CS 490: NATURAL LANGUAGE PROCESSING

Dan Goldwasser, Abulhair Saparov

Lecture 3: Text Classification II

PREVIOUSLY

- We began discussing methods for text classification.
- Text classification tasks:
  - Spam/phishing detection
  - Sentiment analysis
  - Topic classification
  - Authorship detection
- Each example in a text classification task consists of:
  - Input: Some text
  - Output: Label/class (e.g., `SPAM` or `NOT SPAM`)

PREVIOUSLY

- We began discussing methods for text classification.
- Machine learning methods for text classification:
  - Naïve Bayes
  - Perceptron
- Today, we will cover additional machine learning methods.
  - Specifically, logistic regression
- We will also discuss how to train them.

PREVIOUSLY: PERCEPTRON

- Recall the perceptron: $f_w(x) = \text{sign}\{w^\top \Phi(x)\}$
- The input is x.
- E.g., an email that we want to classify as SPAM or NOT SPAM.
- The function Φ converts the input text into a real-valued vector $\Phi(x)$.
- $w^\top \Phi(x)$ is the dot product between the vectors w and $\Phi(x)$.
- The output is either $+1$ or -1, which we can interpret SPAM or NOT SPAM.
- Example:
  - Let's say we choose $\Phi(x)$ to consist of two features:

$$\Phi(x) = \begin{bmatrix} \text{count the number of monetary amounts that are > \$100} \\ \text{count the number of phone numbers} \end{bmatrix}$$

PREVIOUSLY: PERCEPTRON

Dear customer,

Your Subscription was successfully completed today, and your account will be credited with $400.99.Within the next 24 hours, the transaction will show up in your account statement. Please get in touch with our billing department right once if you think this transaction was not authorized or if you want to terminate your membership.

Customer Id	SDAF2354W76TER
Invoice Number	9187248935EW
Customer-Care No	+1 (951)-(262)-(3062)

- What is $\Phi(x)$?

$$\Phi(x) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- Since $\Phi(x)$ is 2-dimensional, w must also have the same dimension.
- Let's say w is the following:

$$w = \begin{bmatrix} 0.7 \\ 0.2 \end{bmatrix}$$

- Then, we can compute

$$\begin{aligned} f_w(x) &= \text{sign}\{w^T \Phi(x)\} \\ &= \text{sign}\{0.7 + 0.2\} \\ &= +1 \end{aligned}$$

- The model correctly predicts that this input is SPAM.

OTHER MACHINE LEARNING METHODS

- Perceptron model: $f_w(x) = \text{sign}\{w^T \Phi(x)\}$
- We can replace the above function with something different:
  - Another well-known classification model is logistic regression.

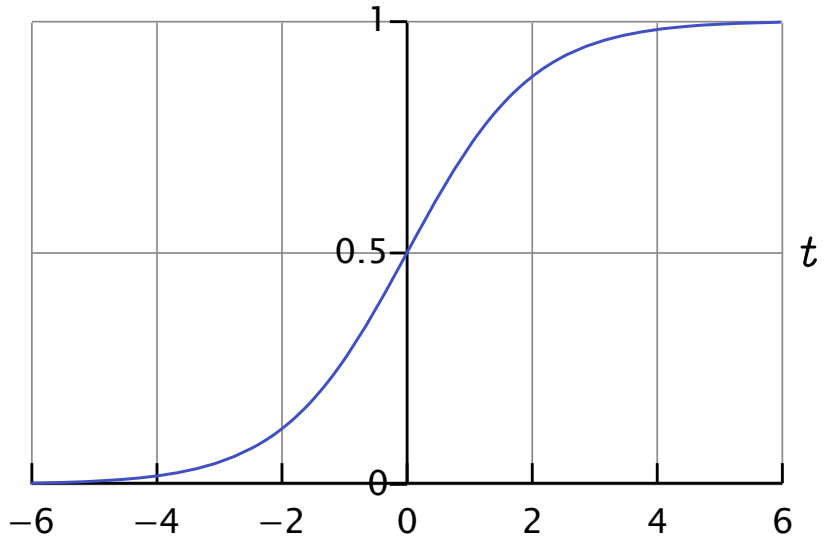
$$f_w(x) = \frac{\exp(w^T \Phi(x))}{1 + \exp(w^T \Phi(x))}$$

- $\exp(t)/(1 + \exp(t))$ is called the logistic function or sigmoid function.
  - Often written as $\sigma(t)$.
- So we can rewrite the logistic regression model equivalently:

$$f_w(x) = \sigma(w^T \Phi(x))$$

LOGISTIC REGRESSION

- Plot the logistic function: $\sigma(t) = \exp(t)/(1 + \exp(t))$

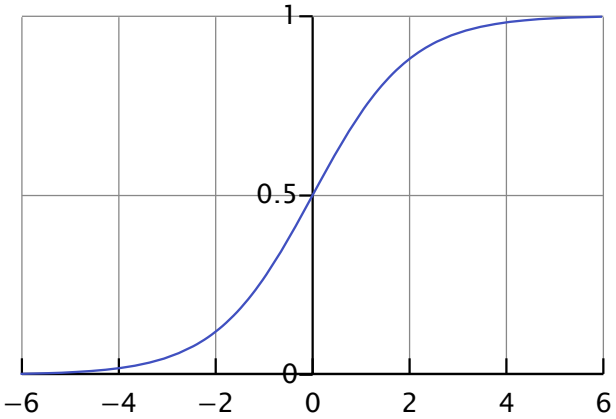


- The output of the logistic function is between 0 and 1.
- This function is commonly used to model probabilities.

