STA 314: Statistical Methods for Machine Learning I

Lecture 5 - Linear Classification

Chris J. Maddison

University of Toronto

Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target

- Examples of binary classification
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent

Today's Agenda

Today's agenda:

- Binary classification.
  - Model, loss function
  - Limitations
- Logistic regression and convexity.
- Gradient descent for binary classification.

Binary linear classification

- **classification:** given a D-dimensional input $\mathbf{x} \in \mathbb{R}^D$ predict a discrete-valued target
- **binary:** predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples. Sorry.
  - $t \in \{0, 1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- **linear:** model prediction y is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Some Simplifications

- **Eliminating the threshold.** Assume without loss of generality (WLOG) that the threshold $r = 0$:

$$\mathbf{w}^\top \mathbf{x} + b \geq r \iff \mathbf{w}^\top \mathbf{x} + \underbrace{b - r}_{\triangleq w_0} \geq 0.$$

- **Eliminating the bias parameter.** Add a dummy feature x_0 which always takes the value 1. The weight $w_0 = b$ is equivalent to a bias parameter (same as linear regression).

- **Simplified model.** Receive input $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$:

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Examples

- Let's consider some simple examples to examine the properties of our model
- Let's focus on minimizing the training set error, and forget about whether our model will generalize to a test set.

Examples

NOT

x_0	x_1	t
1	0	1
1	1	0

- Suppose this is our training set, with the dummy feature x_0 included.

- Which conditions on w_0, w_1 guarantee perfect classification?
  - When $x_1 = 0$, need: $z = w_0 x_0 + w_1 x_1 \geq 0 \iff w_0 \geq 0$
  - When $x_1 = 1$, need: $z = w_0 x_0 + w_1 x_1 < 0 \iff w_0 + w_1 < 0$
- Example solution: $w_0 = 1, w_1 = -2$
- Is this the only solution?

Examples

AND

x_0	x_1	x_2	t
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2$$

$$\text{need: } w_0 < 0$$

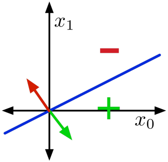
$$\text{need: } w_0 + w_2 < 0$$

$$\text{need: } w_0 + w_1 < 0$$

$$\text{need: } w_0 + w_1 + w_2 \geq 0$$

Example solution: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$

Input Space, or Data Space for NOT example

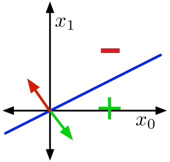


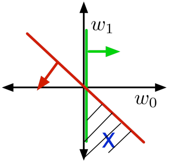
x_0	x_1	t
1	0	1
1	1	0

- Training examples are points
- Weights (hypotheses) $\mathbf{w}$ can be represented by half-spaces
  $H_+ = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} \geq 0\}$, $H_- = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} < 0\}$
  - The boundaries of these half-spaces pass through the origin (why?)
- The boundary is the decision boundary: $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0\}$
  - In 2-D, it's a line, but in high dimensions it is a hyperplane
- If the training examples can be perfectly separated by a linear decision rule, we say data is linearly separable.

The Geometric Picture

Weight Space





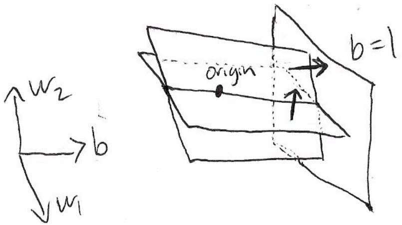
$w_0 \geq 0$

$w_0 + w_1 < 0$

- Weights (hypotheses) $\mathbf{w}$ are points
- Each training example $\mathbf{x}$ specifies a half-space $\mathbf{w}$ must lie in to be correctly classified: $\mathbf{w}^\top \mathbf{x} \geq 0$ if $t = 1$.
- For NOT example:
  - $x_0 = 1, x_1 = 0, t = 1 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 \geq 0\}$
  - $x_0 = 1, x_1 = 1, t = 0 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 + w_1 < 0\}$
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible, otw it is infeasible.

The Geometric Picture

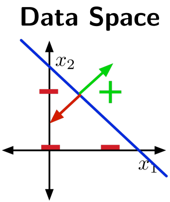
- The **AND** example requires three dimensions, including the dummy one.
- To visualize data space and weight space for a 3-D example, we can look at a 2-D slice:

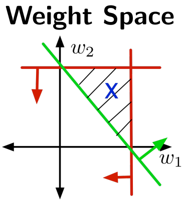


- The visualizations are similar, except that the decision boundaries and the constraints need not pass through the origin.

