Machine Learning Methods for Neural Data Analysis

Lecture 3: Spike Sorting by Deconvolution

Scott Linderman

STATS 220/320 (NBIO220, CS339N). Winter 2021.

Announcements

- **Happy** Inauguration **Day!**
- Lab 1 due at 11:59pm tomorrow.
  - Errata: please add `atol=1e-4` to assert in Problem 3b (log probability).
- My office hours are at 1pm today. Jaime's are at 5pm tomorrow.
- Course notes are updated with Ch 2.2.

Announcements

- **Happy** Inauguration **Day!**
- Lab 1 due at 11:59pm tomorrow.
  - Errata: please add `atol=1e-4` to assert in Problem 3b (log probability).
- My office hours are at 1pm today. Jaime's are at 5pm tomorrow.
- Course notes are updated with Ch 2.2.

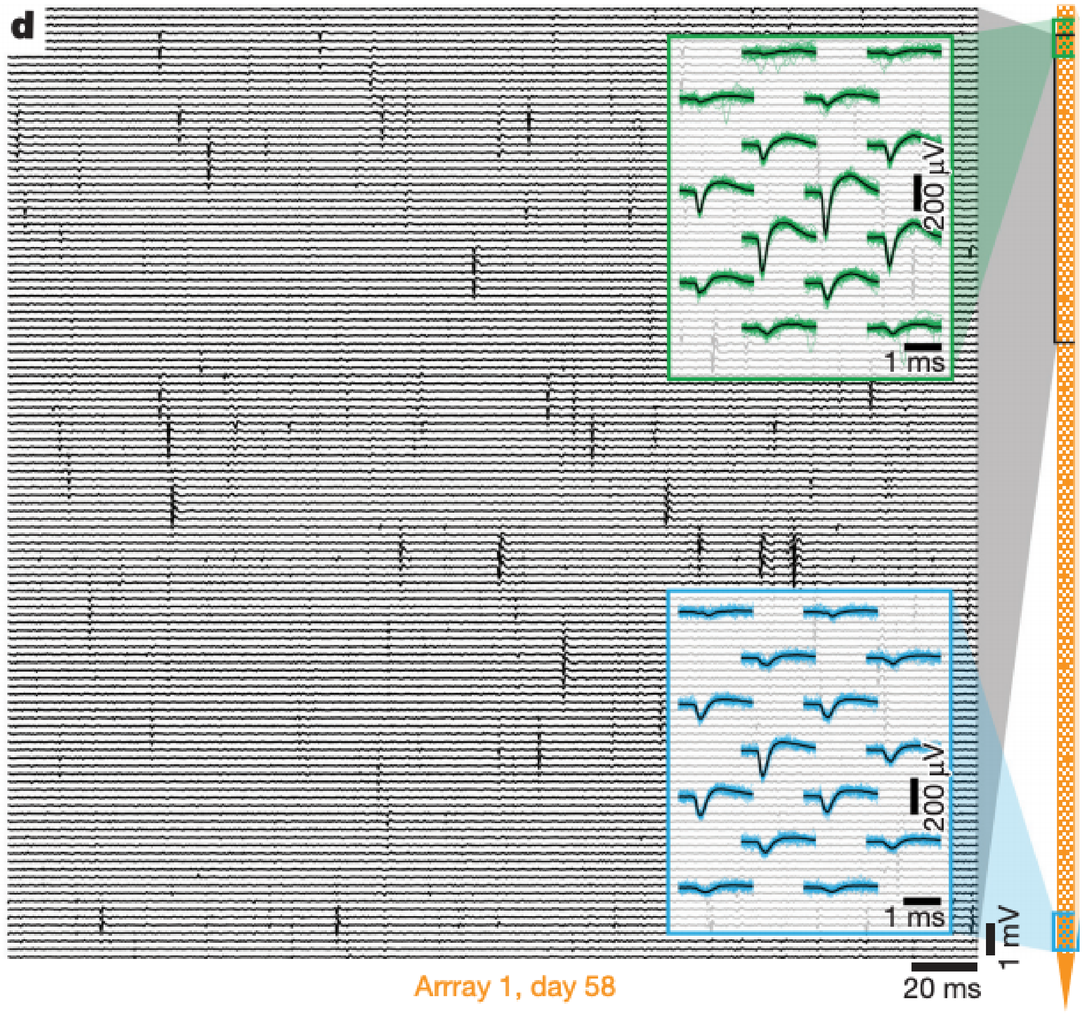
Agenda

1. Recap of the basic model and its limitations
2. Convolution and cross-correlation
3. Spike sorting by deconvolution
4. Maximum a posteriori inference

