Training Neural Networks 1

Katharina Kann — CSCI/LING5832

Let’s Talk about Weights…

- So… where do weights come from?

Let’s Talk about Weights…

- So… where do weights come from?
- We’ll learn the weights from a training set labeled with the right answer.
    - For instance, for sentiment analysis, we can use 1 and 0 as labels to stand for the right answer

Let's Talk about Weights…

- So… where do weights come from?
- We'll learn the weights from a training set labeled with the right answer.
    - For instance, for sentiment analysis, we can use 1 and 0 as labels to stand for the right answer
- The system answers will be between 0 and 1
    - (Why?)

Loss Functions

Let’s Talk about Weights…

- To learn the weights, we need some measure of how well (or badly) we’re doing with a current set of weights.
  - We’ll call that measure a **loss function**

Let’s Talk about Weights…

- To learn the weights, we need some measure of how well (or badly) we’re doing with a current set of weights.
    - We’ll call that measure a **loss function**
- The lower the loss the better we’re doing

Let’s Talk about Weights…

- To learn the weights, we need some measure of how well (or badly) we’re doing with a current set of weights.
  - We’ll call that measure a **loss function**
- The lower the loss the better we’re doing
- We want to minimize the loss

Loss Functions

- There are lots of ways to measure how well we're doing on a test/validation set.
  - How many examples are right?

Loss Functions

- There are lots of ways to measure how well we're doing on a test/validation set.
    - How many examples are right?
    - How close our answers are to the correct answers?
        - What does close mean?

Cross-Entropy Loss

- For a given example, we’ll go with the probability assigned to the right answer by the model

Cross-Entropy Loss

- For a given example, we’ll go with the probability assigned to the right answer by the model
- In the binary case, the right answer is always either 1 or 0.
    - The model produces a number between 1 and 0, that’s the starting point for the loss.

Cross-Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- y is the correct answer
- $\hat{y}$ is the system answer

Cross-Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- If the correct answer for an example is 1 and the system output is .7 then the probability of the correct answer is…

Cross-Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- If the correct answer for an example is 1 and the system output is .7 then the probability of the correct answer is…
    - .7

Cross-Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- If the correct answer for an example is 0 and the system output is .8 then the probability of the correct answer is…

Cross-Entropy Loss

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

- If the correct answer for an example is 0 and the system output is .8 then the probability of the correct answer is…
  - .2

