CRAVATH

MAY 2024

Tech Explainers

NEURAL NETWORKS

INTRODUCTION

In this Tech Explainer, we will provide a foundational understanding of neural networks, a pivotal technology that underpins many methods of machine learning (ML). Neural networks are sophisticated algorithms that consist of interconnected nodes, or neurons, arranged in layers that process and manipulate data. This architecture allows neural networks to learn from large datasets and apply this learning to make complex predictions.

Whereas artificial intelligence (AI) is a broad, function-oriented category describing any system that can simulate human intelligence, ML simulates the human ability to learn from ingested data. Neural networks are the functional components powering this capability, enabling tasks such as image and speech recognition, natural language processing and autonomous decision-making.

To bring the abstract concept of neural networks to life, we will walk through a very basic actual application of a neural network that can make a decision based on a given set of inputs. We will then describe how more complex systems layer additional components to provide the "intelligence-like" results we have come to expect from AI systems. First, though, we will introduce neural networks.

WHAT IS A NEURAL NETWORK?

A neural network is a collection of interconnected "nodes" arrayed in successive layers, often represented as depicted in Figure 1.[1]

1 In particular, we depict a feedforward neural network, a unidirectional network in which information flows only forward from the input nodes. This means that in feedforward neural networks, input is entirely independent of output. Recurrent neural networks, by contrast, have bidirectional information flows, meaning output from one step is passed on as input to a previous step (*e.g.*, predicting the next word in a sentence requires preceding words in the sentence).

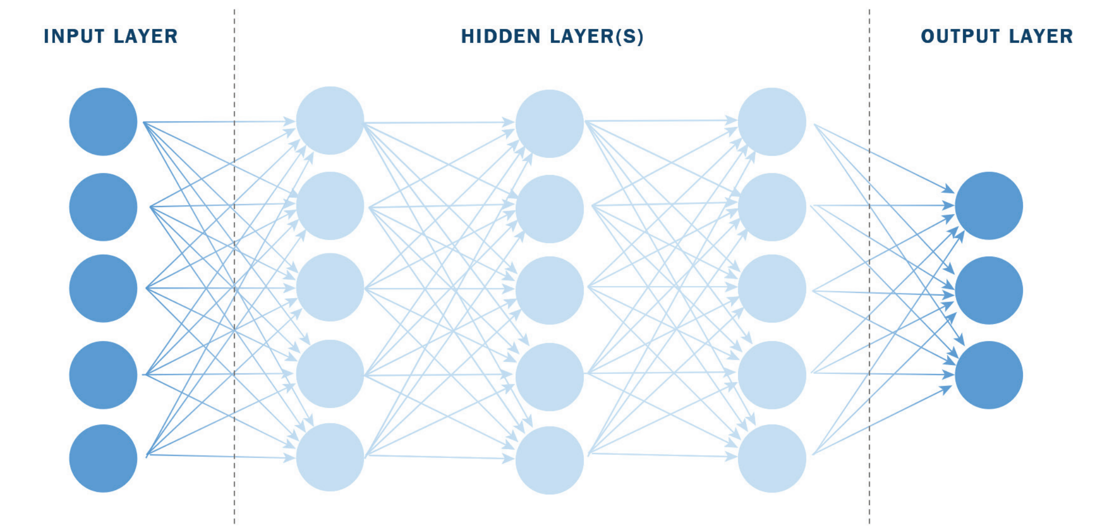

FIGURE 1

The nodes (circles) in the neural network are called "neurons". The neurons are arranged in layers. Each neuron in the first "input layer" (the left-most layer in the diagram) represents one value inputted to the system (for example, input signals from the outside world). Each neuron in the input layer formulates a single output, which is propagated as input to each neuron in the subsequent layer, a "hidden layer". Each neuron in a hidden layer uses a mathematical operation to synthesize these multiple inputs in a particular way, governed by custom settings (called "weights"), which cause the neuron to give some inputs (*i.e.*, the outputs from some nodes in the prior layer) greater weight than others in formulating its output. Therefore, each neuron essentially reports to the next layer its conclusion about the previous layer, given the patterns it was designed to focus on pursuant to the various weight settings. As we demonstrate later, neural networks may consist of many hidden layers. The final "output layer" neurons use mathematical operations to produce the final result, or prediction, of the system.

