

Mathematics of Image and Data Analysis
Math 5467

Lecture 24: Universal Approximation and
Convolutional Neural Networks

Instructor: Jeff Calder
Email: jcalder@umn.edu

<http://www-users.math.umn.edu/~jwcalder/5467S21>

Announcements

- HW4 due April 30, Project 3 due May 9.
- Please fill out *Student Rating of Teaching (SRT)* online as soon as possible, and before **May 3**.
 - You should have received an email from Office of Measurement Services with a link.
 - You can also find a link on our [Canvas website](#).

Last time

- Classification with neural networks.

Today

- Universal approximation
- Convolutional neural networks

Universal approximation

Part of the success of neural networks is due to the fact that they can approximate any continuous function, given enough parameters. In particular, we can approximate any function by a **2-layer** neural network.

- This cannot fully explain this success, since other methods, like polynomial fitting, can achieve the same universal approximation results.
- It also does not explain why deeper networks can perform better, or why gradient descent finds networks that generalize.

Today we'll consider a 2-layer neural network with N hidden nodes and ReLU activations, which has the form

$$(1) \quad a_+ = \max\{a, 0\} = \text{Relu}(a). \quad f_N(x) = \sum_{i=1}^N a_i (w_i x + b_i)_+ = \sum_{i=1}^N a_i \sigma(w_i x + b_i) + b \text{ not needed.}$$

Handwritten annotations: "2nd layer" above the sum, "1st layer" above the inner sum, and a red bracket under the second sum with the text "+ b not needed".

This is a function $f_N : \mathbb{R} \rightarrow \mathbb{R}$ and the weights $a_i, w_i, b_i \in \mathbb{R}$.

$$\text{If } a > 0, a \text{Relu}(x) = a \max\{x, 0\} = \max\{ax, 0\}.$$

Handwritten note: "Not true if $a < 0$."

Lipschitz functions

We say a function $u : \mathbb{R} \rightarrow \mathbb{R}$ is *Lipschitz continuous* if there exists $C > 0$ such that

$$(2) \quad |u(x) - u(y)| \leq C|x - y|.$$

The smallest such constant is called the *Lipschitz constant* of u and denoted

$$(3) \quad \text{Lip}(u) = \sup_{\substack{x, y \in \mathbb{R} \\ x \neq y}} \frac{|u(x) - u(y)|}{|x - y|}.$$

$$|u(x) - u(y)| \leq \text{Lip}(u) |x - y|.$$

Ex: (i) $f(x) = mx + b$ is Lipschitz, $\text{Lip}(f) = m$.

(ii) If $|f'| \leq M$ then $\text{Lip}(f) \leq M$

(iii) $f(x) = \sqrt{|x|}$ is not Lipschitz



Universal approximation

Theorem 1. Let $\varepsilon > 0$, let $u : \mathbb{R} \rightarrow \mathbb{R}$ be Lipschitz continuous, and let $R > 0$. There exists a 2-layer ReLU neural network $f_N(x)$ of the form (1) with $N = \underline{2(R\text{Lip}(u)\varepsilon^{-1} + 1)}$ hidden nodes such that

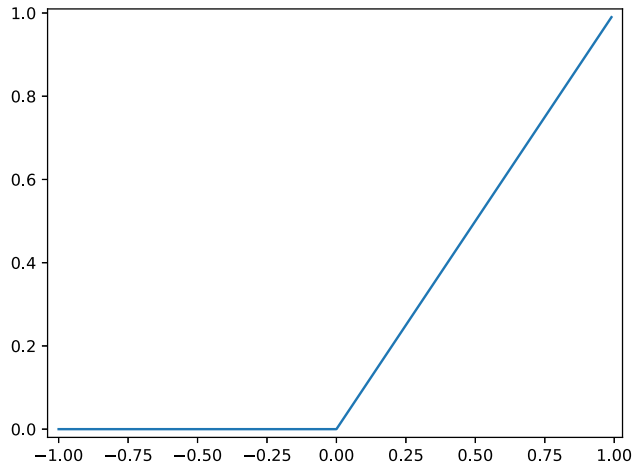
(4) *slightly wrong.*
$$\max_{-R \leq x \leq R} |f_N(x) - u(x)| \leq \varepsilon.$$

