ECE 1513: Introduction to Machine Learning

Lecture 5: Gradient Descent and Classification with Confidence

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Quick Recap: *Unsupervised vs Supervised Learning*

In Unsupervised Learning, samples are unlabeled

- *Data* $\rightsquigarrow$ *Collection of samples* $\mathbb{D} = \{\boldsymbol{x}_n : n = 1, \dots, N\}$
- *Model* $\rightsquigarrow$ *Captures a pattern observed in data, e.g., fitting into clusters*
- *Learning algorithm* $\rightsquigarrow$ *It takes* $\mathbb{D}$ *and returns a* good *model*

In Supervised Learning, samples are labeled

- *Data* $\rightsquigarrow$ *Collection of samples* $\mathbb{D} = \{(\boldsymbol{x}_n, \boldsymbol{v}_n) : n = 1, \dots, N\}$
- *Model* $\rightsquigarrow$ *Captures relation between data samples and their labels*
- *Learning algorithm* $\rightsquigarrow$ *It takes* $\mathbb{D}$ *and returns a* good *model*

Quick Recap: *Supervised Learning*

- *Labeled dataset*

$$\mathbb{D} = \{(\boldsymbol{x}_n \in \mathbb{X}, \boldsymbol{v}_n \in \mathbb{V}) : n = 1, \ldots, N\}$$

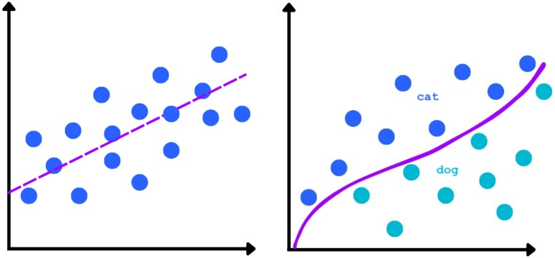
- *Model from hypothesis set $\mathbb{H}$ that relates samples to labels*

$$f : \mathbb{X} \mapsto \mathbb{V}$$

- *Learning algorithm finds optimal model within hypothesis set*

$$\mathcal{A} : \mathbb{D} \mapsto f^\star \in \mathbb{H}$$

Quick Recap: *Regression versus Classification*



In linear regression and classification, we focus on linear models

$$f(\boldsymbol{x}) = \mathbf{w}^{\mathsf{T}} \boldsymbol{x}$$

Today's Agenda: *Gradient Descent and Support Vectors*

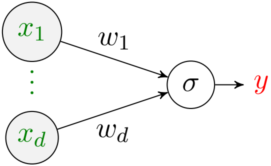
Today, we discuss two key concepts that enable us efficiently classify, i.e.,

Gradient Descent Algorithm and *Support Vectors Classifiers*

The concepts are discussed as follows

- *Classification via Maximum Likelihood*
  - ↳ *Cross-entropy*
- *Gradient Descent Algorithm*
- *Classification with Confidence*
  - ↳ *Support Vector Classifiers*

Logistic Regression



For inference, we compute

$$y = \sigma\left(\boldsymbol{x}^{\mathsf{T}}\mathbf{w}^{\star}\right) \rightsquigarrow \begin{cases} \hat{v} = 1 & y \geqslant 0.5 \\ \hat{v} = 0 & y < 0.5 \end{cases}$$

Soft Output

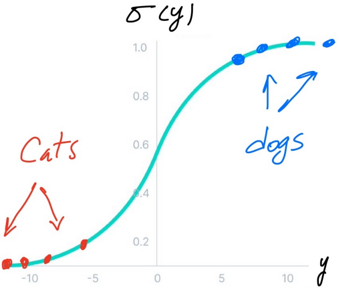
Our model does not compute label. It computes its probability!

Logistic Regression: *Training*

For training, we thought of regression problem

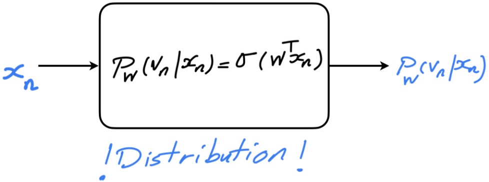
$$\mathbf{w}^\star = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \left(\sigma\left(\mathbf{w}^\mathsf{T} \boldsymbol{x}\right) - v_n \right)^2$$

? *What are we doing in this approach?*



Logistic Regression: *Sigmoid Output as Probability*

! *We know that we compute a probability*



! *We are learning a distribution*

$$P_{\mathbf{w}}\left(v|\boldsymbol{x}\right) = \begin{cases} \sigma\left(\mathbf{w}^{\mathsf{T}}\boldsymbol{x}\right) & v = 1 \\ 1 - \sigma\left(\mathbf{w}^{\mathsf{T}}\boldsymbol{x}\right) & v = 0 \end{cases}$$

↳ *We could maximize the likelihood*

Logistic Regression: *Maximum Likelihood Estimate*

Let's compute the likelihood of the dataset

$$\mathcal{L}\left(\mathbb{D}\right) = \prod_{n=1}^{N} P_{\mathbf{w}}\left(v_n|\boldsymbol{x}_n\right)$$

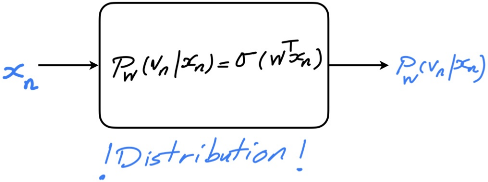
We are more comfortable with log-likelihood

$$\mathcal{S}\left(\mathbb{D}\right) = \frac{1}{N}\sum_{n=1}^{N} \log P_{\mathbf{w}}\left(v_n|\boldsymbol{x}_n\right)$$

Recall: Maximum Likelihood

Find $\mathbf{w}$ *such that the likelihood is maximized*

Logistic Regression: *Maximum Likelihood Estimate*



With $y_n = \sigma\left(\mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n\right)$, we can write

$$\log P_{\mathbf{w}}\left(v_n | \boldsymbol{x}_n\right) = \begin{cases} \log\left(y_n\right) & v = 1 \\ \log\left(1 - y_n\right) & v = 0 \end{cases}$$

$$= v_n \log y_n + \left(1 - v_n\right) \log\left(1 - y_n\right) = D\left(y_n, v_n\right)$$