Proper training and "learning" of the system involves setting and then adjusting the weights of each neuron. This process is accomplished without a human dictating the importance of each of the different factors represented by each neuron. Rather, the weights are automatically adjusted based on training data. In "supervised learning", AI systems may be trained using massive datasets in which the data points are labeled—for example, a system is given millions of images labeled "apple" and "orange" and adjusts the weight of each input (color, texture, etc.) until it can correctly identify a new image as an apple or an orange. In "unsupervised learning", the data is unlabeled—the system is given millions of unlabeled images of apples and oranges and adjusts the weights in order to group similar images (apples and oranges, ripe fruit and rotten fruit, etc.). Beyond this, though, we will not be covering training in this introductory article, and will presume the neural networks we explore have been pre-trained with all weights and other variable parameters properly set for the desired function.

DECISION MAKING: THE SINGLE-LAYER PERCEPTRON

Neural networks are, at their core, prediction engines. To show why, we begin with an overly simplistic example of a neural network, called a perceptron[2]—with a single hidden (or middle) layer comprising one neuron—that, in our example, predicts whether you want to go skiing today.[3] The system ingests three input values: snow quality, avalanche danger and whether you will be accompanied by a friend. For simplicity, this system takes binary values (0 or 1), so the three nodes of the input layer will look like this:

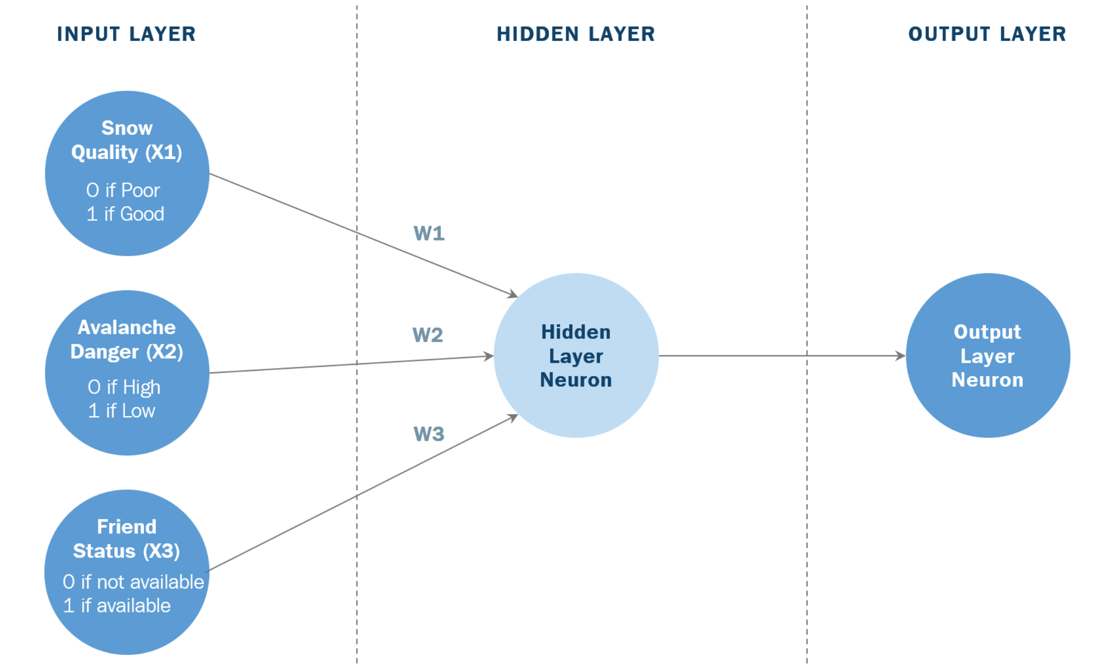

FIGURE 2

The neuron in the hidden layer will take each input, multiply it by a corresponding weight (each weight will be different because you care differently about the three factors), and pass it through a function described below to determine whether or not you should go skiing based on the given inputs.

The weights, as the name implies, allow the system to adjust the impact of different inputs. In mathematical terms, we say that the neuron computes the weighted sum of the inputs, which equals the sum of each input value multiplied by its weight: in our three-input example, $(x_1 \times w_1) + (x_2 \times w_2) + (x_3 \times w_3)$.[4] For example, you may care only minimally whether a friend is available to join the trip (let's say $w_3=2$), whereas snow quality may be a bigger factor ($w_1=4$) and avalanche danger may be the biggest ($w_2=5$). With these weight values, our neuron uses the following function to process any given input set: $(x_1 \times 4) + (x_2 \times 5) + (x_3 \times 2)$.