Furthermore, if u' is Lipschitz continuous then we need only

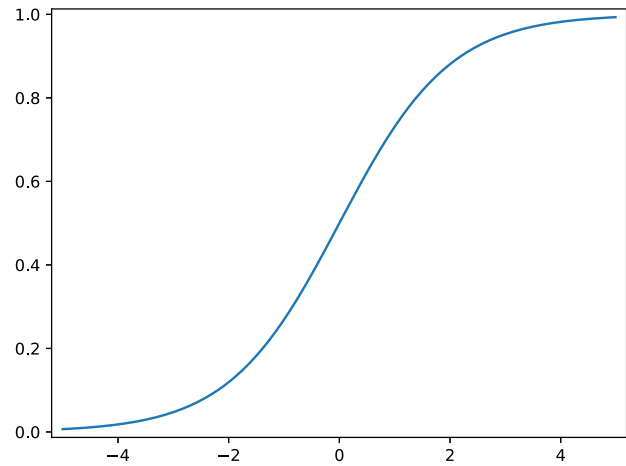
(5)
$$N = 2(R \sqrt{\underbrace{\text{Lip}(u')}_{\text{Lip}(u')} \varepsilon^{-1}} + 1)$$

hidden nodes.

Activations



(a) ReLU



(b) Sigmoid

Figure 1: Plots of the ReLU and Sigmoid activation functions. Both activation functions have the behavior that they give zero, or close to zero, responses when the input is below a certain threshold, and give positive responses above.

Bump functions

The proof is based on the construction of bump functions out of 2-layer networks.
For ReLU the bump function is

3 hidden nodes

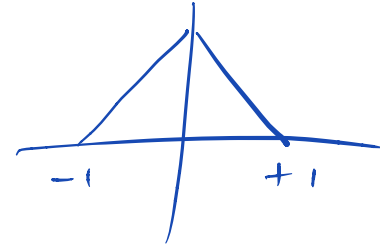
$$g(x) = (x + 1)_+ - 2x_+ + (x - 1)_+.$$

For other activations the construction can be different. For the sigmoid activation a bump function is

$$g(x) = \sigma(x + 1) - \sigma(x - 1).$$

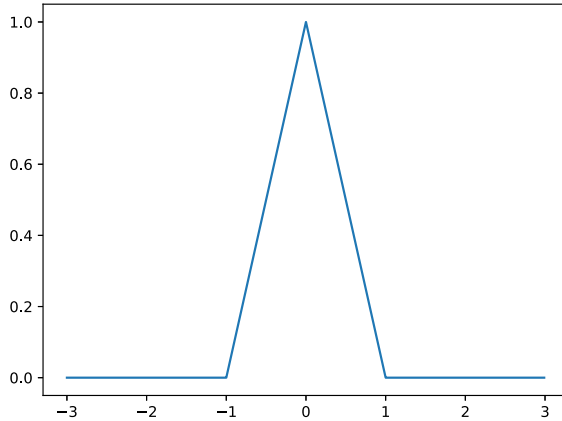
→

$$g(x) = \begin{cases} 0, & x \leq -1 \\ x+1, & -1 \leq x \leq 0 \\ 1-x, & 0 \leq x \leq 1 \\ 0, & x \geq 1 \end{cases}$$

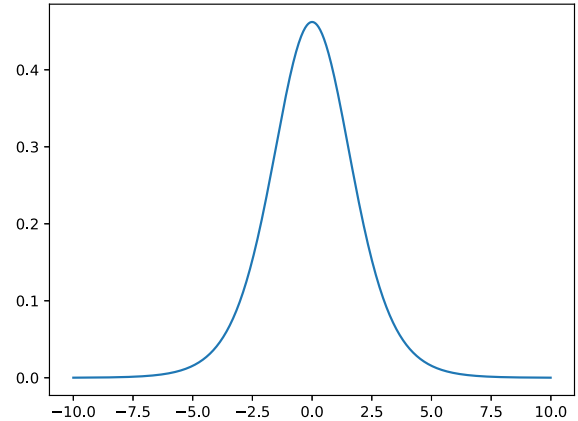


Bump functions

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



(a) ReLU bump



(b) Sigmoid bump

Figure 2: Examples of bump functions constructed from 2-layer neural networks with ReLU and sigmoid activations.

