ECE284 Fall 21 W1S2

Low-power VLSI Implementation for Machine Learning

Prof. Mingu Kang

UCSD Computer Engineering

Announcement

1. If you just enrolled, please visit “Canvas – Pages – In-class Video” tab.

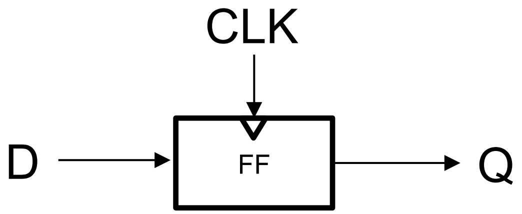
2. If you just enrolled, please complete three modules.

3. Enrollment issue. Please contact me. ece284ucsd@gmail.com

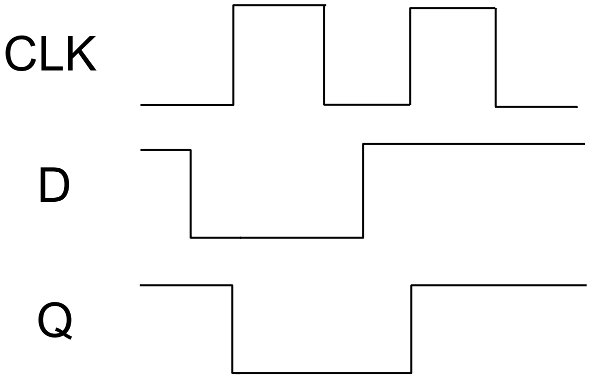
4. Linuxcloud & Datahub check.

5. Office hour change:
   - Instructor: Friday 11 – 11:50 am
   - TA: Friday 3 – 3:50 pm

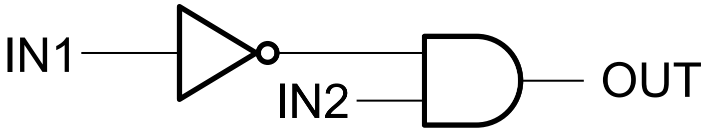
Verilog Review



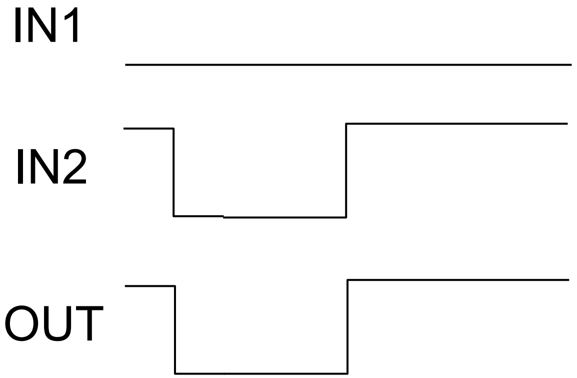
Sequential logic




```
reg Q;
reg D;

always @ (posedge clk) begin
    Q <= D;
end
```

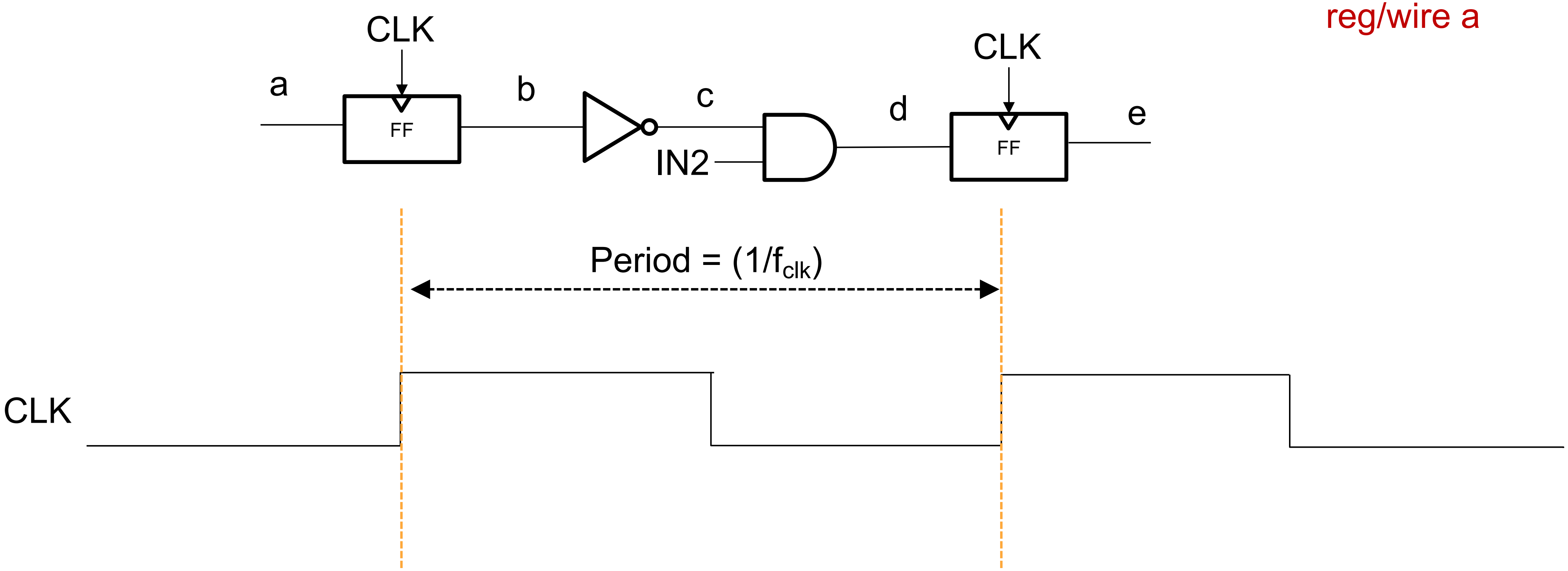


Combinational logic




```
wire IN1;
wire IN2;
wire OUT;

assign OUT = (!IN1) && (IN2);
```

Verilog Review



reg b, e
wire c, d
reg/wire a

Verilog Review: Frequently used patterns

```
module risc (clk, add, subt, a, b, out);

  input add, subt, clk;
  input [3:0] a;
  input [3:0] b;
  output [7:0] out;

  reg out_q;

  assign out = out_q;

  always @ (posedge clk) begin
    if (add)
      out_q <= a+b;
    else if (subt)
      out_q <= a-b;
  end

endmodule
```

```
module risc (clk, add, subt, a, b, out);

  input add, subt, clk;
  input [3:0] a;
  input [3:0] b;
  output reg [7:0] out;

  always @ (posedge clk) begin
    if (add)
      out <= a+b;
    else if (subt)
      out <= a-b;
  end

endmodule
```

Verilog Review: Frequently used patterns

```
module risc (clk, add, subt, a, b, out);

  input add, subt, clk;
  input [3:0] a;
  input [3:0] b;
  output reg [7:0] out;


  always @ (posedge clk) begin
    if (add)
      out <= a+b;
    else if (subt)
      out <= a-b;
  end

endmodule
```

```
module risc (clk, add, subt, a, b, out);

  input add, subt, clk;
  input [3:0] a;
  input [3:0] b;
  output reg [7:0] out;

  assign result = (add)? a+b : ((subt)? a-b : result);

  always @ (posedge clk) begin
    out <= result;
  end

endmodule
```

