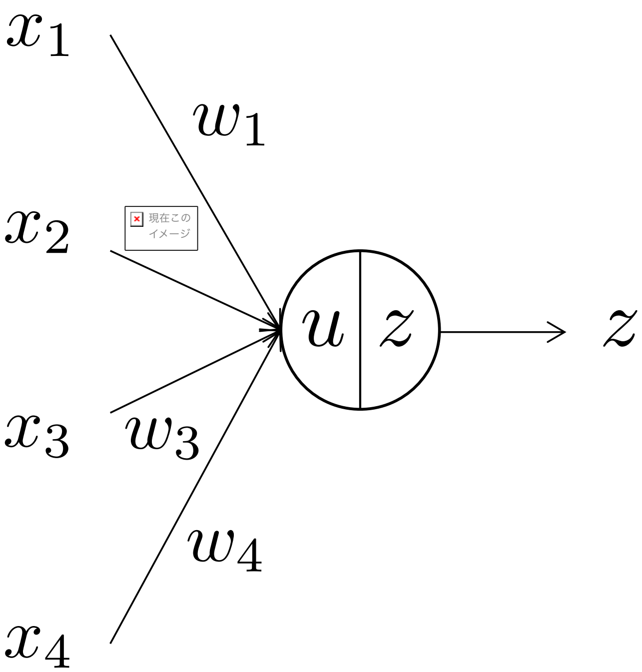
Feedforward neural networks

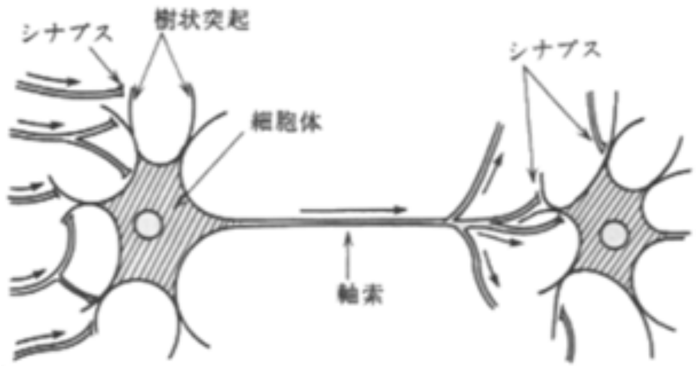
- Units and activation functions
- Multi-layer feedforward networks
- Design of output layers
- Loss functions
- The backpropagation algorithm

Units (neurons)