LOGISTIC REGRESSION

- Logistic regression:

$$f_w(x) = \frac{\exp(w^\top \Phi(x))}{1 + \exp(w^\top \Phi(x))} = \sigma(w^\top \Phi(x))$$



- We can interpret the output of this function as the probability that the input x is classified as SPAM.
- If the dot product between w and $\Phi(x)$ is high, then the model will predict that the input is SPAM with high probability.
- If the dot product between w and $\Phi(x)$ is low, the predicted probability that the input is SPAM is low.

LINEAR DECISION BOUNDARIES

- Like perceptron and naive Bayes, logistic regression is a linear classifier.
  - They have linear classification boundaries.
  - We can determine the classification boundary of the perceptron by inspecting the values of x where the sign of $f_w(x) = w^T x$ flips. (for simplicity, assume $\Phi(x) = x$)
  - The sign flips when $w^T x = 0$,
  - What does this boundary look like?
  - Consider the case in 2 dimensions:

    $w^T x = w_1 \cdot x_1 + w_2 \cdot x_2 = 0$
  - We can rearrange the terms to write x_2 as a function of x_1.

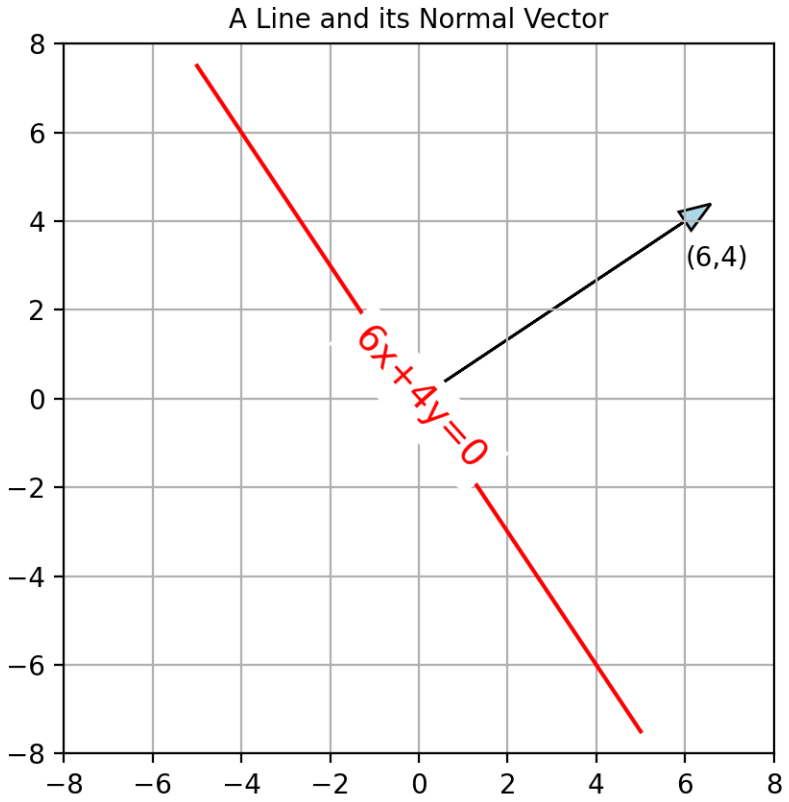
    $x_2 = (-w_1/w_2) \cdot x_1$
  - This is a line!

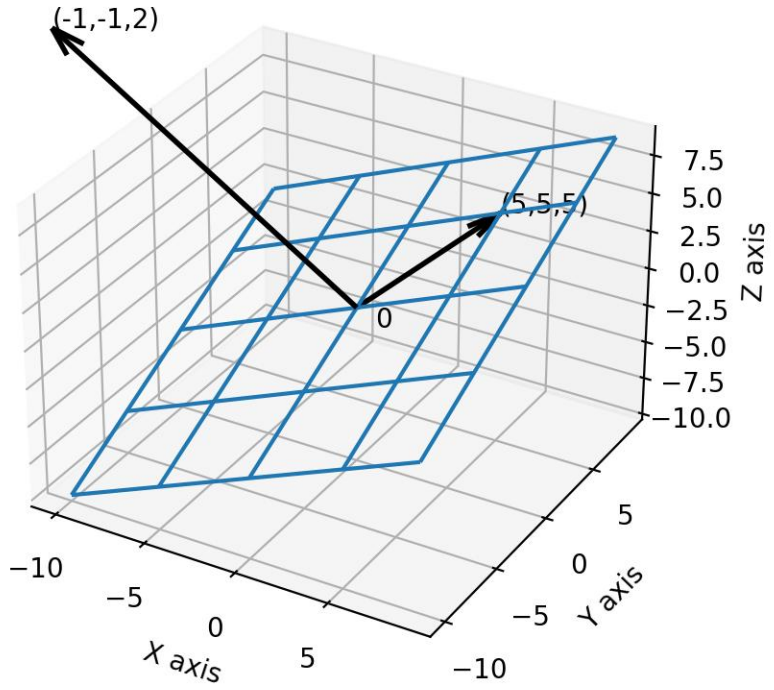
    (that passes through the origin)

LINEAR DECISION BOUNDARIES

- Like perceptron and naive Bayes, logistic regression is a linear classifier.
  - They have linear classification boundaries.
  - We can determine the classification boundary of the perceptron by inspecting the values of x where the sign of $f_w(x) = w^T x$ flips. (for simplicity, assume $\Phi(x) = x$)
  - The sign flips when $w^T x = 0$,
  - What does this boundary look like?
  - What if we have more than 2 dimensions?
  - The dot product of any two vectors is 0 if they are perpendicular.
  - The set of points x for which $w^T x = 0$ is the set of all vectors that are perpendicular to w.
  - In 3 dimensions, this is a plane (that passes through the origin).
  - In n dimensions, this is a hyperplane (that passes through the origin).

