Stanford MEDICINE

Stanford | Center for Health Education

FUNDAMENTALS OF MACHINE LEARNING FOR HEALTHCARE

MODULE 3 - CONCEPTS AND PRINCIPLES OF MACHINE LEARNING IN HEALTHCARE PART 2

LEARNING OBJECTIVES

- Attain basic grasp of the mechanics through which neural networks are trained (i.e. understand the roles of Loss, Gradient Descent, and Backpropagation during the model fitting procedure) and which clinical use cases are best suited for these modeling approaches.
- Learn about common loss functions for network training and the relative differences and advantages
- Attain a comprehensive understanding of the concepts and mechanisms of DNN, CNN, and RNN architectures and begin understanding applications in medicine
- Learn about some of the most common original and important neural networks like AlexNet, VGG, GoogLeNet, ResNet
- Recognize the opportunities and challenges with reinforcement learning in healthcare applications
- Learn about advanced neural network architectures for tasks ranging from text classification to object detection and segmentation

DEEP LEARNING AND NEURAL NETWORKS

Why “deep” learning?

- In most traditional machine learning methods (SVMs, linear regression), the number of parameters is limited to the number of input features
- Deep learning models are machine learning models that organize parameters into hierarchical layers
- Features are multiplied and added together repeatedly, with the outputs from one layer of parameters being fed into the next layer – before a prediction can be made

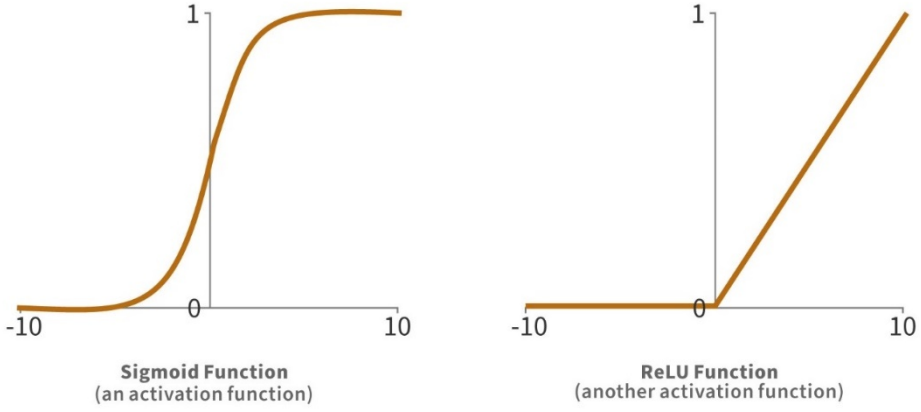
- This increased interaction between features and model parameters increases the complexity of the functions learnable by the model

Parameters and Linear Combinations

- Recall: Parameters are the set of numbers within a model that are used to define the model function -- you can think of it as the coefficients of the function -- that are adjusted or "trained" during the machine learning process to accurately transform inputs into desired outputs. They are just numbers that are multiplied and added in various ways
- The combination of parameters with a set of inputs via multiplication and addition is known as a **linear combination**, and this comes from the fact that in cases where we have one feature and one parameter, the resulting function is a line, and this can be naturally extended to higher dimension

Activations

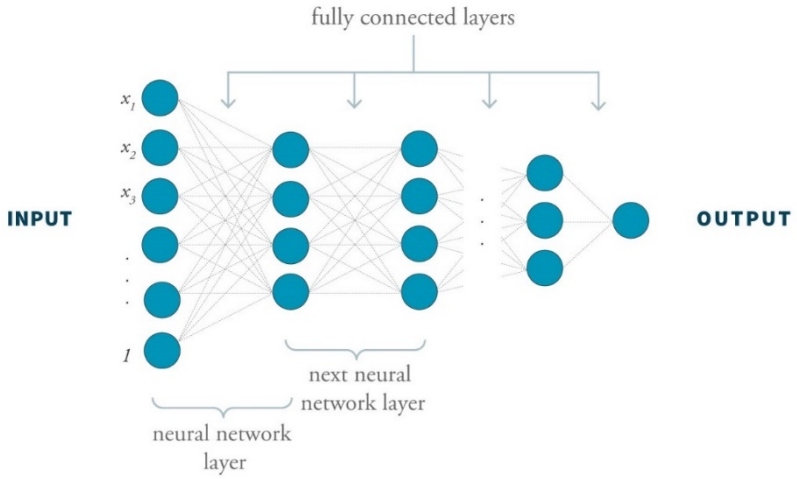
- Recall: We use logistic regression when the output label is categorical. We used the sigmoid function to transform the function from a line to an "S" shape in order to fit our categorically labeled data
- The sigmoid function, as we had mentioned earlier, is known as a nonlinear transformation– in comes a line, out comes something else. In deep learning terminology, we often use the term activation functions to refer to the nonlinear transformations we use, and we call the result of a linear combination followed by an activation function as the activations associated with the parameters involved