The Geometric Picture

Visualizations of the **AND** example





- Slice for $x_0 = 1$ and
- example sol: $w_0 = -1.5,\ w_1 = 1,\ w_2 = 1$
- decision boundary:

$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$

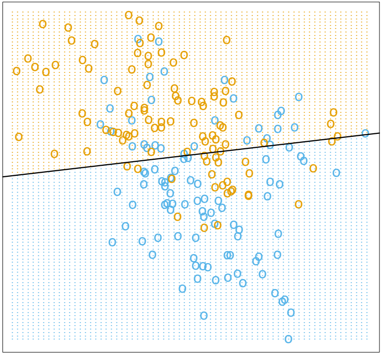
$\implies\ -1.5 + x_1 + x_2 = 0$

- Slice for $w_0 = -1.5$ for the constraints
- $w_0 < 0$
- $w_0 + w_2 < 0$
- $w_0 + w_1 < 0$
- $w_0 + w_1 + w_2 \geq 0$

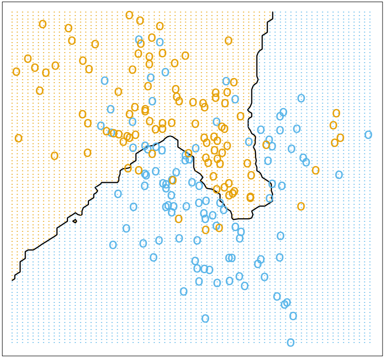
Linear Classifiers vs. KNN

Linear classifiers and KNN have very different decision boundaries:

Linear Classifier



K Nearest Neighbours

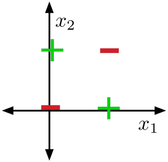


Linear Classifiers vs. KNN

- KNN models are typical higher variance, low bias.
- Linear classifiers are low variance, high bias.
- Computing the prediction of a KNN model is more computationally expensive than a linear at test time.
- Fitting linear classifiers is more computationally expensive than KNN at training time.

Limits of Linear Classification

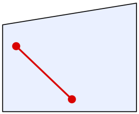
How bad is the bias of linear classifiers? Some datasets are not linearly separable, e.g. **XOR**



Visually obvious, but how to show this? Let's consider the structure of linear classifier predictions.

Limits of Linear Classification

Convex Sets



- A set $\mathcal{S}$ is convex if any line segment connecting points in $\mathcal{S}$ lies entirely within $\mathcal{S}$. Mathematically,

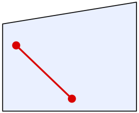
$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{S} \quad \Longrightarrow \quad \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in \mathcal{S} \quad \text{for } 0 \leq \lambda \leq 1.$$

- A simple inductive argument shows that for $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{S}$, weighted averages, or convex combinations, lie within the set:

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N \in \mathcal{S} \quad \text{for } \lambda_i > 0, \; \lambda_1 + \cdots \lambda_N = 1.$$

Limits of Linear Classification

Convex Sets



- For a linear classifier, the set of points that have the same prediction is a convex set. To see why consider $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^D$ with prediction $y = 1$:
- I.e., $\mathbf{w}^\top \mathbf{x}_1 \geq 0$, $\mathbf{w}^\top \mathbf{x}_2 \geq 0$. Then, for $0 \leq \lambda \leq 1$ consider the point $\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$. This point is also labelled $y = 1$:

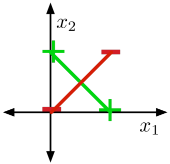
$$\begin{aligned} \mathbf{w}^\top(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) &= \lambda \mathbf{w}^\top \mathbf{x}_1 + (1-\lambda)\mathbf{w}^\top \mathbf{x}_2 \\ &\geq \lambda \cdot 0 + (1-\lambda) \cdot 0 \\ &= 0 \end{aligned}$$

- Similar for two points with prediction $y = 0$.

Limits of Linear Classification

Showing that XOR is not linearly separable (proof by contradiction)

- If two points have the same prediction, then all points on the line segment connecting them also have the same prediction.
- Suppose there were some feasible weights (hypothesis) that perfectly classify the XOR set.
- If this hypothesis predicts $y = 1$ for all positive examples, then points on the green line segment must have prediction $y = 1$.
- Similarly, the points on the red line segment must all have prediction $y = 0$.



