Machine Learning Journal

Hey Siri: An On-device DNN-powered Voice Trigger for Apple’s Personal Assistant

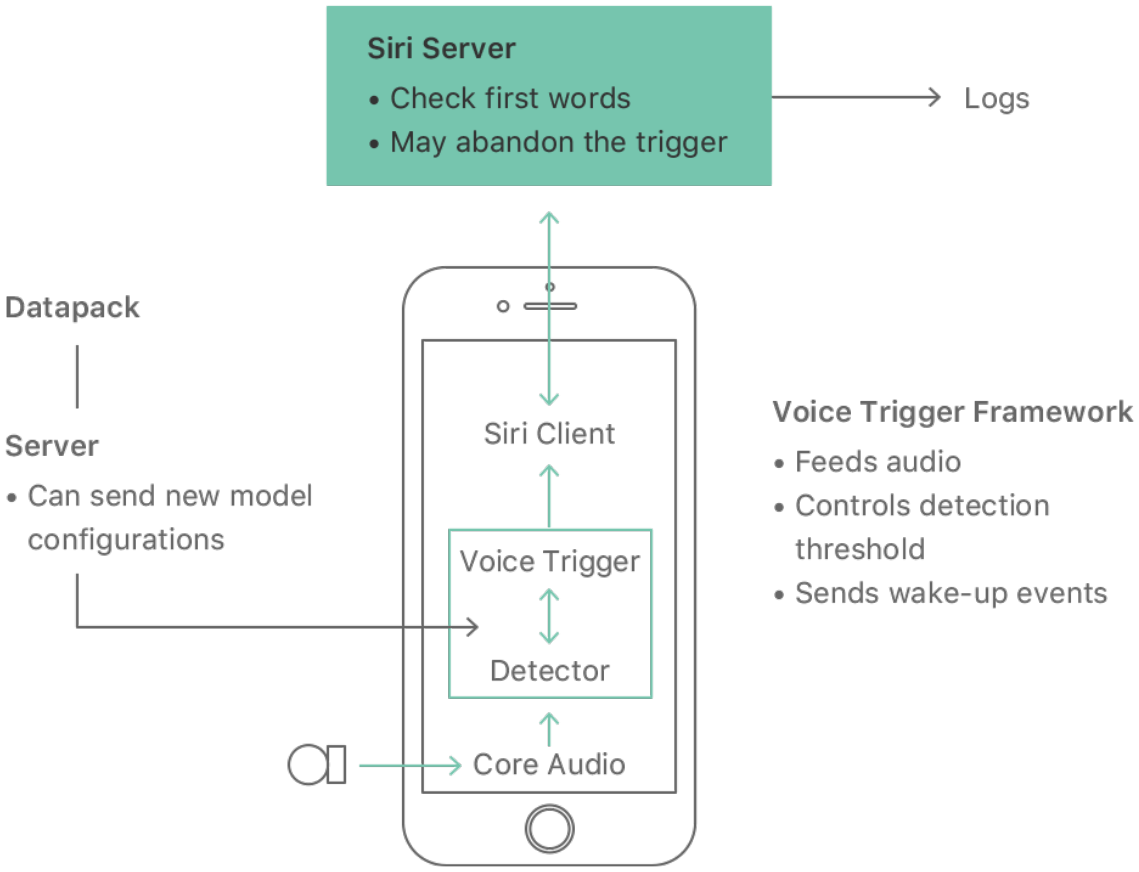
Vol. 1, Issue 6 · October 2017
by Siri Team

The “Hey Siri” feature allows users to invoke Siri hands-free. A very small speech recognizer runs all the time and listens for just those two words. When it detects “Hey Siri”, the rest of Siri parses the following speech as a command or query. The “Hey Siri” detector uses a Deep Neural Network (DNN) to convert the acoustic pattern of your voice at each instant into a probability distribution over speech sounds. It then uses a temporal integration process to compute a confidence score that the phrase you uttered was “Hey Siri”. If the score is high enough, Siri wakes up. This article takes a look at the underlying technology. It is aimed primarily at readers who know something of machine learning but less about speech recognition.