2 Frank Rosenblatt implemented the first perceptron in 1957 on an IBM 704 mainframe computer. Perceptrons always use supervised learning of binary classifiers.

3 As discussed above, we assume that our model has been trained with a sufficiently large dataset; in our case, the machine must have been given information on snow quality, avalanche danger and whether you will be accompanied by a friend, for each day you chose to ski or not to ski.

4 For *n* inputs, this can be represented using the summation operator as: $\sum_{j=1}^{n} x_j w_j$.

So on the worst possible ski day (poor snow quality, high avalanche danger and no available friend), the weighted sum will be $(0\times4)+(0\times5)+(0\times2)=0$. At the other extreme, on a day with all factors in your favor (good snow quality, low avalanche danger and available friend), the weighted sum will equal $(1\times4)+(1\times5)+(1\times2)=11$.

The hidden layer neuron takes this weighted sum[5] and passes it through a function (called an "activation function") that translates the result into the next layer's input. The activation function is accompanied by a threshold value. In our simple perceptron example, the activation function is a basic unit step function that yields a binary decision.[6] If the activation function yields a result under the threshold value, the output will be 0, so the neuron remains inactive; for results above the threshold value, the output will be 1, so the neuron "fires". The final "output layer", in turn, takes as input the result of the hidden layer and translates that into a final output: an answer to your ski trip quandary.[7] If the hidden layer neuron returns 0, the output layer outputs "do not go skiing". If the hidden layer neuron returns 1, the output layer outputs "do go skiing" and the trip is on.

This process can be summarized in a general sense with the diagram below:

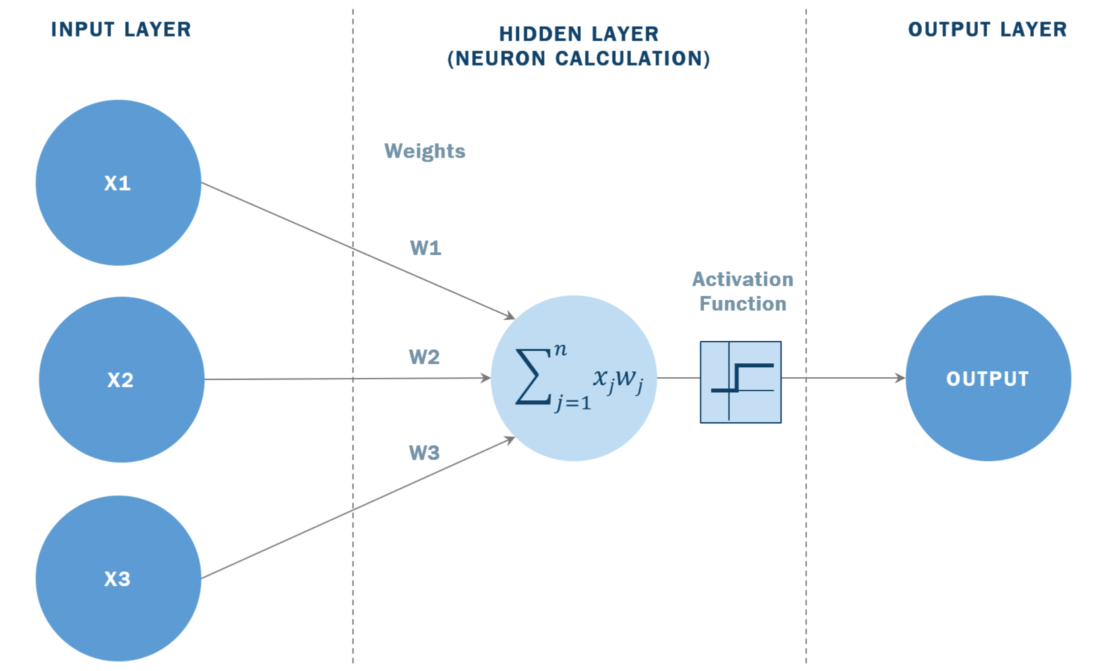

FIGURE 3

5 The hidden layer neuron typically adds to the weighted sum a bias term, transforming our calculation into $(x_1\times w_1)+(x_2\times w_2)+(x_3\times w_3)+b$. We do not cover bias in this introductory article.

