + b^{(1)}\right);$$

$$h^{(2)} = g^{(2)}\left(W^{(2)\top} h^{(1)} + b^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example: $h(x) = \log(1 + e^{-x})$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example: $h(x) = \log(1 + e^{-x})$ $= f(g(x))$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

How to compute gradient?

$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$

$f(g(x)) \rightarrow ?$

$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$

$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$

$\vdots$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example: $h(x) = \log(1 + e^{-x})$

$= f(g(x))$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$f(y) = \log(y)$

$g(x) = 1 + e^{-x}$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example:

$$h(x) = \log(1 + e^{-x}) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$f(y) = \log(y) \qquad \nabla f(y) = 1/y$$

$$g(x) = 1 + e^{-x} \qquad \nabla g(x) = -e^{-x}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example:

$$h(x) = \log(1 + e^{-x}) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$f(y) = \log(y) \qquad g(x) = 1 + e^{-x}$$

$$\nabla f(y) = 1/y \qquad \nabla g(x) = -e^{-x}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example:

$$h(x) = \log(1 + e^{-x}) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$f(y) = \log(y) \qquad g(x) = 1 + e^{-x}$$

$$\nabla f(y) = 1/y \qquad \nabla g(x) = -e^{-x}$$

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

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

An example:

$$h(x) = \log(1 + e^{-x}) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$f(y) = \log(y) \qquad g(x) = 1 + e^{-x}$$

$$\nabla f(y) = 1/y \qquad \nabla g(x) = -e^{-x}$$

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

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

An example: $h(x) = \log(1 + e^{-x})$
$= f(g(x))$

$f(y) = \log(y)$ $\quad$ $g(x) = 1 + e^{-x}$

$\nabla f(y) = 1/y$ $\quad$ $\nabla g(x) = -e^{-x}$

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

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \rightarrow ?$$

How to compute gradient?

$$f(x) \to \nabla f(x) = \frac{df}{dx}$$

$$h^{(1)} = g^{(1)}\left(W^{(1)\top} x + b^{(1)}\right);$$

$$h^{(2)} = g^{(2)}\left(W^{(2)\top} h^{(1)} + b^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \to ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \to ?$$

$$f_i \to x_i$$

How to compute gradient?

$$f(x) \to \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \to ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\dots\left(f_2(\underbrace{f_1(x)}_{x_1})\right)\right) \to ?$$

$$f_i \to x_i$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$h^{(1)} = g^{(1)}\left(W^{(1)\top} x + b^{(1)}\right);$$

$$h^{(2)} = g^{(2)}\left(W^{(2)\top} h^{(1)} + b^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\dots\left(f_2\left(\underbrace{f_1(x)}_{x_1}\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$f_n\left(\dots\left(f_2(x_1)\right)\right) \rightarrow ?$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\underbrace{\left(f_2\left(f_1(x)\right)\right)}_{x_2}\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

How to compute gradient?

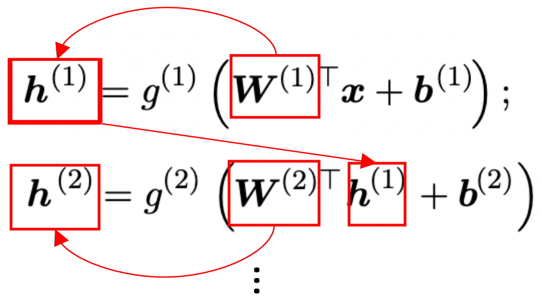
$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$



- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

Q: is $\frac{dx_n}{dx_1}$ enough to update the model (a lot of layers)?

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \cdots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

Q: is $\frac{dx_n}{dx_1}$ enough to update the model (a lot of layers)?

NO. There are parameters to be determined in each layer

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

Q: is $\frac{dx_n}{dx_1}$ enough to update the model (a lot of layers)?

NO. There are parameters to be determined in each layer

We still need $\frac{dx_n}{dx_i}, for\ i = 1, \dots, n-1$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

Q: gradient at other hidden layers?

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

How to compute gradient?

$$f(x) \rightarrow \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

Q: gradient at other hidden layers?

$$\frac{dx_n}{dx_i}$$

How to compute gradient?

$$f(x) \to \nabla f(x) = \frac{df}{dx}$$

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right);$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\vdots$$

What if we have composition structure?

f and g both have their own parameters

x is the parameter of function g

$$f(g(x)) \to ?$$

- Chain rule of calculus (generalize to multi-dimensional cases)

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \to ?$$

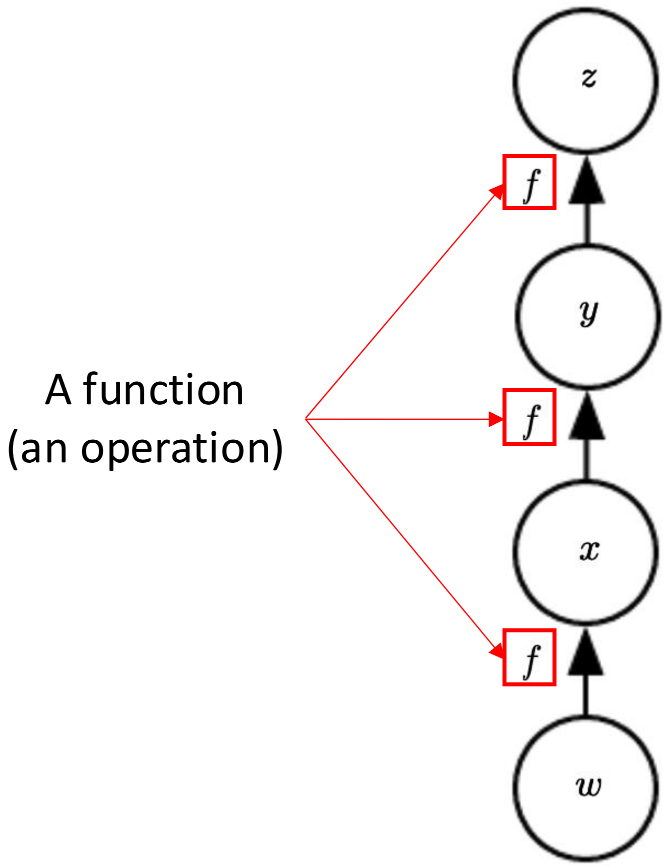
$$f_i \to x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

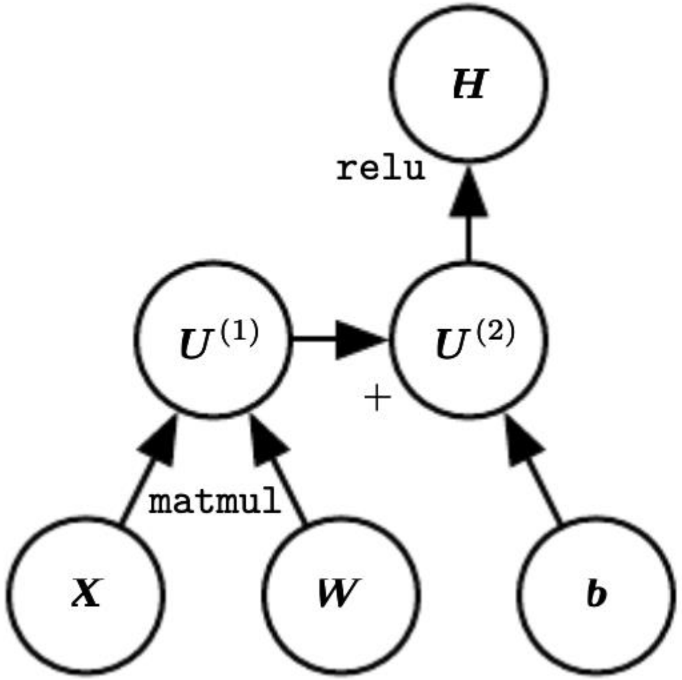
Q: gradient at other hidden layers?

$$\frac{dx_n}{dx_i} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \ldots \cdot \frac{dx_{i+1}}{dx_i}$$

Computation graphs



Computation graphs



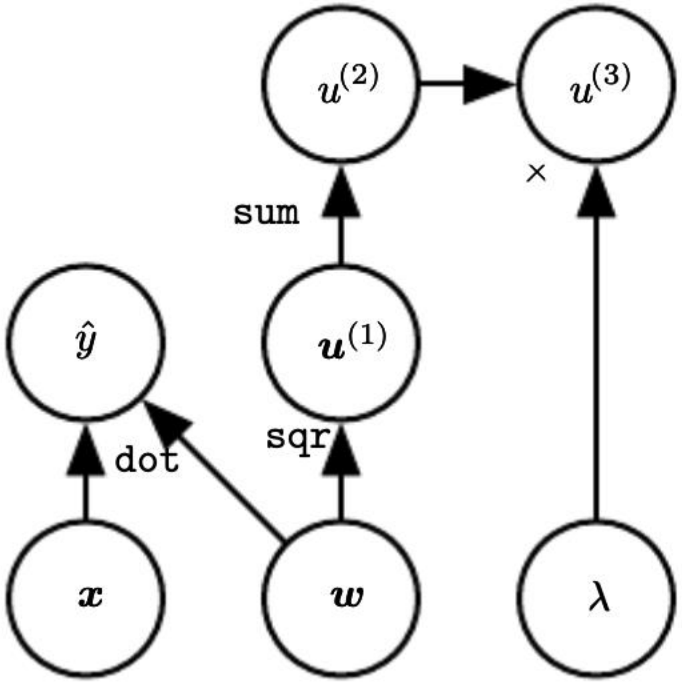


Figure 6.8
“Deep Learning”