Verilog Review: Frequently used patterns

```
module risc (clk, inst, a, b, out);

  input inst;
  input clk;
  input [3:0] a;
  input [3:0] b;
  output reg [7:0] out;


  always @ (posedge clk) begin
    case (inst)
     1'b0: out <= a+b;
     1'b1: out <= a-b;
    endcase
  end

endmodule
```

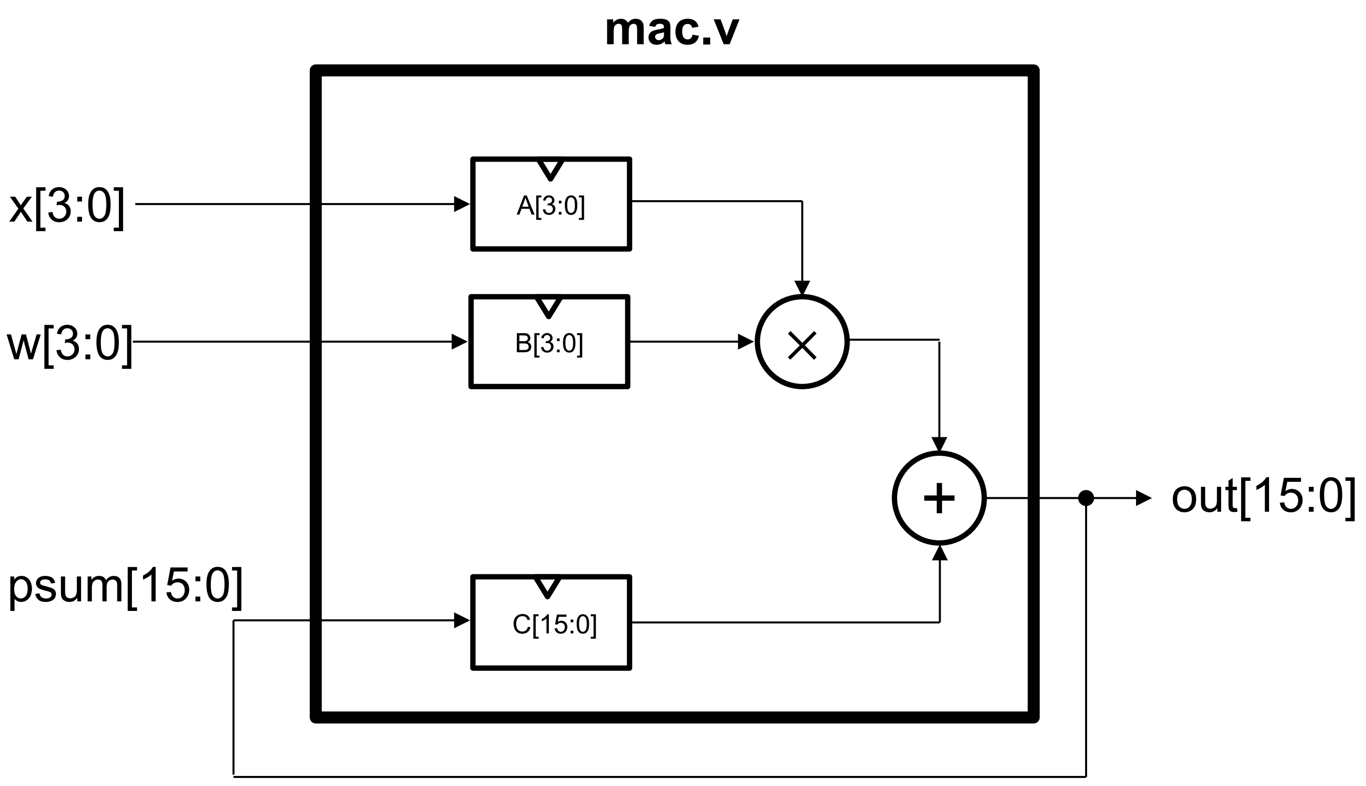
```
module risc (clk, inst, a, b, out);

  parameter bw = 4;

  input inst;
  input clk;
  input [bw-1:0] a;
  input [bw-1:0] b;
  output reg [2*bw-1:0] out;

  always @ (posedge clk) begin
   case (inst)
     1'b0: out <= a+b;
     1'b1: out <= a-b;
   endcase
  end

endmodule
```

Verilog Review: Frequently used patterns

```
module risc (clk, a, b, out);

    parameter bw = 4;

    input clk;
    input [2*bw-1:0] a;
    input [2*bw-1:0] b;
    output reg [2*bw-1:0] out;

    wire [2*bw-1:0] out_temp;
    genvar i;

    for (i = 0; i < 2; i=i+1) begin
      assign out_temp[ bw*(i+1)-1 : bw*i ] = a[bw*(i+1)-1 : bw*i ] * b[bw*(i+1)-1 : bw*i];
    end

    always @ (posedge clk)
      out <= out_temp;

endmodule
```

out[7:0]   = a[3:0]*b[3:0]
out[15:8] = a[7:4]*b[7:4]

[Verilog] MAC Example



Alias

- **iveri** = 'iverilog -o compiled -c'
- **irun** = 'vvp compiled'
- **wave** = 'gtkwave'

Example commands

1. iveri filelist
   (filelist includes all *.v files)
2. irun
3. wave mac_tb.vcd

