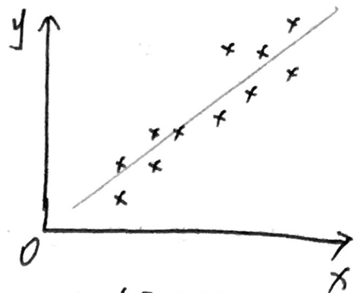
▷ Linear Regression on $\mathbb{R}^+$

- problem:

$$\mathcal{D} = \{(x_i, y_i) \mid i=1,\cdots,N,\ x_i, y_i \in \mathbb{R}^+\} \text{ is the ground truth data.}$$



we want to use linear function, more specifically, Rectified linear function, to approximate the Data. That is,

$$\hat{y} = \max(wx+b, 0) = \varphi(f(x))$$

Here $\varphi = \max(x, 0)$ is called Rectified Linear Unit (ReLU)

$f = wx+b$ is the linear function

To solve this problem, a "best" w and b are needed to be found. Rewrite this problem, we have

$$(w^*, b^*) = \arg\min_{w,b}.\ L(w,b)$$

where $L(w,b)$ is a measure of how "bad" w,b are, a simple choice of which is mean squared error, i.e.,

$$L(w,b) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i(w,b))^2$$

Note:

- In process of finding optimal w and b, (x_i, y_i) is known, hence $\hat{y}_i = \hat{y}_i(w,b)$.
- In process of using founded w^* and b^* to predict y, (w^*, b^*) ~~and x~~ is known, hence $\hat{y} = \hat{y}(x)$

- Optimization with Gradient Descent with momentum

Init: $v_w, v_b = 0$.
choose θ, γ

Iter:

$$\left| \begin{aligned} v_w^+ &= (1-\theta)\frac{\partial L}{\partial w} + \theta v_w \\ w^+ &= w - \gamma \cdot v_w^+ \end{aligned} \right.$$

$$\left| \begin{aligned} v_b^+ &= (1-\theta)\frac{\partial L}{\partial b} + \theta v_b \\ b^+ &= b - \gamma \cdot v_b^+ \end{aligned} \right.$$

- Calculation of $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$

recall that

$$L(w,b) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i(w,b)\right)^2 \qquad \text{let } \tilde{L} = \left(y_i - \hat{y}_i(w,b)\right)^2.$$

$$= \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \varphi_1(f_i(w,b))\right)^2$$

By chain rule,

$$(*) \begin{cases} \dfrac{\partial L}{\partial w} = \dfrac{1}{2N}\displaystyle\sum_{i=1}^{N} \dfrac{\partial \tilde{L}(w,b)}{\partial \varphi_1(f_i(w,b))} \cdot \dfrac{\partial \varphi_1(f_i(w,b))}{\partial f_i(w,b)} \cdot \dfrac{\partial f_i(w,b)}{\partial w} = \dfrac{1}{2N}\displaystyle\sum_{i=1}^{N} \dfrac{\partial \tilde{L}}{\partial \varphi_1} \cdot \dfrac{\partial \varphi}{\partial f_i} \cdot \dfrac{\partial f_i}{\partial w} \\[2ex] \dfrac{\partial L}{\partial b} = \dfrac{1}{2N}\displaystyle\sum_{i=1}^{N} \dfrac{\partial \tilde{L}(w,b)}{\partial \varphi_1(f_i(w,b))} \cdot \dfrac{\partial \varphi_1(f_i(w,b))}{\partial f_i(w,b)} \cdot \dfrac{\partial f_i(w,b)}{\partial w} = \dfrac{1}{2N}\displaystyle\sum_{i=1}^{N} \dfrac{\partial \tilde{L}}{\partial \varphi_1} \cdot \dfrac{\partial \varphi}{\partial f_i} \cdot \dfrac{\partial f_i}{\partial b} \end{cases}$$

where

$$\frac{\partial \tilde{L}}{\partial \varphi_1} = 2\left(\varphi_1(f_i(w,b)) - y_i\right) \qquad \frac{\partial f_i}{\partial w} = x_i$$

$$\frac{\partial \varphi_1}{\partial f_i} = \begin{cases} 1, & f_i(w,b) > 0 \\ 0, & f_i(w,b) < 0 \end{cases} \qquad \frac{\partial f_i}{\partial b} = 1$$

so $(*)$ becomes

$$\frac{\partial L}{\partial w} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} x_i(wx_i+b), & wx_i+b > 0 \\[2ex] 0, & wx_i+b < 0 \end{cases}$$

