Session 6 - Backpropagation and Advanced Architectures

- Due Saturday by 9am
- Points 0
- Available after Feb 24 at 12am

Session 6 - Backpropagation & Architectural Basics

We'll cover backpropagation followed by architectural basics today.

Backpropagation

Neural networks come in many forms, but the main form we need to master is that of "fully connected layers". A network with many such (or other) layers is called "deep".

Let's look at a very simple neural network.

Let's make sure we talk about the Error and path small errors are taking to finally get accumulated at E_Total.

What is that **σ** doing there?

A linear activation function has two major problems: (https://www.analyticssteps.com/blogs/7-types-activation-functions-neural-network)

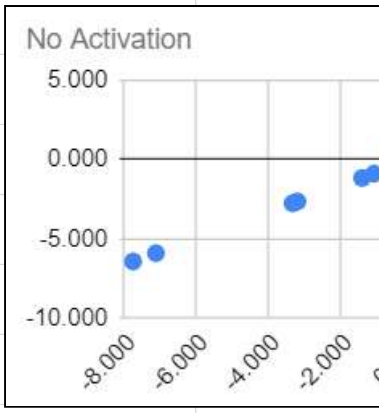
1. Not possible to use backpropagation (gradient descent) to train the model—the derivative of the function is a constant and has no relation to the input, X. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.

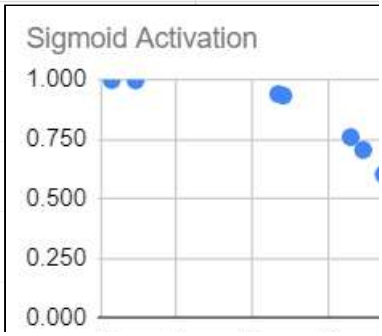
2. All layers of the neural network collapse into one—with linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network into just one layer.

That is why we need activation functions (other than just linear).

Activations

Input	Weight	Tem	Ou	No Act	ReLU	Sigmoid	Tanh
-0.488	0.831		-0.406	-0.406	0.000	0.600	-0.385
0.850			0.706	0.706	0.706	0.330	0.608
-0.506			-0.421	-0.421	0.000	0.604	-0.398
-3.171			-2.635	-2.635	0.000	0.933	-0.990
-7.724			-6.419	-6.419	0.000	0.998	-1.000
2.573			2.138	2.138	2.138	0.105	0.973
-3.305			-2.747	-2.747	0.000	0.940	-0.992
0.205			0.170	0.170	0.170	0.458	0.168
-1.050			-0.873	-0.873	0.000	0.705	-0.703
0.000			0.000	0.000	0.000	0.500	0.000
0.086			0.072	0.072	0.072	0.482	0.072

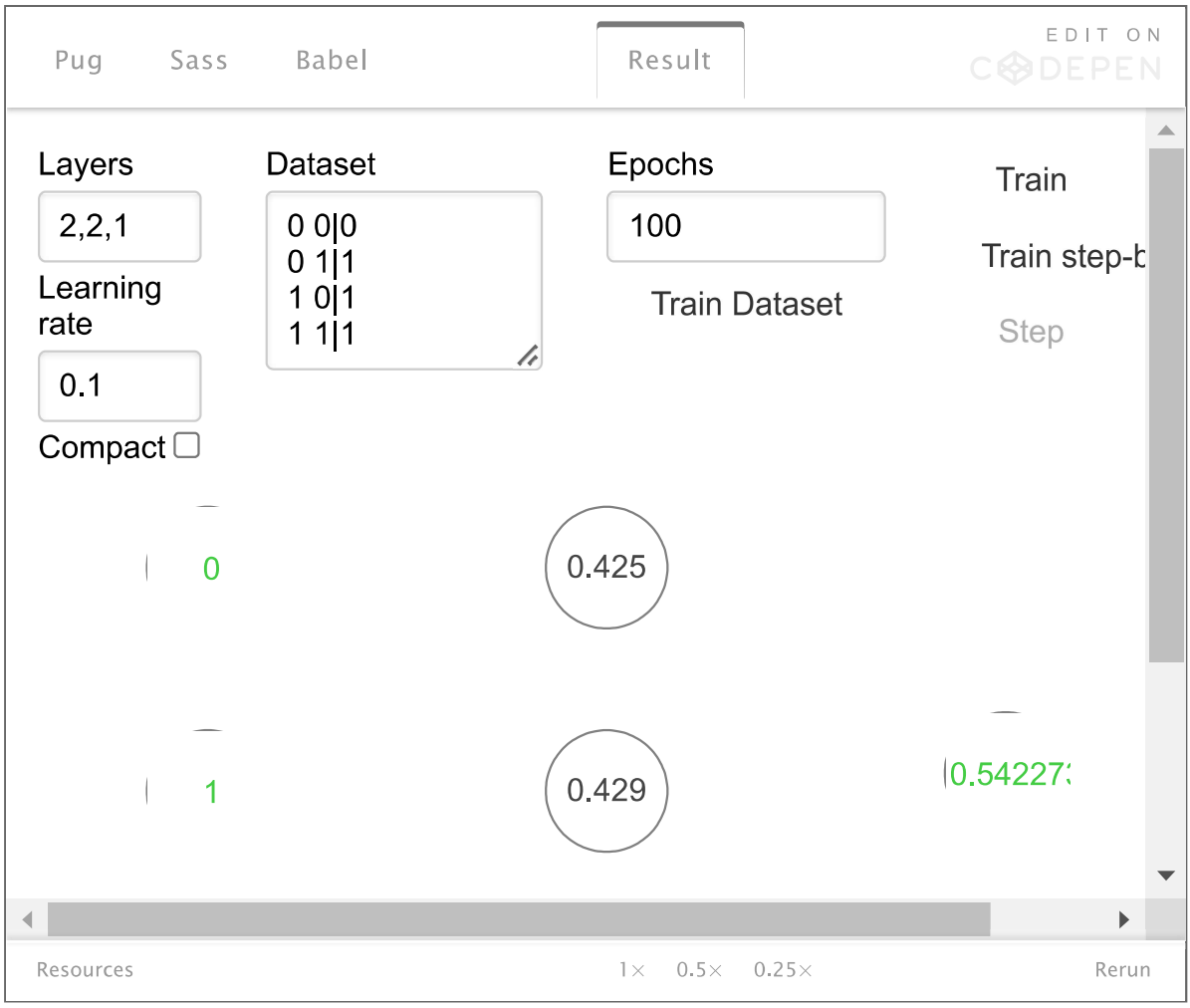




Sheet1 Forward Backward Next Forward

Let's look at **this (https://xnought.github.io/backprop-explainer/)** amazing backpropagation explainer.

Let's make an NN in the **Excel Sheet (https://canvas.instructure.com/courses/8491182/files/245030782?wrap=1) (https://canvas.instructure.com/courses/8491182/files/245030782/download?download_frd=1)** and this time with full backpropagation involved!