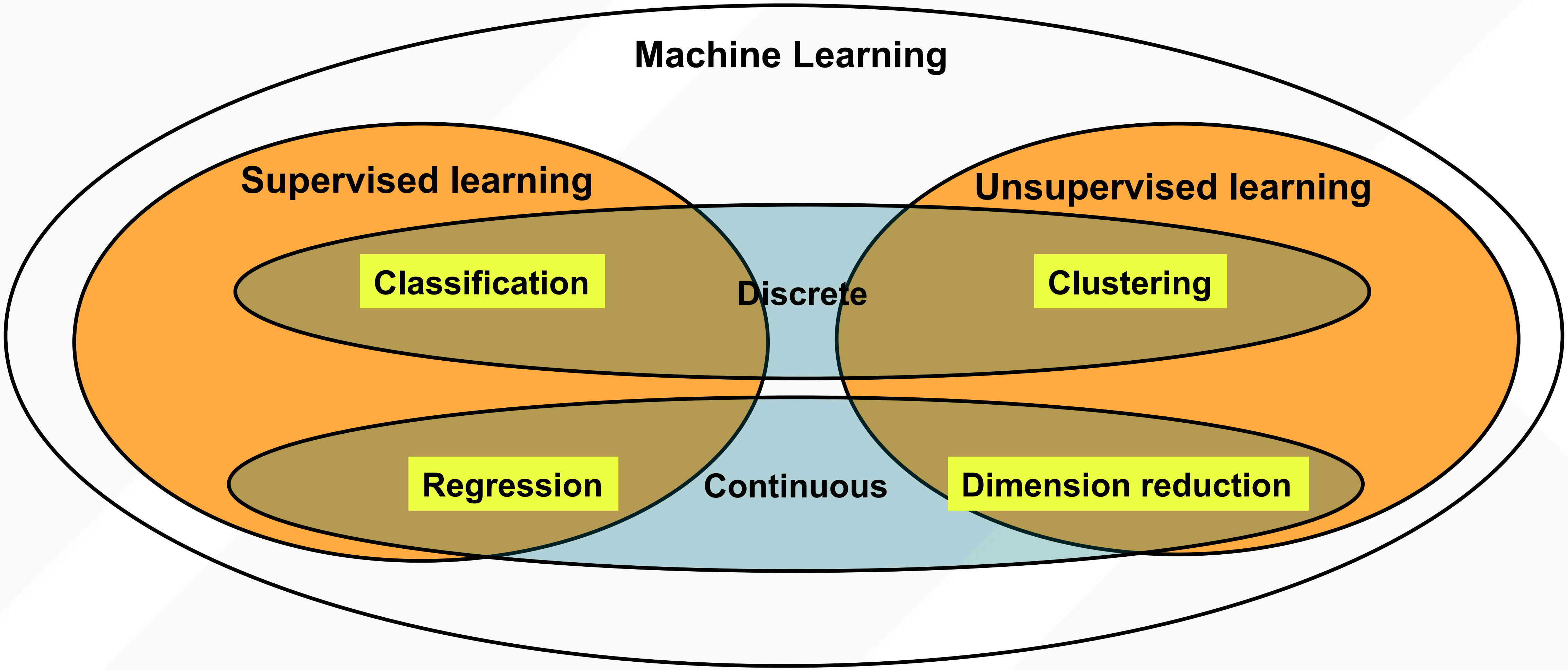
- Weight / Activation: 4-bit, psum / output: 16-bit
- Weight, activations, psum are latched, but output is not latched
- Weight and activation data are in b_data.txt and a_data.txt files
- Output is fed back to psum input

[HW1] SFP Design

- In: 4-bit, out: 16-bit
- Control bits: acc, relu, reset
- If reset, internal latch “psum_q” becomes zero
- If acc == 1, psum_q will be updated with “psum_q + In” in the next rising edge
- If relu == 1, psum_q is negative number, psum_q will be updated to be zero in the next rising edge
- out port is just connected to the psum_q

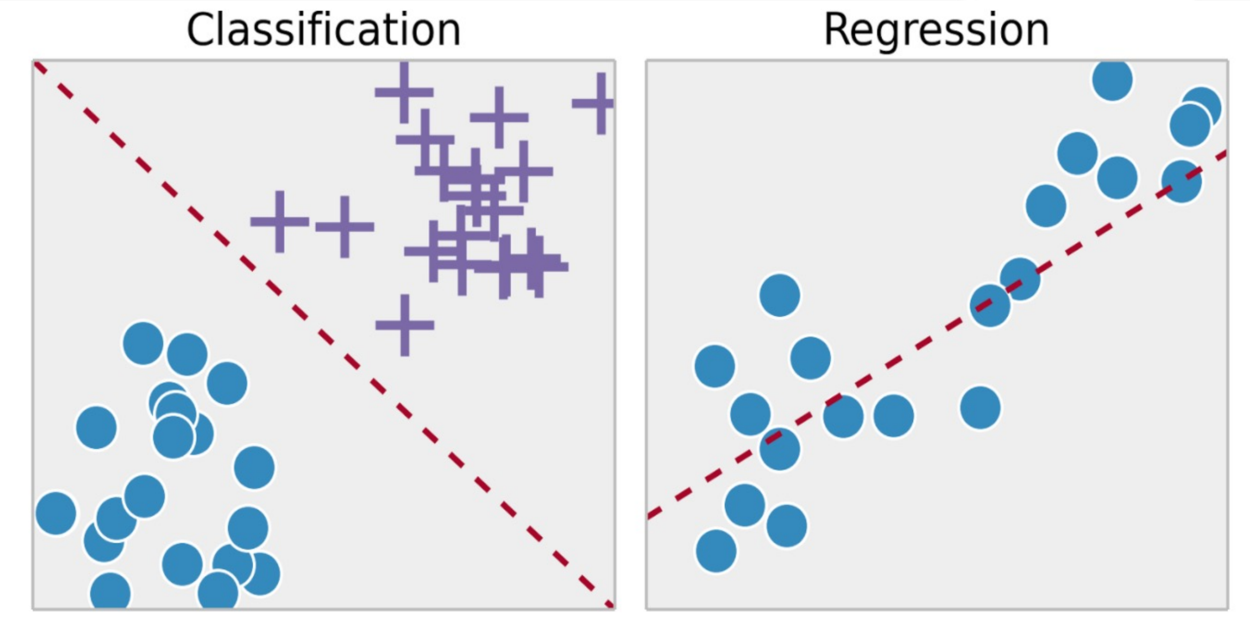
- Sample vcd file is attached in git.

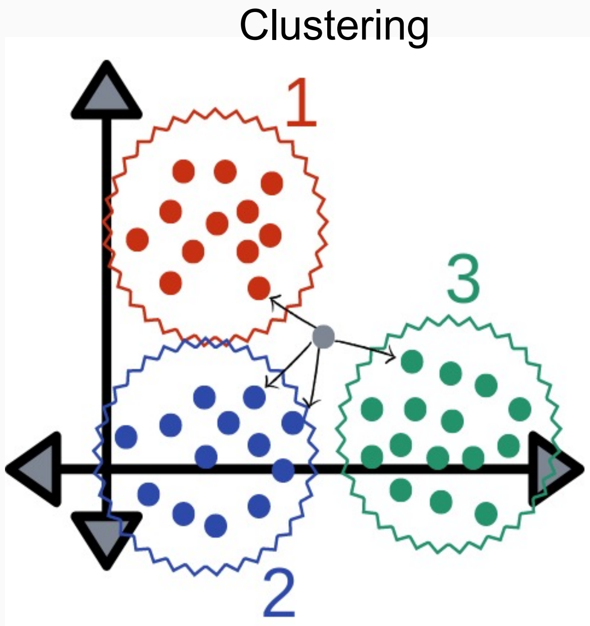
Types of Machine Learning

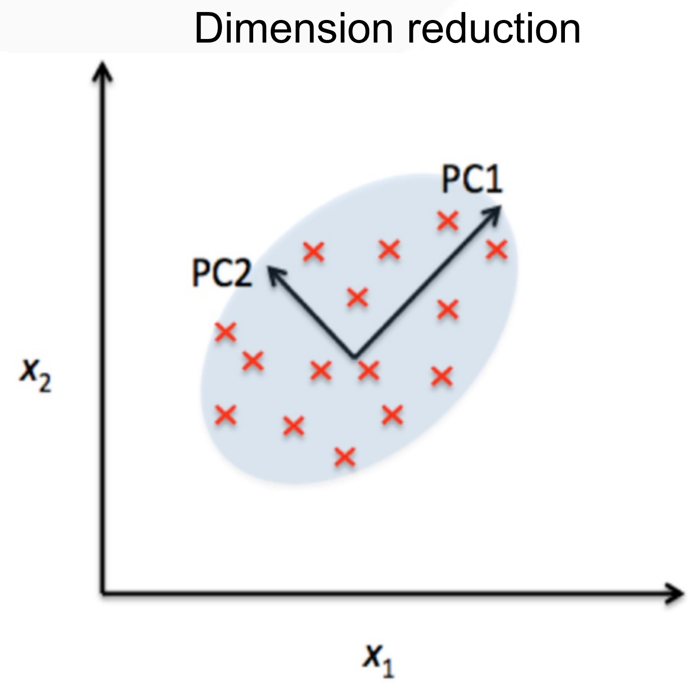


https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

Types of Machine Learning – cont.