$$u = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

$$z = f(u)$$





Activation functions

$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$z = f(u)$$

- f is called an activation function

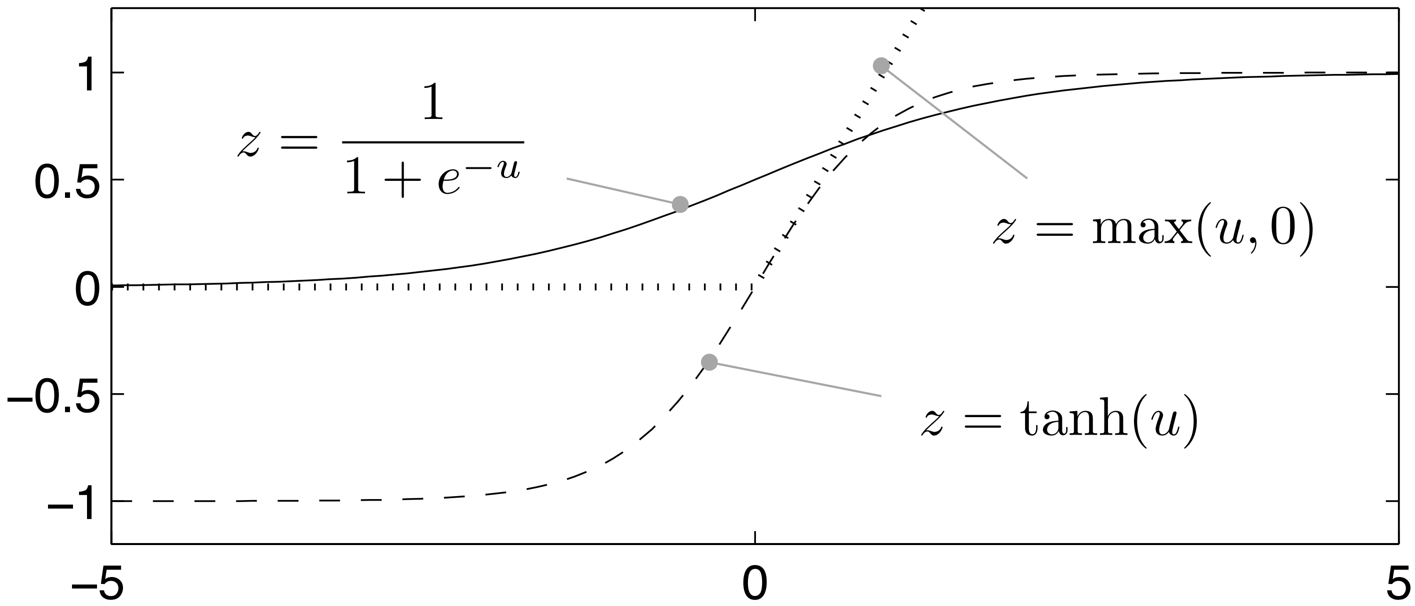
$z = \dfrac{1}{1+e^{-u}}$	logistic function	Classical
$z = \tanh(u)$	hyperbolic tangent	
$z = \max(u, 0)$	ReLU: Rectified Linear Unit	Now the most popular

Activation functions

$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$z = f(u)$$

- *f* is called an activation function



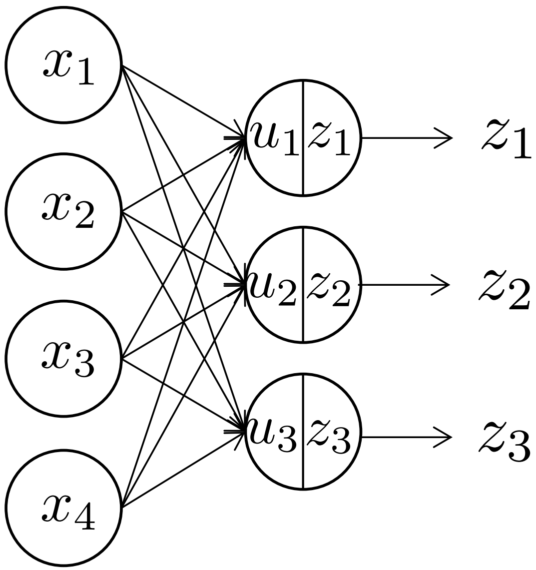
Single-layer networks

$$u_j = \sum_{i=1}^{I} w_{ji} x_i + b_j$$

$$z_j = f(u_j)$$

$$\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{z} = \mathbf{f}(\mathbf{u})$$



$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_J \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_J \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_J \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1I} \\ \vdots & \ddots & \vdots \\ w_{J1} & \cdots & w_{JI} \end{bmatrix}, \quad \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_J) \end{bmatrix}$$