Fully Connected Layers

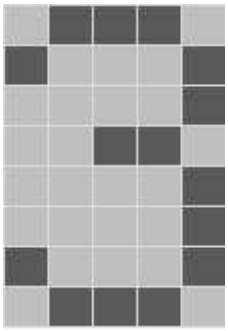
This is how Fully Connected Layers look like:

Each line you see is the **weight** we are training. Those circles are "temporary" values that will be stored. Once you train the model, *lines are what all matter!*

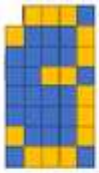
We know from our past session what digit this pattern represents.

But still! Exactly, that's the point.

This is what it actually represents.

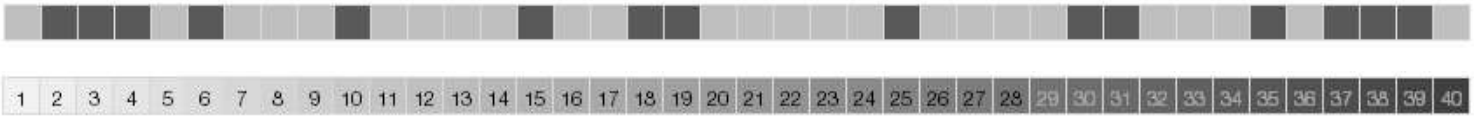


The process which converts our 2D data into 1D is shown below:



As you can see we lose spatial information when we use fully connected layers.

What if we can add this information back?



We have tried to add this information back in a linear fashion, but that didn't result in huge improvements and wasn't researched further for longer.

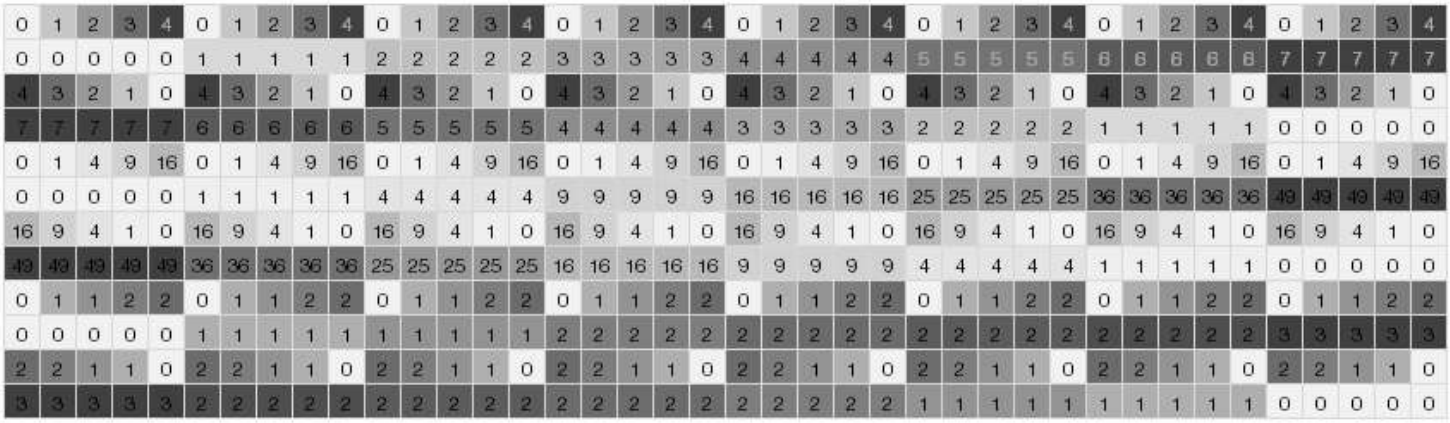
More ideas on position encoding:

The above one was definitely an improvement (for images), but why do you think this data representation should cause an improvement?

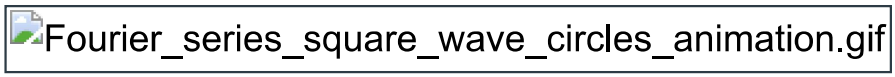
For representing 1 data point, we are now using 2 dimensions.

This is a general trend that we'll see. The more dimensions that we use, the better the representation is.

But how much can we really go?

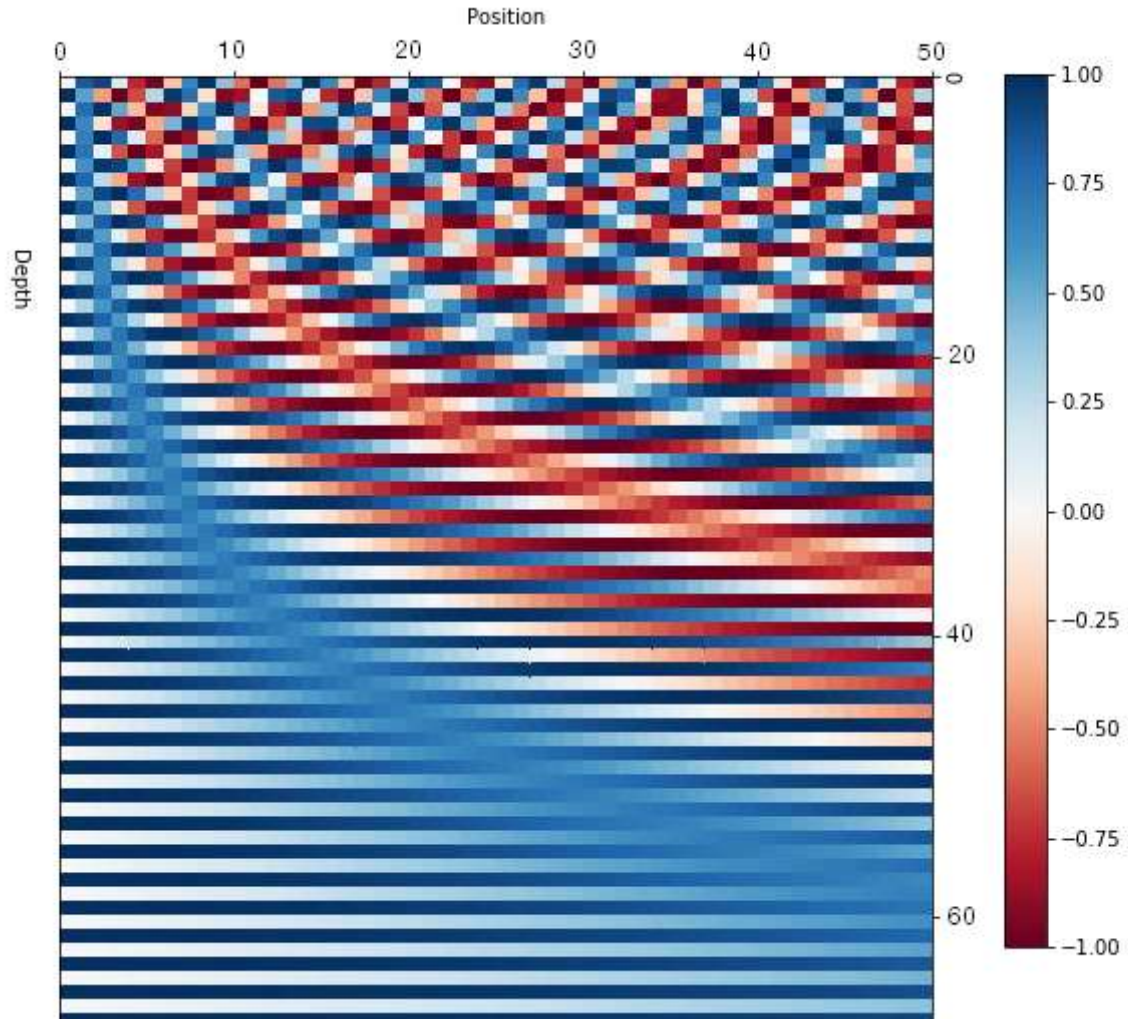


The positional encoding ideas above were a good idea but were very difficult for DNNs to take out when merged with the data. That's when people started thinking about Fouriers! Fourier transforms can represent nearly every kind of signal, and DNNs can extract them when mixed with our actual word embeddings:

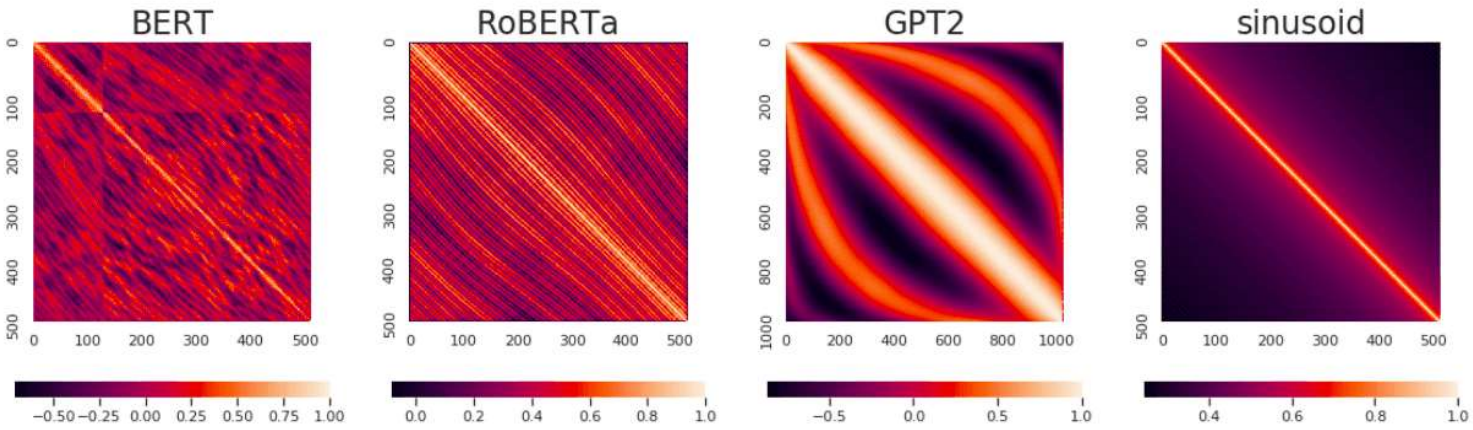


Ideas from FTs helped in designing better positional encodings (all of this is what we'll cover in future segments, but is here to remind you why FC layers are still relevant)

Above by the way is a human representation of how PE should be designed.



But then we asked NNs, and here is their answer:



DNNs are only slightly better at designing positional encodings, hence there is no consensus on whether to move away from our FT-based encodings or not.

VGG - When the last layer screwed it up! - Pre-2014

PARAMETERS

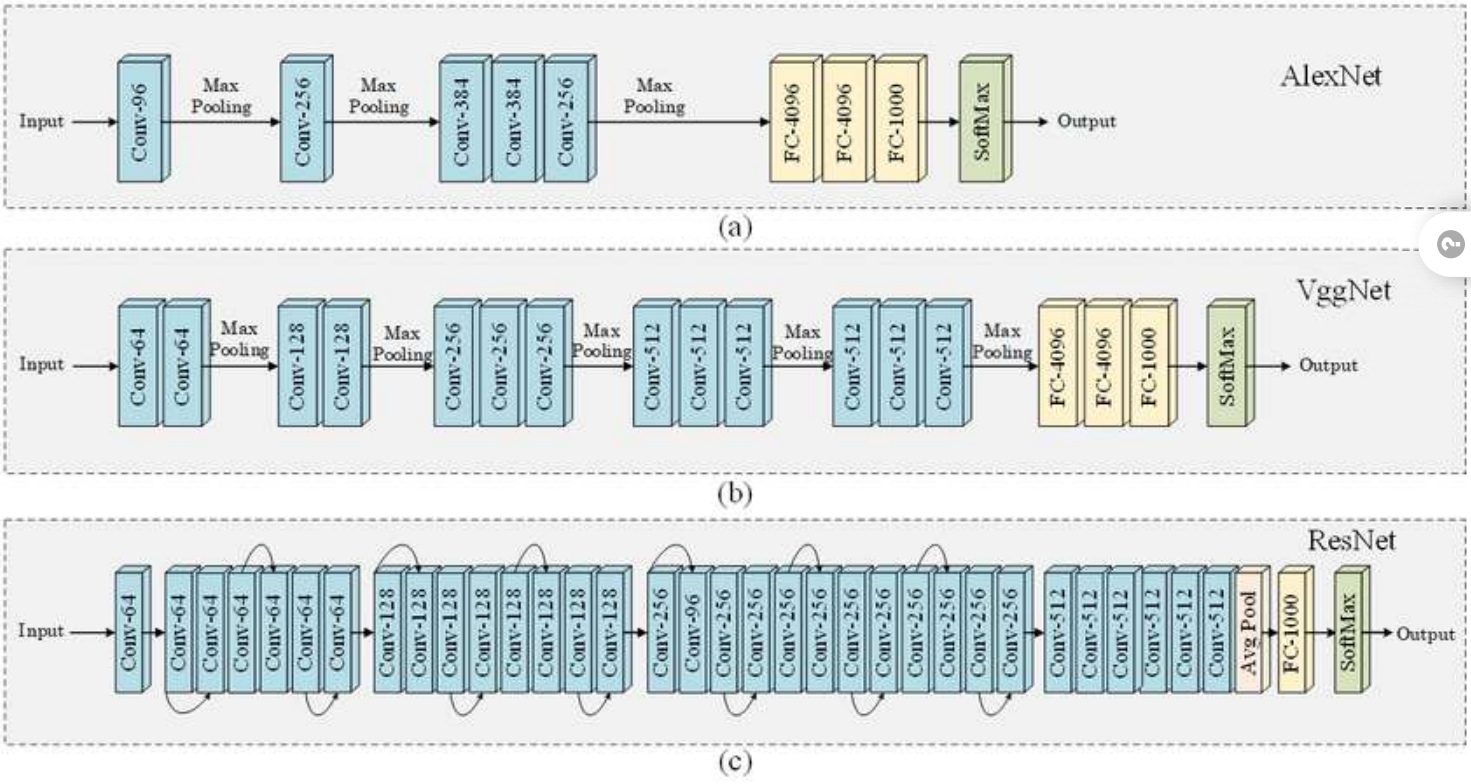
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]

Total params: 138357544

Modern Architecture - Post-2014

What we have covered (expand-squeeze channels) as architecture is for understanding only. This architecture is called Squeeze and Expand.

