Natural Language Processing from Scratch

Essentials

Yonghui Wu

HOBI — COM — UF

September 29, 2022

Preface

This is a series of training on deep learning in natural language processing (NLP). You will learn the mathematical theories of machine learning and optimization, design of neural network architectures, and **most importantly**, implementation of them in program languages. This training will cover mainstream deep learning models for NLP from 2011 to 2016, including convolutional neural network, recurrent neural network (LSTM), and transformers (e.g., BERT).

Prerequisite:

1. Mathematics, including function, convex function; calculus, derivatives, partial derivatives
2. Machine learning classifiers, optimization theory
3. Programming skills such as C++, Java, Python, TensorFlow, Pytorch.

Table of contents

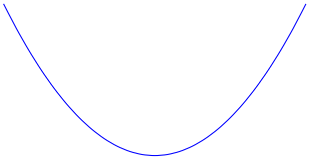
Function

Function

A function $f : X \to Y$ is a mapping from a set X to a set Y

Convex Function

- The line segment between any two points on the graph of the function lies above the graph between the two points.
- Has no more than one maximum/minimum



$$y = x^2$$

Calculus

Definition

Derivative

- The derivative of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value).
- $\dfrac{df(x)}{dx}$ or $f'(x)$, $f''(x)$
- Partial firt order derivative: $\frac{\partial f(x)}{\partial x}$
- $\dfrac{de^x}{dx} = e^x$, $\dfrac{dln(x)}{dx} = \frac{1}{x}$

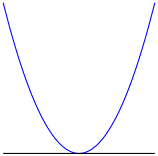
Find maxima/minima using derivatives

Definition

The slope is 0 at maxima/minima of a function.

Let $f(x) = x^2$, $f(x)$ is at maxima/minima when:

$$\frac{df(x)}{dx} = 2x = 0 \rightarrow x = 0$$



$y = x^2$

$\arg\max, \arg\min$

Definition ($\arg\max$)

Arguments of the maxima: $\arg\max_x f(x) = \{x : f(s) \geq f(x)\}$, for all $s \in X$

Definition ($\arg\min$)

Arguments of the minima: $\arg\min_x f(x) = \{x : f(s) \leq f(x)\}$, for all $s \in X$

Log-likelihood function

Definition (Likelyhood)

The likelihood function $\mathcal{L}(\theta)$ (often simply called the likelihood) is the joint probability of the observed data (D) viewed as a function of the parameters θ of the chosen model.

Likelihood function over the parameter space on dataset with N samples $(x_i, y_i) \in D$:

$$\prod_{i=1}^{N} \mathcal{L}(\theta : (x_i, y_i))$$

Find θ that maximize the Log-likelyhood:

$$\arg\max_{\theta} \ln(\prod_{i=1}^{N} \mathcal{L}(\theta : (x_i, y_i)))$$

What is machine learning?

Machine Learning

1. Construct programs that automatically **improve** with experiences. [1]
2. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E. [1]

[1] Tom M. Mitchell

Machine learning and mathmatics

- Machine learning models are derived from mathmatic algorithms
- Machine learning has real-world constrains
  - Trainable - get model trained in reasonable time
  - Computer use many approximation algroithms to reduce computing cost
  - Constraints on hardware - over flow, under flow
  - Numbers in machines are not exactly the real value, e.g., floating-point numbers have 16 bit accuracy or 32 bit
  - for 64-bit float point number, Max$=1.7976931348623157e+308$, Min$=2.2250738585072014e-308$
  - **Overflow:** Float point number $>$ Max
  - **Underflow:** Float point number $<$ Min

Mathmatical setting for classification

Definition (Training Data)

A set of $(x^m, y^m), m = \{1, 2, \ldots, M\}$, *where* $x^m \in R^d$ *(the input data) and output class* $y^m = 0, 1$

Definition (Learning goal)

Learn function $f : x \to y$ *to predict correctly on new input* x, *with respect to parameter* θ.

