Deep Learning Basics

Frank E. Curtis, Lehigh University

presented at

DIMACS/TRIPODS/MOPTA Summer School, Lehigh University

10–12 August 2018

Outline

Outline

Neural Networks: Basics

What is a neural network?

Neural Networks: Basics

What is a neural network?

- A *brain*! (But what does this mean really?)

Neural Networks: Basics

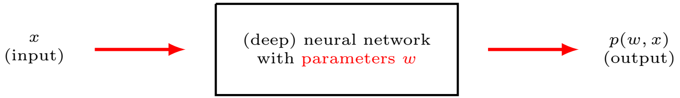
What is a neural network?

- A *brain*! (But what does this mean really?)
- A *computational graph*.

Neural Networks: Basics

What is a neural network?

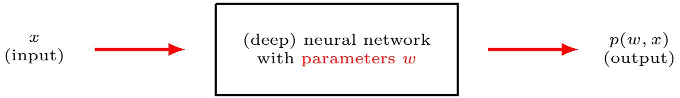
- A *brain*! (But what does this mean really?)
- A *computational graph*.
- A function



Neural Networks: Basics

What is a neural network?

- A *brain*! (But what does this mean really?)
- A *computational graph*.
- A function

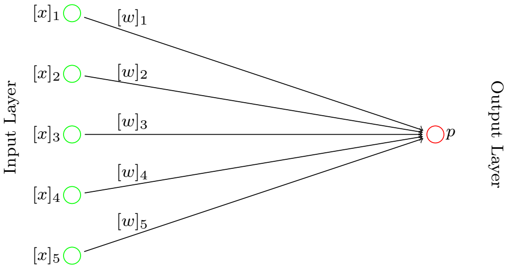


- We can *train* the neural network, i.e., make it *learn*, so that, given any input x, the output $p(w, x)$ is what we want to see.

A Simple Network

Let's use...

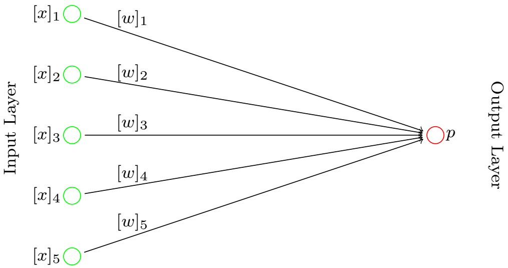
- *nodes* (*neurons*) to represent input/output values.
- *edges* to represent multiplication by a weight.



In this network, the output value is given by

$$p = [x]_1 \cdot [w]_1 + \cdots + [x]_d \cdot [w]_d = \sum_{j=1}^{d} [x]_j \cdot [w]_j = x^T w.$$

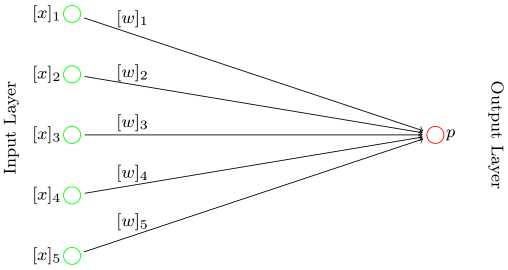
A Simple Network and Loss Function



Suppose we want the weights so that $x_i^T w \approx y_i$ for all $i \in \{1, \ldots, n\}$. Then:

$$\min_{w \in \mathbb{R}^d} f(w), \quad \text{where} \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i^T w, y_i) \quad \text{for some loss } \ell.$$

A Simple Network and Loss Function



Suppose we want the weights so that $x_i^T w \approx y_i$ for all $i \in \{1, \ldots, n\}$. Then:

$$\min_{w \in \mathbb{R}^d} f(w), \quad \text{where} \quad f(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i^T w, y_i) \quad \text{for some loss } \ell.$$

For example, this is one way to describe linear regression using a network.

Image Classification: Digits (MNIST)

Suppose our inputs are *images* of digits we want to classify.

