Good morning, I am Pam. I work at Stripe. And I've been there for about 2 and 1/2 years now. And most of what I do is building deep learning models for our various products. And I'll be sharing how we've applied deep learning to predict fraud from raw behavioral data. OK, so Stripe is a global technology company. And we build economic infrastructure for the internet. Businesses of every size from startups to public companies like Salesforce and Facebook use our software to accept online payments and run sophisticated financial operations in more than 100 countries. We combine economic infrastructure with a set of applications for new business models, like crowdfunding and marketplaces, fraud prevention, analytics, and more. Only about 3% of the GDP happens online today. Stripe wants to help companies get started and thrive and ultimately grow the GDP of the internet. So building fraud prevention into the fabric of our economic infrastructure is core to our mission. Stripe is available for businesses in more than 32 countries. And over 80% of American adults bought something on Stripe in 2017. Unlike physical stores, online businesses are responsible for paying the costs associated with fraud. And so we take that responsibility of keeping our platform safe from fraud very seriously. One of the types of fraud that we tackle is detecting fraudulent transactions that are processed on legitimate businesses. Across the Stripe network, our machine learning powered fraud product, Radar, prevented more than $4 billion in attempted fraud in 2017 alone. And alongside ensuring that our users experience better fraud protection over time, we also prevent fraudulent actors from using Stripe to create illegitimate businesses whose sole purpose is to engage in fraudulent activity. All right, historically Stripe has applied classical machine learning techniques such as decision trees to fraud detection. There are already a large number of real world applications with classical ML. And these techniques are really easy to understand. This is especially important because our users really care about understanding why the model made a given prediction. Some of the benefits of using classical machine learning techniques are that you can represent models as data. And so your machine learning production systems can be runtime independent from when you actually train them or use them offline with the experimentation systems. Some of the drawbacks of this is that improving the model's predictive accuracy comes from continually curating increasingly complicated features. And this is known as feature engineering. It's a manual process. You're kind of continually experimenting. And it's not necessarily monotonically improving in terms of how powerful your model actually gets. And on top of all of that, you kind of do need to have complex infrastructure to support feature engineering, very complicated things. And this infrastructure also needs to ensure that you are having consistency across online scoring and offline experimentation. It needs to handle aggregates and counters and large drawings in real time. So at a high level when we actually engineer these features, we're basically searching for combinations which demonstrate fraudulent intent over time. As an example, if one of the patterns that we look at is on average the time between purchases on a given card was every few days. But in the last hour, we've seen this card purchase repeatedly on the same business, this might be a very common pattern for when we can detect fraud. And we might engineer features to capture the average time between purchases per card as well as the purchases per card in the last hour. But we kind of took a step back and thought about this at a higher level and asked the question, what if we could engineer a model that actually learns these patterns directly from raw event sequences. So there's a class of deep learning models called recurrent neural networks, which can work with variable length sequences of events. This model class has been successfully applied to

many real world natural language processing applications, such as translation between languages, speech recognition, and image captioning across companies like Google and Facebook and Amazon and other places. Inherently the model class is able to remember signals from the past when it's learning about future events and can learn to perform a variety of operations that we consider a core part of our toolkit when we're actually feature engineering. So while these models seem to work remarkably well for natural language processing tasks, it's the structural information in these tasks that the model is able to capture. So we can generally consider applying these models towards sequences of events where the structural information in the sequence itself matters. And I'll get a little bit more concrete here because that was pretty abstract. So let's take a look at an example of behavior sequences that a user might take and how that can be used with an RNN. So let's say our user signs up for an account. And next they actually verify their account. During this process the RN'S is continually trying to predict, given this new event and the events before what is the probability of fraud. Once they've actually gone through the verification step, they might choose to add or update their address and any additional account details that we require. And perhaps this looks a little fishy. And the network thinks that the sequence of actions combined with some of the details added are reminiscent of fraudulent behavioral patterns. After which the user might choose to log out. Nothing particularly strange about this action. But the network remembers that there was something a little fishy and kind of remained suspicious. And maybe a few days later, the user may choose to log back in, which doesn't seem too alarming. And many well intentioned users in the past have behaved similarly. So maybe this kind of drops down in terms of probability of fraud. But then as soon as they begin to process payments shortly thereafter, once they've logged in, the network has learned that this particular sequence