Definition (Optimize a loss function)

Minimize loss function (least square): $Loss = \sum_{m=1}^{M}(f_w(W^T x^m) - y^m)^2$

The goal of training is to find $\arg\min_\theta Loss$

Traning, develop, test datasets

Reasonable data splition settings

- N-fold cross validation on a training dataset
- A traning set and a test set
- A training, a development, and a test dataset.

The BIG NO!

Never use any test data in training.

Train machine learning models use gradient descent - GD

General procedures to train machine learning models:

Definition

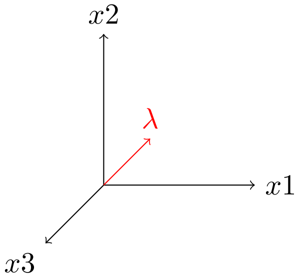
W : parameters in a machine learning model; D: data set; λ : learning rate

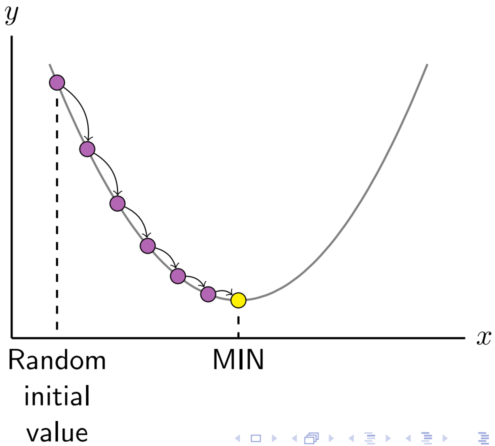
1. Initilize W with random real numbers
2. Compute the gradients for parameters: $\nabla_w Loss$
3. Update $W \leftarrow W - \lambda(\nabla_W Loss)$, λ is the learning rate
4. Loop 2 and 3, until stop creteria satisfied

To train parameter W^k, need an algorithm to start from random values, calculate graidents, derive new parameters W^{k+1} for the next iteration.

Why learning rate?

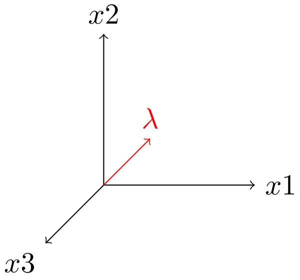
The gradients are a vector determining the descending directions, learning rate is to control 'how far' the descending goes.

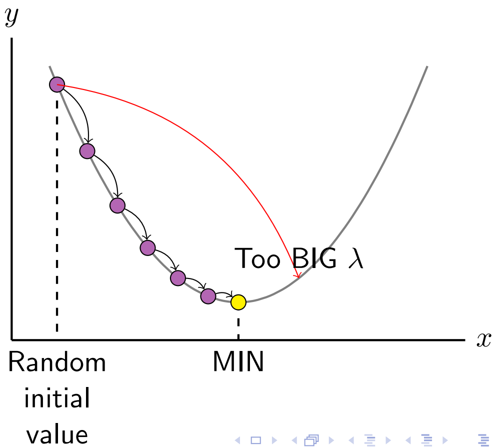




Why learning rate?

The gradients are a vector determining the descending directions, learning rate is to control 'how far' the descending goes.