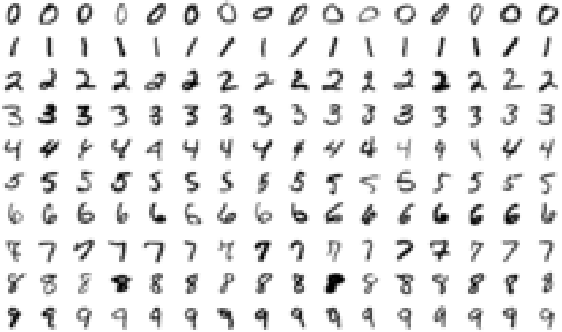
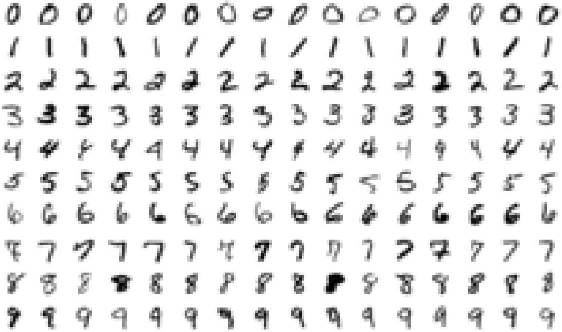


Image Classification: Digits (MNIST)

Suppose our inputs are *images* of digits we want to classify.



Now, instead of a single vector w, we want 10 (one for each digit).

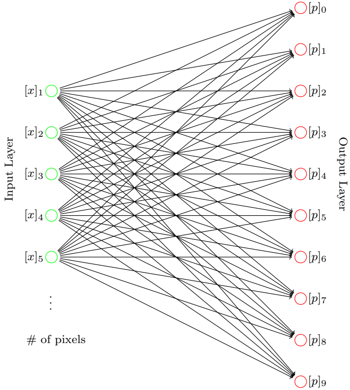
- Define the *vector* w_j for $j \in \{0, \ldots, 9\}$.

Network with Multiple Outputs

- Let the variables be

$$W = \begin{bmatrix} w_0^T \\ \vdots \\ w_9^T \end{bmatrix}.$$

- Then, $p = Wx$.



Network with Multiple Outputs

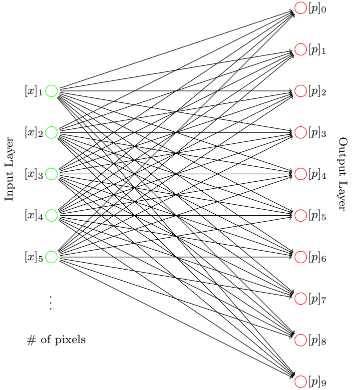
- Let the variables be

$$W = \begin{bmatrix} w_0^T \\ \vdots \\ w_9^T \end{bmatrix}.$$

- Then, $p = Wx$.
- Want to choose W such that, when the correct digit is $j \in \{0, \dots, 9\}$,

$$[p]_j > [p]_i \text{ for } i \neq j.$$

- For example, this can be used for *multi-class regression*
- (... more on this later).



Why the network?

- This is all nice so far, but why did we need the network?

Why the network?

- This is all nice so far, but why did we need the network?
- With typical ℓ, multi-class regression is a nice convex problem.
- We didn't really need the network for it.

Why the network?

- This is all nice so far, but why did we need the network?
- With typical ℓ, multi-class regression is a nice convex problem.
- We didn't really need the network for it.
- The thing is, for performing various tasks
- . . . image classification, speech recognition, machine translation, etc.
- . . . people have found that better results can be achieved by using more complicated *deep* networks involving *layered nonlinear transformations*.
- Inspiration from models of the human brain.

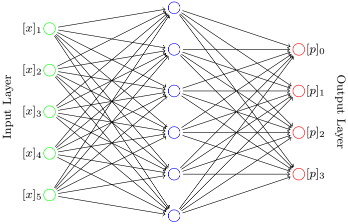
Outline

Hidden Layers

- Rather than go directly from inputs to outputs
- . . . add one or more *hidden* layers.

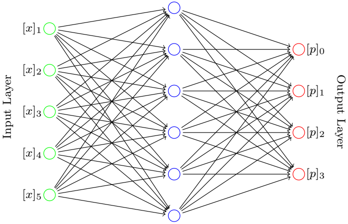
Hidden Layers

- Rather than go directly from inputs to outputs
- ...add one or more *hidden* layers.



Hidden Layers

- Rather than go directly from inputs to outputs
- ...add one or more *hidden* layers.