Hands-Free Access to Siri

To get Siri’s help, say “Hey Siri”. No need to press a button as “Hey Siri” makes Siri hands-free. It seems simple, but quite a lot goes on behind the scenes to wake up Siri quickly and efficiently. Hardware, software, and Internet services work seamlessly together to provide a great experience.

Figure 1. The Hey Siri flow on iPhone



Being able to use Siri without pressing buttons is particularly useful when hands are busy, such as when cooking or driving, or when using the Apple Watch. As Figure 1 shows, the whole system has several parts. Most of the implementation of Siri is "in the Cloud", including the main automatic speech recognition, the natural language interpretation and the various information services. There are also servers that can provide updates to the acoustic models used by the detector. This article concentrates on the part that runs on your local device, such as an iPhone or Apple Watch. In particular, it focusses on the detector: a specialized speech recognizer which is always listening just for its wake-up phrase (on a recent iPhone with the "Hey Siri" feature enabled).

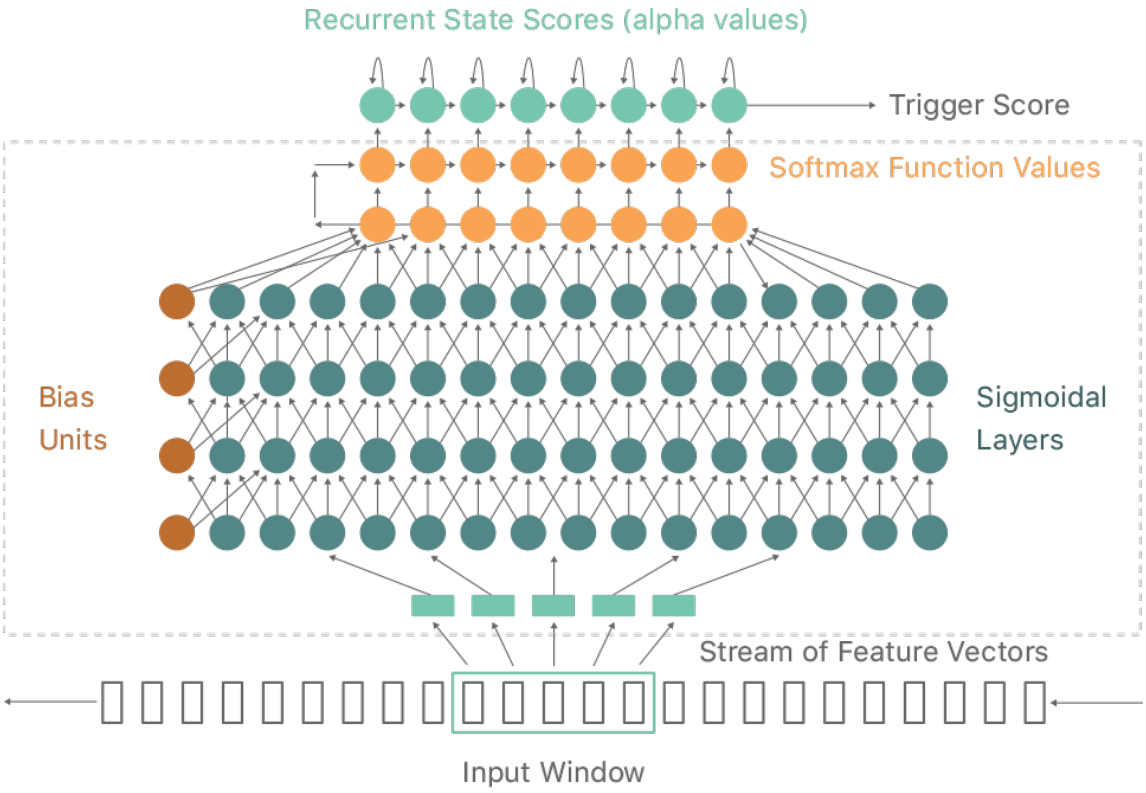
The Detector: Listening for "Hey Siri"