It is a precise language to describe structure of operations in neural networks

Computation graphs

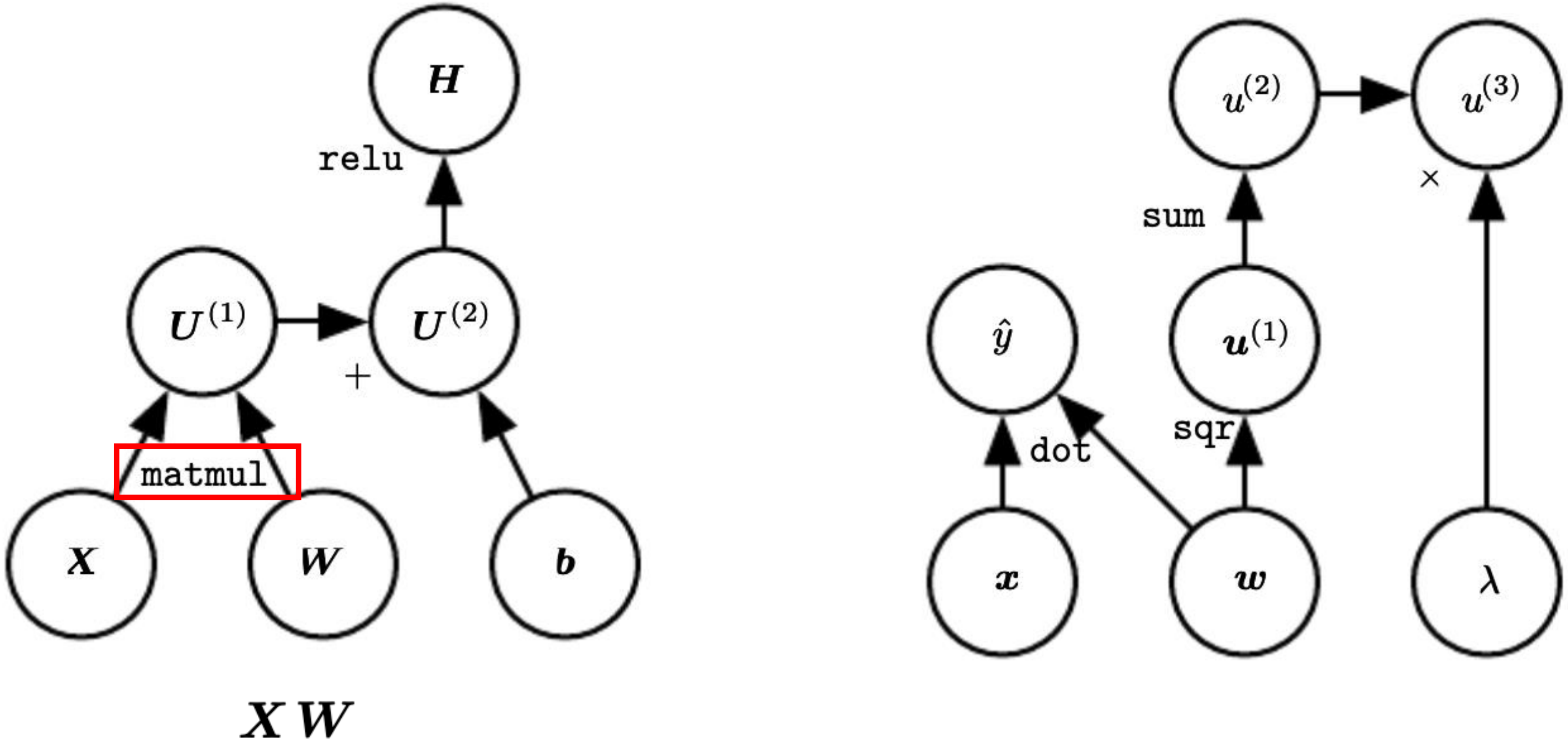
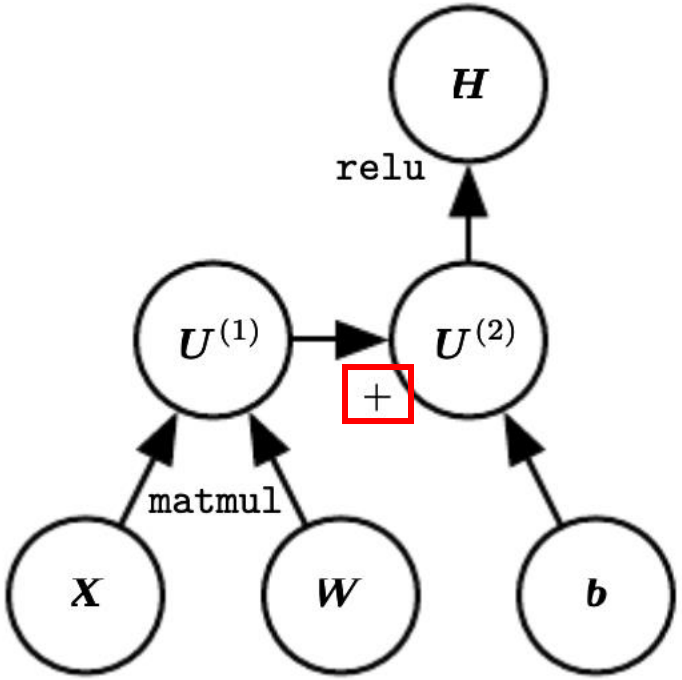


Figure 6.8
"Deep Learning"

It is a precise language to describe structure of operations in neural networks

Computation graphs



$XW + b$

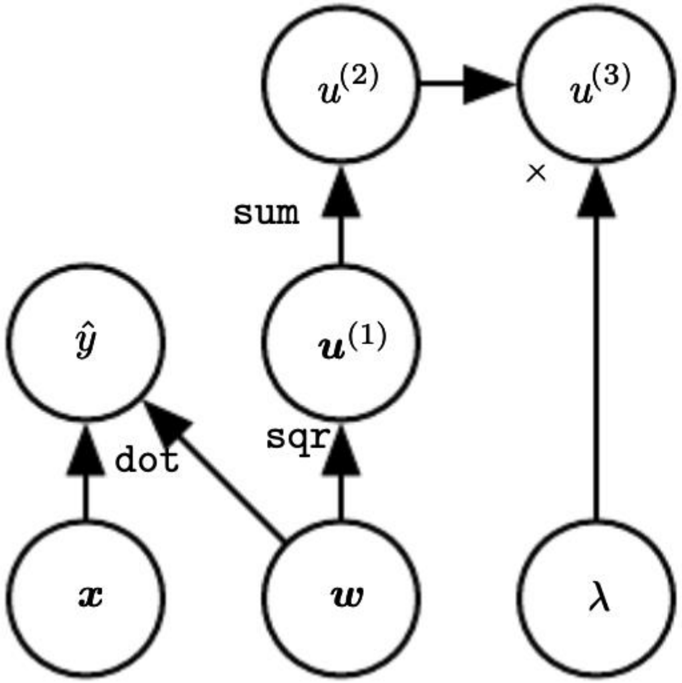
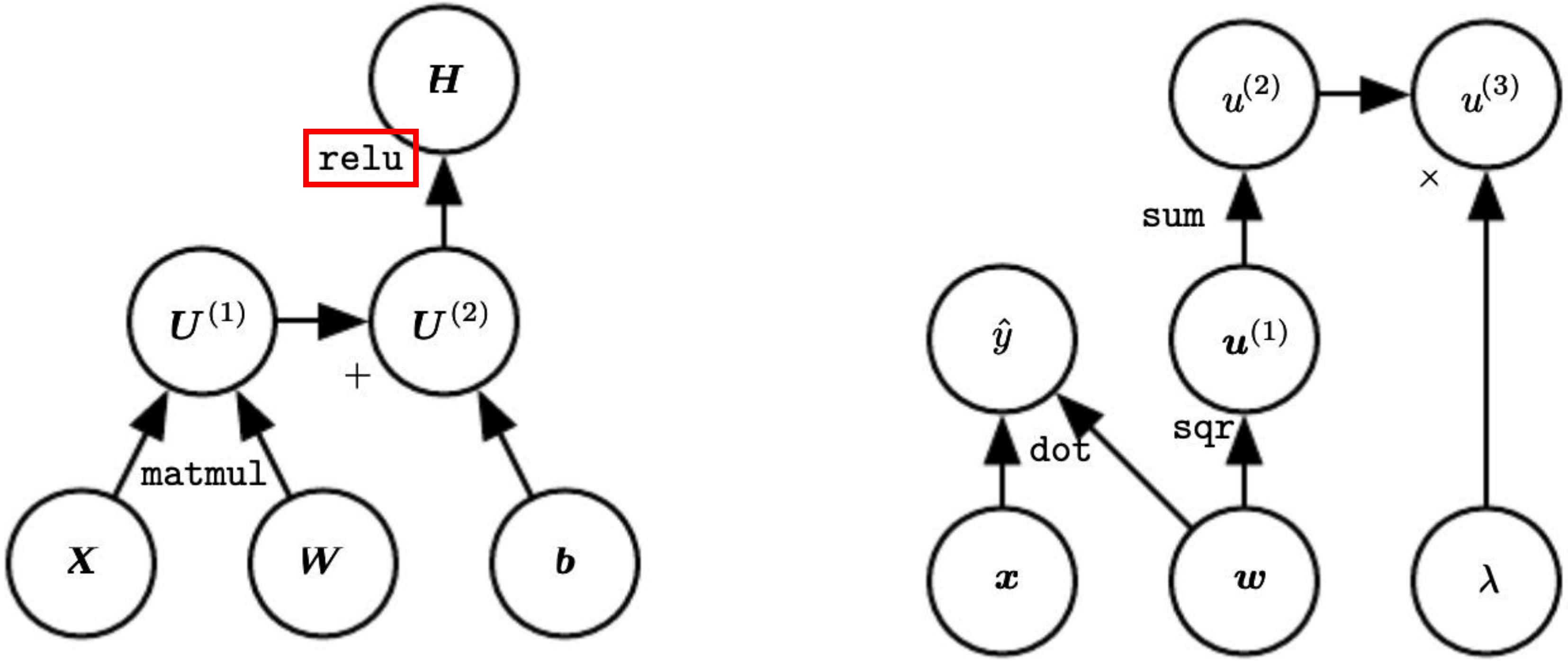


Figure 6.8
"Deep Learning"

It is a precise language to describe structure of operations in neural networks

Computation graphs



$$\max\{\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}, 0\}$$

Figure 6.8
"Deep Learning"

It is a precise language to describe structure of operations in neural networks

Computation graphs

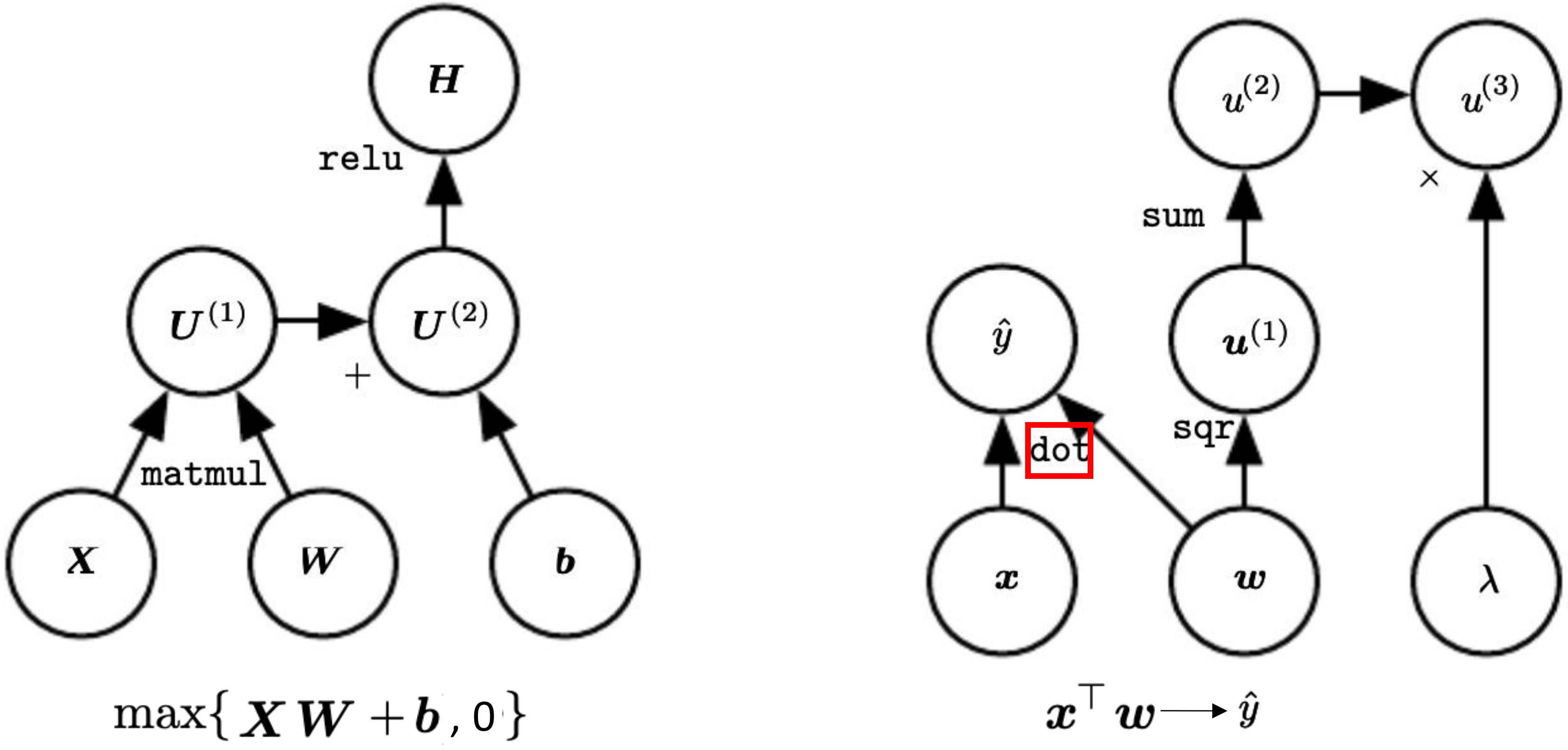
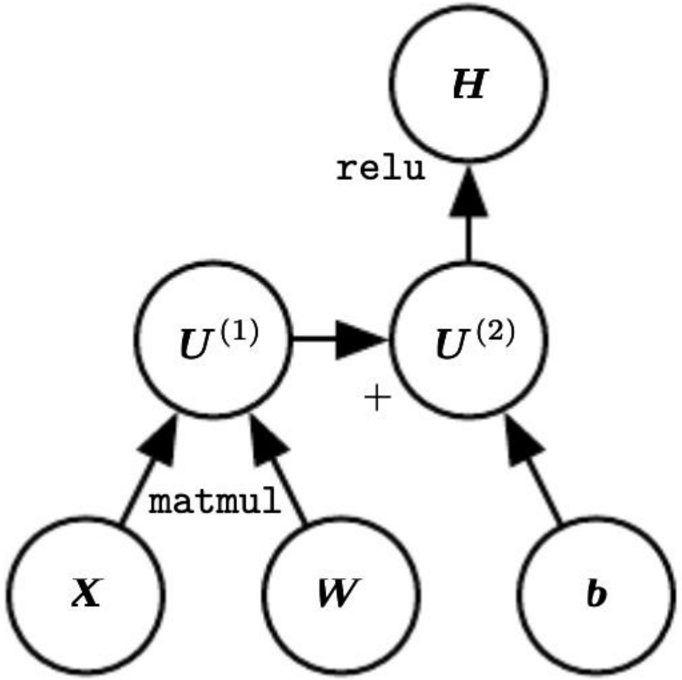