- But hypothesis cannot predict both $y = 1$ and $y = 0$ for the intersection. Contradiction!

Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for **XOR**:

$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

x_1	x_2	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	t
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable. (Try it!)

Summary — Binary Linear Classifiers

- **Summary:** Targets $t \in \{0, 1\}$, inputs $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$, and model is defined by weights $\mathbf{w}$ and

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Towards Logistic Regression

Loss Functions

- How can we find good values for $\mathbf{w}$?
- If training set is linearly separable, we could solve for $\mathbf{w}$ using linear programming
  - We could also apply an iterative procedure known as the *perceptron algorithm* (but this is primarily of historical interest).
- If it's not linearly separable, the problem is harder
  - Data is almost never linearly separable in real life.
- Instead: define loss function then try to minimize the resulting cost function
  - Recall: cost is loss averaged (or summed) over the training set

Attempt 1: 0-1 loss

- Seemingly obvious loss function: 0-1 loss

$$
\begin{aligned}
L_{0-1}(y, t) &= \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases} \\
&= \mathbb{I}[y \neq t]
\end{aligned}
$$

- The $\hat{\mathcal{R}}$ is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate:

$$
\hat{\mathcal{R}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]
$$

Attempt 1: 0-1 loss

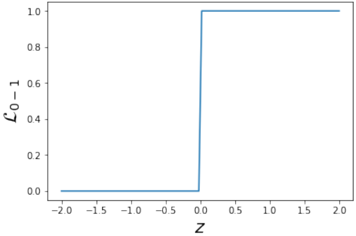
- Problem: how to optimize? In general, a hard problem (can be very hard computationally)
- This is due to the step function (0-1 loss) not being nice (continuous/smooth/convex etc)

Attempt 1: 0-1 loss

- Minimum of a function will be at its critical points.
- Let's try to find the critical point of 0-1 loss
- Chain rule:

$$\frac{\partial L_{0-1}}{\partial w_j} = \frac{\partial L_{0-1}}{\partial z} \frac{\partial z}{\partial w_j}$$

- But $\partial L_{0-1}/\partial z$ is zero everywhere it's defined!



  - $\partial L_{0-1}/\partial w_j = 0$ means that changing the weights by a very small amount probably has no effect on the loss.
  - Almost any point has 0 gradient!

Attempt 2: Linear Regression

- Sometimes we can replace the loss function we care about with one which is easier to optimize. This is known as relaxation with a smooth surrogate loss function.
- One problem with L_{0-1}: defined in terms of final prediction, which inherently involves a discontinuity
- Instead, define loss in terms of $\mathbf{w}^\top \mathbf{x}$ directly
  - Redo notation for convenience: $z = \mathbf{w}^\top \mathbf{x}$

Attempt 2: Linear Regression

- We already know how to fit a linear regression model. Can we use this instead?

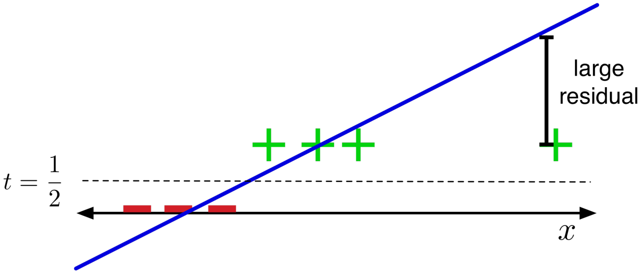
$$z = \mathbf{w}^\top \mathbf{x}$$

$$\mathcal{L}_{\mathrm{SE}}(z, t) = \frac{1}{2}(z - t)^2$$

- Doesn't matter that the targets are actually binary. Treat them as continuous values.
- For this loss function, it makes sense to make final predictions by thresholding z at $\frac{1}{2}$ (why? hint: recall the best prediction that minimizes squared loss and then how that should be turned into the best prediction that minimizes 0-1 loss.)

Attempt 2: Linear Regression

The problem:



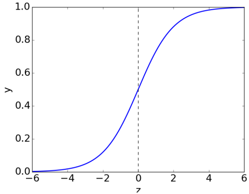
- The loss function hates when you make correct predictions with high confidence!
- If $t = 1$, it's more unhappy about $z = 10$ than $z = 0$.

Attempt 3: Logistic Activation Function

- There's obviously no reason to predict values outside [0, 1]. Let's squash y into this interval.