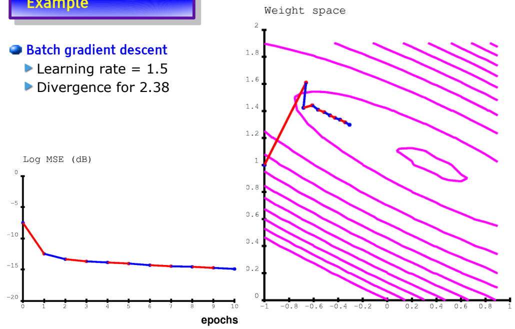




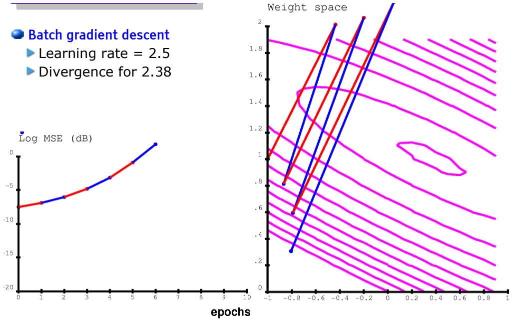
Learning rate example

Example

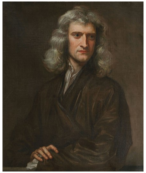
- **Batch gradient descent**
  - Learning rate = 1.5
  - Divergence for 2.38



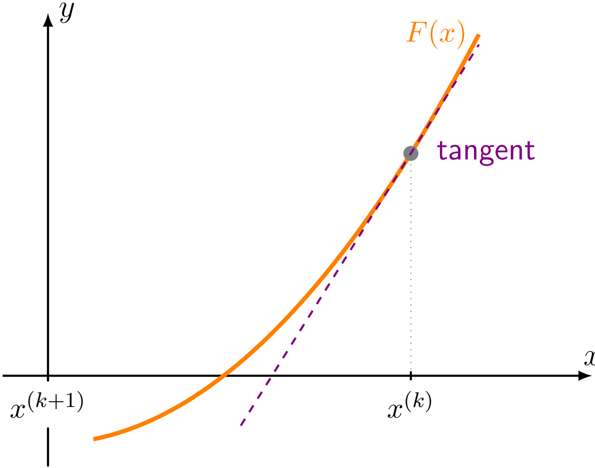
- **Batch gradient descent**
  - Learning rate = 2.5
  - Divergence for 2.38



Newton's solution to optimize convex Loss function?



- Newton method is a iterative algorithm to find the point x^*, where $f(x^*) = 0$.
- We use Newton's approximation on $f'(x)$, i.e., the local or global minimal point, $f'(x) = 0$. $f(x)$ is the LOSS function.



Derive iterative equation from Newton's method I

Definition (Taylor series of a function)

- An infinite sum of terms that are expressed in terms of the function's derivatives at a single point x.
- $f(a) = \sum_{n=0}^{n=\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$

The Taylor series of Loss function $f(x)$ at point a can be approximated as:

$$f(x=a) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 \tag{1}$$

Derive iterative equation from Newton's method II

To find the minumum:

$$\frac{f(x)}{da} = -f'(a) - f''(a)(x-a) = 0$$
$$\Rightarrow x = a - \frac{f'(a)}{f''(a)}$$
$$\Rightarrow x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

When x is a vector, denoted as $\boldsymbol{x}$, the Taylor expression becomes:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x^k}) + f'(\boldsymbol{x^k})(\boldsymbol{x} - \boldsymbol{x^k}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x^k})\boldsymbol{H}(\boldsymbol{x^k})(\boldsymbol{x} - \boldsymbol{x^k})$$

Derive iterative equation from Newton's method III

Where $\boldsymbol{H}(\boldsymbol{x}^{\boldsymbol{k}})$ is the Hassian matrix around $\boldsymbol{x}^{\boldsymbol{k}}$. Let $\boldsymbol{g}^{\boldsymbol{k}} = \boldsymbol{f}'(\boldsymbol{x}^{\boldsymbol{k}}), \boldsymbol{H}^{\boldsymbol{k}} = \boldsymbol{H}(\boldsymbol{x}^{\boldsymbol{k}})$, we have:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{\boldsymbol{k}}) + \boldsymbol{g}^{\boldsymbol{k}}(\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{k}}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{k}})\boldsymbol{H}^{\boldsymbol{k}}(\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{k}}) \tag{2}$$

