Machine Learning Course - CS-433

Neural Nets – Representation Power

Dec 8, 2016

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Motivation

How "powerful" are neural nets? More precisely, what functions $f(\mathbf{x})$ can they represent, or better, what functions can they approximate? We will see that even relatively simple nets (with at most two hidden layers) are capable of approximating any continous function arbitrarily closely on a bounded domain, assuming only that we allow a large number of nodes and arbitrary weights and biases.

We will not take a rigorous approach. Rather, we will follow the lead of Michael Nielson and give a heuristic but rather convincing argument which shows why neural nets are so expressive.

If you are interested in a rigorous approach we recommend that you read either "Approximation by superposition of a sigmoidal function" by Cybenko (1989), or "Universal approximation bounds for superpositions of a sigmoidal function" by Barron (1993). Both of these papers deal with networks with a single layer and sigmoids as approximation functions. Since then, many further approximation results for various network structures, activation functions, and approximation measures have been derived.

Before we get into our heuristic argument let us state the main theorem of the paper by Barron. This gives you a flavor of what kind of results can be rigorously proved.

Lemma. *Let $f : \mathbb{R}^D \to \mathbb{R}$ be a function such that*

$$\int_{\mathbb{R}^D} |\omega||\tilde{f}(\omega)|d\omega \leq C,$$

where

$$\tilde{f}(\omega) = \int_{\mathbb{R}^D} f(\omega)e^{-j\omega^\top \mathbf{x}} f(\mathbf{x})d\mathbf{x}$$

is the Fourier transform of $f(\mathbf{x})$.
Then for all $n \geq 1$, there exists a function f_n of the form

$$f_n(\mathbf{x}) = \sum_{j=1}^{n} c_j \phi(\mathbf{x}^\top \mathbf{w}_j + b_j) + c_0,$$

i.e., a function that is representable by a NN with one hidden layer with n nodes and sigmoids as activation functions so that

$$\int_{|\mathbf{x}|\leq r} (f(\mathbf{x}) - f_n(\mathbf{x}))^2 d\mathbf{x} \leq \frac{(2Cr)^2}{n}.$$

Discussion: First note that the condition on the Fourier transform is a "smoothness condition." E.g., functions so that $\int_{\mathbb{R}^D} |\omega||\tilde{f}(\omega)|d\omega < \infty$ can be shown to be continuously differentiable.
Second note that the lemma only guarantees a good approximation in a bounded domain. The larger the domain, the more nodes we need in order to approximate a function to the same level (see the term r^2, where r is the radius of the ball where we want the approximation to be good, in the upper bound).

Third, this is an approximation "in average", more precisely in L_2-norm. We will mostly discuss approximations in this sense but come back to this point at the end.
Fourth, the approximation f_n with n terms corresponds exactly to our model of a neural net with one hidden layer containing n nodes and sigmoids as activiation functions.
In words the lemmas says that a sufficiently "smooth" function can be approximated by a neural net with one hidden layer and the approximation error goes down like one over the number of nodes in the hidden layer. Note that this is a very fast convergence.

Approximation in Average

We will now give a very simple and intuitive explanation why neural nets with a sigmoid as activation function and at most two hidden layers have already a large expressive power. This explanation will not be rigorous and it will fall far short of the stated lemma by Barron which proves that in fact we only need one hidden layer and not too many hidden nodes. But it is quick and quite convincing. We will search for an approximation "in average", i.e., an approximation so that the integral over the absolute value of the difference is arbitrarily small.

Functions $\mathbb{R} \to \mathbb{R}$

We start with a scalar function $f(x)$ on a bounded domain. Recall that if this function is also continuous then it is Riemann integrable, i.e., it can be approximatex arbi-

