so the formula of perceptron trick

In ML literature, **"perceptron trick" usually means the *kernelized perceptron*** (analogous to the kernel trick in SVMs).
Below are the **exact formulas**, clean and mathematical.

---

1. Standard Perceptron (Primal Form)

Model

$$\hat{y} = \text{sign}(w^\top x + b)$$

Update rule (for a misclassified point $(x_i, y_i),\ y_i \in \{-1, +1\}$)

$$\text{if } y_i(w^\top x_i + b) \le 0: \quad \begin{cases} w \leftarrow w + \eta y_i x_i \\ b \leftarrow b + \eta y_i \end{cases}$$

---

2. Dual Representation of Perceptron

Initialize $\alpha_i = 0\ \forall i$

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Decision function

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i^\top x) + b$$

Update

$$\text{if } y_i f(x_i) \le 0: \quad \alpha_i \leftarrow \alpha_i + \eta$$

3. Perceptron Trick (Kernelized Perceptron)

Key substitution (the "trick")

$$x_i^\top x \quad \Rightarrow \quad K(x_i, x)$$

where

$$K(x_i, x) = \phi(x_i)^\top \phi(x)$$

Kernelized Decision Function

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

Kernelized Update Rule

$$\text{if } y_i f(x_i) \le 0: \quad \alpha_i \leftarrow \alpha_i + \eta$$

(no explicit w, no explicit $\phi(x)$)

4. Common Kernels Used in Perceptron Trick

- **Linear**

$$K(x, z) = x^\top z$$

- **Polynomial**

$$K(x, z) = (x^\top z + c)^d$$

- **RBF (Gaussian)**

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

---

5. Final Compact Formula (What You Should Remember)

$$\boxed{f(x) = \sum_{i \in \mathcal{M}} \alpha_i y_i K(x_i, x) + b}$$

where
$\mathcal{M}$ = indices of misclassified samples (support vectors of perceptron).

If you want next:

- full **mathematical derivation** (primal → dual → kernel)
- **comparison with SVM kernel trick**
- **why perceptron has no closed-form solution**

tell me which one.

look these equations and tell me what they are

I carefully looked at your handwritten page.
Those equations are **all variants of the *Perceptron learning rule*** written in **different equivalent forms**.

I'll map **each part of your notes → its exact mathematical meaning**, no fluff.

---

1. `∑ w_i x_i ≥ 0` and `∑ w_i x_i < 0`

These are **decision conditions** of the perceptron.

You are using labels:

$$y \in \{+1, -1\}$$

Decision rule:

$$\hat{y} = \begin{cases} +1 & \text{if } \sum_i w_i x_i \geq 0 \\ -1 & \text{if } \sum_i w_i x_i < 0 \end{cases}$$

This is equivalent to:

$$\hat{y} = \text{sign}(w^\top x)$$

---

2. `w_{n+1} = w_0 + \eta x_i`

This is the **perceptron weight update for a positive class error**.

Written properly:

$$\text{If } y_i = +1 \text{ and misclassified:} \quad w \leftarrow w + \eta x_i$$

This comes from the general rule:

$$w \leftarrow w + \eta y_i x_i$$

with $y_i = +1$.

---

3. `w_{n+1} = w_0 - \eta x_i`

This is the **perceptron weight update for a negative class error**.

$$\text{If } y_i = -1 \text{ and misclassified:} \quad w \leftarrow w - \eta x_i$$

Again from:

$$w \leftarrow w + \eta y_i x_i$$

with $y_i = -1$.

---

4. Boxed Formula

`w_n = w_0 + η (y_i - \hat y_i) x_i`

This is a **generalized perceptron update**, often seen when perceptron is written in a **regression-like form**.

- y_i = true label
- $\hat{y}_i$ = predicted label
- $(y_i - \hat{y}_i) \in \{-2, 0, +2\}$

Interpretation:

- If prediction is correct → update = 0
- If wrong → pushes weights in the correct direction

This is mathematically equivalent to:

$$w \leftarrow w + \eta y_i x_i$$

for misclassified points.