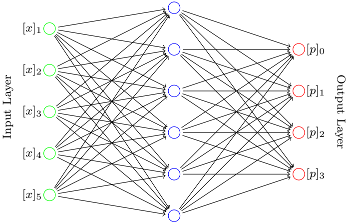


- Now the network performs the computation

$$p = W_2 W_1 x \quad \Longleftarrow \quad f \text{ will be nonconvex!}$$

Hidden Layers

- Rather than go directly from inputs to outputs
- ...add one or more *hidden* layers.



- Now the network performs the computation

$$p = W_2 W_1 x \quad \Longleftarrow \quad f \text{ will be nonconvex!}$$

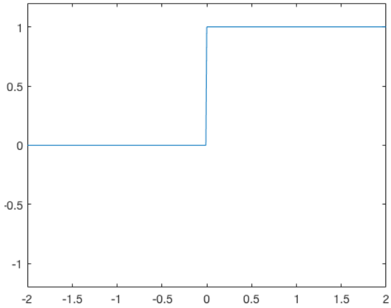
- Can also include *bias* terms such that

$$p = W_2(W_1 x + \omega_1) + \omega_2.$$

Activation Functions

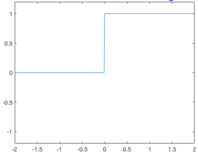
Again, inspired by neuroscience...

- Use *activation functions* to approximate the "firing or not" of a neuron.
- We want to approximate a 0-1 step function.

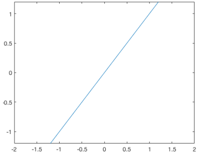


Activation Functions: Examples

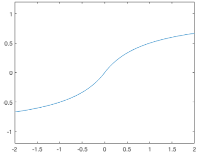
Step



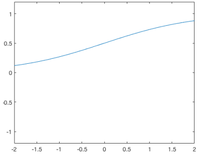
Identity



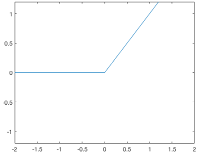
Softsign



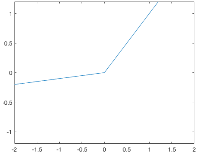
Sigmoid



ReLU



Leaky ReLU



(Here, ReLU = "rectified linear unit".)

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer
- 512 neurons in the first hidden layer
- 256 neurons in the second hidden layer

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer
- 512 neurons in the first hidden layer
- 256 neurons in the second hidden layer
- 10 neurons in the output layer

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer
- 512 neurons in the first hidden layer
- 256 neurons in the second hidden layer
- 10 neurons in the output layer
- ReLU activation functions between layers
- Softmax activation at the final layer (remember this for later!), i.e.,

$$\frac{\exp([p]_i)}{\sum_{j=0}^{9} \exp([p]_j)} \quad \text{for all} \quad i \in \{0, \ldots, 9\}.$$

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer
- 512 neurons in the first hidden layer
- 256 neurons in the second hidden layer
- 10 neurons in the output layer
- ReLU activation functions between layers
- Softmax activation at the final layer (remember this for later!), i.e.,

$$\frac{\exp([p]_i)}{\sum_{j=0}^{9} \exp([p]_j)} \quad \text{for all} \quad i \in \{0, \ldots, 9\}.$$

How many optimization variables are there?

Example: Classifying Digits (MNIST)

For the MNIST dataset, a network may include the following.

- $28 \times 28 = 784$ neurons in the input layer
- 512 neurons in the first hidden layer
- 256 neurons in the second hidden layer
- 10 neurons in the output layer
- ReLU activation functions between layers
- Softmax activation at the final layer (remember this for later!), i.e.,

$$\frac{\exp([p]_i)}{\sum_{j=0}^{9} \exp([p]_j)} \quad \text{for all} \quad i \in \{0, \dots, 9\}.$$

How many optimization variables are there?

$$784 \times 512 + 512 \times 256 + 256 \times 10 = 535040$$

Extensions

The examples so far are *fully connected* networks.

- Various other types exist, designed for different tasks.
  - . . . recurrent
  - . . . residual
  - . . . etc.
- Different types of *layers*
  - . . . convolutional
  - . . . pooling
  - . . . etc.

Outline

Convolutional neural networks

A convolutional neural network (CNN) uses convolutional layers.

- These try to capture *spatial relationships* between inputs.

Convolutional neural networks

A convolutional neural network (CNN) uses convolutional layers.