Proof of Universal Approximation Theorem

We can dilate / scale / shift bump functions

$$c g(a(x-b)) = c g(ax-ab)$$

$$= c(ax-ab+1)_+ - 2c(ax-ab)_+ + c(ax-ab-1)_+$$

is a 2-layer neural network with

3 hidden nodes. So a 2-layer network can have the form

$$\sum_{i=1}^m c_i g(a_i(x-b_i)).$$

There are $3m$ nodes in the network.

Let $h > 0$, define $x_i = hi$, $i \in \mathbb{Z}$.

Define $N = 3(2m+1)$ nodes

$$f_N(x) = \sum_{i=-m}^m u(x_i) g(h^{-1}(x - x_i)).$$

where $m = \lceil h^{-1}R \rceil =$ smallest integer larger than $h^{-1}R$.

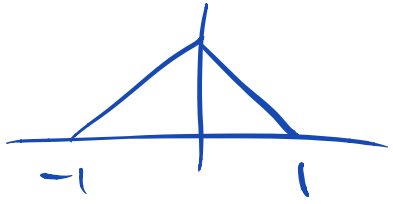
Hence

$$x_m = mh \geq R$$

$$x_{-m} = -mh \leq -R$$

$$m \leq h^{-1}R + 1$$

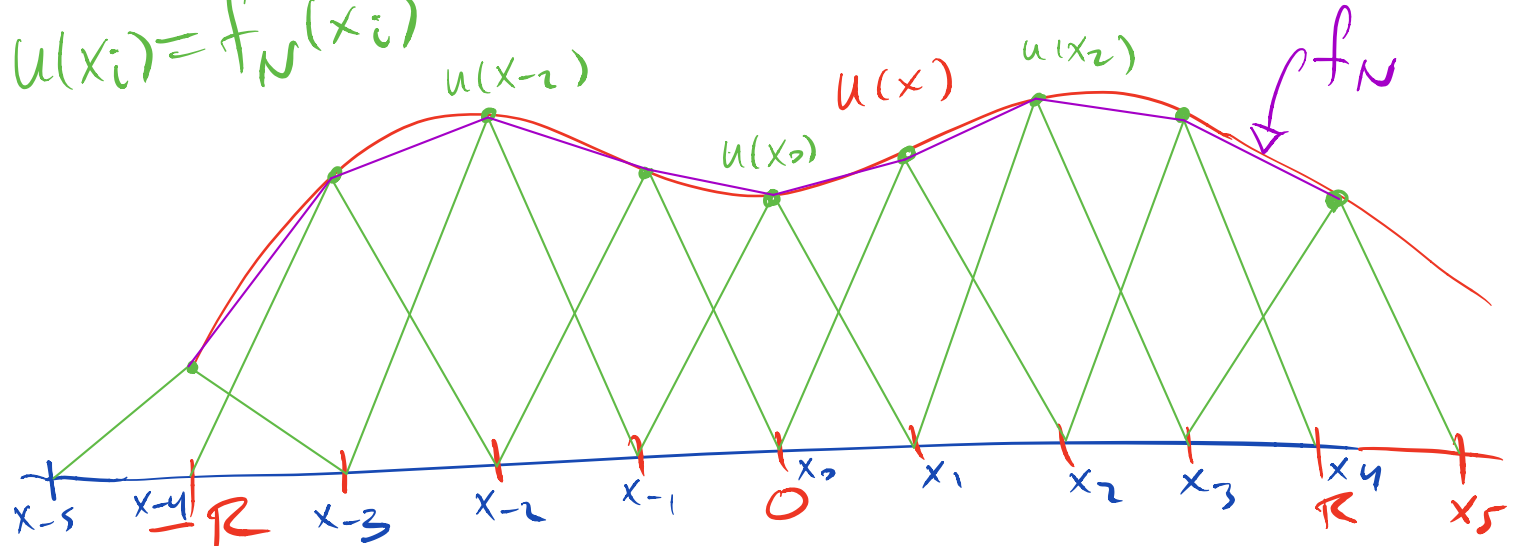
Recall



$$g(x) = (x+1)_+ - 2x_+ + (x-1)_+$$

satisfies $g(x) = 0$ for $|x| \geq 1$

$$u(x_i) = f_N(x_i)$$



We exactly interpolate the points x_i

$$u(x_i) = f_N(x_i)$$

which follows from $g(x) = 0$ for $|x| \geq 1$

$$g(h^{-1}(x-x_i)) = 0 \text{ for } |x-x_i| \geq h$$

Hence

$$g(h^{-1}(x_j-x_i)) = \begin{cases} 1, & i=j \\ 0, & i \neq j \end{cases}$$

Between grid points, f_N is linear,

and so for $x_i \leq x \leq x_{i+1}$

$$f_N(x) = \left(1 - \frac{x-x_i}{h}\right)u(x_i) + \left(\frac{x-x_i}{h}\right)u(x_{i+1})$$

