Machine Learning

Overview, Neural Networks, NIST Special Database 19 and other topics

Adam Robinson

2020-05-30

Background

What Qualifies as Machine Learning?

- Tunes a mathematical model to fit a dataset "*learning* how to reproduce it"
- Essentially a **really** fancy case of linear regression (but not linear in most cases)

Regression

(Reproducing Real Valued Data)

$$f : \mathbb{R}^N \rightarrow \mathbb{R}^M$$

Classification

(Classifying Members of a Dataset)

$$f : \mathbb{R}^N \rightarrow \mathbb{N}^M$$

$$N \equiv \text{number of inputs}$$

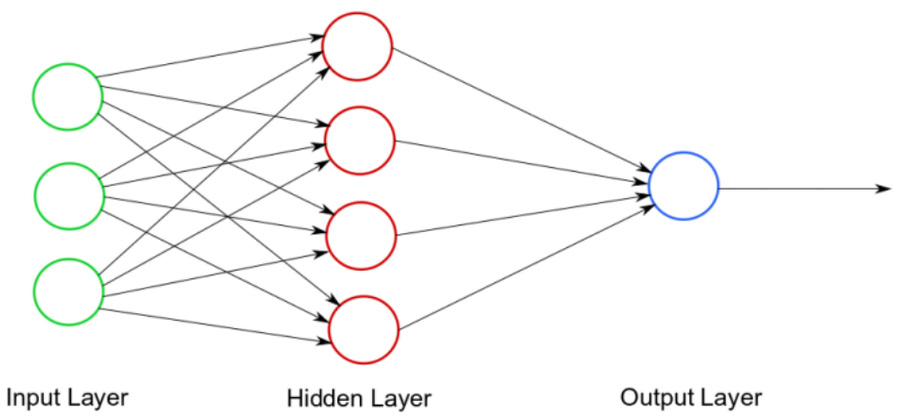
$$M \equiv \text{number of outputs}$$

Types of Models

Supervised

(Your Dataset Contains Correct Answers)

Neural Networks (Very Popular)



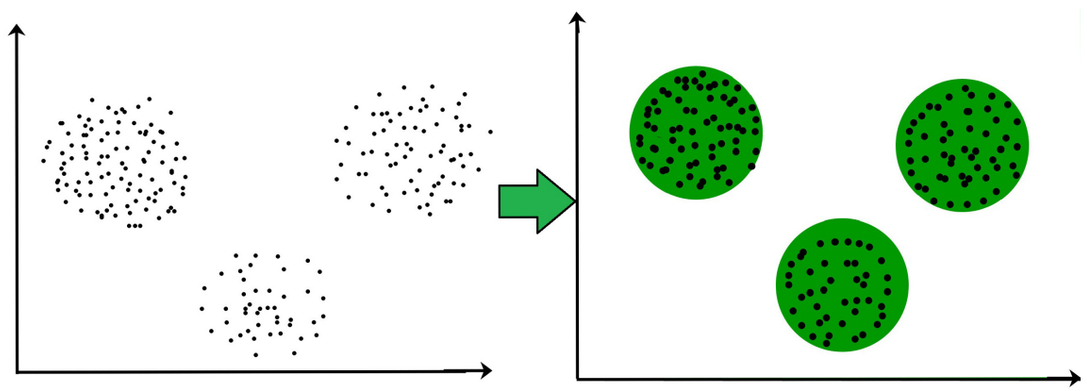
Other Types

- Support Vector Machine
- Random Forest
- Logistic Regression

Unsupervised

(Your Dataset Does **Not** Contain Correct Answers)

Clustering Models



Clustering Models

- K-Means
- Mixture Models
  - BGMM
- DBSCAN

https://www.geeksforgeeks.org/clustering-in-machine-learning/

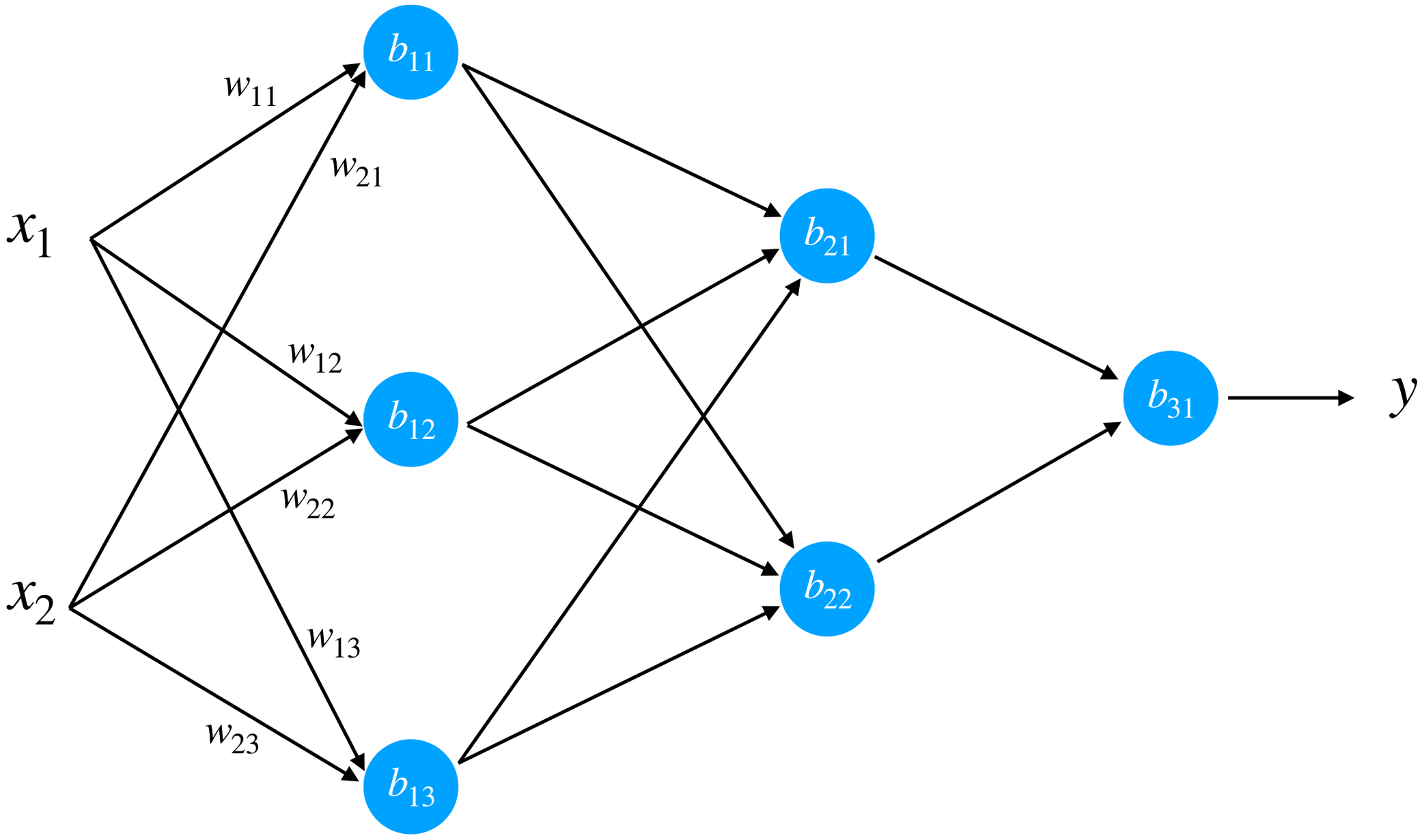
https://www.kdnuggets.com/2017/10/neural-network-foundations-explained-gradient-descent.html

The Training Process

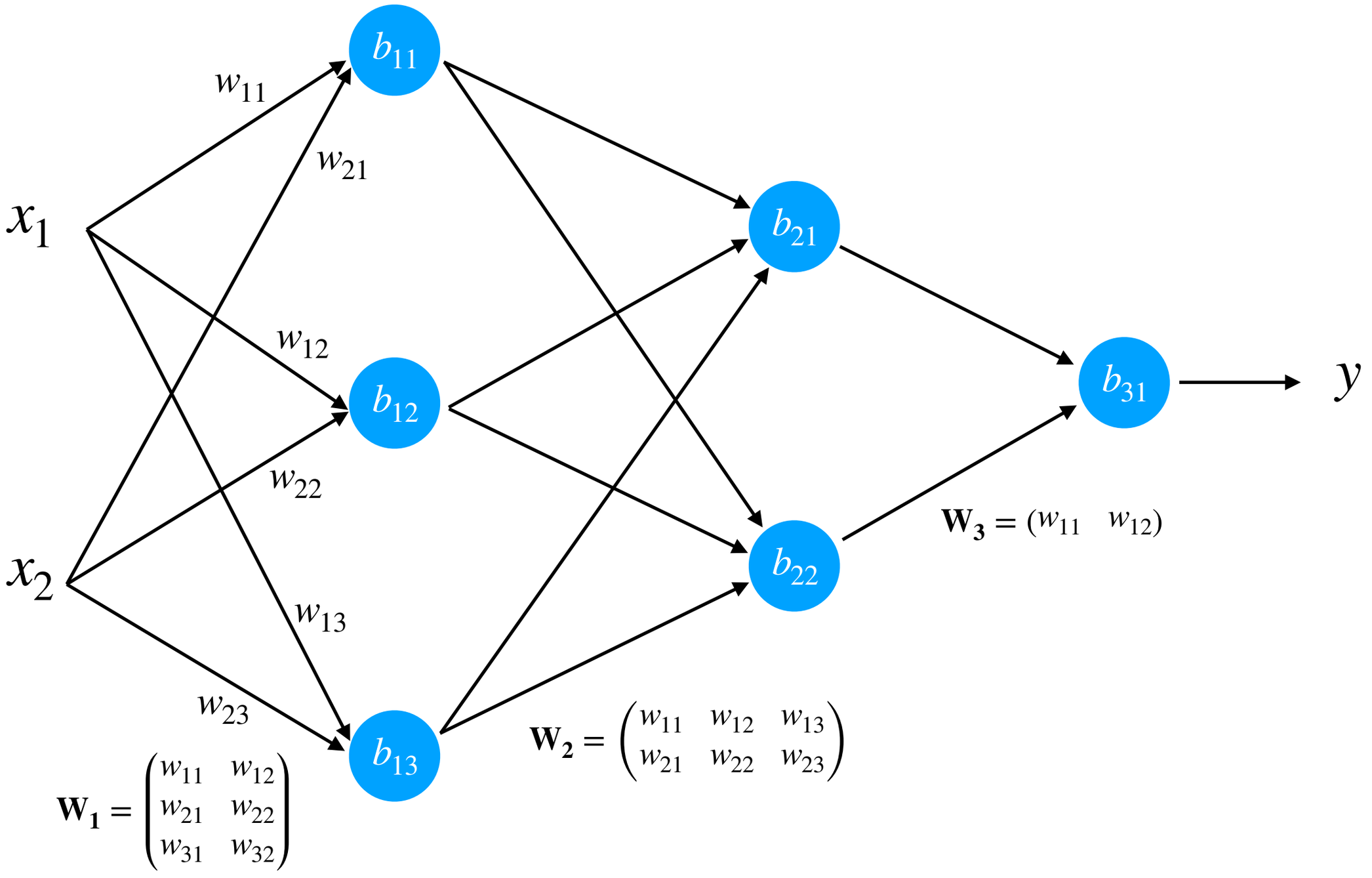
1. Split the data into a training and validation set
2. Evaluate the model using the training data
3. Compare the model results to the known answer
4. Compute the error
5. Correct the model parameters
6. Check the error against the validation dataset
7. Go back to step 2