Multi-layer networks

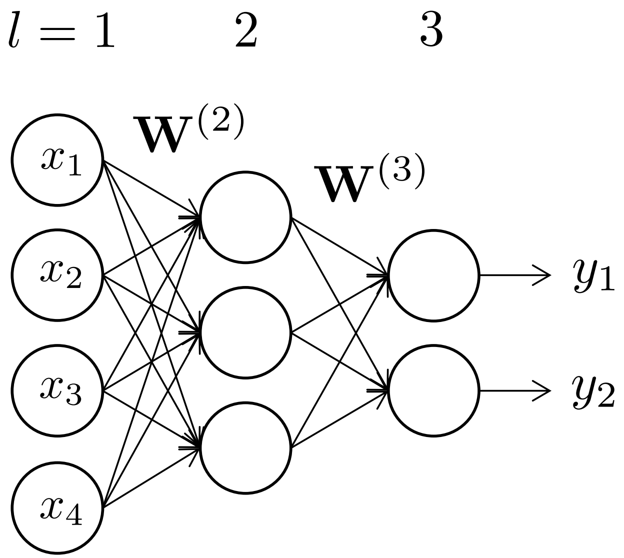
1[st] (input) layer $\mathbf{x} \equiv \mathbf{z}^{(1)}$

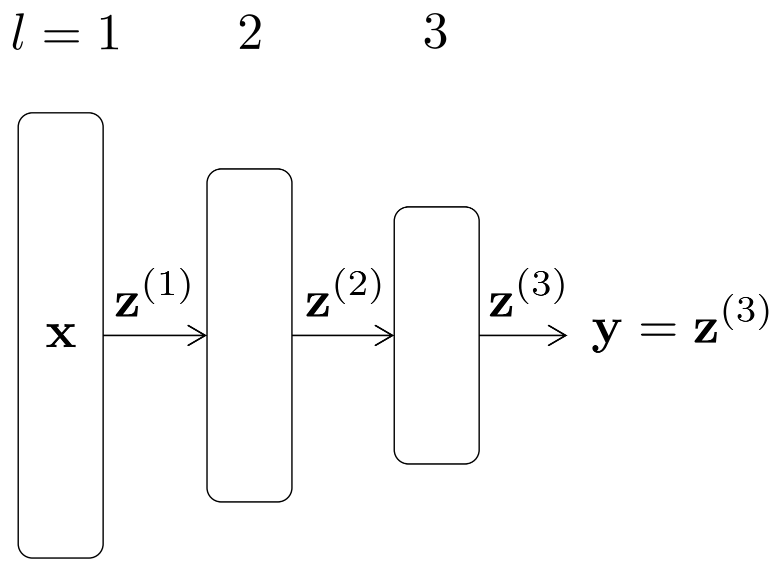
Propagation from lth to (l+1)th layer

$$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)})$$

Lth (output) layer $\mathbf{y} \equiv \mathbf{z}^{(L)}$





Training a feedforward net

- A network represents a function

$$\underbrace{\mathbf{y}(\mathbf{x}; \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(L)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(L)})}_{\mathbf{y}(\mathbf{x}; \mathbf{w})}$$

- We use a set of pairs of an input **x** and an associated output **d**

$$\{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \ldots, (\mathbf{x}_N, \mathbf{d}_N)\}$$

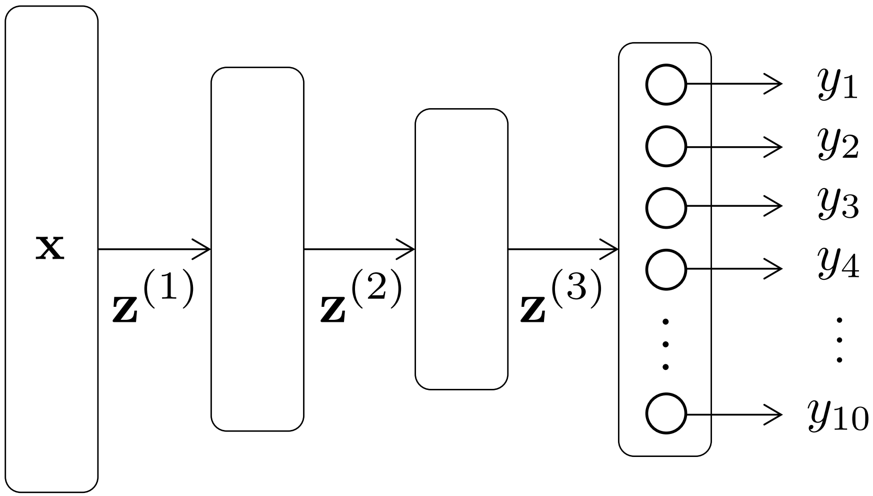
- We wish to determine parameters w =(W, b) so that the function reproduces the data as accurately as possible