$$\begin{aligned} |f_N(x) - u(x)| &= \left| \left(1 - \frac{x-x_i}{h}\right)(u(x_i) - u(x)) \right. \\ &\quad \left. + \left(\frac{x-x_i}{h}\right)(u(x_{i+1}) - u(x)) \right| \\ &\leq \left(1 - \frac{x-x_i}{h}\right) \underbrace{|u(x_i) - u(x)|}_{\leq \text{Lip}(u)h} + \left(\frac{x-x_i}{h}\right) \underbrace{|u(x_{i+1}) - u(x)|}_{\leq \text{Lip}(u)h} \end{aligned}$$

$$\leq \left(1 - \frac{x - x_i}{h} + \frac{x - x_i}{h}\right) \text{Lip}(u) h$$

$$= \text{Lip}(u) h. \quad \text{want} = \varepsilon \quad \text{Choose } h = \text{Lip}(u)^{-1} \varepsilon$$

$$h^{-1} = \text{Lip}(u) \varepsilon^{-1}$$

$$N = 3(2m+1), \quad m \leq h^{-1}R + 1$$

$$\begin{aligned} N &\leq 3(2(h^{-1}R + 1) + 1) = 6(h^{-1}R + 1) + 3 \\ &= 6h^{-1}R + 9. \end{aligned}$$

$$| \quad N = 6R \text{Lip}(u) \varepsilon^{-1} + 9$$

If u' is Lipschitz, we'll write

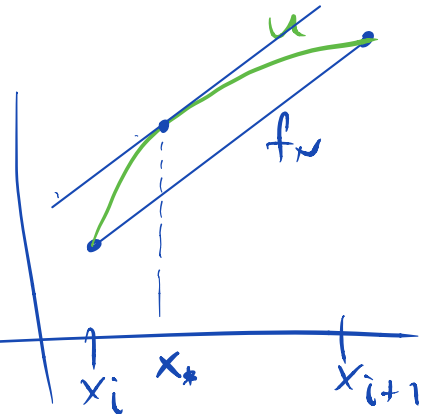
$$f_N(x) = m_i(x - x_i) + u(x_i)$$

where
$$m_i = \frac{u(x_{i+1}) - u(x_i)}{h}$$

By mean-value-theorem

$\exists x_* \in [x_i, x_{i+1}]$ such that

$$u'(x_*) = m_i = \frac{u(x_{i+1}) - u(x_i)}{h}$$



$$\begin{aligned}
f_{\mu}(x) - u(x) &= m_i(x-x_i) + u(x_i) - u(x) \\
&= u'(x_*) \frac{(x-x_i)}{h} + \frac{1}{R} u'(x_i) - u(x) \\
&= u'(x_*)(x-x_*) + u'(x_*)(x_*-x_i) + u(x_i) - u(x) \\
&= u(x_*) + u'(x_*)(x-x_*) - u(x) \\
&\quad + u(x_i) - u(x_*) - u'(x_*)(x_i-x_*)
\end{aligned}$$


Claim:

$$|u(x) - u(x_*) - u'(x_*)(x-x_*)| \leq \frac{1}{2} L_{\text{Lip}}(u') |x-x_*|^2$$