The microphone in an iPhone or Apple Watch turns your voice into a stream of instantaneous waveform samples, at a rate of 16000 per second. A spectrum analysis stage converts the waveform sample stream to a sequence of frames, each describing the sound spectrum of approximately 0.01 sec. About twenty of these frames at a time (0.2 sec of audio) are fed to the acoustic model, a Deep Neural Network (DNN) which converts each

of these acoustic patterns into a probability distribution over a set of speech sound classes: those used in the "Hey Siri" phrase, plus silence and other speech, for a total of about 20 sound classes. See Figure 2.

The DNN consists mostly of matrix multiplications and logistic nonlinearities. Each "hidden" layer is an intermediate representation discovered by the DNN during its training to convert the filter bank inputs to sound classes. The final nonlinearity is essentially a Softmax function (a.k.a. a general logistic or normalized exponential), but since we want log probabilities the actual math is somewhat simpler.

Figure 2. The Deep Neural Network used to detect "Hey Siri." The hidden layers are actually fully connected. The top layer performs temporal integration. The actual DNN is indicated by the dashed box.



We choose the number of units in each hidden layer of the DNN to fit the computational resources available when the "Hey Siri" detector runs. Networks we use typically have five hidden layers, all the same size: 32, 128, or 192 units depending on the memory and power constraints. On iPhone we use two networks—one for initial detection and another as a secondary checker. The initial detector uses fewer units than the secondary checker.

The output of the acoustic model provides a distribution of scores over

phonetic classes for every frame. A phonetic class is typically something like "the first part of an /s/ preceded by a high front vowel and followed by a front vowel."

We want to detect "Hey Siri" if the outputs of the acoustic model are high in the right sequence for the target phrase. To produce a single score for each frame we accumulate those local values in a valid sequence over time. This is indicated in the final (top) layer of Figure 2 as a recurrent network with connections to the same unit and the next in sequence. Inside each unit there is a maximum operation and an add:

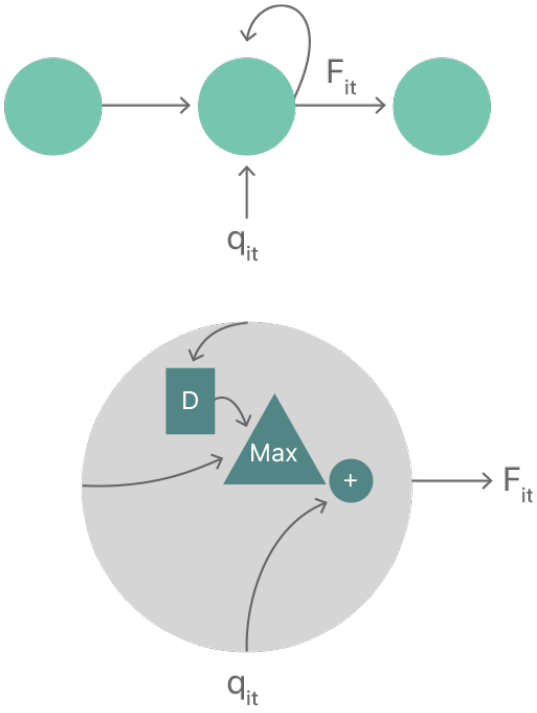
$$F_{i,t} = \max\{s_i + F_{i,t-1}, m_{i-1} + F_{i-1,t-1}\} + q_{i,t}$$

where

- $F_{i,t}$ is the accumulated score for state i of the model
- $q_{i,t}$ is the output of the acoustic model—the log score for the phonetic class associated with the *ith* state given the acoustic pattern around time t
- s_i is a cost associated with staying in state i
- m_i is a cost for moving on from state i

Both s_i and m_i are based on analysis of durations of segments with the relevant labels in the training data. (This procedure is an application of dynamic programming, and can be derived based on ideas about Hidden Markov Models—HMMs.)