Log Likelihood versus Empirical Risk

Let's now find the maximum likelihood estimate for $\mathbf{w}$

$$\begin{aligned}
\mathbf{w}^\star &= \operatorname*{argmax}_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \log P_{\mathbf{w}} \left(v_n | \boldsymbol{x}_n \right) \\
&= \operatorname*{argmax}_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} D \left(y_n, v_n \right) \\
&= \operatorname{argmin}_{\mathbf{w}} - \frac{1}{N} \sum_{n=1}^{N} D \left(y_n, v_n \right)
\end{aligned}$$

This is an empirical risk minimization with

$$\mathcal{L} \left(y, v \right) = -D \left(y, v \right) = v \log \frac{1}{y} + \left(1 - v \right) \log \frac{1}{1 - y}$$

Sample Log Likelihood $\equiv$ Cross Entropy

Moral of Story

Empirical risk minimization is equivalent to maximum likelihood with the sample log-likelihood being proportional to the loss function

The one used in classification has a specific name: *cross-entropy*

Binary Cross-Entropy

Cross-entropy between binary distributions

$$P(v) = \begin{cases} p & v = 1 \\ 1 - p & v = 0 \end{cases} \qquad Q(v) = \begin{cases} q & v = 1 \\ 1 - q & v = 0 \end{cases}$$

is defined as

$$\mathrm{CE}(p, q) = -q \log p - (1 - q) \log (1 - p)$$

Cross Entropy: *Binary Case*

In binary classification, we have

$$\mathcal{L}\left(y, v\right) = -v \log y - \left(1 - v\right) \log\left(1 - y\right)$$

Comparing with the definition, we see that

$$\mathcal{L}\left(y, v\right) = \mathrm{CE}\left(y, v\right)$$

Note

Here, we have $v \in \{0, 1\}$, i.e., we have

$$Q\left(v\right) = \begin{cases} 1 & v = 1 \\ 0 & v = 0 \end{cases} \qquad \text{or} \qquad Q\left(v\right) = \begin{cases} 0 & v = 1 \\ 1 & v = 0 \end{cases}$$

This makes sense, as $Q\left(v\right)$ is distribution of true label that is known to us

Approximating Optimal Model $\equiv$ Training

Except linear regression, none of optimal models is explicitly determined

- *There might be a unique minimizer, but not explicit*
  - ↳ *Logistic regression*
- *It might be an enormous number of minimizers*
  - ↳ *Almost all neural networks as we will see!*

We would be hence OK to find a rather fair local minimizer

Training

The procedure of finding a minimizer for the empirical risk is called training

Algorithmic Approach to Training

In practice, we use algorithmic approaches: *say the model depends on* $\mathbf{w}$

```
GenericTraining():
Start with w
while w not converged do
    Update w
end while
Return w
```

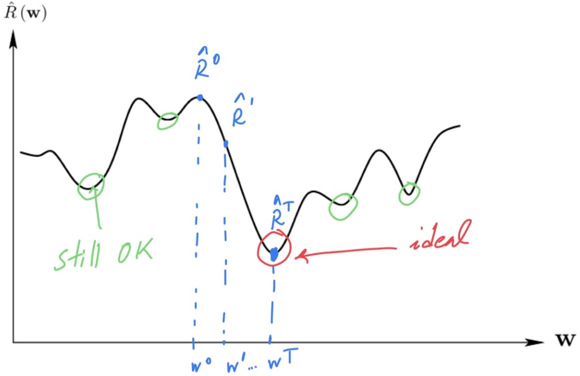
? *What is the right approach to update?*

Optimization Algorithm $\equiv$ *Optimizer*

Let's look at it from a more generic perspective: *we deal with an optimization*

$$\min_{\mathbf{w}} \hat{R}(\mathbf{w})$$

and want to design an optimizer



General Optimizer

`Optimizer():`

1: Choose some threshold ϵ and step size η
2: Start with an arbitrary $\mathbf{w}^0$
3: **while** $t < 1$ or $\|\mathbf{w}^t - \mathbf{w}^{t-1}\|^2 > \epsilon$ **do**
4: Compute a moving direction

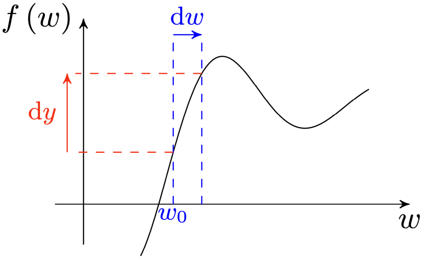
$$\boldsymbol{\mu}^t \leftarrow \texttt{direction.calculator}\left(\hat{R}, \mathbf{w}^t\right)$$

5: Update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \eta\boldsymbol{\mu}^t$
6: Update $t \leftarrow t + 1$
7: **end while**
8: Return final $\mathbf{w}^t$

We need to find a systematic approach to compute a good direction $\boldsymbol{\mu}$

? *What is a good direction?*

Derivatives: *Intuitive Interpretation*

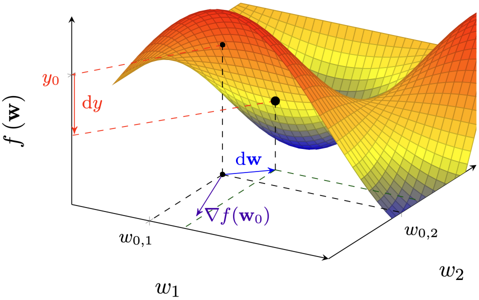


For a simple 1-dimensional function, derivative gives the slope

its sign always points to the direction that the function increases

The function decreases if we move towards the negative direction

Gradients: *Extension to Higher Dimensions*



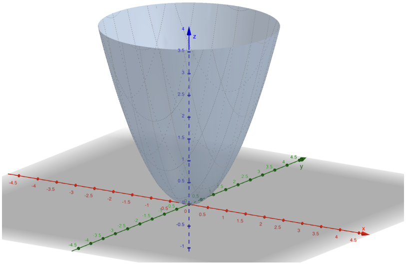
Gradient of a function shows same property in higher dimension