- The logistic function is a kind of sigmoid, or S-shaped function:

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



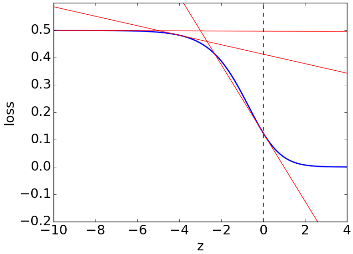
- $\sigma^{-1}(y) = \log(y/(1-y))$ is called the logit.

- Now let's try this loss and set its gradient to 0:

$$z = \mathbf{w}^\top \mathbf{x}$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\mathrm{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

Attempt 3: Logistic Activation Function

The problem: (plot of L_{SE} as a function of z, assuming $t = 1$)



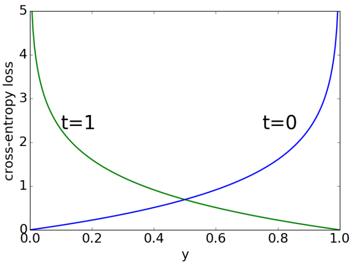
$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w_j}$$

- Let's consider a wrong prediction. For $z \ll 0$, we have $\sigma(z) \approx 0$.
- $\frac{\partial L}{\partial z} \approx 0$ (check!) $\implies \frac{\partial L}{\partial w_j} \approx 0 \implies$ derivative w.r.t. w_j is small $\implies w_j$ is like a critical point
- But this is the wrong conclusion! Our prediction is really wrong, so ideally we should be far from a critical point.

Logistic Regression

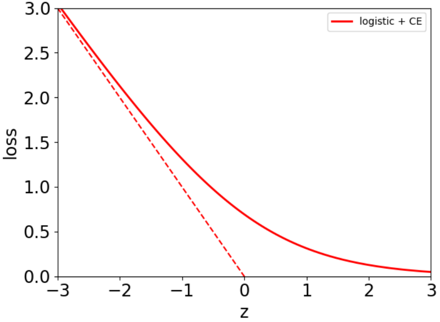
- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$. If $t = 0$, then we want to heavily penalize $y \approx 1$.
- The pundits who were 99% confident Clinton would win were much more wrong than the ones who were only 90% confident.
- Cross-entropy loss (aka log loss) captures this intuition:

$$
\begin{aligned}
L_{\mathrm{CE}}(y, t) &= \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases} \\
&= -t \log y - (1 - t) \log(1 - y)
\end{aligned}
$$



Logistic Regression:

$$
\begin{aligned}
z &= \mathbf{w}^\top \mathbf{x} \\
y &= \sigma(z) \\
&= \frac{1}{1 + e^{-z}} \\
\mathcal{L}_{\mathrm{CE}} &= -t \log y - (1 - t) \log(1 - y)
\end{aligned}
$$



Plot is for target $t = 1$.

Logistic Regression — Numerical Instabilities

- If we implement logistic regression naively, we can end up with numerical instabilities.
- Consider: $t = 1$ but you're really confident that $z \ll 0$.
- If y is small enough, it may be numerically zero. This can cause very subtle and hard-to-find bugs.

$$\begin{aligned} y &= \sigma(z) && \Rightarrow y \approx 0 \\ \mathcal{L}_{\mathrm{CE}} &= -t \log y - (1-t)\log(1-y) && \Rightarrow \text{ computes } \log 0 \end{aligned}$$

Logistic Regression — Numerically Stable Version

- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function.

$$L_{\mathrm{LCE}}(z, t) = L_{\mathrm{CE}}(\sigma(z), t) = t \log(1 + e^{-z}) + (1 - t) \log(1 + e^{z})$$

- Numerically stable computation:

```
E = t * np.logaddexp(0, -z) + (1-t) * np.logaddexp(0, z)
```

Gradient Descent for Logistic Regression

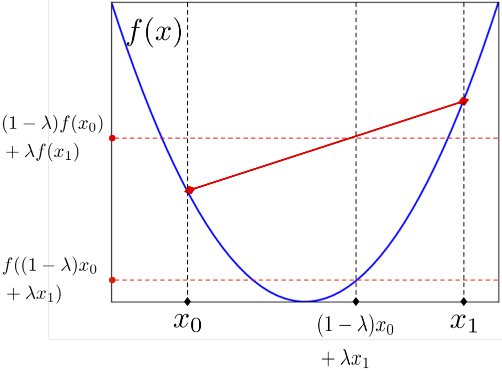
- How do we minimize the cost $\hat{\mathcal{R}}$ for logistic regression? No direct solution.
  - Taking derivatives of $\hat{\mathcal{R}}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- Perhaps we should consider using gradient descent from last lecture? But will this work?
- Luckily it will, but we should be a bit careful to understand why it works.
- It works because the logistic loss is a convex function.

