Neural Networks Workshop

Tony Allen

Department of Mathematics
Purdue University

July 1, 2019

Helpful Resources

- Goodfellow, Bengio, Courville's *Deep Learning* https://www.deeplearningbook.org/
- Francis Chollet's *Deep Learning with Python* https://github.com/fchollet/deep-learning-with-python-notebooks
- Dr. Buzzard MA598 Course Notes https://www.math.purdue.edu/~buzzard/MA598-Spring2019/
- Nick Winovich's SIAM@Purdue TensorFlow Workshop https://www.math.purdue.edu/~nwinovic/workshop.html

What is Machine Learning?

Machine Learning shifts the paradigm from programming for answers to programming to discover rules.

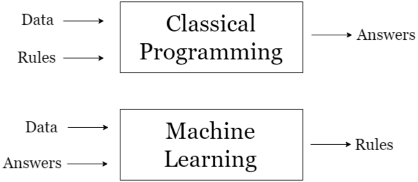
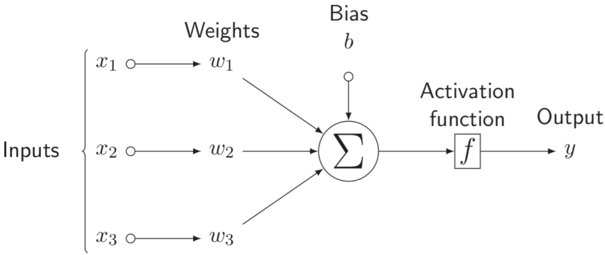


Diagram adapted from Francis Chollet's *Deep Learning with Python*

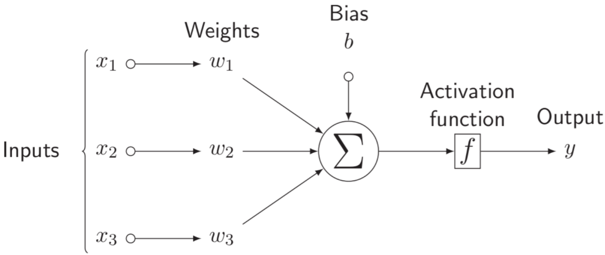
Artificial Neuron: The building blocks of a neural network



Mathematically,
$$
\begin{aligned}
y &= f(w_1x_1 + w_2x_2 + w_3x_3 + b) \\
&= f(\mathbf{w}^T\mathbf{x} + b)
\end{aligned}
$$

Diagram from Nick Winovich

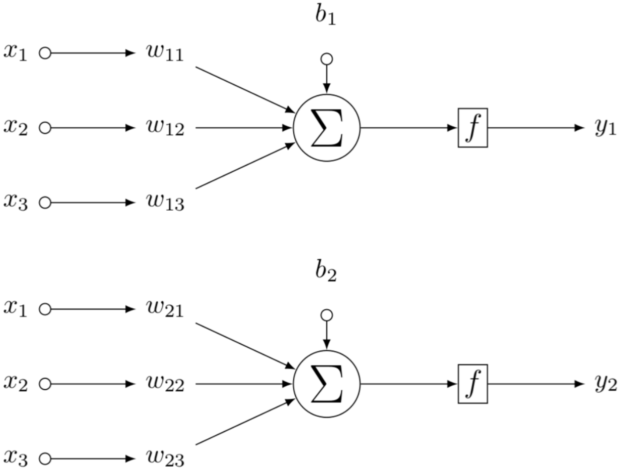
Artificial Neuron: The building blocks of a neural network



Mathematically,
$$y = f(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$
$$= f(\mathbf{w}^T \mathbf{x} + b)$$

Diagram from Nick Winovich

Neural Networks



Neural Networks

We can combine the corresponding equations

$$y_1 = f(\mathbf{w_1}^T \mathbf{x} + b_1)$$
$$y_2 = f(\mathbf{w_2}^T \mathbf{x} + b_2)$$

into one matrix-vector product equation

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

If we have N inputs and M outputs, then W is a **Dense** $M \times N$ matrix.

(for the picky: f is applied element-wise)