LINEAR DECISION BOUNDARIES





[avishek.net/2021/03/29/vectors-normals-hyperplanes.html]

LOGISTIC REGRESSION

- What is the decision boundary for logistic regression?
- The boundary is where the logistic function is 0.5.

$$\frac{1}{1 + \exp(w^T x)} = \frac{1}{2}$$

$$1 + \exp(w^T x) = 2$$

$$\exp(w^T x) = 1$$

$$w^T x = 0$$

- The decision boundary for logistic regression is also linear (i.e., a hyperplane),
  - If we assume $\Phi(x) = x$.
  - As discussed last lecture, we can use more complex feature functions $\Phi(x)$ to obtain non-linear decision boundaries.

TRAINING

- How do we learn the weights $\mathbf{w}$?
- Let's continue our running example of training a spam detection model.
- Let $x_1, x_2, \ldots, x_N$ be the set of emails in the training dataset.
- Let $y_1, y_2, \ldots, y_N$ be the set of labels,

  where $y_i = +1$ if the i-th email is SPAM,

  and $y_i = -1$ if the i-th email is NOT SPAM.
- To form an optimization problem, we need a loss function.
- The loss function measures the "distance" from the current model's predictions and the ground truth.

TRAINING

- There are many different choices for loss functions.
- But for probabilistic models like naïve Bayes and logistic regression, the likelihood is a common choice for the loss function.
- The likelihood is the probability of the outputs/observations, conditioned on the inputs.

$$p(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(y_i \mid x_i)$$

since each example is independent of each other

$$= \prod_{i=1}^{N} \underbrace{\sigma(w^T \Phi(x_i))}^{\mathbb{1}\{y_i \text{ is SPAM}\}} \underbrace{\left(1 - \sigma(w^T \Phi(x_i))\right)}^{\mathbb{1}\{y_i \text{ is NOT SPAM}\}}$$

this is probability that the logistic regression model will predict "positive"

this is probability that the logistic regression model will predict "negative"

- If we interpret $y_i = 1$ as positive (SPAM) and $y_i = 0$ as negative (NOT SPAM), we can write the likelihood equivalently:

$$= \prod_{i=1}^{N} \sigma(w^T \Phi(x_i))^{y_i} \left(1 - \sigma(w^T \Phi(x_i))\right)^{1 - y_i}$$

TRAINING

- So we have the following likelihood:

$$p(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots, x_N)$$
$$= \prod_{i=1}^{N} \sigma(w^T \Phi(x_i))^{y_i} \left(1 - \sigma(w^T \Phi(x_i))\right)^{1 - y_i}$$

- Given a training dataset, the x_i and y_i are fixed (i.e., constant).
- The only variable above is w.
- To train a probabilistic model, we want to maximize the likelihood.
  - That is, we want to find the value of w that maximizes the above function.
  - I.e., find the maximum likelihood estimate (MLE) of w.
- This is an example of optimization.
  - We want to optimize the function above and find the value of w that maximizes it.

TRAINING

- So we have the following likelihood:

$$p(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} \sigma(w^T \Phi(x_i))^{y_i} \left(1 - \sigma(w^T \Phi(x_i))\right)^{1 - y_i}$$

- Notice that if our training set is large (i.e., N is large), the above expression will be a product of many terms between 0 and 1.
  - This will result in a very small numeric value that can be rounded to 0.
  - Recall from last lecture: this is called underflow.
- What can we do to avoid this?

TRAINING

- Instead of maximizing the likelihood, we maximize the log likelihood.

$$\log p(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots, x_N)$$
$$= \sum_{i=1}^{N} y_i \log(\sigma(w^T \Phi(x_i))) + (1 - y_i) \log\left(1 - \sigma(w^T \Phi(x_i))\right)$$

- For any a and b, if a > b, then log(a) > log(b).
- In machine learning, we pose the optimization problem as a minimization of a loss function.
  - You can think of "high loss" as bad, since it means the model is very far from the ground truth.
  - Thus, we want to minimize the loss.
- To turn the likelihood or log likelihood into a loss function, we simply negate it.
  - Minimizing the negative log likelihood (NLL) is equivalent to maximizing the log likelihood.

TRAINING

- We have an objective function that we want to optimize:

$$L(w) = -\sum_{i=1}^{N} y_i \log(\sigma(w^T \Phi(x_i))) + (1 - y_i) \log\left(1 - \sigma(w^T \Phi(x_i))\right)$$

- From calculus, one way to find optima is to take the derivative of $L(w)$, set it equal to 0, and then solve for w.
- This works for naïve Bayes!
  - If you do this derivation on the naïve Bayes likelihood, you will get exactly the solution from last lecture!
- But for many functions, there are no closed-form solutions for w.
  - This includes most loss functions in machine learning.
  - As well as the function above.
- We need a different approach to optimize the above loss function.

GRADIENT ASCENT

- Consider the following algorithm:
- Start at some initial point $\mathtt{w}_0$.
- Repeat the following:
  - Find the direction $\mathtt{g}$ at the current position $\mathtt{w}_t$ where the function $\mathtt{L(w)}$ is increasing most steeply.
  - Move a small distance in the direction of $\mathtt{g}$.
  - New position is $\mathtt{w}_{t+1}$.
- Eventually, for large enough T, $\mathtt{w}_T$ will be a maximum of $\mathtt{L(w)}$.

- Consider the analogy of climbing a hill or a mountain.
  - $\mathtt{L(w)}$ is the altitude of the current coordinate $\mathtt{w}$.
  - If we keep moving in the direction of steepest ascent, we will eventually reach the top of the nearest hill or mountain.

GRADIENT ASCENT

- How do we find the direction of steepest ascent?
- This is the gradient.
- The gradient of a function $L(w)$ is written as ∇L or $\nabla_w L$.

$$\nabla L = \begin{bmatrix} \partial L / \partial w_1 \\ \vdots \\ \partial L / \partial w_n \end{bmatrix}$$

- $\partial L / \partial w_i$ is the partial derivative of L with respect to w_i.
  - This is the derivative of L where we consider everything is constant except w_i.