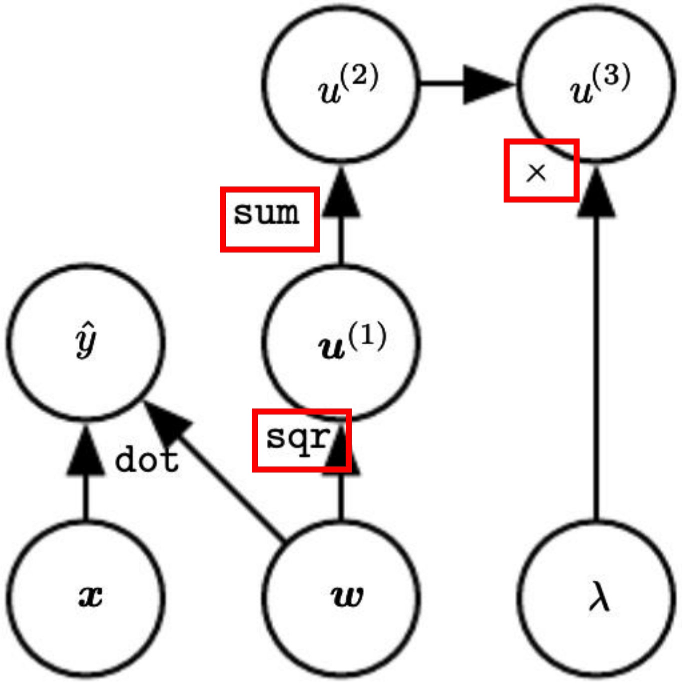
Figure 6.8
"Deep Learning"

It is a precise language to describe structure of operations in neural networks

Computation graphs



$\max\{XW + b, 0\}$

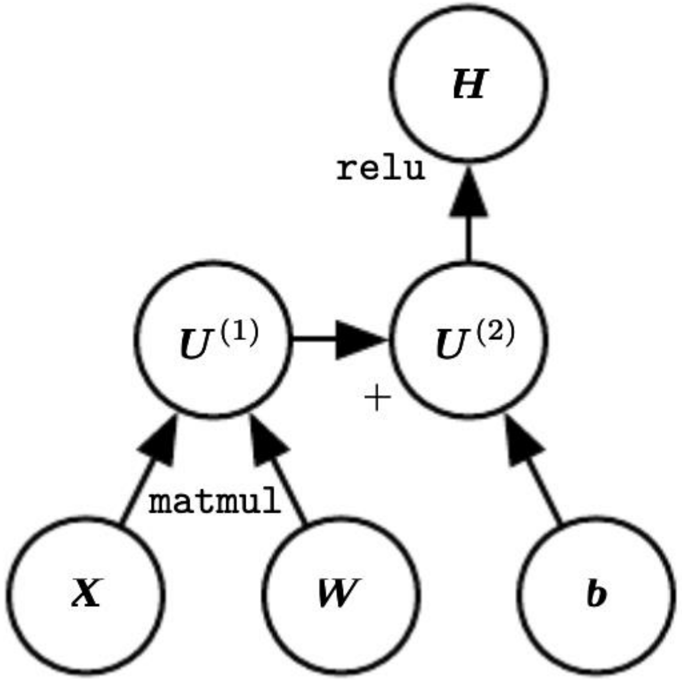


$\begin{cases} x^\top w \longrightarrow \hat{y} \\ \lambda \sum_i w_i^2 \end{cases}$

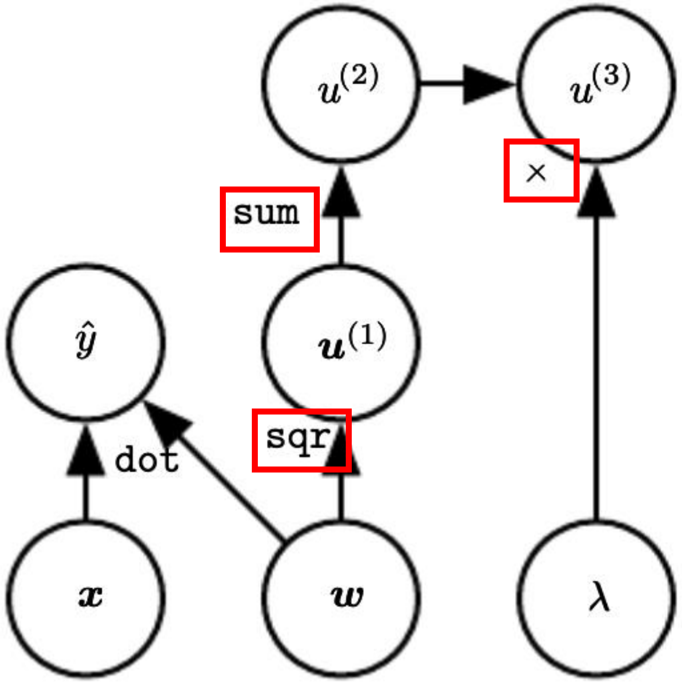
Figure 6.8
"Deep Learning"

It is a precise language to describe structure of operations in neural networks

Computation graphs



$\max\{\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}, 0\}$



$\boldsymbol{x}^\top \boldsymbol{w} \longrightarrow \hat{y}$

$\lambda \sum_i w_i^2$ Regularization

Figure 6.8
“Deep Learning”

It is a precise language to describe structure of operations in neural networks

Forward propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda\Omega(\theta)$

$$h^{(l)}\left(\ldots h^{(3)}\left(h^{(2)}\left(h^{(1)}(x)\right)\right)\right)$$

Algorithm 6.3 in "Deep Learning"

Forward propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
 $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
 $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

$$h^{(l)}\left(\ldots h^{(3)}\left(h^{(2)}\left(h^{(1)}\left(h^{(0)}\right)\right)\right)\right)$$

Algorithm 6.3 in "Deep Learning"

Forward propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

$$h^{(l)}\left(\ldots h^{(3)}\left(h^{(2)}\left(h^{(1)}\left(h^{(0)}\right)\right)\right)\right)$$

Algorithm 6.3 in "Deep Learning"

Forward propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

$$h^{(l)}\left(\ldots h^{(3)}\left(h^{(2)}\left(h^{(1)}\left(h^{(0)}\right)\right)\right)\right)$$

Algorithm 6.3 in “Deep Learning”

Backward propagation

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ Gradient from loss

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$

end for

Algorithm 6.4 in "Deep Learning"

Backward propagation

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ Gradient from loss

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$ Gradient from activation layer

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$

end for

Algorithm 6.4 in "Deep Learning"

Backward propagation

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ Gradient from loss

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$ Gradient from activation layer

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$ Gradient from regularization

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$ Gradient from regularization

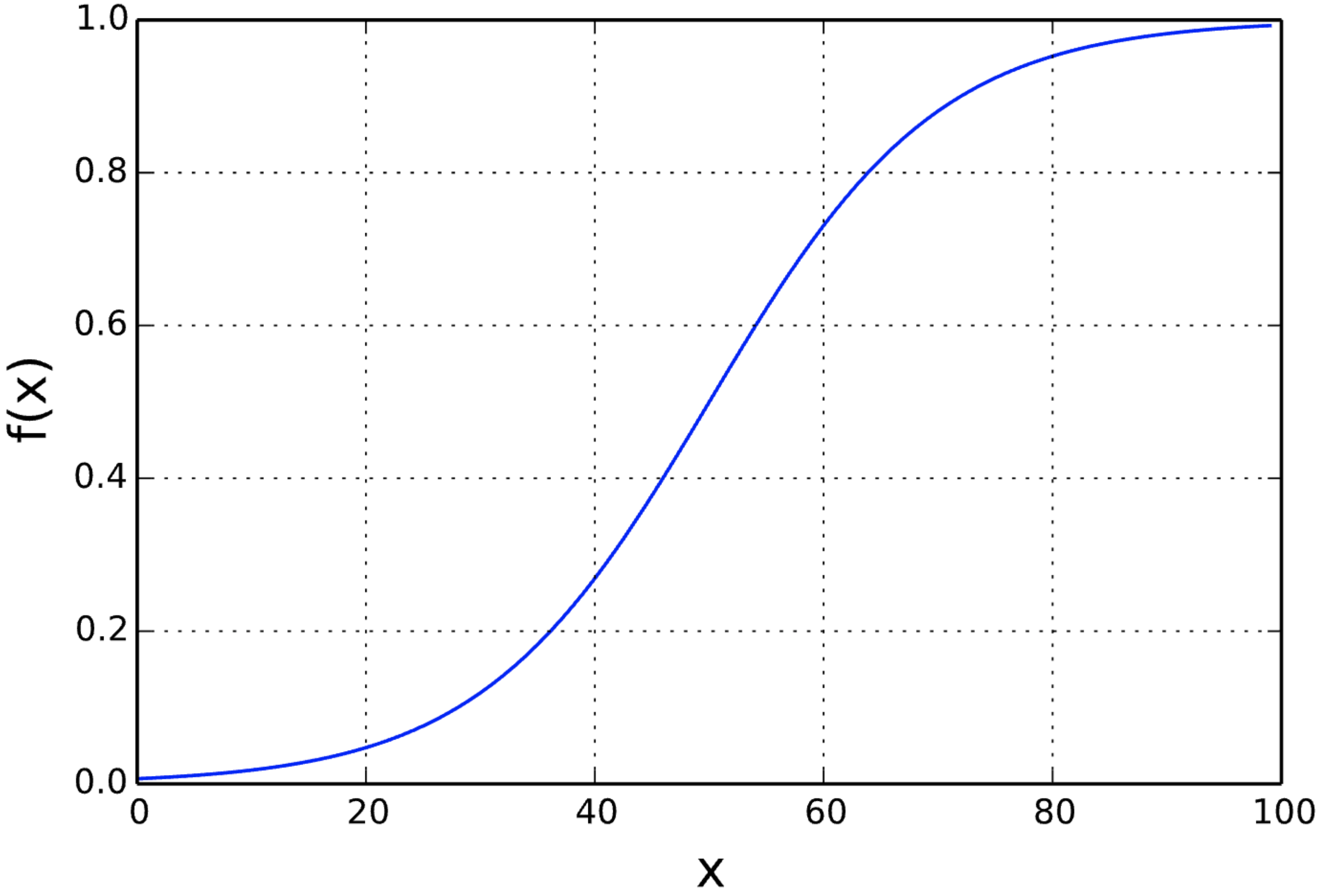
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$

end for

Algorithm 6.4 in "Deep Learning"