$$\mathbf{y}(\mathbf{x}_n; \mathbf{w}) \sim \mathbf{d}_n$$

Designing the output layer and loss function for regression

- Place the same number of units as the target variable at the output layer
- Choose tanh or identity func. for the activation function of the output layer
- Choose the sum of squared difference for the loss function

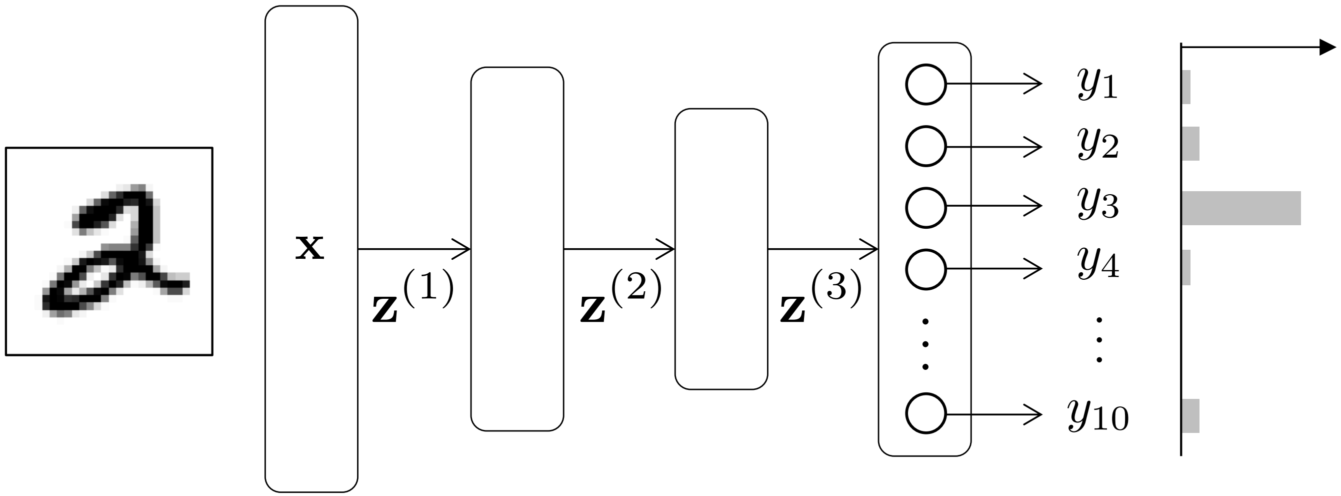
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\|\mathbf{d}_n - \mathbf{y}(\mathbf{x}_n;\mathbf{w})\|^2$$



Designing the output layer and loss function for classification

- Place the same number of units as the number of classes at the output layer
- Choose softmax function for the activation function of the output layer
    - We regard the output of the k^{th} unit as the likelihood of class k
- Choose the *cross entropy loss* for the loss function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n; \mathbf{w})$$



$$p(\mathcal{C}_k|\mathbf{x}) = y_k = z_k^{(L)}$$

Softmax and cross-entropy

- We employ 1-of-k coding (one-hot vector) for representing each class

$$\mathbf{d} = [d_1, d_2, \ldots, d_K]$$

- Softmax function:

$$y_k \equiv z_k^{(L)} = \frac{\exp(u_k^{(L)})}{\sum_{j=1}^{K} \exp(u_j^{(L)})} \qquad \left(\sum_{k=1}^{K} y_k = 1 \right)$$

- We can interpret the output of each unit as a posterior probability of the corresponding class
- Difference between the output of the net and the target value