Assume $x > x_*$

$$\begin{aligned}u(x) - u(x_*) &= \int_{x_*}^x u'(t) dt \\&= \int_{x_*}^x u'(x_*) + u'(t) - u'(x_*) dt \\&= \int_{x_*}^x u'(x_*) dt + \int_{x_*}^x u'(t) - u'(x_*) dt \\&= u'(x_*) (x - x_*) + \int_{x_*}^x u'(t) - u'(x_*) dt\end{aligned}$$

$$|u(x) - u(x_*) - u'(x_*) (x - x_*)| \leq \left| \int_{x_*}^x u'(t) - u'(x_*) dt \right|$$


$$\leq \int_{x_*}^x \text{Lip}(u') |t - x_*| dt$$

$$\begin{aligned}
 &= \text{Lip}(u') \int_{x_*}^x t - x_* \, dt \\
 &= \frac{1}{2} \text{Lip}(u') |x - x_*|^2,
 \end{aligned}$$

which proves the claim \square

$$\begin{aligned}
 |f_N(x) - u(x)| &= \left| u(x_*) + u'(x_*)(x - x_*) - u(x) \right. \\
 &\quad \left. + u(x_i) - u(x_*) - u'(x_*)(x_i - x_*) \right| \\
 &\leq \left| u(x_*) + u'(x_*)(x - x_*) - u(x) \right| \\
 &\quad + \left| u(x_i) - u(x_*) - u'(x_*)(x_i - x_*) \right|
 \end{aligned}$$

Taylor expansion

$$\leq \frac{1}{2} \text{Lip}(u') |x - x_{\#}|^2 + \frac{1}{2} \text{Lip}(u') |x_i - x_{\#}|^2$$

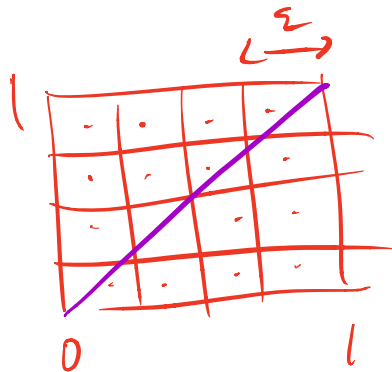
$$= \text{Lip}(u') h^2 \stackrel{\text{want}}{=} \varepsilon \quad \text{choose } h = \sqrt{\text{Lip}(u')^{-1} \varepsilon}$$

...



Note: In dimension n , need $O(\varepsilon^{-n})$
hidden nodes.

↑
curse of
dimensionality



ε^{-2} blocks

$$\|1\| = \sqrt{1 + \dots + 1} = \sqrt{n}$$

Overfitting

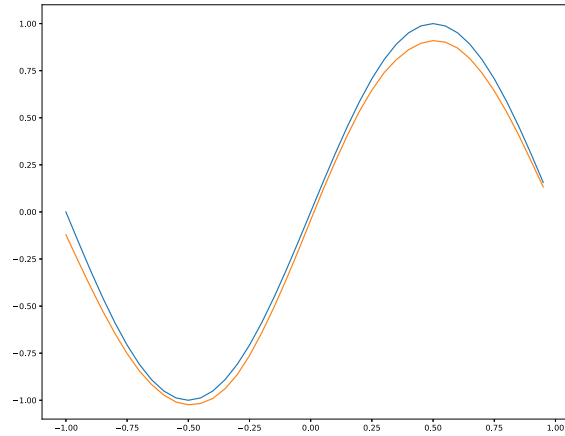
The function

$$f_N(x) = \sum_{i=-m}^m y_i g(h^{-1}(x - x_i)),$$

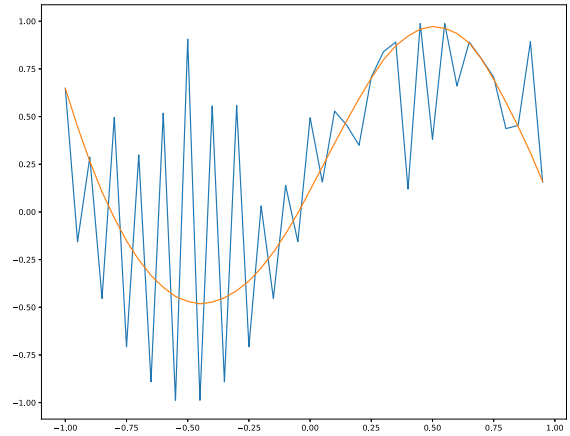
constructed in the proof exactly fits the data $f_N(x_i) = y_i$. This means a network with m nodes can exactly fit m datapoints, simply via memorization.