Neural Networks

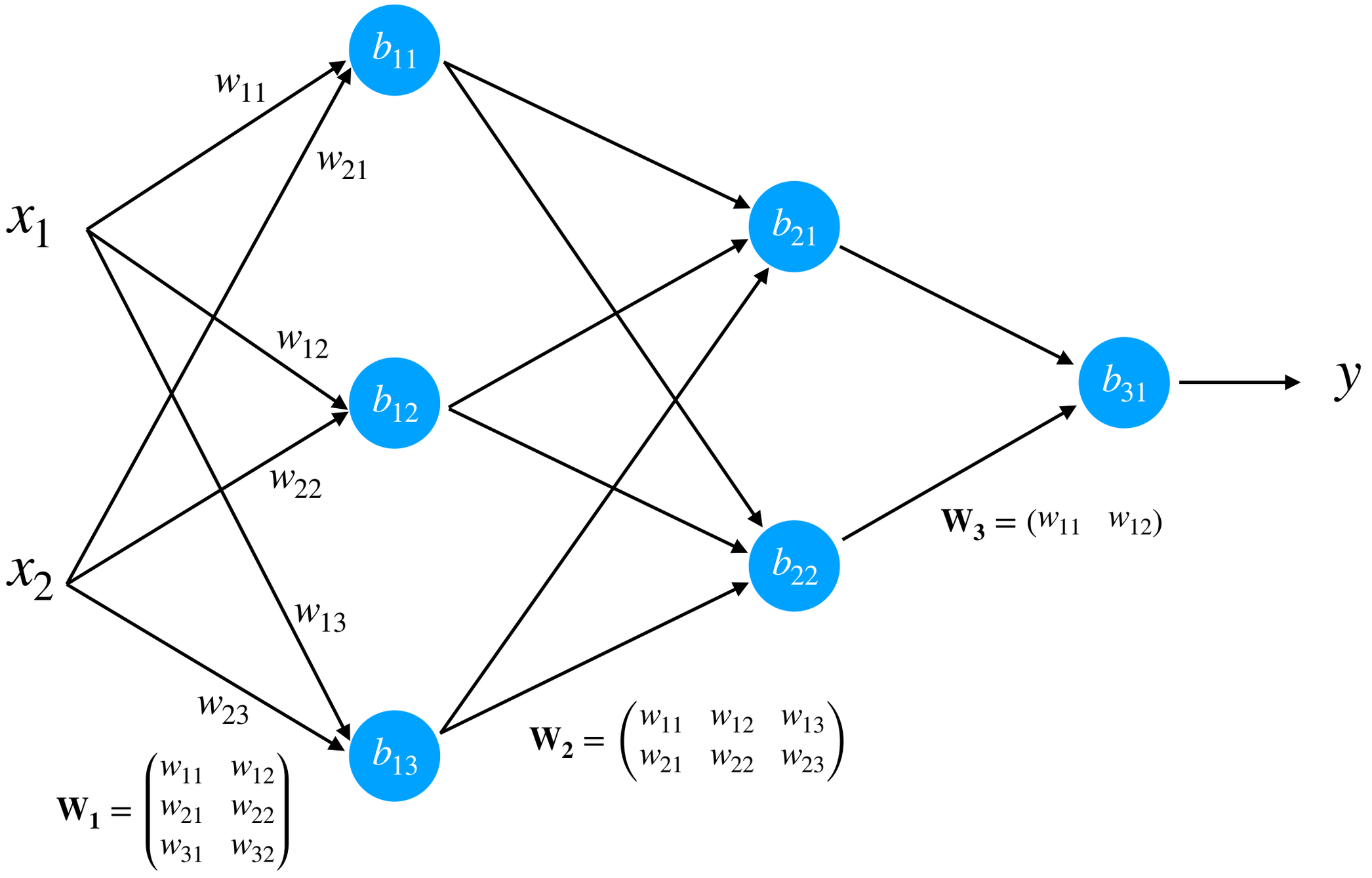
Structure of a Neural Network



Structure of a Neural Network



Structure of a Neural Network



$$X_0 = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow X_1 = \mathbf{W_1} X_0 + \mathbf{b_1} \longrightarrow X_2 = \mathbf{W_2} X_1 + \mathbf{b_2} \longrightarrow y = \mathbf{W_3} X_2 + \mathbf{b_3}$$

Activation Functions

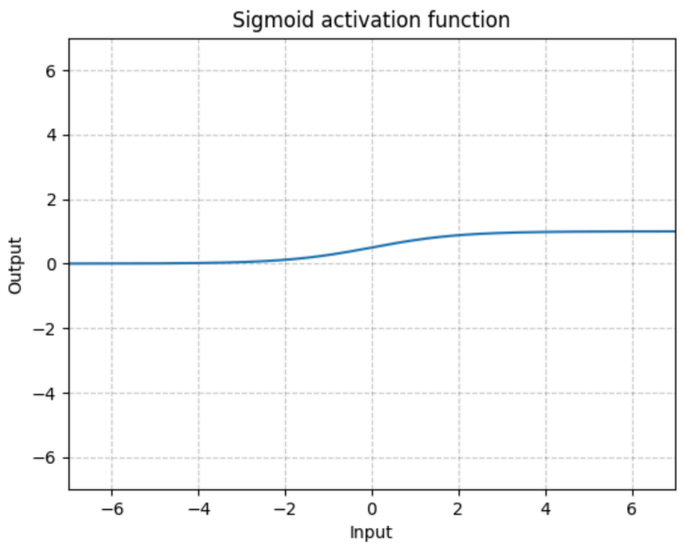
Networks generally contain an activation function

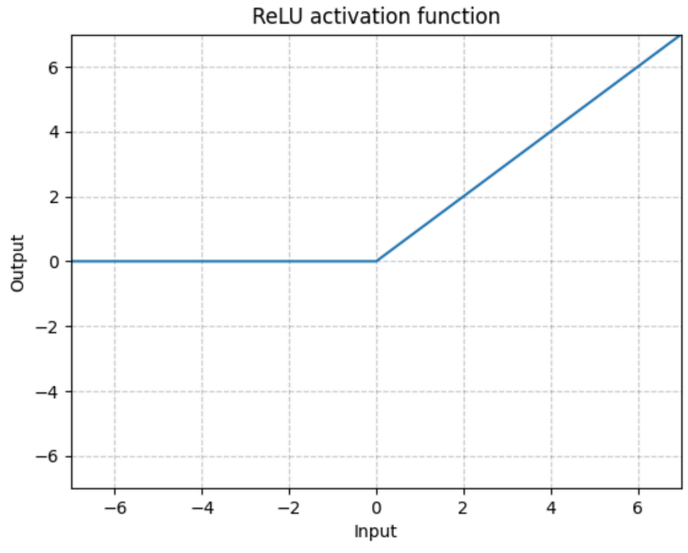
$$f\left(x_1, x_2, \ldots, x_N\right) = x_1 w_1 + x_2 w_2 + \ldots + x_N w_N + b$$