Figure 3. Visual depiction of the equation

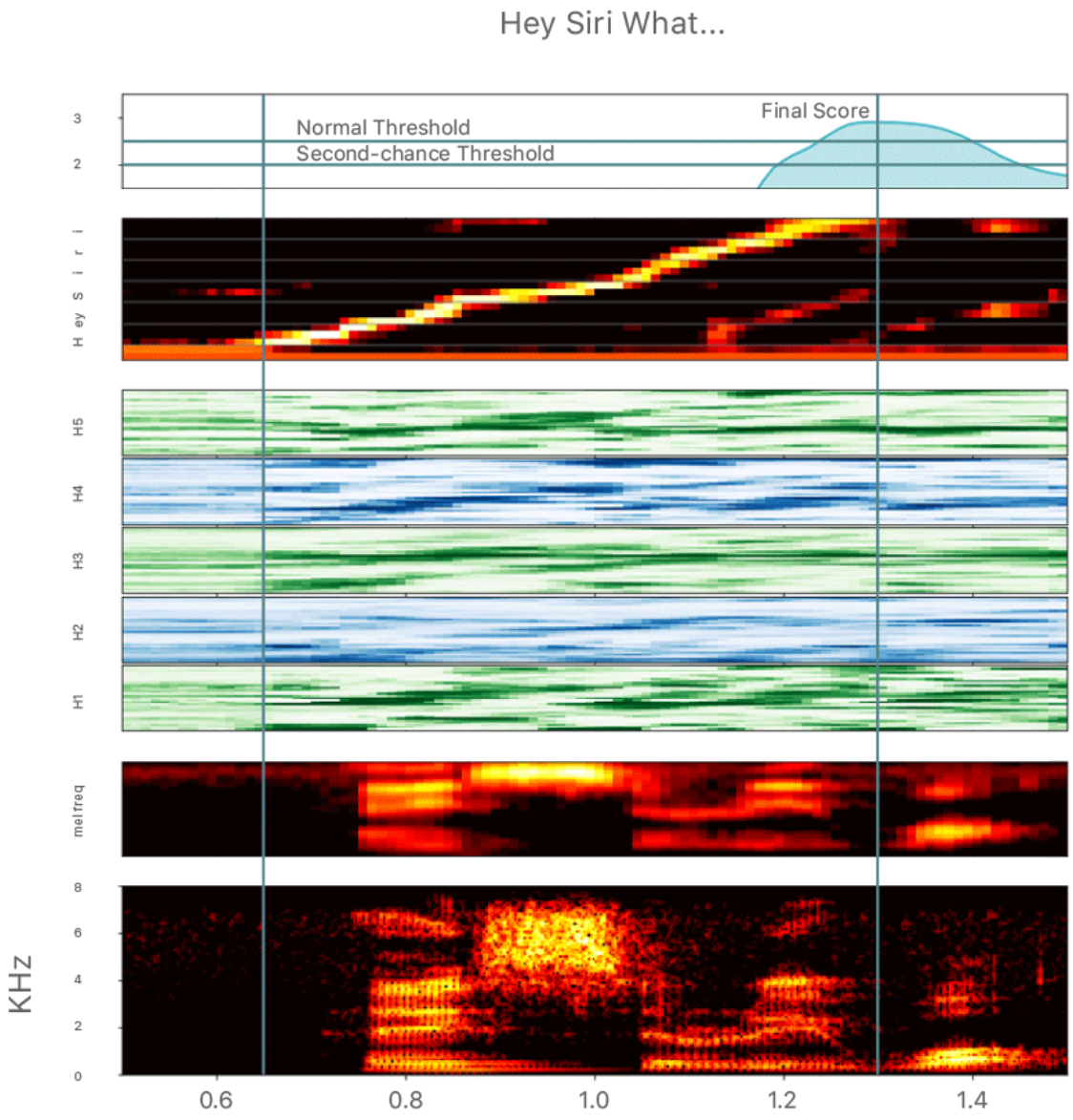


Each accumulated score $F_{i,t}$ is associated with a labelling of previous frames with states, as given by the sequence of decisions by the maximum operation. The final score at each frame is $F_{i,t}$, where the last state of the phrase is state I and there are N frames in the sequence of frames leading to that score. (N could be found by tracing back through the sequence of max decisions, but is actually done by propagating forwards the number of frames since the path entered the first state of the phrase.)