GRADIENT EXAMPLE

- Consider the function $L(w) = \exp(-(w_1^2 + w_2^2))$.
- This function looks like a smooth hill.
- We can compute its gradient:

$$\partial L / \partial w_1 = -2w_1 \cdot \exp(-(w_1^2 + w_2^2))$$

$$\partial L / \partial w_2 = -2w_2 \cdot \exp(-(w_1^2 + w_2^2))$$

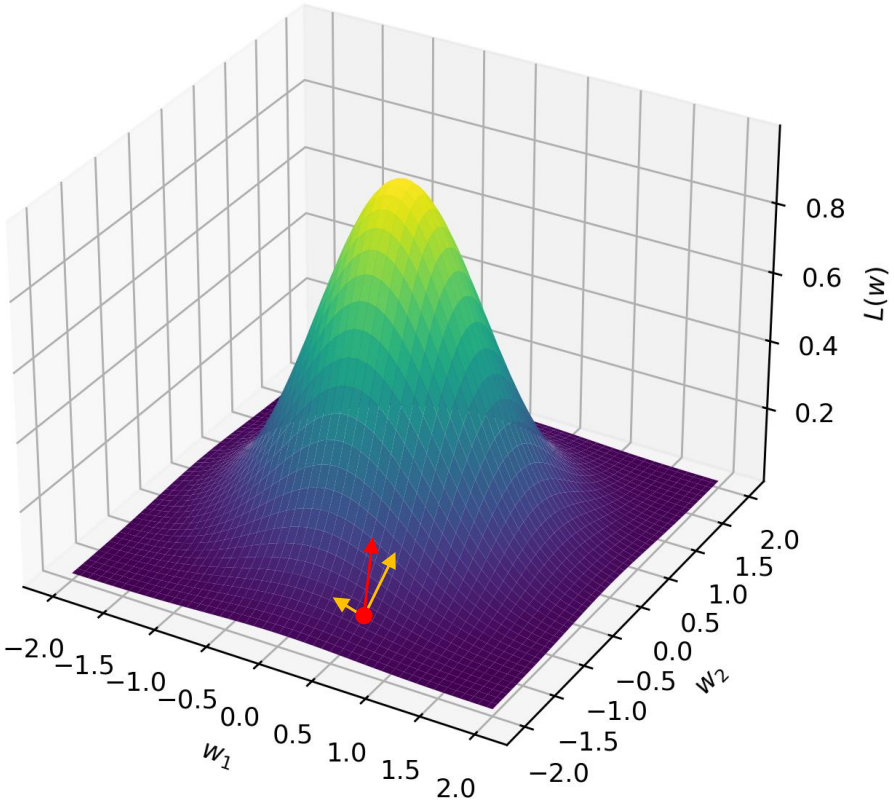
- So the gradient at $(0.5, -1.0)$ is:

$$[-\exp(-(0.25 + 1)),\ 2 \cdot \exp(-(0.25 + 1))]$$

$$= [-\exp(-1.25),\ 2 \cdot \exp(-1.25)]$$

$$= [-0.29,\ 0.57]$$

- This vector points toward the origin/peak.