---

5. The "Perceptron Trick" Arrow

When you wrote "**perceptron trick**", what it refers to is:

$$w = \sum_{i \in \mathcal{M}} \alpha_i y_i x_i$$

and later:

$$x_i^\top x \;\Rightarrow\; K(x_i, x)$$

That leads to the **kernelized perceptron**:

$$f(x) = \sum_i \alpha_i y_i K(x_i, x)$$

---

6. Final Classification of What You Wrote

Expression in notebook	What it is
`∑ w_i x_i ≥ 0`	Perceptron decision boundary
`w + ηx`	Update for misclassified positive sample
`w − ηx`	Update for misclassified negative sample
`w = w + η(y − ŷ)x`	Unified perceptron update
"perceptron trick"	Dual + kernel formulation

One-line summary (important)

> **Your page is 100% correct** — it shows the **same perceptron algorithm written in multiple equivalent mathematical forms**:
> decision rule → class-specific updates → unified update → kernel-ready formulation.

If you want next:

- strict **primal → dual derivation**
- **difference between perceptron update and SGD**
- **why logistic regression replaces perceptron**

tell me which one.

what i gave what exactly that is

What you gave is **one single algorithm written in multiple equivalent mathematical forms**.

Exact answer (no ambiguity):

> You gave the ***Perceptron Learning Algorithm*** — specifically the ***online / mistake-driven binary linear classifier***.

More precisely, your page contains these exact components:

1. **Linear decision function**

$$f(x) = \sum_i w_i x_i \quad (\text{sign decides the class})$$

2. **Binary classification rule**

$$\hat{y} = \text{sign}(w^\top x)$$

3. **Mistake-driven update**

$$\text{update only if } y_i \neq \hat{y}_i$$

4. **Class-dependent updates**

$$y = +1 : \ w \leftarrow w + \eta x$$
$$y = -1 : \ w \leftarrow w - \eta x$$

5. **Unified update rule**

$$\boxed{w \leftarrow w + \eta(y_i - \hat{y}_i)x_i}$$

What it is NOT

- ❌ Not logistic regression
- ❌ Not SVM
- ❌ Not gradient descent on a smooth loss
- ❌ Not backpropagation

Proper textbook name

Depending on context, it is called:

- **Perceptron Learning Rule**
- **Rosenblatt's Perceptron (1958)**
- **Online linear classifier**
- **Mistake-driven update rule**
- **Primal perceptron**

One-line identity (remember this)

$$\boxed{\text{Your notes} = \text{classical perceptron algorithm in primal form}}$$

If you want, next I can:

- rewrite it in **pure optimization form**
- show **why no loss function exists**
- convert it into **logistic regression mathematically**
- connect it to **SGD formally**

Just say which.

rewrite it in pure optimization form

show why no loss function exists

convert it into logistic regression mathematically

Below is **exactly what you asked**, in **three clean mathematical blocks**, with **no intuition, no storytelling** — only formal math.

I. Perceptron in Pure Optimization Form

Problem Setup

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\}$$

Hard Classification Constraint

$$y_i(w^\top x_i) \; > \; 0 \quad \forall i$$

Feasibility Optimization (Perceptron View)

The perceptron is **not** solving a minimization of a smooth objective.
It solves a **constraint satisfaction problem**:

$$\boxed{\text{Find } w \in \mathbb{R}^d \quad \text{s.t.} \quad y_i w^\top x_i \geq 0 \;\; \forall i}$$

This is a **feasibility problem**, not a minimization.

Algorithmic Projection Form

Define violation indicator:

$$\mathbb{I}_i(w) = \begin{cases} 1 & \text{if } y_i w^\top x_i \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron update:

$$\boxed{w_{t+1} = w_t + \eta \, y_i x_i \quad \text{if } \mathbb{I}_i(w_t) = 1}$$

This is a **projected correction**, not gradient descent.