of actions is associated with a high probability of fraudulent intent and start to really present a high probability of fraud score. So when we apply these models to a sequence of behavioral data, we can extract different patterns common to fraudulent intent with less effort in engineering these patterns as features. By allowing the model to do a lot of the heavy lifting, we can train a model to predict the probability of fraud given the current sequence of events. This modeling approach is also more generally applicable to and reusable for making other types of predictions, not just probability of fraud from event sequences as well. You might have different data sets where perhaps you have a label for the entire sequence of behaviors as opposed to each event having a individual label. And in either case, you can actually apply different varieties and flavors of RNNs where you're either predicting a many to one, you're building an architecture that predicts a many to one or many to many setting. And these models are able to learn effectively in both cases. So in practice what we need to do is actually transform the shape of our data sets, such that when we train an RNN to learn to model these events, it's able to understand what it's actually seeing. For our fraud detection purposes, we encode each event in the event sequence such that the type of event is actually a categorical value. So in this case, let's say the event type is sign up, we'll consider that a categorical value. And for timestamps, we actually tried to make it stationary. And so what we do is delta encode the timestamps where that value is transformed into time since last event. For all of our categorical metadata, we map the distinct values to vocabulary. And eventually this will be used to pull out embeddings for each categorical value that are jointly trained with the actual RNN. For numerical metadata, we scale these to fit between 0 and 1, so that the RNN can learn more effectively. Our training data pipeline is built on top of

Spark, which allows us to use SQL or Scala to process and transform our data. This data engine allows us to scale up to produce larger training data sets, which is very useful for deep learning models. These outputs from the pipeline are serialized to parquet, which is a common file format that can be used with the Spark environment and can also be read with Python processes, which is how we actually train our model. When we train the model, we deserialize these pieces of data with pyarrow which is a Python library that is able to read data that has been written in the parquet file format. Each event sequence is represented as a sequence of categorical vocabulary indices. As I mentioned before, these will be used by the model to map from categorical value to embeddings and a sequence of numerical values which are the scaled numeric values from the event itself. And then we label these either as one label per event in the sequence or one label for the whole sequence depending on the actual task at hand. So let's dive into the model architecture itself. The model has an embedding lookup table which is attached to the RNN. This means that after the training data has been deserialized, the model first looks up the appropriate embedding vector for each categorical value in the event. So let's say the event type was sign up, It'll actually pull up a embedding vector for the sign up key. And that will eventually move around relative to fraud when it's trained with the RNN. And one of the challenges here is that the more categorical features you have, the higher the input dimensionality actually gets. And so to work around this, we take all of the categorical vectors for each event and project them down into a lower dimensional vector. So if you recall, we had numeric features from the events themselves as well. And these numeric factors are concatenated to the resulting lower dimensional categorical vector to form the sequence of input feature vectors to the RNN. Once the RNN sees the sequence of input feature vectors, it produces a probability representing whether it considers the sequence fraudulent. And we compute the loss of the output against the target label and back propagate all the way through the embeddings. So the embedding vectors will definitely kind of move around and eventually represent each categorical value within this fraud space. There's another interesting challenge that we actually came across, which was dealing with a large number of categorical values. Since we're jointly learning embeddings, having a very high cardinality set of embeddings increases the model capacity. And this can lead to over fitting quite a bit. But the good news is there are very many different ways to manage over fitting. And I'll take you through a couple of the solutions that we found useful. In the case of having too many distinct categorical values for any specific event, when we dove into the data itself, we realized that there are many cases where there were a long tail of values which occurred very sparingly. And so we were able to encode the long tail as a single categorical value, forcing the model to learn one embedding for all of these long tail keys. We also apply heavy regularization and use drop out aggressively, regularization like weight decay in the loss function and so on. After all of these tricks, we were able to get the model to perform well without actually memorizing the data set. So we've seen some pretty incredible predictive accuracy improvements with this new modeling an approach in both offline back testing as well as scoring on an hourly basis. Given the simplicity of the transformation of these features, we're actually optimistic to see that these results will hold in real time online setting as well. But, you know, fraud is an ever moving target. And so since fraud prevention is an adversarial problem, we expect that the behavioral patterns related to fraud will continue to evolve over time. And we will naturally continue to evolve our approach and improve how we manage fraud prevention effectively. In the near and long term, we have a