Dense (Fully Connected) Layer

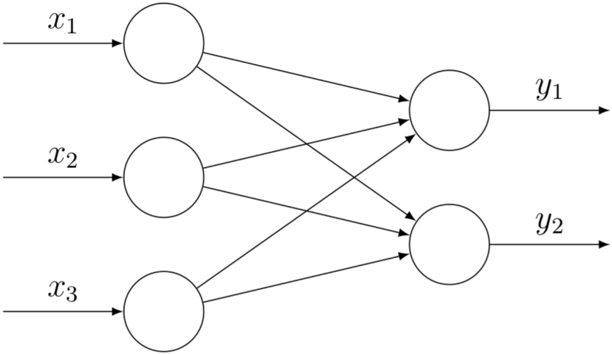
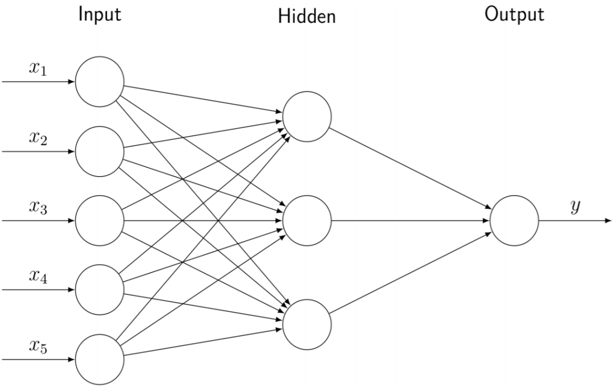


Diagram from Nick Winovich

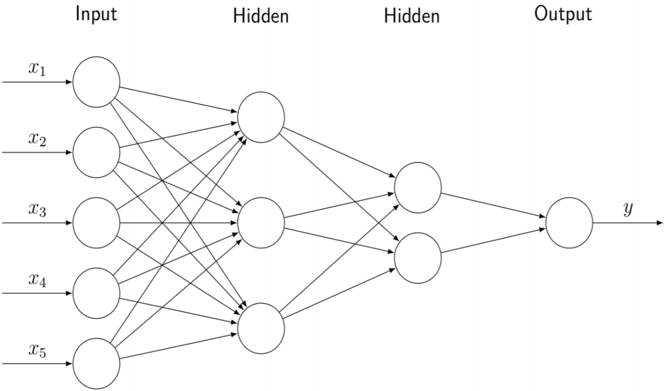
Network Depth



$$y = f_2(\mathbf{W}_2(f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2)$$

Diagram from Nick Winovich

Network Depth



$$y = f_3(\mathbf{W}_3(f_2(\mathbf{W}_2(f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2)) + \mathbf{b}_3)$$

Diagram from Nick Winovich

A Word on Activation Functions

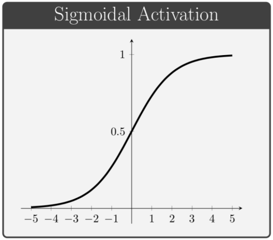
Activation functions are a fundamental component of network architecture; they allow for non-linear modeling capacity, and control the gradient flow that guide training.

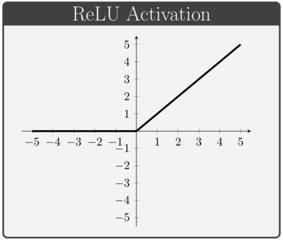
Sigmoidal Unit

$$f(x) = \frac{1}{1 + \exp(-x)}$$

Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$



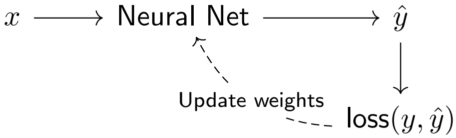


Figures from Nick Winovich

Optimization (How to learn)

Goal: Learn weights so the network gives desired output

Everything today will be Supervised Learning:



Adjust weights $w_{i,j}$ to minimize the loss

Gradient Descent

From calculus: The greatest decrease in a function is in the direction opposite of the gradient.

Let θ be all the parameters (weights and biases) and E be total loss over all data. Then iteratively apply a method called Gradient Descent:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla E_{\theta_k}$$

However, computing gradient of loss over all data can be expensive. So instead compute it over random subsets of data (batches). This leads to Stochastic Gradient Descent algorithms.

Gradient Descent

From calculus: The greatest decrease in a function is in the direction opposite of the gradient.

Let θ be all the parameters (weights and biases) and E be total loss over all data. Then iteratively apply a method called Gradient Descent:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla E_{\theta_k}$$

However, computing gradient of loss over all data can be expensive. So instead compute it over random subsets of data (batches). This leads to Stochastic Gradient Descent algorithms.

PURDUE UNIVERSITY

Gradient Descent

From calculus: The greatest decrease in a function is in the direction opposite of the gradient.

Let θ be all the parameters (weights and biases) and E be total loss over all data. Then iteratively apply a method called <u>Gradient Descent</u>:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla E_{\theta_k}$$

However, computing gradient of loss over all data can be expensive. So instead compute it over random subsets of data (batches). This leads to <u>Stochastic Gradient Descent</u> algorithms.

Back Propagation

How do we compute the gradient, i.e. $\frac{\partial E}{\partial w_{ij}}$?

The answer: Chain Rule!