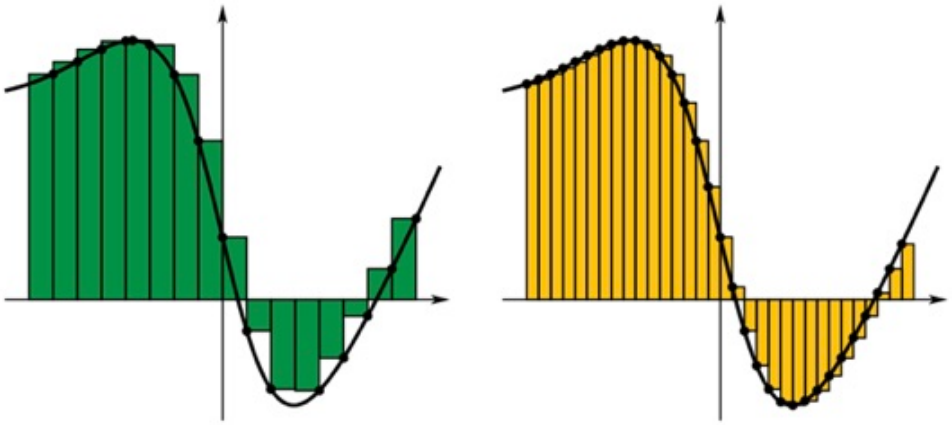

Figure 1: A lower and an upper Rieman sum.

trarily closely by "upper" and "lower" sums of rectangles, see Figure 1. Of course, we might need a lot of such rectangles to approximate the area by lets say ϵ, but for every $\epsilon > 0$ we can find such an approximation.

We will now show that if we do not limit the weights, then with two hidden nodes (of a a neural network with one hidden layer) we can construct a function which is arbitrarily close to a given rectangle. But since, as we have just seen, a finite number of rectangles suffices to approximate a bounded continuous function arbitrarily closely, it follows that with a finite number of hidden nodes of a neural network with one hidden layer we can approximate any such function arbitrarily closely (in the sense that the integral of the absolute value of their difference is arbitrarily small).

The following idea is taken from the tutorial by Michael Nielson, http://neuralnetworksanddeeplearning.com.

Let $\phi(x) = \frac{1}{1+e^{-x}}$ be the sigmoid function. We have encountered it already in the last lecture. But here it is again in Figure 2.

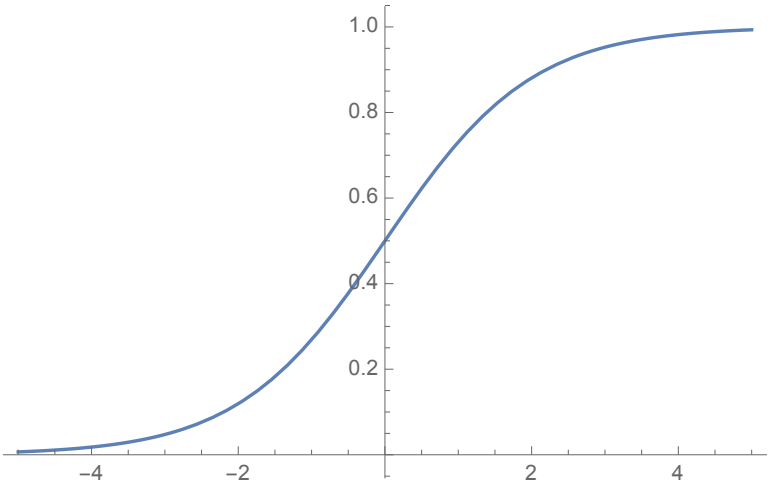

Figure 2: The sigmoid function.

Consider the function $f(x) = \phi(w(x-b))$, where w is the *weight* of a particular edge and $-wb$ is the *bias* term.
Note that if $w \geq 0$ then $f(x)$ is an increasing function that increases from 0 to 1 the more rapidly the larger we choose the weight. In fact, if we set $b = 0$ so that the transition from 0 to 1 happens at $x = 0$, then the derivative of the function at 0 is w. In other words, the *width of the transition* is of order $1/w$.
Further, the transition happens at the spot $x = b$, i.e., at this value of x the function has the value $\frac{1}{2}$. Therefore, if we want to create a rectangle that jumps to 1 at $x = a$ and jumps back to 0 at $x = b$, $a < b$, then we can accomplish this by taking

$$\phi(w(x-a)) - \phi(w(x-b)) \tag{1}$$

and taking w to be very large. This is shown in Figure 3 where the three figures correspond to $w = 10, 20$ and 50, respectively and $a = -3$ and $b = 5$. We see that for large values of w the result is barely distinguishable from a true rectangle.