II. Why No Proper Loss Function Exists (Formally)

Assume there exists a differentiable loss $L(w)$ such that:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

But perceptron update is:

$$\Delta w = \begin{cases} \eta y_i x_i & \text{if } y_i w^\top x_i \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

This implies:

$$\nabla L(w) = \begin{cases} -y_i x_i & \text{if } y_i w^\top x_i \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

Contradiction

- Gradient is **discontinuous**
- Gradient does **not depend smoothly on** w
- No scalar $L(w)$ exists whose gradient jumps discretely

Formal Statement

$$\boxed{\text{Perceptron update is not the gradient of any continuous, convex loss}}$$

It is **rule-based**, not variational.

(Closest Surrogate — Not True Loss)

The *perceptron loss*:

$$\ell_{\text{perc}}(w; x, y) = \max(0, -yw^\top x)$$

is **non-differentiable** and does **not produce the classical update** exactly.

III. Converting Perceptron → Logistic Regression (Mathematically)

Step 1: Replace Hard Constraint with Soft Margin

Perceptron:

$$yw^\top x \geq 0$$

Logistic:

$$P(y|x) = \sigma(yw^\top x)$$

where

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

Step 2: Likelihood Formulation

$$\mathcal{L}(w) = \prod_{i=1}^{n} \sigma(y_i w^\top x_i)$$

Negative log-likelihood:

$$\boxed{J(w) = \sum_{i=1}^{n} \log\left(1 + e^{-y_i w^\top x_i}\right)}$$

This is the **logistic loss**.

Step 3: Gradient of Logistic Loss

$$\nabla J(w) = -\sum_{i=1}^{n} \frac{y_i x_i}{1 + e^{y_i w^\top x_i}}$$

Step 4: SGD Update Rule

$$\boxed{w_{t+1} = w_t + \eta \frac{y_i x_i}{1 + e^{y_i w_t^\top x_i}}}$$

Step 5: Limit Relation (Perceptron as Extreme Case)

As:

$$\|w^\top x\| \to \infty$$

$$\frac{1}{1 + e^{yw^\top x}} \to \begin{cases} 1 & yw^\top x \leq 0 \\ 0 & yw^\top x > 0 \end{cases}$$

Therefore:

$$\boxed{\text{Perceptron} \;=\; \lim_{\text{hard margin}} \; \text{Logistic Regression}}$$

Final Compact Summary

$$\boxed{\begin{aligned} \text{Perceptron}: &\;\; \text{constraint satisfaction, no smooth loss} \\ \text{Logistic Reg.}: &\;\; \text{maximum likelihood, convex loss} \\ \text{Connection}: &\;\; \text{hard-margin limit of logistic} \end{aligned}}$$

If you want next (pure math options only):

- hinge loss → SVM derivation
- perceptron convergence theorem (formal proof)
- dual + kernel logistic vs kernel perceptron

Say which.

than from here how binary cross entropy comes in the picture

Below is the **strict mathematical chain** from **logistic regression → binary cross-entropy**, with **no intuition, no narrative**.

---

1. Model Assumption (Logistic Regression)

Binary labels:

$$y_i \in \{0, 1\}$$

Linear score:

$$z_i = w^\top x_i$$

Sigmoid:

$$p_i = P(y_i = 1 \mid x_i) = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

---

2. Bernoulli Likelihood

$$P(y_i \mid x_i, w) = p_i^{y_i} (1 - p_i)^{1 - y_i}$$

Joint likelihood:

$$\mathcal{L}(w) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i}$$

---

3. Negative Log-Likelihood

$$
\begin{aligned}
J(w) &= -\log \mathcal{L}(w) \\
&= -\sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right]
\end{aligned}
$$

---

4. Binary Cross-Entropy (Exact Identity)

$$
\boxed{\mathcal{L}_{\text{BCE}}(y, p) = -\left[y \log p + (1 - y) \log(1 - p) \right]}
$$