- *its magnitude gives the slope of tangent hyperplane*
- *its direction tells us where to move to increase maximally*
  - *its negative direction tells us where to move to decrease maximally*

Example: *3D Paraboloid*

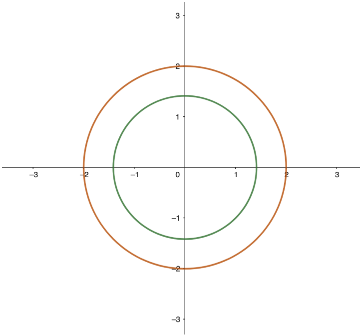
The surface is

$$f(\mathbf{w}) = w_1^2 + w_2^2$$



We can also plot the counters

$$f(\mathbf{w}) = \alpha$$

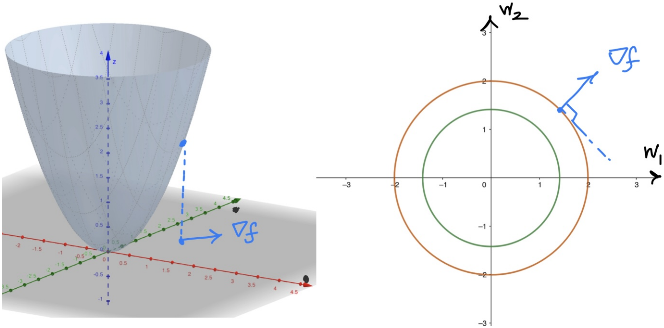


Example: *3D Paraboloid*

Let's compute the gradient

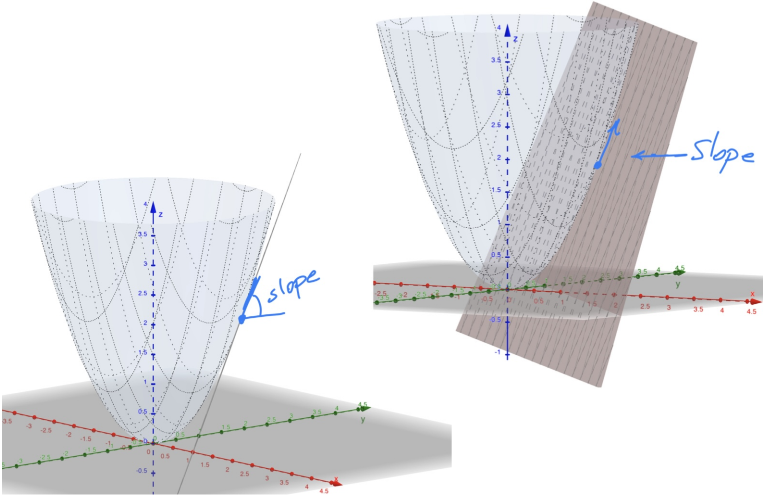
$$f(\mathbf{w}) = w_1^2 + w_2^2 \rightsquigarrow \nabla f = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

It points to the direction that we increase maximally



Example: *3D Paraboloid*

The amplitude is proportional to the slope of touching plane



Gradient Direction $\equiv$ *Steepest Direction*

Moral of Story

Gradient direction is the steepest direction on the surface

Optimizer looks for a direction that makes the function smaller: *we could move in the negative direction of gradient*

$$\boldsymbol{\mu}^t \leftarrow \texttt{direction.calculator}\left(\hat{R}, \mathbf{w}^t\right) = -\nabla \hat{R}\left(\mathbf{w}^t\right)$$

This means that we update

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \textcolor{red}{\eta} \nabla \hat{R}\left(\mathbf{w}^t\right)$$

? *Does it end at a minimum?*

Moving with Gradient

Optimizer stops when

$$\left\|\mathbf{w}^{t+1} - \mathbf{w}^{t}\right\|^2 \leqslant \epsilon$$

As we move with gradient, we have at the stop point

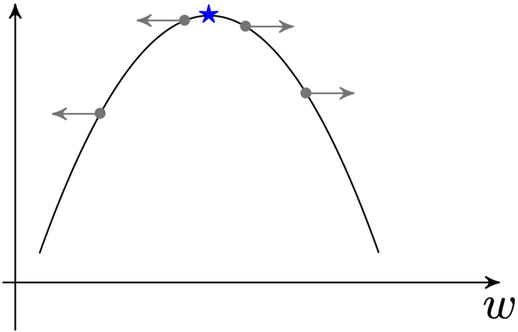
$$\left\|\mathbf{w}^{t+1} - \mathbf{w}^{t}\right\|^2 = \left\|\nabla \hat{R}\left(\mathbf{w}^{t}\right)\right\|^2 \approx 0$$

So, we stop when $\nabla \hat{R}\left(\mathbf{w}^{t}\right) \approx \mathbf{0}$ which is