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n; \mathbf{w})$$

cross-entropy

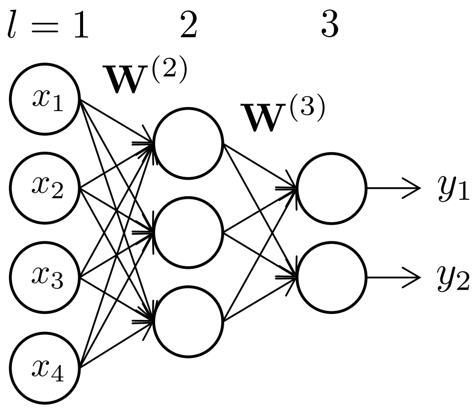
Computation of gradients wrt. weights

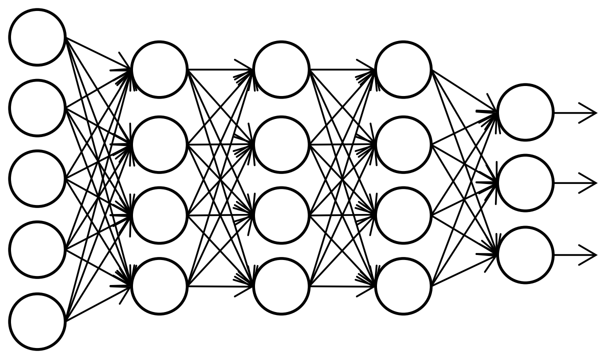
- Theoretically possible by applying the chain rule, but in practice almost intractable for nets with many layers

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = (\mathbf{y}(\mathbf{x}_n) - \mathbf{d}_n)^\top \frac{\partial \mathbf{y}}{\partial w_{ji}^{(l)}}$$

$$\left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \right)$$

$$\begin{aligned}
\mathbf{y}(\mathbf{x}) &= \mathbf{f}(\mathbf{u}^{(L)}) \\
&= \mathbf{f}(\mathbf{W}^{(L)}\mathbf{z}^{(L-1)} + \mathbf{b}^{(L)}) \\
&= \mathbf{f}(\mathbf{W}^{(L)}\mathbf{f}(\mathbf{W}^{(L-1)}\mathbf{z}^{(L-2)} + \mathbf{b}^{(L-1)}) + \mathbf{b}^{L}) \\
&= \mathbf{f}(\mathbf{W}^{(L)}\mathbf{f}(\mathbf{W}^{(L-1)}\mathbf{f}(\cdots\mathbf{f}(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)})\cdots)) + \mathbf{b}^{(L)})
\end{aligned}$$