6 The unit step function, sometimes called the Heaviside step function, works well when data is linearly separable. In a more complicated neural network, where data is not linearly separable, the unit step function would not work well. Popular activation functions in the real world include sigmoid and rectifier functions (the most popular activation function currently is the Rectified Linear Unit function, or ReLU) and are beyond the scope of this introductory article.

7 In a more advanced system, the output layer would typically be more complex, and, for example, include multiple nodes that each represent the probability of a certain result (with the probabilities of all the nodes, representing all the possible results, adding up to 1).

To illustrate the decision-making capabilities of our system with some exemplary input values, let's assume a threshold value of 6 for the activation function.[8] On a day with good snow ($x_1=1$), but high avalanche danger ($x_2=0$) and no friend ($x_3=0$), the system will advise you to not go skiing: $(1\times4)+(0\times5)+(0\times2)=4$. But on a day with low avalanche danger ($x_2=1$) and an available friend ($x_3=1$), the system will give a green light even when snow conditions are poor ($x_1=0$): $(0\times4)+(1\times5)+(1\times2)=7$.

This example illustrates how a single neuron can make an "intelligent" decision by performing calculations on multiple inputs according to preset parameters. This decision is "intelligent" because it learned from the large amount of training data that it ingested how important each of the factors were in your historical skiing decisions and adjusted the weights of those factors appropriately for predicting future skiing decisions. At the conclusion of the process, the system is able to make decisions that are increasingly accurate even for scenarios not included in its training data.

These systems become exponentially more complex and powerful with the introduction of more input values, more hidden layers and more neurons in each hidden layer—each neuron having its own different set of weights. This added complexity allows different neurons to focus on different patterns, and also to be able to identify patterns within patterns. In the next section, we will build upon our basic single-layer perceptron to illustrate what happens when additional neurons and layers are added to the system.

ADDING COMPLEXITY

Let's begin by adding a fourth neuron to the input layer representing temperature (for simplicity, ($x_4=0$) when below freezing and ($x_4=1$) when above freezing). Our original neuron of the hidden layer will now have another data point, with an associated weight (which may be low in this instance, *e.g.*, ($w_4=1$), because you do not care that much what the temperature is on the slope) to help determine whether or not to go skiing. But now we add a second neuron to the hidden layer; this neuron will be connected to the same inputs but with different weights because it is trained to come to a conclusion about a different question than our original hidden layer neuron. This new neuron, for example, may decide whether you need to purchase new ski gear, because your current outfit is unfashionable and ill-suited for skiing. For this neuron, therefore, snow conditions and friend availability may play minimal roles, while temperature and avalanche danger may play higher roles. Our new expanded system can be represented by the diagram below:

8 Someone with a threshold value of 5 would be more committed to skiing regardless of conditions; someone with a threshold value of 7, less committed.