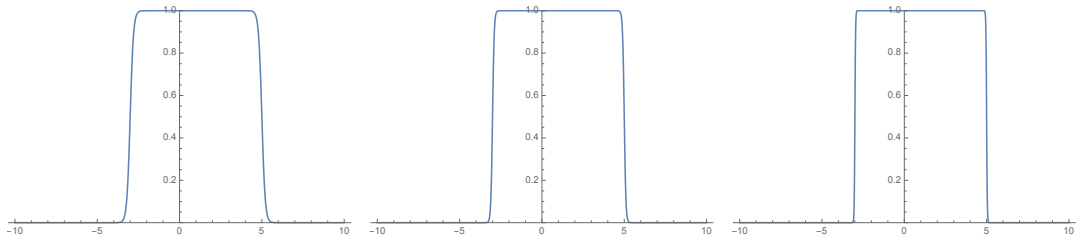

Figure 3: An approximate rectangle of the form $\phi(w(x-a)) - \phi(w(x-b))$ with $w = 10, 20$, and 50, respectively.

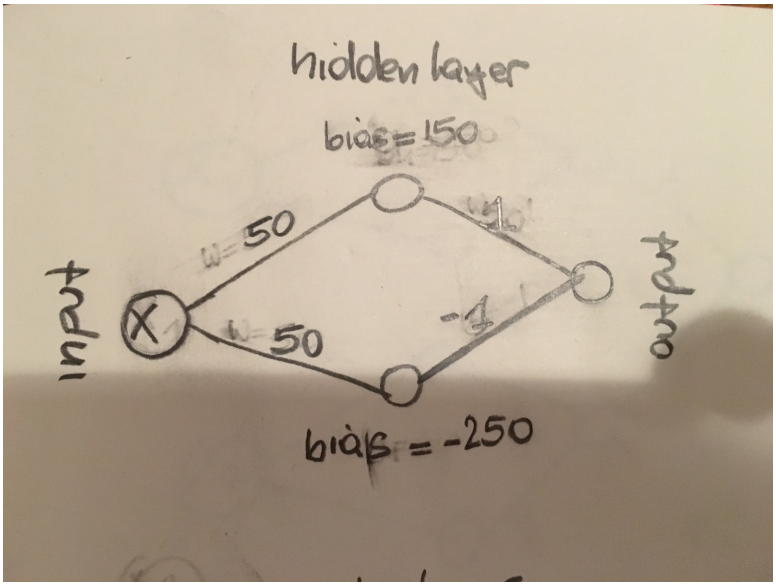

Figure 4: A simple NN implementation of equation (1).

Note that equation (1) has a very simple representation in form of a neural net. This is shown in Figure 4. There is one input node which contains the value x. This value is multiplied by some large weight (in the figure it is 50) to the two hidden nodes. One has a bias of 150 the other one a bias of -250 so that the sums at these two nodes are $50(x+3)$ and $50(x-5)$ respectively. Each node applies the sigmoid function and forwards their values to the output layer. The edge from the hidden node on the top to the output has weight 1 and the one from the hidden node on the bottom to the output has weight -1. The output node adds the two

inputs. The result is $\phi(50(x+3)) - \phi(50(x-5))$, which is approximately a unit-height rectangle from -3 to 5. If we want a rectangle of height h, use the weights h and $-h$ in the second layer instead of the weights 1 and -1.
It is hopefully clear at this point why any continous function on a bounded domain can be approximated via a neural network with one hidden layer. Let us summarize in telegram style: Take the function. Approximate it in the Riemann sense. Approximate each of the rectangles in the Riemann sum by means of two nodes in the hidden layer of a neural net. Compute the sum (with appropriate sign) of all the hidden layers at the output node. If we are using a Riemann sum with K rectangles we get therefore a neural network approximation with one hidden layer containing $2K$ nodes
A few remarks are in order:

1. The same intuition applies to many activiation functions. All we have used is that the activiation function has left limit 0 and right limit 1.

2. Whereas Barron's result gave us a strong bound on the number of required nodes, our intuitive explanation gave us no bound.

3. The above approximation only works if we allow the weights to become arbitrarily large. Of course, very large weights would likely cause problems in practice. It is also not clear why we should need them. There is no fundamental reason why we should first very precisely approximate rectangles which then in turn approximate a continous function.

