DSC 190

Machine Learning: Representations

Lecture 14 | Part 1

Basic Backpropagation

Computing the Gradient

- To train a neural network, we can use gradient descent.
- Involves computing the gradient of the cost function.
- **Backpropagation** is one method for efficiently computing the gradient.

The Gradient

$$
\begin{aligned}
\nabla_{\vec{w}} C(\vec{w}) &= \nabla_{\vec{w}} \frac{1}{n} \sum_{i=1}^{n} \left(f(\vec{x}^{(i)}; \vec{w}) - y_i \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \nabla_{\vec{w}} \left(f(\vec{x}^{(i)}; \vec{w}) - y_i \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} 2 \left(f(\vec{x}^{(i)}; \vec{w}) - y_i \right) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})
\end{aligned}
$$

Interpreting the Gradient

$$\nabla_{\vec{w}} C(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} 2 \left(f(\vec{x}^{(i)}; \vec{w}) - y_i \right) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$$

- The gradient has one term for each training example, $(\vec{x}^{(i)}, y_i)$
- If prediction for $\vec{x}^{(i)}$ is good, contribution to gradient is small.
- $\nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$ captures how sensitive $f(\vec{x}^{(i)})$ is to value of each parameter.

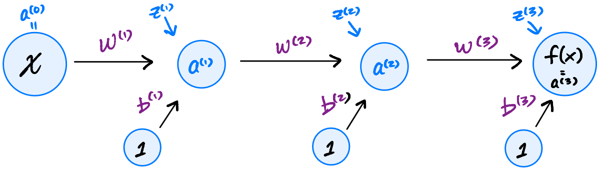
The Chain Rule

- Recall the **chain rule** from calculus.
- Let $f, g : \mathbb{R} \to \mathbb{R}$
- Then:

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g(x)$$

- Alternative notation: $\frac{d}{dx} f(g(x)) = \frac{df}{dg} \frac{dg}{dx}(x)$

The Chain Rule for NNs

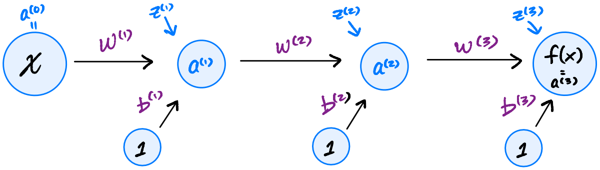


$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma\left(z^{(l)}\right)$$

$$\vec{w} = \left(w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)}\right)$$

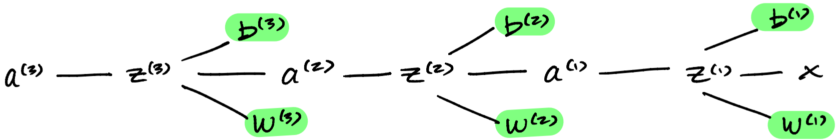
Computation Graphs



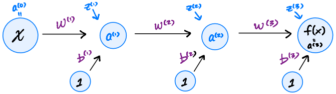
$$z^{(\ell)} = w^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \sigma(z^{(\ell)})$$

$$\vec{w} = (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$



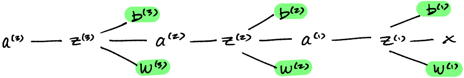
Example



$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

$$\vec{w} = (W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$

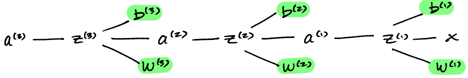


General Formulas

- Derivatives are defined recursively
- Easy to compute derivatives for early layers if we have derivatives for later layers.
- This is **backpropagation**.

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

$$\frac{\partial f}{\partial a^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell+1)}} \cdot \frac{\partial a^{(\ell+1)}}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}}$$



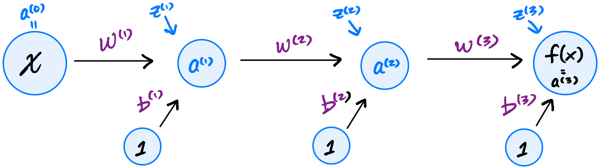
Warning

- The derivatives depend on the network **architecture**
  - Number of hidden nodes / layers

- Backprop is done automatically by your NN library

Backpropagation

Compute the derivatives for the last layers first; use them to compute derivatives for earlier layers.