Almost all the computation in the "Hey Siri" detector is in the acoustic model. The temporal integration computation is relatively cheap, so we disregard it when assessing size or computational resources.

You may get a better idea of how the detector works by looking at Figure 4, which shows the acoustic signal at various stages, assuming that we are using the smallest DNN. At the very bottom is a spectrogram of the waveform from the microphone. In this case, someone is saying "Hey Siri what …" The brighter parts are the loudest parts of the phrase. The Hey Siri pattern is between the vertical blue lines.

Figure 4. The acoustic pattern as it moves through the detector



The second horizontal strip up from the bottom shows the result of analyzing the same waveform with a mel filter bank, which gives weight to frequencies based on perceptual measurements. This conversion also smooths out the detail that is visible in the spectrogram and due to the fine-structure of the excitation of the vocal tract: either random, as in the /s/, or periodic, seen here as vertical striations.

The alternating green and blue horizontal strips labelled H1 to H5 show the numerical values (activations) of the units in each of the five hidden layers. The 32 hidden units in each layer have been arranged for this figure so as to put units with similar outputs together.

The next strip up (with the yellow diagonal) shows the output of the acoustic model. At each frame there is one output for each position in the phrase, plus others for silence and other speech sounds. The final score, shown at the top, is obtained by adding up the local scores along the bright diagonal

according to Equation 1. Note that the score rises to a peak just after the whole phrase enters the system.

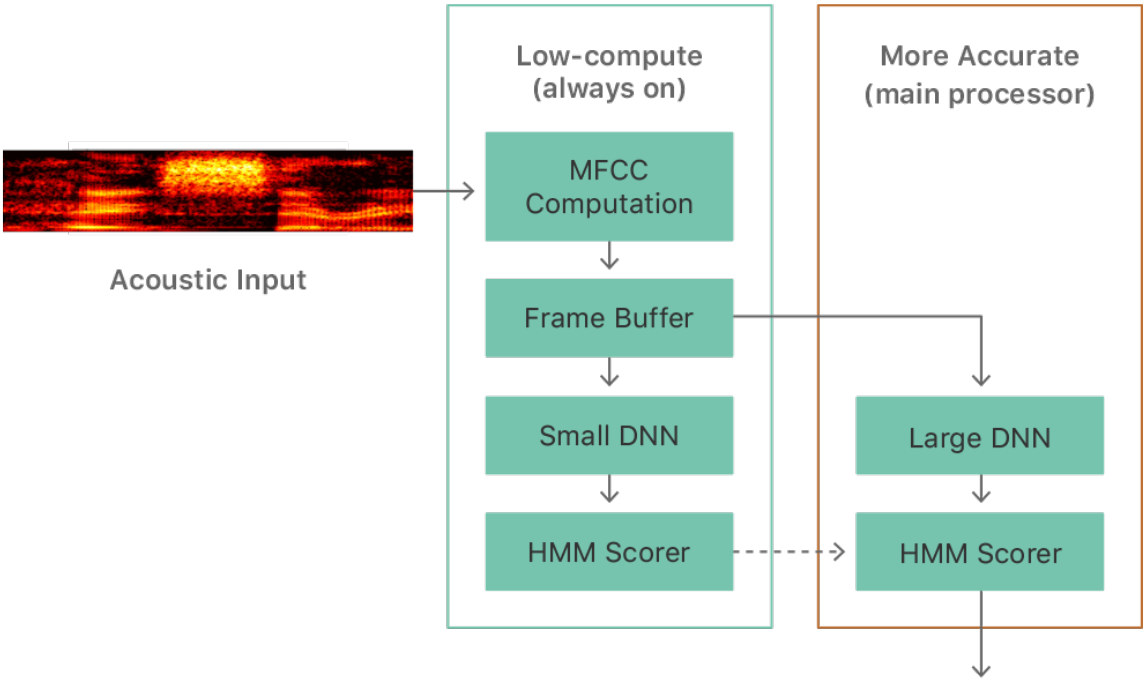
We compare the score with a threshold to decide whether to activate Siri. In fact the threshold is not a fixed value. We built in some flexibility to make it easier to activate Siri in difficult conditions while not significantly increasing the number of false activations. There is a primary, or normal threshold, and a lower threshold that does not normally trigger Siri. If the score exceeds the lower threshold but not the upper threshold, then it may be that we missed a genuine "Hey Siri" event. When the score is in this range, the system enters a more sensitive state for a few seconds, so that if the user repeats the phrase, even without making more effort, then Siri triggers. This second-chance mechanism improves the usability of the system significantly, without increasing the false alarm rate too much because it is only in this extra-sensitive state for a short time. (We discuss testing and tuning for accuracy later.)