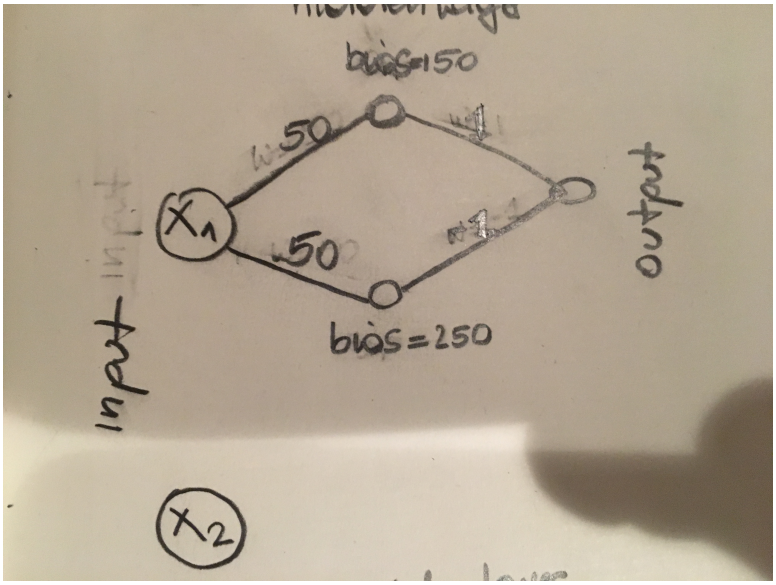

Figure 5: A 2-D rectangle bounded in one dimension and unbounded in the other.

Functions $\mathbb{R}^D \to \mathbb{R}$

So far we have talked about a real function of a single variable. What about functions over $\mathbb{R}^D$? We will see that the same principle applies but we will need one extra layer. The extra idea that is needed to extend the above scheme form one to several dimensions is already present in $\mathbb{R}^2$. We will therefore stick to this case. You should have no trouble figuring out how to extend it to $\mathbb{R}^D$.

As before, it will suffice if we can show how to approximate any two-dimensional rectangle. In fact, all we need are two-dimensional rectangles that are parallel to the two axes.

Let the two inputs/axes be x_1 and x_2, represented by two nodes in the input layer. Assume that we want to represent a rectangle of height 1 that extends from a_1 to b_1, $a_1 < b_1$, on the x_1 axis and from a_2 to b_2, $a_2 < b_2$, on the x_2 axis.

Consider the neural net in Figure 5. It represents a rectangle that goes from a_1 to b_1 in the direction of the x_1 axis but is

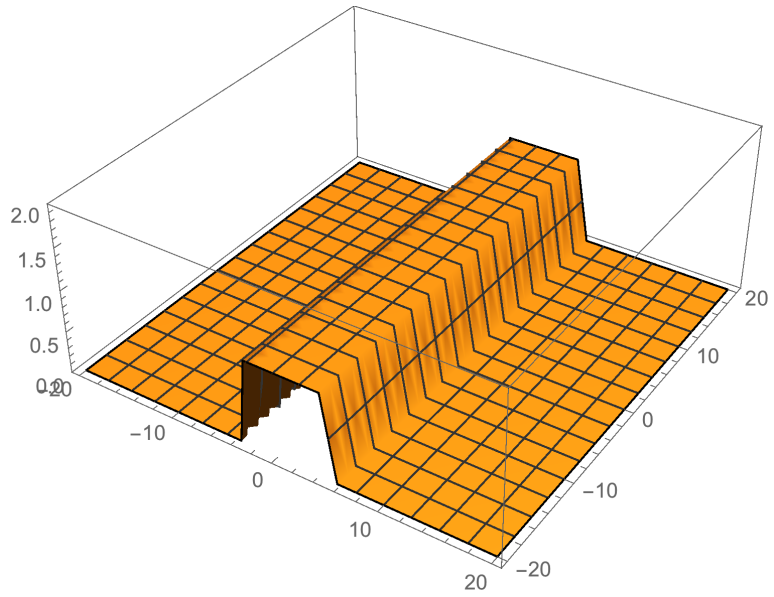

Figure 6: The function $\phi(w(x_1 + a_1)) - \phi(w(x_1 - b_1))$ corresponding to the NN shown in Figure 5. Here, $w = 50$, $a_1 = -3$, and $b_1 = 5$.

unbounded in the x_2 direction. The function it represents is shown Figure 6.

Let us add to this rectangle another one that is unbounded in the x_1 direction but extends from a_2 to b_2 in the direction of the x_2 axis. The corresponding neural net is shown in Figure 7 and the plot of the corresponding function is shown in Figure 8. This is close to what we want. In the region where we want the function to be 1 it is in fact close to 2 since we have there the sum of two functions, each of height 1. A trivial scaling by a factor 1/2 would bring it to the desired value in this region. But unfortunately we still have in addition the two "arms" that extend to infinity along each axis. Those have height 1.