- Examples of activation functions
    - Sigmoid
    - ReLU (more popular): Many recent models use what is the ReLU activation function, which stands for Rectified Linear Unit. This function passes input values as-is if they're positive, but turns all negative inputs into zero. It's called a rectified

linear unit because it rectifies, or passes through, only the positive side of the input range, and what it does pass through is linear. And the rectification makes the function not a line, such that it is a nonlinear transformation

Neural Network Building Block: Dense Layers



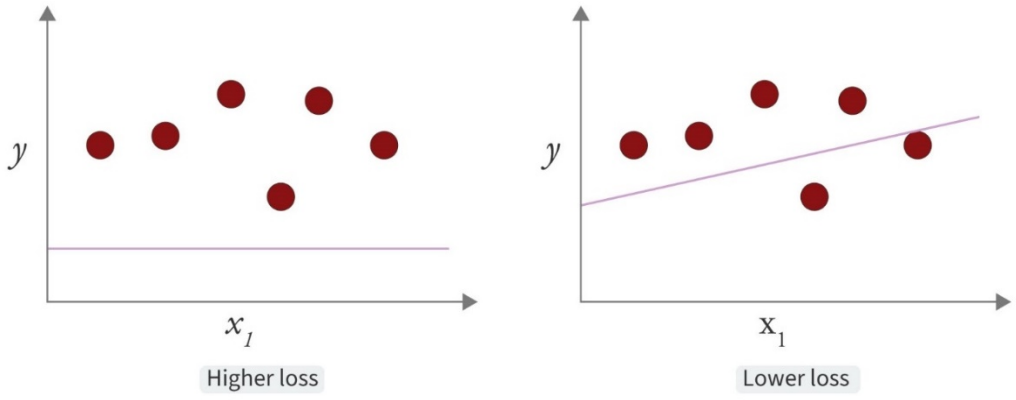
- The neurons of a neural network are, for all intents and purposes, miniature logistic regression models
- A **layer** of a neural network consists of a set of neurons that each take the same input
- **Fully connected layer** (also called a **dense layer** or a **linear layer**) is a set of neurons that take the same input. These neurons execute a linear combination of the inputs and the layer's parameters produce an output. These outputs can then be fed into another layer.
- The architecture of a neural network is how these layers are organized

Reviewing the training loop:

- We split our data into train, validation, and test datasets
- We take a pass through our training dataset:
    - Our model will make a prediction based on the sample's features
    - We will compute the loss between the model's prediction and the sample's label. The loss is a numerical value representing how far the prediction is from the label. Low loss is good, and high loss is bad
    - The model will then update its parameters in a way that will reduce the loss it produces the next time it sees that same sample. This is known as an **optimization step**
- Periodically, for example after we have taken a pass through our training dataset, we can evaluate our model on a validation set

    - In this phase, we assess if the parameters the model has learned produce accurate predictions on data that it has not yet observed, in other words the validation set.
    - The model does not learn from these samples because we do not execute the optimization step during this phase. Without the optimization step, the model cannot update its parameters, which in turn prevents learning
    - The validation set is a measure of how the model will do "in the real world." We save a version of the model if it gives us the best validation performance we have seen so far
- The process is repeated multiple times, each time with different training configurations. This is known as **hyperparameter** tuning.
- This module focuses on the optimization step, which is comprised of three components:
    1. Loss
    2. Gradient Descent
    3. Backpropagation

Loss: Informs the way in which all of the different supervised machine learning algorithms determine how close their estimated labels are to the true labels



- The goal of training an algorithm is to find a function/model that contains the best set of parameters that results in the lowest loss across all of the dataset examples
- There are multiple loss functions that exist, and some are better adapted for some problem types than others. Some factors to consider:
    - What kind of problem are we solving?
    - How much weight should we put on outlier labels?
    - Are the number of samples we have in each category roughly equal? (for classification)