Most of the modern architecture follows this architecture:



Major Points:

1. ResNet is the latest among the above. You can clearly note 4 major blocks.
2. The total number of kernels increases from 64 > 128 > 256 > 512 as we proceed from the first block to the last (unlike what we discussed where at each block we expand to 512. Both architectures are correct, but 64 ... 512 would lead to lesser computation and parameters.

3. Only the most advanced networks have ditched the FC layer for the GAP layer. In TSAI we will only use GAP Layers.
4. Nearly every network starts from 56x56 resolution!

SoftMax

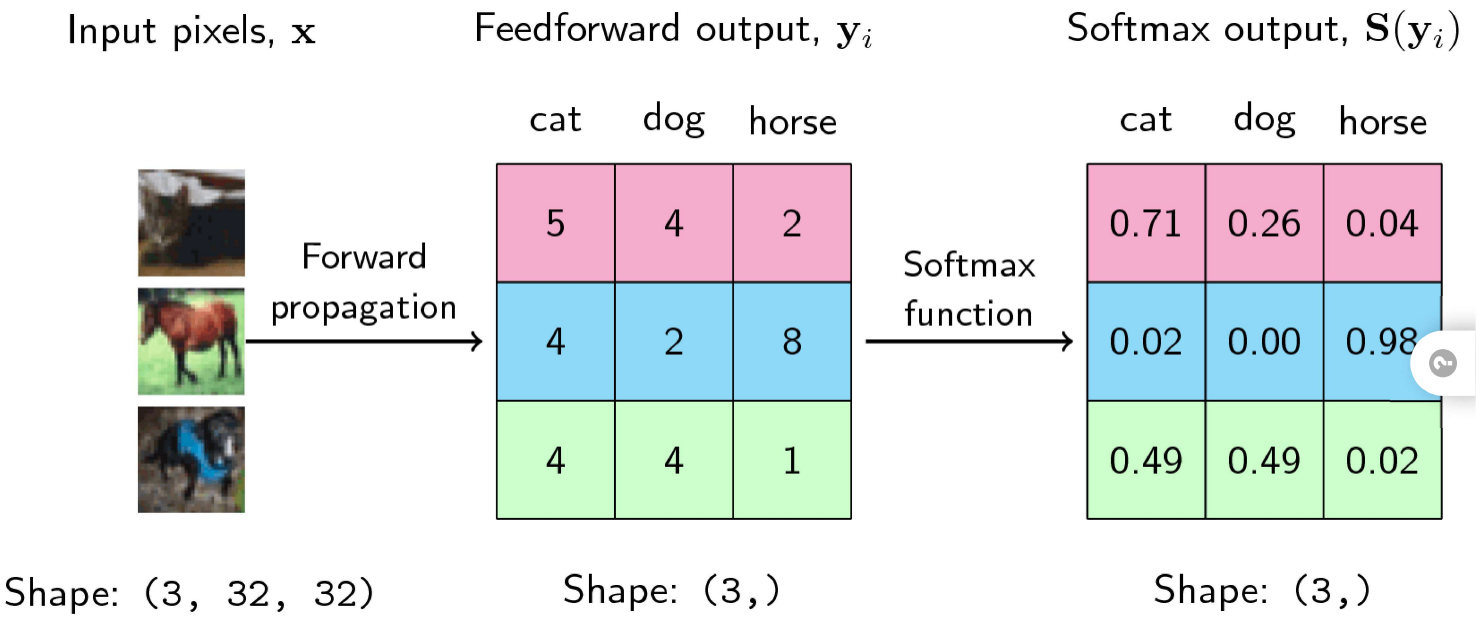
This is what our code looked like after the second session:

```
x = F.relu(self.conv7(x)) # never before the last layer
x = x.view(-1, 10)
return F.log_softmax(x)
```

First SoftMax. What is softmax?

In mathematics, the softmax function, also known as softargmax[1] or normalized exponential function is a function that takes as input a vector of K real numbers and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.

The **softmax** function is often used in the final layer of a neural network-based classifier.



Why Softmax is not probability, but likelihood!

The output of the softmax describes the ***likelihood*** (https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/) (or if you may, the confidence) of the neural network that a particular sample belongs to a certain class. Thus, for the first example above, the neural network assigns confidence of 0.71 that it is a cat, 0.26 that it is a dog, and 0.04 that it is a horse. The same goes for each of the samples above.

We can then see that one advantage of using the softmax at the output layer is that it improves the interpretability of the neural network. By looking at the softmax output in terms of the network's confidence, we can then reason about the behavior of our model.

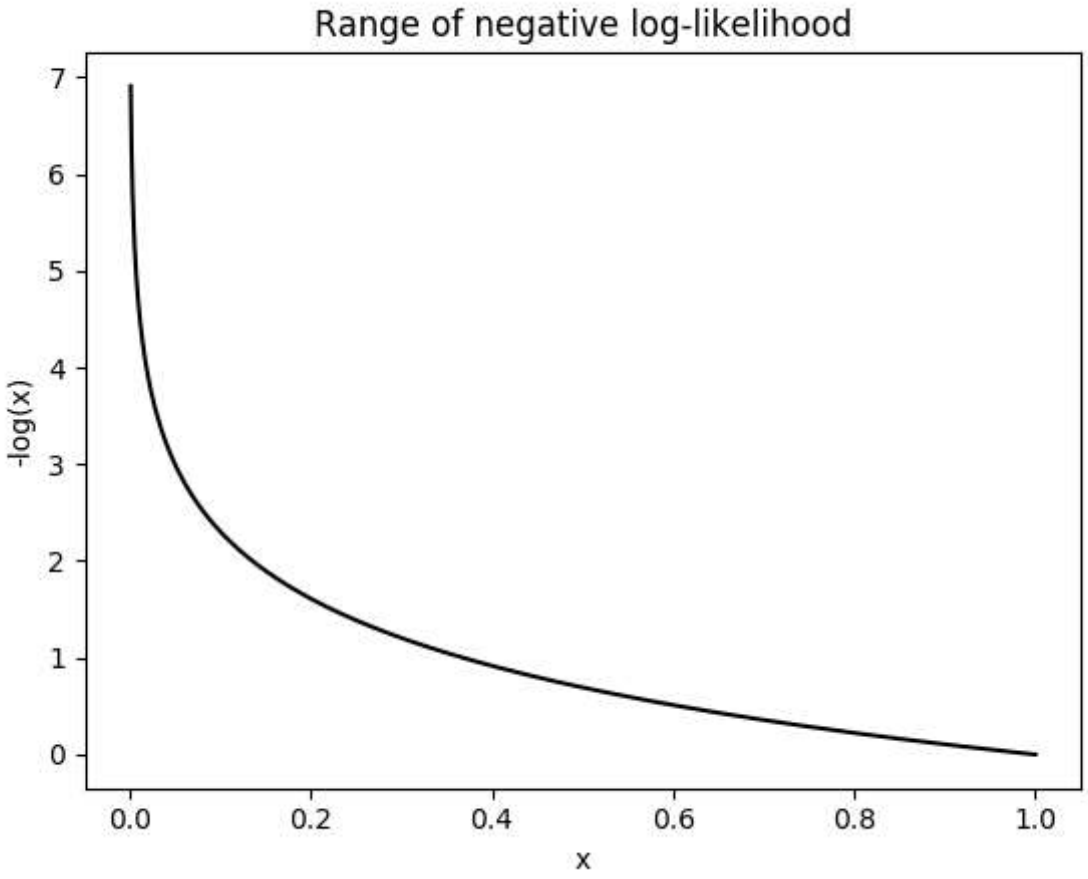
Negative Log-Likelihood

In practice, the softmax function is used in tandem with the negative log-likelihood (NLL). This loss function is very interesting if we interpret it in relation to the behavior of softmax. First, let's write down our loss function:

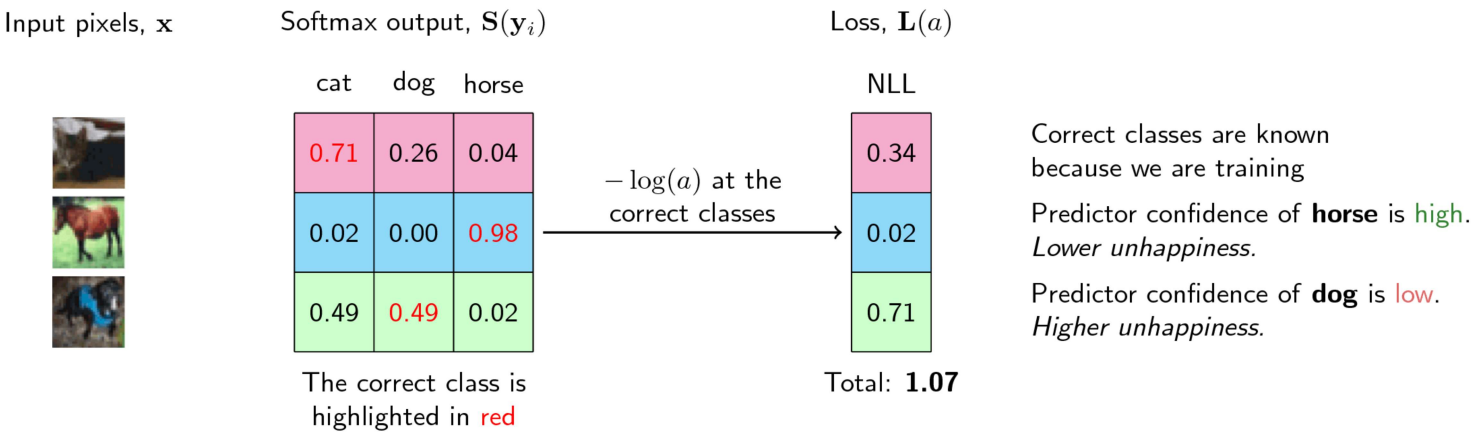
$$L(y) = -\log(y)$$

When we train a model, we aspire to find the minima of a loss function given a set of parameters (weights). We can interpret the loss as the "unhappiness" of the neural network concerning its parameters. The higher the loss, the higher the unhappiness, which we don't want. We want to make our network happy.

So if we are using the negative log-likelihood as our loss function, when does it become unhappy? And when does it become happy? Let's see a plot:



The negative log-likelihood becomes unhappy at smaller values, where it can reach infinite unhappiness (that's too sad), and becomes happy at larger values. *Because we are summing the loss function to all the correct classes*, what's actually happening is that whenever the network assigns high confidence at the correct class, the unhappiness is low, and vice-versa.



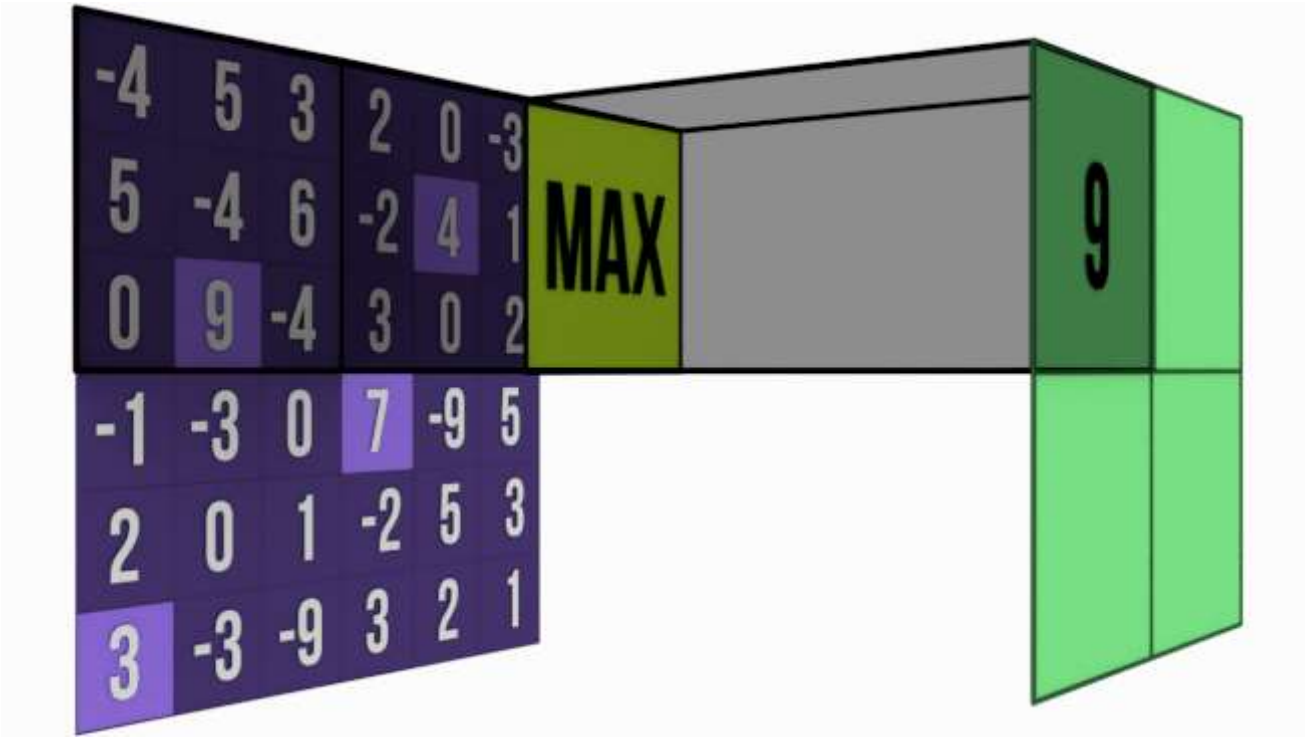
But why not log(SoftMax)?

```
def log_softmax(input, dim=None, _stacklevel=3):
    r"""Applies a softmax followed by a logarithm.
    While mathematically equivalent to log(softmax(x)), doing these two
    operations separately is slower, and numerically unstable. This function
    uses an alternative formulation to compute the output and gradient correctly.
    See :class:`~torch.nn.LogSoftmax` for more details.
    Arguments:
        input (Variable): input
        dim (int): A dimension along which log_softmax will be computed.
    """
```

[Reference (https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/)]