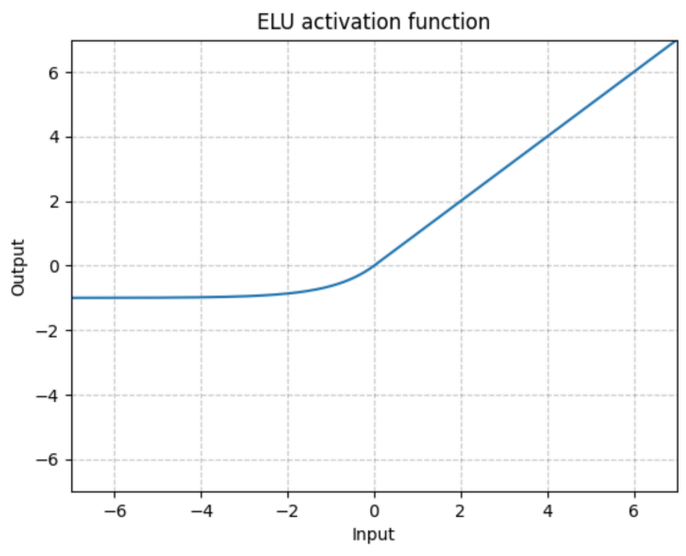
$$\downarrow$$

$$F(\mathbf{X}) = A\left(f\left(x_1, x_2, \ldots, x_N\right)\right)$$

Some Examples

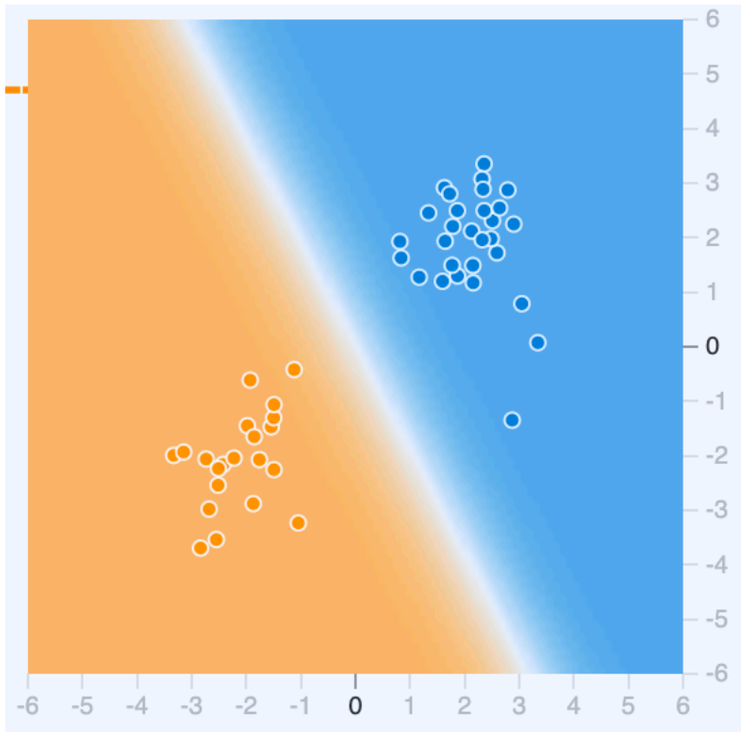






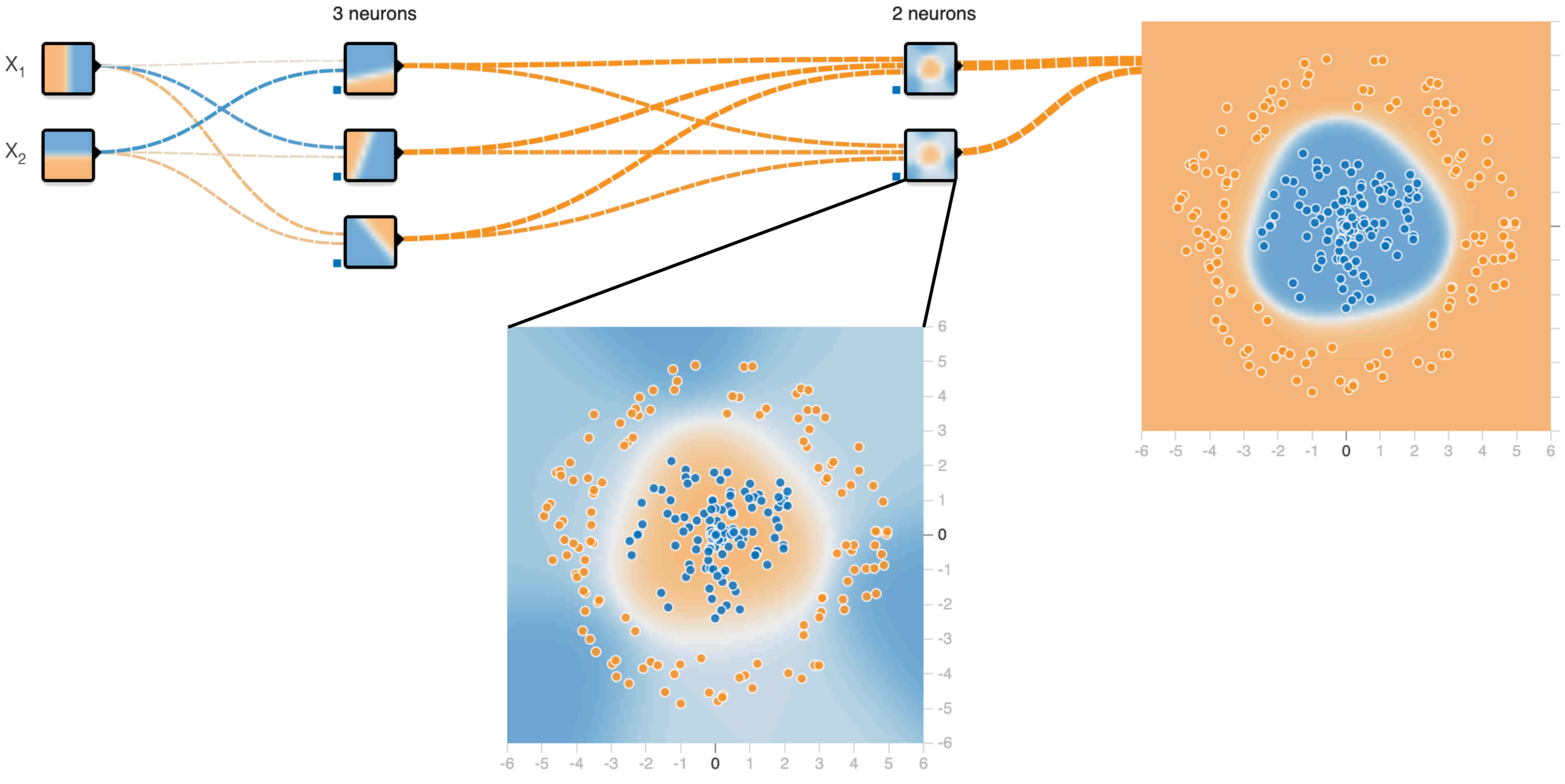
Why?

- Nodes (perceptrons) are linear functions
- They generalize to any number of input dimensions
- They can be ordered sequentially to form more complicated functions



$$f\left(x_1, x_2, \ldots, x_N\right) = x_1 w_1 + x_2 w_2 + \ldots + x_N w_N + b$$

Why?



How To Train Your Neural Network

Loss Functions

- Mean Absolute Error
- Mean Squared Error
- Cross Entropy Loss

$$\mathrm{MAE} = \mathrm{E}\left[\left|x_n - a_n\right|\right]$$

$$\mathrm{MSE} = \mathrm{E}\left[\left(x_n - a_n\right)^2\right]$$

Optimization Algorithms

- Stochastic Gradient Descent (SGD)
- Adam (no relation)
- AdamW
- Adadelta
- Adagrad
- LBFGS

Gradients and Optimization

All optimization algorithms make use of "gradients" to determine how to modify the parameters of the model.

$\sigma \equiv$ the chosen loss function

$\mathbf{P} \equiv$ the set of all parameters in the model

$$\mathbf{P} = \left\{ \mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_K, \mathbf{b}_K \right\} = \left\{ p_1, p_2, \ldots, p_J \right\}$$

$K \equiv$ number of layers in the network

$J \equiv$ number of parameters in the network

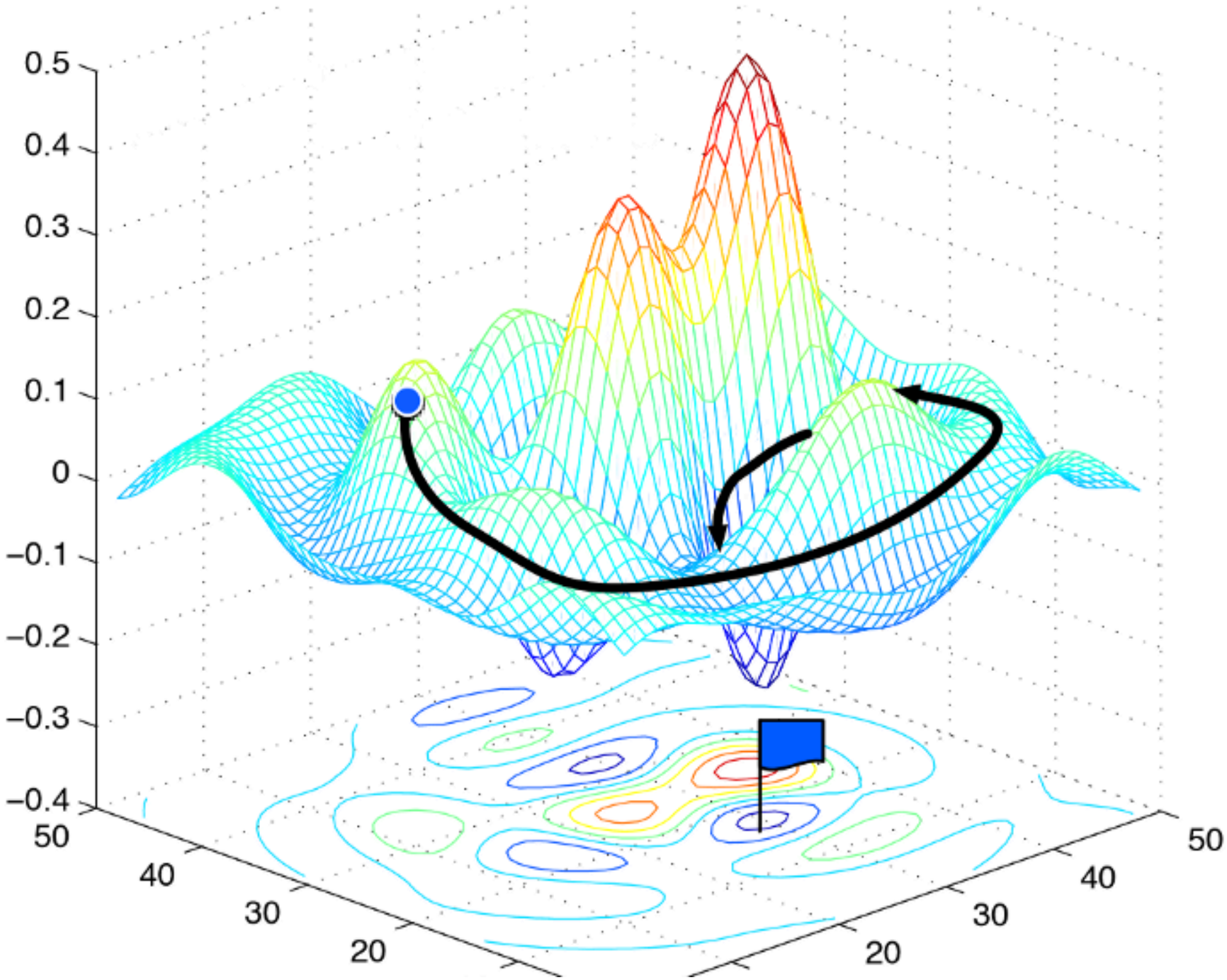
$$\vec{\nabla} F = \left\{ \frac{\partial \sigma}{\partial p_1}, \frac{\partial \sigma}{\partial p_2}, \ldots, \frac{\partial \sigma}{\partial p_J} \right\} \qquad \sigma = \sigma \left(y, a \right)$$

$y \equiv$ output of the network

$a \equiv$ actual correct output (from training data)

Gradients and Optimization

An Example of an Error Surface



How it's Done

PyTorch

- Contains its own computer algebra system that analytically calculates gradients for any network architecture you can think of
- Contains implementations of various optimization algorithms
- Contains implementations of various advanced loss functions that are suitable for complicated models
- Contains pre-programmed methods for constructing different types of neural networks
- Implements all major functionality as optimized GPU operations (10-100x speedup over CPU operations)
- Designed for Python users

PyTorch

Only 23 Lines of code to define a four layer network that combines convolutional layers, fully connected layers, ReLU activation functions, dropout layers and pooling layers.