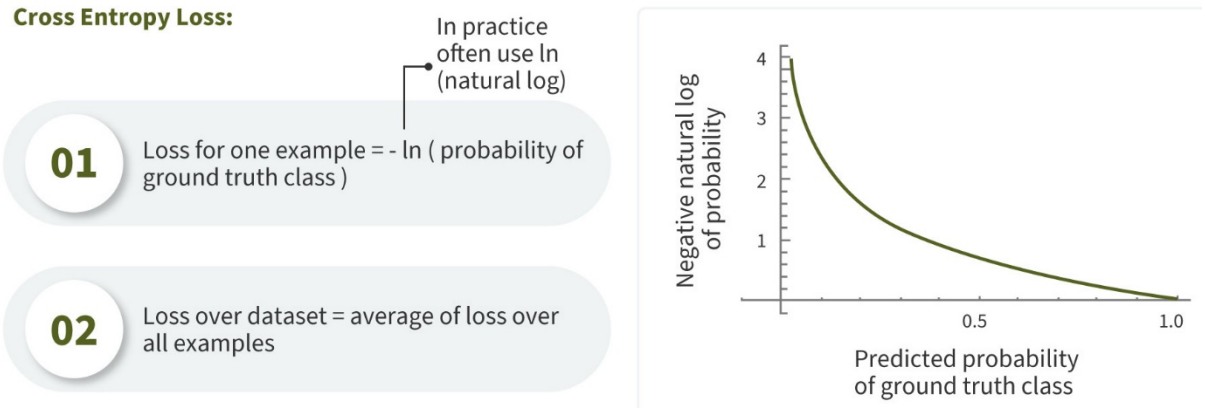
Loss functions: A mathematical function that processes all of the individual losses to come up with a way to decide how well a model performs

- Mean Squared Error (MSE)
  - The simplest and most common loss function
  - To calculate the MSE, you take the difference between the model predictions and the true label, which is also known as the ground truth, square it, and average it out across the whole dataset
  - Squaring (1) gets rid of the sign (+/-) of the difference between the prediction and the ground truth and (2) emphasizes outliers
- Mean Absolute Error (MAE)
  - To calculate the MAE, you take the difference between the model predictions and the ground truth, then take the absolute value to that difference. Finally you average the sample losses across the whole dataset
  - Since we did not square the error, all of the errors will be weighted on the same linear scale. Thus, we do not put more weight on our outliers

IMPORTANT CONCEPTS IN DEEP LEARNING

Cross Entropy Loss: The most common loss function in classification settings. This function determines the loss by the difference in the probabilities of the predictions.

The function is simple– you sum the negative log of the model's predicted probability for the ground truth class. Because probabilities are between 0 and 1, the log value is some negative number.

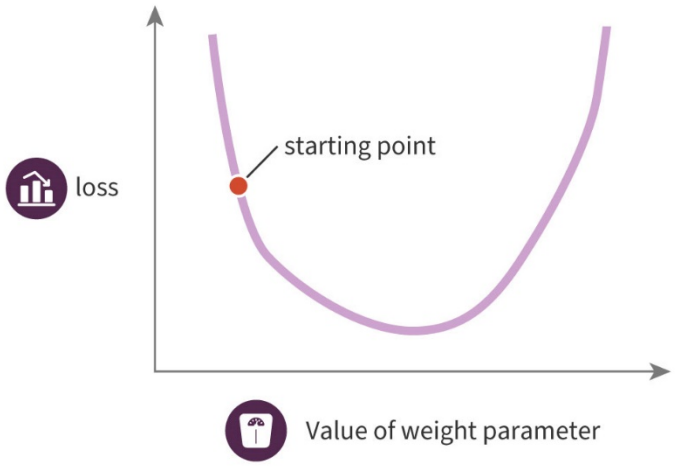


Sanity check:

- The log of a value that is close to 0 is a large negative number. Because we are using the negative log, this flips to being a large *positive* number
- The negative log of a value that is close to 1 is close to 0
- These dynamics are in line with what we need from a good classification loss function. In order to achieve a low loss, a classifier will have to produce probabilities for the ground truth class that are close to 1

Gradient Descent: Optimization algorithm to find good model parameters by minimizing a loss function

- The loss computed for a given sample gives each model parameter a sense of where to go; whether or not it needs to increase or decrease its value in order to allow the model as a whole to produce a better prediction the next time



- Each parameter has a starting value – often this number is set randomly
- The job of the gradient descent algorithm is to first figure out where the current parameter value is in relationship to the parameter value that produces the optimal loss. Then, it adjusts the parameter value in the correct direction
- To do this, it will find the slope (a.k.a. the derivative) of the loss function with respect to the parameter and figure out which direction to move. This is called calculating the gradient