- Neural network
- Support vector machine

…

- Linear regression
- Polynomial Regression

…

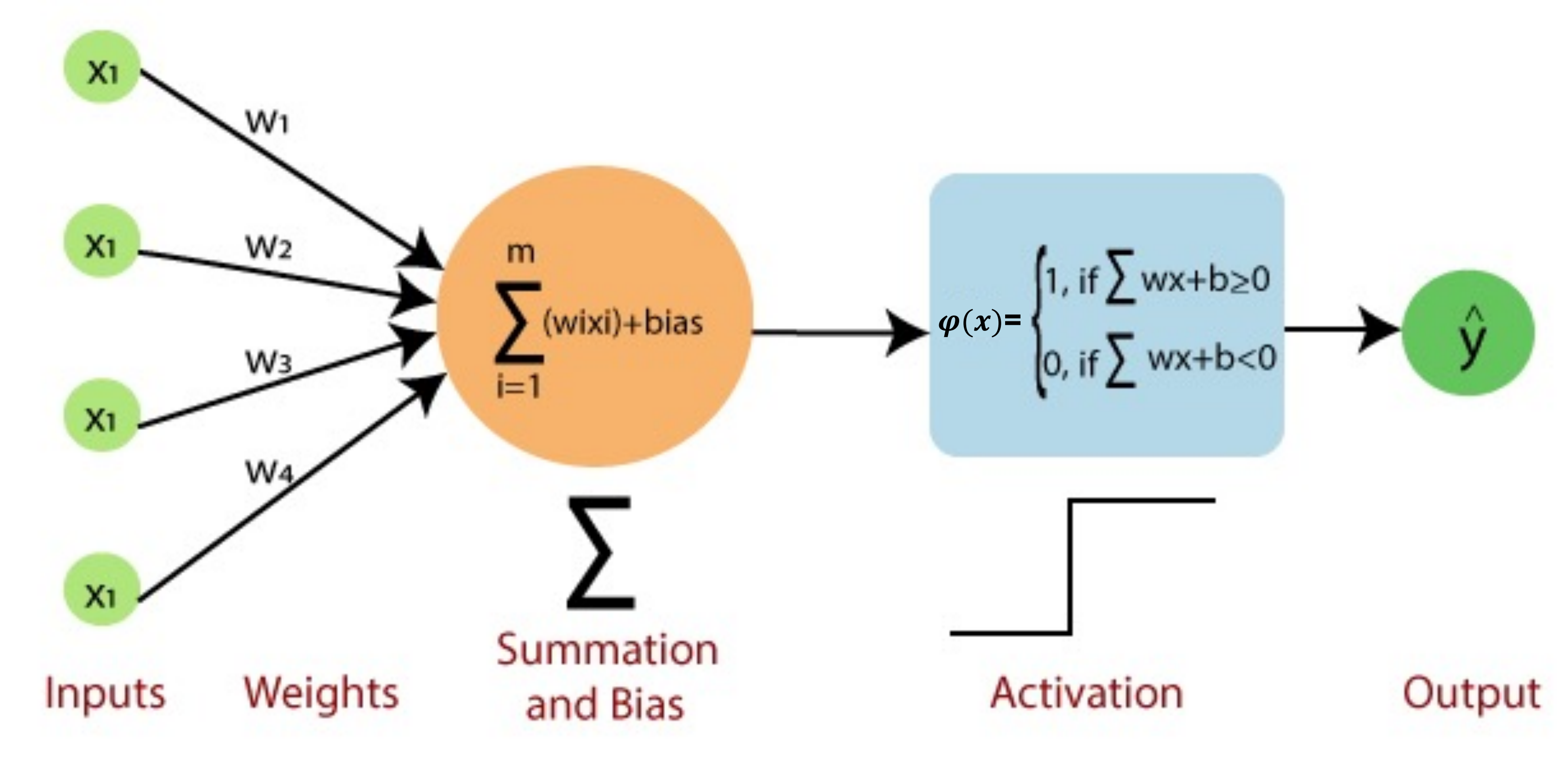
- K-nearest neighbor

…

- Principle component analysis (PCA)
- Linear discriminant analysis (LDA)