lot of different future challenges for this work. And it spans sort of systems and modeling work as well. Some of the things that we want to talk about-- some of the things that we want to enable are explanations for this particular model. So when our users care very deeply about why the model made a certain prediction, we should be able to let them know why this is actually the case. Other things that we're hoping to try are more or newer modeling techniques like applying attention mechanisms that have been successful in other contexts, being able to train and serve on data sets that don't fit in memory. And so this kind of does span the gamut of systems and modeling work. So if I can leave you with a few takeaways from this talk. In addition to feature engineering, it's worthwhile investing and investigating custom model architectures. So if you design a model that is well suited for your machine learning task and appropriate for your data or perhaps apply existing model architectures to the task at hand, that's actually a worthwhile investment, as worthwhile as it is to feature engineer. In our case, by combining RNNs and minimally transforming metadata from real-world events, we were able to shift the computational lifting from data engines where complicated features were being aggregated to the modeling hardware and thus reduce the complexity of our feature transformation pipelines while improving our predictive accuracy. And finally moving towards investing in modeling engineering. Model engineering brings with a sort of mindset shift that when we experiment moving from figuring out which complicated transformations to apply to features, it kind of goes towards instead to figuring out which high value raw signals we can use to capture the most information about the machine learning task. I'll open it up for questions since that was a fairly fast talk. Yeah, go ahead. Thanks, Pam. I have two questions. The first one is your labeling a multivariant or is it a binary one? Well in these cases-- so in the case for fraud prevention, we actually have just a binary class classifier at the bottom of the RNN. But there is no reason why it can't extend towards a multi class label. The second question is who actually regulated it for you? Um, that's a great question. So we have a-- we have events that come in on Stripe that allow us to understand that if a specific payment has been processed whether or not it's also been disputed. And so that allows us to understand whether or not it was-- and why it was disputed. And so that allows us to understand whether or not it was associated with fraud. Cool. I had a question about-- I have two actually. The first was so this approach of a learning behavior automatically other than featuring an aggregate, how do you get around the problem of some the aggregate [INAUDIBLE] probably a good signal for fraud detection. Are you relying on the RNN to learn that normal like location for a person or are those still encoded as aggregate features? Yeah, that's a great question. So for features like that one, we actually use those as categorical features. And that ends up becoming a high cardinality categorical feature, right? So by-- Categorical feature of all locations? Yeah. And as much as-- the embedding vectors themselves actually will capture enough of the numeric value for that particular categorical feature, that it's able to learn how to represent those effectively. So you are able to completely trust the automated [INAUDIBLE] Yeah. My other question was what is the scale of positive labels you have here if you can talk about it? Sure. So order of magnitude is probably in the hundreds of thousands for positive labels. And we obviously have an imbalanced data set with most, as you would with most fraud cases. And so we try to down sample in intelligent ways, but also apply up weighting to the labels in the loss function itself. And do you have a sense that this would not have worked until you had some volume of positive labels? Well, we started-- so when we started training the model at the beginning, we actually started with a much smaller

data set to kind of speed up our iteration and overall experimentation. And when you start with a much smaller data set, what you want to do is kind of restrict and constrain the model capacity itself. And so you can make the RNN less deep, make the embeddings less wide, and kind of have it be-- have the amount of capacity that represents about half of the data set that you have. So the total number of neurons that you have in your network is equivalent to like half the size of your data set. And that generally kind of is a good rule of thumb to get started with. Yeah. Thanks for your talk. I have two questions. The first is, like what's a range of the number of RNNs that you have in this sequence? Like you know, in one of your first slides you showed how you go from one RNN or a sequence-- the sequential data, yeah, is this like hundreds of events or tens of events? Yeah, so we actually-- I'm using RNNs here as a general class of models. At the final detail, we're actually using an LSTM to train this. And we've truncated our sort of event sequences to about 800 or so. And it's able to kind of learn anywhere between like 5 to 800. So we start with about 3, I think, on one end and cap it at about 800. Yeah. And my second question is, how do you deal with the delay of when this is a question about labeling your data and the delayed labels because of disputes coming in maybe many days, weeks after the occurrence? Yeah, so this is a normal challenge that we're still working on. And some normal tricks that we can kind of try and apply in the future are updating more recent data pieces that we see. So there is a distribution of like how many of the disputes actually come in over time, right? So most of them will kind of come in within the first n days. And then in the next chunk, the rest of them will kind of come in. And so you can kind of bias it towards learning more from the more recent samples, if that helps and that's something that we're hoping to try in the future. But right now what we do is wait the period of time and train from a little bit