$$z^{(\ell)} = w^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \sigma(z^{(\ell)})$$

$$\vec{w} = (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$

DSC 190

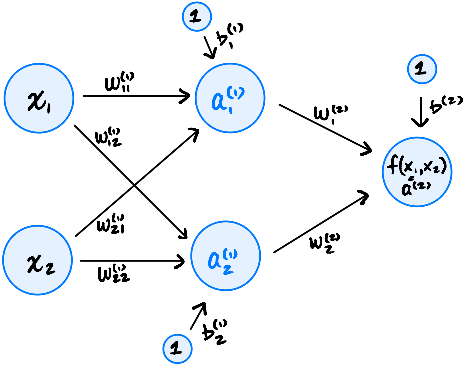
Machine Learning: Representations

Lecture 14 | Part 2

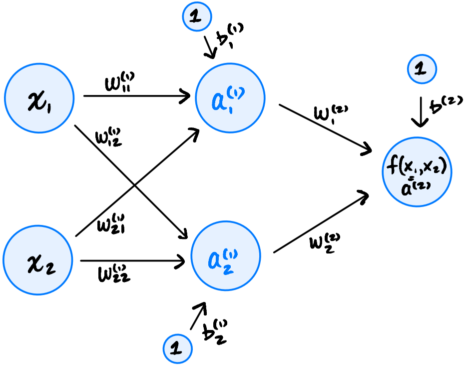
A More Complex Example

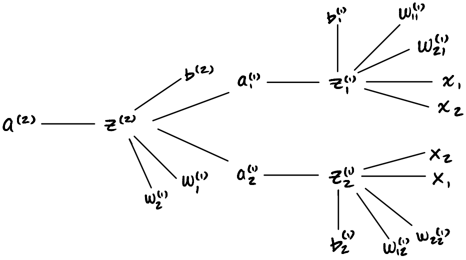
Complexity

- The strategy doesn't change much when each layer has more nodes.



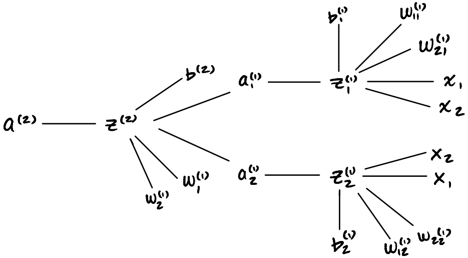
Computational Graph





Example

General Formulas



$$\frac{\partial f}{\partial w_{ij}^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \cdot \frac{\partial z^{(\ell)}}{\partial w_{ij}^{(\ell)}}$$

$$\frac{\partial f}{\partial a^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell+1)}} \cdot \frac{\partial a^{(\ell+1)}}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}}$$

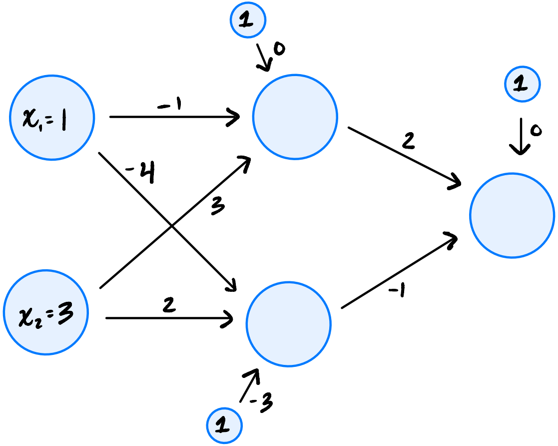
DSC 190

Machine Learning: Representations

Lecture 14 | Part 3

Intuition Behind Backprop

Intuition



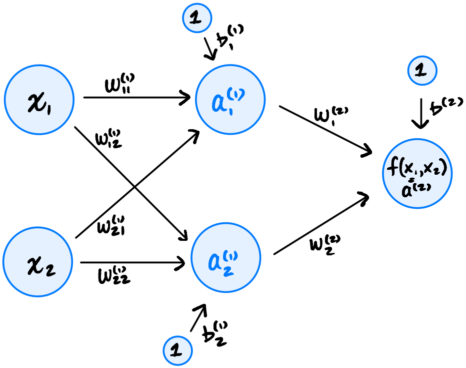
DSC 190

Machine Learning: Representations

Lecture 14 | Part 4

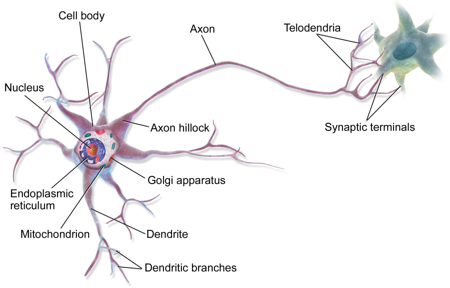
Hidden Units

Hidden Units



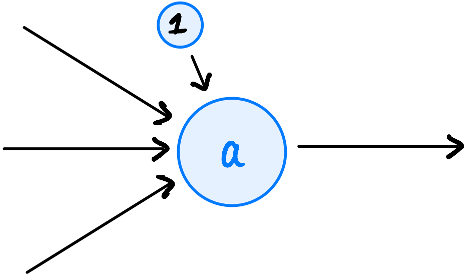
Neuron

- Neuron accepts signals along **synapses**.
- Synapses have weights.
- If weighted sum is “large enough”, the neuron fires, or **activates**.