In practice it is hard (but not impossible) to get neural networks to overfit like this, even in the severely over parameterized regime. This is somewhat of a mystery still in the community, but is related to the optimization algorithms used to train neural networks.

Noisy labels



(a) Smooth labels



(b) Noisy labels

Figure 3: Example of a network fitting smooth and noisy data. The network has 10000 hidden nodes and only 40 training points.

Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are one of the most powerful tools for image processing. They are special cases of fully connected neural networks that replace the linear function in a neuron with the convolution

$$(W * I)(i, j) = \sum_{p, q=-N}^N W(N + 1 + p, N + 1 + q)I(i + p, j + q).$$

Here I is a 2D image, W is a $(2N + 1) \times (2N + 1)$ matrix representing the filter, which is trainable. N is the width of the filter, which is normally small compared to the size of the image I .

Key points:

- **Translation Invariance:** The convolution applies the same filter W to *all* locations in the image, finding features no matter where they are within the image.
- **Locality:** The small size of filters restricts the features to be localized in the image.

Typical CNN architecture

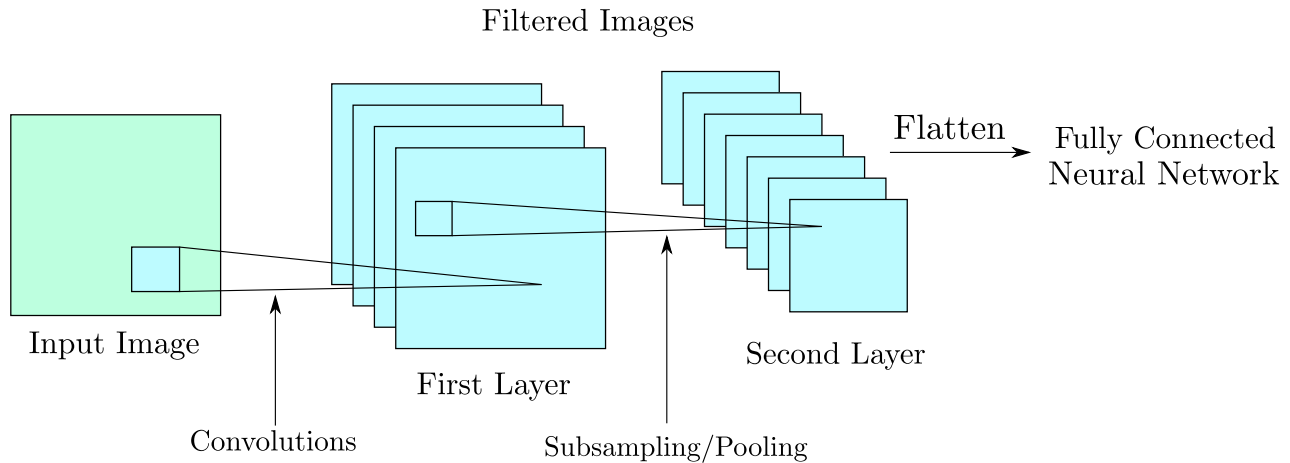


Figure 4: An example of a typical Convolutional Neural Network (CNN) architecture.

Pooling

Pooling is a form of subsampling that introduces translation invariance into CNNs, and allow future layers to pick up on larger scale features in the image, leading to a multiscale analysis.

Max Pooling (4x4) \rightarrow (2x2)