Backpropagation: The key technique that breaks down gradient computation into easier gradient computations that are then combined together, and is the secret sauce for allowing us to obtain gradients for large neural network models

- At a high level: backpropagation allows parameters near the end of the network to send information about how to adjust to the parameters near the beginning of the network. The

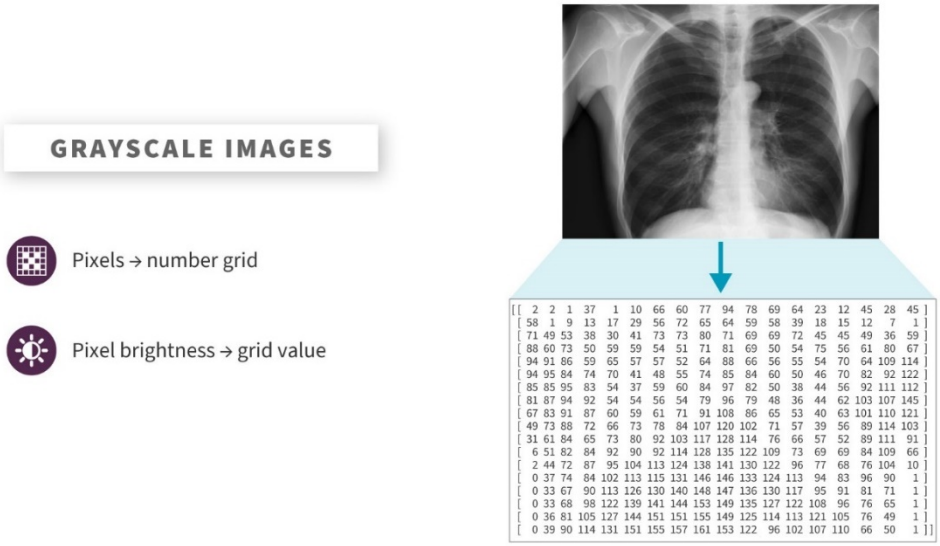
process goes from the end of the network to the beginning (back) and sends information from layer to layer (propagation)

Things to keep in mind about Gradient Descent:

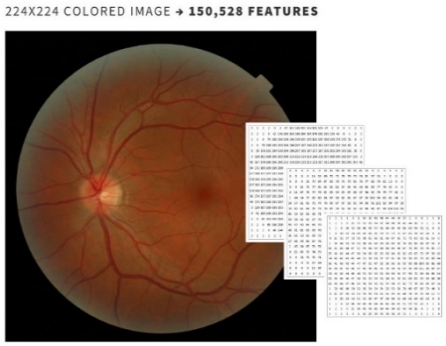
1. There are many variations of gradient descent
2. There are a number of other factors that can customize the way you move towards minimizing loss according to the gradient

REPRESENTING UNSTRUCTURED IMAGE AND TEXT DATA

Images are represented as one or many grids of numbers stacked on top of each other.

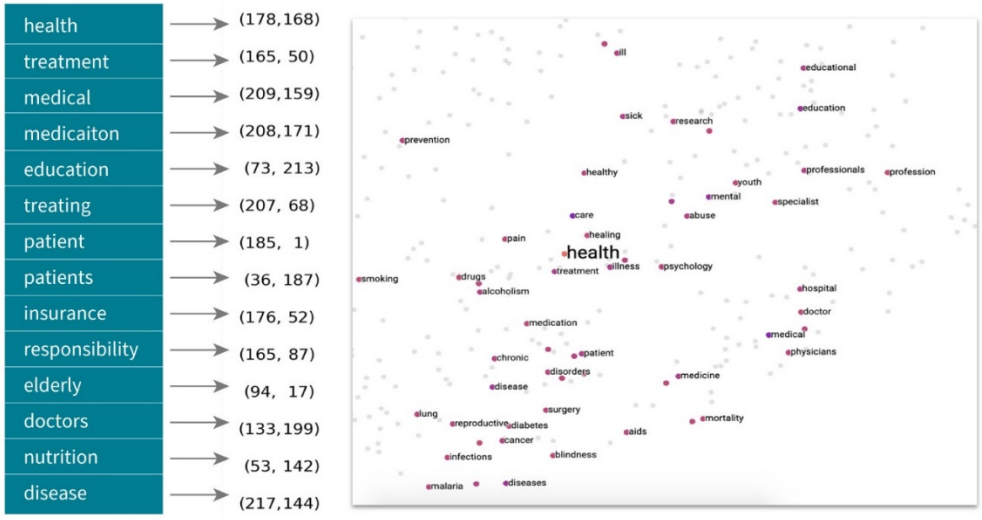


A low number is black and a high number is white. The numbers generally range from 0 to 255, where 255 is the maximum number representable by a single byte in a computer. And if we imagine overlaying numbers that represent each pixel brightness value over a black and white image what you will have as a result is a grid.