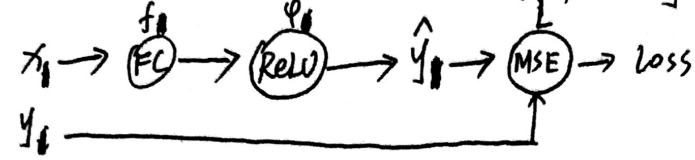
$$\frac{\partial L}{\partial b} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} (wx_i+b), & wx_i+b > 0 \\[2ex] 0, & wx_i+b < 0 \end{cases}$$

- Motivation: modulize the calculation of $\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$
  (Engineering-oriented)

As we can see from (*), $\frac{\partial L}{\partial \varphi}, \frac{\partial \varphi}{\partial f_i}$ are used repeatly. Hence things would be easier if we seperate the calculation of $\frac{\partial L}{\partial \varphi}, \frac{\partial \varphi}{\partial f_i}, \frac{\partial f_i}{\partial w}$ and $\frac{\partial f_i}{\partial b}$, although not so obviously in this simplest example.

To achieve this, we rewrite $L(w,b)$ graphically.

net:



Here $x, y, \hat{y} \in \mathbb{R}^{1\times N}$

Roughly we define such a graph as a (training) network. (1) To compute $L(w,b)$, we call net.forward(x, y), which in this case is as following

```
# MATLAB
function  loss = forward(net, x, y)
    ~~Loss = L.forwardLoss(~~
    ŷ = φ.forward(f.forward(x))
    loss = L.forwardLoss(ŷ, y)
end
```

(2) To update parameters w, b, we call net.update_params(γ, θ, y, ŷ)

```
# MATLAB
function  net = update_params(net, γ, θ, y, ŷ)
    ∂L/∂φ = L.backwardLoss(y, ŷ)                      y, ŷ, ∂L/∂φ ∈ IR^{1×N}
    ∂L/∂f = φ.backward(f(w,b), ∂L/∂φ)                , f(w,b) ∈ IR^{1×N}
    [∂L/∂x, ∂L/∂w, ∂L/∂b] = f.backward(x, ∂L/∂f)     , x ∈ IR^{1×N}
    f = f.update_params(γ, θ, ∂L/∂w, ∂L/∂b)
end
```

By doing this, we could seperately define the forward and backward of each "layer", which is the circled ~~Pat~~ part in the network: ~~FC~~, fully connected Layer (FC), rectifed linear Unit layer (ReLU), and Mean squared error (MSE).

To make this simple, we give the one-variable version, which is simply a rewritten version of our previous formulas in an engineering way. Take FC layer as an example, ReLU and MSE are similar.

```
% FC Layer
classdef FC < handle
    properties
        w
        b
    end
    properties (private)
        v_w
        v_b
    end
    methods
        function layer = FC()
            layer.v_w = 0
            layer.v_b = 0
            layer.w = randn()
            layer.b = randn()
        end
        function z = forward(layer, x)
            z = layer.w · x + layer.b          , x ∈ IR^{1xN}
        end
        function [∂L/∂x, ∂L/∂w, ∂L/∂b] = backward(layer, x, ∂L/∂z)
            ∂L/∂x = layer.w · x                , x ∈ IR^{1xN}
            ∂L/∂w = Σ (∂L/∂z)^T · x            , ∂L/∂w ∈ IR
            ∂L/∂b = Σ ∂L/∂z                    , ∂L/∂b ∈ IR
        end
        function layer = update_params(layer, γ, θ, ∂L/∂w, ∂L/∂b)
            layer.v_w = layer.v_w · θ + ∂L/∂w · (1-θ)
            layer.w = layer.w - γ · layer.v_w
            layer.v_b = layer.v_b · θ + ∂L/∂b · (1-θ)
            layer.b = layer.b - γ · layer.v_b
        end
    end
end
```