- *Minimum*
- *Maximum*
- *Saddle Point*

Moving with Gradient

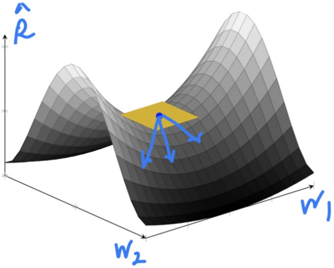
? *Can it get trapped at a maximum?*

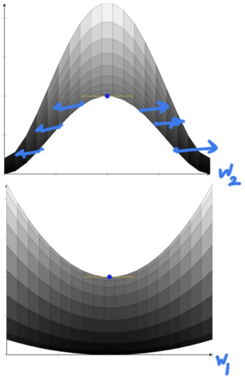


! *No!*

Moving with Gradient

? *Can it get trapped at a saddle point?*

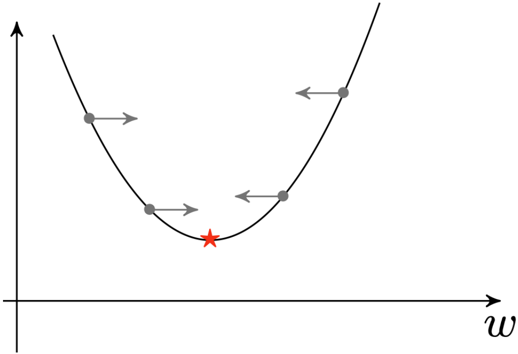




! *No!*

Moving with Gradient

? *Can it get trapped at a minimum?*



! *Yes!*

Gradient Descent Algorithm

```
GradientDescent():
```

Choose some threshold ϵ and step size η
Start with an arbitrary $\mathbf{w}^0$
while $t < 1$ or $\|\mathbf{w}^t - \mathbf{w}^{t-1}\|^2 > \epsilon$ **do**
  Compute the gradient at $\mathbf{w}^t$
  Update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \nabla \hat{R}\left(\mathbf{w}^t\right)$
  Update $t \leftarrow t + 1$
end while
Return final $\mathbf{w}^t$

The algorithm starts at $\mathbf{w}^0$ and moves to a local minimum at step η

↳ *η is what we call learning rate*

Generic Form: *Gradient-based Optimizer*

You might hear the term

Gradient-based Optimizer

This is pretty much the same thing

`GradientOptim():`

1. Choose some threshold ϵ and step size η
2. Start with an arbitrary $\mathbf{w}^0$
3. **while** $t < 1$ or $\|\mathbf{w}^t - \mathbf{w}^{t-1}\|^2 > \epsilon$ **do**
4. Compute the gradient at $\mathbf{w}^t$
5. Update $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta f_\beta\left(\nabla \hat{R}\left(\mathbf{w}^t\right)\right)$
6. Update $t \leftarrow t + 1$
7. **end while**
8. Return final $\mathbf{w}^t$

$f_\beta(\cdot)$ *is an operation usually some linear operation*

↳ *We mention it later when we get to SGD*

Example: *Linear Regression*

In linear regression, the empirical risk is

$$\hat{R}\left(\mathbf{w}\right) = \frac{1}{N}\|\mathbf{X}^{\mathsf{T}}\mathbf{w} - \boldsymbol{v}\|^2$$

At point $\mathbf{w}^t$, we have

$$\nabla\hat{R}\left(\mathbf{w}^t\right) = \frac{2}{N}\left(\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{w}^t - \mathbf{X}\boldsymbol{v}\right)$$

We then update as

$$\begin{aligned}\mathbf{w}^{t+1} &\leftarrow \mathbf{w}^t - \frac{2\eta}{N}\left(\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{w}^t - \mathbf{X}\boldsymbol{v}\right)\\ &\leftarrow \left(\mathbf{I}_d - \frac{2\eta}{N}\mathbf{X}\mathbf{X}^{\mathsf{T}}\right)\mathbf{w}^t + \frac{2\eta}{N}\mathbf{X}\boldsymbol{v}\end{aligned}$$

Example: *Linear Regression*

? *When does it stop?*

When $\mathbf{w}^{t+1} \approx \mathbf{w}^{t}$, i.e.,

$$\mathbf{w}^{t} \approx \mathbf{w}^{t} - \frac{2\eta}{N}\left(\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{w}^{t} - \mathbf{X}\boldsymbol{v}\right)$$

or equivalently

$$\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{w}^{t} - \mathbf{X}\boldsymbol{v} \approx 0 \rightsquigarrow \mathbf{w}^{t} \approx \left(\mathbf{X}\mathbf{X}^{\mathsf{T}}\right)^{-1}\mathbf{X}\boldsymbol{v}$$

! *Bingo! Same as the analytic result!*

Convergence Guarantee

Gradient descent is guaranteed to end to a local minimum

- *Global convergence is only guaranteed if empirical risk is convex*
  - ↳ *Linear and logistic regression*
- *Learning rate is crucial in the behavior*
  - ↳ *With very large learning rates it can diverge*
  - ↳ *With very small learning rates it takes long time*
  - ↳ *With very small learning rates we may get trapped in bad local minimum*

Further Read

- Goodfellow
  - ↳ Chapter 4: *Section 4.3 – 4.5* ***Gradient Descent***
- Bishop
  - ↳ Chapter 5: *Section 5.2* ***Optimizers***
- ESL
  - ↳ Chapter 10: *Section 10.10* ***Numerical Optimization***

Earlier Example: *Cat or Dog*

- *Dataset*

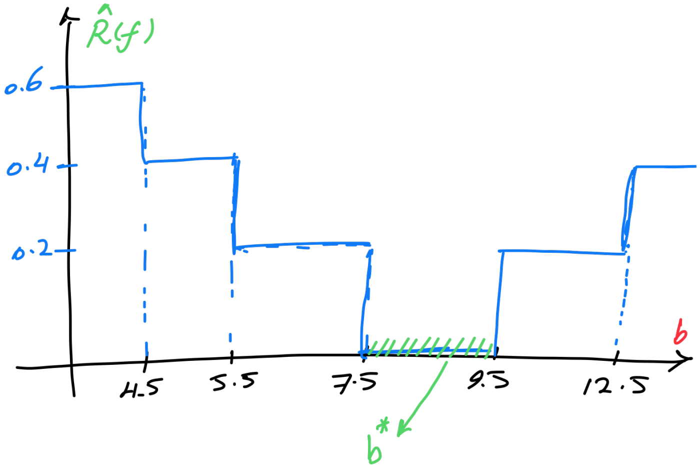
$$\mathbb{D} = \{(x, v) = (5.5, \textit{cat}), (4.5, \textit{cat}), (12.5, \textit{dog}), (9.5, \textit{dog}), (7.5, \textit{cat})\}$$