Instead of one grid, there are three grids for color images, each grid of numbers representing the brightness of Red, Green, and Blue, respectively. The magnitude of each number represents how much of a color exists in a given pixel. If a certain location in the grids has high red and blue values, it will show up to the human eye as purple.

Words are represented using **word embeddings**. A word embedding is a geometric (think "spatial") representation of a word. Here is a simple example of some 2-dimensional word embeddings.



Typically the number of dimensions are very large, and typically 300- or 1024-dimensional word embeddings are used, so that all the nuanced relationships in language can be captured encoded via coordinates in space.

The number of input features becomes enormous when we move into the unstructured data scenario.

- The number of features in a color image that sized to 224 pixels x 224 pixels (common among machine learning models) would 150,528 features for each image.
- A single sentence of text in the English language consists of about 10 words, and common word embeddings are in 1024-dimensional space. That would be be 10 * 1024 = 10,240 features for each sentence.

In order to train neural network models on high-dimensional input data, we have to be creative in the way that the network architectures are constructed.

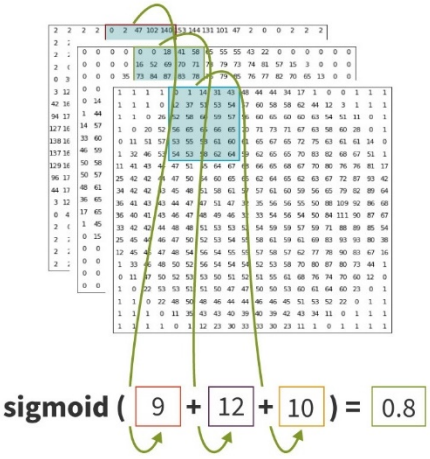
CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs): Designed with images in mind.

Recall: Images are pixel grids. Using dense layers, it would be extremely difficult to process these pixel grids.

- Issue 1: Each parameter would be dedicated to only one or a handful of pixel locations at a time. In nearly all practical settings, images present objects in highly variable positions.
- Issue 2: The pixels above, below, and next to a given pixel are highly related. Flattening the image into a feature vector destroys the spatial information we have about the image.

Convolutional layers solves these two issues using convolutional filters (kernels).



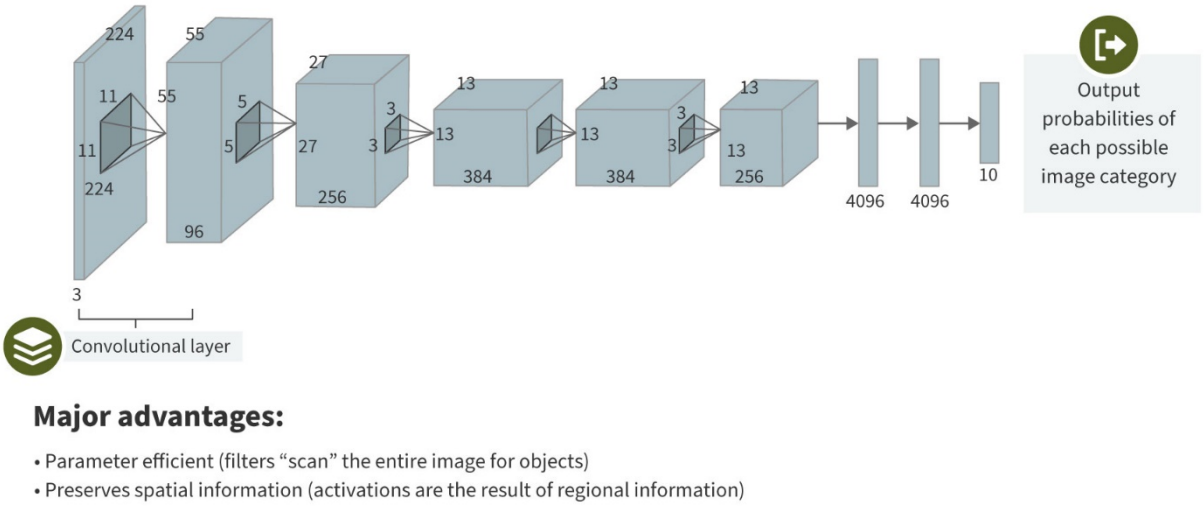
0.9	0.4	0.8				

Convolutional filters are small groups of parameters that are multiplied and summed (i.e. a linear combination) with 1 small patch (or window) of the image at a time. The output of each linear combination is placed relative to the location of the input patch in a new, sometimes smaller number grid.