Without writing any code, PyTorch will automatically analytically calculate gradients.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = self.conv2(x)
    x = F.relu(x)
    x = F.max_pool2d(x, 2)
    x = self.dropout1(x)
    x = torch.flatten(x, 1)
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)
    x = self.fc2(x)
    output = F.log_softmax(x, dim=1)
    return output
```

Why are Analytical Gradients so Important

In the past, you had two options.

1. Manually derive an expression for the partial derivative of the loss function with respect to the parameters, for every parameter in the network.
2. Use finite difference to numerically calculate every gradient.

Option One

- Takes a lot of man hours
- Needs to be redone every time you change your model

Option Two

- Takes no human effort
- Insanely slow to execute

Why are Analytical Gradients so Important

Now you don't have to choose.

Option Three (PyTorch)

- No human intervention necessary to calculate gradients
- Executes almost as fast as manually derived and optimized gradient expressions
    - In some cases this is also faster
- Models can be modified without having to consider the way the modifications alter gradient expressions

Feature Engineering, What and Why

What is it

- Converts raw data into something more readily usable by a neural network

Examples:

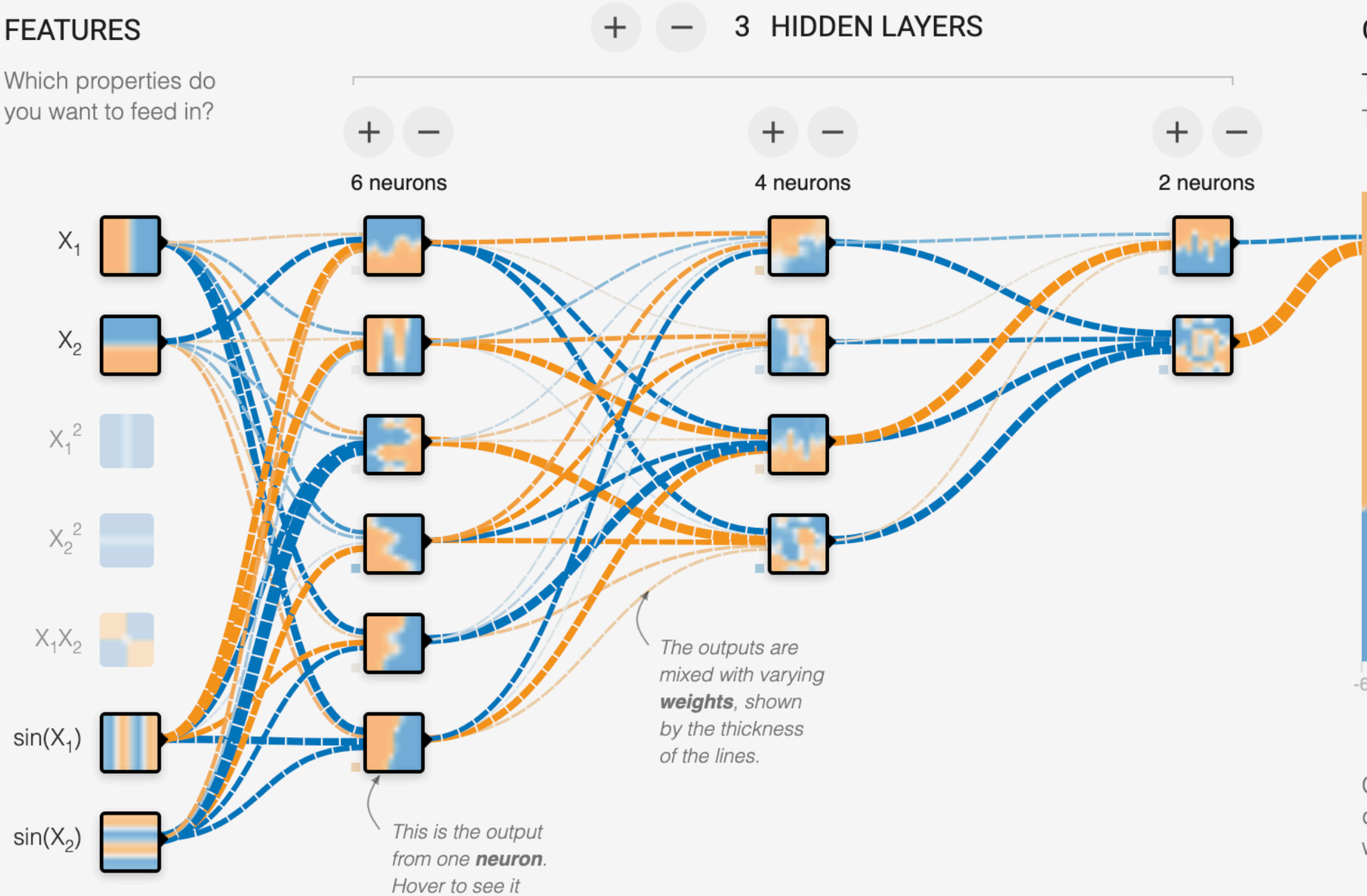
- Downscaling images
- Normalizing values to between zero and one
- Calculating shape moments

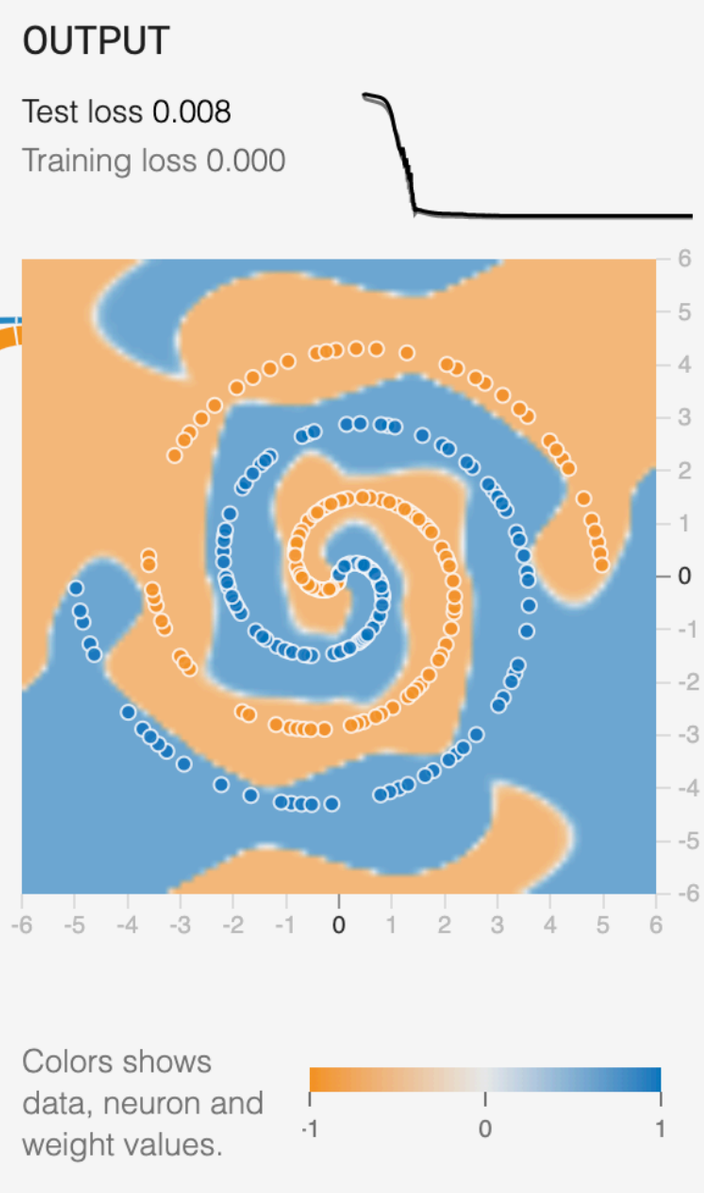
Why do you do it

- Neural networks generally have a fixed number of inputs
- Real world data is not always fixed in size
- The range and variance of a dataset can be very high (not good for neural networks)

Feature Engineering, A Simple Example

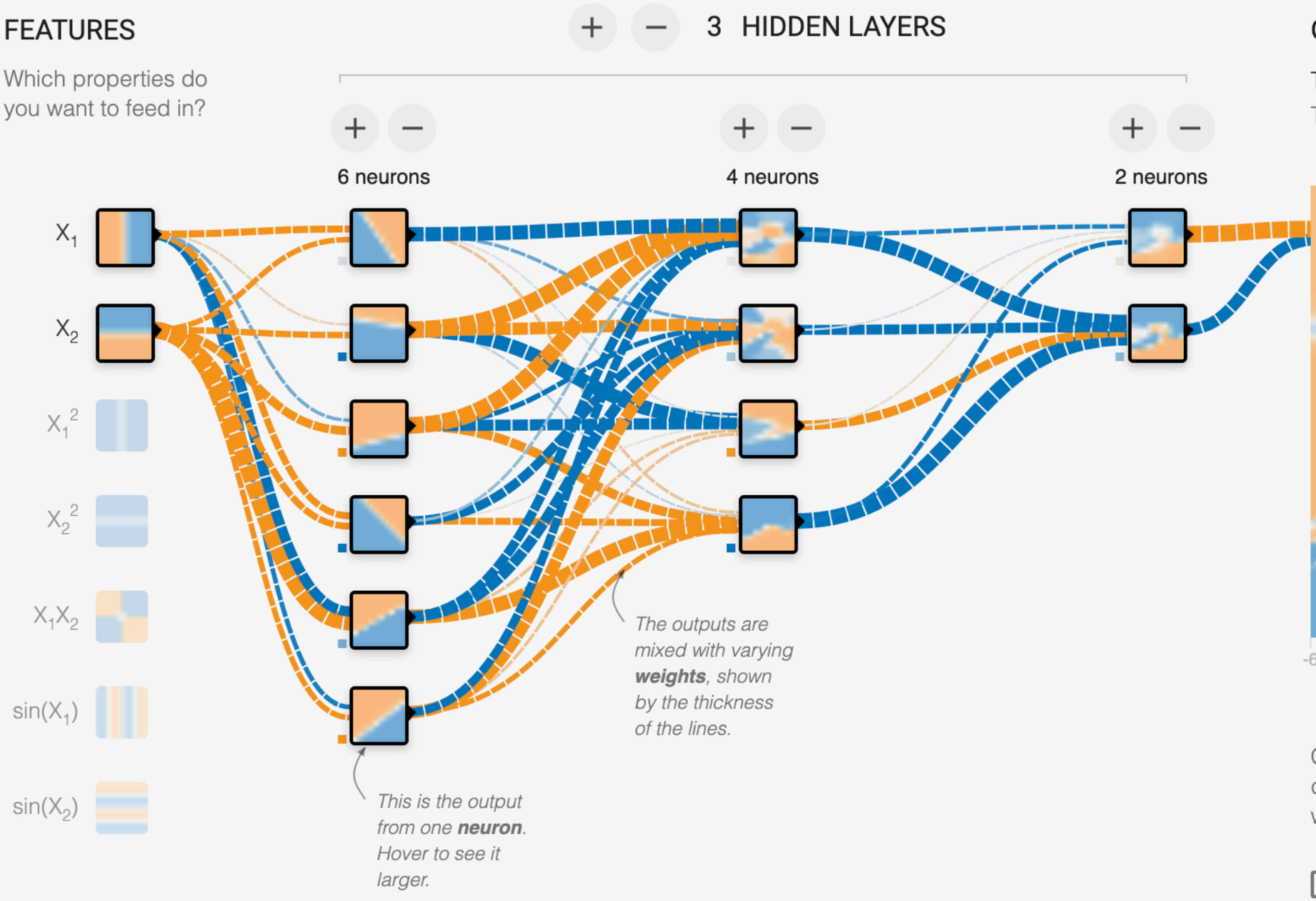
With Features