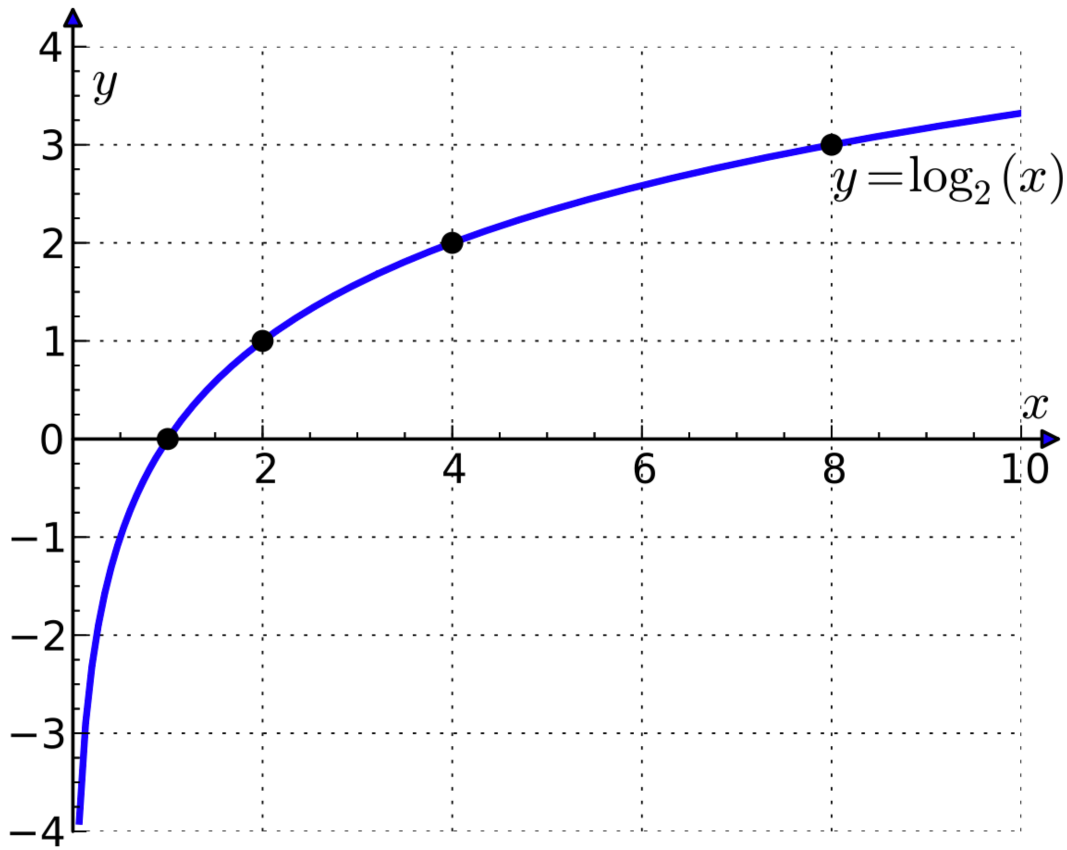
Logs

- Probabilities aren’t exactly what we want. We want worse performance to have high loss and good performance to have low loss.

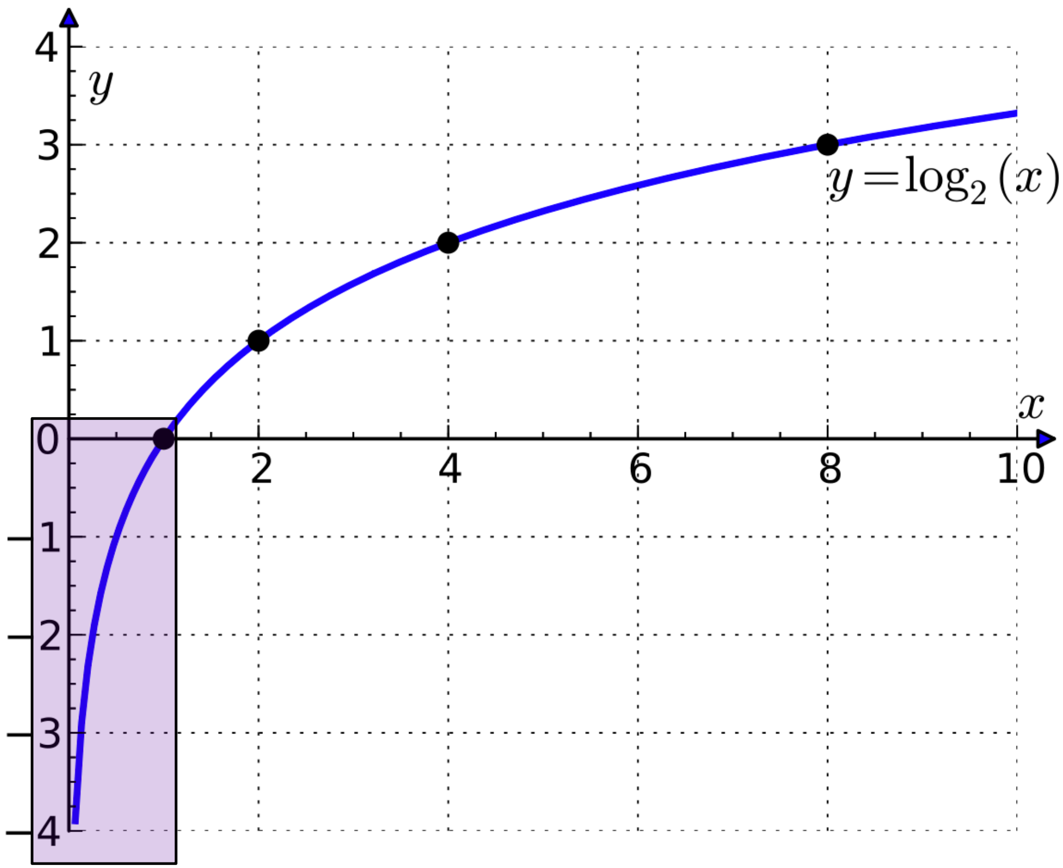
Logs

- Probabilities aren’t exactly what we want. We want worse performance to have high loss and good performance to have low loss.
- So we’ll take the negative of the log of the probability assigned to the correct answer as the loss.

