SUMMARY

- LEARNING = EXPIRIENCE (DATA) + TASK + PERFORMANCE
  (E) (T) (P)
- INTRODUCTION TO PROBABILITY (SKIPPED)
- **UNSUPERVISED LEARNING**:

  GIVEN DATA $x_1, \dots, x_n$ (NO LABELS y_i) -> OUTPUT : PATTERN BEHIND DATA

  -> EG. CLUSTERING

  - **CLUSTERING**:

    PROCESS OF GROUPING OBJECTS INTO CLASSES OF SIMILAR OBJECTS

    - THE CONCEPT OF "SIMILARITY" WILL BE DEFINED BY DISTANCE
    - 2 ALGORITHMS:
      - K-MEANS ALGORITHM (-> ALREADY COVERED) : HARD CLUSTERING METHOD
      - MIXTURE OF GAUSSIANS

      L-> $\forall$ POINT -> $\exists!$ CLUSTER

    - **K-MEANS**:

      GIVEN A SET OF OBSERVATIONS $(x_1, \dots, x_n) \,/\, x_i \in \mathbb{R}^d$, $k \leq n$ (k ↑)

      , PARTITION THE n OBSERVATIONS INTO K SETS $S = \{S_1, \dots, S_k\}$

      , SUCH THAT THE SETS MINIMIZE THE WITHIN-CLUSTER SUM OF SQUARE:

      -> $\underset{S}{\arg\min} \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2$ ($\forall$ SET, $\forall$ POINT in S_i) / μ_i : MEAN OF POINTS IN S_i

· GAUSSIAN MIXTURE MODEL : https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

· NOT HARD CLUSTERING : GMM ASSIGN TO EACH POINT A PROBABILITY THAT THE POINT BELONGS TO A CLUSTER

-> $\forall$ POINT, K CLUSTER -> $(\tilde{\pi}_1, \dots, \tilde{\pi}_K) \Big/ \sum_{i=1}^{K} \tilde{\pi}_i = 1$

($\tilde{\pi}_i$: P OF BELONGING TO CLUSTER i)

· K "CLUSTERS" -> K GAUSSIAN COMPONENTS $\sim \mathcal{N}(\bar{\mu}_i, \Sigma_i)$

-> $p(x) = \sum_{i=1}^{K} p(x \mid Y=i)\, P(Y=i)$

(OBSERVED DATA; MIXTURE COMPONENTS; MIXTURE PROPORTION)

$P(Y=i)$: P THAT OUTPUT $\in$ GAUSSIAN i

· ASSUMING THAT $p(x \mid Y=i) = \mathcal{N}(\mu_i, \sigma^2 I)$;

-> THE PARAMETERS TO FIND ARE $\mu_1, \dots, \mu_K, \sigma^2, \tilde{\pi}_1, \dots, \tilde{\pi}_K$

($\mu_1, \dots, \mu_K$: MEAN OF THE K GAUSSIANS; σ^2: VAR. OF K GAUSS.; $P[Y=i]$)

-> ALL THIS PARAMETERS CAN BE FOUND USING MLE :

$\theta = [\mu_1, \dots, \mu_K, \sigma^2, \tilde{\pi}_1, \dots, \tilde{\pi}_K]$

↳ IN GENERAL : $\sigma^2 \mapsto \Sigma_1, \dots, \Sigma_K$

-> EXPECTATION - MAXIMIZATION ALGORITHM :

· E - STEP : ASSIGN CLUSTERS TO EACH POINT, IN A SOFT WAY

· M - STEP : UPDATING CLUSTERS IN K - MEANS

E-step

Compute "expected" classes of all datapoints for each class

$$P(y_j = i | x_j, \theta^{t-1}) = \frac{\exp(-\frac{1}{2\sigma^2}\|x_j - \mu_i^{t-1}\|^2)\pi_i}{\sum_{i=1}^{K} \exp(-\frac{1}{2\sigma^2}\|x_j - \mu_i^{t-1}\|^2)\pi_i}$$

In K-means "E-step" we do hard assignment. EM does soft assignment

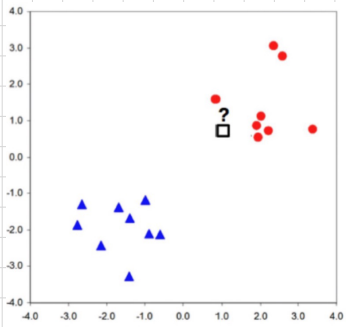
M-step

Compute Max. like $\boldsymbol{\mu}$ given our data's class membership distributions (weights)

$$\mu_i^t = \sum_{j=1}^{n} w_j x_j \quad \text{where } w_j = \frac{P(y_j = i | x_j, \theta^{t-1})}{\sum_{l=1}^{n} P(y_l = i | x_l, \theta^{t-1})}$$

KNN, PERCEPTRONS:

KNN:



- A NEW TEST POINT GETS THE SAME LABEL AS THE K MOST CLOSE TRAINING POINTS, BY MAJORITY VOTING

S: SENSITIVE TO OUTLIERS

- KNN CAN BE USED FOR REGRESSION TOO:

  GIVEN THE TEST POINT X:

  - COMPUTE DISTANCE $D(x, x_i)$ TO EVERY TRAINING EXAMPLE, SELECT K CLOSEST AND THEIR Y_i
  - OUTPUT: MEAN OF $Y_{i,1}, \dots, Y_{i,K}$ $\rightarrow$ $\hat{Y} = \frac{1}{K} \sum_{j=1}^{K} Y_i$

- VALUE OF K: USE A VALIDATION SET AND VARY K

  $\rightarrow$ SELECT K THAT GIVES, IN GENERAL, THE BEST PERFORMANCE

- DISTANCE METRICS:

  DERIVED FROM THE MINKOWSKI DISTANCE, $L_p(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$

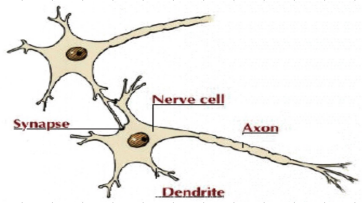
  - $p = 1$: L_1 NORM (MANHATTAN DISTANCE) $= L_1(x, y) = \sum_{i=1}^{d} |x_i - y_i|$
  - $p = 2$: L_2 NORM (EUCLIDEAN DISTANCE) $= L_2(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$
  - $p = \infty$: L_∞ NORM (CHEBYSHEV DISTANCE) $= L_\infty(x, y) = \max_i |x_i - y_i|$
  - $p = 0$: L_H (HAMMING DISTANCE) $= L_H(x, y) = \sum_{i=1}^{d} 1_{x_i \neq y_i}$

- KNN TRICKS:

  - PARTIAL DISTANCE CALCULATION: D COMPUTED ON SUBSET $r < d$ OF THE FULL d DIMENSION
  - PRE-STRUCTURING/SEARCHING FOR PROTOTYPES: EXPLORE SIMILARITIES IN SAMPLES $\rightarrow$ DATA AS SEARCH TREES
  - EDITING: USELESS POINTS ARE PRUNED $\rightarrow$ COMPLEXITY REDUCTION
  - FEATURE NORMALIZATION

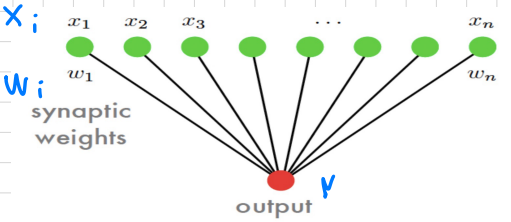
• PERCEPTRON :

- FROM BIOLOGY :
  - GOOD BEHAVIOUR -> REWARD, BAD BEHAVIOUR -> PUNISHED (OR NO REWARD)
- NEURONS :
  - SOMA (CPU) : CELL BODY, COMBINES SIGNALS
  - DENDRITE (INPUT BUS) : COMBINE INPUTS FROM OTHER CELLS
  - SYNAPSE (INTERFACE)
  - AXON (CABLE) : TRANSPORTS ACTIVATION SIGNAL TO NEURONS AT DIFFERENT LOCATION



-> IMPLEMENTATION ON MACHINE :

- NEURON :



$$f(x) = \sum_i w_i \cdot x_i = \langle \overline{w}, \overline{x} \rangle$$

SCALAR PRODUCT

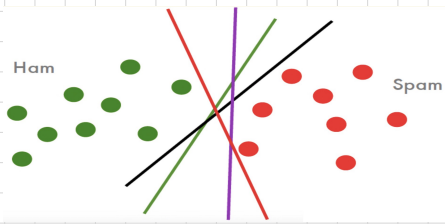
- PERCEPTRON :

NON LINEAR DECISION FUNCTION (ex. SIGMOID, RELU)

BIAS : LINEAR OFFSET

$$f(x) = G \cdot \left(\langle w, x \rangle + b \right)$$

IF BINARY OUTPUT -> LINEAR SEPARATING HYPERPLAN



SEE ALSO NOTION

—> $\overline{w}$ AND b ARE THE PARAMETERS TO LEARN

- LEARNING ALGORITHM : ~> FOR BINARY CLASSIFIER -> OUT : $\{-1, 1\}$

1. SET $w = 0$, $b = 0$
2. REPEAT $\forall$ POINT $x_i \in X$ (TRAINING DATASET) :

   IF $y_i \left[\langle w, x_i \rangle + b \right] \leq 0$: (PREDICTED LABEL, TRUE LABEL)

   IF <FORMULA> ≤ 0 -> x_i CLASSIFIED WRONG

   <FORMULA> ≤ 0 <=> $\begin{cases} y_i = -1, x_i = 1 \\ \text{OR} \\ y_i = 1, x_i = -1 \end{cases}$

   <=> CLASSIFIED WRONG -> NOT $\forall i$, ONLY $i \in I$

   w, b UPDATE

   $w \leftarrow w + y_i x_i$ ~> $w = 0 + y_1 x_1 + y_2 x_2 + \dots + y_m x_m = \sum_{i \in I} y_i x_i$ (STEP 0, STEP 1, STEP 2, STEP m)

   $b \leftarrow b + y_i$ ~> (ANALOG AS FOR w)

   UNTIL ALL POINTS ARE CLASSIFIED

- **CONVERGENCE THEOREM**:

  IF $\exists$ 2 VALUES (w^*, b^*) / $\forall i \left[\langle x_i, w^* \rangle + b^* \right] \geq \rho$, $\forall i$

  -> PERCEPTRON CONVERGES TO A LINEAR SEPARATOR AFTER A FINITE NUMBER OF STEPS BOUNDED BY:

  $$\left((b^*)^2 + 1\right)\left(r^2 + 1\right)\rho^{-2}, \quad \|x_i\| \leq r$$

  (PROOF ON NOTION, OR PAG. 11-12)

- IT IS NEEDED TO ONLY STORE ERRORS

- **CONCEPTS AND VERSION SPACE**:

  - REALIZABLE CONCEPT:

    - $\exists$ SOME f THAT CAN SEPARATE DATA, f INCLUDED IN CONCEPT SPACE

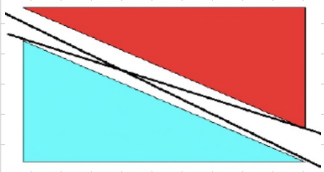
      -> FOR PERCEPTRON, DATA ARE LINEARLY SEPARABLE

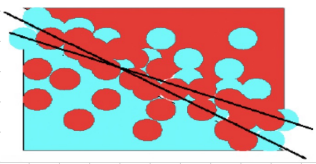
  - UNREALIZABLE CONCEPT:

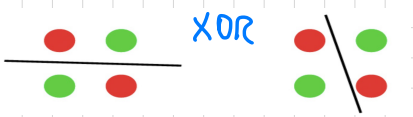
    - DATA NOT SEPARABLE

      -> $\exists$ SUITABLE f: DATA NON DISTINGUISHABLE

      ex. XOR -> NOT LINEARLY SEPARABLE

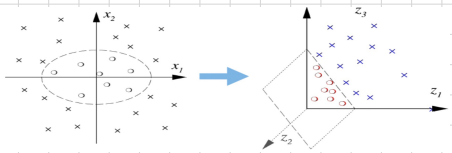






• NON - LINEARITY AND PRE-PROCESSING :