further back. Yeah. I was just wondering about the embeddings. Do you train those within the model or are they some how trained externally? So these particular embeddings that I'm talking about are trained within the model. There is an another piece of future work that we're hoping to do which is pull in additional embeddings that we've trained separately from model that kind of learn a very different set of representations for given categorical values and kind of use them directly within the model as well. And then about kind of the size of the embeddings that you use, would you restrict those or let the regularization take care of that, just give it a huge embedding space and let the regularization do the work? Yeah, so I'd say it's a combination of both. We've kind of constrained our embeddings to be fairly small, somewhere within the 32 to 128 dimension range, depending on the actual data set that we're looking at. And our drop out values tend to be somewhere between 0.25 and 0.75. That's like the sort of most useful regularization technique that we have in the model itself. OK, so I want to know how imbalanced it is for your data? And you mentioned that you used a down sampling, so how this works? And another thing that after you train your model and put into the production, how you explain your model, how to explain your readouts? Yeah, so explanation models are sort of in the future work part of it. And we're going to be investigating things like LIME and other kinds of approaches. In sort of the applied setting when I was trying to actually figure out what the model was learning and whether or not it was learning enough and whether the model architecture needed to change, what I found was that plotting heat map probabilities over each event gave me a sense of what the model was learning and whether or not it was maybe forgetting too much towards the end or not learning enough towards the beginning. And I was able to modify the architecture of that way. And your first question was,

how do we do down sampling. So this varies depending on the task at hand. And for some of our tasks, we down sample by using specific heuristics that we already know about, sort of how the data distribution looks like and how to separate out or how to pull out obviously good things and not down sample such that we lose a lot of information in that process. Does that answer your question? Are you down sampling [INAUDIBLE] Yes. Yes. Negative being good users and good actors. [INAUDIBLE] No. So for some of the data sets I've trained this with, I kind of kept it to a 3 to 1 ratio for. Others have kept it to a 9 to 1 ratio. And both of those tend to actually kind of work pretty well. I haven't extended it beyond that because we can't fit more than that on a given machine. But we trained this on a GPU. And so, you know, it kind of has a limited amount of host memory. Yeah, go ahead. So I have a question about how you define your sequences for training. Do you take the transactions within a given month and then work backwards from those transactions to define a sequence? We take the transactions within a specific time period and work backwards for that. And then for our testing data, we do sort of a look back window for n events from the transaction that we're trying to predict for. Thanks. Can you tell us something about how in the business context you act upon the outputs of the model? I mean, a single transaction might just be declined. But if you've learned anything from your experience in actually actioning on the results of the model that has helped you to feedback and tune model in the business context? That's a great question. Um, I would say that given how nascent this project has been, we haven't-- I think we've mostly just seen that it's improved our ability to make these predictions so much that there hasn't been as much investigation and analysis on taking the learnings from this iteration and feeding it back into the actual model architecture itself or how we carve up our data set or how we change our sequences of events. But that certainly is something that we will be doing in the future. Can you share how you convert event to categorical data types? So you have bunch, many events, but you need to convert into finite number of categorical card values. So how did you define categorical card type and convert them? Yeah, so the way that the sort of-- so typically when you kind of take categorical features and use them with a decision tree or something like that, you might one hot encode it or you might, you know, use some sort of rate over time to encode that as a numeric feature, right? And the way that we look at it is taking these sort of actual keyed-- the actual value that a categorical feature is associated with. So let's take a look at this example again. We have the event type being sign up. And the string sign up being a specific categorical value that we can then try and learn an embedding for. That leads to all the distinct event types will actually have their own embedding vectors associated with them. And so we'll take every single distinct categorical value and kind of create a large vocabulary that is representing one categorical value to a numeric index. And that numeric index them is used to look up an embedding vector that is trained with the model. You have to support this how do you know that [INAUDIBLE] Sorry, when you have two different events and-- How do you know that the [INAUDIBLE] Well, they'll have the same key. So if they're both sign up-- so if a user has logged in and logged out and logged in again and the event type was log in, log out, log in, the two log in events will have the same vectors associated with them. But the metadata associated with that specific log in event will be different, right? So it'll have a different timestamp. It might have-- maybe you've logged in from your mobile device, as opposed to your laptop. And those will also have other pieces of metadata associated with each of those events that have different categorical values. But the log in key itself will map to the same embedding vector because