Gradient vanish



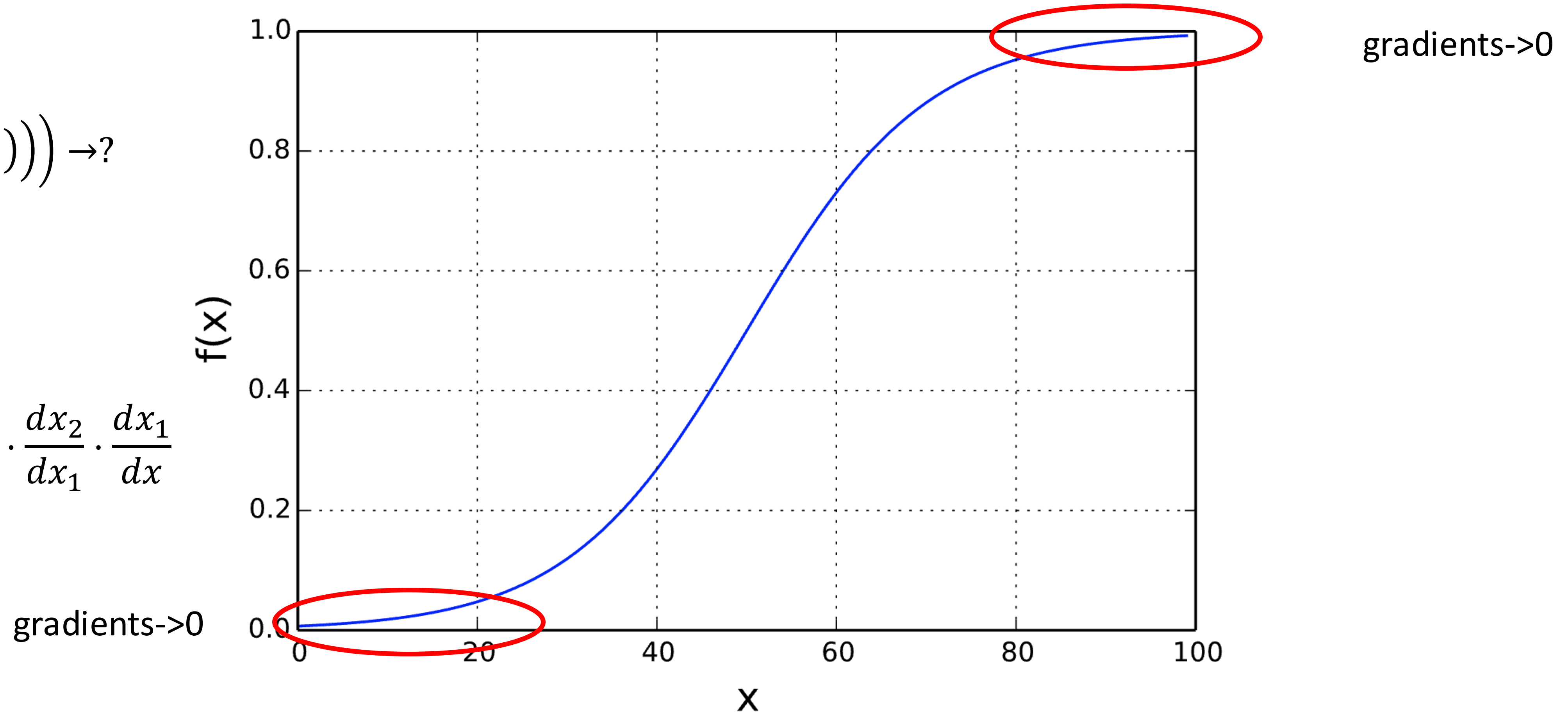
Sigmoid function

Gradient vanish

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$



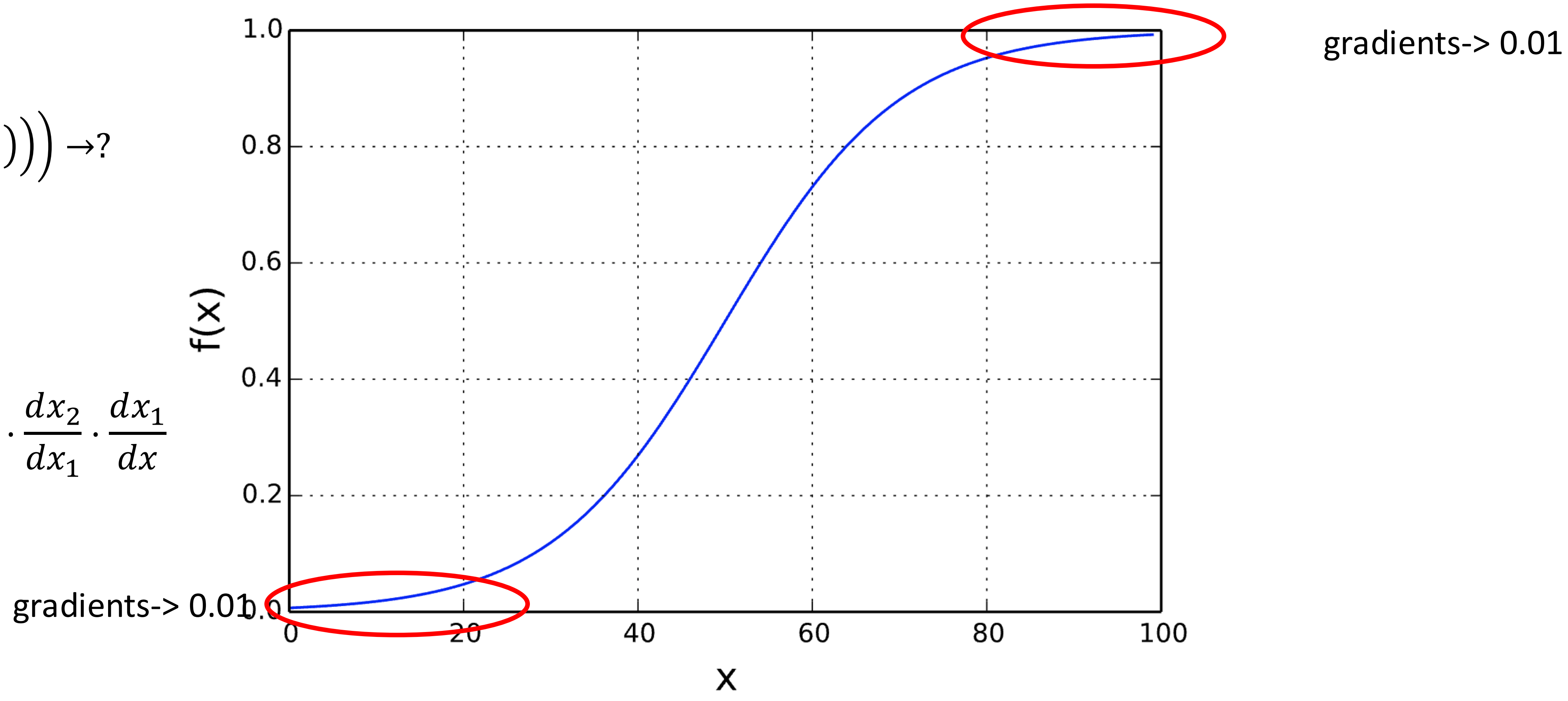
Sigmoid function

Gradient vanish

$$f_n\left(\ldots\left(f_2\left(f_1(x)\right)\right)\right) \to ?$$

$$f_i \to x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$



Sigmoid function

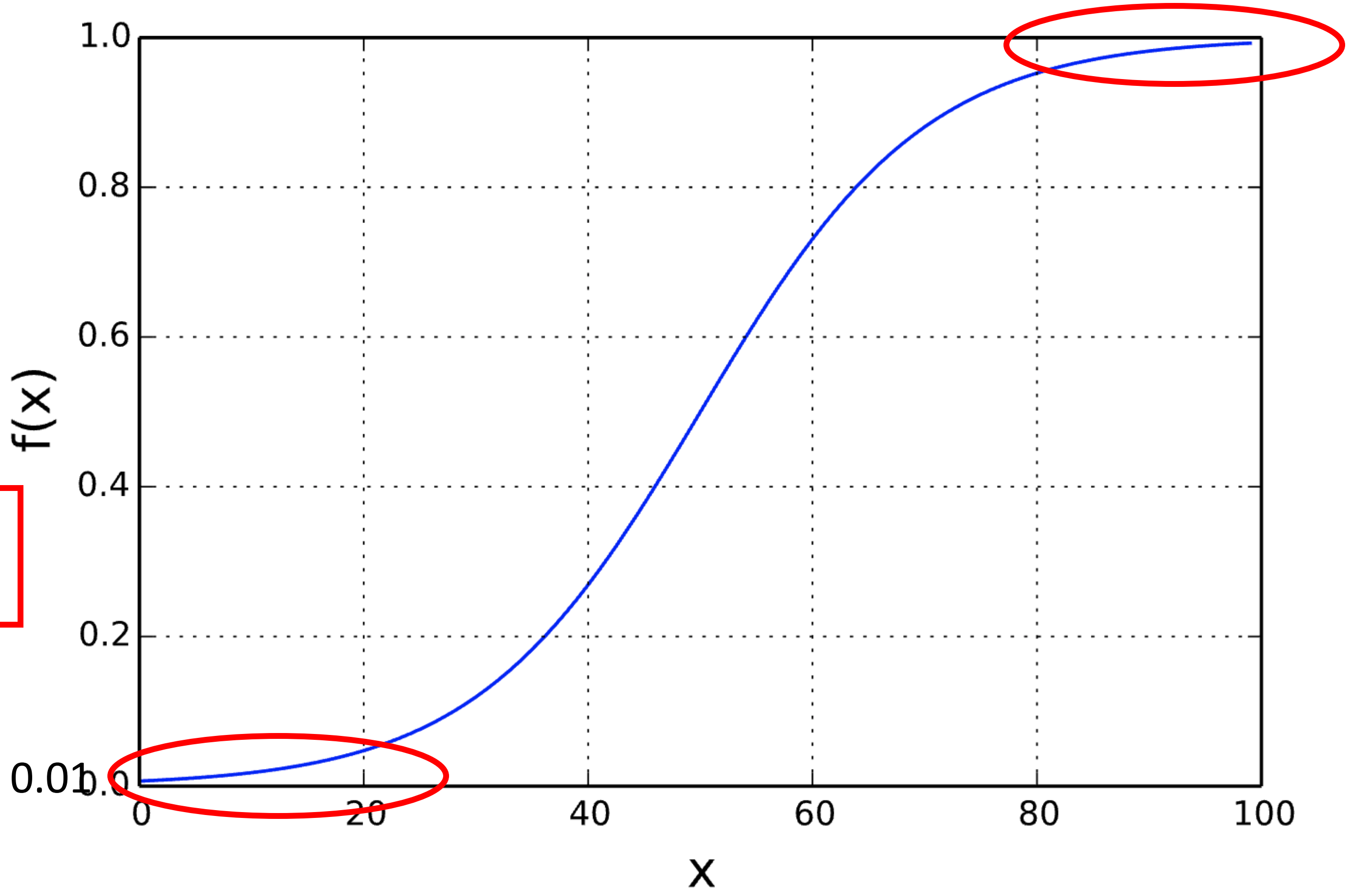
Gradient vanish

$$f_n\left(\ldots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

gradients-> 0.01



gradients-> 0.01

Sigmoid function

Gradient vanish

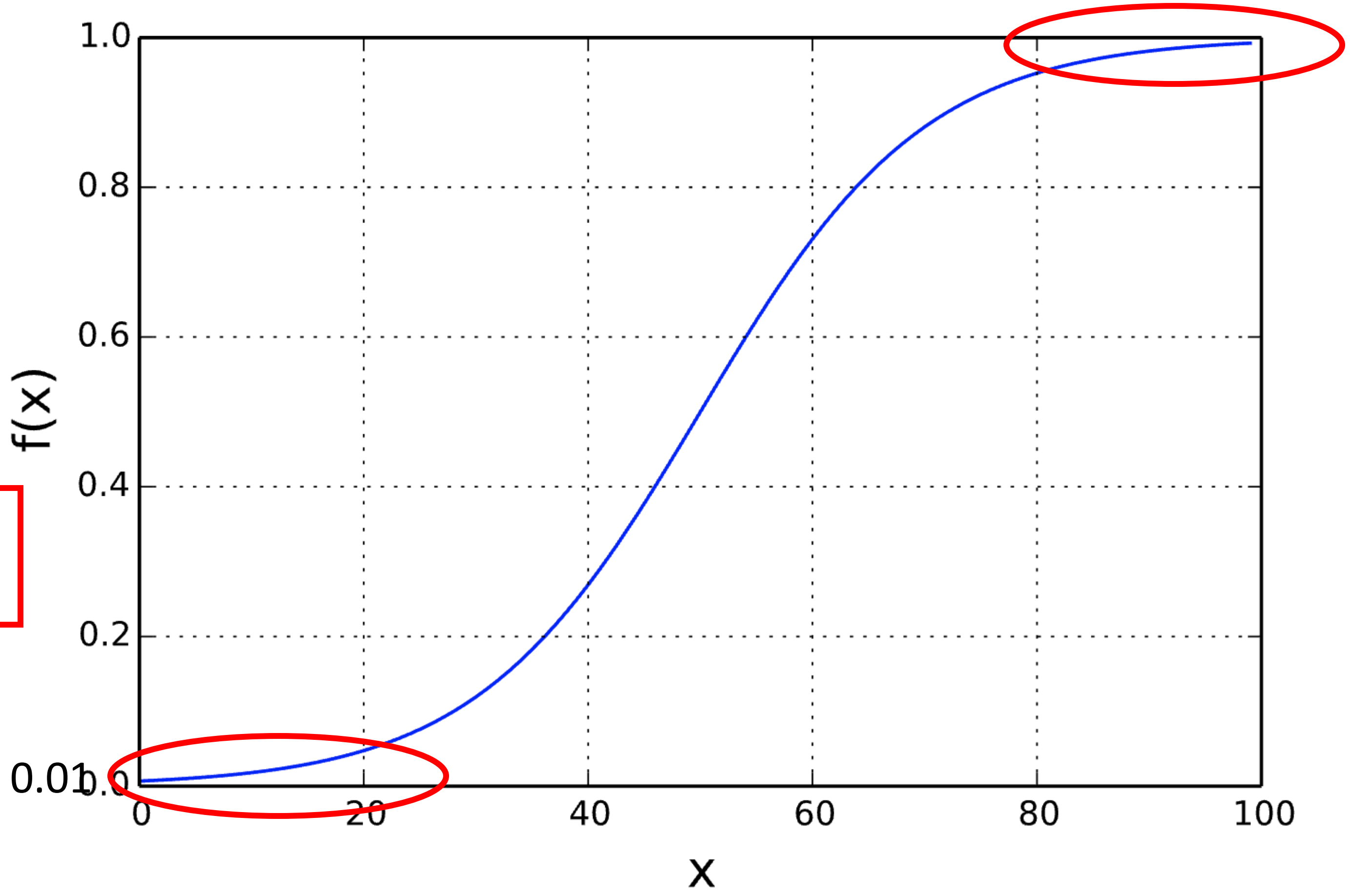
$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \cdots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

$\rightarrow 0.01^n$

gradients-> 0.01