1	2	1	-1
4	7	1	0
-1	0	0	0
0	0	0	0

7	1
0	0

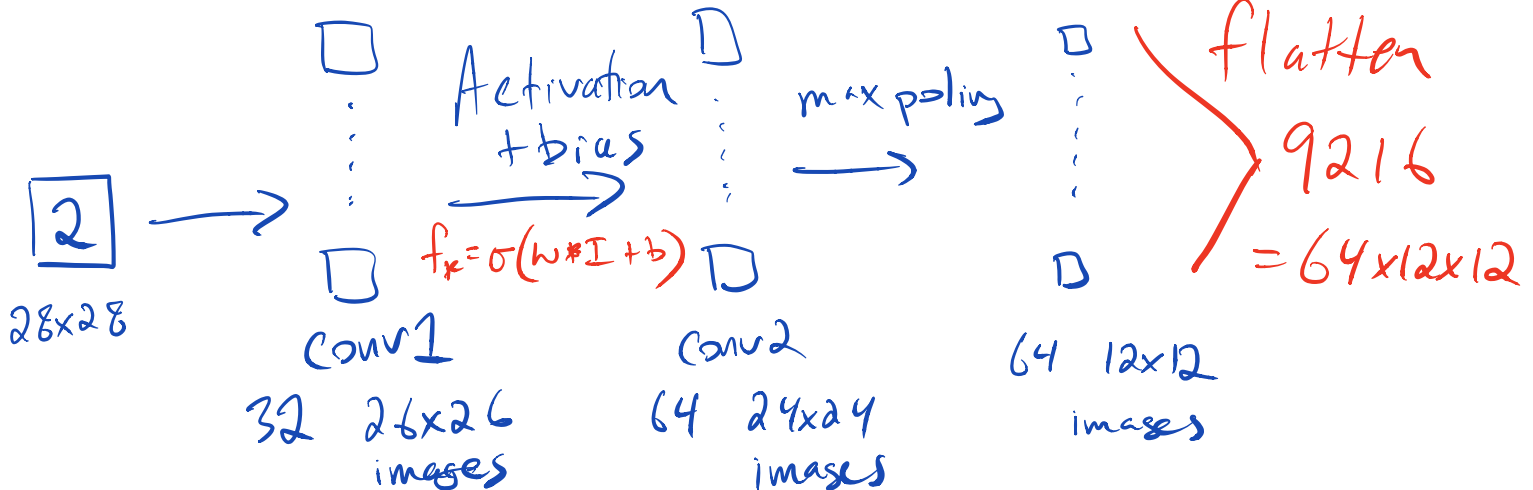
Example on MNIST

We ran an experiment testing a simple convolutional neural network for classification of MNIST digits.

- 4 layer neural network

- First 2 layers are convolutional with 32 and 64 channels
- Final 2 layers are fully connected with 128 hidden nodes.

- Output of the convolutional layers is flattened into a length 9216 array of features to feed into the fully connected layer.



MNIST Accuracy

- Accuracy is 99.04% after 14 epochs.
- 98.07% after single pixel shift
- 92.06% after two-pixel shift
- 75% after three-pixel shift

The translation invariance is better than with fully connected networks due to the pooling and translation invariance of convolutional. To get better translation invariance we can introduce more pooling into the network.

Convolutional filters

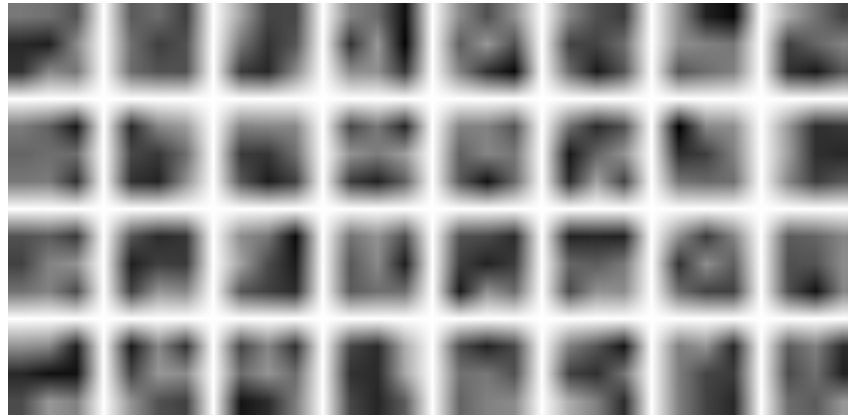


Figure 5: The 32 3×3 filters from the first layer of the convolutional neural network for classifying MNIST digits.

First layer channels



Figure 6: The 32 channels of output from the first convolutional layer acting on an image of a 2.