Recap

High-density probes (e.g. Neuropixels)

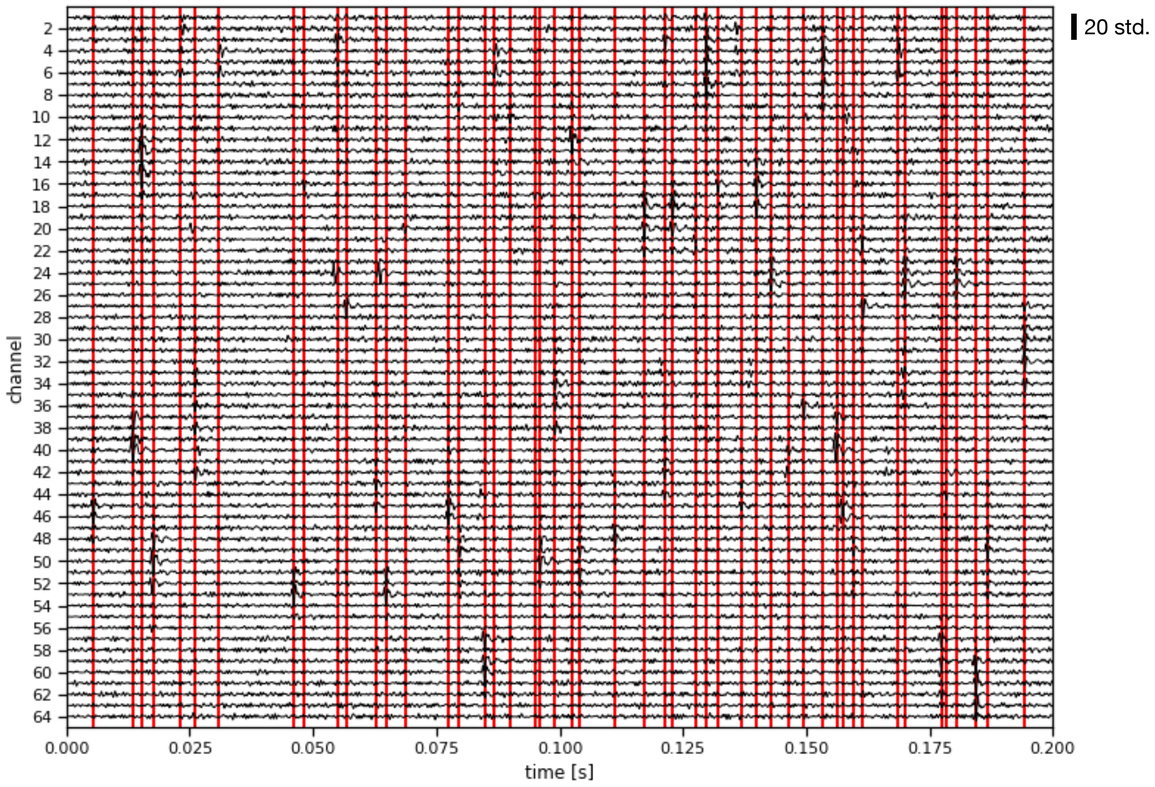
- The raw data is a **multidimensional time series** of **voltage measurements**, one for each recording site on the probe.
- When neurons near the probe fire an **action potential**, it registers a **spike in the voltage** on nearby channels.
- Typical recordings detect spikes from **O(100) neurons**.