- *Linear classifier*

$$y = \begin{cases} \text{cat} & x < b \\ \text{dog} & x \geqslant b \end{cases}$$

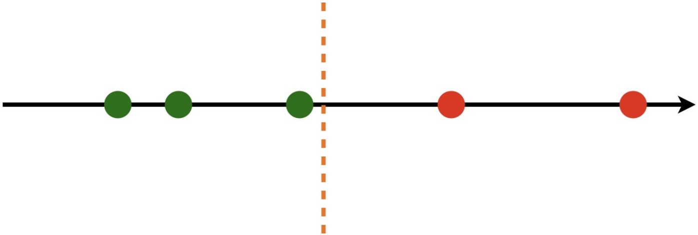
- *Learning algorithm* $\equiv$ *empirical risk minimization*

Earlier Example: *Cat or Dog*

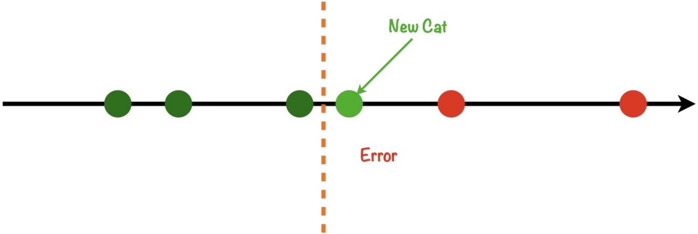


Earlier Example: *Cat or Dog*

This threshold is optimal

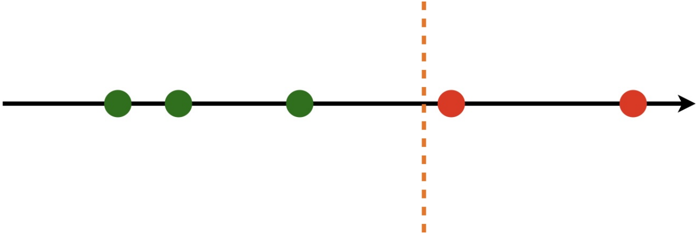


But seems to have poor confidence with new samples

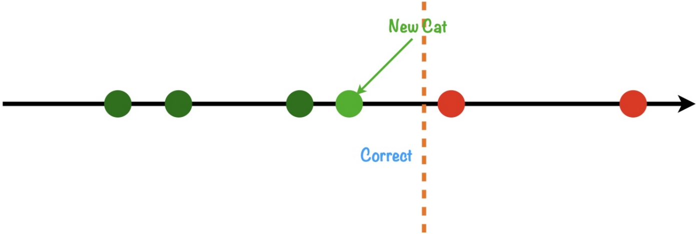


Earlier Example: *Cat or Dog*

Maybe we can push right

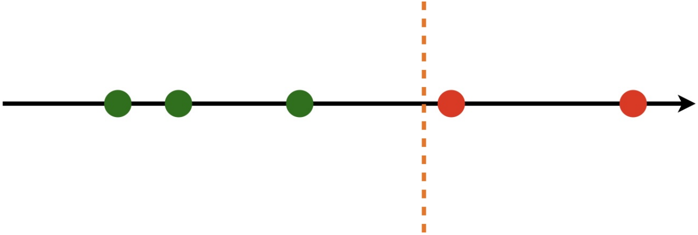


It seems to work

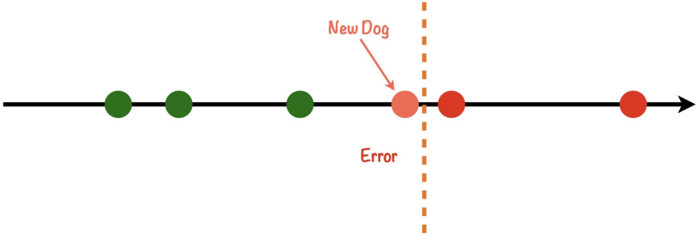


Earlier Example: *Cat or Dog*

But we could have a new dog sample

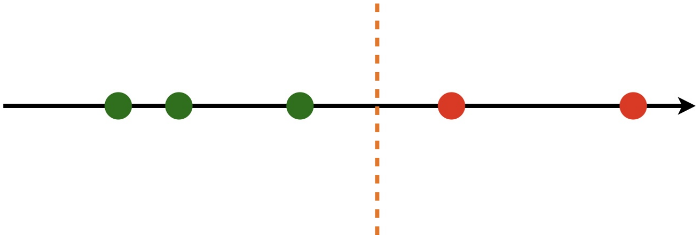


It seems not to be confident again

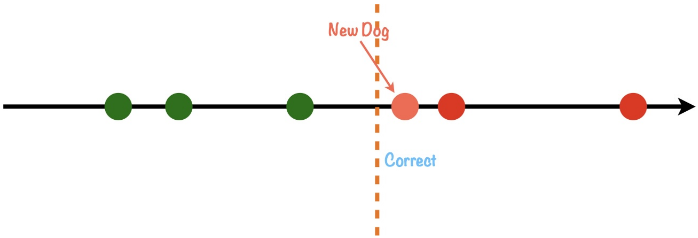


Earlier Example: *Cat or Dog*

Maybe we can push left



Middle point seems to be best



Confidence for Generalization

We have in general many thresholds that minimize the empirical risk

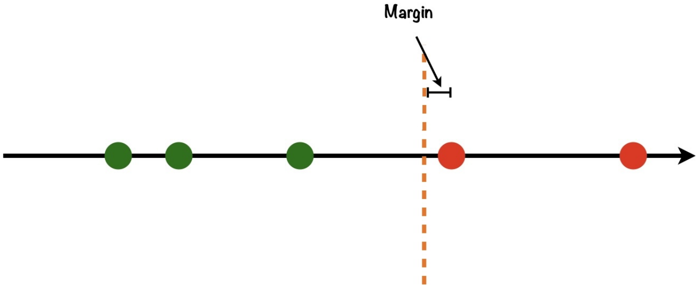
- *We could look among them for the one that gives us maximal confidence*
- *Confidence describes how well we thing the choice generalizes*
  - ↳ *This threshold works perfect on the dataset*
  - ↳ *We want to minimize the chance of having error with new samples*
- *For maximal confidence, we could maximize the margin*