gradients-> 0.01

Sigmoid function

Gradient explosion

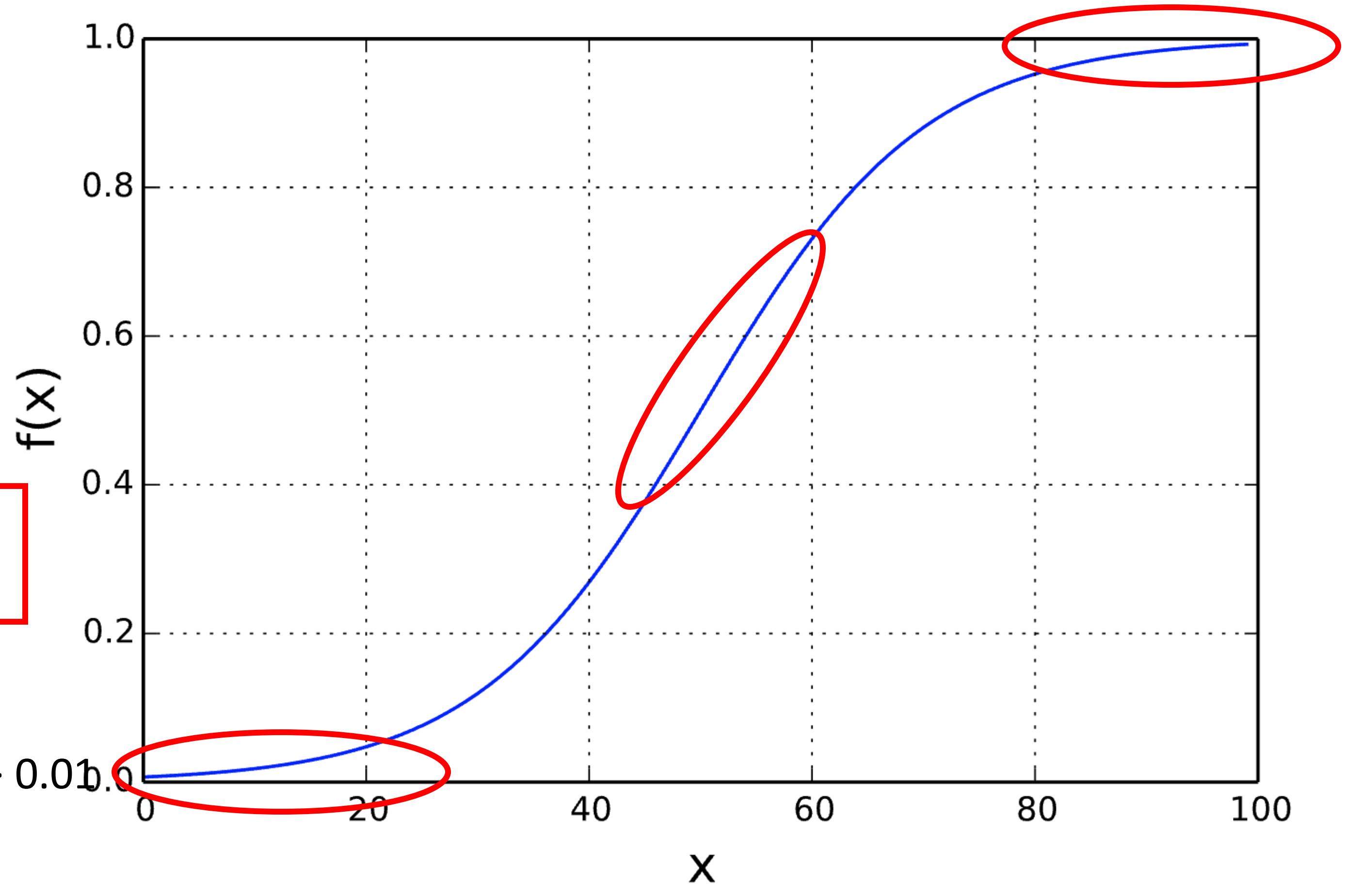
$$f_n\left(\ldots\left(f_2\left(f_1(x)\right)\right)\right) \to ?$$

$$f_i \to x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

$\to 1.1^n$

gradients-> 0.01



gradients-> 0.01

Sigmoid function

Gradient explosion

$$f_n\left(\ldots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

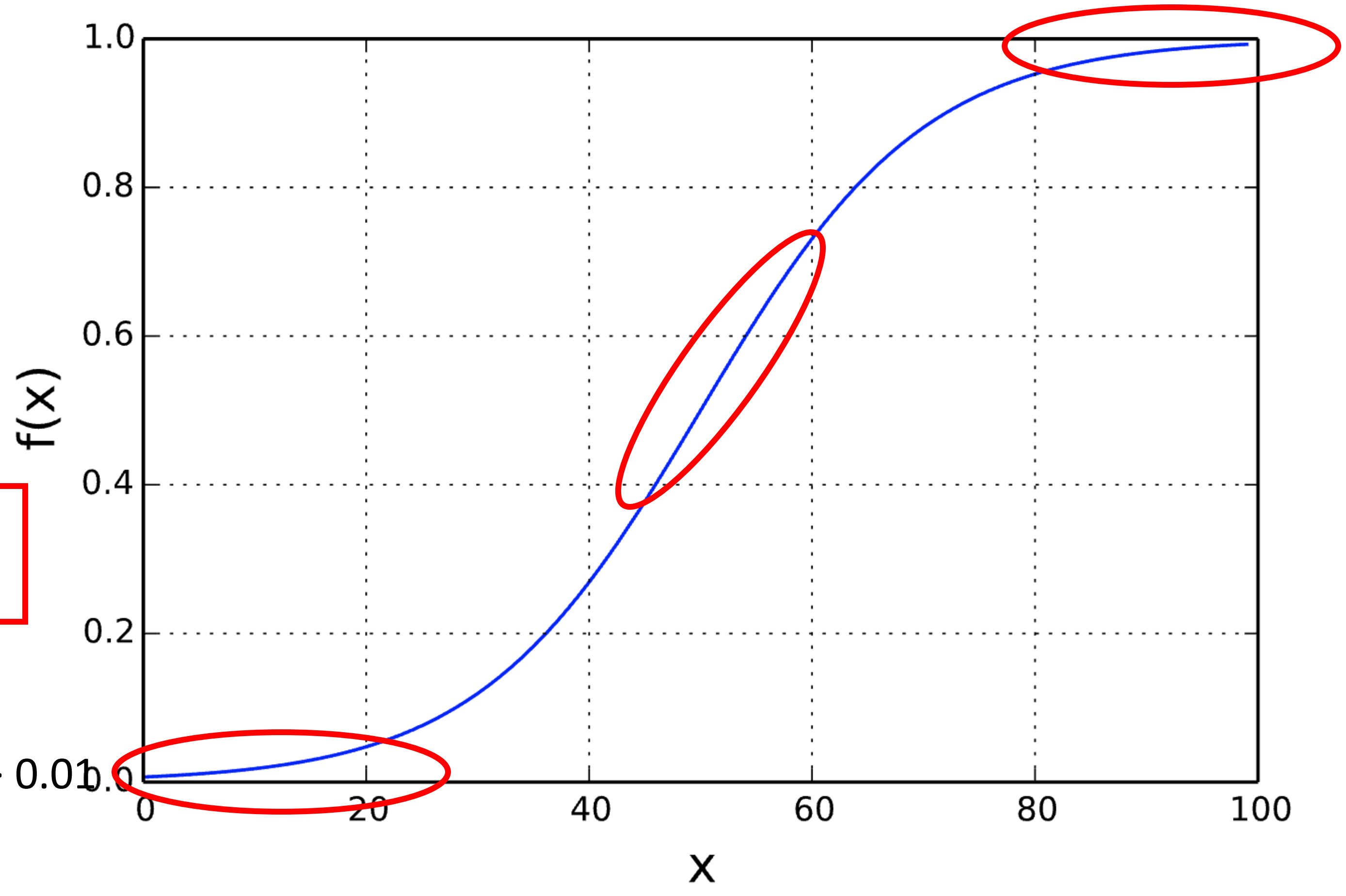
$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx} = \frac{dx_n}{dx_{n-1}} \cdot \ldots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

$\rightarrow 1.1^n$

$1.1^{100} = 13781$

gradients-> 0.01



gradients-> 0.01

Sigmoid function

In today's class

- Backpropagation: an optimization algorithm to train NNs
- An example of training a Softmax classifier