You can think of the filter as an "feature detector." Since only one value is produced for every patch in the image, in a trained model one can imagine a high number being computed if something of interest is found in the patch and a low number if not.

One can stack convolutional layers: the grid of activations produced by one convolutional layer can act as the "image" input of the next layer.

- Since each pixel in the next layer is comprised of information from a patch of pixels in the inputs of the previous layer, the later layers of the CNN are then able to "see" larger and larger patches of the images. The amount of raw pixel information a convolutional filter can see at any given moment is called its "receptive field."



Convolutional layers are what make CNNs, CNNs. CNNs are trained just like dense neural networks, albeit computing the gradient for gradient descent becomes a slightly more complex procedure. CNNs are immensely popular in subdomains of medicine such as radiology and nuclear medicine, among others.

NATURAL LANGUAGE PROCESSING AND RECURRENT NEURAL NETWORKS

Natural Language Processing (NLP) is a machine learning method that can enable computers to understand and organize human languages. It requires a different type of neural network architecture.

The difficulty lies in the neural networks ability to understand not the vocabulary, but the meaning and context behind each word.

There are two major architectures currently being used for NLP– **Recurrent Neural Networks (RNNs)** and **Transformers**.

Recurrent Neural Networks, or RNNs, are a type of model architecture typically used in scenarios where the unstructured data comes in the form of sequences. Most commonly, they are used to solve Natural Language Processing (NLP) tasks.

- NLP tasks often take a slightly different form than typical machine learning tasks due to the fact that **inputs and outputs can take the form of sequences**.
- A sentence can be thought of as a sequence of words, which we discussed would look like a sequence of word embeddings.
- Like with images, we can consider flattening this time sequence data into one vector and feed it into a dense neural network. This has a few issues:
    - Issue 1: Just like with images, each parameter of the first layer of a DNN would be assigned to a single feature of a word embedding at a single timestep. Sequences (especially in language) are far too dynamic for this.
    - Issue 2: There would be no way to vary the length of the output. The final layer of a DNN always produces an output of fixed size.

RNNs address the above two issues by doing the following:

- They process information from one timestep at a time. In language, this means processing word embedding at a time.
- They can store information about past timesteps in what is called the *context* vector. The context vector from the previous timestep is used as additional input to the current timestep feature vector in order to give the RNN information about the past.

The core component of a recurrent neural network is known as an **RNN cell**.

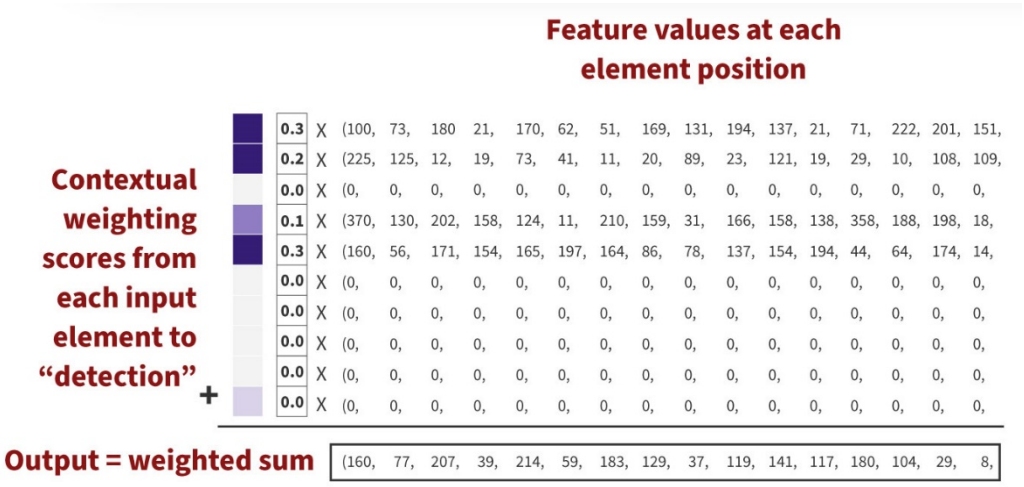
- RNN cells can take as input both the output from earlier layers of the neural network and an additional "context" set of values that can be used to pass information from one timestep to the next.

THE TRANSFORMER ARCHITECTURE FOR SEQUENCES

The Transformer architecture is quickly replacing RNNs in sequence-based problems.