- These try to capture *spatial relationships* between inputs.
- In the example below, a *filter* is applied—to compute the sum of elementwise products—to look for a diagonal pattern.

$$
\begin{array}{|c|c|c|c|}
\hline
1 & 8 & 0 & 2 \\ \hline
9 & 1 & 7 & 0 \\ \hline
2 & 8 & 0 & 8 \\ \hline
1 & 0 & 9 & 2 \\ \hline
\end{array}
\quad\Bigg|\quad
\begin{array}{|c|c|c|c|}
\hline
1 & 8 & 0 & 2 \\ \hline
9 & 1 & 7 & 0 \\ \hline
2 & 8 & 0 & 8 \\ \hline
1 & 0 & 9 & 2 \\ \hline
\end{array}
\qquad
\begin{array}{|c|c|}
\hline
0 & 1 \\ \hline
1 & 0 \\ \hline
\end{array}
\qquad
\begin{array}{|c|c|c|}
\hline
17 & 1 & 9 \\ \hline
3 & 15 & 0 \\ \hline
9 & 0 & 17 \\ \hline
\end{array}
$$

In a CNN, these matrix operations are “vectorized”.

CNN: Illustration

A random filter simply blurs the data,[1] which might not help to classify the image

[1] https://www.filmcomment.com/article/interview-anjelica-huston

CNN: Illustration

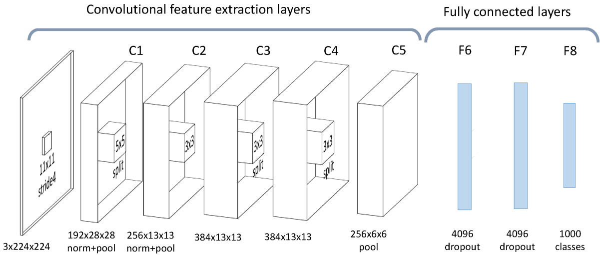
A random filter simply blurs the data,[1] which might not help to classify the image, but certain filters can reveal edges and other features.

In a CNN, the entries of the filter are optimization variables; training the network means we're trying to find the best filters to use to classify images.

[1] https://www.filmcomment.com/article/interview-anjelica-huston

Example Network: ILSVRC Winner

2012 ImageNet Large Scale Visual Recognition Competition[2]



For this network,

- inputs are 224×224 images
- outputs are values for 1000 categories
- over 60,000,000 optimization variables!
- few million inputs, training = weeks on a supercomputer

[2] Photo courtesy of Léon Bottou

Recent Progress

For classifying images in the ILSVRC:

- Prior to 2012, a good error rate was around 25%
- In 2012 with a CNN, this dropped to 16%
- In 2017, out of 38 teams, 29 achieved *under* 5% error

All within the last 6 years!

Outline

General Form

Generally, a neural network model is a function of the form

$$p(w, x) = a_L(W_L(\cdots a_2(W_2(a_1(W_1(x)))) \cdots)),$$

where

- each $W_j(\cdot)$ is linear.
- w represents *all* parameters in the W_j's.
- each $a_i(\cdot)$ is (typically) nonlinear.
- if L is large, then we say the network is *deep*.

General Form

Generally, a neural network model is a function of the form

$$p(w, x) = a_L(W_L(\cdots a_2(W_2(a_1(W_1(x)))) \cdots)),$$

where

- each $W_j(\cdot)$ is linear.
- w represents *all* parameters in the W_j's.
- each $a_i(\cdot)$ is (typically) nonlinear.
- if L is large, then we say the network is *deep*.

Training involves solving an optimization problem to minimize

$$f(x) = \sum_{i=1}^{n} \ell(p(w, x_i), y_i).$$

Computing Derivatives

We shall see that minimizing f requires computing derivatives.

Computing Derivatives

We shall see that minimizing f requires computing derivatives.

But how do we compute derivatives when our function is

$$p(w, x) = a_L(W_L(\cdots a_2(W_2(a_1(W_1(x)))) \cdots))$$

and the variables are the entries defining the W_j functions?

Computing Derivatives

We shall see that minimizing f requires computing derivatives.

But how do we compute derivatives when our function is

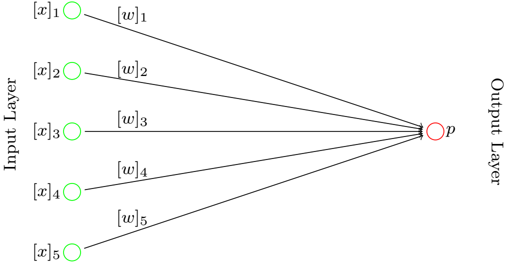
$$p(w, x) = a_L(W_L(\cdots a_2(W_2(a_1(W_1(x)))) \cdots))$$

and the variables are the entries defining the W_j functions?