Recap

Finding putative spikes

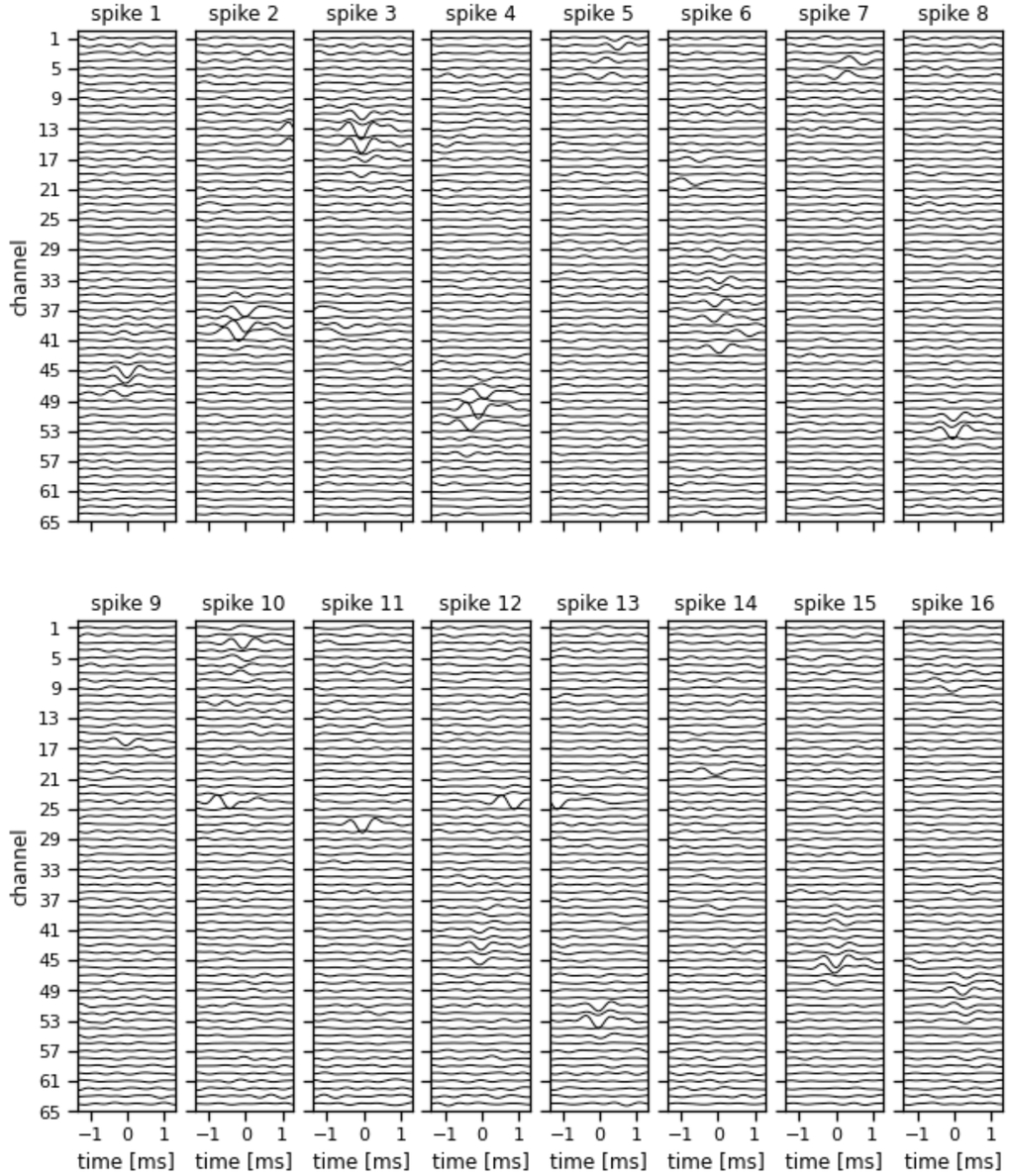
- In Lab 1, we used a simple “peak finding” algorithm to **find candidate spikes.**
- For example, we looked for negative peaks at least 4 standard deviations in amplitude on each channel.
- Then we combined across channels to get estimates of spike times.



Recap

Thought Experiment

- After finding putative spikes, we **extracted short windows** (2-3ms) around each one.
- **Thought experiment:** what is the probability of no other spikes occurring in the same 2ms window? Let s_{nt} denote number of spikes by neuron n in window $[t, t + \Delta t)$ and assume $s_{nt} \sim \text{Po}(\lambda \Delta t)$. Assume $N = 100$ independent neurons with rates of $\lambda = 10$ spikes/sec and windows of size $\Delta t = 2$ms.