Back Propagation

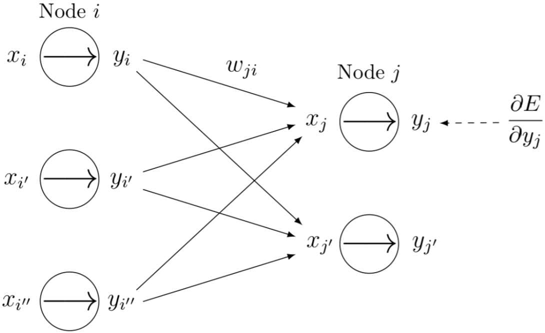


Diagram from Nick Winovich

Back Propagation

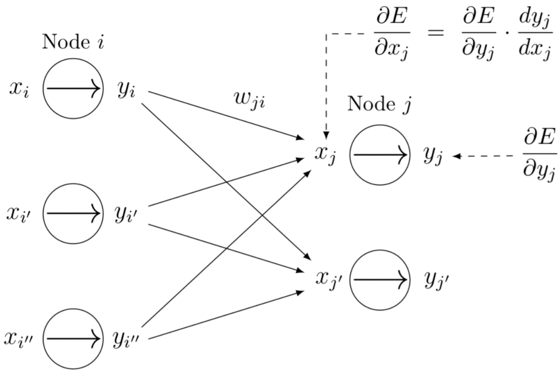


Diagram from Nick Winovich

Back Propagation

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \cdot y_i$$

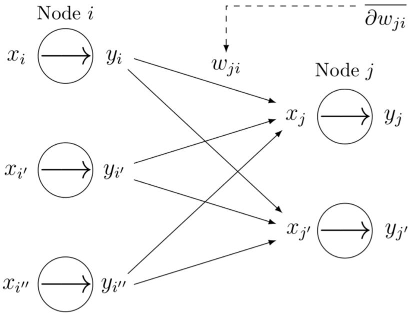


Diagram from Nick Winovich

Back Propagation

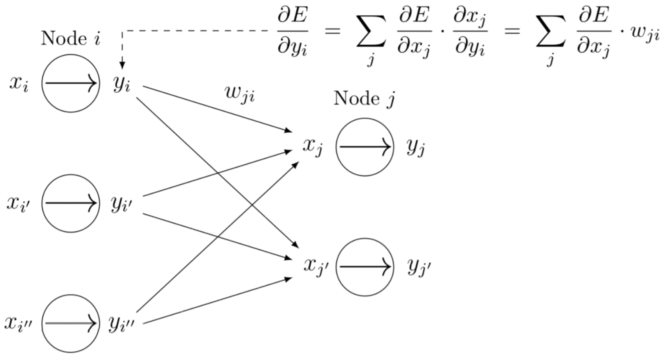


Diagram from Nick Winovich

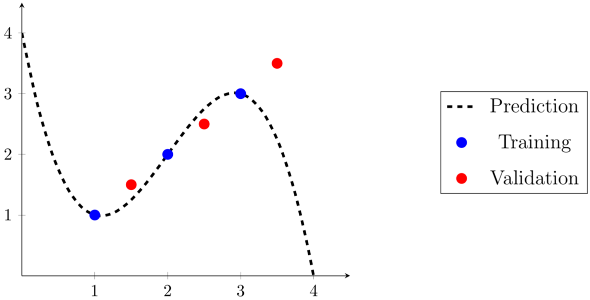
Let's actually do something! (Exercise 1)

The Notebook to follow along can be found on the Workshop homepage:
`https://engineering.purdue.edu/ChanGroup/MLworkshop2019.html`

PURDUE UNIVERSITY

Overfitting

In some cases, a network can learn too much. That is, it can learn to perform well on the training data, but fail to generalize to testing data. Solutions include **Regularization** and **Dropout**.



Regularization and Dropout

Regularization adds a penalty for large weights to the loss function. Commonly, we use

- L_1 norm, which encourages sparsity
- L_2 norm, which encourages small weights

$$\text{loss} = \text{loss} + \lambda \|\theta\|_1 \text{ (or } \|\theta\|_2)$$

Dropout temporarily ignoring random nodes (with fixed probability p) during each training iteration. Ensures no individual node dominates.

Exercise 2

In this exercise, we will try to improve the previous model by adding dropout. You should use the Keras Documentation (`https://keras.io/`) to create a network with at least 2 hidden layers that use dropout. Plot the training loss and print the test accuracy, and compare to the previous model.

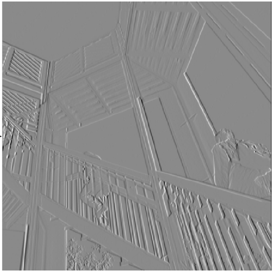
Convolutional Neural Networks (CNNs)

CNNs are useful when the data is spatially structured (e.g. images).