Thus:

$$
\boxed{\text{Logistic Regression Loss} \equiv \text{Binary Cross-Entropy}}
$$

---

5. Alternative Label Encoding Equivalence

If

$$
y_i \in \{-1, +1\}
$$

Then:

$$
J(w) = \sum_{i=1}^{n} \log\left(1 + e^{-y_i w^\top x_i}\right)
$$

which is algebraically equivalent to BCE under the mapping:

$$
y_{0/1} = \frac{y_{-1/+1} + 1}{2}
$$

---

6. Gradient (Explicit)

$$
\nabla_w J(w) = \sum_{i=1}^{n} (p_i - y_i) x_i
$$

SGD form:

$$
\boxed{w \leftarrow w + \eta (y_i - p_i) x_i}
$$

---

7. Position in the Full Chain

$$
\boxed{\text{Perceptron} \xrightarrow{\text{softening}} \text{Logistic Regression} \xrightarrow{\text{Bernoulli MLE}} \text{Binary Cross-Entropy}}
$$

Final Identity (Key Line)

$$
\boxed{\text{Binary Cross-Entropy is the negative log-likelihood of a Bernoulli model}}
$$

If you want next (still pure math):

- multiclass → softmax cross-entropy
- BCE vs hinge loss vs exponential loss (formal relations)
- why MSE fails for classification (Hessian analysis)

Say which.

again got every thing messes create a step by step picture what and what happend

Below is a **clean, linear, step-by-step "picture in words"** of **what happens and in what order**, from **Perceptron → Logistic Regression → Binary Cross-Entropy**.
Read **top to bottom only**. Nothing is mixed.

STEP 0 — Starting Point (What you already had)

Binary labels

$$y \in \{-1, +1\}$$

Linear score

$$z = w^\top x$$

Hard decision

$$\hat{y} = \text{sign}(z)$$

This is the **perceptron world**.

STEP 1 — Perceptron World (Hard, Non-probabilistic)

Constraint

$$yz \geq 0$$

Update (only on mistake)

$$w \leftarrow w + \eta y x$$

Key properties:

- No probability
- No likelihood
- No smooth objective
- Just constraint satisfaction

⬇️

Problem: cannot measure *how wrong* a prediction is

STEP 2 — Replace HARD decision with SOFT score

Keep the same linear model:

$$z = w^\top x$$

Replace `sign(z)` with a **continuous squashing function**:

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

Now:

$$p = \sigma(z)$$

Interpretation:

$$p \in (0,1)$$

⬇️

Now the model outputs a **number**, not just ±1.

STEP 3 — Change label space to match probability

Switch labels:

$$y \in \{0,1\}$$

Now:

$$p = P(y=1 \mid x)$$

This step is **purely mathematical re-encoding**:

$$y_{0/1} = \frac{y_{-1/+1} + 1}{2}$$

⬇️

Now we can write a probability model.

STEP 4 — Write the Probability Model (Bernoulli)

For one sample:

$$P(y \mid x, w) = p^{y}(1-p)^{1-y}$$

This is a **Bernoulli distribution**.

For all data:

$$\mathcal{L}(w) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

⬇️

Now we have a **likelihood**.

STEP 5 — Convert Likelihood → Optimization Objective

Take negative log:

$$J(w) = -\sum_{i=1}^{n} [y_i \log p_i + (1-y_i)\log(1-p_i)]$$

This expression is **named**:

$$\boxed{\text{Binary Cross-Entropy}}$$

Nothing new is invented here — it is **just −log(likelihood)**.

STEP 6 — Final Training Rule (Gradient Descent)

Gradient:

$$\nabla_w J = \sum_{i=1}^{n} (p_i - y_i)x_i$$

Update:

$$w \leftarrow w + \eta(y_i - p_i)x_i$$

Compare with perceptron:

$$\text{Perceptron: } y - \hat{y} \qquad \text{Logistic: } y - p$$

Same structure, **softened error**.

ONE-LINE FLOW (THIS IS THE "PICTURE")

```sql
Linear score (wᵀx)
        ↓
Hard sign  →  Soft sigmoid
        ↓
±1 labels →  0/1 labels
        ↓
Constraint → Probability model
        ↓
No loss    → Bernoulli likelihood
        ↓
–          → Binary Cross-Entropy
```

FINAL LOCK-IN STATEMENT (Very Important)

$$\boxed{\text{Binary Cross-Entropy} = \text{Negative Log-Likelihood of Logistic Regression}}$$

Nothing extra. Nothing mixed.
Just **one logical chain**.