Neuron

- Neuron accepts weighted inputs.
- If weighted sum is “large enough”, the neuron fires, or **activates**.



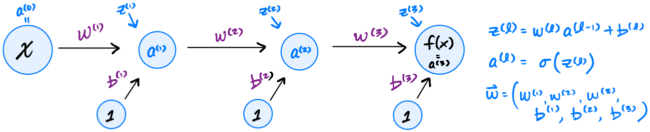
Activation Functions

- A function g determining whether – and how strong – a neuron fires.
- We have seen two: ReLU and linear.
- Many different choices.
- Guided by intuition and only a little theory.

Backpropagation

- The choice of activation function affects performance of backpropagation.

- Example:



$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Vanishing Gradients

- A major challenge in training deep neural networks with backpropagation is that of **vanishing gradients**.

- The gradient for layers far from the output becomes very small.

- Weights can't be changed.

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

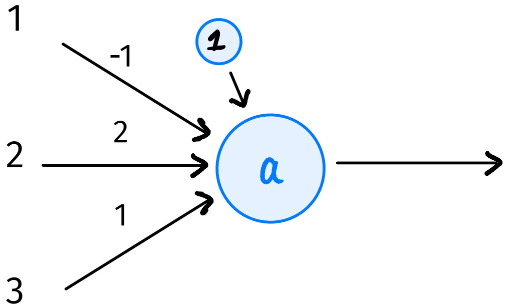
Main Idea

Some activation functions promote “healthier” gradients.

Linear Activations

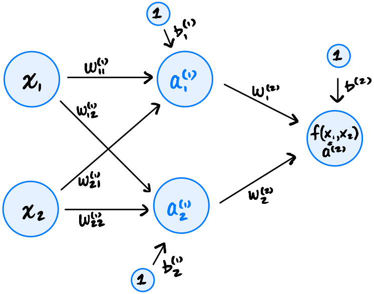
- A **linear** unit's activation function is:

$$g(z) = z$$

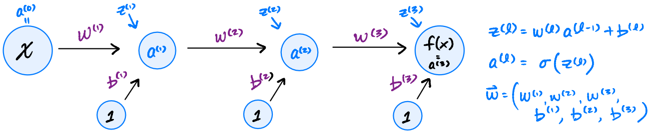


Problem

- Linear activations result in a linear prediction function.



Backprop. with Linear Activations



$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Summary: Linear Activations

- **Good:** healthy gradients, fast to compute
- **Bad:** still results in linear prediction function when layers are combined

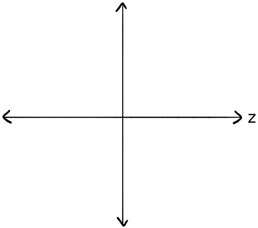
Sigmoidal Activations

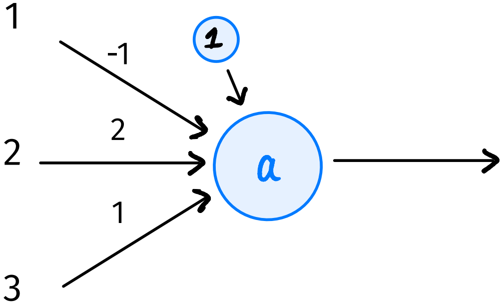
- A basic nonlinearity.
- Neuron is either “on” (1), “off” (0), or somewhere in between.
- Very popular before introduction of the ReLU.