• DATA ARE MAPPED INTO FEATURE-SPACE : $x \longmapsto \phi(x)$



→ SOLUTIONS BASED ON $\phi(x_i)$

	1	2	3	4	5	6	7	8	9	0
Loops	0	0	0	1	0	1	0	2	1	1
3 Joints	0	0	0	0	0	1	0	0	1	0
4 Joints	0	0	0	1	0	0	0	1	0	0
Angles	0	1	1	1	1	0	1	0	0	0
Ink	1	2	2	2	2	2	1	3	2	2

e.g. OCR RECOGNITION — PIXELS $\mapsto \phi$(PIXELS)

→ RECOGNIZE CHAR BY FEATURES

• PERCEPTRON ALGORITM ON FEATURES :

1. SET $w = 0$, $b = 0$
2. REPEAT $\forall$ POINT $x_i \in X$ (→ TRAINING DATASET) :

IF $y_i (w \cdot \phi(x_i) + b) \leq 0$:

w, b UPDATE

$w \leftarrow w + y_i \phi(x_i)$ → $w = \sum_{i \in I} y_i \phi(x_i)$

$b \leftarrow b + y_i$

UNTIL ALL POINTS ARE CLASSIFIED ~> $y_i (w \cdot \phi(x_i) + b) > 0, \forall i$

→ $f(x) = \sum_{i \in I} y_i \langle \phi(x_i), \phi(x) \rangle + b$

• GOOD FEATURE ENGINEERING NEEDS DOMAIN EXPERTS, EXPENSIVE TO COMPUTE :

└ SOL: → COMPUTE MANY FEATURES AND HOPE THAT ∃ GOOD ONES

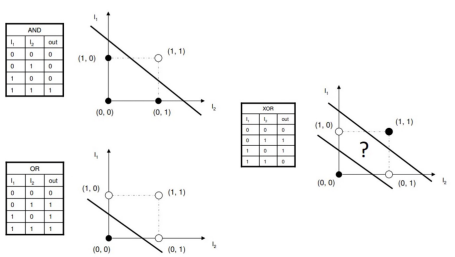
, TRY TO SELECT FEATURES EFFICIENTLY : NO CORRELATION, NOT TOO MANY

• SOLVING XOR : https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182

→ NOT LINEARLY SOLVABLE, BUT IN 3 DIMENSIONS:

$(x_1, x_2) \longmapsto (x_1, x_2, x_1 x_2)$

→ IN 3D IT IS LINEARLY SOLVABLE



• QUADRATIC FEATURE : $\phi(x) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$

- **KERNEL FUNCTION:** https://youtu.be/Q7vT0--5VII

FEATURE EXTRACTING CAN BE VERY COSTLY; EX. 2^{nd} ORDER FEATURES IN 1000 DIMENSIONS

SOL: DON'T COMPUTE $\langle \dots, \dots \rangle$, COMPUTE $K(x, x')$ KERNEL FUNCTION

DEF.: $K: X \times X \longmapsto \mathbb{R} \;/\; K(x, x') = \langle \phi(x), \phi(x') \rangle$

↳ SOMETIMES $K(x, x')$ IS MUCH CHEAPER THEN $\phi(x)$

- ALGORITHM: SCORE 66/86

$$f(x) = \sum_{i \in I} \nu_i \langle \phi(x_i), \phi(x) \rangle + b = \sum_{i \in I} \nu_i K(x_i, x) + b$$

- POLYNOMIAL KERNELS

EX.

Simple Example: $x = (x1, x2, x3)$; $y = (y1, y2, y3)$. Then for the function $f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3)$, the kernel is $K(x, y) = (\langle x, y \rangle)^2$.

EXPLICIT CALCULUS →

Let's plug in some numbers to make this more intuitive: suppose x = (1, 2, 3); y = (4, 5, 6). Then:
f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)
f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)
<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

USING $K(x, y)$ →

Now let us use the kernel instead:
K(x, y) = (4 + 10 + 18) ^2 = 32^2 = 1024
Same result, but this calculation is so much easier.

- **MERCER'S THEOREM:**

$\forall$ SYMMETRIC $K: X \times X \longmapsto \mathbb{R} \;/\; K$ IS SQUARE INTEGRABLE IN $X \times X$

AND SATISFIES:

$$\int_{X \times X} K(x, x') f(x) \cdot f(x') \, dx \, dx \geq 0 \;, \forall f \in L_2(X)$$

$$\hookrightarrow \exists \; \phi_i : X \longmapsto \mathbb{R} \;, \exists \; \lambda_i \geq 0$$

$$/ \quad K(x, x') = \sum_i \lambda_i \phi(x) \phi(x') \;, \forall x, x' \in X$$

→ IT GIVES INFO ON WHICH FUNCTIONS CAN BE KERNEL FUNCTIONS

- DISTANCE IN FEATURE SPACE :

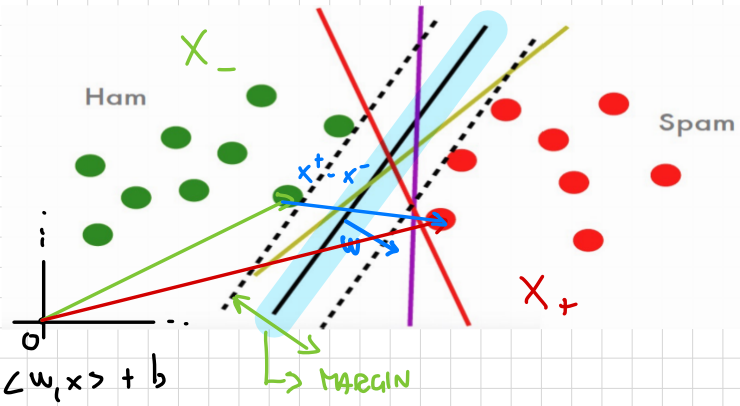
$$d^2(x, x')^2 := \| \phi(x) - \phi(x') \|^2 = K(x,x) + K(x', x') - 2K(x, x')$$

- KERNEL MATRIX : $K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle = K(x_i, x_j)$

  ↳ K_{ij} TELLS US THE SIMILARITY OF $\phi(x_i)$ AND $\phi(x_j)$

SVM :

LARGE MARGIN CLASSIFIER :



- $\langle w, x \rangle + b \leq -1$
- $\langle w, x \rangle + b \geq 1$

- LINEAR FUNCTION : $f(x) = \langle w, x \rangle + b$

MARGIN OPTIMIZATION :

- MARGIN : $\frac{\langle x_+ - x_- , w \rangle}{\|w\|} =$ (dot product is a measure of similarity)

OR $\frac{1}{\|w\|}$ (half margin)

$$= \frac{1}{\|w\|} \left[\underbrace{[\langle x_+, w \rangle + b]}_{1} - \underbrace{[\langle x_-, w \rangle + b]}_{-1} \right] = \frac{2}{\|w\|}$$

- OPTIMIZATION PROBLEM :

"WIDTH" MAXIMIZATION (MARGIN)

- MAXIMIZE $_{w,b}$ $\frac{1}{\|w\|}$, CONSTRAINT : $\forall_i \left[\langle x_i, w \rangle + b \right] \geq 1$

MINIMIZE L_2 REGULARIZATION
↳ HOW WELL DATA ARE CLASSIFIED

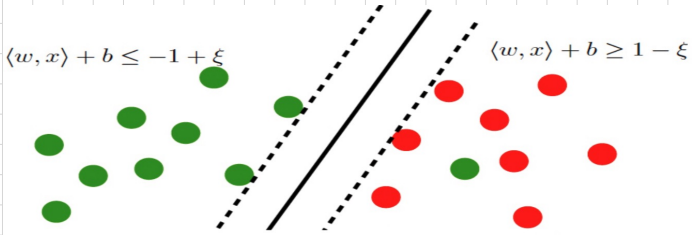
- MINIMIZE $_{w,b}$ $\frac{1}{2} \|w\|^2$, CONSTRAINT : $\forall_i \left[\langle x_i, w \rangle + b \right] \geq 1$

-> LAGRANGE METHOD :

- $L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i \underbrace{\left[\forall_i \left[\langle x_i, w \rangle + b \right] - 1 \right]}_{\text{CONSTRAINT}}$

↳ ... -> $w = \sum_i \forall_i \alpha_i x_i$, $\forall_i \left[\langle w, x_i \rangle + b \right] = 1$

SOFT MARGIN CLASSIFIER :



- IN A DATASET LIKE ON FIGURE, HARD MARGINS ARE NOT POSSIBLE

→ NEED TO "RELAX" MARGINS → ADDING A SLACK VARIABLE ξ .

- $\langle w, x \rangle + b \leq -1 \longrightarrow \langle w, x \rangle + b \leq -1 + \xi$
- $\langle w, x \rangle + b \geq 1 \longrightarrow \langle w, x \rangle + b \geq 1 - \xi$

- OPTIMIZATION PROBLEM :
  - MINIMIZE $\underset{w,b}{\min} \frac{1}{2}\|w\|^2 + C \cdot \sum_i \xi_i$, CONSTRAINT : $\forall_i \left[\langle w, x_i \rangle + b\right] \geq 1 - \xi_i$, $\xi_i > 0$
    - ALWAYS FEASABLE

→

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 + C \cdot \sum_i \xi_i - \sum_i \alpha_i \left[\forall_i \left[\langle x_i, w \rangle + b\right] + \xi_i - 1\right] - \sum_i \eta_i \xi_i$$

- ... → KARUSH KUHN TUCKER CONDITIONS :

$$\begin{cases} \alpha_i = 0 & \rightarrow \forall_i \left[\langle w, x_i \rangle + b\right] \geq 1 \\ 0 < \alpha_i < C & \rightarrow \forall_i \left[\langle w, x_i \rangle + b\right] = 1 \\ \alpha_i = C & \rightarrow \forall_i \left[\langle w, x_i \rangle + b\right] \leq 1 \end{cases}$$

NON-LINEAR SEPARATION :

→ KERNEL TRICK : $\begin{cases} x_i \longmapsto \phi(x_i) \\ \langle x_i, x \rangle \longmapsto k(x_i, x), \quad \langle x_i, x_j \rangle \longmapsto k(x_i, x_j) \end{cases}$

→

- **Linear soft margin problem**

$$\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$\text{subject to } y_i\left[\langle w, x_i \rangle + b\right] \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

- **Dual problem**

$$\underset{\alpha}{\text{maximize}} \ -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i$$

$$\text{subject to } \sum_i \alpha_i y_i = 0 \text{ and } \alpha_i \in [0, C]$$

- **Support vector expansion**

$$f(x) = \sum_i \alpha_i y_i \langle x_i, x \rangle + b$$

⟶

- **Linear soft margin problem**

$$\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$\text{subject to } y_i\left[\langle w, \phi(x_i) \rangle + b\right] \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

- **Dual problem**

$$\underset{\alpha}{\text{maximize}} \ -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j) + \sum_i \alpha_i$$

$$\text{subject to } \sum_i \alpha_i y_i = 0 \text{ and } \alpha_i \in [0, C]$$

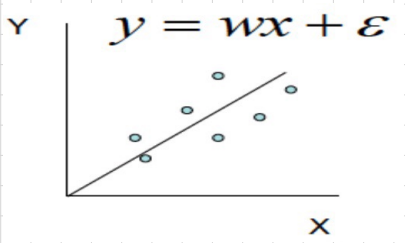
- **Support vector expansion**

$$f(x) = \sum_i \alpha_i y_i k(x_i, x) + b$$

- **REGRESSION**:

X: INPUT FEATURES

- $Y = X \cdot W + \varepsilon$ → NOISE / OTHER PARAMETERS



- **LINEAR REGRESSION**:

GOAL: ESTIMATE W FROM $\langle x_i, Y_i \rangle$ PAIRS

→ MINIMIZE LSE (LEAST SQUARE ERROR).

$$\arg\min_W \sum_i (Y_i - W x_i)^2$$

$$\rightarrow \frac{\partial}{\partial w} \sum_i (Y_i - W x_i)^2 = 0 \rightarrow \dots \rightarrow W = \frac{\sum_i x_i Y_i}{\sum_i x_i^2}$$

- BIAS TERM W_0:

- $Y = W_0 + W_1 X + \varepsilon \longrightarrow$

$$W_0 = \frac{\sum_i Y_i - W_1 x_i}{n}$$

$$W_1 = \frac{\sum_i x_i (Y_i - W_0)}{\sum x_i^2}$$

- **MULTIVARIATE REGRESSION**:

$$Y = W_0 + W_1 X_1 + W_2 X_2 + \dots + W_K X_K + \varepsilon$$

BUT ALSO...