Then, by let $f'(\boldsymbol{x}) = 0$, we have:

$$\boldsymbol{x}^{\boldsymbol{k+1}} = \boldsymbol{x}^{\boldsymbol{k}} - (\boldsymbol{H}^{\boldsymbol{k}})^{\boldsymbol{-1}}\boldsymbol{g}^{\boldsymbol{k}}$$

Approximation to Newton's Approximation method

Barriers to apply classic Newton in machine learning:

- Computational cost
- Only works for convex function
- $(\boldsymbol{H^k})^{-1}$ is not guranteed to be exist

Quasi Newton - find a approximate matrix $\boldsymbol{A^k}$ for the Hassian matrix $(\boldsymbol{H^K})^{-1}$. In another word, we are trying to find a approx function $\boldsymbol{F(x)}$:

$$\boldsymbol{f(x) \approx F(x) = f(x^k) + g^k(x - x^k) + \frac{1}{2}(x - x^k)(A^k)^{-1}(x - x^k)}$$

Improved newton algorithms:

- BFGS Quasi Newton
- L-BFGS: limited memory BFGS

Train non-deep learning models using GD

Definition

W : parameters in a machine learning model; D: data set; λ : learning rate

1. Initilize W with random real numbers
2. Compute the gradients : $\nabla_w Loss$ using **ALL samples** in D
3. Update $W \leftarrow W - \lambda(\nabla_W Loss)$, λ is the learning rate
4. Loop 2 and 3, until $f'(x) < \epsilon$ or maximum iteration exhausted.

The gradient was caculated using all samples in D; the stop creteria is $f'(x) < \epsilon$ or maximum iteration exhausted. No development data required in training.

Stochastic Gradient Decent (SGD)

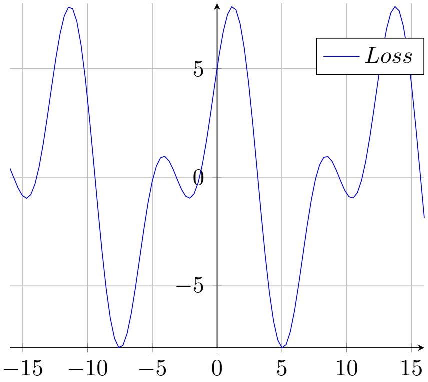
Definition

W : parameters in a machine learning model; D: data set; λ : learning rate

1. Initilize W with random real numbers
2. Compute the gradients : $\nabla_w Loss$ using **mini-batch(1 - N)** of D
3. Update $W \leftarrow W - \lambda(\nabla_W Loss)$, λ is the learning rate
4. Calculate perforance using a **validation set**, dump the model if better than prevous
5. Loop 2 - 5, stop if no improvement (e.g., in in 5 steps, early stop) or maximum iteration exhausted.

Why need validation set in deep learning?

- Deep learning models are not convex functions with many local maximums.
- Need to compare all local maximum to pick up the best one according to the validation performance.



How to correctly interpret training LOSS?

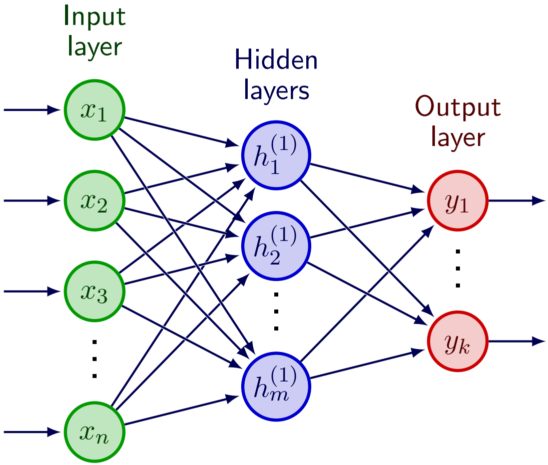
Use it, but don't trust it.

- The loss (on training) we are minimizing is not the loss in real test
- If optimize too well on training, we overfit.