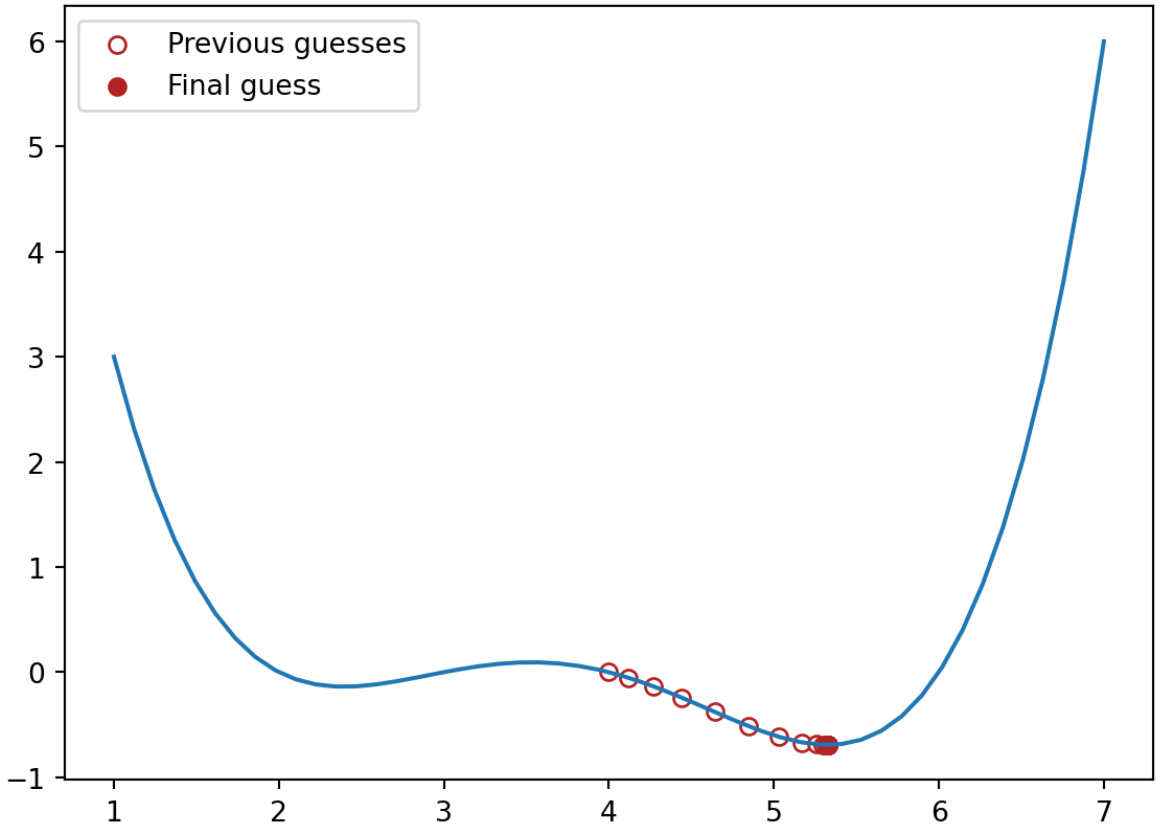
GRADIENT ASCENT

- Now we can write the full gradient ascent algorithm:
- Start at some initial point $\mathbf{w}_0$.
- Repeat the following for $t = 0, 1, 2, \ldots, T$:
  - Compute the gradient ∇L at $\mathbf{w}_t$. For shorthand, this can be written as $\nabla L|_{\mathbf{w}_t}$.
  - Set the new position $\mathbf{w}_{t+1}$ to $\mathbf{w}_t + \eta \cdot \nabla L|_{\mathbf{w}_t}$.
- η is the learning rate.
- Eventually, for large enough T, $\mathbf{w}_T$ will be a maximum of $L(\mathbf{w})$.

GRADIENT DESCENT

- In machine learning, we often want to minimize the loss function.
- To do so, we simply flip the direction of movement:
  - Instead of moving toward the direction of steepest ascent, we move toward steepest descent.
- Start at some initial point w_0.
- Repeat the following for $t = 0, 1, 2, \ldots, T$:
  - Compute the gradient ∇L at w_t.
  - Set the new position w_{t+1} to $w_t - \eta \cdot \nabla L|_{w_t}$.
- This is called gradient descent.

GRADIENT DESCENT



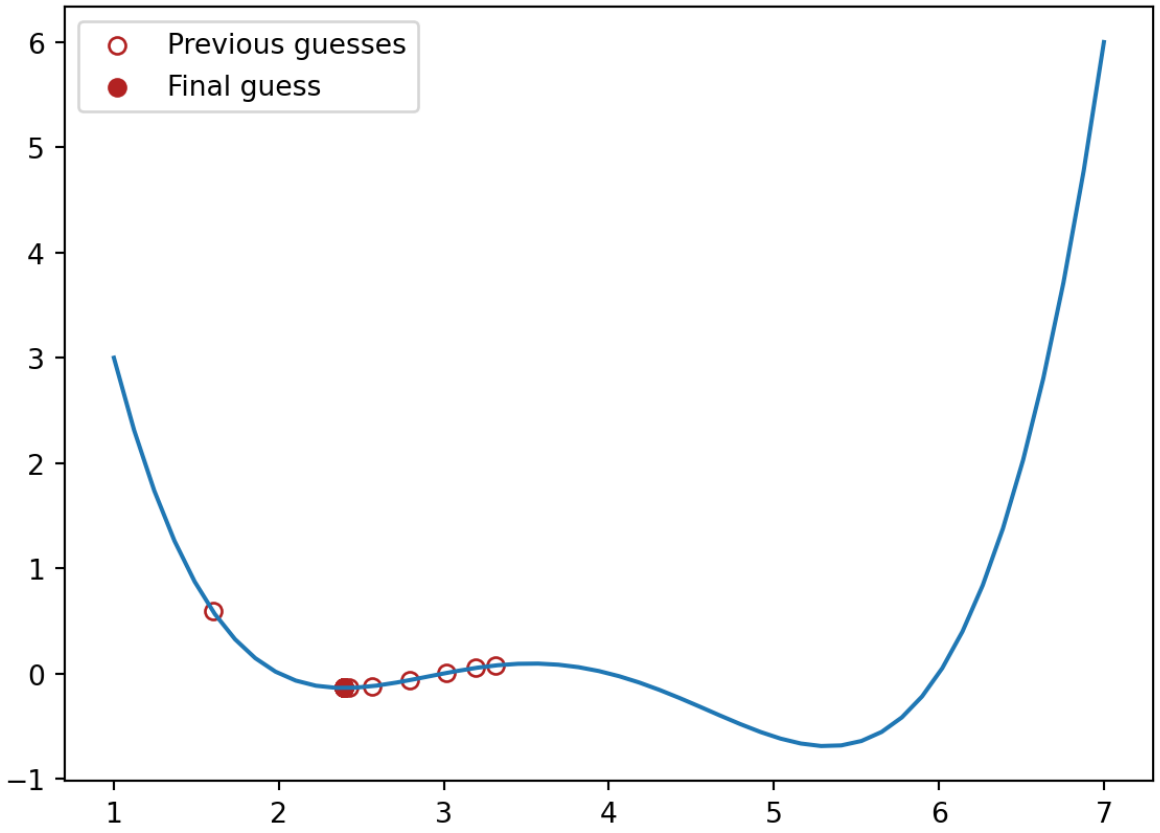
The learning rate η must be carefully set.

What happens if it's too large?

Too small?

[ds100.org/course-notes-su2]