? *How do we define the margin?*

Defining Margins

Margin

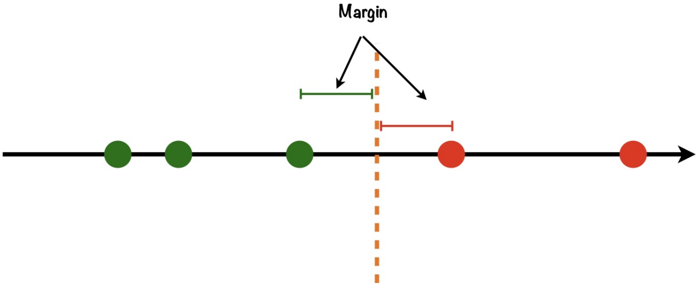
Margin is the minimum distance to the classification boundary



Classifying with Maximum Margin

Maximal Margin

We try to maximize the margin



Binary Classification

We now try to formulate everything concretely

Consider a binary classification problem with dataset

$$\mathbb{D} = \left\{ (\boldsymbol{x}_n, v_n) : \boldsymbol{x}_n \in \mathbb{R}^d \text{ and } v_n \in \{-1, 1\} \right\}$$

We want to use a linear classifier, i.e.,

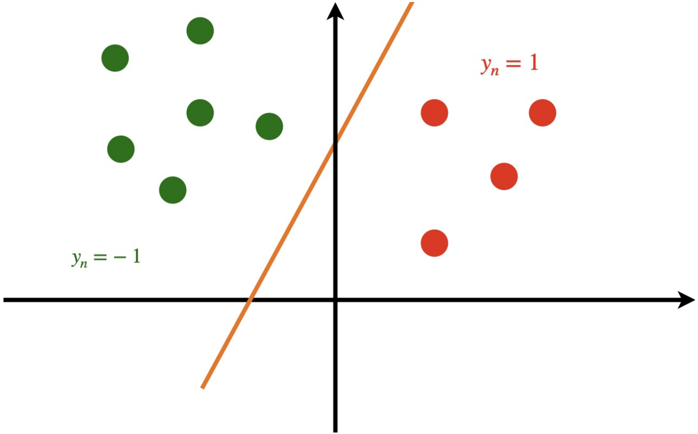
$$z_n = \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n \rightsquigarrow y_n = \begin{cases} 1 & z_n \geqslant 0 \\ -1 & z_n < 0 \end{cases}$$

A New Desire

We have maximal margin $\equiv$ confident classification

Binary Classification

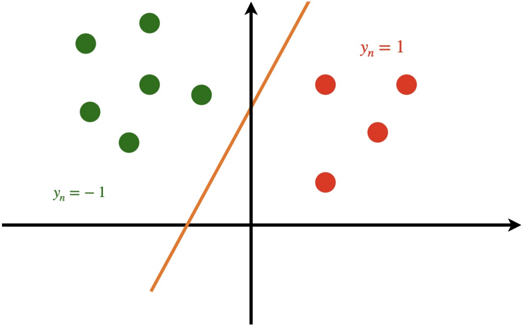
Recall that linear classifier is separation by a hyperplane



Binary Classification: *Error-Free Classification*

With perfect boundary, we have

$$y_n = \begin{cases} 1 & v_n = 1 \\ -1 & v_n = -1 \end{cases} \rightsquigarrow z_n = \begin{cases} \geqslant 0 & v_n = 1 \\ < 0 & v_n = -1 \end{cases} \rightsquigarrow v_n \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n \geqslant 0$$



Binary Classification: *Forcing Maximal Margin*

In basic linear classification, we want zero error on training set, i.e.,

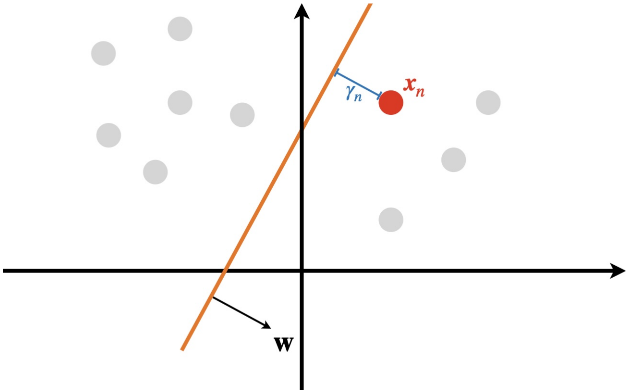
Find $\mathbf{w}$ *such that* $v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 0$ *for all* n

If we want to have also maximal margin, we can write

$\max_{\mathbf{w}}$ *Margin subject to* $v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 0$ *for all* n

- ? *How we can express the margin?*

Finding Margin: Positive Side



Note

$\mathbf{w}$ *is orthogonal to the boundary*

Finding Margin: Positive Side

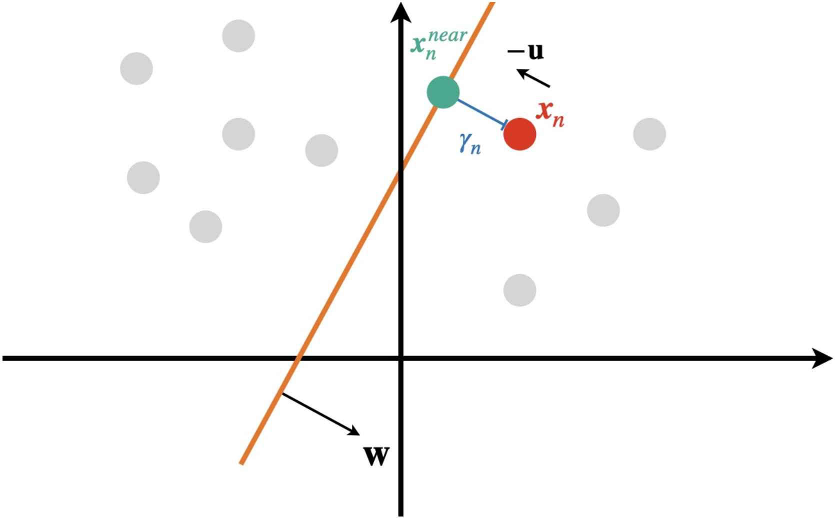
Unit vector orthogonal to boundary $\mathbf{w}^\mathsf{T}\boldsymbol{x} = 0$

$$\mathbf{u} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Margin of point $\boldsymbol{x}_n$ is step γ_n to the nearest point on the boundary

$$\boldsymbol{x}_n^{\textit{near}} = \boldsymbol{x}_n - \gamma_n \mathbf{u} = \textit{on the boundary}$$