SGD vs GD

- SGD is very fast at begining, very slow when approaching a minimum
- GD is very low in the begining, very fast when approaching a minimum
- The LOSS on test set saturates long before the training set

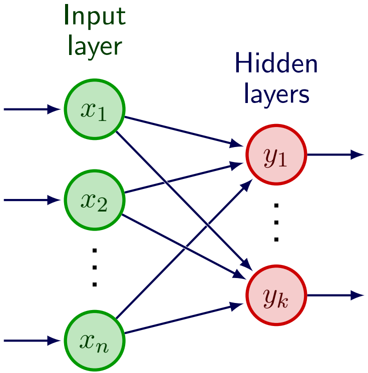
Neural networks



Neural networks are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs and produces a single real-valued output. [1]

[1] Tom M. Mitchell

Linear layer



$$y_1 = w_{11} * x_1 + w_{12} * x_2 + \cdots + w_{kn} * x_n$$

Let:

$\vec{X}^{-1} = [x_1, x_2, \ldots, x_n]$,

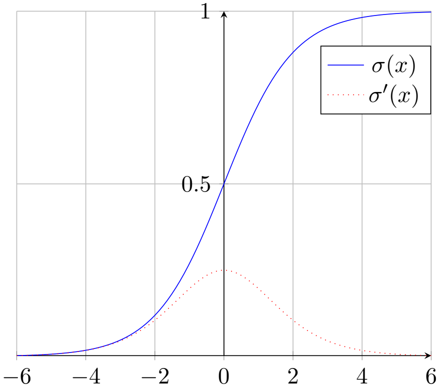
$\vec{Y}^{-1} = [y_1, \ldots, y_k]$

$$\vec{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{kn} \end{bmatrix}$$

$$\vec{Y} = \vec{W} \times \vec{X}$$

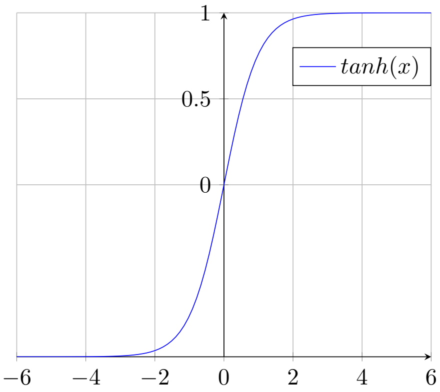
Non-linear layers/neurons I

Sigmoid: $\sigma(x) = \frac{1.0}{1.0+e^{-x}}$



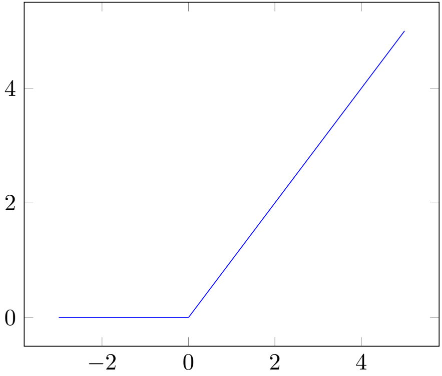
Non-linear layers/neurons II

Tanh : $tanh(x)$



Non-linear layers/neurons III

Rectified linear unit (ReLU)



How to choose neurons?

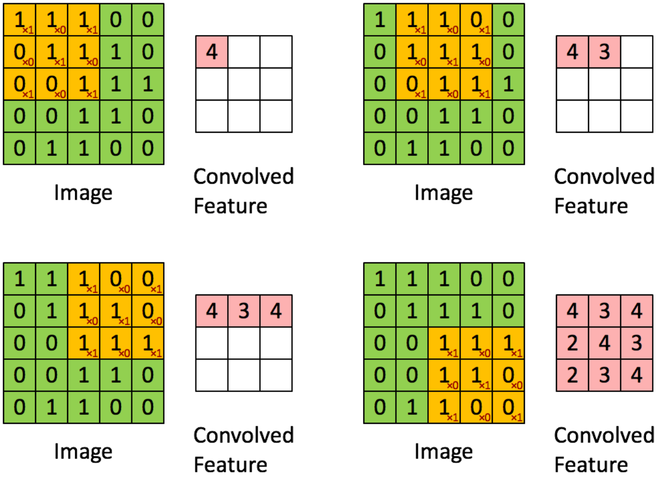
- The more non-linearity the better
- Easy to calculate derivatives
- Fast to calculate derivatives

Intuition: "If you are confident about the direction, go faster (large gradient), other wise, go slower.