Log Probability



Log Probability



Cross-Entropy Loss for Logistic Regression

$$p(y|x) = \hat{y}^{y} (1 - \hat{y})^{1-y}$$

Cross-Entropy Loss for Logistic Regression

$$p(y|x) \;=\; \hat{y}^{y}\,(1-\hat{y})^{1-y}$$

$$\begin{aligned}
\log p(y|x) \;&=\; \log\left[\hat{y}^{y}\,(1-\hat{y})^{1-y}\right] \\
&=\; y\log\hat{y} + (1-y)\log(1-\hat{y})
\end{aligned}$$

Cross-Entropy Loss for Logistic Regression

$$p(y|x) \;=\; \hat{y}^{y}\,(1-\hat{y})^{1-y}$$

$$\begin{aligned}\log p(y|x) \;&=\; \log\left[\hat{y}^{y}\,(1-\hat{y})^{1-y}\right] \\ &=\; y\log\hat{y}+(1-y)\log(1-\hat{y})\end{aligned}$$

$$L_{CE}(w,b) \;=\; -[y\log\sigma(w\cdot x+b)+(1-y)\log\left(1-\sigma(w\cdot x+b)\right)]$$

In-Class Exercise

- You are classifying tweets into positive and negative, using a logistic regression classifier.
- Compute the cross-entropy loss for
  - y = 1, y_hat = 0.66
  - y = 1, y_hat = 0.7
  - y = 0, y_hat = 0.2
  - y = 0, y_hat = 0.8
- How many examples have been predicted correctly by your classifier?

Learning

- We want to find the weights that minimize the average loss of an entire training set.

$$\mathscr{L}(\Theta) = \frac{1}{m} \Sigma_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

Learning

- We'll do this by starting with a random set of weights and then iteratively updating those weights to lower this cost.

$$\mathscr{L}(\Theta) = \frac{1}{m}\Sigma_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