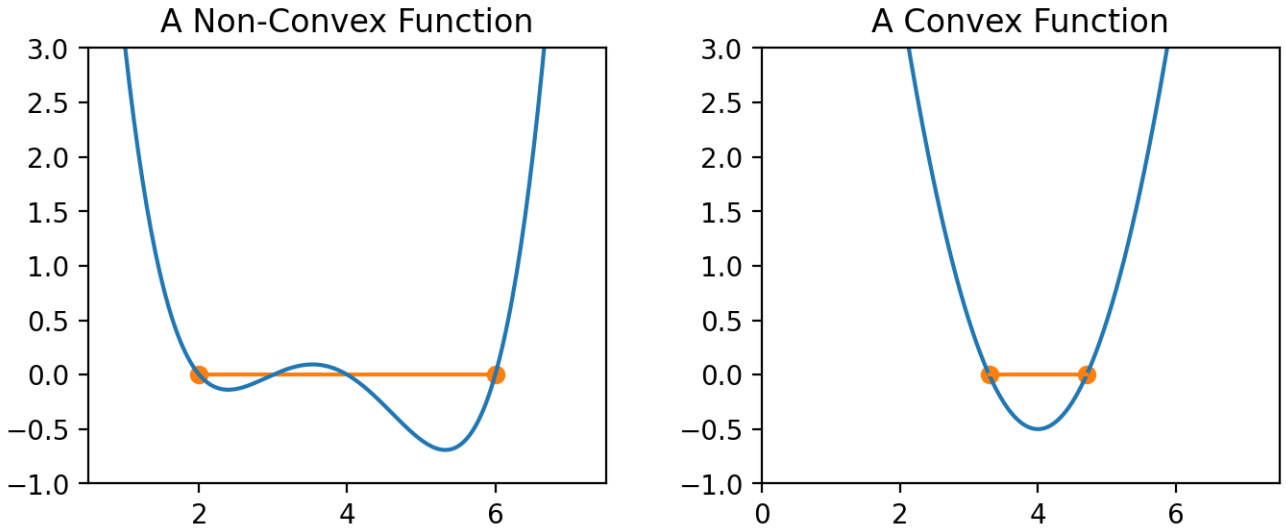
GRADIENT DESCENT



Gradient descent may find a local minimum.

[ds100.org/course-notes-su2]

CONVEXITY



For convex functions, if we find the local minimum, it is guaranteed to be globally minimal.
Many real-world functions are not convex.
For example, neural network training objectives are non-convex.
Fun fact: The logistic regression objective function is convex!

[ds100.org/course-notes-su2]

TRAINING

- So now we have all the ingredients to train a logistic regression classification model.
- What is the gradient for logistic regression?
- Recall:

$$\mathrm{L}(\mathrm{w}) = -\sum_{i=1}^{N} \mathrm{y}_i \log(\boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i))) + (1 - \mathrm{y}_i)\log\left(1 - \boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i))\right)$$

$$\nabla_{\mathrm{w}}\mathrm{L} = -\sum_{i=1}^{N} \mathrm{y}_i \nabla_{\mathrm{w}}\log(\boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i))) + (1 - \mathrm{y}_i)\nabla_{\mathrm{w}}\log\left(1 - \boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i))\right)$$

- We can utilize a neat fact about derivatives of the logistic function:

$$\frac{d}{dt}\log(\boldsymbol{\sigma}(t)) = 1 - \boldsymbol{\sigma}(t) \text{ and } \frac{d}{dt}\log(1 - \boldsymbol{\sigma}(t)) = -\boldsymbol{\sigma}(t)$$

- Which we use to simplify the gradient above:

$$\nabla_{\mathrm{w}}\mathrm{L} = -\sum_{i=1}^{N} \mathrm{y}_i (1 - \boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i)))\Phi(\mathrm{x}_i) - (1 - \mathrm{y}_i)\boldsymbol{\sigma}(\mathrm{w}^{\mathsf{T}}\Phi(\mathrm{x}_i))\Phi(\mathrm{x}_i).$$

TRAINING

- So now we have all the ingredients to train a logistic regression classification model.
- What if the training set is very large? (i.e., we have > millions of emails)
  - The training set will not fit in memory.
- Computing the gradient requires iterating over the full dataset.

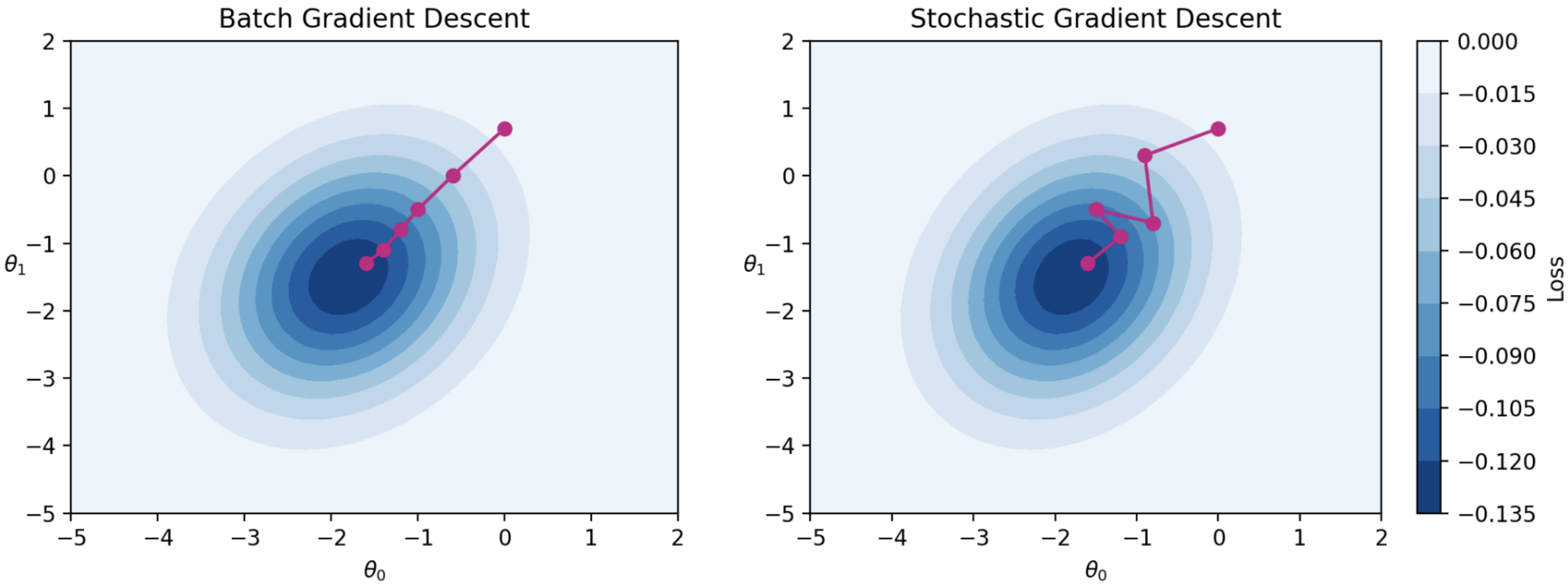
$$\nabla_w L = -\sum_{i=1}^{N} y_i (1 - \sigma(w^T \Phi(x_i))) \Phi(x_i) - (1 - y_i) \sigma(w^T \Phi(x_i)) \Phi(x_i)$$