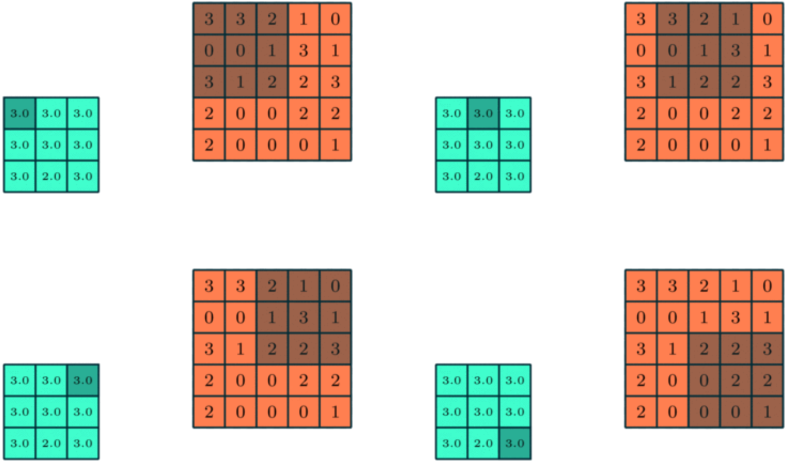
Convolutional architecture

- Convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN).
- CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN)
- CNNs are good to capture the Spatial and Temporal dependencies
- Widely used in computer image/vision, NLP.

Reduce image dimension use convolution layer



Pooling layer: Max pooling, average pooling



Convolution on text

pt developed an agonal respiration and was declared dead .

Convolution on text

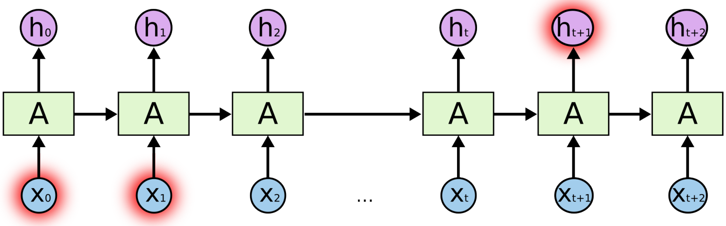
pt **developed** **an** **agonal** respiration and was declared dead .

Convolution on text

pt developed an agonal respiration and was declared dead .

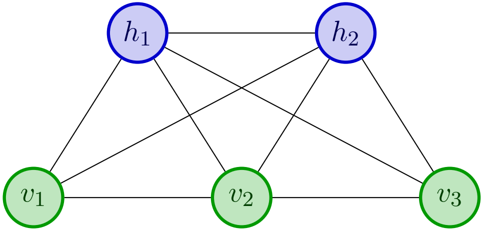
Recurrent neural network

- Unique architecture with loop
- $h_{t+1} = f(O_t : X_{t+1})$



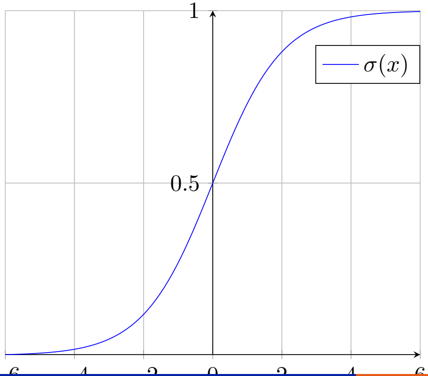
Boltzman machine

Fully connected graph.