Learning

Updated weights

Current weights

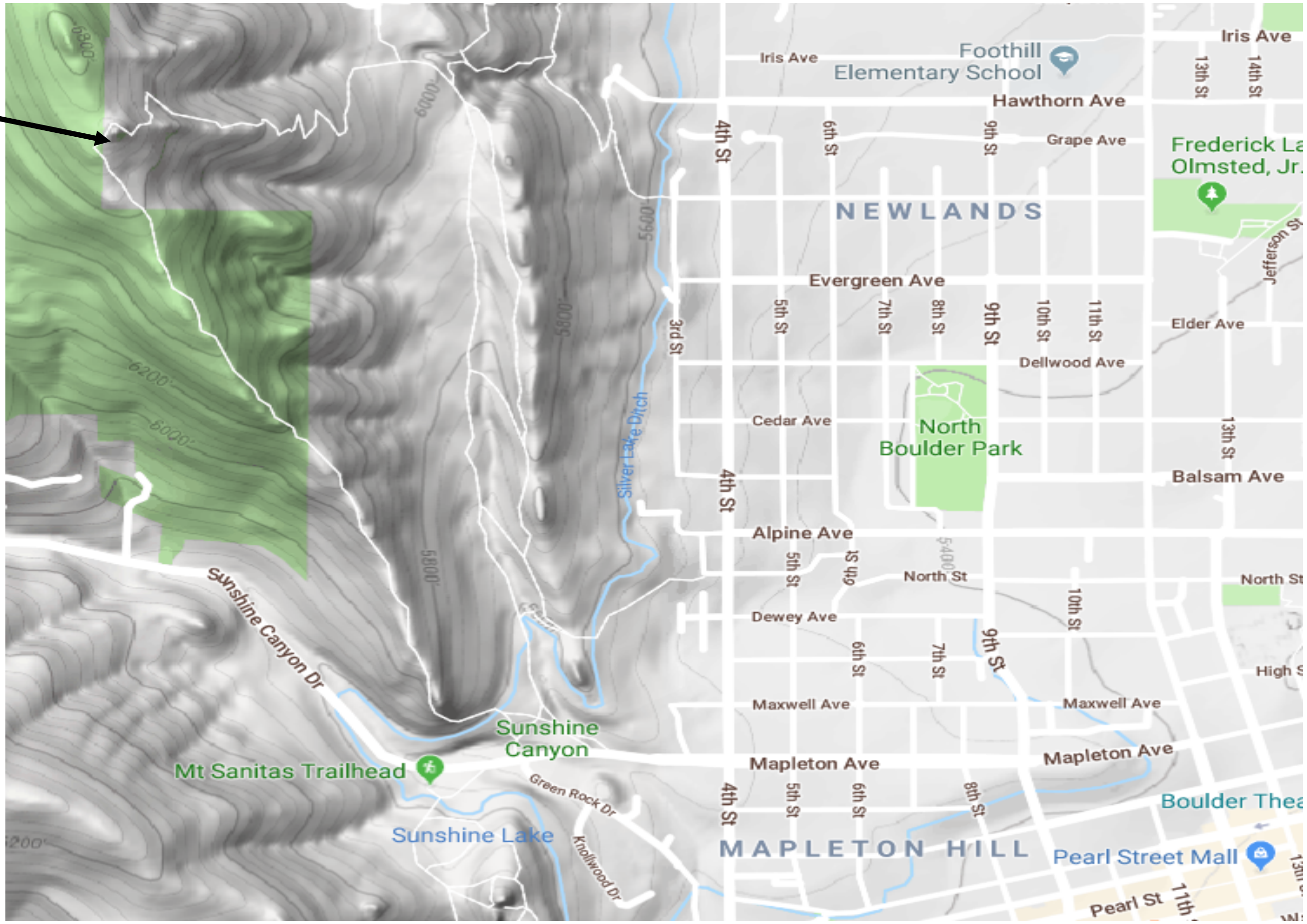
$$w_{t+1} = w_t - \mu\Delta$$

Learning rate

Gradient (vector of partial derivatives of the weights with respect to the loss).

Motivation

This is you.
Find the fastest
way down.



Derivatives

- Fortunately, we know how to do that from calculus.
- If we take the derivative of the loss function with respect to the weights that will tell us the direction and magnitude of change we should make to each weight.

Derivatives

$$\frac{\mathrm{d}}{\mathrm{dx}} x^n = n x^{n-1} \quad (1)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} cos(x) = -sin(x) \quad (2)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} sin(x) = cos(x) \quad (3)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} log(x) = \frac{1}{x} \quad (4)$$