Finding Margin: Positive Side



Finding Margin: Positive Side

Any point on the boundary satisfies $\mathbf{w}^\mathsf{T} \boldsymbol{x} = 0$

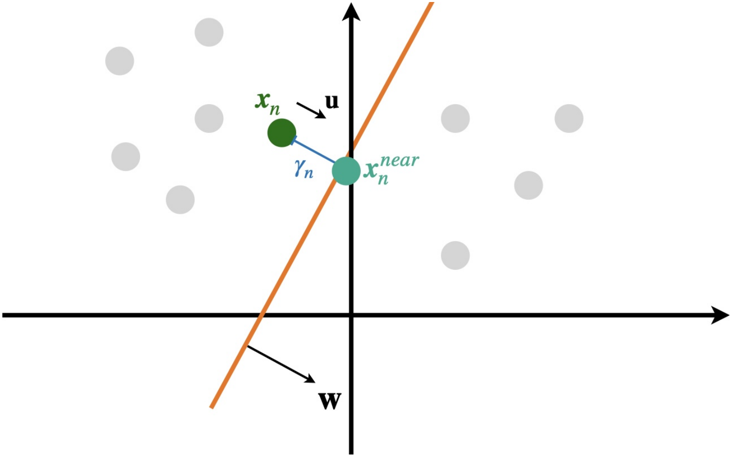
$$\mathbf{w}^\mathsf{T} \boldsymbol{x}_n^{\textit{near}} = \mathbf{w}^\mathsf{T} \left(\boldsymbol{x}_n - \gamma_n \mathbf{u} \right) = 0$$

This concludes

$$\gamma_n = \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_n}{\mathbf{w}^\mathsf{T} \mathbf{u}} = \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_n}{\|\mathbf{w}\|}$$

Finding Margin: Negative Side

? *What if we are on the other side, i.e.,* $v_n = -1$?



Finding Margin: Negative Side

When $v_n = -1$, we can write

$$\mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n^{\textit{near}} = \mathbf{w}^{\mathsf{T}} \left(\boldsymbol{x}_n + \gamma_n \mathbf{u} \right) = 0$$

This concludes

$$\gamma_n = -\frac{\mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n}{\mathbf{w}^{\mathsf{T}} \mathbf{u}} = -\frac{\mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n}{\|\mathbf{w}\|}$$

Finding Margin: Generic

We can say in general that for any $(\boldsymbol{x}_n, v_n)$, we have

$$\gamma_n = v_n \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_n}{\|\mathbf{w}\|}$$

Margin

Margin is the shortest distance to the boundary

$$\mathcal{M} = \min_n v_n \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_n}{\|\mathbf{w}\|} = v_{n^\star} \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_{n^\star}}{\|\mathbf{w}\|}$$

$(\boldsymbol{x}_{n^\star}, v_{n^\star})$ *is the point closest to the boundary*

Finding Margin: Key Observation

? *Does it matter how large $\mathbf{w}$ is scaled?*

Let us scale $\mathbf{w}$ as $\mathbf{w} \leftarrow \alpha \mathbf{w}$, we have

$$\gamma_n = v_n \frac{\alpha \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n}{\alpha \|\mathbf{w}\|} = v_n \frac{\mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n}{\|\mathbf{w}\|}$$

! *No!*

Conclusion

We can scale $\mathbf{w}$ as we wish!

Finding Margin

Let's scale $\mathbf{w}$ such that the closest point satisfies

$$v_{n^\star} \mathbf{w}^\mathsf{T} \boldsymbol{x}_{n^\star} = 1$$

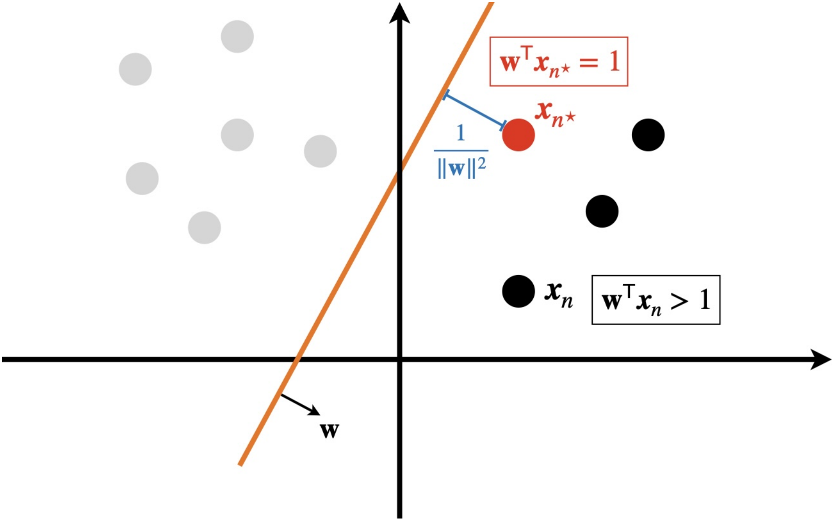
- *For the nearest points, we have*

$$\gamma_{n^\star} = v_{n^\star} \frac{\mathbf{w}^\mathsf{T} \boldsymbol{x}_{n^\star}}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} = \mathcal{M}$$