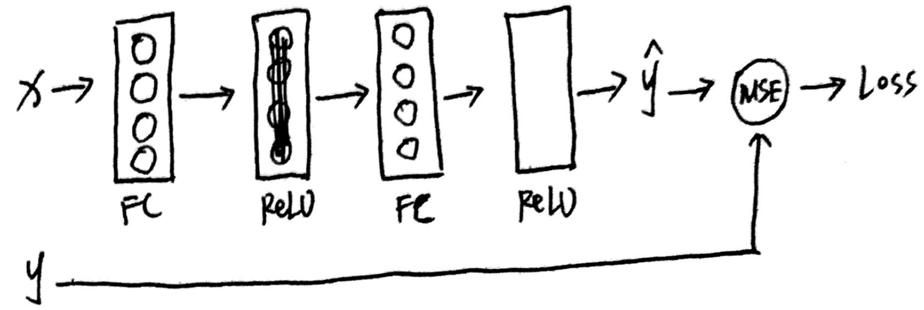
- Going deeper and wider

The network used in our example is width 1, depth 2 network.

$$x \rightarrow \text{FC} \rightarrow \text{ReLU} \rightarrow \hat{y} \rightarrow \text{MSE} \rightarrow \text{loss}$$
$$y \rightarrow \text{MSE}$$

we could make this ~~more~~ deeper and wider with previous work of modulization.

network:



The corresponding formular of this network is

$$y = \varphi(f_2(\varphi(f_1(x)))) \quad , x, y \in \mathbb{R}^{n\times 1}$$
$$f_i \triangleq w_i^T x + b \quad , w_i \in \mathbb{R}^{n\times 1}, b \in \mathbb{R} \quad , i=1,2.$$
$$\varphi(x) = \max(x, 0) \quad \text{pointwise max.}$$
$$L(w_1, b_1, w_2, b_2) = \frac{1}{2N} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_2^2$$

The forward ~~and~~ param_update and backward of the network could be easilly done once we define the act of each type of layer.

- General framework

for any version of approximation by composition of simple functions.

$$y = \varphi_n(\varphi_{n-1}(\cdots(\varphi_2(\varphi_1(x)))))$$

the only thing we need to do then is define:

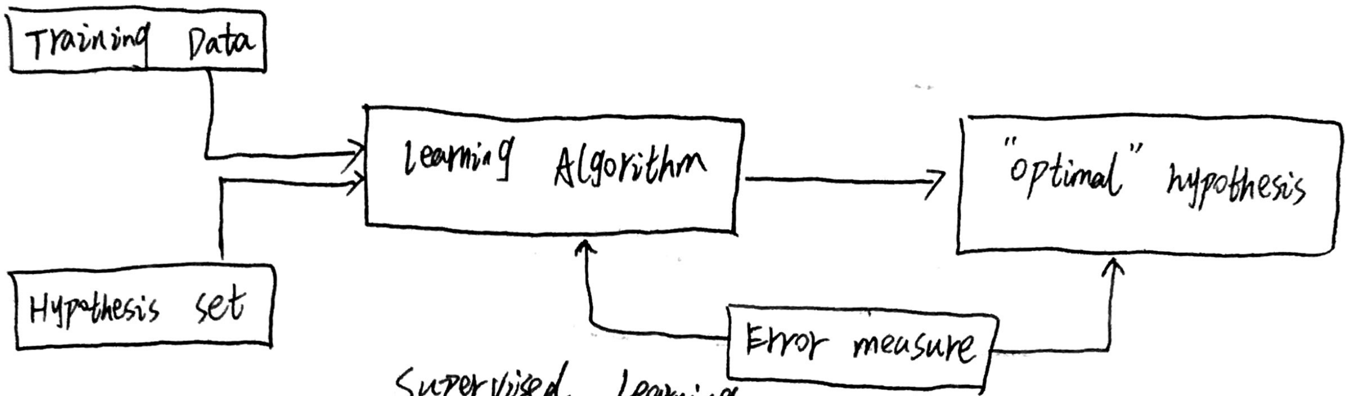
φ_i.backward

φ_i.forward

φ_i.update_param ⇝ if necessary

Currently, linear combination and convolution are two main ~~most~~ simple function that requires parameter updates, where convolution layer is very suitable for image and computer vision tasks.

· ~~Con~~ Summary



Supervised Learning

— 陆吾生 "A primer of Machine Learning"

Building blocks of Deep learning:

- Training data
- Validation data: not used in training, to test the performance
- Hypothesis set: the network
- Loss function: describe how "bad" the network performs
- Optimizer: learning Algorithm