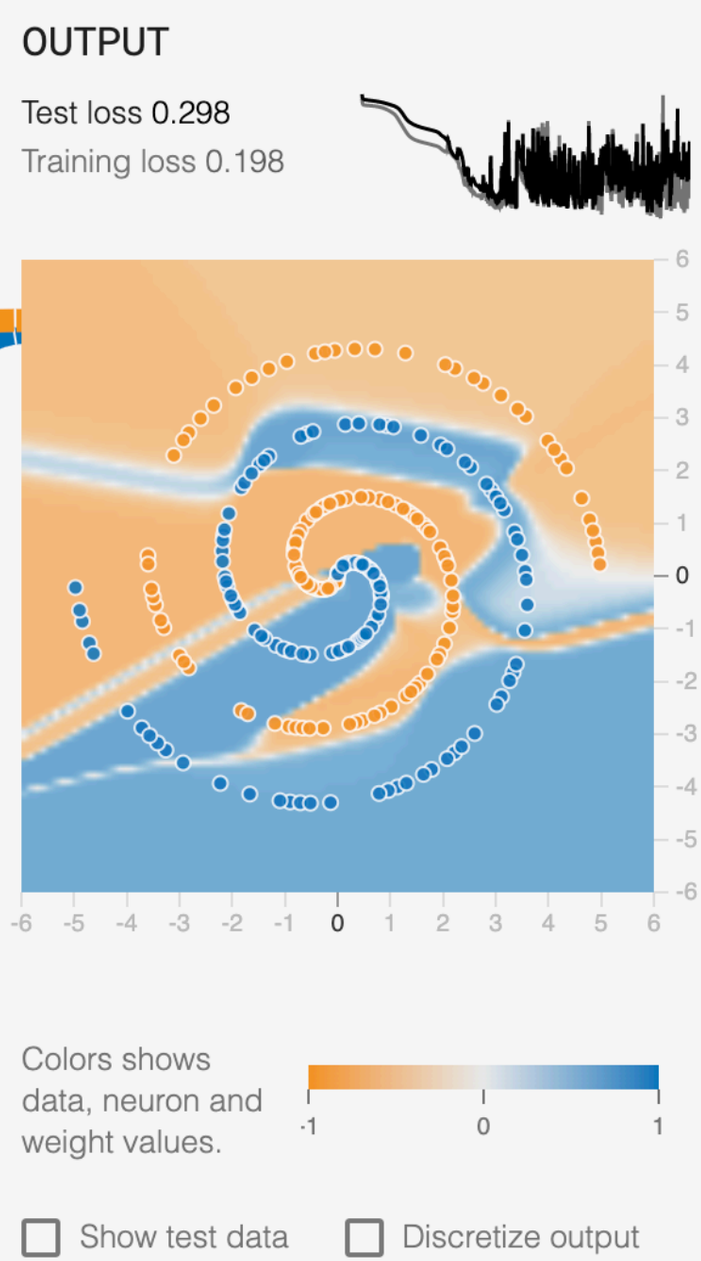




Feature Engineering, A Simple Example

Without Features





Features: A More Advanced Example

- **Describe the atomic neighbor list as a fixed length vector (necessary for a neural network)**
- **Have shown excellent promise as input data for neural network potentials**
- **Based on the local monopole, dipole, quadrupole and n-pole moments formed by atoms in the neighbor list**
- **The number of parameters varies and is controlled by special "hyper-parameters"**

The Algorithm [1]

$$G_i^{(m)} = \sum_{j,\,k} P_m(\cos\theta_{ijk}) f(r_{ij}) f(r_{ik})$$

$$f(r) = e^{(r - r_0)^2/\sigma^2} f_c(r)$$

$$f_c(r) = \begin{cases} \dfrac{(r - r_c)^4}{d^4 + (r - r_c)^4} & r < r_c \\ 0 & r \geq r_c \end{cases}$$

Optional

$$\hat{G}_i^{(m)} = \ln\left(\frac{G_{i\,(original)}^{(m)}}{r_n^2} + \frac{1}{2}\right)$$

Generating a Feature Set: Hyper-Parameters

Interaction cutoff radius ⟶ $\dfrac{(r-\underline{r_c})^4}{d^4+(r-\underline{r_c})^4}$

Interaction truncation distance ⟶ $\dfrac{(r-r_c)^4}{\underline{d}^4+(r-r_c)^4}$

Sigma (gaussian width) ⟶ $e^{(r-r_0)^2/\underline{\sigma}^2}$

Legendre Polynomial orders ⟶ $\underline{P_m}\left(\cos\theta_{ijk}\right)$

Gaussian center locations ⟶ $e^{(r-\underline{r_0})^2/\sigma^2}$

$$number\ of\ inputs = number\ of\ r_0\ \cdot\ number\ of\ P_m$$

How do I Engineer Features?

- There isn't a prescriptive way to do this.
- It's usually a combination of intuition and guess and check.
- Information theory related metrics can be used to predict how well features will perform in a neural network.
    - They don't help you decide what those features should be, just which ones you should use

How do I Engineer Features?

Feature Scoring Metrics

$$Merit\ S_k = \frac{k\overline{\rho_{fo}}}{\sqrt{k + k\left(k-1\right)\overline{\rho_{ff}}}}$$