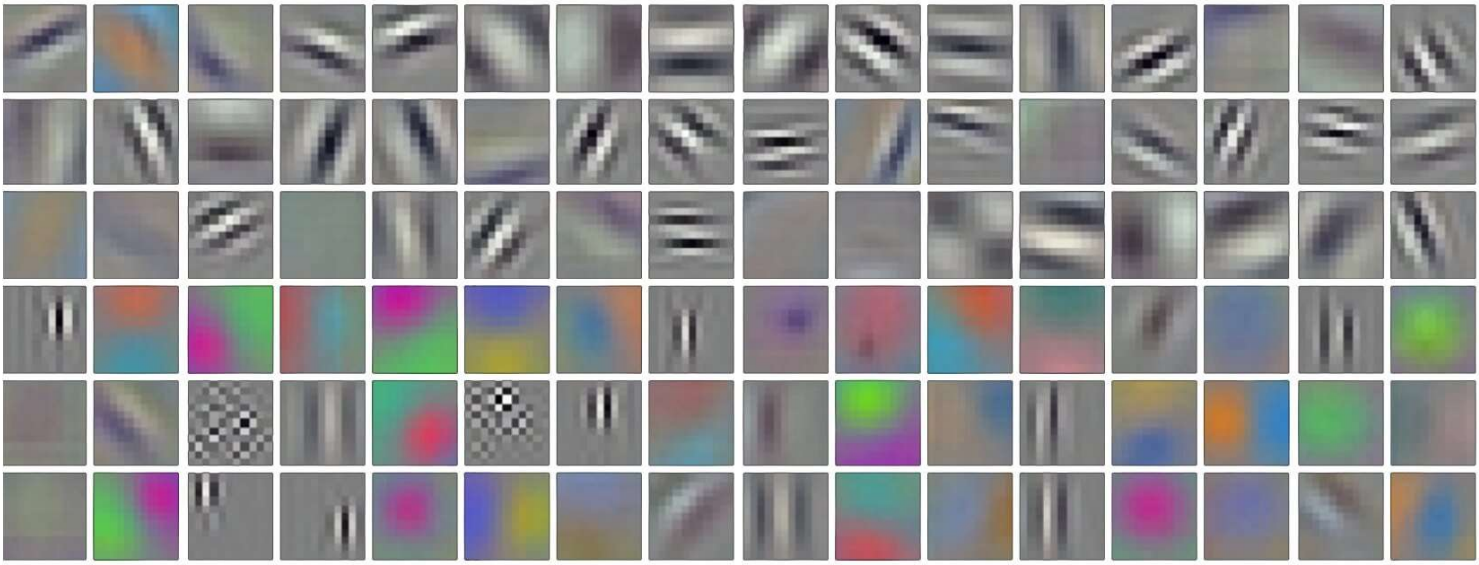
Architectural Blocks

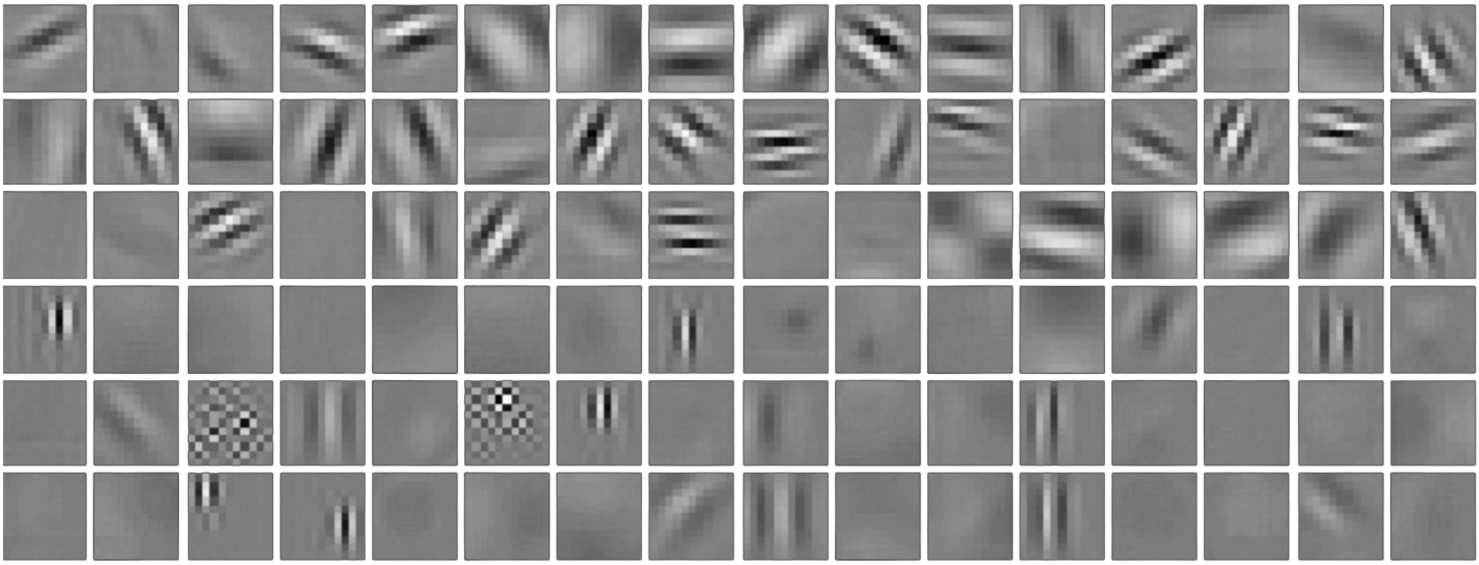
MAX POOLING

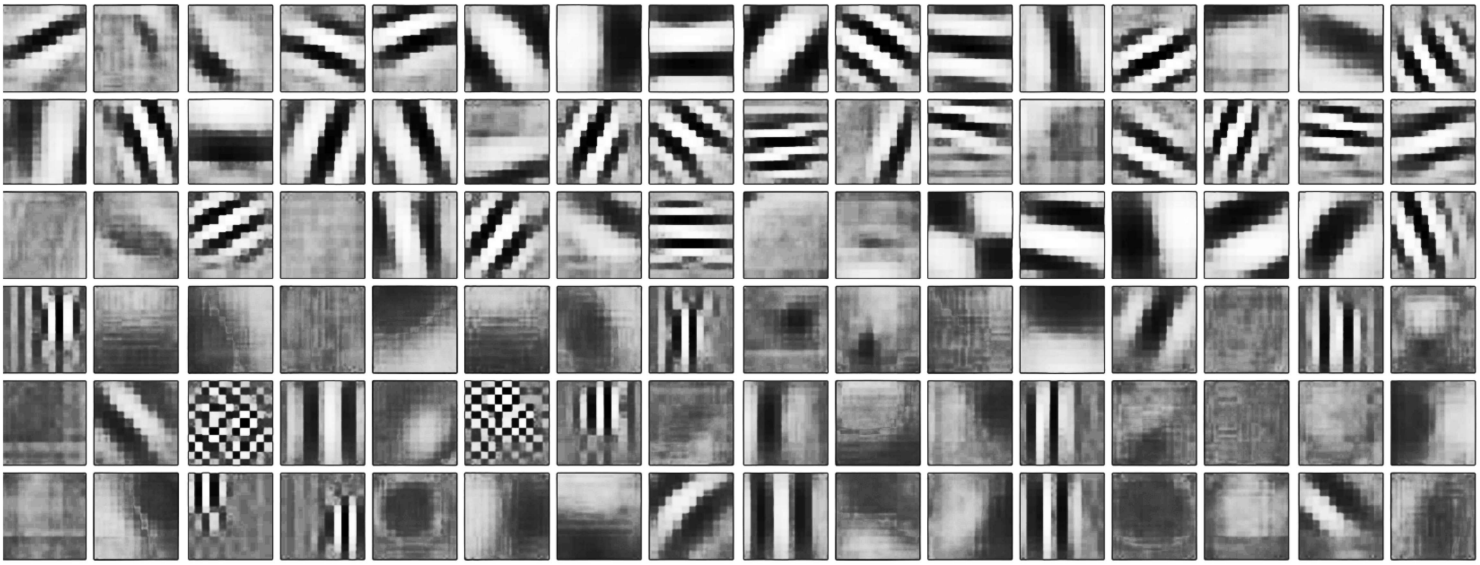


- used when we need to reduce channel dimensions
- not used close to each other
- used far off from the final layer
- used when a block has finished its work (edges-gradients/textures-patterns/parts-of-objects/objects