If you want next (still step-wise):

- **same picture for SVM → hinge loss**
- **why MSE does NOT appear**
- **perceptron → exponential loss → AdaBoost**

Tell me which chain you want.

create same above kind of picture for multiple classification problem

Below is the **same clean "picture-in-steps"**, but now for the **MULTI-CLASS classification problem**
(from **binary → multi-class**, ending at **softmax + categorical cross-entropy**).

Read **top → bottom only**.

STEP 0 — Problem Setup (Multiclass)

Classes:

$$y \in \{1, 2, \dots, K\}$$

One-hot encoding:

$$y \in \{0,1\}^K, \quad \sum_{k=1}^{K} y_k = 1$$

Input:

$$x \in \mathbb{R}^d$$

STEP 1 — Linear Scores (Generalization of Perceptron)

Instead of one score, we now have **K scores**:

$$z = W^\top x \quad \Rightarrow \quad z_k = w_k^\top x$$

Decision rule (hard):

$$\hat{y} = \arg\max_k z_k$$

This is the **multiclass perceptron view**.

⬇️

Problem: still hard decisions, no probabilities.

STEP 2 — Replace HARD argmax with SOFT normalization

Apply **softmax** to scores:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Now:

$$p_k \in (0,1), \quad \sum_{k=1}^{K} p_k = 1$$

⬇️

Scores → **probability distribution over classes**

STEP 3 — Probabilistic Model (Categorical Distribution)

For one sample:

$$P(y \mid x, W) = \prod_{k=1}^{K} p_k^{y_k}$$

This is a **Categorical (Multinoulli) distribution**.

For dataset:

$$\mathcal{L}(W) = \prod_{i=1}^{n} \prod_{k=1}^{K} p_{ik}^{y_{ik}}$$

⬇️

Now we have a valid **likelihood**.

STEP 4 — Likelihood → Optimization Objective

Take negative log-likelihood:

$$J(W) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \log p_{ik}$$

This expression is **named**:

$$\boxed{\text{Categorical Cross-Entropy}}$$

Nothing new is added — just **−log(likelihood)**.

STEP 5 — Gradient (Training Rule)

Softmax + cross-entropy gives:

$$\nabla_{w_k} J = \sum_{i=1}^{n} (p_{ik} - y_{ik}) x_i$$

SGD update:

$$\boxed{w_k \leftarrow w_k + \eta (y_{ik} - p_{ik}) x_i}$$

Compare forms:

- Perceptron (binary): $y - \hat{y}$
- Logistic (binary): $y - p$
- Softmax (multi): $y_k - p_k$

Same structure, higher dimension.

ONE-LINE FLOW (THE "PICTURE")

mathematica Copy code

```
Linear scores (WᵀX)
        ↓
Hard argmax
        ↓
Softmax normalization
        ↓
Class probabilities
        ↓
Categorical likelihood
        ↓
Negative log-likelihood
        ↓
Categorical Cross-Entropy
```

FINAL LOCK-IN TABLE (IMPORTANT)

Problem type	Output layer	Probability model	Loss
Binary	Sigmoid	Bernoulli	Binary Cross-Entropy
Multiclass	Softmax	Categorical	Categorical Cross-Entropy

FINAL STATEMENT (DO NOT FORGET)

$$\boxed{\text{Softmax} + \text{Cross-Entropy} = \text{Maximum Likelihood for Multiclass Logistic Regression}}$$

If you want next, we can extend **the same picture** to:

- **One-vs-Rest vs Softmax (formal difference)**
- **Multiclass SVM → hinge loss**
- **Why softmax + MSE is mathematically wrong**
- **Connection to neural network last layer**

Tell me which chain you want next.

answer in just yes or no does these are the steps in correct order that make logistic regression as them

Yes.