$$Y = W_0 + W_1 X_1^2 + \dots + W_K X_K^2 + \varepsilon$$

↳ FEATURES ARE POLYNOMIAL

BUT REGRESSION IS LINEAR

- $X \longmapsto \phi(X) = X^K$ (IN THIS CASE)

- Polynomial: $\phi_j(x) = x^j$ for j=0 ... n
- Gaussian: $\phi_j(x) = \frac{(x - \mu_j)}{2\sigma_j^2}$
- Sigmoid: $\phi_j(x) = \frac{1}{1 + \exp(-s_j x)}$

· GENERAL LINEAR REGRESSION PROBLEM:

- $y = \sum_{j=0}^{m} w_j \phi_j(x)$

→ MINIMIZE LSE:

GENERIC POLYNOMIAL REGRESSION

$$\arg\min_w J(w) = \sum_i \left(y_i - \sum_j w_j \phi_j(x^i) \right)^2$$

SIMILAR TO LINEAR REGRESSION

$$\rightarrow \frac{\partial}{\partial w} J(w) = 0 \rightarrow \dots \rightarrow \sum_i y^i \phi(x^i)^T = w^T \left[\sum_i \phi(x^i) \phi(x^i)^T \right]$$

$\sum_i \phi(x)^2$ IN MATRIX NOTATION

IF WE DEFINE $\phi = \begin{pmatrix} \phi_0(x^1) & \phi_1(x^1) & \dots & \phi_m(x^1) \\ \phi_0(x^2) & \phi_1(x^2) & \dots & \phi_m(x^2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x^n) & \phi_1(x^n) & \dots & \phi_m(x^n) \end{pmatrix}$

POWERS OF EACH FEATURE

DIFFERENT FUNCTION ON EACH FEATURE

$$\rightarrow \underset{(k+1)\times 1}{w} = \underbrace{\left(\underbrace{\phi^T}_{(\dots)} \underbrace{\phi}_{n \times (k+1)} \right)^{-1}}_{(\dots)} \cdot \underbrace{\phi^T}_{(\dots)} \cdot \underbrace{y}_{n \times 1}$$

PSEUDO - INVERSE

- ALSO MLE FOR w IS THE SOLUTION ABOVE

- **LOGISTIC REGRESSION:** https://bit.ly/3UDyedq

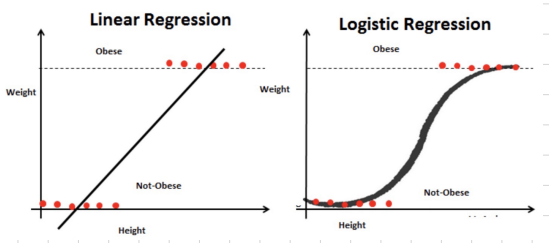
USEFUL WHEN Y IS NOT CATEGORICAL

WHEN OUTPUT IS BINARY / DISCRETE → ∃ MORE EFFICIENT MODELS THAN LINEAR REGRESSION FOR CLASSIFICATION

- CLASSIFICATION: $p(Y|X;\theta)$ / θ: PARAMETERS $\bar{w}$ OF REGRESSION

- WHEN Y IS BINARY → LOGISTIC REGRESSION IS BETTER THAN LINEAR REGRESSION

→ LINEAR FUNCTION ⟼ SIGMOID FUNCTION: $g(h) = \frac{1}{1+e^{-h}}$



$$L_y \begin{cases} p(Y=0|X;\theta) = g(w^T X) = \frac{1}{1+e^{w^T X}} \\ p(Y=1|X;\theta) = 1 - g(w^T X) = \frac{e^{w^T X}}{1+e^{w^T X}} \end{cases}$$

- HOW TO LEARN PARAMETERS $\theta = \bar{w}$? → MLE

→ $L(Y|X;w) = \prod_i (1 - g(x_i;w))^{y_i} \cdot g(x_i;w)^{1-y_i}$

- MLE: $\frac{\partial}{\partial w_i} L(Y|X;w) = 0$ DO NOT LEAD TO A CLOSED-FORM SOLUTION

↳ WE USE GRADIENT TO ADJUST w_i: GRADIENT ASCENT STEP

$$w^{(\tau)} \leftarrow w^{(\tau)} + \underbrace{\varepsilon}_{\text{SMALL CONSTANT}} \sum_{j=1}^{N} x_i^T \{ y_i - (1 - g(x_i;w)) \}$$

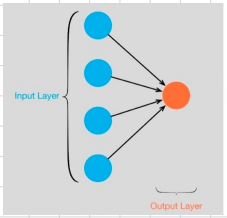
- **ALGORITHM FOR LOGISTIC REGRESSION:**

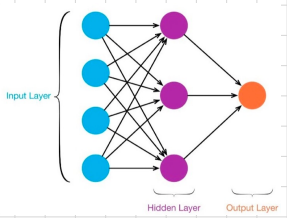
1. CHOOSE λ
2. START WITH A GUESS FOR $\bar{w}$
3. $\forall_\tau$. $w^{(\tau)} \leftarrow w^{(\tau)} + \varepsilon \sum_{j=1}^{N} x_i^T \{ y_i - (1 - g(x_i;w)) \}$
4. IF NO IMPROVEMENT FOR $\log L(Y|X;\bar{w})$ → STOP

↳ else: GO TO STEP 3.

- IF NOT ENOUGH SAMPLE → REGULARIZATION: $p(Y=1, \theta|X) \propto p(Y=1|X;\theta) p(\theta)$ (POSTERIOR; PRIOR INFO)

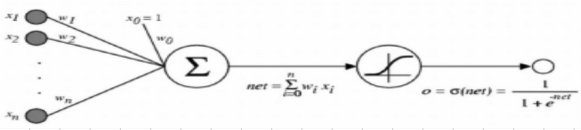
MULTI LAYER PERCEPTRON :





- 1 PERCEPTRON —> LINEAR BOUNDARIES
- MULTI LAYER PERCEPTRON —> MORE COMPLEX BOUNDARIES : NON-LINEAR

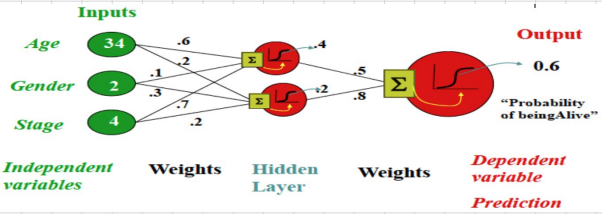
- GIVEN A PERCEPTRON :



-> WE NEED TO SOLVE THE PROBLEM :

$$\bar{w} = \arg\min_{\bar{w}} \sum \frac{1}{2} \left(Y_i - \hat{f}(X_i, \bar{w}) \right)^2$$

- PERCEPTRON LEARNING RULE :



- $\forall$ TRAINING EXAMPLE $d \in D$:

1. COMPUTE GRADIENT $\nabla E_d[\bar{w}]$:

- $E_d[\bar{w}]$: TOT ERROR FUNCTION = $\frac{1}{2} \sum \left(t_d - o_d \right)^2$

  TARGET OUTPUT, PREDICTED OUTPUT

  $\hookrightarrow \nabla E_d[\bar{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$

2. UPDATE $\bar{w}$ USING THE TRAINING RULE :

- $\bar{w} \leftarrow \bar{w} - \eta \cdot \nabla E_D[\bar{w}]$ / η : LEARNING RATE

  $\hookrightarrow \frac{\partial E_D[\bar{w}]}{\partial w_j} = \dots = -\sum_d (t_d - o_d) o_d (1 - o_d) x_d^i$ (PROOF)

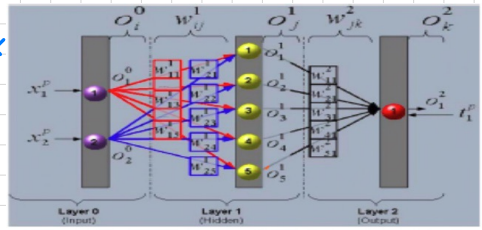
PROOF,

$$\begin{aligned}
\frac{\partial E_D[\bar{w}])}{\partial w_j} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d)(-\frac{\partial o_d}{\partial w_i}) \\
&= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_i} \frac{\partial net_d}{\partial w_i} \\
&= -\sum_d (t_d - o_d) o_d (1 - o_d) x_d^i
\end{aligned}$$

- **BACK PROPAGATION :** https://towardsdatascience.com/how-does-back-propagation-work-in-neural-networks-with-worked-example-bc59dfb97f48

ONCE THE NETWORK OUTPUT IS COMPUTED, AN ERROR IS GENERATED ON NN, BETWEEN THE PREDICTED OUTPUT AND THE TRUE OUTPUT

-> THIS ERROR IS USED TO ADJUST THE $\vec{W}$ OF THE NN.



- GENERAL $\vec{W}$ UPDATE :

$$W_{i,J} \leftarrow W_{i,J} + \underbrace{\eta \cdot \delta_J \cdot X^J}_{\text{UPDATE}}$$

J : INDEX OF LAYER

i : INDEX OF WEIGHT, IN LAYER J

* CONSIDERING THIS NN

- WHERE :

~> UPDATE IS DIFFERENT BASED ON IF $W_{i,J}$ ∈ OUTPUT / HIDDEN LAYER

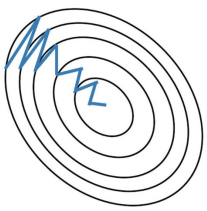
- $W_{i,J} \in$ OUTPUT UNIT : $\delta_J = \delta_K \leftarrow O_K^2 \left(1 - O_K^2\right)\left(t - O_K^2\right)$
- $W_{i,J} \in$ HIDDEN UNIT : $\delta_J = \delta_h \leftarrow O_h^1 \left(1 - O_h^1\right) \cdot \sum_{k \in \text{outputs}} w_{h,k} \cdot \delta_k$

- THIS BACK PROPAGATION IS DOING GRADIENT DESCENT OVER THE ENTIRE WEIGHTS OF THE NETWORK
- IT WILL FIND A LOCAL MINIMUM, NOT A GLOBAL ONE
- A NEW PARAMETER γ CALLED "WEIGHT MOMENTUM" CAN BE USED TO SPEED UP LEARNING AND NOT GET STUCK IN A LOCAL MINIMUM

->

$$\Delta w_{i,J} = \underbrace{\left(\eta \cdot \frac{\partial E}{\partial w_{i,J}}\right)}_{\text{BASIC UPDATE RULE}} + \left(\underbrace{\gamma}_{\text{WEIGHT MOMENTUM}} \cdot \underbrace{\Delta w_{i,J}^{(t-1)}}_{\text{WEIGHT UPDATE AT PREVIOUS ITERATION}}\right)$$

Stochastic Gradient Descent **withhout** Momentum

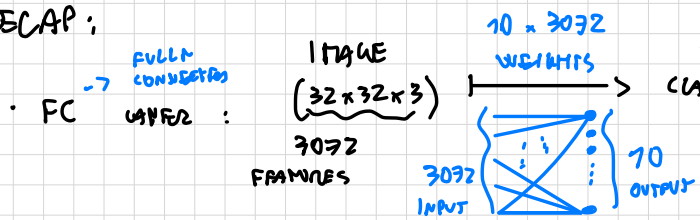


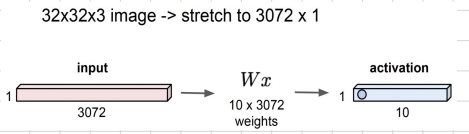
Stochastic Gradient Descent **with** Momentum

• CONVOLUTIONAL NEURAL NETWORKS (CNN):

• RECAP:

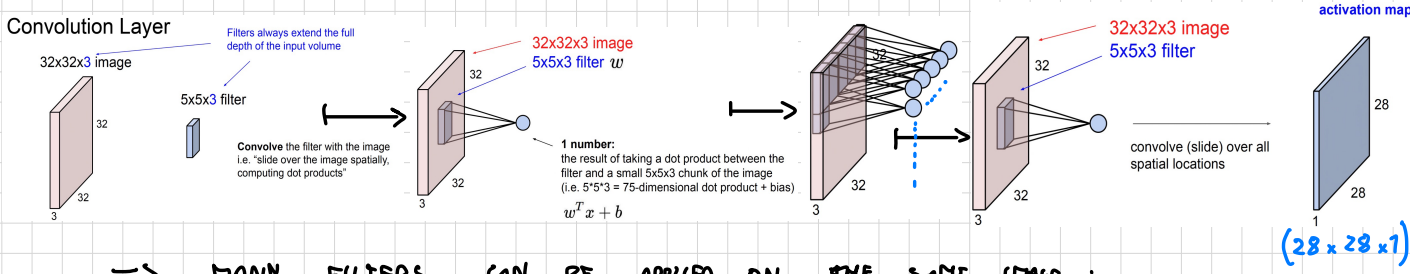
• FC (FULLY CONNECTED) LAYER:



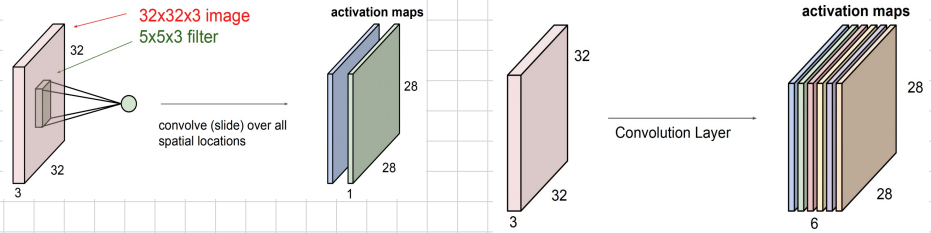


-> CONVOLUTIONAL LAYER:

A FILTER IS CONVOLVED OVER THE ORIGINAL IMAGE, AND THE FULL DEPTH OF THE INPUT VOLUME IS PRESERVED:

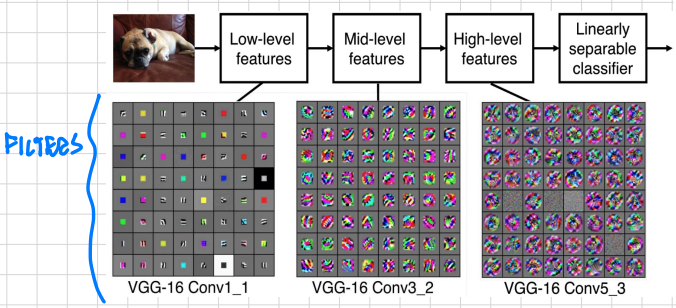


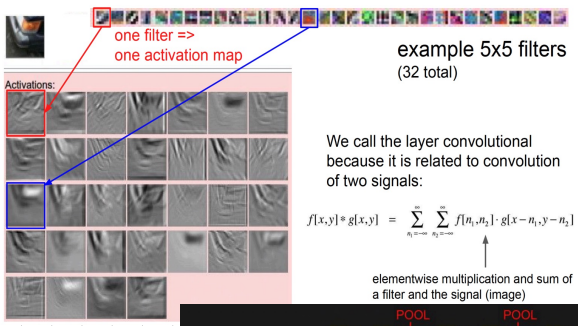
-> MANY FILTERS CAN BE APPLIED ON THE SAME IMAGE:

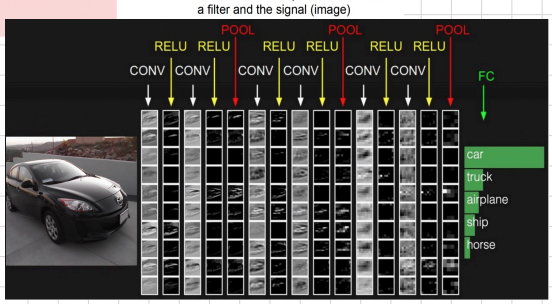


• SO, 6 FILTERS OF 5 x 5 x 3 APPLIED ON SAME IMAGE 32 x 32 x 3:
-> NEW "IMAGE" 28 x 28 x 6

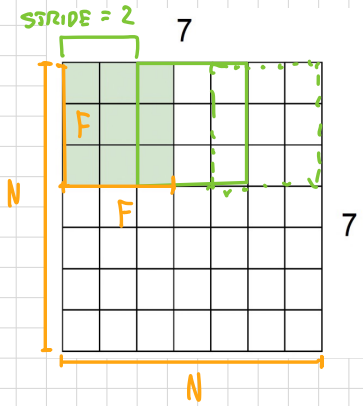
• AFTER A FILTER IS APPLIED AN ACTIVATION FUNCTION CAN BE USEFUL TO ACTIVATE ONLY MOST RELEVANT FEATURES:







- CLOSER LOOK AT SPATIAL DIMENSION:



ex.

- INPUT: 7 x 7
- FILTER: 3 x 3

- STRIDE: OFFSET BETWEEN SUCCESSIVE POSITIONS OF A FILTER
  - FILTER APPLIED WITH STRIDE = 2 ? -> OK: 3 x 3 OUTPUT
  - FILTER APPLIED WITH STRIDE = 3 ? -> NOT POSSIBLE

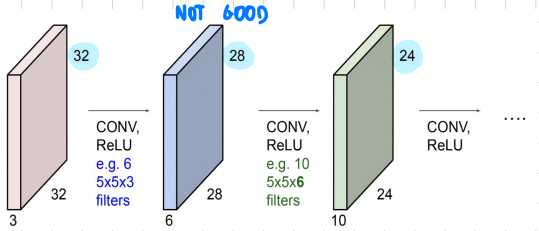
- IN GENERAL:

$$\text{OUTPUT SIZE} = \frac{(N - F)}{\text{STRIDE}} + 1$$ -> IF OUTPUT SIZE $\notin \mathbb{Z}$: NOT POSSIBLE

- IN PRACTICE IT IS COMMON TO ADD ZERO-PADDING TO THE BORDER
  - -> USUALLY: STRIDE = 1, $P = \frac{1}{2}(F - 1)$

$$\text{OUTPUT SIZE} = \frac{(N + 2P - F)}{\text{STRIDE}} + 1$$

... BUT SHRINKING TOO FAST IS NOT GOOD



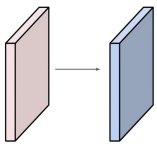
ex.

-> $N = 32, \; F = 5, \; \text{STRIDE} = 1, \; P = 2$

- DEPTH = 10 ∵ 10 FILTERS APPLIED
- $\text{OUTPUT SIZE} = \frac{(32 + 2 \cdot 2 - 5)}{1} + 1 = 32$

-> OUT SHAPE: $(32 \times 32 \times 10)$

Examples time:



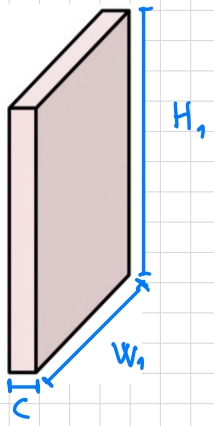
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2
5 x 5 x 3
Output volume size: ?

Number of parameters in this layer?

- PARAMETERS: VALUES IN THE FILTER TO ADJUST
  - -> $\forall$ FILTER: $5 \cdot 5 \cdot 3$ WEIGHTS + 1 BIAS = 76 PARAMETERS/FILTER
  - -> # FILTERS = 10 -> 760 PARAMETERS IN TOTAL

- SUMMARY:

ASSUME INPUT = $W_1 \times H_1 \times C$



- CONV. LAYER NEEDS 4 HYPERPARAMETERS:
  - K : n° OF FILTERS
  - F : SIZE OF FILTER
  - S : STRIDE
  - P : ZERO-PADDING

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

→ OUTPUT : $W_2 \times H_2 \times K$

$$W_2 = \frac{(W_1 + 2P - F)}{S} + 1$$

$$H_2 = \frac{(H_1 + 2P - F)}{S} + 1$$

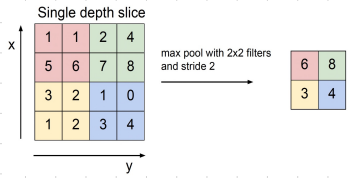
→ n° OF PARAMETERS : $F^2C \cdot K + K$ ↝ BIAS

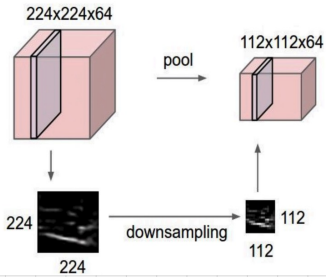
INTRO TO DEEP LEARNING 1

POOLING :

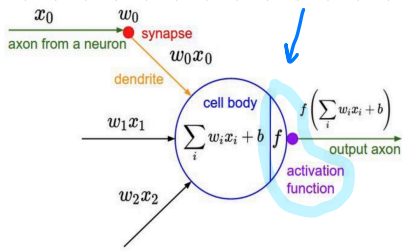
EXTRACTION OF ONLY MOST RELEVANT VALUES

eg. MAX - POOLING

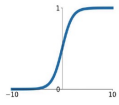




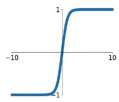
ACTIVATION FUNCTIONS :



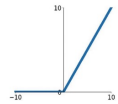
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



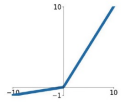
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

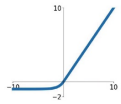


Leaky ReLU
$\max(0.1x, x)$



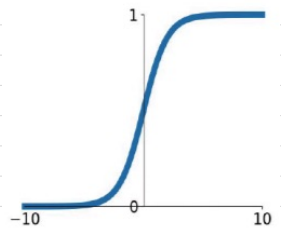
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



SIGMOID :

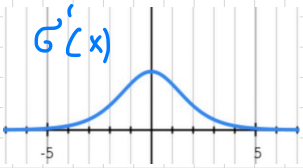
$$\sigma(x) = \frac{1}{1+e^{-x}} \quad , \quad \sigma : \mathbb{R} \longmapsto [0, 1]$$



- PROBLEMS :

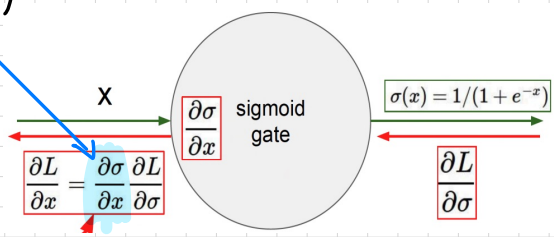
1. SATURATED NEURONS "KILL" THE GRADIENT :

- $\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$



-> WEIGHT CAN NEVER CHANGE

IF ALL GRADIENTS FLOWING BACK WILL BE 0



eg.

INPUT $X = -10$: $\sigma(x) \approx 0$, $\frac{d\sigma(x)}{dx} = 0 \cdot (1 - 0) = 0$

INPUT $X = 0$: $\sigma(x) \approx 1$, $\frac{d\sigma}{dx} = 1 \cdot (1 - 1) = 0$

INPUT $X = 10$: $\sigma(x) = 0$, $\frac{d\sigma}{dx} = 0 \cdot (1 - 0) = 0$

-> $\sigma(x)$ DOESN'T HELP IN LEARNING FAST : OFTEN $\frac{d\sigma}{dx}$ WILL BE 0

2. OUTPUT ARE NOT ZERO-CENTERED :

· CONSIDER WHAT HAPPENS WHEN ALL $X > 0$:

LOCAL GRADIENT OF $\sigma(x)$ IS ALWAYS >0

$$\frac{\partial L}{\partial w} = \sigma\left(\sum_i w_i x_i + b\right)\left(1 - \sigma\left(\sum_i w_i x_i + b\right)\right) x \times upstream\_gradient$$

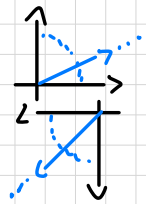
$x > 0$ BY Hp

→ $sign\left(\frac{\partial L}{\partial w}\right) = sign\left(\text{UPSTREAM GRADIENT}\right)$

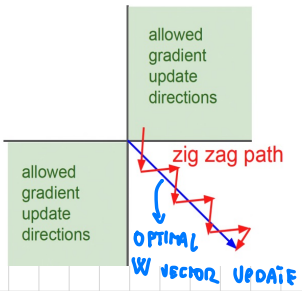
↳ GRADIENT UPDATE WILL HAVE ONLY 2 DIRECTIONS :

① ALL POSITIVE ;

② ALL NEGATIVE



→ THIS WILL LEAD TO A ZIG-ZAG PATH

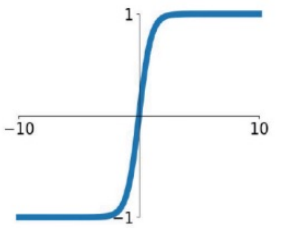


3. $\exp()$ IS A BIT COMPUTE EXPENSIVE

· $\tanh()$:

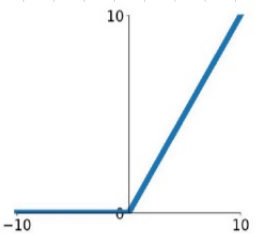
V: ZERO-CENTERED

S: KILLS GRADIENT WHEN SATURATED



· ReLU :

$ReLU(x) = \max(0, x)$, $RELU : \mathbb{R} \longmapsto [0, +\infty]$



V:

DO NOT SATURATE IN + REGION

COMPUTATIONALLY EFFICIENT

CONVERGES MUCH FASTER THEN $\sigma(x)$, $\tanh()$ $(6\times)$

S:

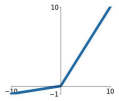
NOT ZERO-CENTERED

GRADIENT WHEN $X < 0$

- OTHERS:

EXPONENTIAL

LEAKY ReLU:



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x
- **will not "die".**

Leaky ReLU

$f(x) = \max(0.01x, x)$

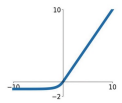
Parametric Rectifier (PReLU)

$f(x) = \max(\alpha x, x)$

backprop into \alpha (parameter)

• ELU:

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

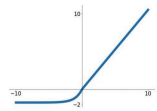
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

(Alpha default = 1)

- Computation requires exp()

• SELU:

Scaled Exponential Linear Units (SELU)



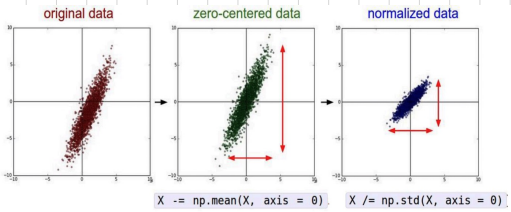
- Scaled versionof ELU that works better for deep networks
- "Self-normalizing" property;
- Can train deep SELU networks without BatchNorm
  - (will discuss more later)

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha(e^x - 1) & \text{otherwise} \end{cases}$$

$\alpha = 1.6733, \lambda = 1.0507$

- IN PRACTICE:
  - USE ReLU, IN CASE TRY LEAKY ReLU, ELU, SELU
  - DO NOT USE tanh(), GCX)

• DATA PREPROCESSING:



- NORMALIZATION : OR STANDARDIZATION $\frac{X - \mu}{\sigma}$

WHITENING

- PCA OR WHITENING: $X / \text{VarCoV matrix} \longmapsto X / \text{VarCoV matrix} = I$

↳ PRACTICE FOR IMAGES: SUBTRACT MEAN IMAGES, SUBTRACT PER CHANNEL MEAN AND /: STD

• WEIGHT INITIALIZATION:

- W = "SMALL RANDOM NUMBERS". OK FOR SMALL NETWORK, PROBLEMS WITH DEEPER NETWORKS

→ ALL ACTIVATIONS TEND TO ZERO FOR DEEPER NETWORK LAYERS

→ GRADIENT $\frac{\partial L}{\partial W} \longmapsto 0$ → NO LEARNING

- W = "XAVIER INITIALIZATION":

```
W = np.random.randn(Din, Dout) / np.sqrt(Din)
```

→ STD = $\sqrt{D_{IN}}$ → SIZE OF X / FOR CONV. LAYERS: $D_{IN} = F^2 \cdot$ INPUT_CHANNELS

↳ ACTIVATIONS ARE NICELY SCALED FOR ALL LAYERS

- IT DOES NOT WORK WELL FOR ReLU: XAVIER ASSUMES ZERO-CENTERED ACTIVATION

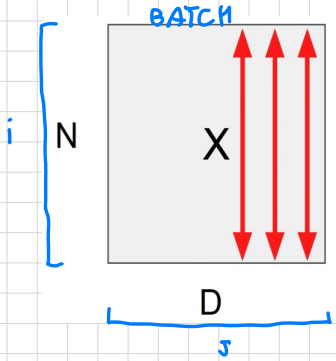
- W = "KAIMING / MSRA INITIALIZATION":

```
W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
```

→ STD = $\sqrt{\frac{2}{D_{IN}}}$ → IT SOLVES XAVIER PROBLEMS WITH NON-ZERO CENTERED FUNCTIONS

• BATCH NORMALIZATION :

INPUT X : $N \times D$



PER CHANNEL MEAN : $\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$ SHAPE $(D,)$

PER CHANNEL VAR : $\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$ SHAPE $(D,)$

-> NORMALIZED X : $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ SHAPE $(N \times D)$

• OUTPUT : $y_{i,j} = \gamma_j \cdot \hat{x}_{i,j} + \beta_j$ SHAPE $(N \times D)$

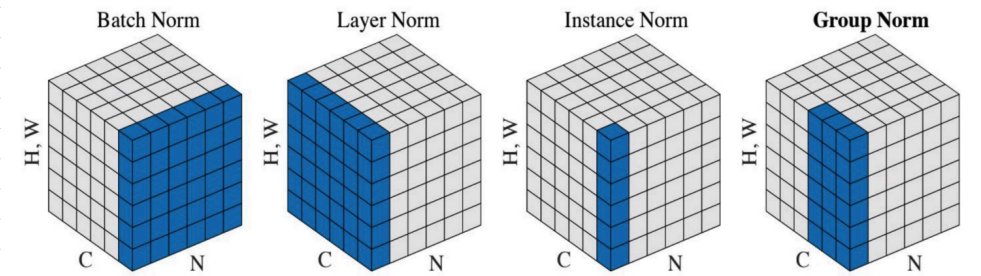
L-> LEARNING $\gamma = \sigma$, $\beta = \mu$ WILL RECOVER THE IDENTITY FUNCTION

• AT TEST TIME μ_j AND σ_j^2 COMES FROM VALUES SEEN DURING TRAINING

• BN IS USUALLY INSERTED AFTER FC OR CONVOLUTIONAL LAYERS AND BEFORE NON-LINEARITY (ACTIVATION FUNCTIONS)

V:
- IT MAKES DEEP NETWORKS MUCH EASIER TO TRAIN
- IMPROVEMENTS ON GRADIENT FLOW
- ALLOWS HIGHER LEARNING RATES -> FASTER CONVERGENCE
- NETWORKS MORE ROBUST TO INITIALIZATION

S:
- BEHAVES DIFFERENTLY DURING TRAINING AND TESTING (COMMON SOURCE OF BUGS)



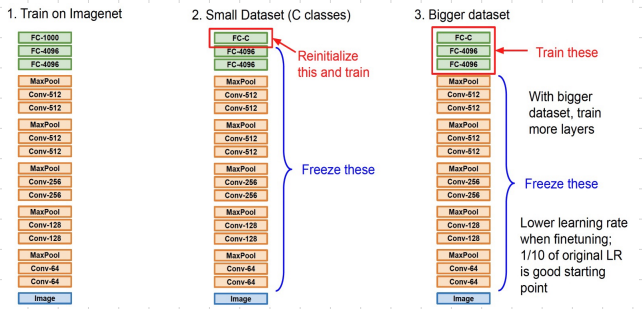
TRANSFER LEARNING :

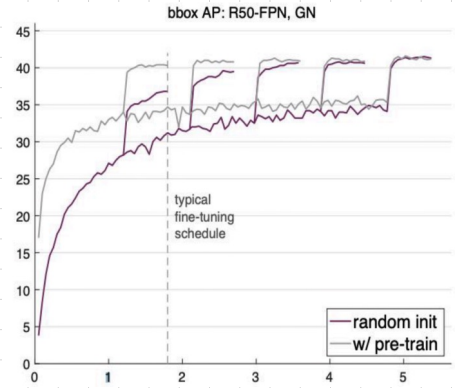
USE PARAMETERS LEARNT FROM A NN USED IN A TASK, INTO A NEW SIMILAR NN, FOR SIMILAR PURPOSES

- ex. TRAINING ON IMAGES :

1. TRAIN NN ON IMAGE NET

2.
   IF SMALL DATASET :
   -> FREEZE ALL LAYERS AND REINITIALIZE LAST ONE
   IF BIG DATASET :
   -> USE LOWER LEARNING RATE (ex. 1/10 η) AND REINITIALIZE AND TRAIN MORE LAYERS





Training from scratch can work just as well as training from a pretrained ImageNet model for object detection

But it takes 2-3x as long to train.

They also find that collecting more data is better than finetuning on a related task

INTRO TO DEEP LEARNING 2 :

LEARNING THEORY :

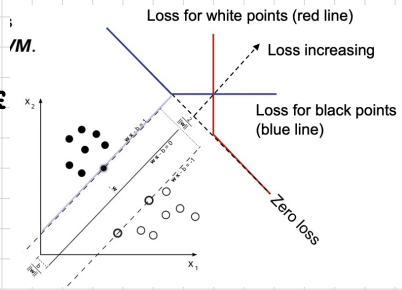
(SLIDES 1 - 32, COVERED AGREGOR IN MML)

OPTIMIZATION :

HINGE LOSS :

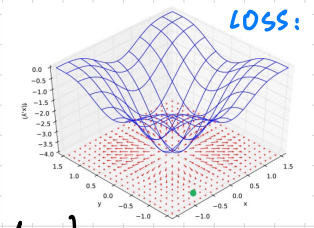
IN SVMs, IT PENALIZES DATA POINTS WHICH ARE INSIDE THE MARGIN. SVMs MINIMIZE HINGE LOSS

- $L = \max(0, 1 - t \cdot y(x))$

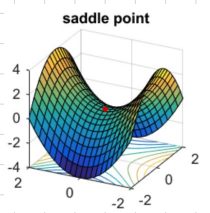


- IN ORDER TO REACH A MINIMUM IN LOSS WE NEED TO TAKE SMALL STEPS IN DIRECTION OF THE NEGATIVE GRADIENT : $-\nabla_w L(W)$

-> WEIGHTS UPDATE : $W^{(t+1)} = W^t - \alpha \cdot \nabla_w L(W)$ (LEARNING RATE)



L> GD PROBLEM : SADDLE POINTS -> WHERE $\nabla(\cdot)$ VANISHES

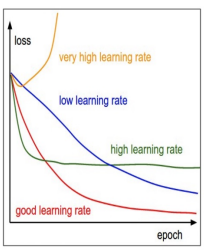


- GD IS OFTEN COMPUTED ON MINI-BATCHES, NOT FULL DATASET

L> THIS APPROACH IS CALLED SGD (STOCASTIC GRADIENT DESCENT)

L> IT IS "STOCASMIC" BECAUSE IT IS USED THE GRADIENT OF A MINI-BATCH INSTEAD OF THE ONE OF THE FULL DATASET

- MOMENTUM : TAKES INTO ACCOUNT PREVIOUS STEPS OF THE GRADIENT (E-I)
  - RMS PROP
  - ADA GRAD
  - ADAM



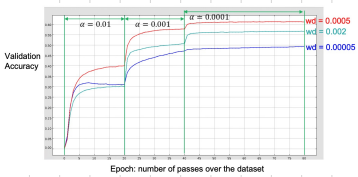
=> Learning rate decay over time!

step decay: e.g. decay learning rate by half every few epochs.

exponential decay: $\alpha = \alpha_0 e^{-kt}$

1/t decay: $\alpha = \alpha_0/(1+kt)$

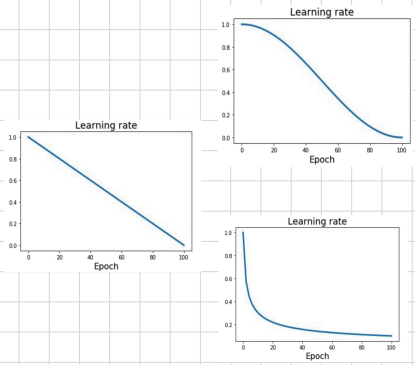
$1/\sqrt{t}$ decay (ADAGRAD): $\alpha = \alpha_0/\sqrt{1+kt}$



LEARNING RATE SCHEDULES:

IT IS USEFUL TO CHANGE LR DURING EPOCHS:

- COSINE: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\pi \frac{t}{T}\right)\right)$ — α_t: LR AT EPOCH t; α_0: INITIAL LR; T: TOT N OF EPOCHS
- LINEAR: $\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$
- INVERSE SQRT: $\alpha_t = \alpha_0 / \sqrt{t}$