Sigmoidal Activations

- A **sigmoidal** unit's activation function is:

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

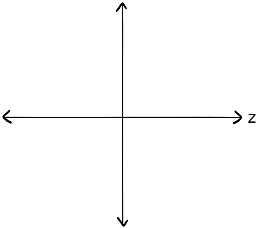


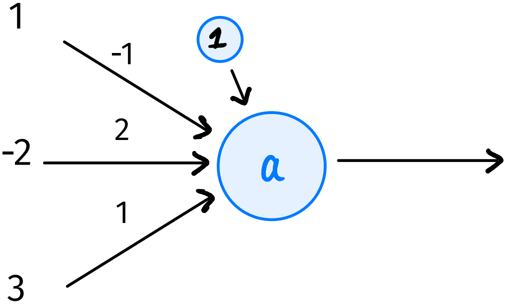


Sigmoidal Activations

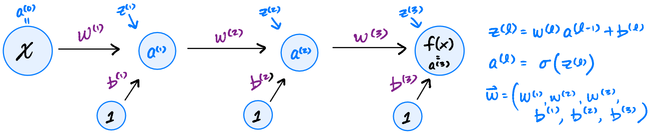
- A **sigmoidal** unit's activation function is:

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





Backprop. with Sigmoids



$$g'(z) = g(z)(1 - g(z)) \qquad \frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(l)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Problem: Saturation

- Large/small inputs lead $g(z)$ to be very close to 1 or -1.
- Here, the derivative $\sigma'(z) \approx 0$.
- Vanishing gradients!
- Makes learning deep networks with gradient-based algorithms very difficult.

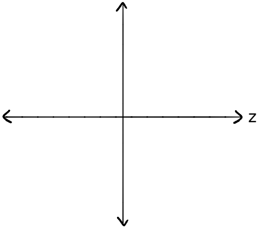
ReLU

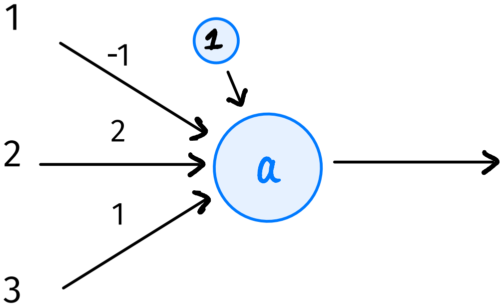
- Linear activations have strong gradients, but combined are still linear.
- Sigmoidal activations are non-linear, but when saturated lead to weak gradients.
- Can we have the best of both?

ReLU

- A **rectified linear** unit's (ReLU) activation function is:

$$g(z) = \max\{0, z\}$$

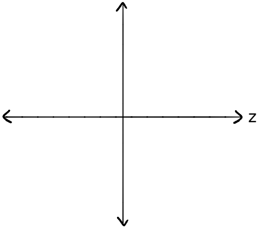


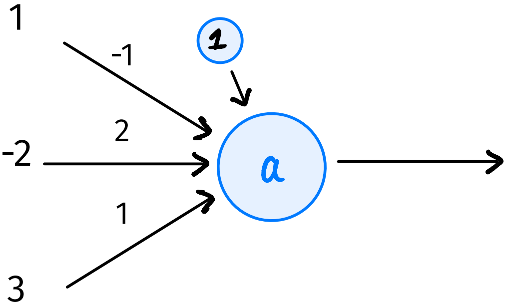


ReLU

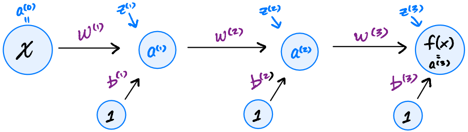
- A **rectified linear** unit's (ReLU) activation function is:

$$g(z) = \max\{0, z\}$$





Backprop. with ReLU



$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$

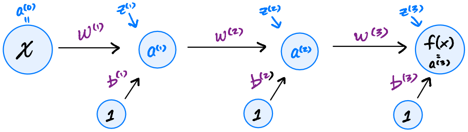
$a^{(\ell)} = \sigma(z^{(\ell)})$

$\vec{w} = (W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Backprop. with ReLU

- **Problem**: If inputs < 0, ReLU "deactivates" and gradients are not passed back.



$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \sigma(z^{(\ell)})$$

$$\vec{w} = (W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$

Fixing Deactivated ReLUs

- One fix: initialize all biases to be small, positive numbers.

- Ensures that most units are active to begin with.

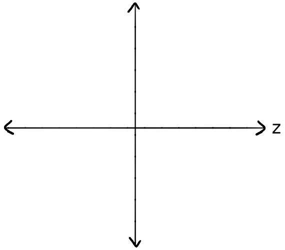
- Another fix: modify the ReLU.

Leaky ReLU

- A **leaky ReLU** activation function is:

$$g(z) = \max\{\alpha z, z\} \qquad 0 \leq \alpha < 1$$

- Usually, $\alpha \approx 0.01$. Nonzero derivative.