A 1-layer neural network

- Parameters: w, b
- non-linear function: $\sigma(x) = \frac{1}{1+exp(-x)}$
- $f(x) = \sigma(w \cdot x + b)$



let y^* be the true value, Loss function be the squared error (this is avariation of Mean Squared Error [MSE]):

$$Loss = \frac{1}{2}\sum(f(x) - y^*)^2$$

Let $z = wx + b$, the derivative for $\sigma(x)$ is:

$$\sigma'(z) = \frac{d}{dz}\left(\frac{1}{1+exp(-z)}\right) = \sigma(z)(1-\sigma(z))$$

Calculate gradient for w

$$\frac{\partial(Loss)}{\partial w} = \frac{\partial(\frac{1}{2}\sum(f(x)-y)^2)}{\partial w}$$
$$= \sum(f(x)-y)\cdot\sigma'(z)\cdot\frac{\partial(z)}{\partial w}$$

To optimize W using SGD, for a specific data point (x^*, y^*)

$$\nabla_w = (f(x^*)-y^*)\cdot\sigma'(wx^*+b)\cdot\frac{\partial(wx^*+b)}{\partial w}$$
$$= (f(x^*)-y^*)\cdot\sigma'(wx^*+b)\cdot x^* \qquad //\text{treat } x \text{ as a constant}$$

Train the 1-layer neural network using SGD

1. Initilize W with random real numbers
2. Pick up a sample (x^*, y^*) and perform forward propagation to calculate $Loss = \frac{1}{2}\sum(f(x^*) - y^*)^2$
3. Back propagation to compute gradients for w:
   $(f(x^*) - y^*) \cdot \sigma'(wx^* + b) \cdot x^*$ //treat x as a constant
4. Update $w \leftarrow w - \lambda(\nabla_w)$, λ is the learning rate
5. Calculate perforance using a **validation set**, dump the model if better than prevous
6. Loop 2 - 5, stop if no improvement in 5 steps (early stop) or maximum iteration exhausted.

General training procedure for neural networks

- **Forward propagation** to calcualte Loss. In forward propagation, treat model parameters as constants, inputs x_i as variables.
- **Back propagation** to calculate the graident. In back propagation, treat model parameter as variables, inputs x_i as constants, in partial deriatives calculation.

What does "train" means?

"Train" means to identify the best combination of model parameters from a limited set of parameter combinations.
For deep learning models, you can train:

- Learning rate
- Epoch - when the training exhausted all samples in the training, it hit 1 epoch.
- Steps to validate/dump the model, max steps for early stop
- Batch size: how many samples used to calcualte the gradient

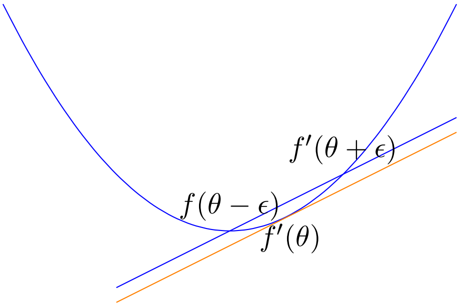
From Neural network to deep learning

- Neural network was proposed around 1970s.
- Neutal network became known as "deep learning" around 2005.
- Why no "deep learning" in 1970s?
    1. Hardware limiation - can't put too many layers in computer
    2. CPU limiation - Back propagation too slow
    3. Error valishing: the gradient scalculated in back propagation become smaller and smaller, which eventually too small to update the random initiated values

Deep learning vs neural networks

1. Deep learning models are neural networks
2. Not all neural networks are deep learning model
3. To be deep learning:
    - Layers > 3
    - Have layers for **high-level featrue learning**
    - Non-linearity

Gradient check in neural network implementation



Gradient checking is to check whethe the implementation of $f'(x)$ is correct w.r.t. the implementation of $f(x)$

$$\frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2*\epsilon}=f'(\theta)$$

Deep learning always better?