$k \equiv number\ of\ features$

$\overline{\rho_{ff}} \equiv mean\ feature\ -\ feature\ correlation$

$\overline{\rho_{fo}} \equiv mean\ feature\ -\ output\ correlation$

How do I Engineer Features?

Feature Scoring Metrics

$$Merit\ S_k = \frac{k\overline{\rho_{fo}}}{\sqrt{k + k\left(k-1\right)\overline{\rho_{ff}}}}$$

$k \equiv number\ of\ features$

$\overline{\rho_{ff}} \equiv mean\ feature\ -\ feature\ correlation$

$\overline{\rho_{fe}} \equiv mean\ feature\ -\ output\ correlation$

Correlations Can Be Calculated With Many Different Methods

$X \equiv$ feature $\qquad$ $Y \equiv$ output

$$\rho_{fo} = I\left(X;Y\right) = \sum_{x\in X}\sum_{y\in Y} p_{XY}\left(x,\,y\right)\log\left(\frac{p_{XY}\left(x,\,y\right)}{p_x\left(x\right)p_y\left(y\right)}\right)$$

← **Slow but very effective**

(Also a numerical nightmare)

How do I Engineer Features?

Feature Scoring Metrics

$$Merit\ S_k = \frac{k\overline{\rho_{fo}}}{\sqrt{k + k\left(k-1\right)\overline{\rho_{ff}}}}$$

$k \equiv number\ of\ features$

$\overline{\rho_{ff}} \equiv mean\ feature\ -\ feature\ correlation$

$\overline{\rho_{fe}} \equiv mean\ feature\ -\ output\ correlation$

Correlations Can Be Calculated With Many Different Methods

$X \equiv$ feature $\qquad$ $Y \equiv$ output

$$\rho_{fo} = I\left(X;Y\right) = \sum_{x\in X}\sum_{y\in Y} p_{XY}\left(x,y\right)\log\left(\frac{p_{XY}\left(x,y\right)}{p_x\left(x\right)p_y\left(y\right)}\right)$$

← **Slow but very effective**

(Also a numerical nightmare)

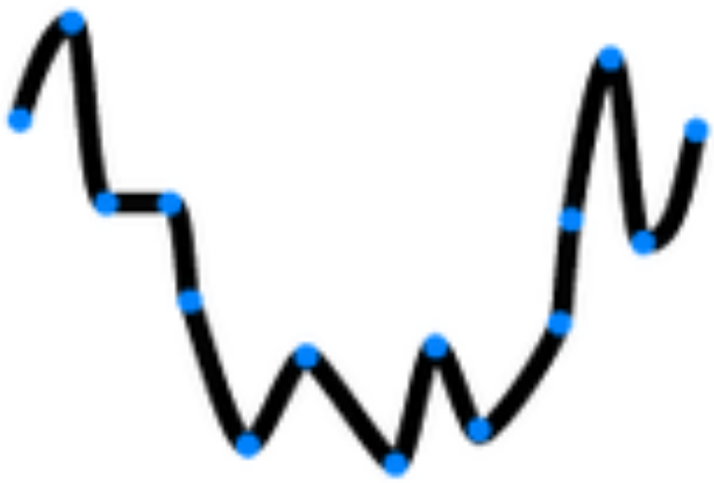
$$\rho_{fo} = \rho(X,Y) = \frac{\mathrm{E}\left[\left(X-\mu_X\right)\left(Y-\mu_Y\right)\right]}{\sigma_X\sigma_Y}$$

← **Fast but limited accuracy**

(Only recognizes linear relationships)

Overfitting (why you have a validation dataset)

Overfit



Correctly Fit