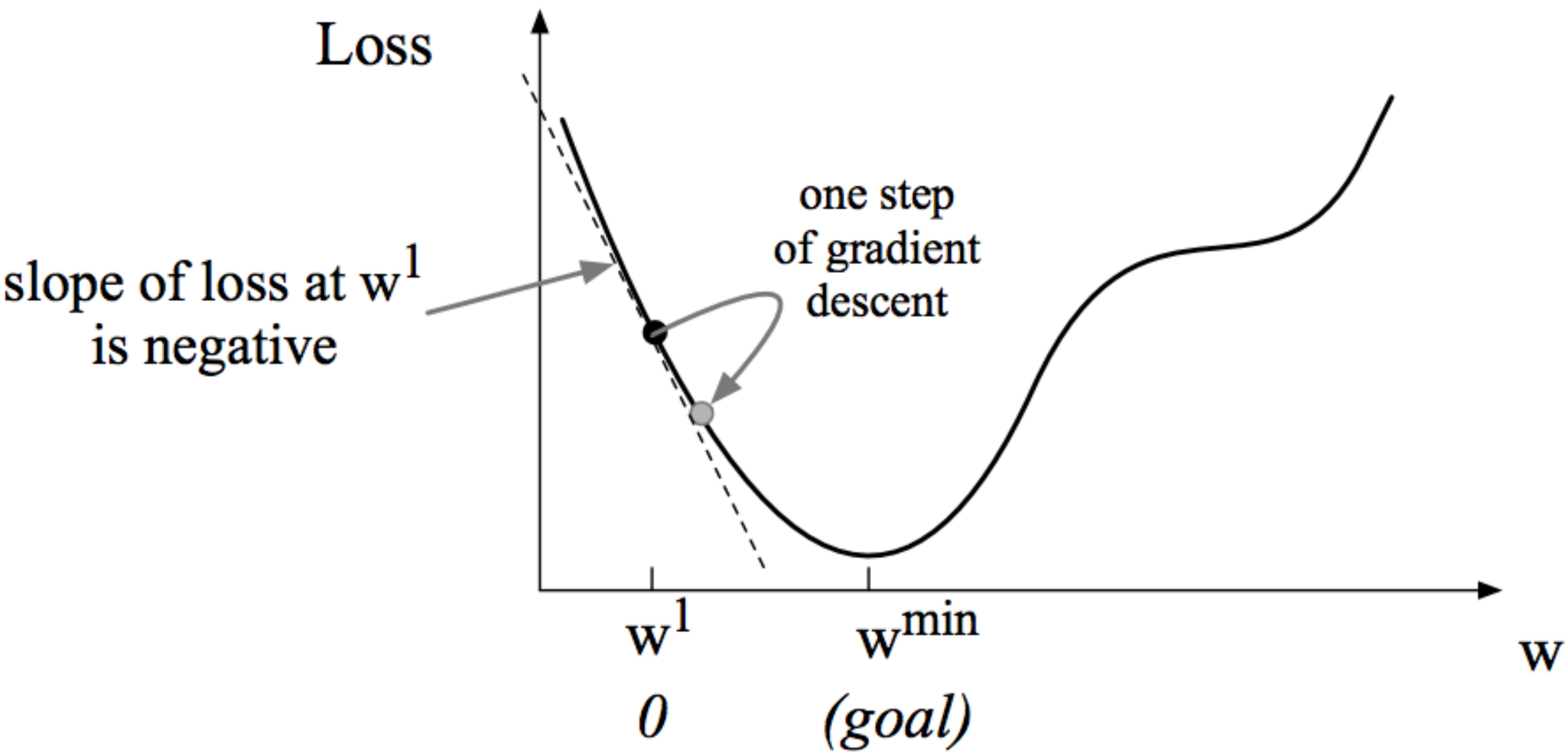
$$\frac{\mathrm{d}}{\mathrm{dx}} (b + cx) = c \quad (5)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} (f(x) + g(x)) = \frac{\mathrm{d}}{\mathrm{dx}} f(x) + \frac{\mathrm{d}}{\mathrm{dx}} g(x) \quad (6)$$

$$\frac{\mathrm{d}}{\mathrm{dx}} (f(x) g(x)) = g(x) \frac{\mathrm{d}}{\mathrm{dx}} f(x) + f(x) \frac{\mathrm{d}}{\mathrm{dx}} g(x) \quad (7)$$

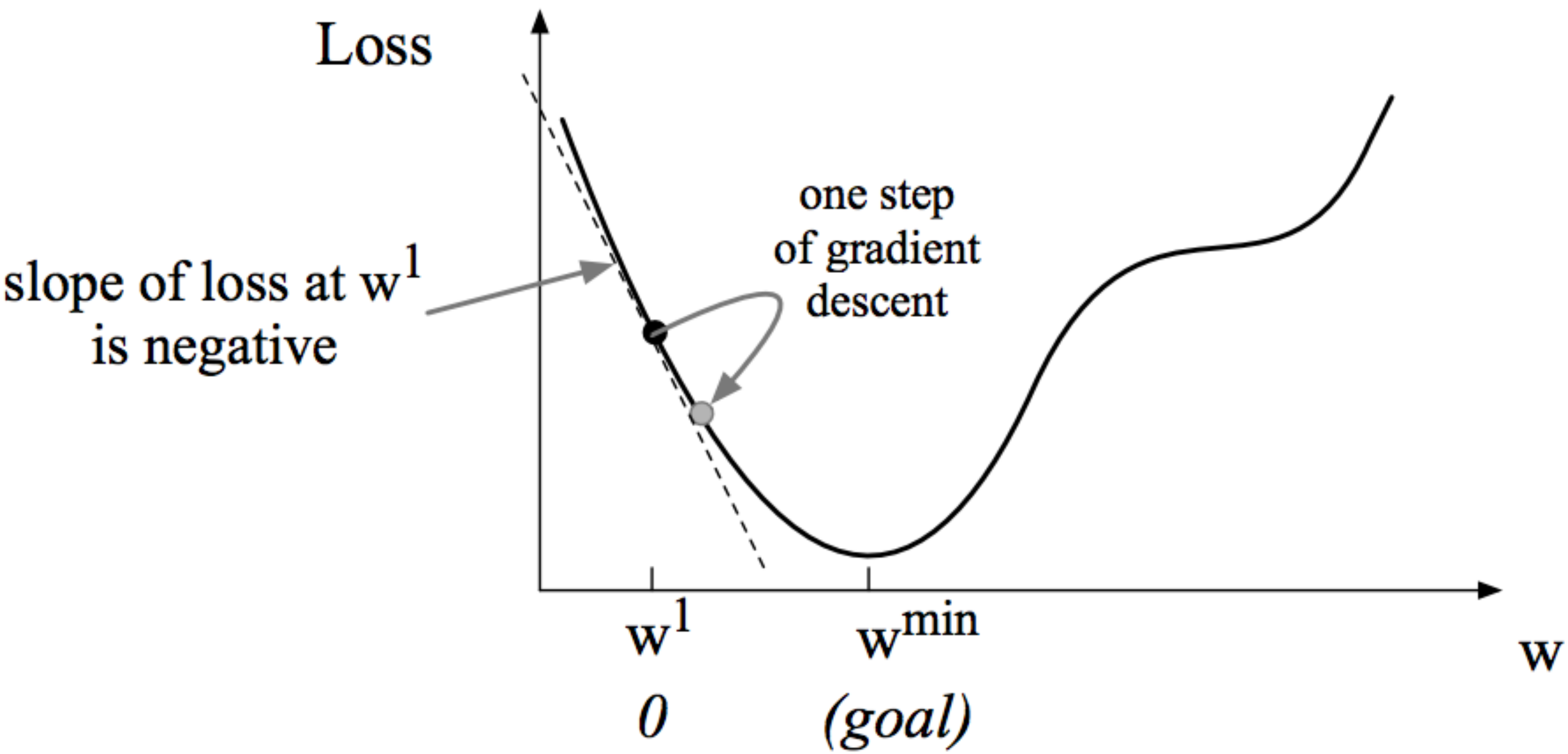
Taken from [here]