- RNNs start to lose effectiveness if they have to deal with long-range dependencies in a given sentence.
- Further, on the more technical side, it is hard to "parallelize" the process of an RNN. One of the most powerful features of modern computing systems is the ability to do many tasks at the same time. For a given element in a sequence, RNNs need the information from the past elements in order to output predictions, therefore their computations have to happen sequentially. For long sequences, this becomes a problem.

The Transformer architecture is built around a layer known as the **self-attention layer**, which allows for the processing of sequences all at once, while at the same time producing outputs that are aware of the rest of the sequence.



- Self-attention layers compute a contextual relatedness signal, or weight, between every element and the others in an input sequence.
- The element at each position is transformed using a weighted sum of feature values from the elements in other positions, based on the strength of the contextual relatedness. Self-attention layers can produce multiple weighted outputs to encode different types of context.
- They look at the entire input sequence and then "pay attention" to context elements variably based on what element values actually are.

Transformers directly address some of the problems associated with RNNS.

- Self-attention layers looks at the entire input sequence at once, so it avoids the forgetting problem associated with RNNs on long sequences.
- Since entire input sequences are processed at once, self-attention layers are more efficient than RNNs which have to sequentially process input elements one by one.

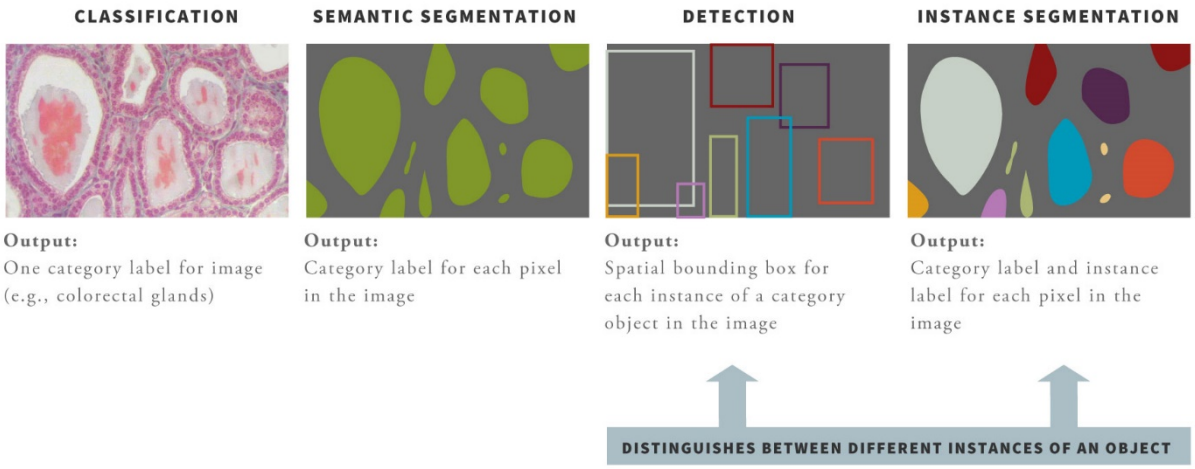
Transformer architectures stack many self-attention layers on top of each other. So far, they have proven themselves to be both faster and better performing than RNN architectures in many settings.

OVERVIEW OF COMMON NEURAL NETWORKS

ImageNet: Benchmark dataset that encouraged the creation of numerous now widely-used CNN architectures.

- AlexNet, named for its primary inventor, Alex Krizhevsky, was introduced in 2012 and made a splash by cutting the best previous error rate on the ImageNet challenge by almost half.
    - Architecture: 8 layers -- 5 convolutional layers and 3 fully connected layers at the end. First architecture to use the ReLU activation function.
- VGG and GoogLeNet architectures were introduced in 2014. VGG was named for the Visual Geometry Group at Oxford University where it was developed, and GoogLeNet, as you probably guessed, came from Google.
    - Both the VGG and GoogLeNet architectures were still CNNs but significantly deeper - 19 and 22 layers - respectively, and came in at the top of the ImageNet challenge that year with significant further improvements.
    - The VGG architecture looks much like the AlexNet one, but found success through smaller filter or receptive field sizes at each layer, which enabled a deeper network to work well.
    - The GoogLeNet architecture, on the other hand, looks a little more different, with modules that they called Inception modules stacked on top of each other.
        - Each Inception module has the structure of several parallel convolutional pathways.
        - Called Inception because these were like mini-CNNs within a CNN.
        - Because of the Inception modules, GoogLeNet is also interchangeably referred to and perhaps more commonly known as the Inception network.
- ResNet architecture, developed by Microsoft Research in 2015. The first architecture to beat a decently accurate human benchmark on the ImageNet dataset
    - Variants of the architecture had 152 layers or even more, compared to the previous architectures of around 20 layers, and this moment was known as the “Revolution of Depth” in deep learning
    - ResNets introduced a new type of mini-module that they stacked in their CNN called residual blocks. These blocks is have skip connections which pass inputs or intermediate inputs to later portions of the network, which allowed information from the beginning of the network to reach the end of the network more directly