Responsiveness and Power: Two Pass Detection

The "Hey Siri" detector not only has to be accurate, but it needs to be fast and not have a significant effect on battery life. We also need to minimize memory use and processor demand—particularly peak processor demand.

To avoid running the main processor all day just to listen for the trigger phrase, the iPhone's Always On Processor (AOP) (a small, low-power auxiliary processor, that is, the embedded Motion Coprocessor) has access to the microphone signal (on 6S and later). We use a small proportion of the AOP's limited processing power to run a detector with a small version of the acoustic model (DNN). When the score exceeds a threshold the motion coprocessor wakes up the main processor, which analyzes the signal using a larger DNN. In the first versions with AOP support, the first detector used a DNN with 5 layers of 32 hidden units and the second detector had 5 layers of 192 hidden units.

Figure 5. Two-pass detection



Apple Watch presents some special challenges because of the much smaller battery. Apple Watch uses a single-pass "Hey Siri" detector with an acoustic model intermediate in size between those used for the first and second passes on other iOS devices. The "Hey Siri" detector runs only when the watch motion coprocessor detects a wrist raise gesture, which turns the screen on. At that point there is a lot for WatchOS to do—power up, prepare the screen, etc.—so the system allocates "Hey Siri" only a small proportion (~5%) of the rather limited compute budget. It is a challenge to start audio capture in time to catch the start of the trigger phrase, so we make allowances for possible truncation in the way that we initialize the detector.

"Hey Siri" Personalized

We designed the always-on "Hey Siri" detector to respond whenever anyone in the vicinity says the trigger phrase. To reduce the annoyance of false triggers, we invite the user to go through a short enrollment session. During enrollment, the user says five phrases that each begin with "Hey Siri." We save these examples on the device.

We compare any possible new "Hey Siri" utterance with the stored examples as follows. The (second-pass) detector produces timing information that is used to convert the acoustic pattern into a fixed-length vector, by taking the average over the frames aligned to each state. A separate, specially trained

DNN transforms this vector into a "speaker space" where, by design, patterns from the same speaker tend to be close, whereas patterns from different speakers tend to be further apart. We compare the distances to the reference patterns created during enrollment with another threshold to decide whether the sound that triggered the detector is likely to be "Hey Siri" spoken by the enrolled user.

This process not only reduces the probability that "Hey Siri" spoken by another person will trigger the iPhone, but also reduces the rate at which other, similar-sounding phrases trigger Siri.

Further Checks

If the various stages on the iPhone pass it on, the waveform arrives at the Siri server. If the main speech recognizer hears it as something other than "Hey Siri" (for example "Hey Seriously") then the server sends a cancellation signal to the phone to put it back to sleep, as indicated in Fig 1. On some systems we run a cut-down version of the main recognizer on the device to provide an extra check earlier.