Single Weight



Single Weight

Why do we want to find a minimum?



Partial Derivative

- Of course, in real applications we have many features/weights not just one.
- So we need the vector of the partial derivatives of the loss with respect to the weights.

Cross-Entropy Loss Partial Derivative

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Computed answer right answer

Take the difference

Multiply by the feature value of the input corresponding to the weight

Stochastic Gradient Descent

Stochastic Gradient Descent

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

    # where: L is the loss function

    #   f is a function parameterized by θ

    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$

    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat T times

    For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

    Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?

    Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?

    $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?

    $\theta \leftarrow \theta - \eta\, g$ # go the other way instead

return θ

Stochastic Gradient Descent

- Imagine a logistic regression classifier:
  - $w = [2.5, -5, -1.2, 0.5, 2.0, 0.7]$, $b = 0.1$
- Example :
  - $x = [3, 2, 1, 3, 0, 4.15]$, $y = 1$
- Prediction:

Stochastic Gradient Descent

- Imagine a logistic regression classifier:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7], b = 0.1
- Example :
  - x = [3, 2, 1, 3, 0, 4.15], y = 1
- Prediction:
  - sigmoid(2.5*3-2*5-1*1.2+0.5*3+2.0*0+0.7*4.15+0.1) = sigmoid(0.805) = 0.69

Stochastic Gradient Descent

- Imagine a logistic regression classifier:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7], b = 0.1
- Example :
  - x = [3, 2, 1, 3, 0, 4.15], y = 1
- Prediction:
  - sigmoid(2.5*3-2*5-1*1.2+0.5*3+2.0*0+0.7*4.15+0.1) = sigmoid(0.805) = 0.69
- Loss:

$$L_{CE}(w,b) = -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))]$$

Stochastic Gradient Descent

- Imagine a logistic regression classifier:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7], b = 0.1
- Example :
  - x = [3, 2, 1, 3, 0, 4.15], y = 1
- Gradient: $$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$
  - [-0.31*3, -0.31*2, -0.31*1, -0.31*3, -0.31*0, -0.31*4.15] = [-0.93, -0.62, -0.31, -0.93, 0, -1.28]

Stochastic Gradient Descent

$$w_{t+1} = w_t - \mu\Delta$$

- Current weights:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7]
- Gradient:
  - [-0.93, -0.62, -0.31, -0.93, 0, -1.28]
- Assume a learning rate of 1 for simplicity!
- New weights:
  - [3.42687096, -4.38208602, -0.89104301, 1.42687096, 2. , 1.9821715]