Amazingly, it can be done!

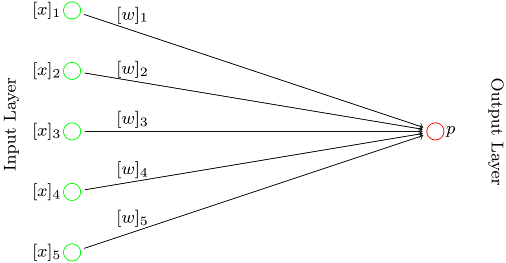
- Procedure known as back-propagation (backprop).
- Special case of automatic differentiation (AD).

Back-propagation: Main Idea



Derivative of output with respect to weight on edge leading to it:

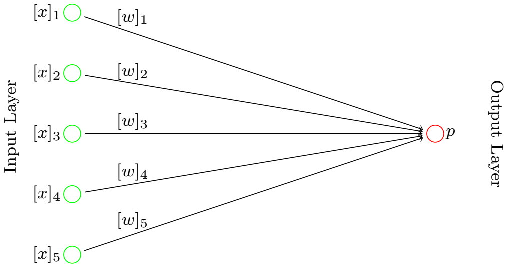
Back-propagation: Main Idea



Derivative of output with respect to weight on edge leading to it:

$$\frac{\partial p}{\partial [w]_j} = [x]_j .$$

Back-propagation: Main Idea

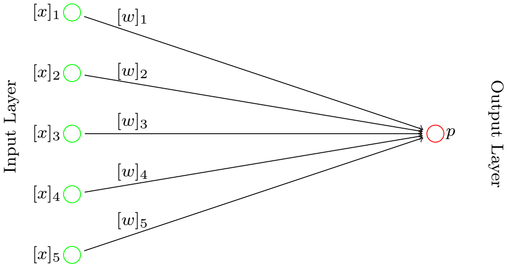


Derivative of output with respect to weight on edge leading to it:

$$\frac{\partial p}{\partial [w]_j} = [x]_j .$$

Derivative of loss function with respect to $[w]_j$:

Back-propagation: Main Idea



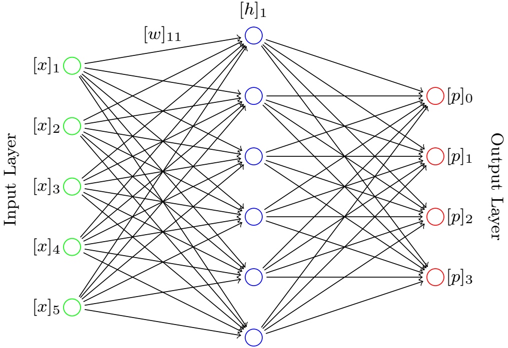
Derivative of output with respect to weight on edge leading to it:

$$\frac{\partial p}{\partial [w]_j} = [x]_j.$$

Derivative of loss function with respect to $[w]_j$: Chain rule!

$$\frac{\partial \ell}{\partial [w]_j} = \frac{\partial \ell}{\partial p} \frac{\partial p}{\partial [w]_j}$$

Back-propagation: Main Idea



Derivative of loss with respect to weight on indicated edge:

$$\frac{\partial \ell}{\partial [w]_{11}} = \sum_{j=0}^{3} \frac{\partial \ell}{\partial [p]_j} \frac{\partial [p]_j}{\partial [w]_{11}} = \sum_{j=0}^{3} \frac{\partial \ell}{\partial [p]_j} \frac{\partial [p]_j}{\partial [h]_1} \frac{\partial [h]_1}{\partial [w]_{11}}$$

If more layers, have to keep *propagating backwards*.

Outline

Multiclass regression

For concreteness, let's discuss a particular loss function for multiclass regression.

Multiclass regression

For concreteness, let's discuss a particular loss function for multiclass regression.

- *Cross entropy* is a measure of the difference between distributions.

Multiclass regression

For concreteness, let's discuss a particular loss function for multiclass regression.