But those additional unwanted "arms" are easily supressed. Just add a sigmoid function with some large weight and bias lets say 3/2 to the node that sums up the two rectangles. A

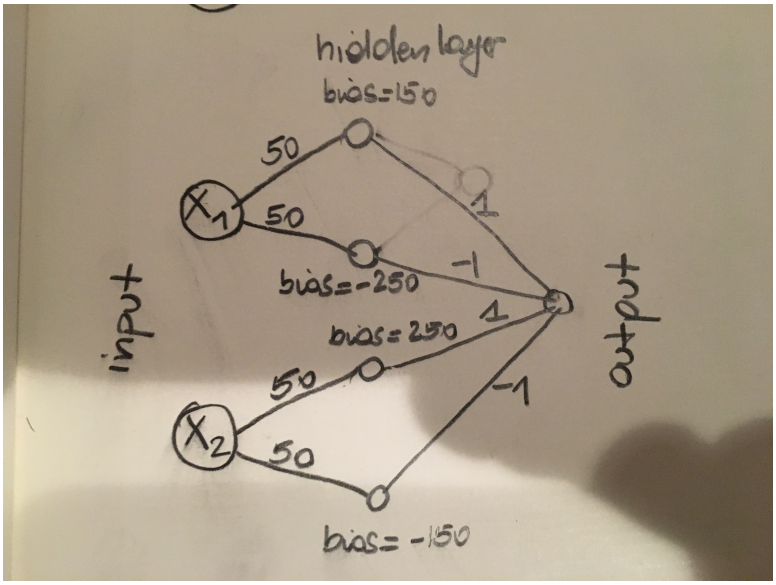

Figure 7: Sum of two rectangles, each bounded in one dimension and unbounded in the other.

plot of the resulting function is shown in Figure 9. So instead of being the output node, as it was before, this node now becomes a hidden node, forming one extra layer of hidden nodes. With this we have created the desired 2-dimensional rectangle. We can approximate any sufficiently smooth 2-dimensional function on a bounded domain by using a suitable number of those and adding them at the output node.

Point-wise Approximations and other Activation Functions

So far we have considered approximations "in average". I.e., we have seen for Barron's result that the L_2 norm can be made arbitrarily small. And for our intuitive derivation we took the Riemann integral as a starting point, i.e., we considered the L_1 norm.
We can also ask if a point-wise approximation with vanishing

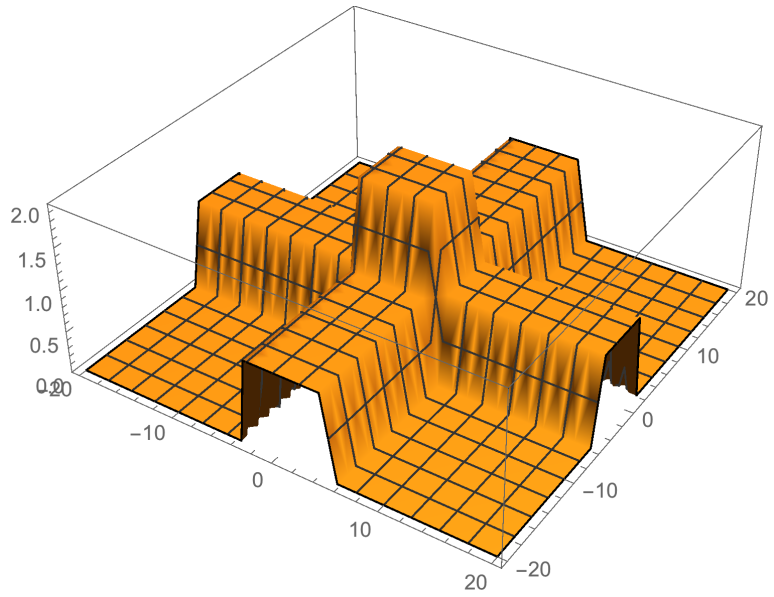

Figure 8: The function $g(x_1, x_2) = \phi(w(x_1+a_1)) - \phi(w(x_1 - b_1)) + \phi(w(x_2 + a_2)) - \phi(w(x_2 - b_2))$ corresponding to the NN shown in Figure 7. Here, $w = 50$, $a_1 = -3$, $b_1 = 5$, $a_2 = -5$, $b_2 = 3$.