Redo the Example

- New weights:
  - w = [3.43, -4.38, -0.89, 1.43, 2., 1.98],
  - b = 0.1
- Example (the same as before!):
  - x = [3, 2, 1, 3, 0, 4.15], y = 1

Redo the Example

- New weights:
  - w = [3.43, -4.38, -0.89, 1.43, 2., 1.98],
  - b = 0.1
- Example (the same as before!):
  - x = [3, 2, 1, 3, 0, 4.15], y = 1
- Score
  - $\boldsymbol{\sigma}$([3.43, -4.38, -0.89, 1.43, 2., 1.98] · [3, 2, 1, 3, 0, 4.15] + 0.1)
  - = $\boldsymbol{\sigma}$(13.247)
  - = 1.

About the Bias

- There's that pesky bias term. That's also a parameter and we forgot to update it as well.

Example

- Imagine a logistic regression classifier:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7], b = 0.1
- Example :
  - x = [3, 2, 1, 3, 0, 4.15]

Example

- Imagine a logistic regression classifier:
  - w = [2.5, -5, -1.2, 0.5, 2.0, 0.7, **0.1**]
- Example :
  - x = [3, 2, 1, 3, 0, 4.15, **1**]

In-Class Exercise 2

- Imagine a logistic regression classifier:
  - w = [0, 0, 0], b = 0
- Example 1:
  - x = [0, 1, 0], y = 0
- Example 2:
  - x = [1, 0, 1], y = 1
- Perform 2 steps of stochastic gradient descent using cross-entropy loss and learning rate 0.1! What parameters do you obtain?

Optimization

- In practice, that can be slow to converge because the algorithm can either be taking steps…
  - …that are too small and hence take us too long to get where we're going
  - …or too large which leads us to overshoot the target and wander around too much

Optimization

- In practice, that can be slow to converge because the algorithm can either be taking steps…
    - …that are too small and hence take us too long to get where we're going
    - …or too large which leads us to overshoot the target and wander around too much
- Fortunately, you don't have to worry about this. Lots of packages available where you need to specify the loss function and parameters and you're done.

Softmax Loss

- What's the loss for the softmax?
  - We'll use the same cross-entropy idea
  - The probability assigned by the model to the correct class
  - Then use the negative log probability of that

Stochastic Gradient Descent

- Batch training
    - Process each example in the training set and accumulate the gradients
    - Do a single update
    - Repeat

Stochastic Gradient Descent

- Batch training
  - Process each example in the training set and accumulate the gradients
  - Do a single update
  - Repeat
- Minibatch training
  - Select N examples and proceed as with batch training
  - Update after each mini-batch
  - N is chosen to maximize parallelism

Building Your Own Sentiment Classifier

Let’s See Some Code!

```python
if __name__ == "__main__":
    train, dev = load_data()
    feature_dict = get_features(train)
    train = make_feature_vectors(train, feature_dict)
    dev = make_feature_vectors(dev, feature_dict)

    # TODO: substitute -1 with the correct value!
    model = MyClassifier(len(feature_dict), -1)

    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=.1)

    eval(model, train)
    eval(model, dev)
    print()

    for i in range(3):
        model.train()
        for (x, y) in train:
            model.zero_grad()
            raw_scores = model(x)
            loss = loss_function(raw_scores.unsqueeze(0), y)
            loss.backward()
            optimizer.step()

        eval(model, train)
        eval(model, dev)
        print()
```

```python
class MyClassifier(nn.Module):

    def __init__(self, num_features, num_labels):
        super(MyClassifier, self).__init__()

        # TODO: substitute -1 with the correct value!
        self.linear = nn.Linear(num_features, -1)

    def forward(self, input):
        return self.linear(input)
```

In-Class Exercise: BYOSC

- Download the sentiment dataset and the prepared code from Canvas (originally from http://help.sentiment140.com/for-students)
  - What do the labels mean?
- Fill in the TODOs in the code
  - What is the accuracy of your classifier?
- Improve your classifier!
  - For example: can you find better features?
  - What is the highest accuracy you can obtain?

Wrapping up

- Discussed today:
    - Cross-entropy loss
    - Stochastic gradient descent

- On Monday: Training neural networks, part 2