…

Example of Machine Learning (Inference): Perceptron



$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi(w^T x + b)$$

Machine Learning – Training vs. Inference

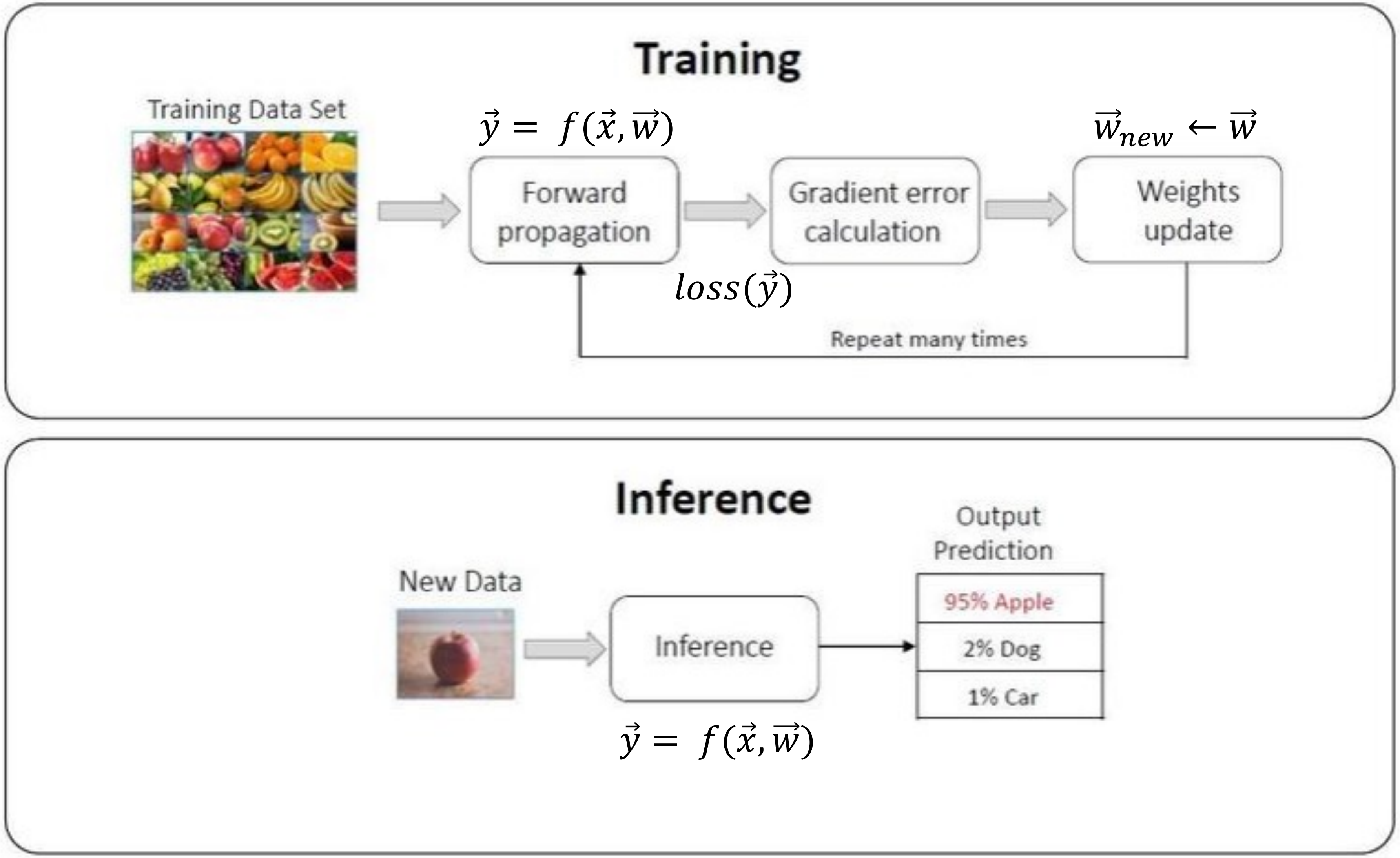
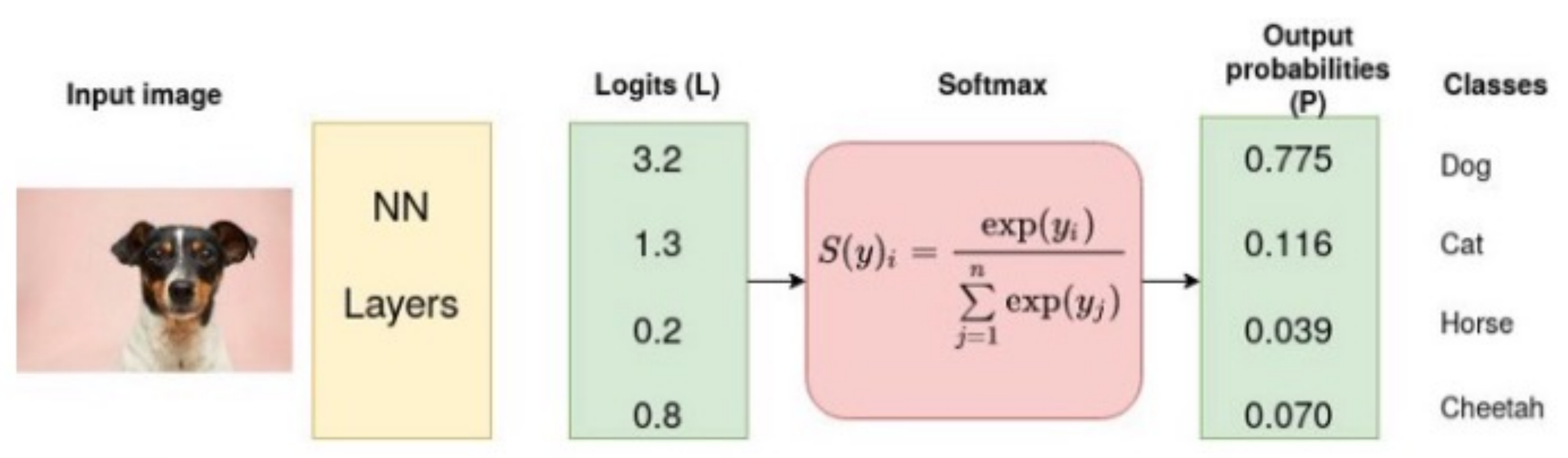
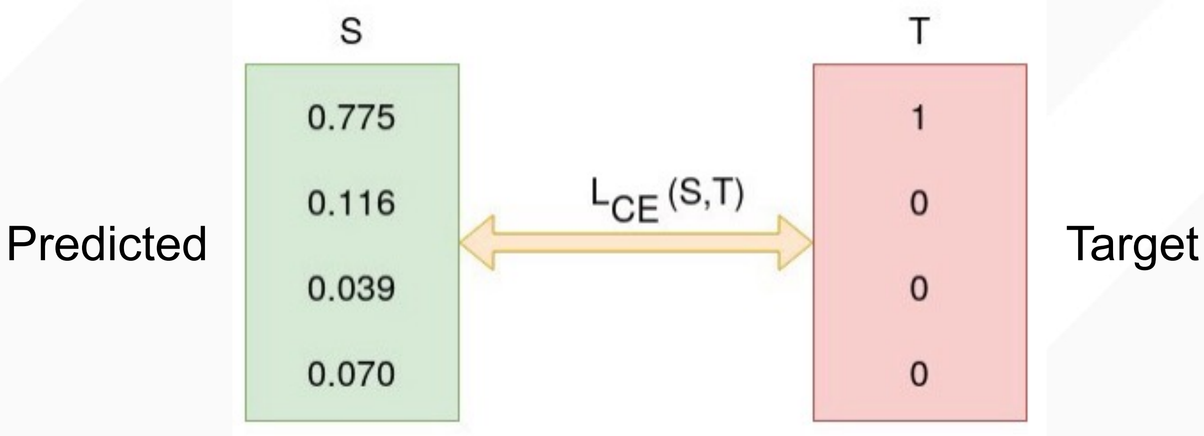


figure: https://www.zdnet.com/

Training





figures: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

Loss (Cost) Function (Mean Squared Error)