- PRACTICAL TIP: BATCH SIZE += N -> LR *= N
- SECOND ORDER OPTIMIZATION: STEP TO MINIMUM OF PARABOLA

IMPROVE TEST ERRORS:

- STOP TRAINING MODEL WHEN ACC ON VALIDATION SET STARTS DECREASING
- MODEL ENSEMBLES: TRAIN MULTIPLE INDEPENDENT MODELS AND AVERAGE THEIR RESULTS AT TEST TIME
- IMPROVE SINGLE MODEL PERFORMANCE:
  - REGULARIZATION: L2, L1, ELASTIC NET (L2 + L1)
  - DROPOUT: V FORWARD PASS -> RANDOMLY SET SOME NEURONS TO 0
    - -> AT TRAINING: ADD SOME RANDOMNESS -> $y = f_W(x, z)$ — z: DROPOUT MASK
    - -> AT TESTING: AVERAGE OUT RANDOMNESS -> $y = f(x) = E_z[f(x, z)]$
  - DATA AUGMENTATION: CREATE NEW VERSION OF SAME DATA
    - L-> ex. IMAGE: HORIZONTAL FLIP, CROPS, CHANGE BRIGHTNESS, ecc.
  - DROPCONNECT: RANDOMLY SET SOME $W = 0$ IN NN AT TRAINING TIME

ecc. (SHOW 117 · 121)

CHOOSING HYPERPARAMETERS:

- STEPS:

1. CHECK INITIAL LOSS L

2. OVERFIT A SMALL SAMPLE:

   TRY TO TRAIN 100% ACC ON A SMALL SAMPLE OF DATA

   - $\downarrow L$: LR TOO LOW (LOSS NOT GOING DOWN)
   - $L \mapsto +\infty$ / NaN : LR TOO HIGH

3. FIND LR / $\downarrow L$:

   USING ARCHITECTURE FROM PREVIOUS STEP, USE 100% TRAINING DATA, TURN ON SMALL WEIGHT DECAY ($L2$, $L1$)

   -> FIND LR / $\downarrow L$ WITHIN ~ 100 ITERATIONS

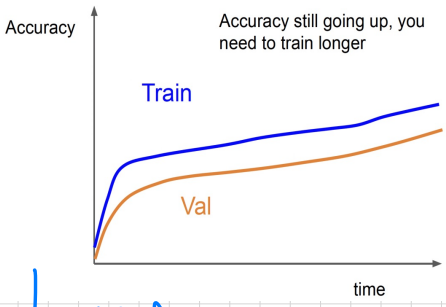
   -> ex. TRY LR = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}

4. CHOOSE SOME VALUES OF LR AND WEIGHT DECAY THAT WORKED WELL BEFORE AND CREATE A GRID, THEN TRAIN FOR 1-5 EPOCHS

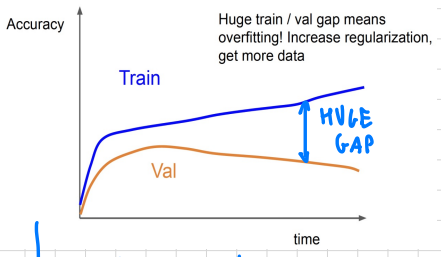
   -> ex. WEIGHT DECAY: 10^{-4}, 10^{-5}, 0

5. REDEFINE GRID AND TRAIN FOR LONGER ~ 10-20 EPOCHS, NO WEIGHT DECAY

6. LOOK AT LOSS AND ACC CURVES

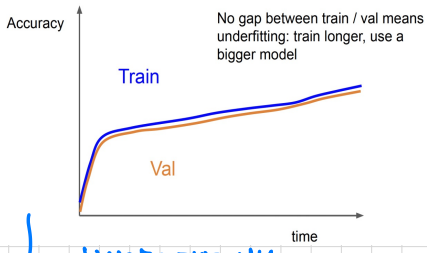


-> TODO: TRAIN LONGER



-> OVERFITTING

-> TODO: INCREASE REGULARIZATION, GET MORE DATA

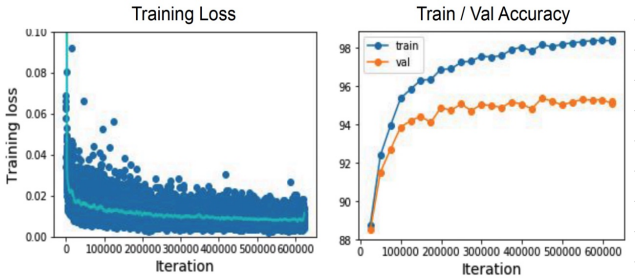


-> UNDERFITTING

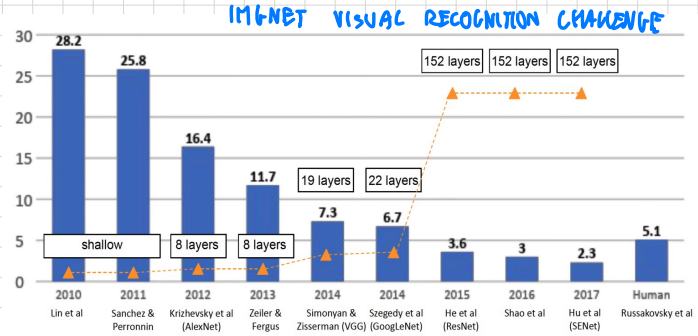
-> TODO: TRAIN LONGER, USE BIGGER MODEL

7. GOTO STEP 5

GOOD CURVES

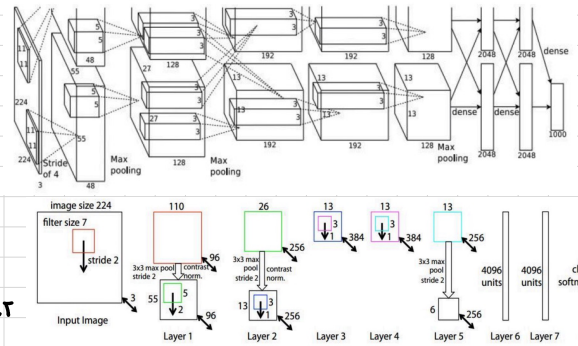


DEEP LEARNING ARCHITECTURES:



· ALEXNET (2012):

1ST CNN FOR IMGNET WITH BIG IMPROVEMENTS



· ZF NET (2013):

PARAMETERS IMPROVED OVER ALEXNET

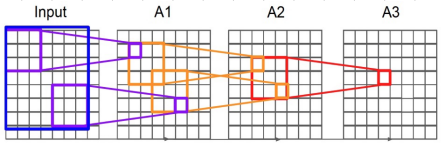
· VGG NET (2014):

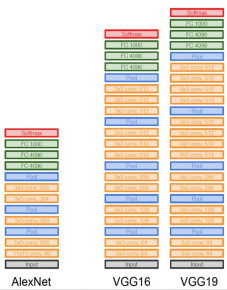
SMALLER FILTERS, DEEPER NETWORK

↳ 7x7 FILTERS ⟶ 3x3 FILTERS

→ 3x3 FILTERS HAS SAME EFFECTIVE RECEPTIVE FIELD AS 7x7 FILTERS

BUT LESS ⇝ PARAMETERS





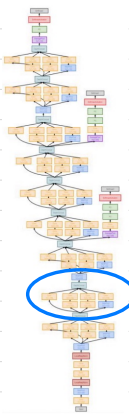
· GOOGLENET:

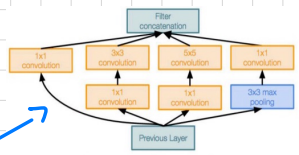
BUT MUCH MORE COMPLEXITY

DEEPER NETWORK, MORE COMPUTATIONAL EFFICIENCY

· INCEPTION MODULE: NETWORK IN A NETWORK

↳ FILTERS APPLIED IN PARALLEL, OUTPUT CONCATENATED CHANNEL-WISE



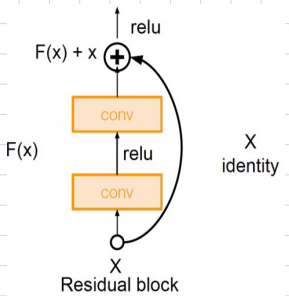


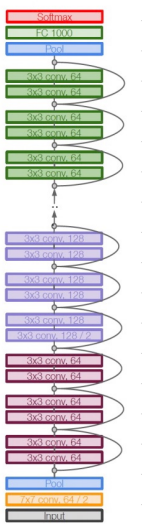
· RESNET: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec

· VERY BIG JUMP IN DEPTH: 22 LAYERS ⟶ 152 LAYERS

· USE OF RESIDUAL CONNECTIONS:

NETWORK TRIES TO FIT RESIDUAL MAPPINGS: LAYERS TRY TO FIT RESIDUALS, NOT TRUE OUTPUT





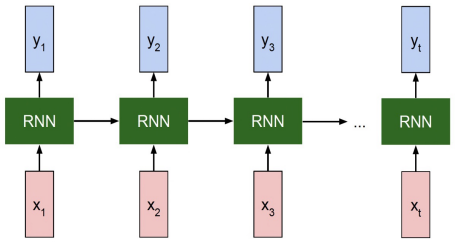
· SENET:

· SQUEEZE AN EXCITATION NETWORK:

ADDED A "FEATURE RECALIBRATION" MODULE: ADAPTIVELY RE-WEIGHTS FEATURE MAPS

- **RECURRENT NN (RNN):**

RNNs HAVE AN INTERNAL STATE THAT IS UPDATED AS A SEQUENCE IS PROCESSED



$$h_t = f_W(h_{t-1}, x_t)$$

new state — old state — input vector at some time step

some function with parameters W

$$y_t = f_{W_{hy}}(h_t)$$

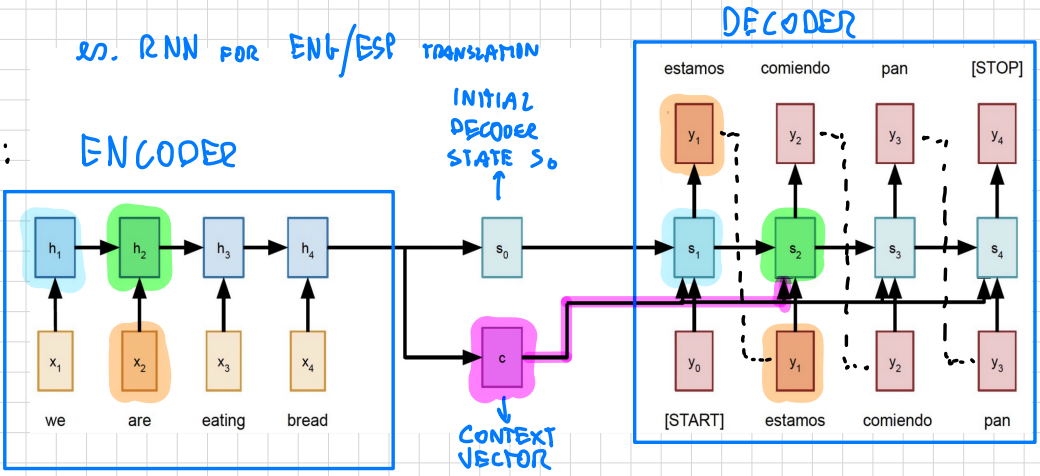
output — new state

another function with parameters W_o

- **TRANSFORMERS**:

- **RNN** (RECALL):

ex. RNN FOR ENG/ESP TRANSLATION



- INPUT SEQUENCE: $x_1, \dots, x_T$ / x_t: GENERIC INPUT

- **ENCODER**:

$\forall$ INPUT x_t -> STATE h_t COMPUTED: $h_t = f_w(x_t, h_{t-1})$ (CURRENT INPUT, PREVIOUS STATE)

-> WHEN FINAL h_T COMPUTED -> OUTPUTS = $\begin{cases} s_0: \text{INITIAL DECODER STATE} \\ C: \text{CONTEXT VECTOR} \end{cases}$

/ OFTEN $C = h_T$, C HAS FIXED LENGTH -> BOTTLENECK

- **DECODER**:

OUTPUT SEQUENCE: $y_1, \dots, y_T$

-> y_t GENERATED FROM $s_t = g_v(y_{t-1}, s_{t-1}, c)$

L-> RNN PROBLEM: C HAS FIXED LENGTH

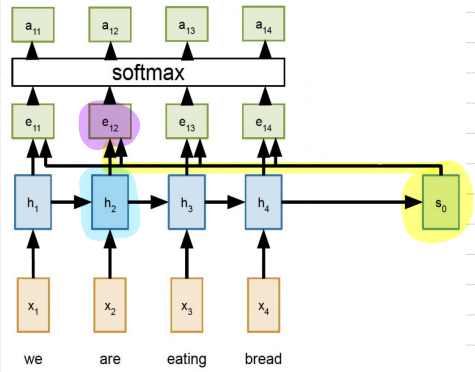
L-> SOL: USE NEW CONTEXT VECTOR $\forall$ STEP OF DECODER

-> A SINGLE C VECTOR REQUIRES TO SUMMARIZE WHICH WORDS ARE MORE IMPORTANT THAN OTHERS IN A FIXED-LENGTH REPRESENTATION

-> IT IS BETTER TO HAVE $\forall y_t$ -> C: DIFFERENT SUMMARY OF INPUT SENTENCE $\forall y_t$

ATTENTION CONCEPT :

- ALIGNMENT SCORES (scalar): $e_{t,i} = f_{ATT}(s_{t-1}, h_i)$ — f_{ATT}: MLP
  - ↳ changes based on start too



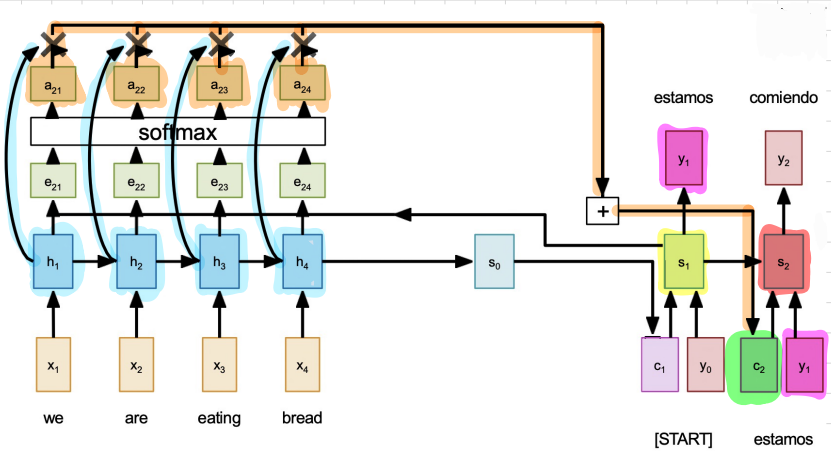
- ATTENTION WEIGHTS :

$$\alpha_{t,i} = \text{softmax}(e_{t,i}) = \frac{\exp(e_{t,i})}{\sum_i \exp(e_{t,i})}$$

- CONTEXT VECTOR :

$$c_t = \sum_i \alpha_{t,i} \cdot h_i$$

- IT ATTENDS THE RELEVANT PART OF THE INPUT SEQUENCE



EX.

"ESTAMOS" = "WE ARE" → IT IS POSSIBLE THAT :

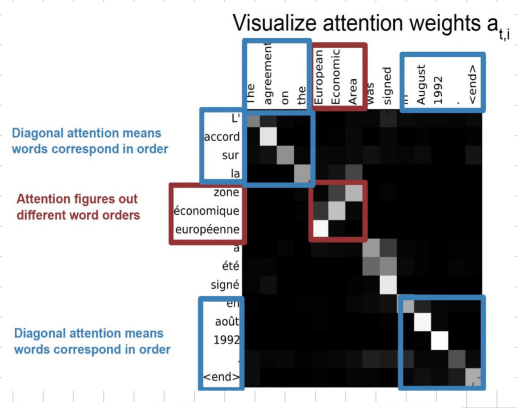
FOR y_1:

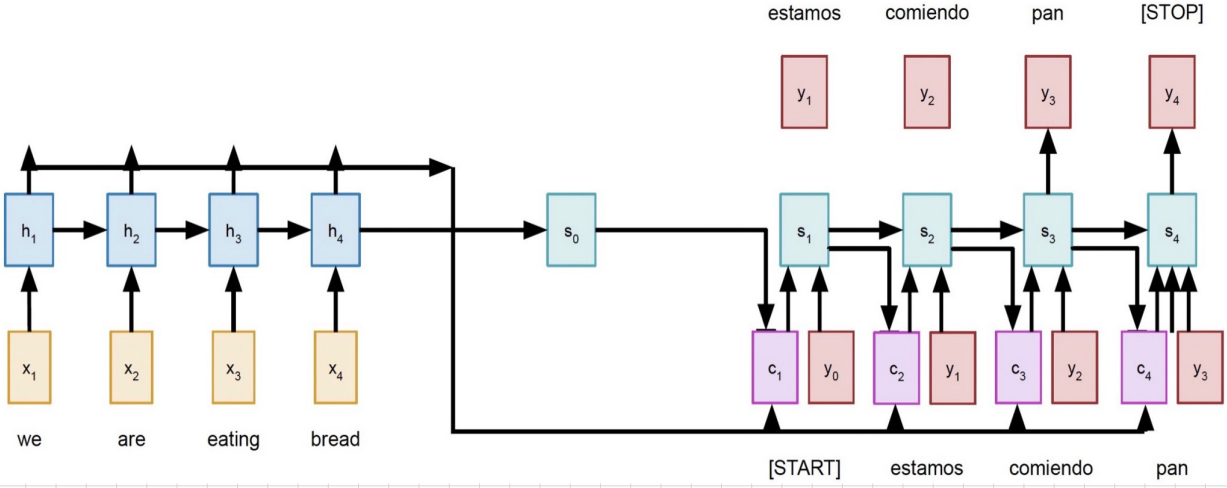
$$\begin{cases} \text{WE} \quad \text{ARE} \\ \alpha_{11} = \alpha_{12} = 0.45 \\ \text{EATING} \quad \text{BREAD} \\ \alpha_{13} = \alpha_{14} = 0.05 \end{cases}$$

- DECODER :

$$s_t = y_w(y_{t-1}, s_{t-1}, c_t)$$

DIFFERENT FROM BASIC RNN





- IMAGE CAPTIONING WITH RNN AND ATTENTION : (SKIPPED, SEE SLIDES 26 - 37)

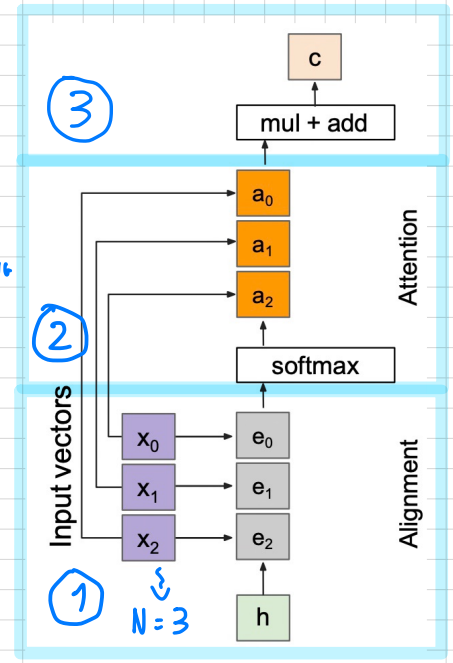
- TRANSFORMERS ARCHITECTURE:

IDEA: GET RID OF RNN, JUST USE ATTENTION CONCEPT ~> "ATTENTION IS ALL YOU NEED", 2017

- GENERAL ATTENTION LAYER:

INPUT SEQUENCE = $\{ x_0, x_1, x_2 \}$ (x_0: word 0, x_1: ...)

1. h : QUERY VECTOR, LAST DECODER STATE s_t IN TRANSLATION EXAMPLE (s_t: in gen.)
   - h.SHAPE = D
   - INPUT VECTORS X.SHAPE = $N \times D$ (# SAMPLE, # FEATURES, V WORD -> EMBEDDING)
   -> ALIGNMENT : $e_i = f_{ATT}(h, x_i)$
2. ATTENTION : $a = \text{SOFTMAX}(e)$ ~> NORMALIZATION
3. OUTPUT CONTEXT VECTOR : $c = \sum_i a_i \cdot x_i$
   - c.SHAPE = D



- N.B.: BY NOW, ATTENTION MECHANISM : $h \longmapsto c$ / c IS ALMOST A LINEAR FUNCTION OF INPUTS
  -> "ATTENTION IS ALL YOU NEED" PAPER PROPOSED SOME CHANGES IN ORDER TO HAVE A FLEXIBLE NON-LINEAR MAPPING

- "ATTENTION IS ALL YOU NEED" CONTRIBUTIONS:

- ALIGNMENT:

$$e_i = f_{ATT}(h, x_i) \longmapsto e_i = \frac{h \cdot x_i}{\sqrt{D}}$$

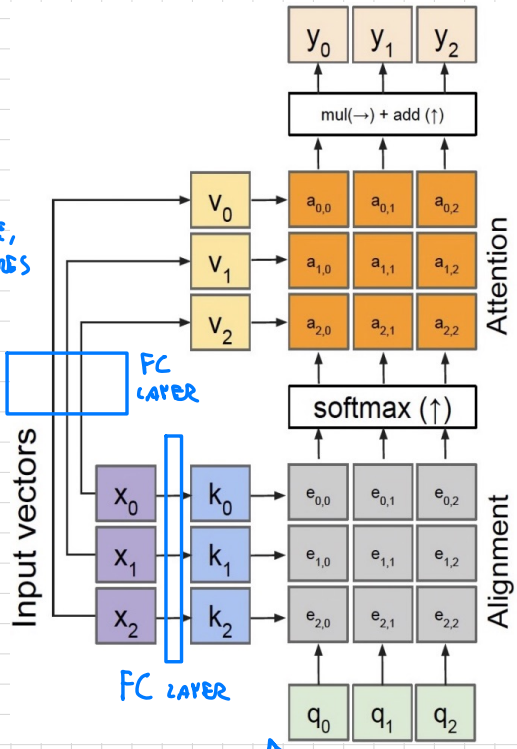
↳ C. SHAPE, # FEATURES

- f_{ATT}: MLP $\longmapsto$ $\langle \dots, \dots \rangle$ DOT PRODUCT
  - ↳ ONLY WORKS FOR WITH KEY & VALUE TRANSFORMATION TRICK
- DIVIDING BY D REDUCES EFFECT OF LARGE MAGNITUDE VECTORS

- MULTIPLE QUERY VECTORS:

$\bar{h}$ (SHAPE D) $\longmapsto$ $\bar{q}$ (SHAPE $M \times D$)

OUTPUTS: $\bar{c} \longmapsto \bar{r} = \{r_0, r_1, r_2\}$



- N.B.:

IN THIS WAY: FROM $\bar{x} \longmapsto e_i, q_i$

-> IT IS POSSIBLE TO ADD MORE EXPRESIVITY TO THE INPUT ADDING FC LAYERS BEFORE ALIGNMENT AND ATTENTION:

$\bar{x}$ —FC→ KEY VECTOR: $\bar{k} = \bar{x} \cdot W_k$, $\bar{x} \xrightarrow{FC} \bar{k} \longmapsto$ ALIGNMENT e_i

$\bar{x}$ —FC→ VALUE VECTOR: $\bar{v} = \bar{x} \cdot W_v$, $\bar{x} \xrightarrow{FC} \bar{v} (+ \bar{c}) \longmapsto$ ATTENTION a_i

↳ USING KEY AND VALUES WE GET:

- q. SHAPE $= M \times D \longmapsto q$. SHAPE $= M \times D_K$ (QUERIES)
- OUTPUTS: r. SHAPE $= D \longmapsto r$. SHAPE $= M \times D_V$

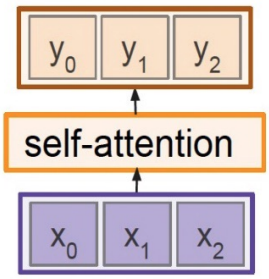
NOW INPUT AND OUTPUT DO NOT NEED TO HAVE SAME DIMENSIONS -> THEY CHANGE BASED ON $\bar{K}$ AND $\bar{V}$