Semantic Segmentation and Object Detection and Instance Segmentation:



- Semantic segmentation
    - Semantic segmentation allows us to obtain pixel-level granularity of where categories are present in an image. However, it does not allow us to differentiate between distinct instances of category objects in the image, for example distinct tumors or lung nodules
- Object detection
    - In this case, the neural network outputs bounding boxes corresponding to the center and height and width of a box that tightly borders each instance such as an individual lung nodule in the image
- Instance segmentation
    - The output of an instance segmentation neural network is both the bounding box of each instance, as well as a pixel mask corresponding to the segmentation of the object within each bounding box

Each of these tasks, because they produce different types of outputs, require different neural network architectures or structures to produce each type of output.

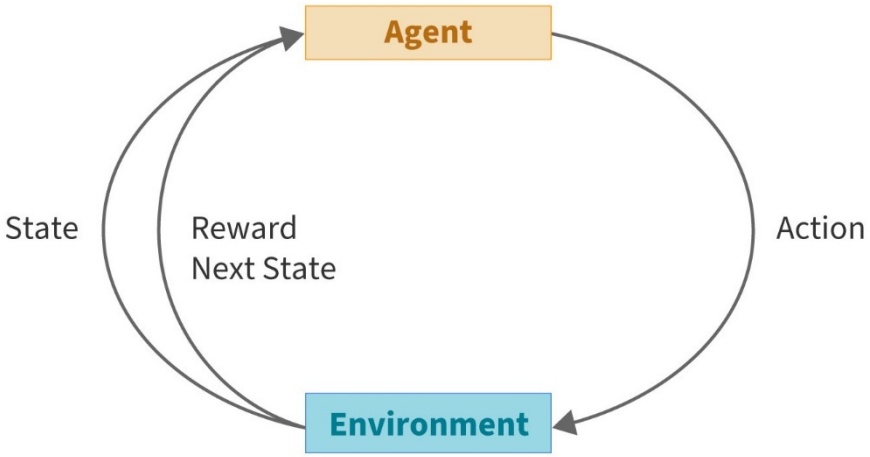
- For segmentation, U-Nets are commonly used. The U-Net architecture contains a downsampling and upsampling pathway. The downsampling pathway extracts important information from the raw image information. The upsampling pathway generates a pixel map of the same shape as the raw image, where each pixel has a value that corresponds with the segmentation prediction
- Object detecting neural networks predict spatial bounding boxes of objects and have both classification and regression branches. The classification branch predicts the category of objects as we've seen before, while the regression branch outputs numerical values corresponding to the location and extent of the object bounding box

- Finally, instance segmentation combines these two tasks. One branch is single category classification for each object or bounding box. Another is a branch of the network that produces a pixel map
    - This technique is especially attractive for imaging tasks that deal with a large number of objects that are repeated or crowded
- These tasks of semantic segmentation, detection, and instance segmentation are more detailed than classification, and so they also require more detailed labels in the training dataset

Deep Learning:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Reinforcement learning, currently the most challenging area of machine learning. It is one of the least explored for healthcare applications.



- Reinforcement learning is centered around the idea of the model interacting with an environment as an "agent", continuously observing the current state of the environment and making decisions based on it.
- We use the word agent here because taking actions is the central concept in reinforcement learning, the AI agent here is basically our algorithm. This is often used in the setting of a gaming or simulation environment.
- Main challenges in healthcare:

- There is not a clear methodology towards environment simulation. In games, environment simulation is easy, because one can just run the game. In healthcare, experiments are much more high stakes.
- Not immediately clear how to reward the agent. In games, there are explicit scoring mechanisms. In healthcare, it would be dangerous to naively index patient wellness on a single metric.

CITATIONS AND ADDITIONAL READINGS

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Komorowski, M., Celi, L.A., Badawi, O. *et al.* The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care. *Nat Med* **24,** 1716–1720 (2018). https://www.nature.com/articles/s41591-018-0213-5