Mean squared error loss

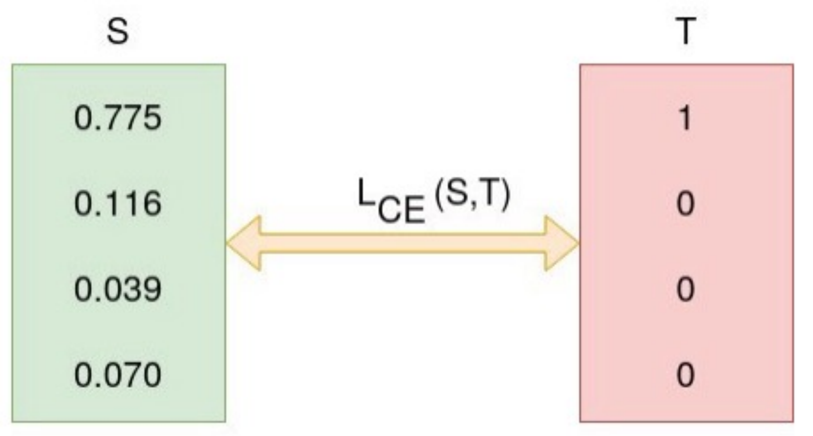
```
# calculate mean squared error
def mean_squared_error(actual, predicted):
    sum_square_error = 0.0

    for i in range(len(actual)):
        sum_square_error += (actual[i] - predicted[i])**2.0
    mean_square_error = sum_square_error / len(actual)
    return mean_square_error
```

https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

Loss (Cost) Function (Cross-Entropy Loss)

Cross-Entropy loss (or logarithmic loss, or logistic loss)

```
# calculate cross entropy
def categorical_cross_entropy(actual, predicted):
    sum_score = 0.0

    for i in range(len(actual)):                # number of class
        sum_score += actual[i] * log(1e-15 + predicted[i])
    mean_sum_score = 1.0 / len(actual) * sum_score
    return -mean_sum_score

```



Predicted Target

$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$

figures: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

Gradient Descent (GD) Algorithm

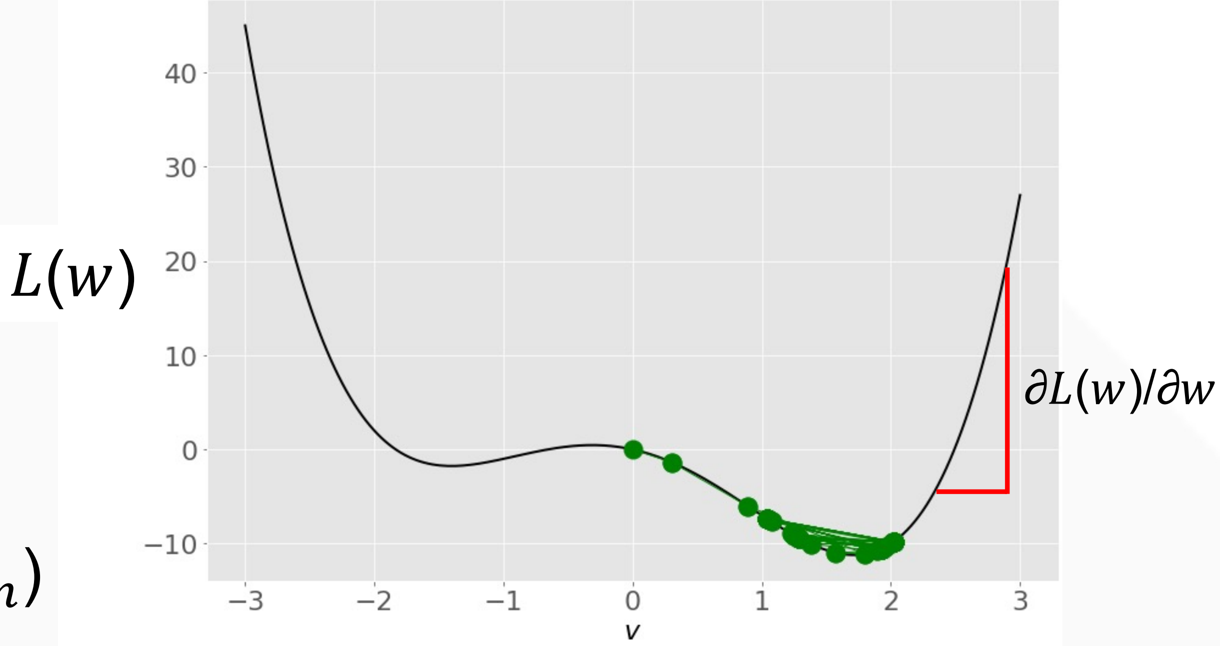
```python
# calculate cross entropy
def gradient_descent(gradient, start, learn_rate, n_iter):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        vector += diff
    return vector
```

Finding optimal **v** to minimize cost L

$\boldsymbol{w} = (w_1, \ldots, w_n)$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla L(w)$

- gradient $\nabla L(w) = (\partial L/\partial w_1, \ldots, \partial L/\partial w_n)$
- learning rate η



$L(w)$ vs. 1-dim w

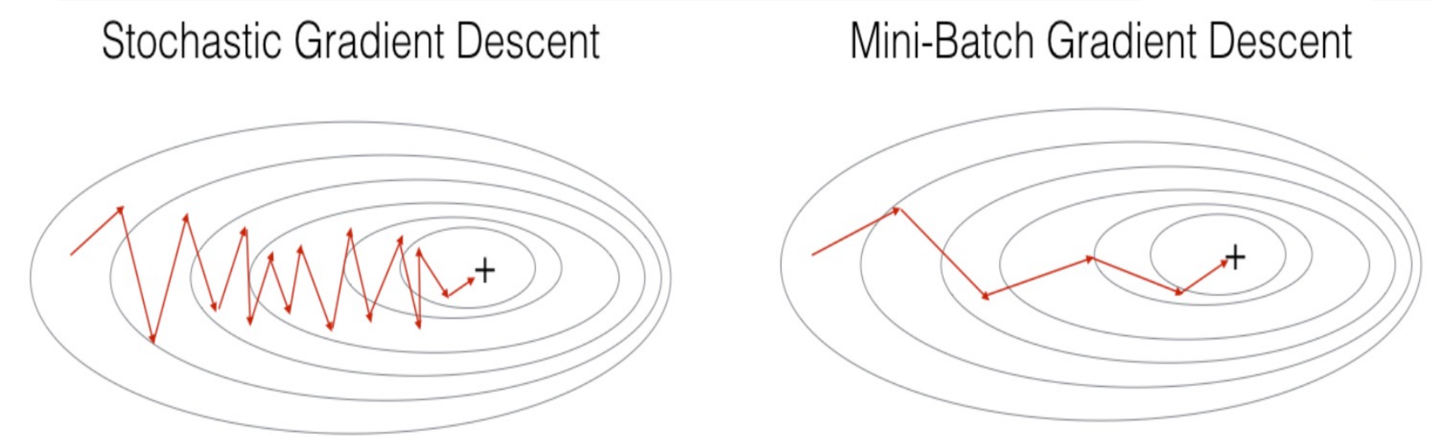
https://realpython.com/gradient-descent-algorithm-python/

Variations of Gradient Decent Algorithms

- (Batch) Gradient Descent algorithm:
  - Every updated, the gradient computed with all the data points $\sum_{i \in all} \nabla_i$
  - Computation is slow, but update is smooth

- Stochastic Gradient Descent (SGD) algorithm:
  - Every updated, the gradient computed with only single data point
  - Computation is fast, but the training is noisy

- Mini-batch Gradient Descent (SGD) algorithm:
  - Every updated, the gradient computed with $\sum_{i \in subset} \nabla_i$



Calculation of Gradient with Mean Square Error Example

$$L(w,b) = MSE = \frac{1}{m}\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right)^2 \qquad \hat{y} = \emptyset(wx+b), \quad \emptyset(x) = x$$

, where m data points are used to train

$$\nabla L(w) = \frac{\partial L}{\partial w} = \frac{1}{m}\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right) * (-2x_j)$$

$$\nabla L(b) = \frac{\partial L}{\partial b} = \frac{1}{m}\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right) * (-2)$$

What if $\hat{y} = \emptyset(\sum_{i=1}^{n} w_i x_i + b), \emptyset(x) = x$?