they're the same string. Hi, I have a question, again, sorry. It's about how you get an unbiased data set to train on if you're catching fraud and having more declines and less is getting through and so there are fewer disputes, your disputes over time will keep changing, I guess. Is that something that you are concerned with or do you just keep retraining and-- I don't know, does that question make sense? That's a problem that I've encountered which leads to bias. Yeah. So this is one of the reasons why I think potentially switching towards a multiclass as opposed to a binary classifier might help us in the future. As we sort of get better and better at predicting that something's going to be fraudulent, we'll block it more aggressively. And that's a specific type of label that we should be trying to predict that's different from the issuer believing that it's going to be fraudulent. But right now we kind of just treat those both as the same and try and make sure that whatever our previous model has captured are new models also able to capture as well and furthermore be able to capture more of that than what the previous model was able to capture. OK. Just to clarify then, are you-- so for your labeling for your future data sets, are you considering something that your current system blocks, you would call that fraud for your future system to learn? Yes. OK. And do you have any one investigator review to make sure those are correct? Yeah, we have a holdout data set that allows sort of some amount of stuff to get through, just so we have some validation to actually check our ability to evaluate these transactions specifically. I have a question about how you create a training data set. On the fraud side normally, fraud case is very little compared to your lawful transaction. So do you use all of them or do you-- how do you sample for your training data set? What's your ratio? How do you select those trained data set? Yeah. So we use all of the positive labels for sure, anything that's labeled positively, we will definitely include an our data set. And for the negative examples, we actually down sample those aggressively. So some of the cases that we've actually tried are a 1 to 3 ratio, a 1 to 9 ratio. And as I mentioned before, I would expand beyond 1 to 9, but we can't fit any more into memory. So what's the maximum you normally pick? What's the ratio, like 1 to-- 1 to 9, I think. 100. 1 to 9. 1 to 9, OK. Yeah. I dropping my laptop on the here. So I have two other questions. One was about that with the sampling. Given that you are sampling so aggressively, how do you back out the true probabilities? Is that a issue? Most of our true probabilities are computed based on the actual holdout data set that we allow for sort of some stuff to get through. And so that's actually where we get a lot of our information from for how well the model is actually performing in the real world and relative to sort of the background of rate of fraud that we expect to see. Yeah. Does that end up being a problem, though, for like the more rare types, like you mentioned there's like multiple classes of fraud? There's some more rare types. Yeah. Um, I would think-- I would say that for the more rare cases of fraudulent patterns-- so given the sort of split between negative and positive examples being so unbalanced, it is actually more concerning to me that we don't see all the kinds of patterns of good behavior than it is that we won't capture all the kinds of fraudulent behavior sequences that we are able to see. The further back we go in time, the more we sort of learn about how users have behaved in the past and over time fraudulently. But it is a concern that we're not capturing as much as we possibly could from the actual good behaviors and so getting sort of false positives is probably a higher concern than anything else. I had one other. But if someone else has a question, I'll let them go. Yeah. How do you deal with the sequences that you know that are strong signal for fraud, but you didn't see it in the training set? How do I deal with sequences where I didn't see it in the training set, but there are strong signals for fraud? Well, I

think what we've seen so far is that if we look back further in time, we capture a lot more of the actual fraudulent behavior patterns. They're kind of-- going past, you know, 3 years or something like that, fraud behavioral patterns have actually changed over time so much that those don't inform our model as much as sort of looking back within last year. So when we train it-- when we actually create our training data set, we do look back pretty far in time. And at a certain point we will eventually-- once we get to the point where we're training on multiple machines, I expect that we'll fully look at like all of the fraud we've seen before. So you mention that you have a very large vocabulary. And then it becomes a smaller embedding. And then you still project it. What's the sort of scale of each of those? So I believe for one of our models, we have something like 300,000 embeddings. And the dimension of each embedding is around 32. We treat certain kinds of categorical features in a way that allows us to kind of even shrink that even further. But most of-- some of the-- a lot of our categorical features do get kind of turned into these long tail encoded as one value versus everything that we do care about that occurs frequently enough that we care about it. And that is all kind of projected down into yet another 32 dimensional feature vector. You can obviously kind of expand that if it's not capturing enough information. But right now that seems to work for our purposes. Yeah. You may have said this at the beginning, but what type of fraud are you focused on? First party fraud? Third party fraud? All fraud? Right, so this is-- we're applying this approach to tow different kinds of fraud that we really do care about. One of them is when a user is basically behaving fraudulently on legitimate businesses. And the other case is where we have users that come and sign up for legitimate businesses that are solely used for fraudulent behavior. Please join me in the round of applause for Pam Vagata. [APPLAUSE] Thank you.