Convex Functions

- A function f is convex if for any $\mathbf{x}_0, \mathbf{x}_1$ in the domain of f,

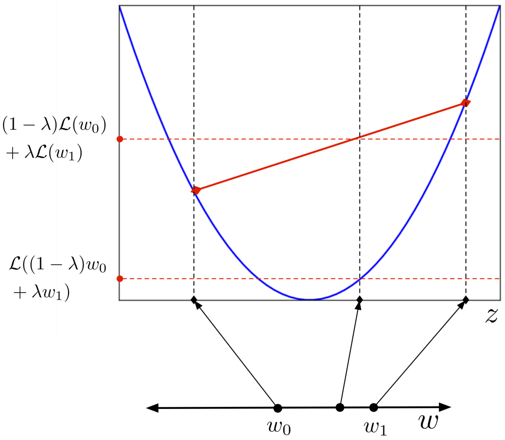
$$f((1-\lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1) \leq (1-\lambda)f(\mathbf{x}_0) + \lambda f(\mathbf{x}_1)$$

- Equivalently, the set of points lying above the graph of f is convex.
- Intuitively: the function is bowl-shaped.



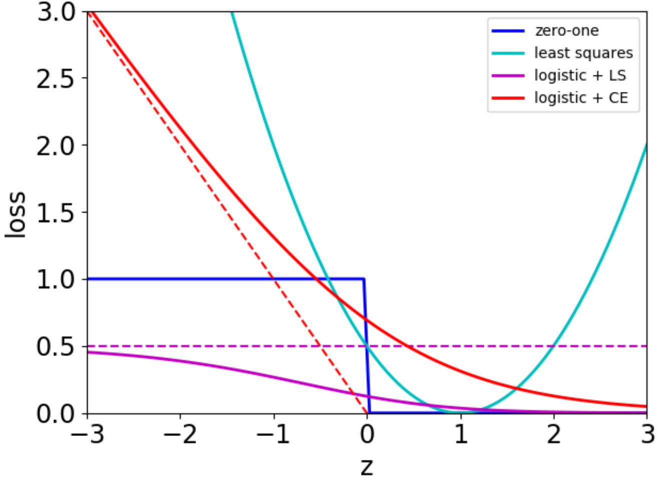
How to tell a loss is convex?

- We just saw that the least-squares loss function $\frac{1}{2}(y - t)^2$ is convex as a function of y
- For a linear model, $z = \mathbf{w}^\top \mathbf{x} + b$ is a linear function of $\mathbf{w}$ and b. If the loss function is convex as a function of z, then it is convex as a function of $\mathbf{w}$ and b.



Convex Functions, Logistic Regression

Which loss functions are convex? (these show for $t = 1$)



Convex Functions and gradient descent

- The key point here is that the logistic loss is convex.
- Convex functions have very nice properties.
  - All critical points are minima.
  - Gradient descent finds the optimal solution.
- So we can use gradient descent to find the minima of the logistic loss!
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - A standard initialization is $\mathbf{w} = 0$. (why?)

Gradient of Logistic Loss

Back to logistic regression:

$$L_{\mathrm{CE}}(y, t) = -\, t \log(y) - (1 - t) \log(1 - y)$$
$$y = 1/(1 + e^{-z}) \quad \text{and} \quad z = \mathbf{w}^{\top}\mathbf{x}$$

Therefore

$$\frac{\partial L_{\mathrm{CE}}}{\partial w_j} = \frac{\partial L_{\mathrm{CE}}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1-t}{1-y}\right) \cdot y(1-y) \cdot x_j$$
$$= (y - t) x_j$$

(verify this)

Gradient descent (for each w_j) update to find the parameters of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \hat{\mathcal{R}}}{\partial w_j}$$
$$= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)}$$

Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- Linear regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \, \mathbf{x}^{(i)}$$

- Logistic regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \, \mathbf{x}^{(i)}$$

- Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- Notice $\frac{1}{N}$ in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses ($\alpha' = \alpha/N$).