Second layer channels

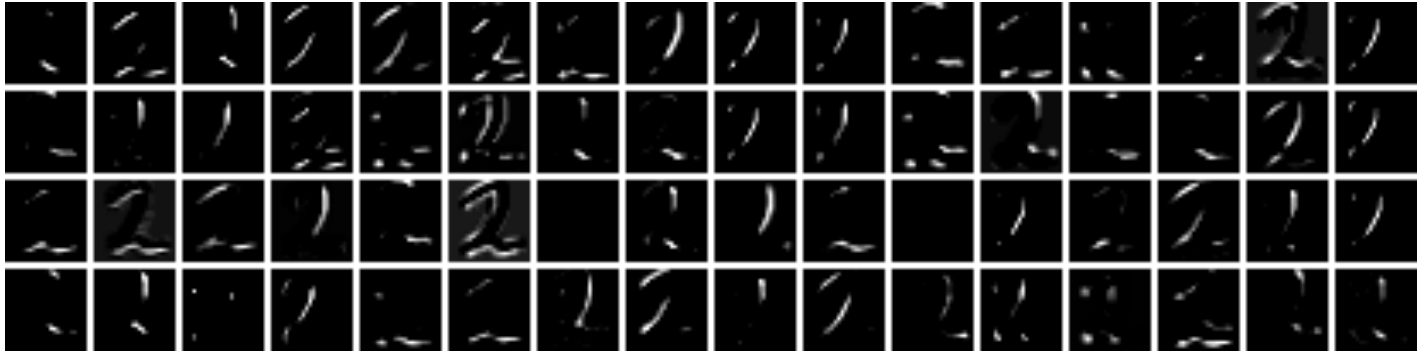
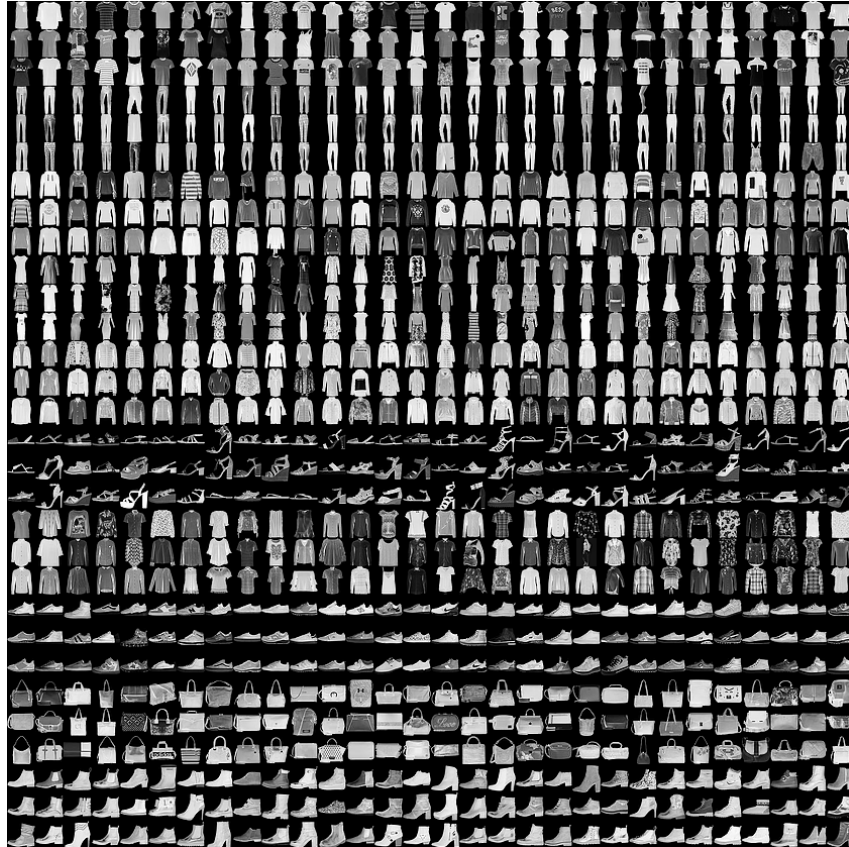


Figure 7: The 64 channels of output from the second convolutional layer acting on an image of a 2. Notice the channels appear to be detecting edges in various directions.

FashionMNIST



FashionMNIST accuracy

- Accuracy is 92.55% after 14 epochs.
- 89.73% after single pixel shift
- 75.94% after two-pixel shift
- 47.38% after three-pixel shift

We could choose a larger network with more channels and hidden layers to achieve better results (around 99%) for FashionMNIST.

Convolutional Neural Networks (.ipynb)