- nn.MaxPool2D()

BATCH NORMALIZATION





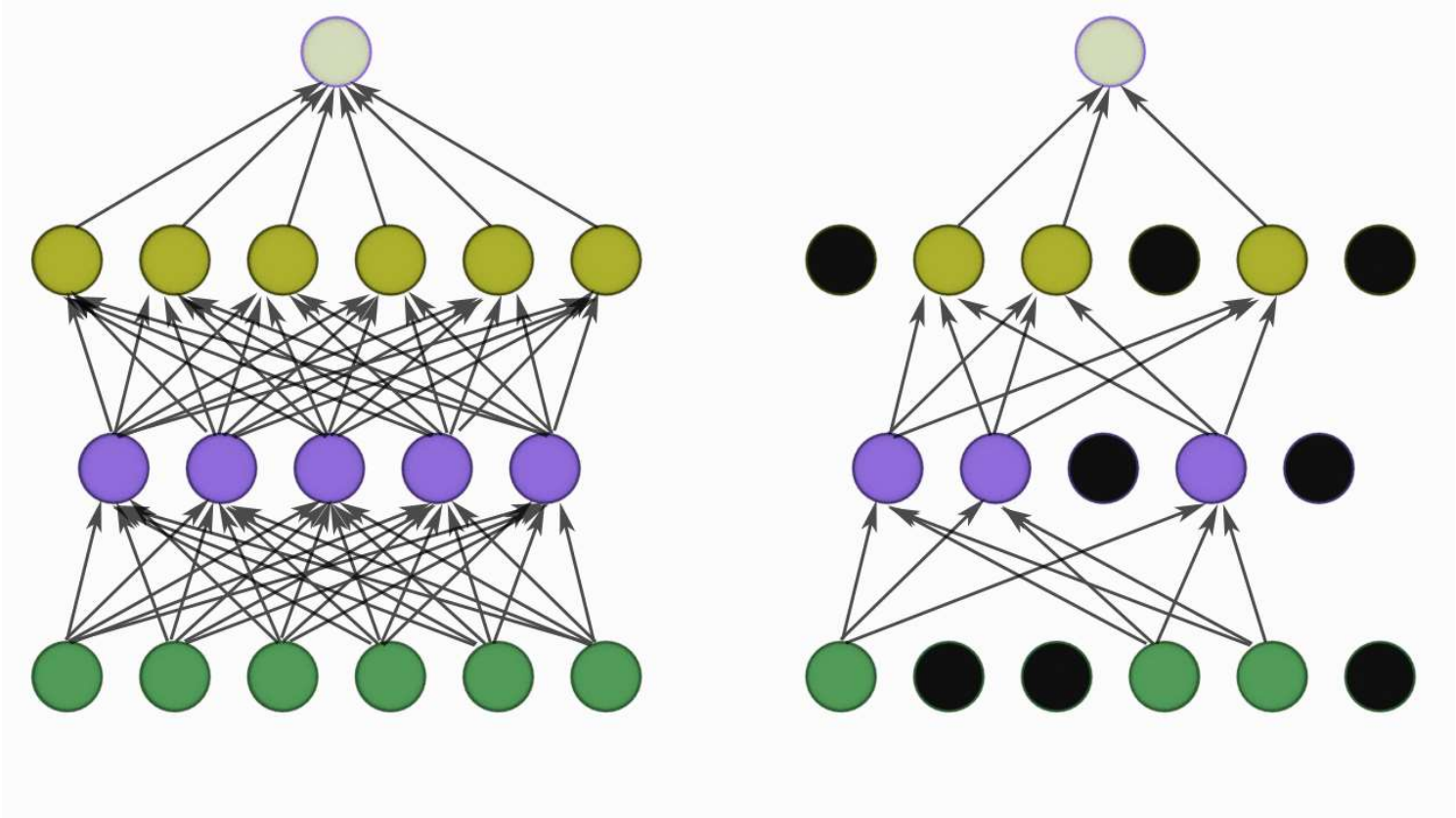


We need a whole dedicated session for BN (next session), so that's all for BN

- used after every layer
- never used before last layer
- *indirectly you have sort of already used it!*
- nn.BatchNorm2d()