- Instead, we can estimate the gradient by randomly sampling one example at each iteration:

$$\nabla_w L \approx -y_i (1 - \sigma(w^T \Phi(x_i))) \Phi(x_i) + (1 - y_i) \sigma(w^T \Phi(x_i)) \Phi(x_i)$$

- This is a noisy estimate of the gradient.
- This optimization algorithm is called stochastic gradient descent (SGD).

STOCHASTIC GRADIENT DESCENT



MINI-BATCH GRADIENT DESCENT

- Estimating the gradient with a single training example is too noisy.
- Instead, for each iteration, we can sample a batch of B random training examples.
  - B is the batch size.
  - Let $b_1, \ldots, b_B$, be the indices of the batch:

$$\nabla_w L \approx -\sum_{i=1}^{B} y_{b_i} (1 - \sigma(w^T \Phi(x_{b_i}))) \Phi(x_{b_i}) - (1 - y_{b_i}) \sigma(w^T \Phi(x_{b_i})) \Phi(x_{b_i})$$

- This is also oftentimes called stochastic gradient descent.

- Gradient descent methods only use the first derivative.
- There are other optimization algorithms that use the second derivative (Hessian).
- But they are more expensive, especially if we have many parameters (w is very large).
- There are algorithms that approximate parts of the Hessian using the gradient.
  - E.g., Adam, Sophia

IMPLEMENTING SGD

- Deriving gradients by hand can be tedious.
- Luckily, there are many machine learning libraries that perform automatic differentiation.
  - These libraries can compute the gradient of a complex function by repeatedly apply the chain rule on simpler components of the function.
  - E.g. `L(w) = f(g(h(w)))`.
  - $\frac{dL}{dw} = \frac{df}{dg}\frac{dg}{dh}\frac{dh}{dw}$.
- These libraries include `PyTorch` and `TensorFlow`.

IMPLEMENTING SGD IN PYTORCH

```python
import torch
import torch.nn as nn

# create model
class LogisticRegression(nn.Module):

    # n_input_features is the input dimension
    def __init__(self, n_input_features):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(n_input_features, 1) # this defines a dot product operation

    def forward(self, x):
        y_predicted = torch.sigmoid(self.linear(x))
        return y_predicted

model = LogisticRegression(n_features)

# construct optimizer
learning_rate = 0.01
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

[adapted from Coelho, 2023, Logistic Regression — PyTorch]

IMPLEMENTING SGD IN PYTORCH

```
# training loop
num_epochs = 100
for epoch in range(num_epochs):
    # this computes the predicted probabilities
    y_predicted = model(X_train)
    loss = criterion(y_predicted, y_train)