Summary: ReLU

- The popular, “default” choice of activation function.
- **Good:** Strong gradient when active, fast to compute.
- **Bad:** No gradient when inactive.

DSC 190

Machine Learning: Representations

Lecture 14 | Part 5

Output Units

Output Units

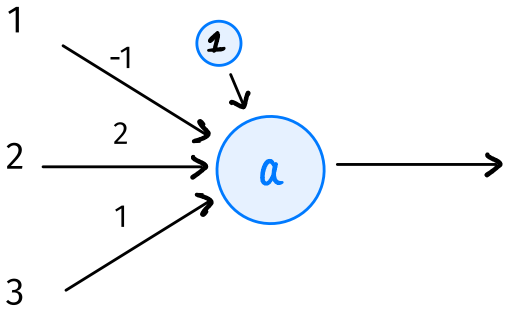
- As with units in hidden layers, we can customize output units.
  - What activation function?
  - How many units?

- Good choice depends on task:
  - Regression, binary classification, multiclass, etc.

- Which loss?

Setting 1: Regression

- Output can be any real number.
- Single output neuron.
- It makes sense to use a **linear activation**.

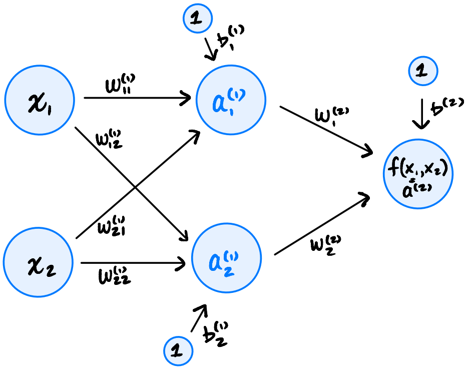


Setting 1: Regression

- Prediction should not be too high/low.
- It makes sense to use the **mean squared error**.

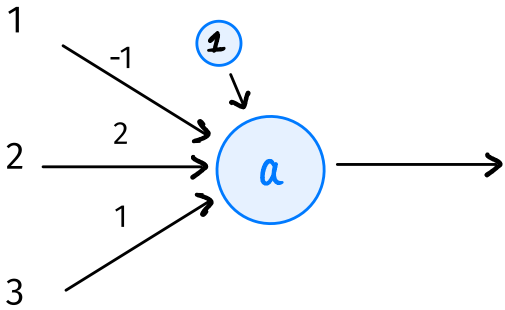
Setting 1: Regression

- Suppose we use linear activation for output neuron + mean squared error.
- This is very similar to least squares regression...
- But! Features in earlier layers are **learned**, non-linear.



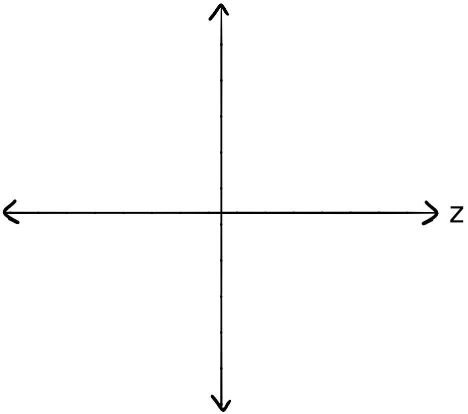
Setting 2: Binary Classification

- Output can be in [0, 1].
- Single output neuron.
- We *could* use a **linear activation**, threshold.
- But there is a better way.



Sigmoids for Classification

- Natural choice for activation in output layer for binary classification: the **sigmoid**.



Binary Classification Loss

- We *could use* square loss for binary classification. There are several reasons not to:

- 1) Square loss penalizes predictions which are “too correct”.

- 2) It doesn’t work well with the sigmoid due to saturation.

The Cross-Entropy

- Instead, we often train deep classifiers using the **cross-entropy** as loss.
- Let $y^{(i)} \in \{0, 1\}$ be true label of ith example.
- The average cross-entropy loss:

$$-\frac{1}{n}\sum_{i=1}^{n} \begin{cases} \log f(\vec{x}^{(i)}), & \text{if } y^{(i)} = 1 \\ \log\left[1 - f(\vec{x}^{(i)})\right], & \text{if } y^{(i)} = 0 \end{cases}$$

The Cross-Entropy and the Sigmoid

- Cross-entropy “undoes” the exponential in the sigmoid, resulting in less saturation.

Summary: Binary Classification

- Use sigmoidal activation the output layer + cross-entropy loss.
- This will promote a strong gradient.
- Use whatever activation for the hidden layers (e.g., ReLU).