The Acoustic Model: Training

The DNN acoustic model is at the heart of the "Hey Siri" detector. So let's take a look at how we trained it. Well before there was a Hey Siri feature, a small proportion of users would say "Hey Siri" at the start of a request, having started by pressing the button. We used such "Hey Siri" utterances for the initial training set for the US English detector model. We also included general speech examples, as used for training the main speech recognizer. In both cases, we used automatic transcription on the training phrases. Siri team members checked a subset of the transcriptions for accuracy.

We created a language-specific phonetic specification of the "Hey Siri" phrase. In US English, we had two variants, with different first vowels in "Siri"—one as in "serious" and the other as in "Syria." We also tried to cope with a short break between the two words, especially as the phrase is often written with a comma: "Hey, Siri." Each phonetic symbol results in three speech sound classes (beginning, middle and end) each of which has its own output from the acoustic model.

We used a corpus of speech to train the DNN for which the main Siri

recognizer provided a sound class label for each frame. There are thousands of sound classes used by the main recognizer, but only about twenty are needed to account for the target phrase (including an initial silence), and one large class class for everything else. The training process attempts to produce DNN outputs approaching 1 for frames that are labelled with the relevant states and phones, based only on the local sound pattern. The training process adjusts the weights using standard back-propagation and stochastic gradient descent. We have used a variety of neural network training software toolkits, including Theano, Tensorflow, and Kaldi.

This training process produces estimates of the probabilities of the phones and states given the local acoustic observations, but those estimates include the frequencies of the phones in the training set (the priors), which may be very uneven, and have little to do with the circumstances in which the detector will be used, so we compensate for the priors before the acoustic model outputs are used.

Training one model takes about a day, and there are usually a few models in training at any one time. We generally train three versions: a small model for the first pass on the motion coprocessor, a larger-size model for the second pass, and a medium-size model for Apple Watch.

"Hey Siri" works in all languages that Siri supports, but "Hey Siri" isn't necessarily the phrase that starts Siri listening. For instance, French-speaking users need to say "Dis Siri" while Korean-speaking users say "Siri ㅇㅑ" (Sounds like "Siri Ya.") In Russian it is "привет Siri " (Sounds like "Privet Siri"), and in Thai "หวัดดี Siri". (Sounds like "Wadi Siri".)

Testing and Tuning

An ideal detector would fire whenever the user says "Hey Siri," and not fire at other times. We describe the accuracy of the detector in terms of two kinds of error: firing at the wrong time, and failing to fire at the right time. The false-accept rate (FAR or false-alarm rate), is the number of false activations per hour (or mean hours between activations) and the false-reject rate (FRR) is the proportion of attempted activations that fail. (Note that the units we use to measure FAR are not the same as those we use for FRR. Even the dimensions are different. So there is no notion of an equal error rate.)

For a given model we can change the balance between the two kinds of error by changing the activation threshold. Figure 6 shows examples of this trade-off, for two sizes of early-development models. Changing the threshold