- Periodically calculating the validation error will help you determine whether or not your model is overfit

- Datasets are usually split into 90-95% training data and 5-10% validation data

- You **never** calculate gradients using validation data and never use it to train the network

Overfitting, how to combat it

- Keep the network as simple as possible
- Apply L2 Regularization
- Use dropout layers
- If possible, slightly modify your dataset with each training iteration

L2 Regularization

Before

$$\sigma = \sigma\left(y, a\right)$$

After

$$\sigma = \sigma\left(y, a\right) + \gamma \mathrm{E}\left[P\right]$$

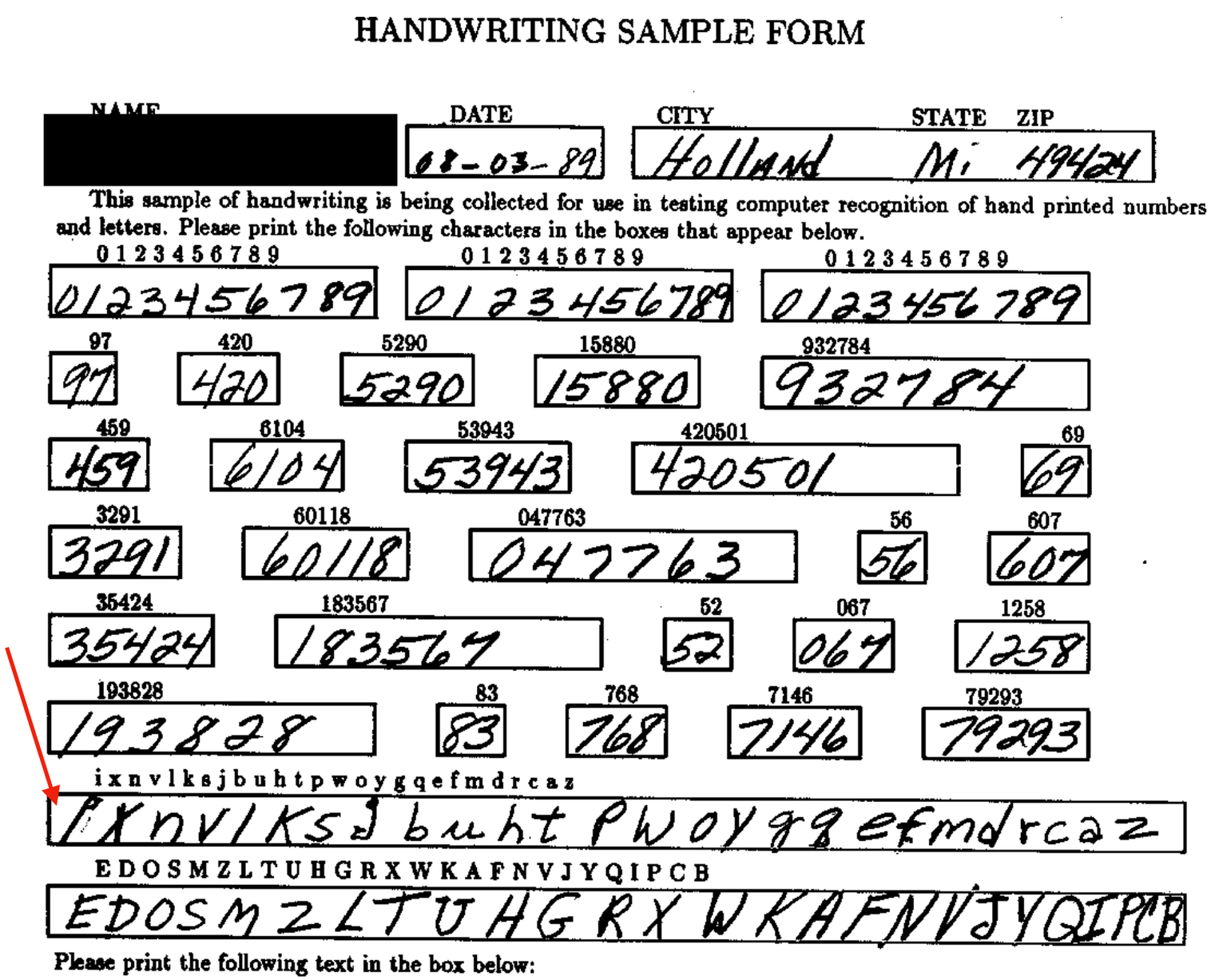
$\gamma \equiv$ Regularization Coefficient

Image Classification

NIST Special Database 19

Handwritten Characters Collected Via Census Forms (and one high school)

- ~800,000 individual characters
- 62 character types (a-z A-Z 0-9)
- 128x128 greyscale images
- Already classified by NIST
- A lot of horrendous handwriting
- The EMNIST database is a subset of this database (in case you've heard of it)



HANDWRITING SAMPLE FORM

NAME [redacted] DATE 08-03-89 CITY Holland STATE Mi ZIP 49424

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9

97 | 420 | 5290 | 15880 | 932784

459 | 6104 | 53943 | 420501 | 69

3291 | 60118 | 047763 | 56 | 607

35424 | 183567 | 52 | 067 | 1258

193828 | 83 | 768 | 7146 | 79293

i x n v l k s j b u h t p w o y g q e f m d r c a z

E D O S M Z L T U H G R X W K A F N V J Y Q I P C B

Please print the following text in the box below:

Why Image Classification is Hard

- A 128x128 image contains 16,384 data points (at minimum)
    - The number of parameters in a neural network is often proportional to the number of inputs squared (268,435,456 in this case)
        - That is too large for all but the most expensive clusters (and completely impractical even for them)