$\nabla L(w_i) = \frac{\partial L}{\partial w_i} = \frac{1}{m}\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right) * (-2x_{ji})$, here $i = 1,2, \ldots n$

Type of Activations

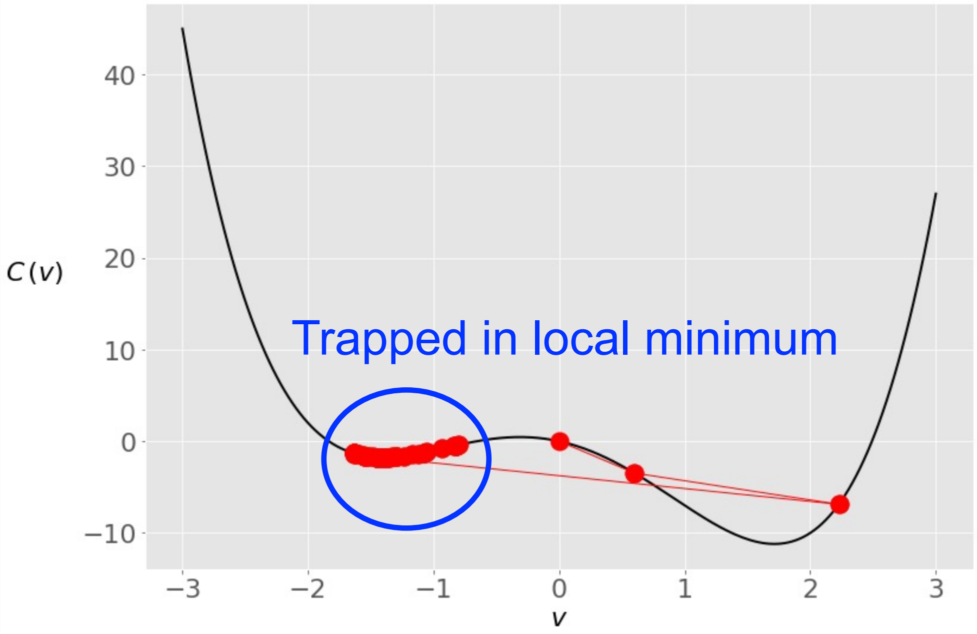
Function Type	Equation	Derivative
Linear	$f(x) = ax + c$	$f'(x) = a$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\,(1 - f(x))$
TanH	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^{x} - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$

https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/

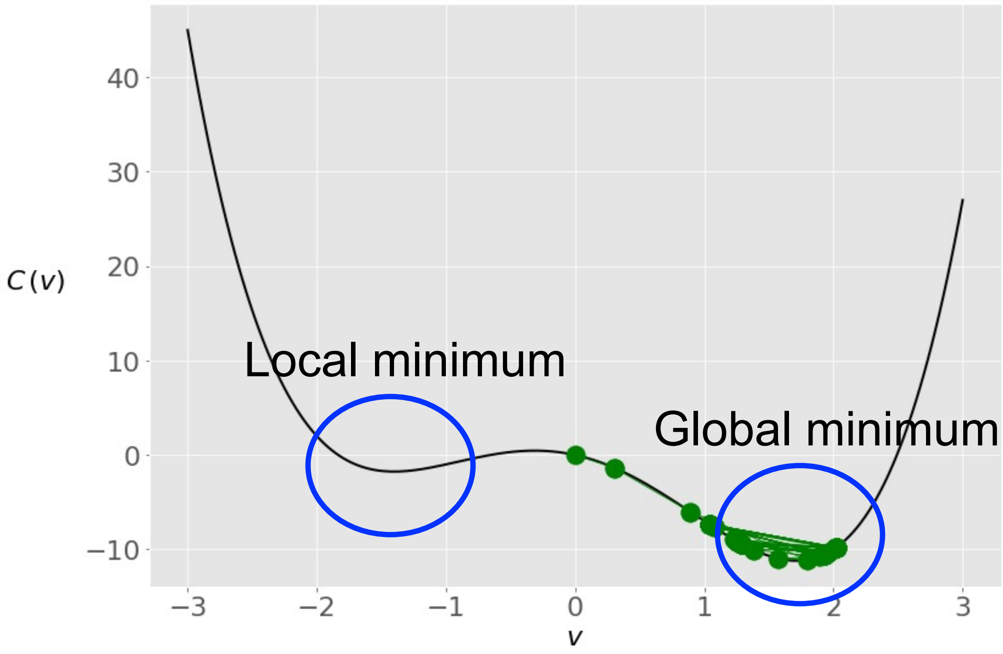
Gradient Descent (Example1)

https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

Impact of Learning Rate (η)



Bad training



Desirable training

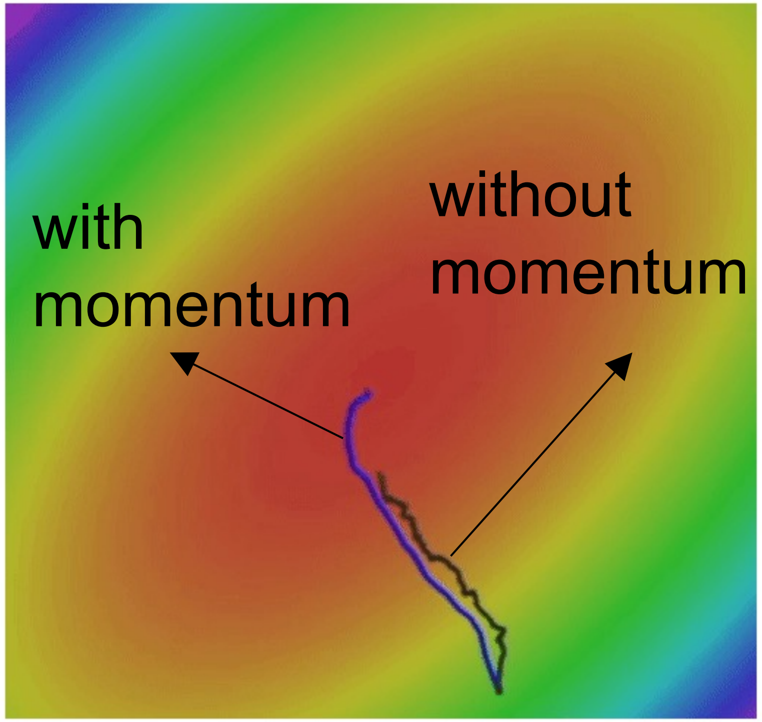
- Large learning rate: - could skip the global minimum
  - might not converge, but oscillates
- Small learning rate: training is very slow