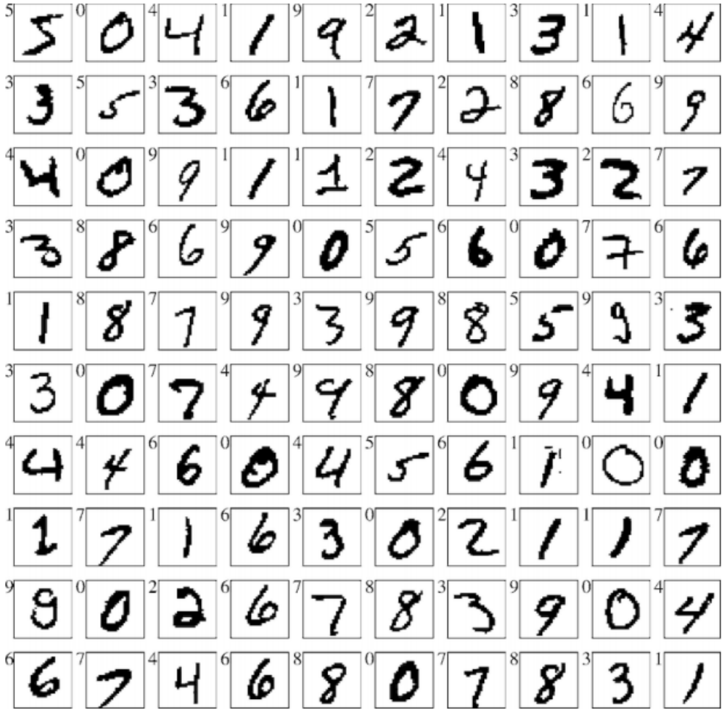
Example: how to train a softmax classifier

The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Task: multi-class classification

- Given a 28×28 image, predict the digit.
- Learn a function $\mathbf{f}: \mathbb{R}^{28\times28} \mapsto \mathbb{R}^{10}$.
- The i-th entry of $\mathbf{f}(\mathbf{x})$ indicates how likely the image $\mathbf{x}$ is the digit i.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

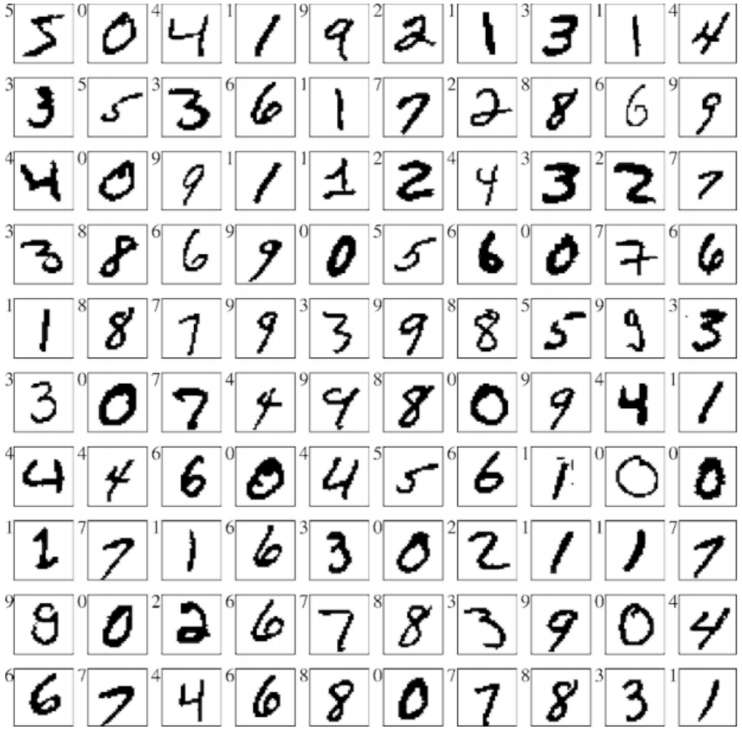
Example: how to train a softmax classifier

Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So **x** becomes 785-dim.)
  Bias term is absorbed

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



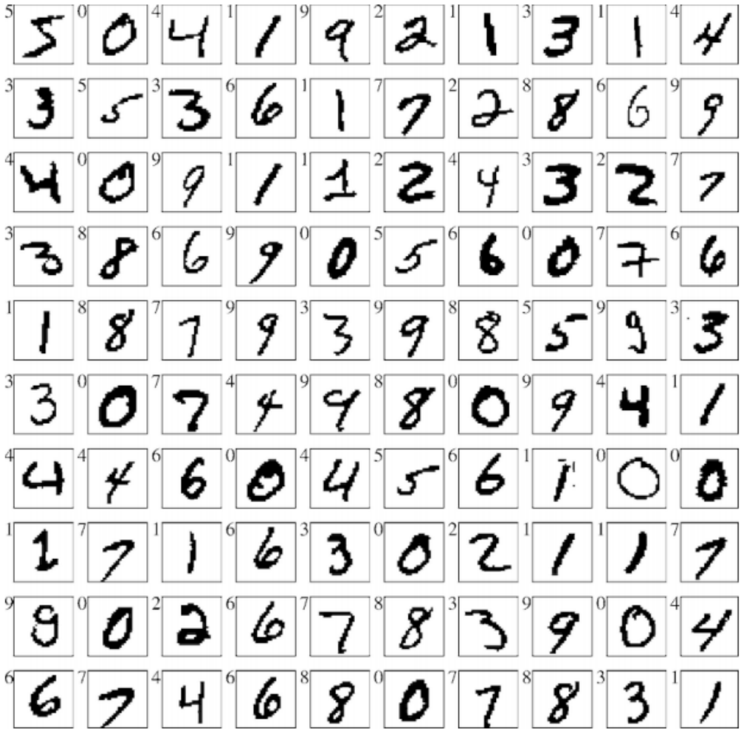
Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So $\mathbf{x}$ becomes 785-dim.)
- Let $\mathbf{W} \in \mathbb{R}^{10 \times 785}$ contain the parameters.
- Let $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{10}$.
- Output a 10-dim vector:

$$\mathbf{f}(\mathbf{x}) = \text{SoftMax}(\mathbf{z}).$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So $\mathbf{x}$ becomes 785-dim.)
- Let $\mathbf{W} \in \mathbb{R}^{10 \times 785}$ contain the parameters.
- Let $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{10}$.
- Output a 10-dim vector:

$$\mathbf{f}(\mathbf{x}) = \text{SoftMax}(\mathbf{z}).$$

$$\text{SoftMax}(\mathbf{z}) = \frac{1}{\sum_{i=0}^{9} \exp(z_i)} [\exp(z_0), \cdots, \exp(z_9)]$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn $\mathbf{W} \in \mathbb{R}^{10\times785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn W $\in \mathbb{R}^{10\times785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn W $\in \mathbb{R}^{10\times 785}$ from the training data

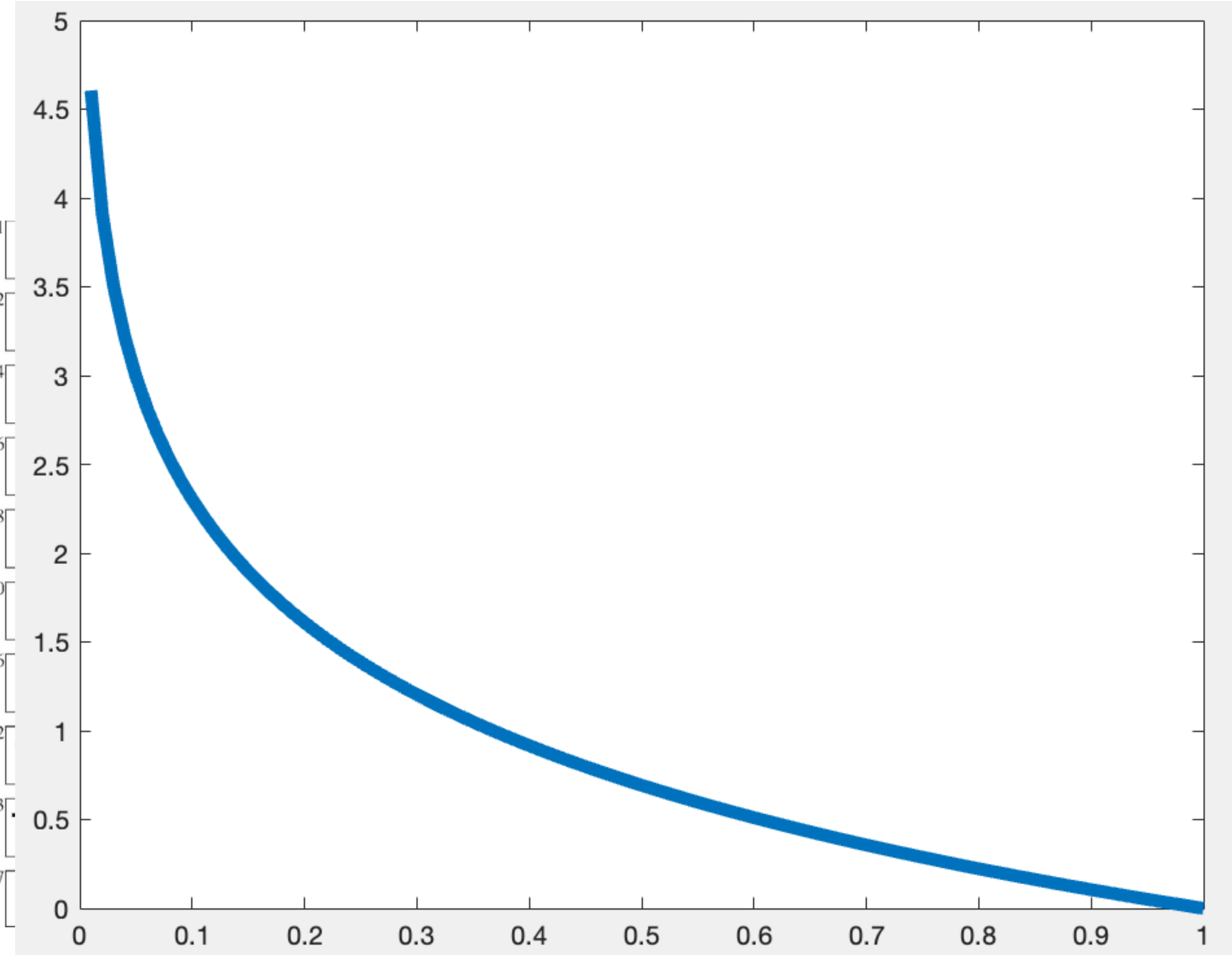
- One-hot encode of the labels
    - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
    - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
    - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

Q: how to interpret CE loss?

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



$-\log(f_i)$

e training data

$1, 2, \cdots, 9\}$.