- Deriving features from images is not a simple task

Convolutional Layers: The Solution

Sliding Kernel

0	1	2
2	2	0
0	1	2

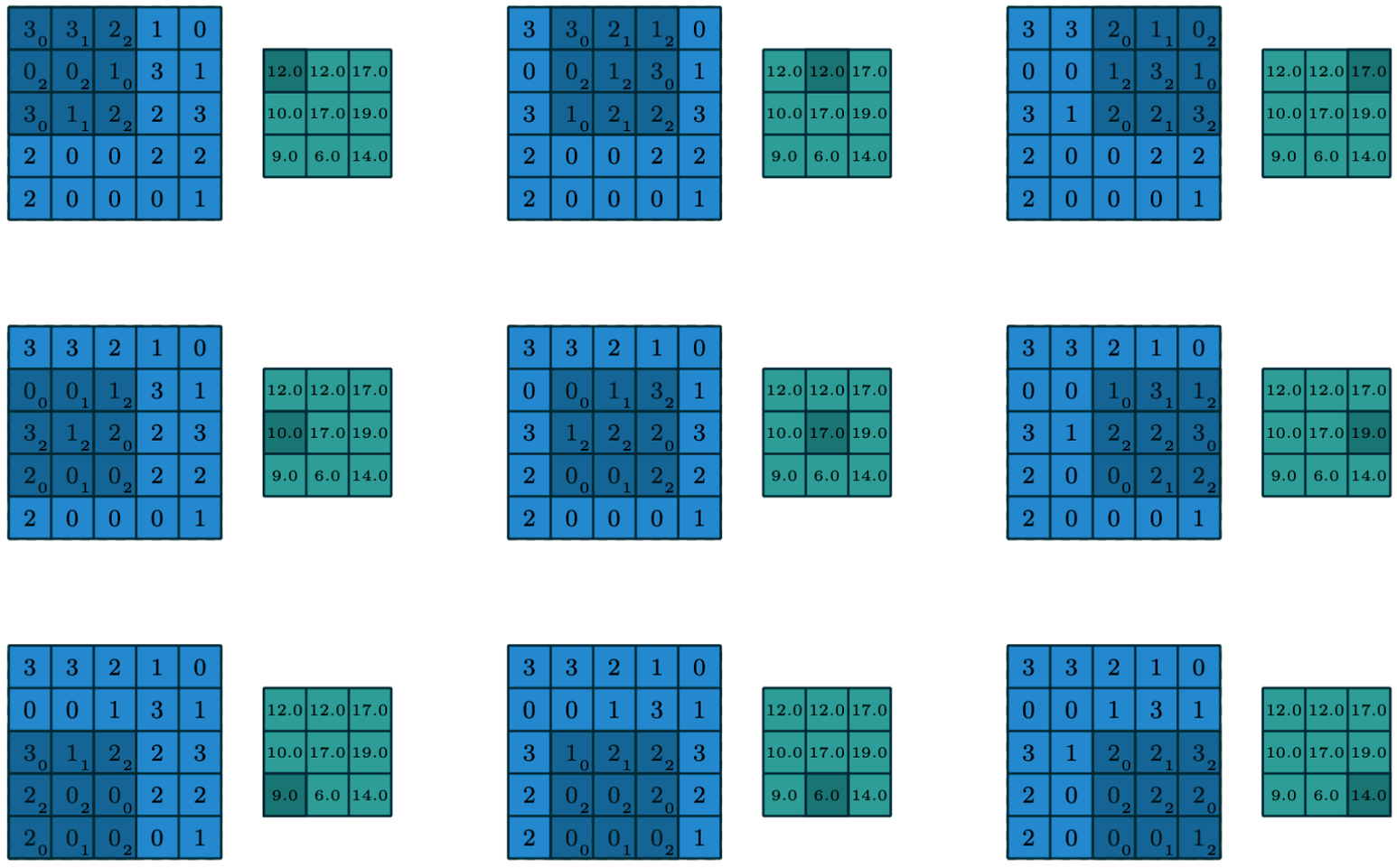
An Excellent Explanation: https://arxiv.org/pdf/1603.07285.pdf

Convolutional Layers: The Solution

Sliding Kernel

0	1	2
2	2	0
0	1	2

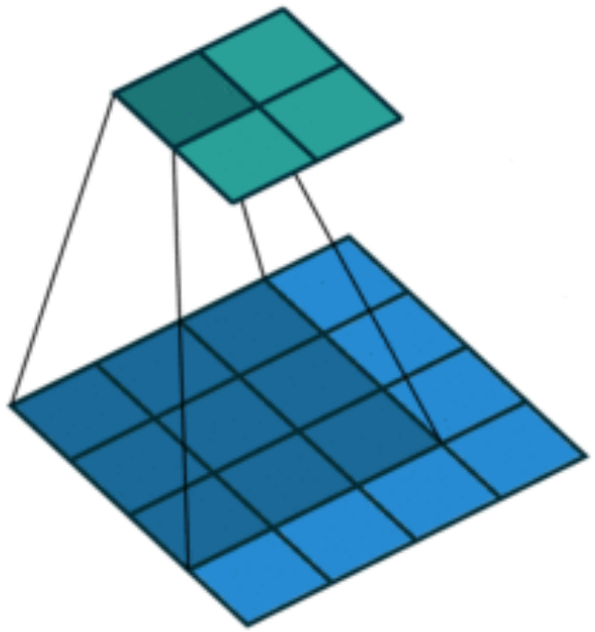
Scanned Over an Image



An Excellent Explanation: https://arxiv.org/pdf/1603.07285.pdf

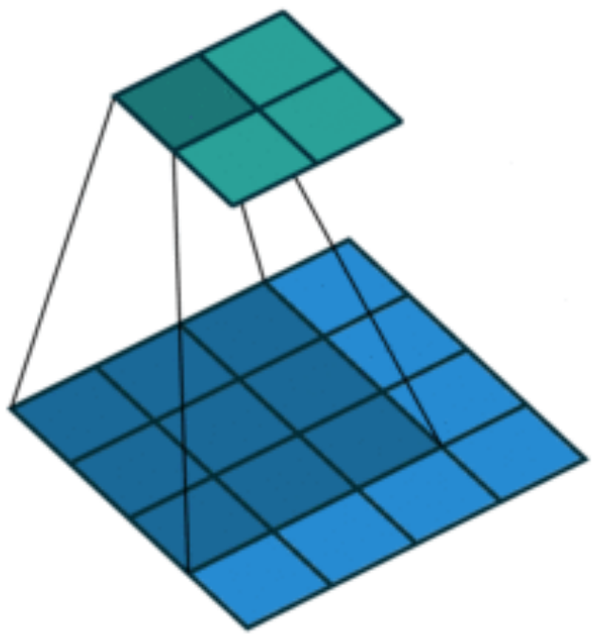
Convolutional Layers

Basic

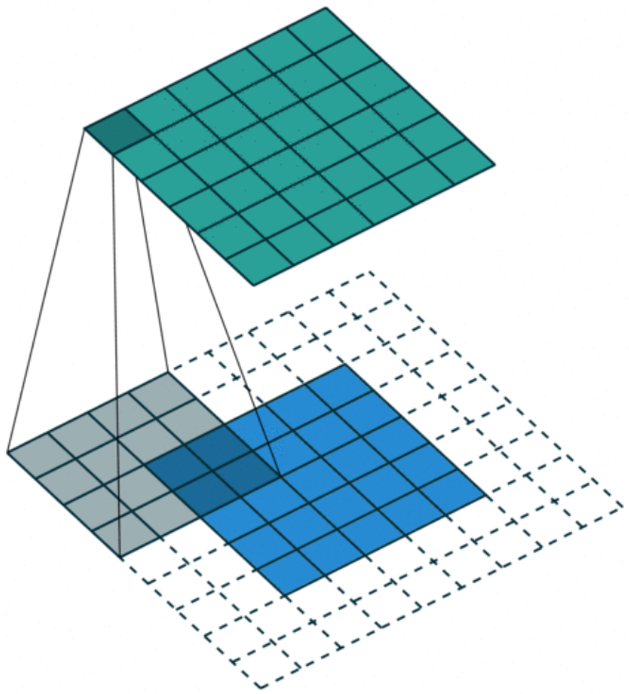


Convolutional Layers

Basic

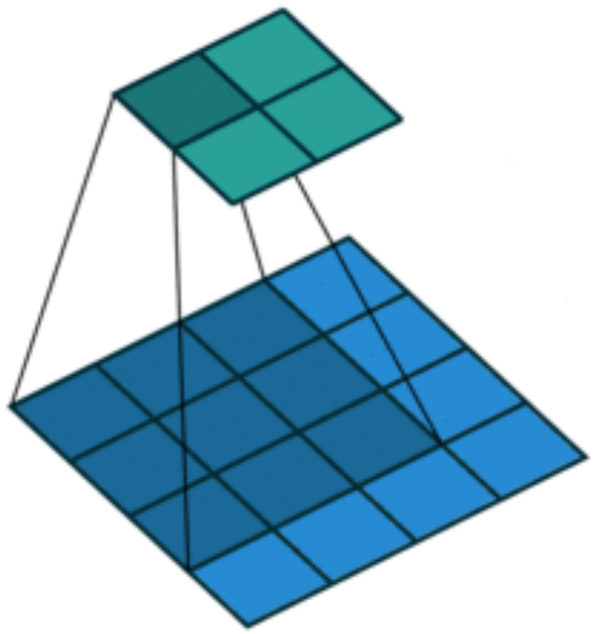


Padded

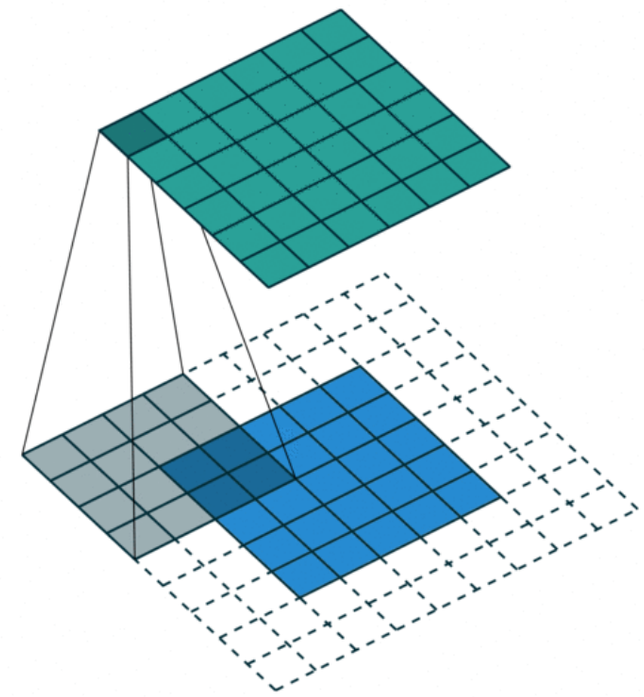


Convolutional Layers

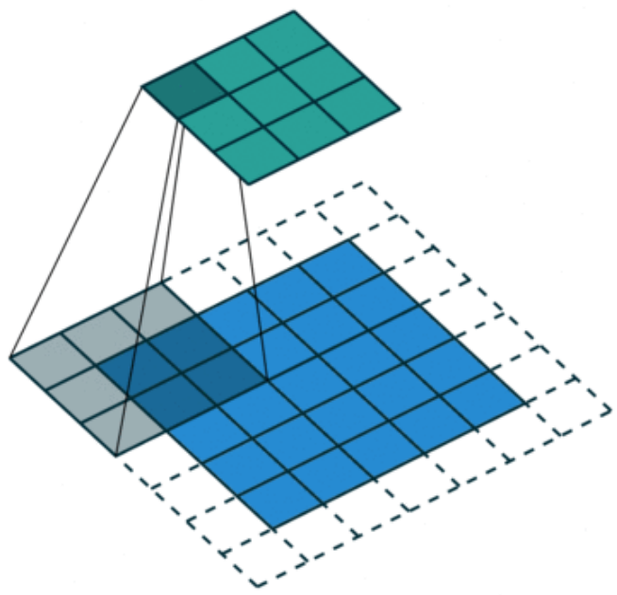
Basic



Padded



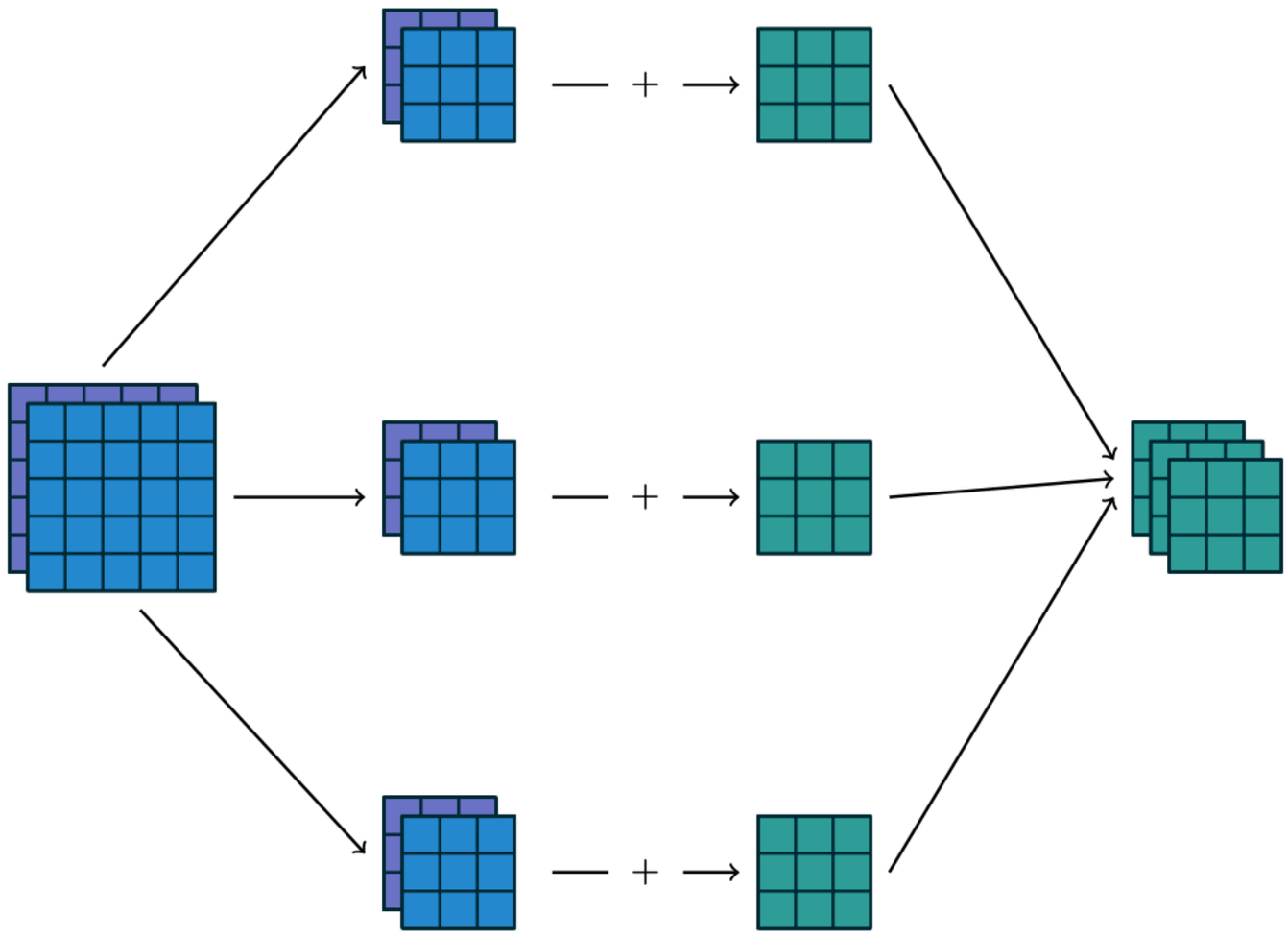
Padded and Strided



Convolution Channels (or Maps)

2 Input Channels

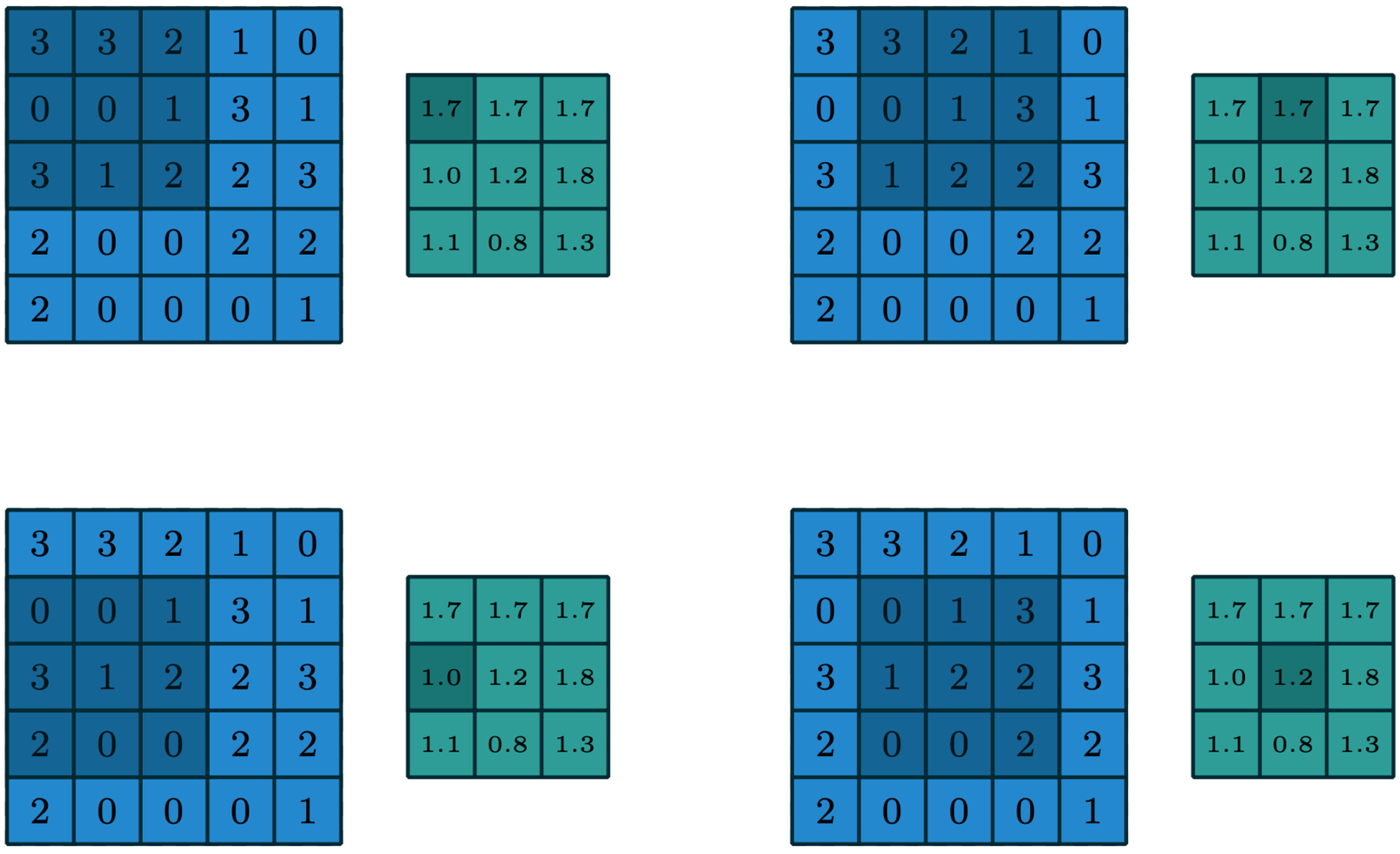
3 Output Channels



Pooling Layers

Used to Reduce the Size of Subsequent Layers

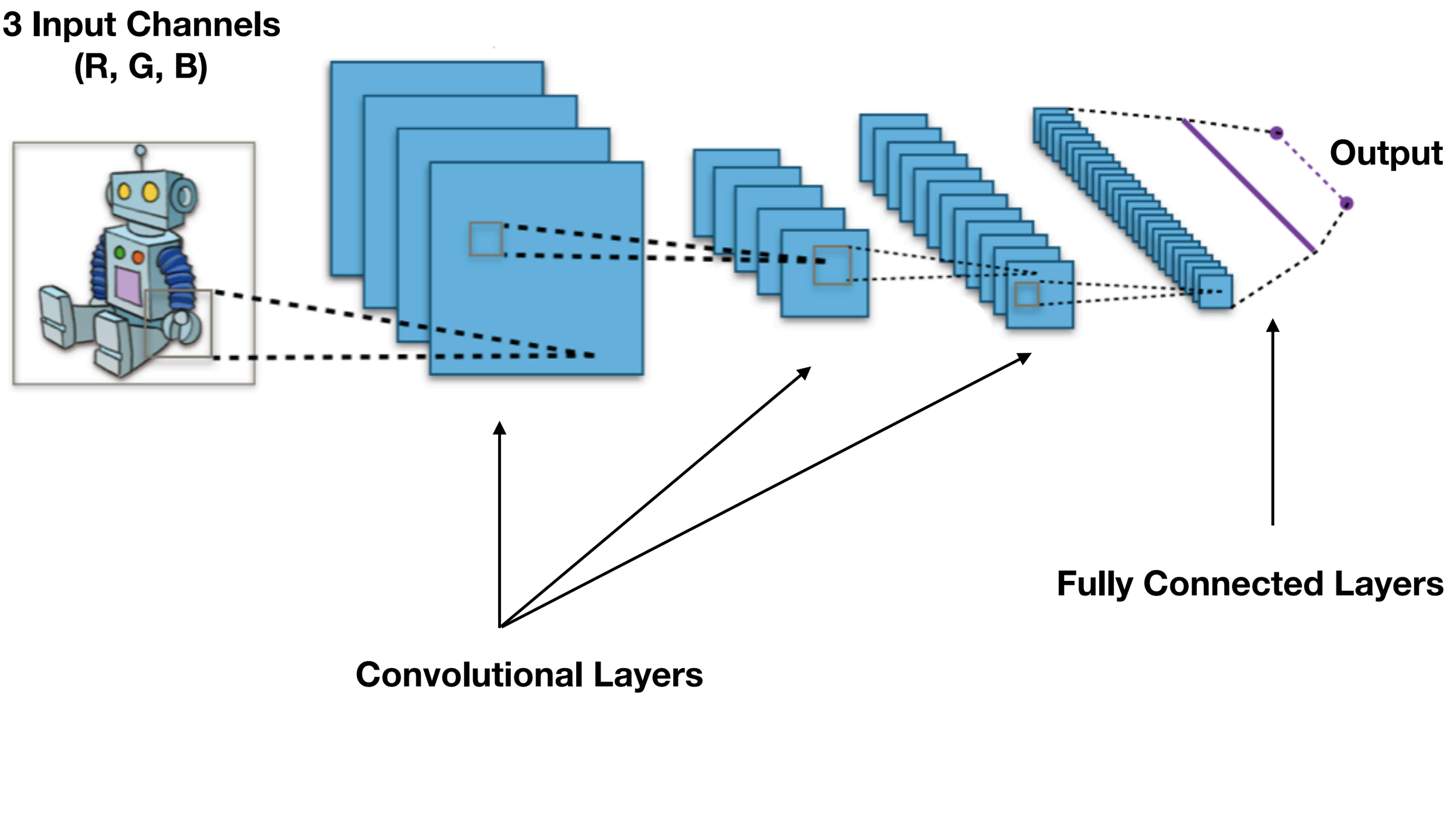
Average Pooling



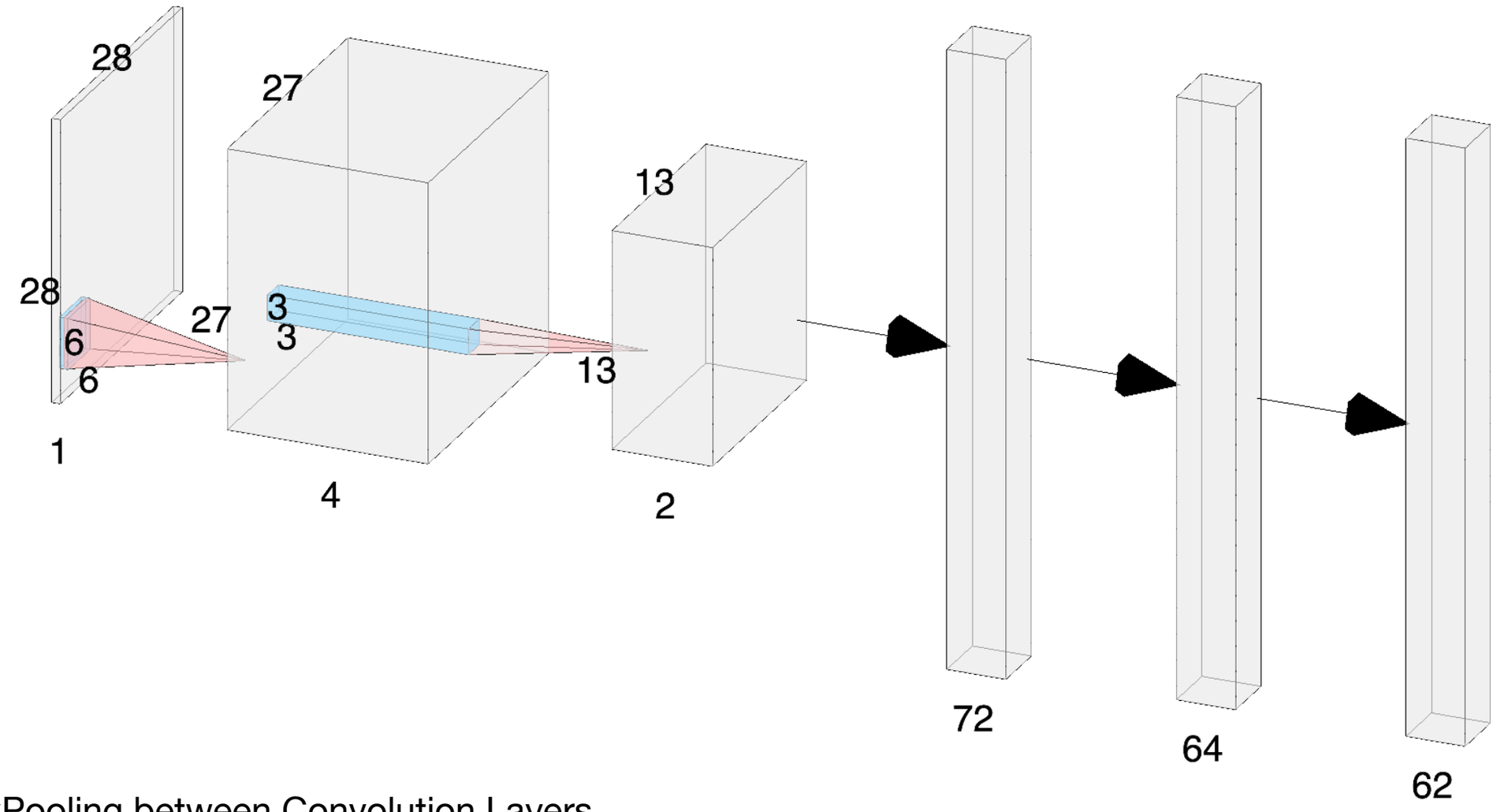
Why it Works

- Convolutional layers have a degree of translational and scale invariance.
    - The kernel for each channel is computed over every chunk of pixels in the image, regardless of size
    - They can process a 128x128 image without having anywhere near as many parameters as a fully connected neural network
    - Pooling layers do not contain trainable parameters. They can be used to reduce larger images down to the size that a network was trained on

General Structure of a CNN



My Work so Far



- MaxPooling between Convolution Layers
- LeakyReLU Activation Function
- 80% Validation Accuracy on the Dataset with ~4 minutes of training
- L2 Regularization Coefficient ~0.045
- Dropout Layer