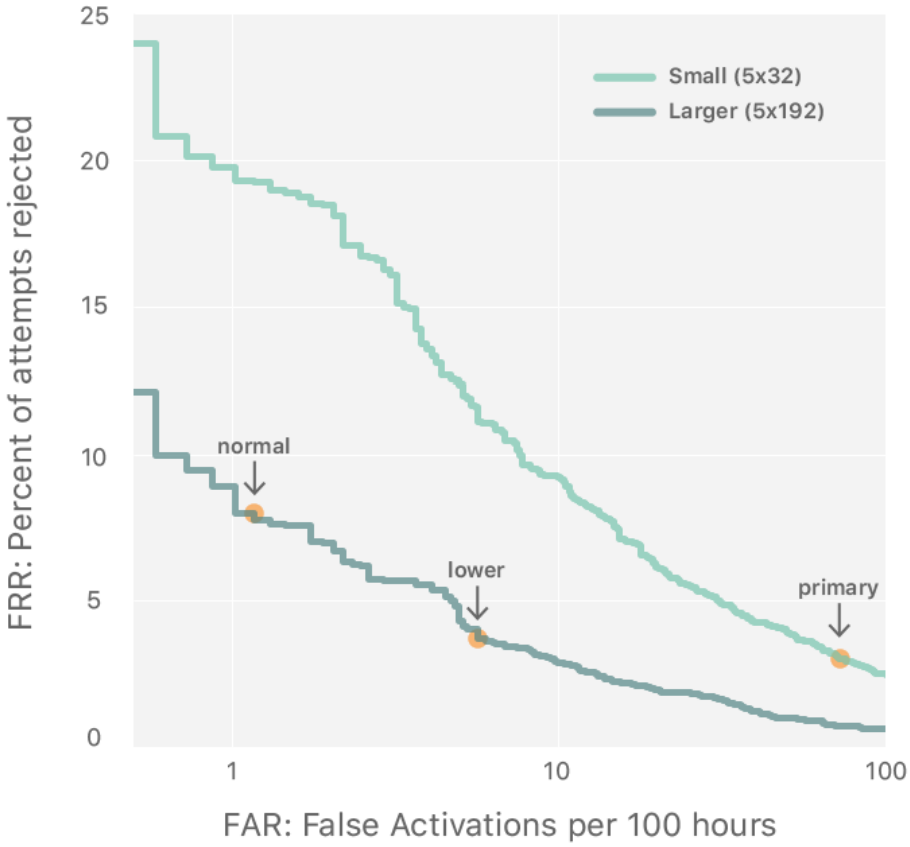
moves along the curve.

During development we try to estimate the accuracy of the system by using a large test set, which is quite expensive to collect and prepare, but essential. There is "positive" data and "negative" data. The "positive" data does contain the target phrase. You might think that we could use utterances picked up by the "Hey Siri" system, but the system doesn't capture the attempts that failed to trigger, and we want to improve the system to include as many of such failed attempts as possible.

At first we used the utterances of "Hey Siri" that some users said as they pressed the Home button, but these users are not attempting to catch Siri's attention, (the button does that) and the microphone is bound to be within arm's reach, whereas we also want "Hey Siri" to work across a room. We made recordings specially in various conditions, such as in the kitchen (both close and far), car, bedroom, and restaurant, by native speakers of each language.

We use the "negative" data to test for false activations (and false wakes). The data represent thousands of hours of recordings, from various sources, including podcasts and non-"Hey Siri" inputs to Siri in many languages, to represent both background sounds (especially speech) and the kinds of phrases that a user might say to another person. We need such a lot of data because we are trying to estimate false-alarm rates as low as one per week. (If there are any occurrences of the target phrase in the negative data we label them as such, so that we do not count responses to them as errors.)

Figure 6. Detector accuracy. Trade-offs against detection threshold for small and larger DNNs



Tuning is largely a matter of deciding what thresholds to use. In Figure 6, the two dots on the lower trade-off curve for the larger model show possible normal and second-chance thresholds. The operating point for the smaller (first-pass) model would be is at the right-hand side. These curves are just for the two stages of the detector, and do not include the personalized stage or subsequent checks.

While we are confident that models that appear to perform better on the test set probably are really better, it is quite difficult to convert offline test results into useful predictions of the experience of users. So in addition to the offline measurements described previously, we estimate false-alarm rates (when Siri turns on without the user saying "Hey Siri") and imposter-accept rates (when Siri turns on when someone other than the user who trained the detector says "Hey Siri") weekly by sampling from production data, on the latest iOS devices and Apple Watch. This does not give us rejection rates (when the system fails to respond to a valid "Hey Siri") but we can estimate rejection rates from the proportion of activations just above the threshold that are valid, and a sampling of just-below threshold events on devices carried by development staff.

We continually evaluate and improve "Hey Siri," and the model that powers

it, by training and testing using variations of the approach described here. We train in many different languages and test under a wide range of conditions.

Next time you say "Hey Siri" you may think of all that goes on to make responding to that phrase happen, but we hope that it "just works!"