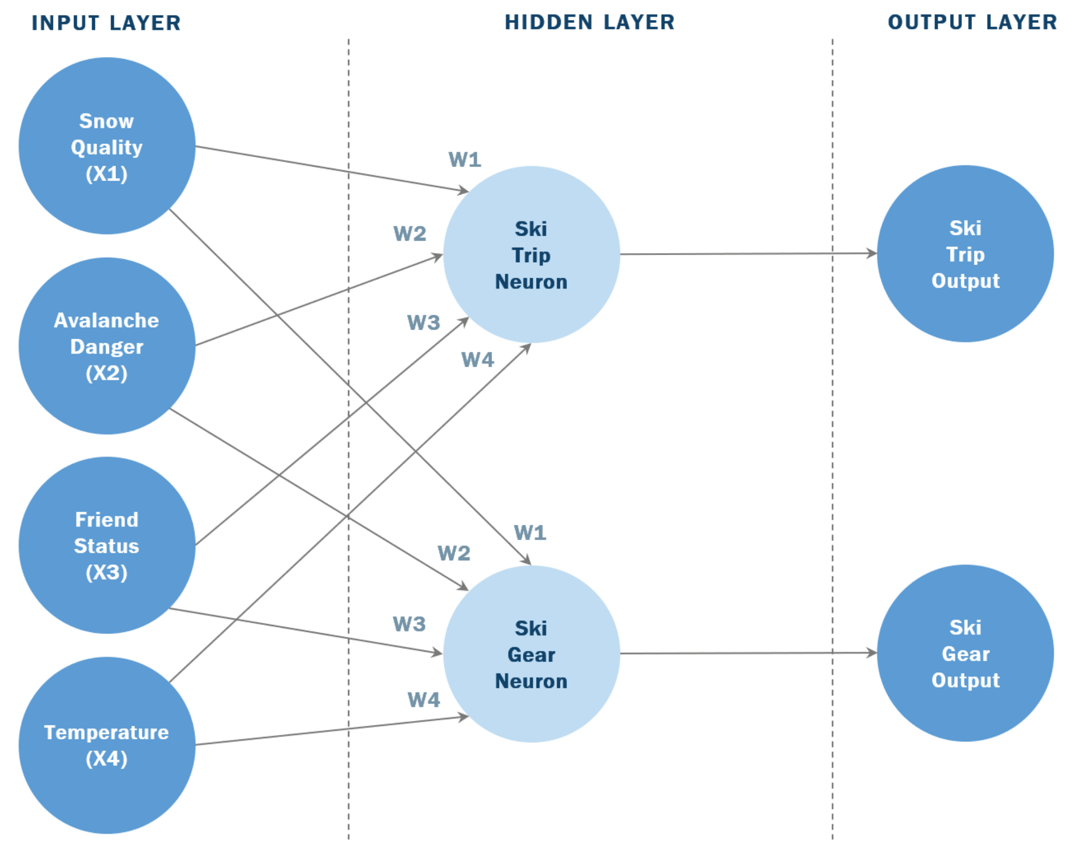

FIGURE 4

This expanded system, though, produces two independent outputs.

We can further expand the system by adding a second hidden layer, which synthesizes the conclusions of the two neurons of the first hidden layer to produce a new, integrated output.[9] For example, we can add a second hidden layer to produce a model that predicts whether you would indulge in some après ski. This neuron receives as input the two outputs of the previous layer—whether you are going skiing and whether you need to purchase new clothing—and computes the weighted sum of these inputs, which it compares against the threshold value of its activation function. If the weighted sum exceeds the threshold, the output will conclude that you will enjoy a beverage by the slopes; if the weighted sum is below the threshold, the output will advise against it. In this example, the weight associated with the skiing input may be low to represent the fact that you will have limited time to enjoy the après ski (this weight having been gleaned from the training data which indicated that historically, on days you spent on the slopes, you were less likely to spend time at the slope-side bar). The weight associated with the clothing purchase, on the other hand, may be high because you are eager to socialize while decked out in new gear. Our further expanded system now looks like this:

9 Neural networks with more than two hidden layers are known as "deep" neural networks.

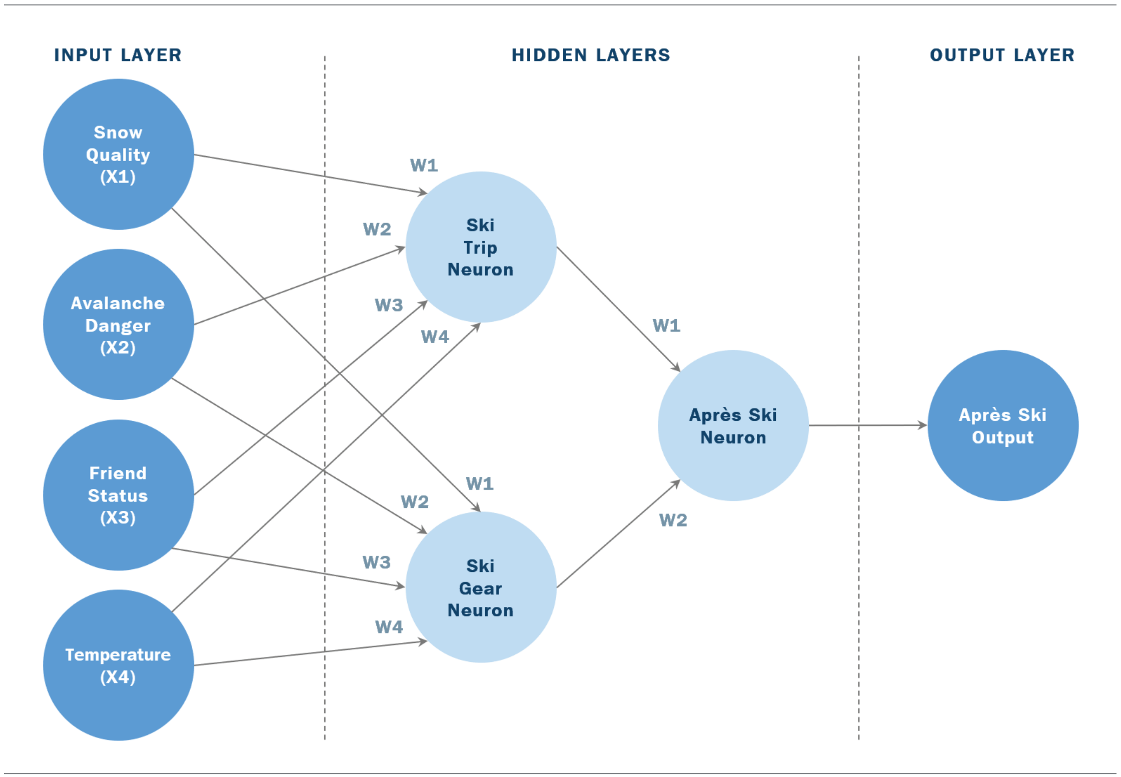

FIGURE 5

We have thus illustrated how additional neurons can provide different perspectives on the same input data, identifying unique patterns. Additional layers can build upon the patterns identified in the previous layers to generate more nuanced outputs. We can expand this idea to an even more complex application. For example, in a facial recognition system, the input layer may represent shading of individual pixels of an image. A first hidden layer may look for patterns of edges or lines embedded in the pixels (each neuron in this layer focusing on a different segment of the picture), and a second layer may look at combinations of edges identified in the first layer for facial features such as eyes, nose or mouth. A subsequent layer may look at certain combinations of those facial features to detect the probability of the picture being a human face.

It should be noted that this description is merely illustrative; in a real system, the layers and neurons may not have such well defined roles and the internal parameters may have been set and tweaked in response to training data without much visibility as to what factor each weight corresponds to. In this way, the training of complex AI systems is very much a "black box" endeavor, where adjustments are made in the dark, so to speak, and final weights are established only because setting them at those values appears to produce desirable outputs in most instances.

This idea can be generalized to numerous use cases, depending on the components built in to the system and its training. At its core, though, the neural network acts as a feature extractor using its carefully adjusted weights and parameters in the hidden layers to discern patterns within the input data and deliver "intelligent" conclusions which may rival (or even surpass) the ability of natural intelligence.

WHY LEGAL PRACTITIONERS SHOULD CARE

Neural networks underpin AI tools that are becoming wildly popular in so many fields, implicating many novel legal questions. These legal issues range from privacy concerns with submitting confidential training and input data to the systems, to licensing (including open source software) and copyright issues with respect to both the act of training and also downstream usage of outputs that are similar to the protected training data. Liability concerns also come into play regarding outputs that are inaccurate, biased or injurious (*e.g.*, an autonomous vehicle crash).

The legal community is also beginning to grapple with unprecedented questions around the IP protectability (*e.g.*, patent, copyright, trade secret) of AI systems and outputs, from an ownership perspective and also with respect to 35 U.S.C. Section 101 (eligibility) and Section 112 (written description/enablement) requirements. Understanding, for example, how parameter settings (weights, etc.) in AI systems are key to its functionality will help inform analyses of which IP protections are most suitable and how these IP protections can be best leveraged to protect such systems.

In general, now that we have demystified the process AI tools use to make decisions at the most basic level, we will be better equipped to face applicable questions and help our clients navigate the attendant cutting-edge issues.

David J. Kappos
+1-212-474-1168
dkappos@cravath.com

Sasha Rosenthal-Larrea
+1-212-474-1967
srosenthal-larrea@cravath.com

Carys J. Webb
+1-212-474-1249
cwebb@cravath.com

David M. Ungar
+1-212-474-1490
dungar@cravath.com

CRAVATH, SWAINE & MOORE LLP

NEW YORK

Two Manhattan West
375 Ninth Avenue
New York, NY 10001
T+1-212-474-1000
F+1-212-474-3700

LONDON

CityPoint
One Ropemaker Street
London EC2Y 9HR
T+44-20-7453-1000
F+44-20-7860-1150

WASHINGTON, D.C.

1601 K Street NW
Washington, D.C. 20006-1682
T+1-202-869-7700
F+1-202-869-7600