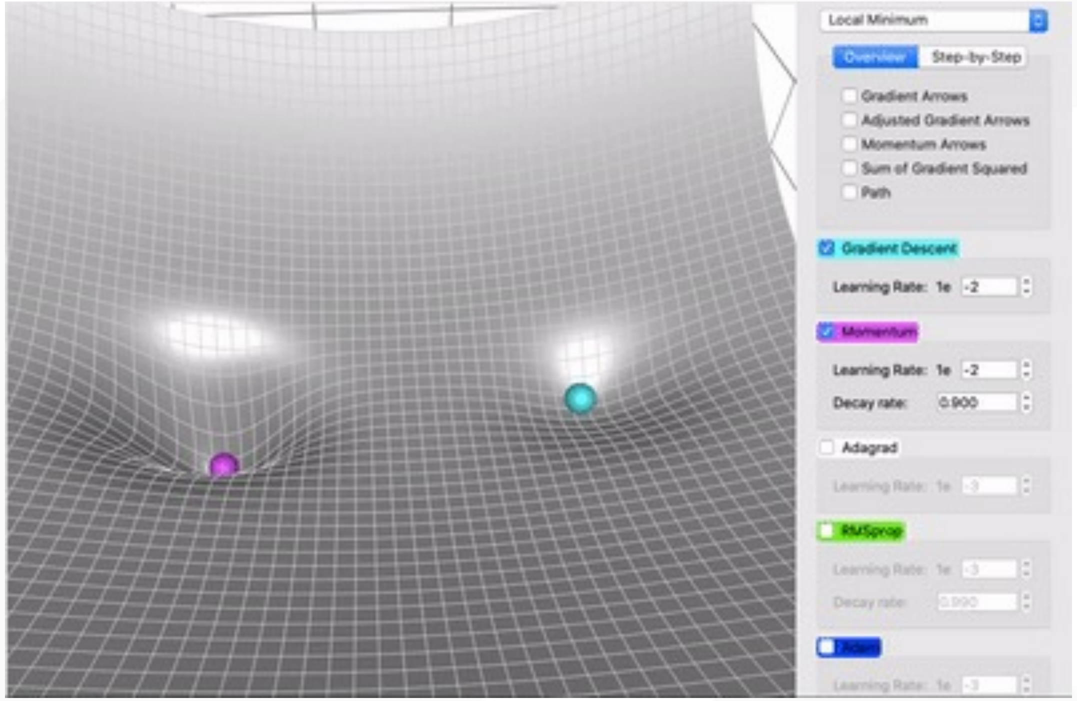
Learning Rate Decaying

```
# calculate cross entropy
def gradient_descent(gradient, start, learn_rate, n_iter):
    vector = start
    initial_learn_rate = large number
    for _ in range(n_iter):
        diff = - learn_rate * gradient(vector)
        vector += diff
        learn_rate = initial_learn_rate * (1 / (1 + decay * iteration))
    return vector
```

- Learning rate decaying:
  - Initially start with large learning rate for fast learning
  - Learning rate decreases as iteration goes on for better convergence

Momentum (v)





Conventional SGD

$\boldsymbol{w} = (w_1, \ldots, w_n)$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla L(w)$

SGD with momentum

$v_{t+1} = \rho v_t + \nabla L(w)$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta v_{t+1}$

- Momentum (purple ball) helps not to fall into local minima

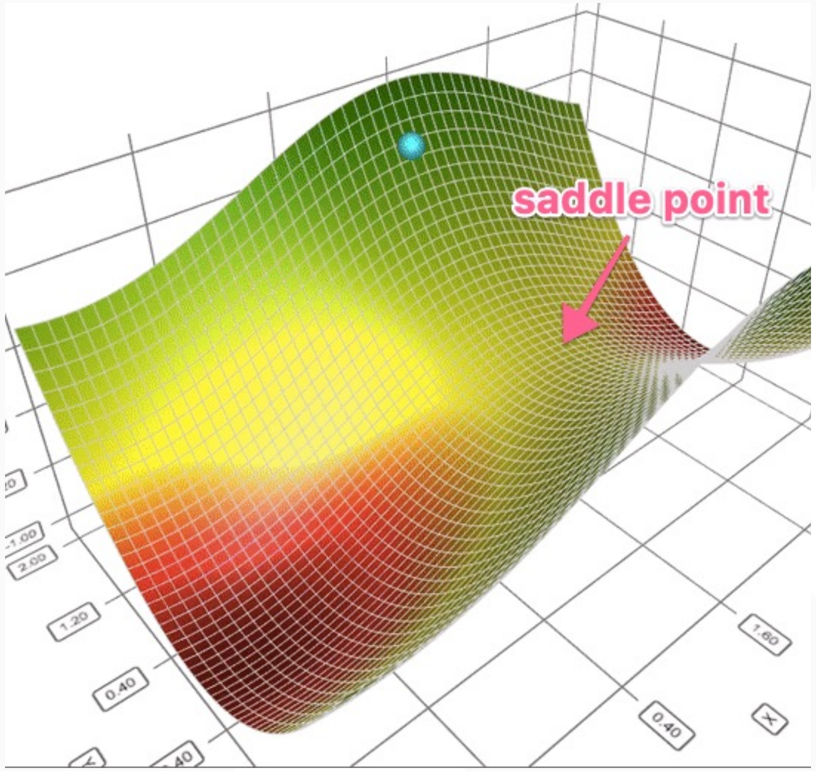
https://morioh.com/p/3f20600908f3
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Adaptive Gradient (AdaGrad)

AdaGrad

$$g_{t+1,i} = g_{t,i} + \nabla L(w_i)^2$$

$$\boldsymbol{w_i} \leftarrow \boldsymbol{w_i} - \eta \frac{\nabla L(w_i)}{\sqrt{g_{t+1,i}} + 1e^{-5}}$$



- AdaGrad (Gray ball):
  - Adaptively change the learning rate
  - Prevent excessive moving only in one direction (cyan ball) by increasing the denominator by accumulating all the movement so far)
  - Accelerate the learning from the sparsely acquired movement.
  - Learning gets slower as time goes by -> RMSProp proposed

Momentum (v)

SGD with momentum

$$v_{t+1} = \rho v_t + \nabla L(w)$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta v_{t+1}$$

RMSProp

$$g_{t+1} = \rho g_t + (1-\rho)\nabla L(w)^2$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\nabla L(w)}{\sqrt{g_{t+1}} + 1e^{-5}}$$

Adam

$$m_{t+1} = \rho_1 m_t + (1-\rho_1)\nabla L(w)$$ momentum

$$v_{t+1} = \rho_2 v_t + (1-\rho_2)\nabla L(w)^2$$ RMSProp

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\nabla L(w)}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$

Perceptron Training with Gradient Descent (Example2)

https://medium.com/@tomgrek/building-your-first-neural-net-from-scratch-with-pytorch-56b0e9c84d54