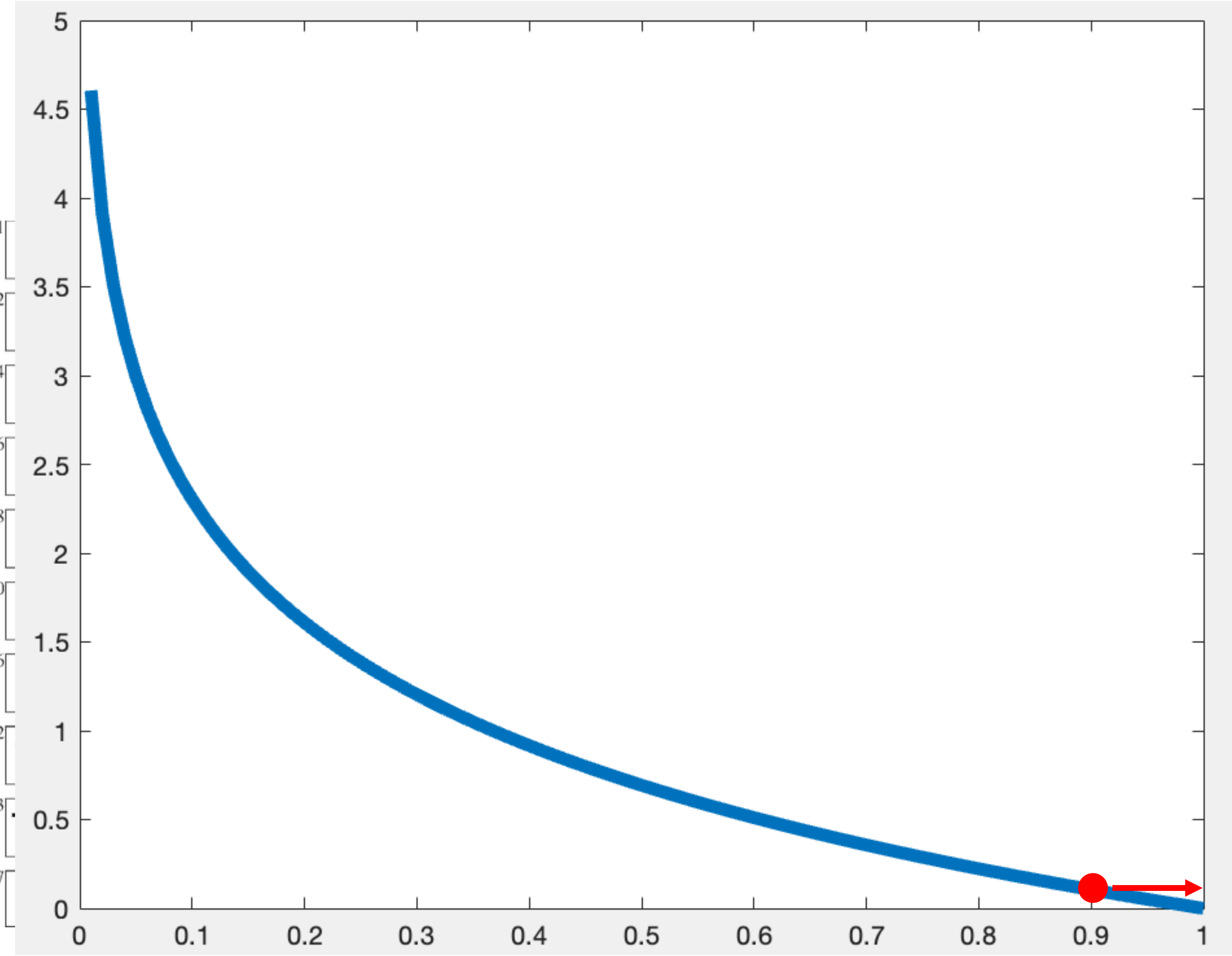
n vector $\{0,1\}^{10}$.

$0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

$\sum_{i=0}^{9} y_i \cdot \log(f_i)$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



e training data

$1, 2, \cdots, 9\}$.

n vector $\{0,1\}^{10}$.

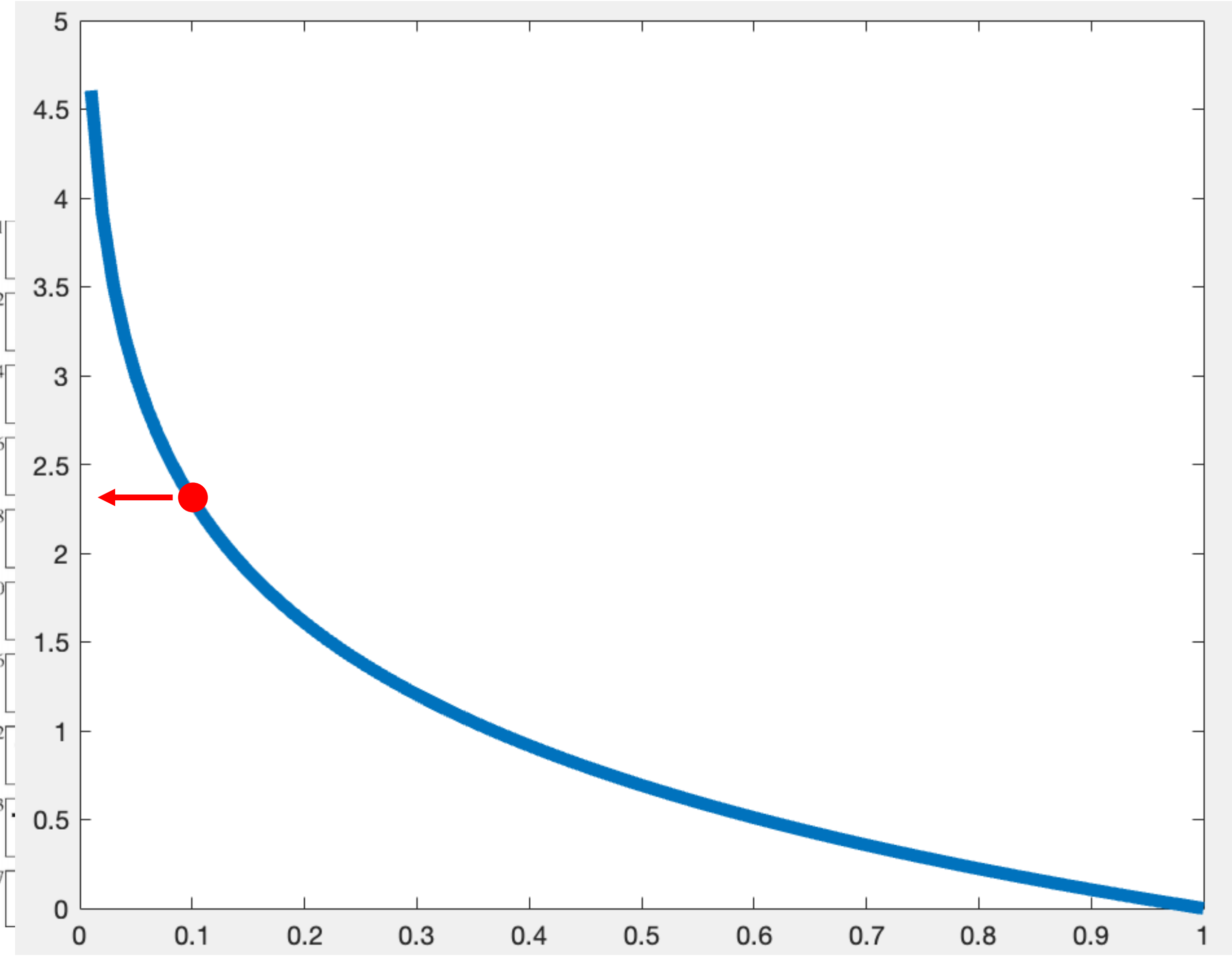
$0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

$\sum_{i=0}^{9} y_i \cdot \log(f_i)$.

$-\log(f_i)$ $\quad$ $- y_i \cdot \log(f_i) \rightarrow 0$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



e training data

$1, 2, \cdots, 9\}$.

vector $\{0,1\}^{10}$.

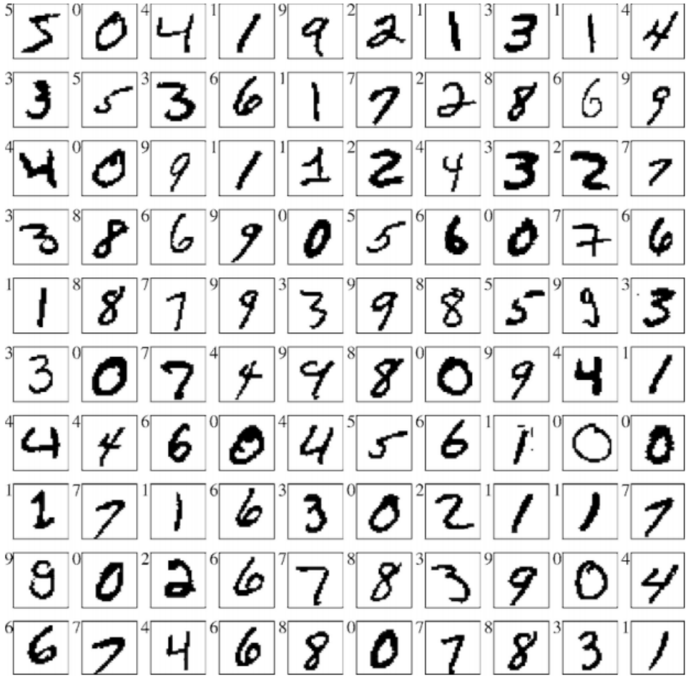
$), 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

$\sum_{i=0}^{9} y_i \cdot \log(f_i)$.

$-y_i \cdot \log(f_i) \rightarrow \infty$ $\qquad$ $-\log(f_i)$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Learn $\mathbf{W} \in \mathbb{R}^{10\times785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

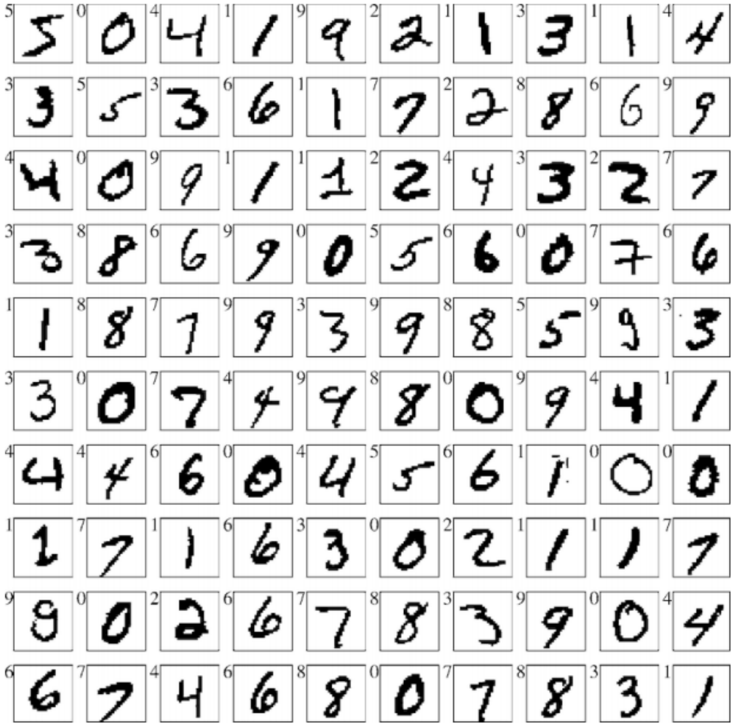
- Solve the optimization model:

$$\mathbf{W}^\star = \underset{\mathbf{W}}{\text{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{CrossEntropy}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\right) \right\}.$$

W is the parameter of **f**

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

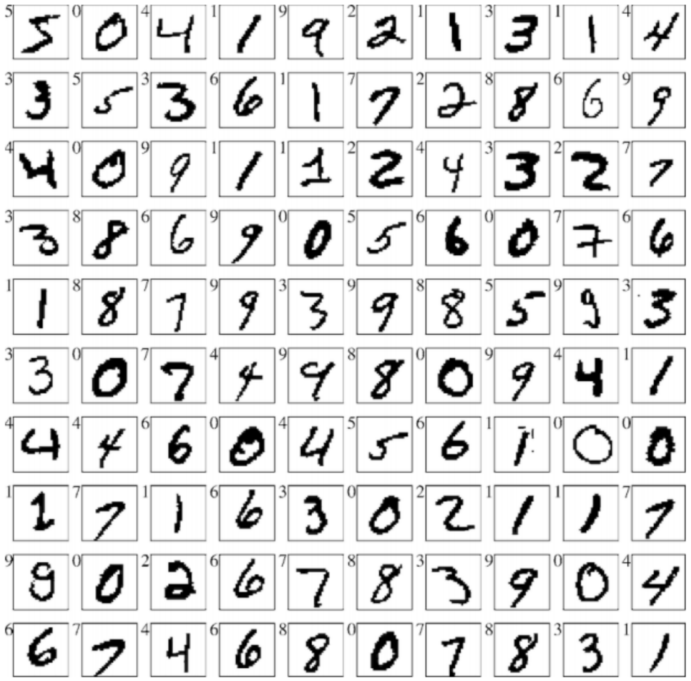


Make prediction for a test sample $\mathbf{x}'$

- Now we have $\mathbf{W}^{\star} \in \mathbb{R}^{10\times785}$.
- For a test sample $\mathbf{x}'$, compute $\mathbf{z} = \mathbf{W}^{\star}\mathbf{x}' \in \mathbb{R}^{10}$.
- Make prediction by $\text{argmax}\,\mathbf{z}$.
  - If the 7-th entry of $\mathbf{z}$ is the largest, then the model thinks the image is digit “7”.