    # computes the gradient
    loss.backward()

    # performs the gradient update step
    optimizer.step()
    optimizer.zero_grad()
```

(not shown: how to construct the training data `X_train` and `y_train`)

[adapted from Coelho, 2023, Logistic Regression — PyTorch]

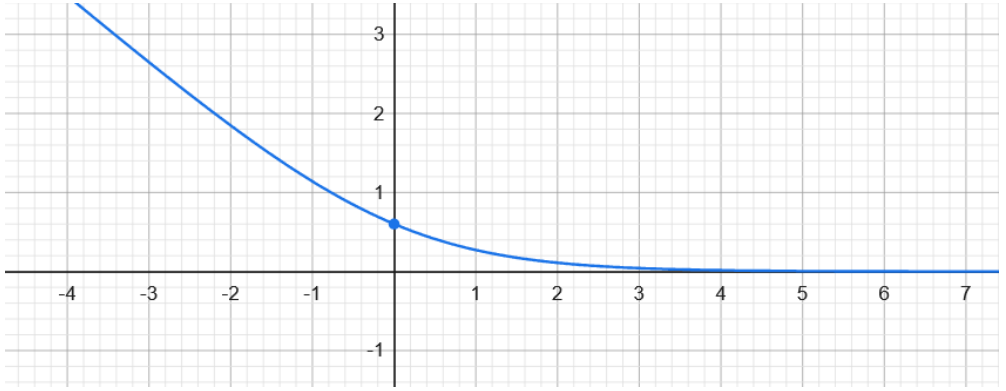
POTENTIAL PROBLEM

- Let's go back to the logistic regression objective:

$$L(w) = -\sum_{i=1}^{N} y_i \log(\sigma(w^T \Phi(x_i))) + (1 - y_i)\log\big(1 - \sigma(w^T \Phi(x_i))\big)$$

- Suppose we use a simple 1-dimensional feature: $\Phi(x_i) = x_i$.
- And we have two training points: x_1 = -1, y_1 = 0, and x_2 = +1, y_2 = 1.
- We can simplify the objective function above:

$$\begin{aligned} L(w) &= -y_2 \log(\sigma(w x_2)) - (1 - y_1)\log(1 - \sigma(w x_1)) \\ &= -\log(\sigma(w)) - \log(1 - \sigma(-w)) \end{aligned}$$



OVERFITTING

- Let's go back to the logistic regression objective:

$$L(w) = -\sum_{i=1}^{N} y_i \log(\sigma(w^T \Phi(x_i))) + (1 - y_i) \log\left(1 - \sigma(w^T \Phi(x_i))\right)$$

- Suppose we use a simple 1-dimensional feature: $\Phi(x_i) = x_i$.
- And we have two training points: $x_1 = -1$, $y_1 = 0$, and $x_2 = +1$, $y_2 = 1$.
- The model will learn progressively larger values for w.
  - It will keep growing towards infinity.
- Consider the probability predicted by logistic regression: $f_w(x) = \sigma(w^T \Phi(x))$.
- If w is large, then the predicted probabilities will be either very close to 1 or very close to 0.
  - This will happen in logistic regression whenever the training data is linearly separable.
  - The trained model is overly confident.
  - It will fit the training data very well, but may generalize poorly to test data.
  - This is called overfitting.

REGULARIZATION

- How can we avoid overfitting?
- We can use regularization.
  - We simply add a term to the loss function that encourages the parameter $\mathbf{w}$ to be "well-behaved".
- L2 regularization:

  $L(\mathbf{w}) = L_{\text{unregularized}}(\mathbf{w}) + \lambda \cdot \sum_{i=1}^{n} w_i^2$

  (the sum is the squared L2 norm of $\mathbf{w}$)
- L1 regularization:

  $L(\mathbf{w}) = L_{\text{unregularized}}(\mathbf{w}) + \lambda \cdot \sum_{i=1}^{n} |w_i|$

  (the sum is the L1 norm of $\mathbf{w}$)
- λ is the regularization parameter/rate.
- Since we aim to minimize the loss function, the regularization terms will encourage the optimization to keep $\mathbf{w}$ from becoming too large.
  - Especially if λ is large.

REGULARIZATION

- For probabilistic models, regularization is equivalent to placing a prior on $\mathbf{w}$.
- Suppose we put a Gaussian prior on $\mathbf{w}$ (also called multivariate normal).
  - The mean of the prior is 0, and its variance is $1/(2\lambda)$.
- The posterior of $\mathbf{w}$ is:

$$p(\mathbf{w} \mid y_1, y_2, \ldots, y_N, x_1, x_2, \ldots, x_N) = \frac{p(y_1, y_2, \ldots, yN \mid \mathbf{w}, x_1, x_2, \ldots, xN)\, p(\mathbf{w})}{p(y_1, y_2, \ldots, yN, x_1, x_2, \ldots, xN)}$$

- Our goal is to find $\mathbf{w}$ that maximizes the posterior (MAP estimate).
  - The denominator doesn't depend on $\mathbf{w}$, so we can replace it with a constant.
- So we can write the log posterior:

$$= \text{constant} + \log p(y_1, y_2, \ldots, yN \mid \mathbf{w}, x_1, x_2, \ldots, xN) + \log p(\mathbf{w})$$

- The left term is the log likelihood, and the right term is the log prior.

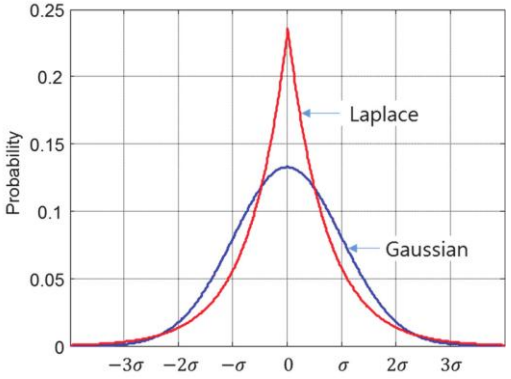
REGULARIZATION

- If the prior distribution is a Gaussian, the log prior will look like:

$$\log p(w) = -(n/2)\log(2\pi) + (n/2)\log(2\lambda) + \lambda \cdot \sum_{i=1}^{n} w_i^2$$

- The first two terms are constant with respect to w,
  - So we can ignore them during optimization.
- The last term is exactly the L2 regularization term!
- Therefore, MLE with L2 regularization is equivalent to MAP with a Gaussian prior on w.
- This is true for any probabilistic model, not just logistic regression.
- We can similarly obtain L1 regularization

  if we change the prior to a Laplace distribution.



[Kim and Bang, 2017]

MULTI-CLASS CLASSIFICATION

- So far, we only considered binary classification.
  - Where there are only two output classes.
- Consider the sentiment analysis task:
  - Given a user review of a product, the task is to classify whether the review is positive, negative, or neutral.
- Logistic regression can be extended to the multi-class setting:
  - Suppose we have K output classes.
  - The probability that the output y is k, given the input x, is:

$$p(y = k) = \frac{\exp(w_k^T \Phi(x))}{\sum_{j=1}^{K} \exp(w_j^T \Phi(x))}$$

  - Notice that now we have K weight vectors.
  - We compute the above probability for all k. The output is now a k-dimensional vector that sums to 1.
  - Note that $f(\alpha) = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp(\alpha_j)}$ is also known as the softmax function.

MULTI-CLASS CLASSIFICATION

- How do we train multi-class logistic regression?
- For probabilistic models, like logistic regression, we can use cross-entropy loss:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p(y_i=k)\log f_w(x_i)_k$$

- Since we know the ground truth labels y_i, $p(y_i=k) = 1$ if the ground truth label for the i-th example is k, and 0 otherwise.

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N}\log f_w(x_i)_{y_i}$$

convex for logistic regression!

- Note that $f_w(x_i)_{y_i}$ is the model's prediction of the probability that the i-th example has label y_i.
  - This is the likelihood of the i-th example.
  - When learning, we are minimizing the loss, which is equivalent to the negative log likelihood.
  - Therefore, minimizing the cross-entropy loss is equivalent to maximizing the likelihood.
  - This is true for any probabilistic model, not just logistic regression.

QUESTIONS?