The key concept behind CNNs is that of kernels/filters. These are used in hand-crafted feature detection.

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel

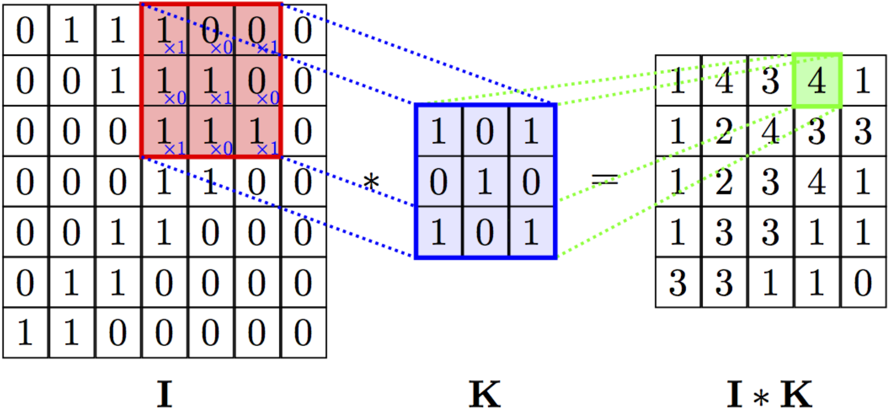


What are good, distinguishing features? How do we mathematically extract such features?

---

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Convolution



https://github.com/PetarV-/TikZ/tree/master

Pop Quiz

Let $I = \begin{bmatrix} 1 & 2 & 0 & 1 & 3 \\ 2 & 0 & 1 & 4 & 0 \\ 7 & 0 & 9 & 5 & 5 \\ 8 & 5 & 2 & 6 & 0 \\ 8 & 0 & 0 & 1 & 4 \end{bmatrix}$ and $K = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}$.

1. What is $(I * K)(1,1)$?
2. What is the size of $I * K$?

Matrix View

In practice, we perform a convolution as one large matrix-vector product that does all the work in one go.

$$I = \begin{bmatrix} 1 & 2 & 0 & 1 & 3 \\ 2 & 0 & 1 & 4 & 0 \\ 7 & 0 & 9 & 5 & 5 \\ 8 & 5 & 2 & 6 & 0 \\ 8 & 0 & 0 & 1 & 4 \end{bmatrix} \text{ and } K = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

$$I * K = \begin{bmatrix} 1 & 1 & 0 & \cdots & 2 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 2 & \cdots & 0 \\ & & & \vdots & & & & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 1 & 0 & \cdots & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \\ \vdots \\ 9 \\ \vdots \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 21 \\ 12 \\ 11 \\ \vdots \\ 22 \end{bmatrix}$$

Words: Toeplitz, Sparse

Convolutional Neural Network

Key Ideas of a CNN:

1. Instead of expensive dense matrix-vector products, do convolutions
2. Everything else stays the same (activation functions, training, etc.)

CNNs scale very well to large images because of their sparse connections and natural space invariance.

Of course I have skipped some details, so let me touch on those:

- Stride
- Padding
- Pooling

PURDUE UNIVERSITY

Convolutional Neural Network

Key Ideas of a CNN:

1. Instead of expensive dense matrix-vector products, do convolutions
2. Everything else stays the same (activation functions, training, etc.)

CNNs scale very well to large images because of their sparse connections and natural space invariance.

Of course I have skipped some details, so let me touch on those:

- Stride
- Padding
- Pooling

Stride and Padding

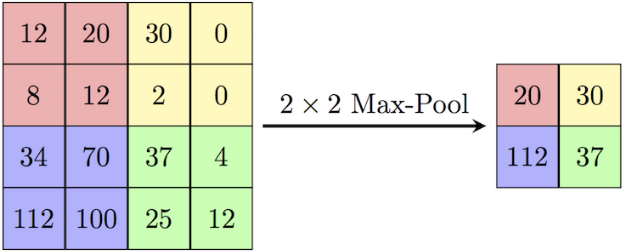
When defining a convolution, we need to specify how fast and to what extent the kernel slides over the image. This is the stride and padding, respectively. Both of these determine the size of the output.

In Keras,

- "strides = 2", determines how many pixels the kernel moves at a time (in this case two)
- "padding = same" puts zeros around the image so that the output is the same size as the input. Called zero-padding.
- "padding = valid" puts zeros in the necessary places so that the convolution stays valid

Max Pooling

Often, we care about the existence of a feature. Max Pooling is one way to reduce dimensionality while keeping information about the existence of a feature.



In my experience, you see this applied after a stride $= 1$ convolution with zero-padding.

https://computersciencewiki.org/index.php/Max-pooling

Exercise 3

In this exercise, we implement a CNN and see how much better it performs on our image classification task.

Fin.