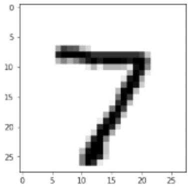
Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Make prediction for a test sample $\mathbf{x}'$

- Now we have $\mathbf{W}^\star \in \mathbb{R}^{10\times785}$.
- For a test sample $\mathbf{x}'$, compute $\mathbf{z} = \mathbf{W}^\star\mathbf{x}' \in \mathbb{R}^{10}$.
- Make prediction by $\text{argmax}\,\mathbf{z}$.
  - If the 7-th entry of $\mathbf{z}$ is the largest, then the model thinks the image is digit "7".



$\mathbf{z} = [-55.7, -141.4, 18.1, 188.3, -91.3, -26.8, -183.6, 411.2, -142.1, 96.2]$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x} \in \mathbb{R}^{785}$.
- $\mathbf{z} = \mathbf{W}\,\mathbf{x} \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}) = \text{SoftMax}(\mathbf{z})$.

Trainable parameters: $\mathbf{W} \in \mathbb{R}^{10 \times 785}$

Train the function by empirical risk minimization (ERM):

- **Training set:** $(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_n, \mathbf{y}_n) \in \mathbb{R}^{785} \times \mathbb{R}^{10}$.
- **Loss function:** $\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=1}^{10} y_i \cdot \log(\mathbf{f}(\mathbf{x})_i)$.
- **Solve ERM:** $\underset{\mathbf{W}}{\text{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{CrossEntropy}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\right) \right\}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

- **How to solve** $\underset{\mathbf{W}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{CrossEntropy}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\right) \right\}$?

- **Stochastic gradient descent (SGD) with momentum** repeats:
  1. Randomly pick j from $\{1, 2, \cdots, n\}$.
  2. Evaluate the gradient $\mathbf{G}_j = \frac{\partial \text{CrossEntropy}(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j))}{\partial \mathbf{W}} \Big|_{\mathbf{W}=\mathbf{W}_{\text{old}}}$.
  3. Update the momentum: $\mathbf{V}_{\text{new}} = \beta \mathbf{V}_{\text{old}} + \mathbf{G}_j$.
  4. Update $\mathbf{W}$ by $\mathbf{W}_{\text{new}} \leftarrow \mathbf{W}_{\text{old}} - \alpha\, \mathbf{V}_{\text{new}}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(1)} = \text{SoftMax}\left(\mathbf{z}^{(1)}\right) \in \mathbb{R}^{d_1}$.
- **Output:** $\mathbf{f}\left(\mathbf{x}^{(0)}\right) = \mathbf{x}^{(1)}$.

A linear model

Trainable parameter:
- $\mathbf{W}^{(0)} \in \mathbb{R}^{10\times 785}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.

Hidden Layer 1

- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.

Hidden Layer 2

- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.

Output Layer

- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \text{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.

- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

MLP

Trainable parameters:

- $\mathbf{W}^{(0)} \in \mathbb{R}^{d_1 \times 785}$,
- $\mathbf{W}^{(1)} \in \mathbb{R}^{d_2 \times d_1}$,
- $\mathbf{W}^{(2)} \in \mathbb{R}^{10 \times d_2}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

Build an optimization model:

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \ \mathbf{y}_j \big) \right\}$$

E.g., the cross-entropy loss

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\left(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \right) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}^{(0)}_{\text{old}}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial\, \mathrm{Loss}\left(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \right)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}^{(0)}_{\text{old}}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}^{(0)}_{\text{new}} = \mathbf{W}^{(0)}_{\text{old}} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \ \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \ \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}_{\mathrm{old}}^{(0)}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial \, \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}_{\mathrm{new}}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)} - \alpha \, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}_{\text{old}}^{(0)}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial\, \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}_{\text{old}}^{(0)}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}_{\text{new}}^{(0)} = \mathbf{W}_{\text{old}}^{(0)} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \text{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\text{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{Loss}\big(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}_{\text{old}}^{(0)}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial \, \text{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)}=\mathbf{W}_{\text{old}}^{(0)}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}_{\text{new}}^{(0)} = \mathbf{W}_{\text{old}}^{(0)} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

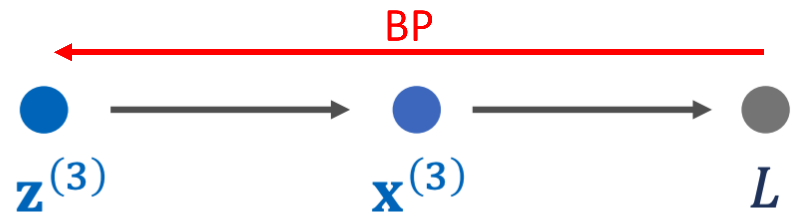
- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \, \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \,)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \ \mathbf{y}_j \,)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.

10-dim vector



Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\dfrac{\partial \, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)$.
- Compute $\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\dfrac{\partial L}{\partial \mathbf{z}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \dfrac{\partial L}{\partial \mathbf{W}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.

$\mathbf{z}^{(3)}$ is a function of $\mathbf{z}^{(2)}$ and $\mathbf{W}^{(2)}$.

Apply the chain rule.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial\, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.

Use it to update $\mathbf{W}^{(2)}$ (e.g., by SGD).

$$\frac{\partial z^{(3)}}{\partial x^{(2)}} = w^{(2)}, \qquad \frac{\partial x^{(2)}}{\partial z^{(2)}} = \begin{cases} 1, & \text{if } z^{(2)} > 0; \\ 0, & \text{else.} \end{cases}$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.

Apply the chain rule again.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$ **?**

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.

Use it to update $\mathbf{W}^{(1)}$ (e.g., by SGD).

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \,\mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\,)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j\,)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.
- $\frac{\partial L}{\partial \mathbf{W}^{(0)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}} \frac{\partial L}{\partial \mathbf{z}^{(1)}}$.

Apply the chain rule again.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial\, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.
- $\frac{\partial L}{\partial \mathbf{W}^{(0)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}} \frac{\partial L}{\partial \mathbf{z}^{(1)}}$. Use it to update $\mathbf{W}^{(0)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

1. Randomly pick a sample $(\mathbf{x}_j, \mathbf{y}_j)$.
2. Run a forward pass (from the input $\mathbf{x}^{(0)}$ to the prediction).
3. Run a backward pass (from the loss to $\mathbf{W}^{(0)}$).

Get the derivatives (stochastic gradients):

$$\frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \quad \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \quad \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}.$$

Update $\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ using the derivatives.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

1. Randomly pick a ~~sample $(\mathbf{x}_j, \mathbf{y}_j)$~~. Several random samples.
2. Run a forward pass (from the input $\mathbf{x}^{(0)}$ to the prediction).
3. Run a backward pass (from the loss to $\mathbf{W}^{(0)}$).

Get the derivatives (stochastic gradients):

~~$$\frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}.$$~~

$$\frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \quad \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \quad \frac{1}{|\mathcal{J}|}\sum_{j\in\mathcal{J}} \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}.$$

Mini-batch should always be used! Set batch size $|\mathcal{J}|$ to 16, 32, 64, …

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

SGD: $\text{BatchSize} = 1.$

- Per-iteration cost is low.
- Lots of iterations to converge.

Mini-Batch: $\text{BatchSize} > 1.$

- Better than the other two, if BatchSize is properly set.

Full Gradient: $\text{BatchSize} = n.$

- Per-iteration cost is n times higher than SGD.
- Convex problem: less number of iterations.
- Neural network: it doesn't work!

See some blogs
https://distill.pub/2017/momentum/
https://ruder.io/optimizing-gradient-descent/

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Rethink BP and chain rule

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx_1} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

$$\frac{dx_n}{dx_i}, for\ i = 1, \dots, n-1$$

Rethink BP and chain rule

$$f(x) \rightarrow \nabla f(x)$$

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Rethink BP and chain rule

$$f(x) \rightarrow \nabla f(x)$$

However, we use **stochastic gradient**, rather than **gradient**

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

However, we use **stochastic gradient**, rather than **gradient**

$f(x) \rightarrow \nabla f(x)$

Stochastic approximation methods are a family of iterative methods typically used for root-finding problems or imation)
collected data is corrupted by noise, or for approximating extreme values of functions which cannot be computed

In a nutshell, stochastic approximation algorithms deal with a function of the form $f(\theta) = \mathrm{E}_{\xi}[F(\theta, \xi)]$ which is
stochastic approximation algorithms use random samples of $F(\theta, \xi)$ to efficiently approximate properties of f su

Recently, stochastic approximations have found extensive applications in the fields of statistics and machine lea
reinforcement learning via temporal differences, and deep learning, and others.[1] Stochastic approximation algo
their theory.[2]

The earliest, and prototypical, algorithms of this kind are the **Robbins–Monro** and **Kiefer–Wolfowitz** algorithms

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Screen shot from https://en.wikipedia.org/wiki/Stochastic_approximation

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

$\hat{\nabla} f(y) \neq \nabla f(y)$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

$\hat{\nabla} f(y) \neq \nabla f(y)$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$\widehat{\frac{dz}{dx}} = \widehat{\frac{dz}{dy}}\widehat{\frac{dy}{dx}}$$

Composition: Biased

Train an MLP softmax classifier

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

- $f_1 \rightarrow ?$
- $f_2 \rightarrow ?$
- $f_3 \rightarrow ?$
- $f_4 \rightarrow ?$
- …

Need to manually implement?

Train an MLP softmax classifier

$$f_n\left(\ldots\left(f_2(f_1(x))\right)\right) \rightarrow ?$$

- $f_1 \rightarrow ?$
- $f_2 \rightarrow ?$
- $f_3 \rightarrow ?$
- $f_4 \rightarrow ?$
- …

Need to manually implement? ✗

Train an MLP softmax classifier

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

```
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

Train an MLP softmax classifier

Define a function $\mathbf{f}: \mathbb{R} \mapsto \mathbb{R}$

Backpropagation:

One iteration:

1. Randomly sample j from $\{1, 2, \cdots, n\}$.
2. **Forward pass**: take x_j as input ($x^{(0)} = x_j$), compute each layer $z^{(1)}, x^{(1)}, z^{(2)}, x^{(2)}, z^{(3)}$.
3. **Backward pass:**
   i. Compute the derivatives $\frac{\partial L}{\partial z^{(3)}}, \frac{\partial L}{\partial w^{(2)}}, \frac{\partial L}{\partial z^{(2)}}, \frac{\partial L}{\partial w^{(1)}}, \frac{\partial L}{\partial z^{(1)}}, \frac{\partial L}{\partial w^{(0)}}$.
   ii. Update $w^{(k)}$ using $\frac{\partial L}{\partial w^{(k)}}$.

Need to compute gradients for each layer?

Train an MLP softmax classifier

Define a function $f: \mathbb{R} \mapsto \mathbb{R}$

Backpropagation:

One iteration:

1. Randomly sample j from $\{1, 2, \cdots, n\}$.
2. **Forward pass**: take x_j as input ($x^{(0)} = x_j$), compute each layer $z^{(1)}, x^{(1)}, z^{(2)}, x^{(2)}, z^{(3)}$.
3. **Backward pass:**
   i. Compute the derivatives $\frac{\partial L}{\partial z^{(3)}}, \frac{\partial L}{\partial w^{(2)}}, \frac{\partial L}{\partial z^{(2)}}, \frac{\partial L}{\partial w^{(1)}}, \frac{\partial L}{\partial z^{(1)}}, \frac{\partial L}{\partial w^{(0)}}$.
   ii. Update $w^{(k)}$ using $\frac{\partial L}{\partial w^{(k)}}$.

Need to compute gradients for each layer?

Train an MLP softmax classifier

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_NN.parameters(), lr=learning_rate, weight_decay=0.00001)
```

```
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)

# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```