error is possible, i.e., we can consider the L_∞ norm.
Further, we have limited our discussion so far to "sigmoid-type" activation function, i.e., activation functions whose left limit is 0 and whose right limit is 1.
So let us now look at point-wise approximations with the *rectified linear function*,

$$(x)_+ = \max\{0, x\}.$$

Many other combinantions (approximation criterion and activation function) are of course possible and have been considered.
Let $f(x)$ be a continous function on a bounded domain. Without loss of generality we can assume that this domain is $[0, 1]$, rescaling and shifting the x-axis if necessary. The classical *Stone-Weierstrass* theorem says that for every $\epsilon > 0$,

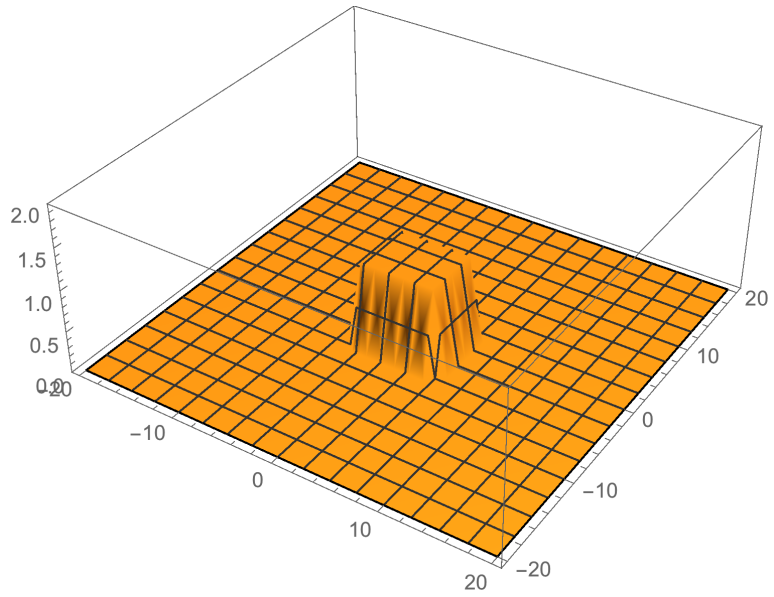

Figure 9: The function $\phi(w(g(x_1, x_2)-3/2))$ where $g(x_1, x_2)$ is the function shown in Figure 8. This is now a good approximation of the desired rectangle.

there exist a polynomial $p(x)$ so that for all $x \in [0, 1]$,

$$|f(x) - p(x)| < \epsilon.$$

But such a function $f(x)$ can also be approximated in L_∞ norm by even simpler functions, namely continuous piecewise-linear functions, see Shektman 1982. Let $q(x)$ be a continous piecewise-linear function. Then it has the form

$$q(x) = \sum_{i=1}^{m} (a_i x + b_i) \mathbb{1}_{\{r_{i-1} \le x < r_i\}},$$

where $0 = r_0 < r_1 < \cdots < r_m = 1$ is a suitable partition of $[0, 1]$. Note that continuity imposes the constraints

$$a_i r_i + b_i = a_{i+1} r_i + b_{i+1}, i = 1, \cdots, m - 1.$$

For our purpose it is more convenient to write it in the alternative form

$$q(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{i=2}^{m} \tilde{a}_i (x - \tilde{b}_i)_+.$$

Here, $\tilde{a}_1 = a_1$ and $\tilde{b}_1 = b_1$ and for $i = 2, \cdots, m$, the remaining parameters can be computed via the relations

$$a_i = \sum_{j=1}^{i} \tilde{a}_i,$$
$$\tilde{b}_i = r_{i-1}$$

Each term in the sum on the right corresponds to one node in a hidden layer with input x, bias $-\tilde{b}_i$, and activation function $(x)_+$. The bias term $\tilde{b}_1$ can be absorbed into the bias term of the output node. This leaves the term $\tilde{a}_1 x$. This term can also be represented by a node in the hidden layer with activiation function $(x)_+$ by chosing $x_0 = 0$, since we only need the this representation to be correct in the range $[0, 1]$. So this shows that we can approximate any continuous function on a bounded domain by a neural net with one hidden layer in the L_∞ norm to arbitrary precision.
We have only considered the one-dimensional case. But it turns out that a similar scheme works also for higher dimensions. We skip the details.