- *Cross entropy* is a measure of the difference between distributions.
- If t is the *true* distribution and q is a *model* distribution, then

$$H(t,q) = \mathbb{E}_t[-\log q] = \underbrace{H(t)}_{\text{entropy of } t} + \underbrace{D_{KL}(t\|q)}_{\text{Kullback-Leibler divergence}}.$$

Multiclass regression

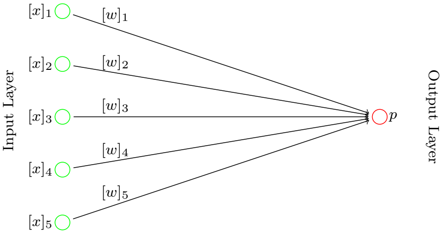
For concreteness, let's discuss a particular loss function for multiclass regression.

- *Cross entropy* is a measure of the difference between distributions.
- If t is the *true* distribution and q is a *model* distribution, then

$$H(t,q) = \mathbb{E}_t[-\log q] = \underbrace{H(t)}_{\text{entropy of } t} + \underbrace{D_{KL}(t\|q)}_{\text{Kullback-Leibler divergence}} .$$

- It can be used to define the loss function in a machine learning context.
- In this context, it is equivalent to logistic regression...

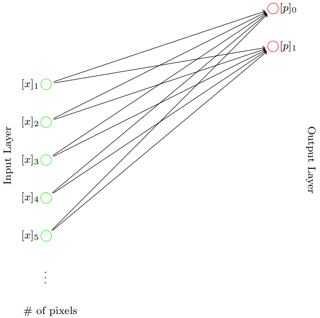
Logistic regression for binary classification



For introducing logistic regression for binary classification,

- one often uses a *single* output value, in this case $p = x^T w$.
- However, this requires an extra step when going to a *multiclass* setting.
- So let's start a slightly different way...

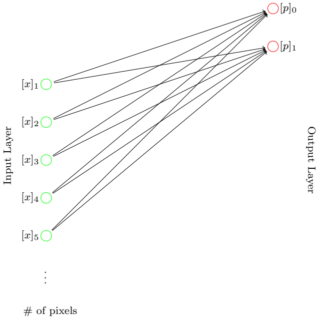
Logistic regression for binary classification



Let's instead suppose that there are two outputs, $[p]_0$ and $[p]_1$.

- We know that the p values depend on a given x...
- ...e.g., we can imagine $[p]_0 = W_0 x$ and $[p]_1 = W_1 x$.

Logistic regression for binary classification



Let's instead suppose that there are two outputs, $[p]_0$ and $[p]_1$.

- We know that the p values depend on a given x...
- ...e.g., we can imagine $[p]_0 = W_0 x$ and $[p]_1 = W_1 x$.
- If, for x, the label is 0, we want $[p]_0 > [p]_1$; otherwise, we want $[p]_0 < [p]_1$.
- We can think of wanting to choose weights such that, for a given x, the p values are the *probabilities* that the correct label is 0 or 1, respectively.

Mapping real numbers to probabilities

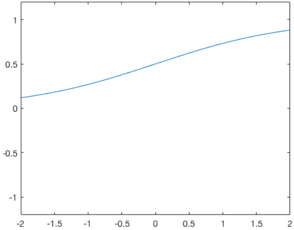
The values $[p]_0$ and $[p]_1$ are real numbers.

- However, we want values that represent *probabilities*; i.e., we want
  - $[p]_0$ to represent the probability that the true label is 0
  - $[p]_1$ to represent the probability that the true label is 1
- How can we map real numbers to probabilities, which are in $[0, 1]$?

Mapping real numbers to probabilities

The values $[p]_0$ and $[p]_1$ are real numbers.

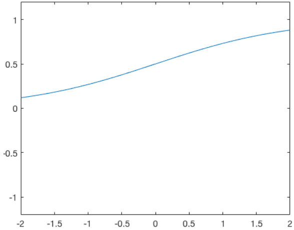
- However, we want values that represent *probabilities*; i.e., we want
  - $[p]_0$ to represent the probability that the true label is 0
  - $[p]_1$ to represent the probability that the true label is 1
- How can we map real numbers to probabilities, which are in $[0, 1]$?
- One possibility is a sigmoid function:



Mapping real numbers to probabilities

The values $[p]_0$ and $[p]_1$ are real numbers.