- *For the all points in the dataset, we have*

$$v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 1$$

Finding Margin: *Visualization*



Training with Maximal Margin

Attention

Note that for all points in the dataset, we have

$$v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 1 \rightsquigarrow v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 0$$

which guarantees zero classification error

We wanted to find $\mathbf{w}$ as

$$\max_{\mathbf{w}} \textit{Margin} \text{ subject to } v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 0 \text{ for all } n$$

We can do it alternatively as

$$\max_{\mathbf{w}} \mathcal{M} \text{ subject to } v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 1 \text{ for all } n$$

Training with Maximal Margin

$$\max_{\mathbf{w}} \mathcal{M} \text{ subject to } v_n \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n \geqslant 1 \text{ for all } n$$

This is written as

$$\max_{\mathbf{w}} \frac{1}{\|\mathbf{w}\|} \qquad \text{subject to } \; v_n \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n \geqslant 1 \text{ for all } n$$

or alternatively

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \qquad \text{subject to } \; v_n \mathbf{w}^{\mathsf{T}} \boldsymbol{x}_n \geqslant 1 \text{ for all } n$$

Support Vector Classifier

Support Vector Classifier

SVC with maximal margin finds $\mathbf{w}$ as

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \qquad \textit{subject to } \; v_n \mathbf{w}^\mathsf{T} \boldsymbol{x}_n \geqslant 1 \textit{ for all } n$$

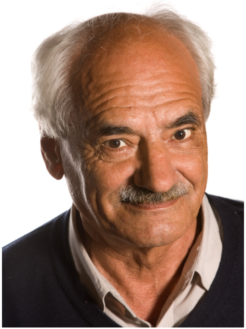
- ? *How can we solve this problem?*
- ! *We use Lagrange Multipliers to do this in the next lecture*

Further Read

- Bishop
  - ↳ Chapter 7: *Section 7.1* ***SVC***
- ESL
  - ↳ Chapter 12: *Section 12.1 – 12.2* ***SVC***

Acknowledgment

Vladimir Vapnik and Alexey Chervonenkis

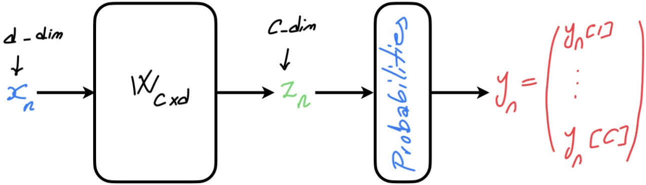


Take a look at VC Theory

Classification with Multiple Classes

- ? *What if we got more than two classes?*
- ! *Maybe, we can scale up the idea*

Say we have C different classes $v_n \in \{1, \dots C\}$



We need a function that transforms output of linear model to probabilities

$$\boldsymbol{y}_n = F(\boldsymbol{z}_n) = F(\mathbf{W}\boldsymbol{x}_n)$$

Softmax: *Distribution Out of Soft Information*

Softmax

Softmax transforms a C-dimensional input to a categorical distribution

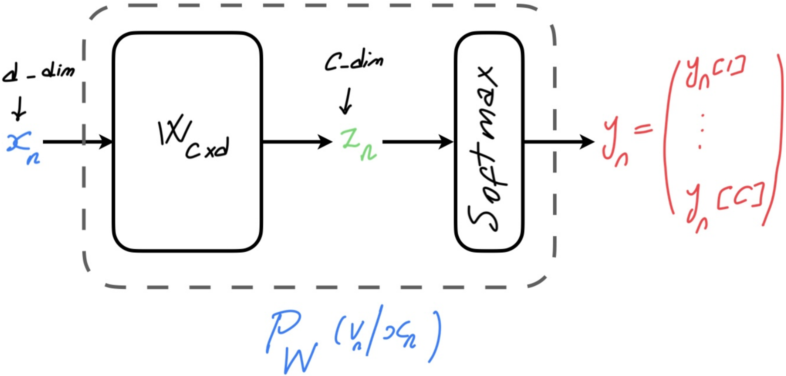
$$y_i = [\mathsf{Soft}_{\max}(\boldsymbol{z})]_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Softmax has the properties we want

- *Its output is a distribution*
- *The output that is larger has higher probabilities*

Maximum Likelihood via *Softmax*

We can again use maximum likelihood to find optimal $\mathbf{W}$



Maximum Likelihood via *Softmax*

We can again use maximum likelihood to find optimal $\mathbf{W}$

$$\mathbf{W}^\star = \operatorname*{argmin}_{\mathbf{W}} \frac{1}{N} \sum_{n=1}^{N} -\log P_{\mathbf{W}} \left(v_n | \boldsymbol{x}_n \right)$$
$$= \operatorname*{argmin}_{\mathbf{W}} \frac{1}{N} \sum_{n=1}^{N} \log \frac{1}{y_n \left[v_n \right]}$$

We can alternatively use one-hot representation of labels

$$\mathbf{u}_n = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \textit{index } 1 \\ \vdots \\ \textit{index } v_n \\ \vdots \\ \textit{index } C \end{array} \rightsquigarrow \log \frac{1}{y_n \left[v_n \right]} = - \sum_{i=1}^{C} u_n[i] \log y_n[i]$$

Cross Entropy: *General Form*

Cross-Entropy

Cross-entropy between two categorical distributions $\mathbf{p}$ *and* $\mathbf{q}$ *is defined as*

$$\mathrm{CE}\left(\mathbf{p}, \mathbf{q}\right) = -\sum_{i=1}^{C} q_i \log p_i$$

So, we could say

$$\mathbf{W}^\star = \operatorname*{argmin}_{\mathbf{W}} \frac{1}{N} \sum_{n=1}^{N} \mathrm{CE}\left(\boldsymbol{y}_n, \mathbf{u}_n\right)$$

We could again say $\boxed{\textit{Maximum Likelihood} \equiv \textit{Empirical Risk with CE Loss}}$