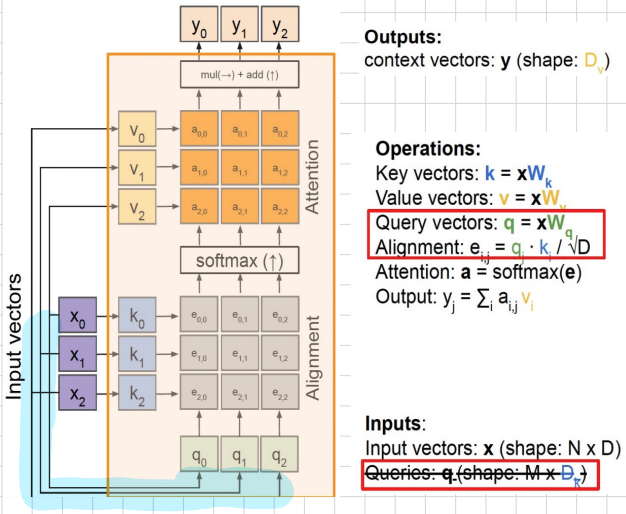
↳ $r_j = \sum_i a_{i,j} \cdot v_i$

- ALIGNMENT: $e_i = \frac{h \cdot x_i}{\sqrt{D}} \longmapsto e_i = \frac{q_i \cdot k_i}{\sqrt{D}}$

- **SELF ATTENTION LAYER:**

$\bar{q}$ IS CALCULATED DIRECTLY FROM $\bar{x}$





- **ATTENTION IN FORMULAS:**

ATTENTION : $A = \text{SOFTMAX}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$ ⇝ HERE $d_k \approx D_k$

- LET'S CONSIDER $D = 3$ → 3 QUERY $\bar{q}$, 3 KEY VECTOR $\bar{k}$, 3 VALUE VECTOR $\bar{v}$

$$Q = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{pmatrix} = \begin{pmatrix} \boldsymbol{q_1} \\ \boldsymbol{q_2} \\ \boldsymbol{q_3} \end{pmatrix} \quad K = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} = \begin{pmatrix} \boldsymbol{k_1} \\ \boldsymbol{k_2} \\ \boldsymbol{k_3} \end{pmatrix} \quad V = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix} = \begin{pmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \\ \boldsymbol{v_3} \end{pmatrix}$$

(# QUERIES = M; D_K ⇝ # FEATURES; $M \times D_K$; 3 QUERIES; $M \times D_K$; $M \times D_V$; D_V)

- **ALIGNMENT:**

$e_i = q_i \cdot k_i$ / IN MATRIX : $E = Q \cdot K^T$

↳ $e_{i,j} = \langle \bar{q}_{i1}, \bar{k}_{i2} \rangle$ (ROWS OF Q AND K)

$$QK^T = \begin{pmatrix} \boldsymbol{q_1} \cdot \boldsymbol{k_1} & \boldsymbol{q_1} \cdot \boldsymbol{k_2} & \boldsymbol{q_1} \cdot \boldsymbol{k_3} \\ \boldsymbol{q_2} \cdot \boldsymbol{k_1} & \boldsymbol{q_2} \cdot \boldsymbol{k_2} & \boldsymbol{q_2} \cdot \boldsymbol{k_3} \\ \boldsymbol{q_3} \cdot \boldsymbol{k_1} & \boldsymbol{q_3} \cdot \boldsymbol{k_2} & \boldsymbol{q_3} \cdot \boldsymbol{k_3} \end{pmatrix} = \overset{j}{\underset{i}{=}} \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} = \begin{pmatrix} \boldsymbol{e_1} \\ \boldsymbol{e_2} \\ \boldsymbol{e_3} \end{pmatrix} \quad (\bar{e}_i)$$

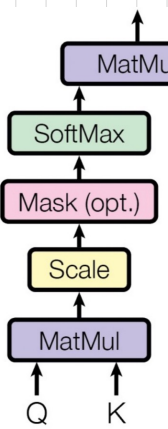
- **ATTENTION:**

$A = \text{SOFTMAX}\left(\frac{E}{\sqrt{d_k}}\right) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{pmatrix}$ ($\bar{a}_i$)

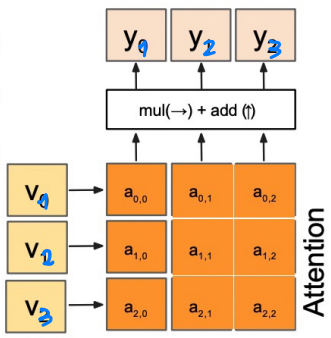
- **OUTPUTS VALUES:**

$\bar{v}_{i,j} = \langle \bar{a}_j, V_i \rangle$ ($\bar{v}_i$)

$$\boldsymbol{y_1} = a_{11}\boldsymbol{v_1} + a_{12}\boldsymbol{v_2} + a_{13}\boldsymbol{v_3}$$
$$\boldsymbol{y_2} = a_{21}\boldsymbol{v_1} + a_{22}\boldsymbol{v_2} + a_{23}\boldsymbol{v_3}$$
$$\boldsymbol{y_3} = a_{31}\boldsymbol{v_1} + a_{32}\boldsymbol{v_2} + a_{33}\boldsymbol{v_3}$$



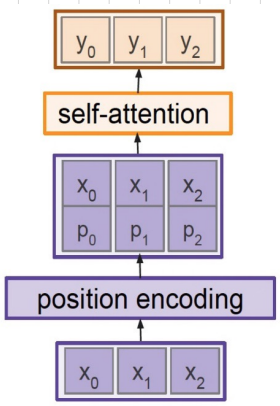
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



- **POSITIONAL ENCODING :**

BY THEMSELVES

USED TO ENCODE ORDERED SEQUENCES, SINCE TRANSFORMERS ARE PERMUTATION EQUIVARIANT



- $\forall x_j \rightarrow \exists!$ POSITION $p_j = pos(j)$

SCALAR

$pos : \mathbb{N} \longmapsto \mathbb{R}^d$ ~> OUTPUT IS A d-DIMENSIONAL VECTOR

- A LOOK UP TABLE NEEDS TO BE LEARNT / LUT. SHAPE = $T \times D$

↳ GIVEN AN INPUT TIME t, IT OUTPUTS ITS d-DIMENSIONAL POSITION

JUST 1 DIMENSION

$$\rightarrow pos(t) = \begin{cases} \begin{bmatrix} \sin(w_1 \cdot t) \\ \cos(w_1 \cdot t) \end{bmatrix}, \begin{bmatrix} \sin(w_2 \cdot t) \\ \cos(w_2 \cdot t) \end{bmatrix}, \dots, \begin{bmatrix} \sin(w_{d/2} \cdot t) \\ \cos(w_{d/2} \cdot t) \end{bmatrix} \end{cases} \Big/ w_k = \frac{1}{10000^{\frac{2k}{d}}}$$

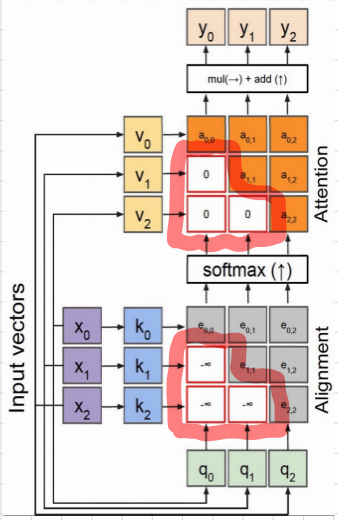
$1 \times D$ (D)

- **MASKED SELF-ATTENTION LAYER :**

IT PREVENTS VECTORS TO LOOK AT FUTURE VECTORS

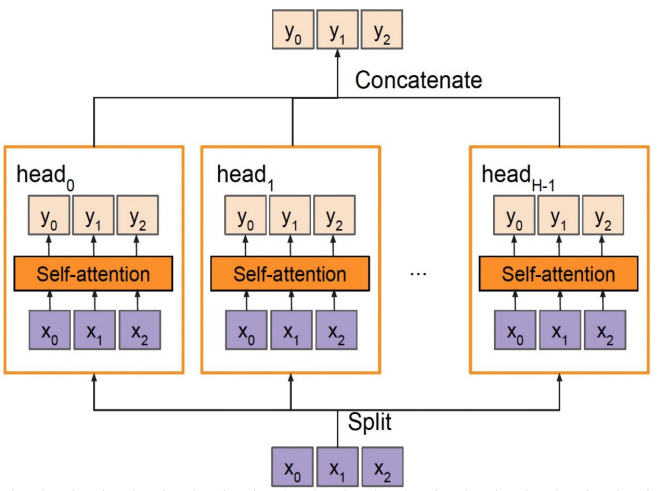
- SOME $e_{i,j} = -\infty$

$\rightarrow \alpha_{i,j} = 0$

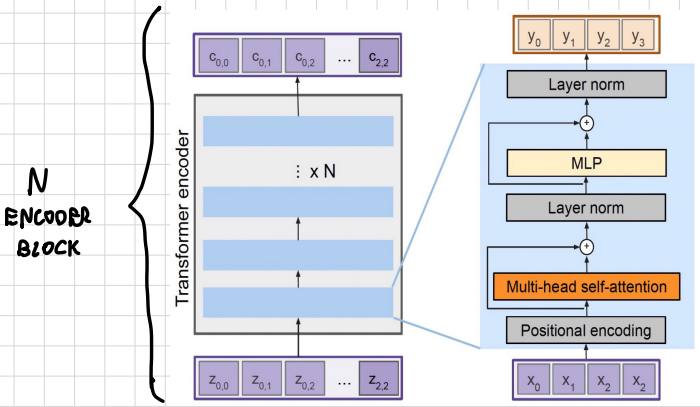


- **MULTI-HEAD SELF-ATTENTION LAYER :**

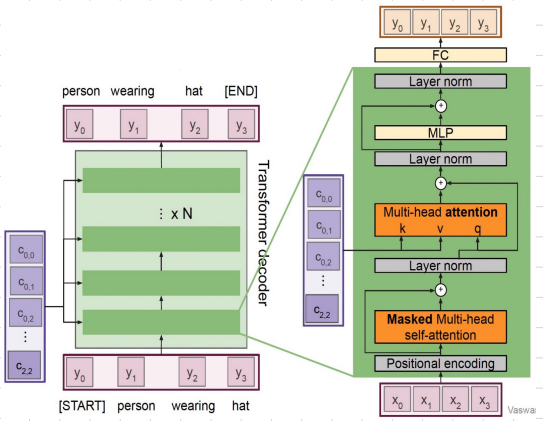
MULTIPLE SELF-ATTENTION HEADS IN PARALLEL



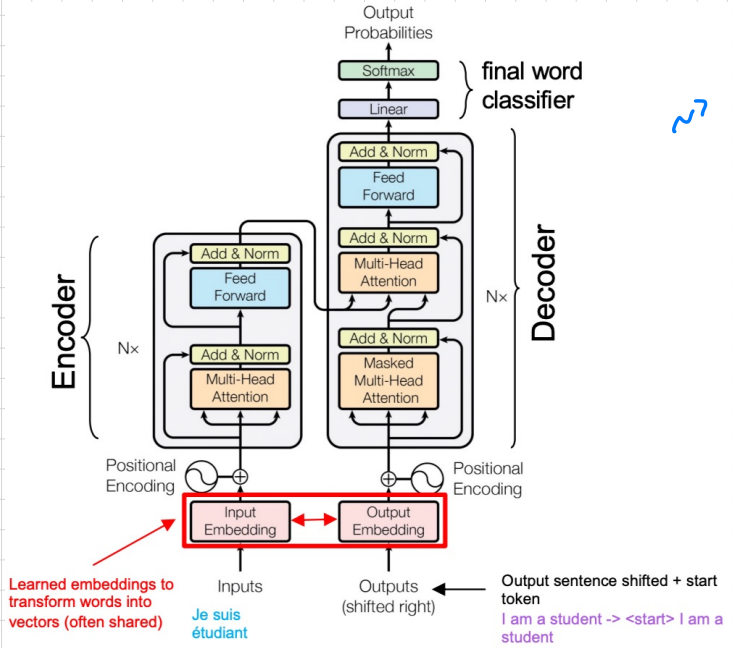
• TRANSFORMER ENCODER BLOCK:



• TRANSFORMER DECODER BLOCK:



• FULL ARCHITECTURE:



"ATTENTION IS ALL YOU NEED"

• RNNs VS TRANSFORMERS:

RNNs
(+) LSTMs work reasonably well for long sequences.
(-) Expects an ordered sequences of inputs
(-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

Transformer:
(+) Good at long sequences. Each attention calculation looks at all inputs.
(+) Can operate over unordered sets or ordered sequences with positional encodings.
(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
(-) Requires a lot of memory: N x M alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

HELPFUL GUIDE → https://jalammar.github.io/illustrated-transformer/