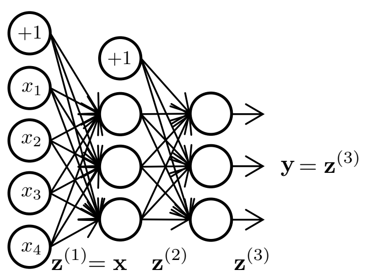


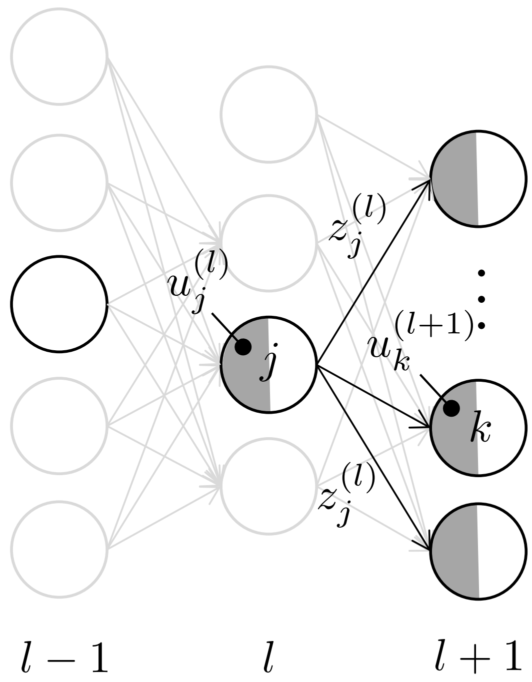
The backpropagation algorithm

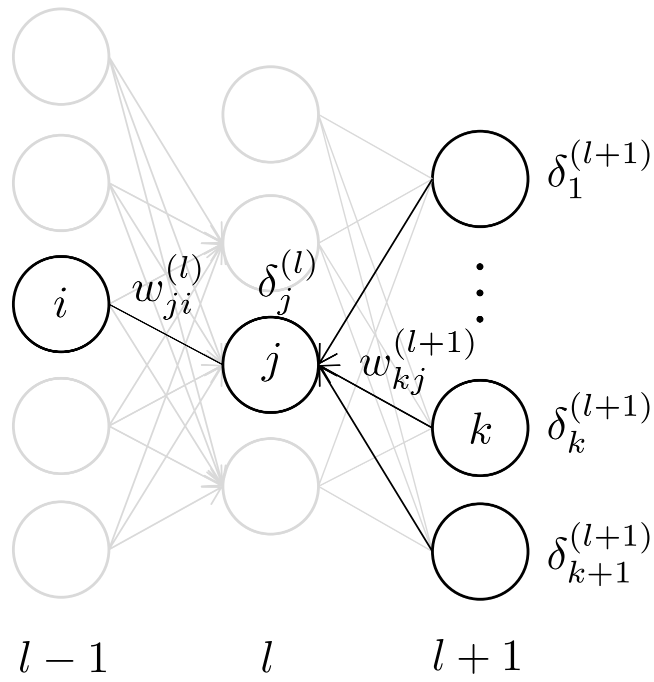
- Gradient at a weight at a a layer can be calculated as:

Define delta: $\delta_j^{(l)} \equiv \frac{\partial E_n}{\partial u_j^{(l)}}$ $\Rightarrow$ Gradient: $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$

Delta's at each layer can be back-propagated: $$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \left(w_{kj}^{(l+1)} f'(u_j^{(l)}) \right)$$







Deltas at output layers

- Regression: identity activation func. & squared loss

$$E_n = \frac{1}{2}\|\mathbf{y}-\mathbf{d}\|^2 = \frac{1}{2}\sum_j (y_j - d_j)^2 \qquad \left(y_j = z_j^{(L)} = u_j^{(L)} \right)$$

$$\delta_j^{(L)} = \frac{\partial E_n}{\partial u_j^{(L)}} = u_j^{(L)} - d_j = z_j^{(L)} - d_j = y_j - d_j$$

- Classification: softmax activation func & cross-entropy loss

$$E_n = -\sum_k d_k \log y_k = -\sum_k d_k \log\left(\frac{\exp(u_k^{(L)})}{\sum_i \exp(u_i^{(L)})}\right)$$

$$\delta_j^{(L)} = -\sum_k d_k \frac{1}{y_k}\frac{\partial y_k}{\partial u_j^{(L)}} \qquad \left(d_k^2 = d_k \quad d_k d_j = 0\ (k \neq j) \quad \sum_k d_k = 1 \right)$$

$$= -d_j(1-y_j) - \sum_{k\neq j} d_k(-y_j) = \sum_k d_k(y_j - d_j) = y_j - d_j$$

$$(f/g)' = (f'g - fg')/g^2$$

Derivatives of activation functions

Table

Activation func.	$f(u)$	$f'(u)$
Logistic	$f(u) = 1/(1+e^{-u})$	$f'(u) = f(u)(1-f(u))$
Hyperbolic tan	$f(u) = \tanh(u)$	$f'(u) = 1 - \tanh^2(u)$
ReLU	$f(u) = \max(u, 0)$	$f'(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases}$