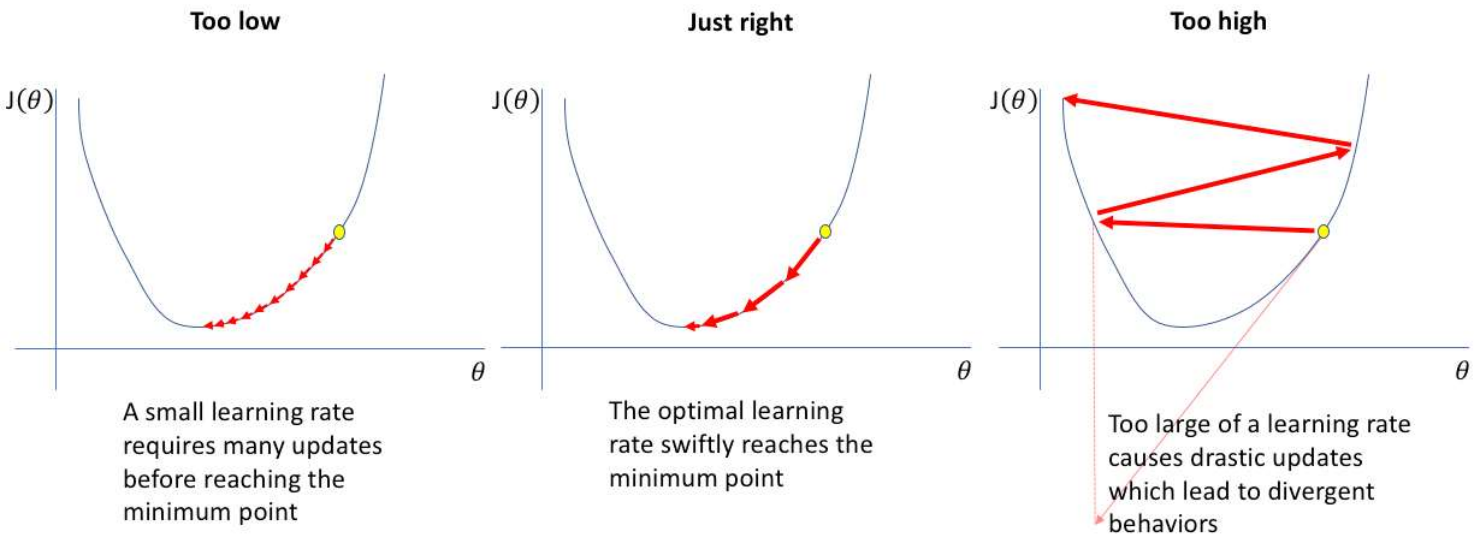
DROPOUT

Needs further discussion about what we will have today (next session)



- used after every layer
- used with small values
- is a kind of regularization
- *not used at a specific location*
- nn.Dropout2d()

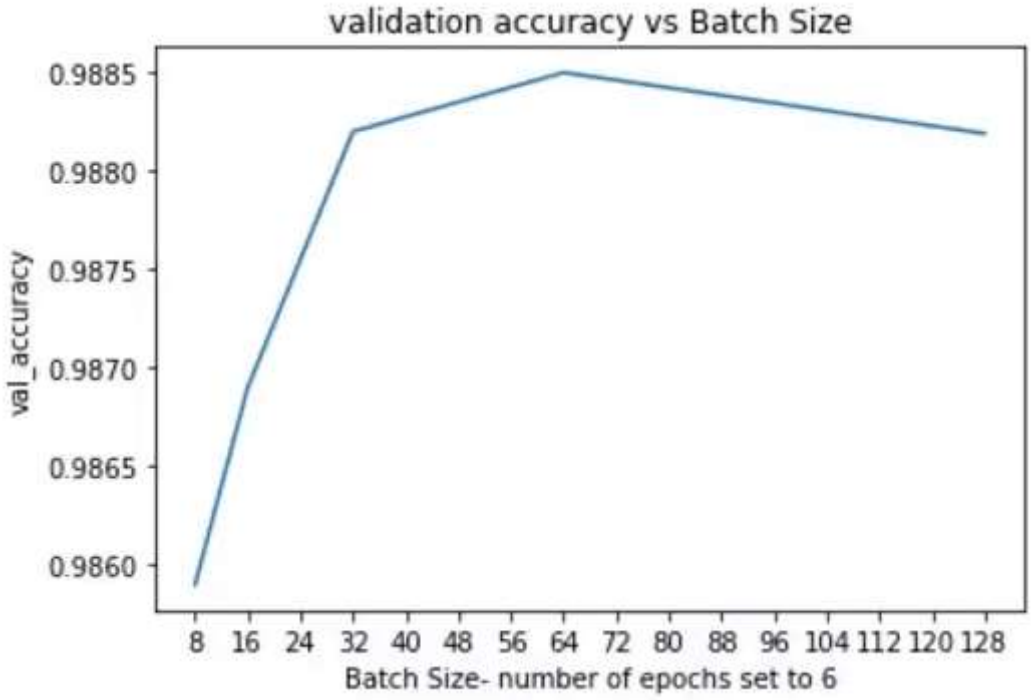
LEARNING RATE



Needs a dedicated session (6-7)

- stick to suggested values
- the main tool to achieve moksh
- optimizer = optim.SGD(model.parameters(), **lr**=0.01, momentum=0.9)

BATCH SIZE



Will be covered in further detail in sessions to come

- don't break your head, look at this curve and see the diff in Val accuracies.
- used as a part of data_loader

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,),(0.3081,))
                   ])),
                   batch_size=batch_size, shuffle=True)
```

Assignment

Assignment:

1. **PART 1[250]:** Rewrite the whole Excel sheet showing backpropagation. Explain each major step, and write it on GitHub.
    1. **Use exactly the same values for all variables as used in the class**
    2. Take a screenshot, and show that screenshot in the readme file
    3. The Excel file must be there for us to cross-check the image shown on readme (no image = no score)
    4. Explain each major step
    5. Show what happens to the error graph when you change the learning rate from [0.1, 0.2, 0.5, 0.8, 1.0, 2.0]
    6. Upload all this to GitHub and then write all the above as part 1 of your README.md file.
    7. Submit details to S6 - Assignment QnA.
2. **PART 2 [250]:** We have considered many points in our last 5 lectures. Some of these we have covered directly and some indirectly. They are:
    1. How many layers,
    2. MaxPooling,
    3. 1x1 Convolutions,
    4. 3x3 Convolutions,
    5. Receptive Field,
    6. SoftMax,
    7. Learning Rate,
    8. Kernels and how do we decide the number of kernels?

9. Batch Normalization,
10. Image Normalization,
11. Position of MaxPooling,
12. Concept of Transition Layers,
13. Position of Transition Layer,
14. DropOut
15. When do we introduce DropOut, or when do we know we have some overfitting
16. The distance of MaxPooling from Prediction,
17. The distance of Batch Normalization from Prediction,
18. When do we stop convolutions and go ahead with a larger kernel or some other alternative (which we have not yet covered)
19. How do we know our network is not going well, comparatively, very early
20. Batch Size, and Effects of batch size
21. etc (you can add more if we missed it here)

3. Refer to this code: **COLABLINK (https://colab.research.google.com/drive/1uJZvJdi5VprOQHROtJIHy0mnY2afjNlx)**
    1. **WRITE IT AGAIN SUCH THAT IT ACHIEVES**
        1. 99.4% validation accuracy
        2. Less than 20k Parameters
        3. You can use anything from above you want.
        4. Less than 20 Epochs
        5. Have used BN, Dropout,
        6. (Optional): a Fully connected layer, **have used GAP.**
        7. To learn how to add different things we covered in this session, you can refer to this code: **https://www.kaggle.com/enwei26/mnist-digits-pytorch-cnn-99 (https://www.kaggle.com/enwei26/mnist-digits-pytorch-cnn-99)** DONT COPY ARCHITECTURE, JUST LEARN HOW TO INTEGRATE THINGS LIKE DROPOUT, BATCHNORM, ETC.
4. This is a slightly time-consuming assignment, please make sure you start early. You are going to spend a lot of effort running the programs multiple times
5. Once you are done, submit your results in S6-Assignment-Solution
6. **You must upload your assignment to a public GitHub Repository. Create a folder called S6 in it, and add your iPynb code to it. THE LOGS MUST BE VISIBLE. Before adding the link to the submission make sure you have opened the file in an "incognito" window.**

7. **If you misrepresent your answers, you will be awarded -100% of the score.**
8. **If you submit a Colab Link instead of the notebook uploaded on GitHub or redirect the GitHub page to Colab, you will be awarded -50%**
9. Submit details to S6 - Assignment QnA.

Video

STUDIO

GM


ERA V2 Session S6