- However, we want values that represent *probabilities*; i.e., we want
  - $[p]_0$ to represent the probability that the true label is 0
  - $[p]_1$ to represent the probability that the true label is 1
- How can we map real numbers to probabilities, which are in $[0, 1]$?
- One possibility is a sigmoid function:



- After some algebra, we end up with two values we can use:

$$[q]_0 = \frac{\exp([p]_0)}{\exp([p]_0) + \exp([p]_1)} \quad \text{and} \quad [q]_1 = \frac{\exp([p]_1)}{\exp([p]_0) + \exp([p]_1)}.$$

(Recall the softmax activation function from earlier.)

Maximum likelihood

With these values, we can consider choosing w to *maximize likelihood*:

$$\max_w \prod_{i=1}^{n} [q]_{y_i} \quad \Longleftrightarrow \quad \max_w \prod_{i=1}^{n} \frac{\exp([p]_{y_i})}{\exp([p]_0) + \exp([p]_1)}$$

If y_i is the label for datum i, then $[p]_{y_i}$ appears in numerator for ith term.

Maximum likelihood

With these values, we can consider choosing w to *maximize likelihood*:

$$\max_w \prod_{i=1}^n [q]_{y_i} \quad \Longleftrightarrow \quad \max_w \prod_{i=1}^n \frac{\exp([p]_{y_i})}{\exp([p]_0) + \exp([p]_1)}$$

If y_i is the label for datum i, then $[p]_{y_i}$ appears in numerator for ith term.

This objective is nasty! Instead, equivalently *minimize negative log-likelihood*:

$$\begin{aligned} -\log\left(\prod_{i=1}^n [q]_{y_i}\right) &= -\sum_{i=1}^n \log([q]_{y_i}) \\ &= -\sum_{i=1}^n \log\left(\frac{\exp([p]_{y_i})}{\exp([p]_0) + \exp([p]_1)}\right). \end{aligned}$$

Maximum likelihood

With these values, we can consider choosing w to *maximize likelihood*:

$$\max_w \prod_{i=1}^n [q]_{y_i} \quad \Longleftrightarrow \quad \max_w \prod_{i=1}^n \frac{\exp([p]_{y_i})}{\exp([p]_0) + \exp([p]_1)}$$

If y_i is the label for datum i, then $[p]_{y_i}$ appears in numerator for ith term.

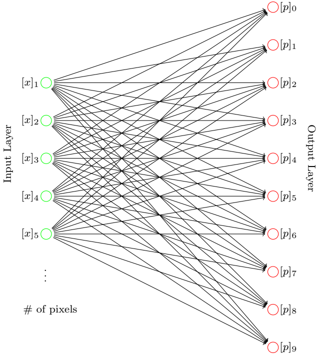
This objective is nasty! Instead, equivalently *minimize negative log-likelihood*:

$$\begin{aligned} -\log\left(\prod_{i=1}^n [q]_{y_i}\right) &= -\sum_{i=1}^n \log([q]_{y_i}) \\ &= -\sum_{i=1}^n \log\left(\frac{\exp([p]_{y_i})}{\exp([p]_0) + \exp([p]_1)}\right). \end{aligned}$$

In terms of comparing probability distributions:

- If $y_i = 0$, then ith term is cross entropy between $(1, 0)$ and $([q]_0, [q]_1)$.
- If $y_i = 1$, then ith term is cross entropy between $(0, 1)$ and $([q]_0, [q]_1)$.

Multiclass setting



More generally, suppose we have labels $\ell \in \{1, \ldots, \Lambda\}$.

- Let $[q]_\ell$ model the probability that the input x has label ℓ.
- Then, so that the probabilities sum to one, let

$$[q]_\ell = \frac{\exp([p]_\ell)}{\sum_j \exp([p]_j)}.$$

Multiclass logistic regression

Over a network, the maximum likelihood problem becomes

$$\begin{aligned}
& \max_w \prod_{i=1}^{n} \left(\frac{\exp([p]_{y_i})}{\sum_j \exp([p]_j)} \right) \\
\implies \quad & \min_w \left(-\sum_{i=1}^{n} \log \left(\frac{\exp([p]_{y_i})}{\sum_j \exp([p]_j)} \right) \right) \\
\implies \quad & \min_w \left(-\sum_{i=1}^{n} \log \left(\frac{\exp([p(w, x_i)]_{y_i})}{\sum_j \exp([p(w, x_i)]_j)} \right) \right).
\end{aligned}$$