Recap

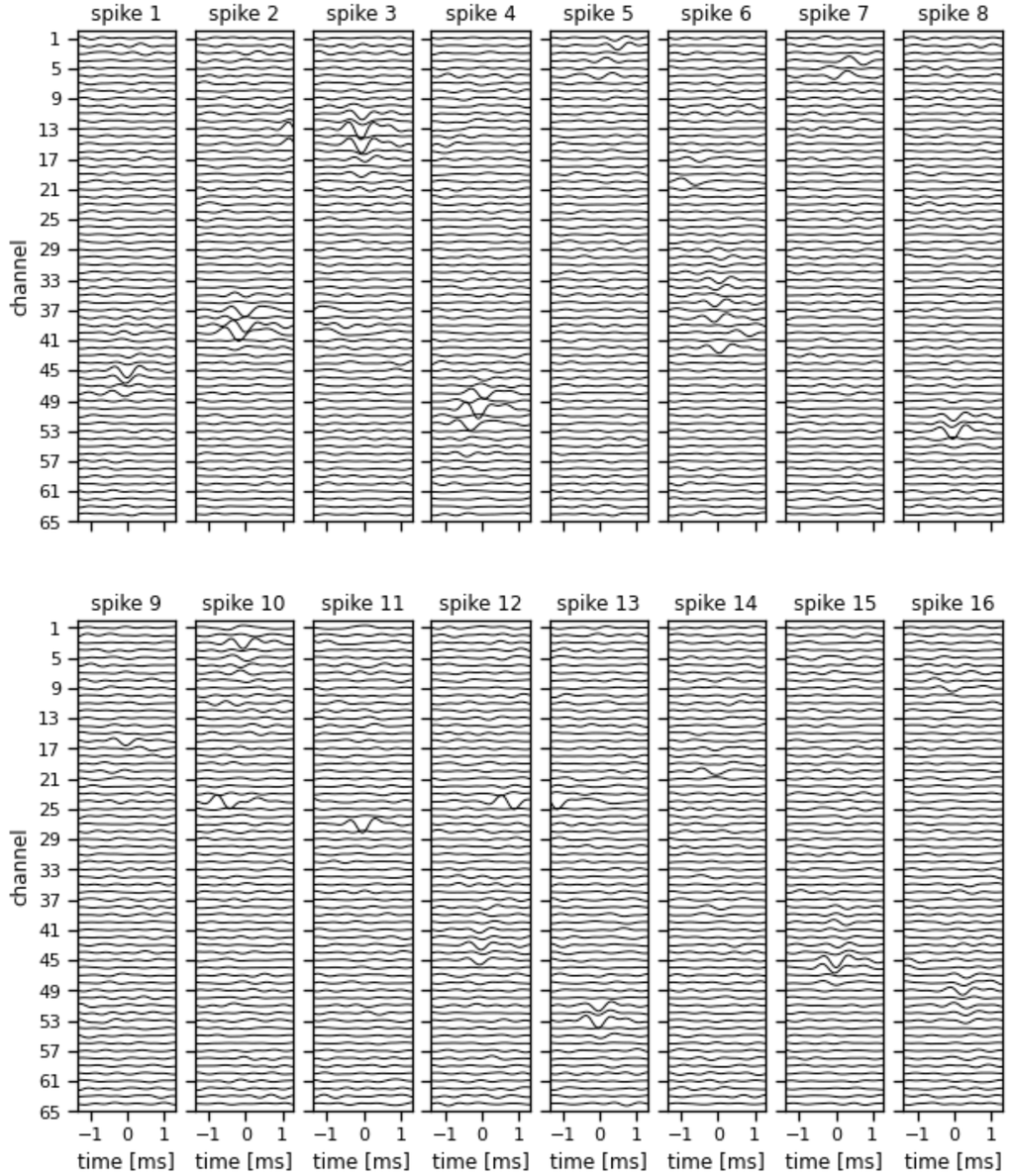
Thought Experiment

- After finding putative spikes, we **extracted short windows** (2-3ms) around each one.
- **Thought experiment:** what is the probability of no other spikes occurring in the same 2ms window? Let s_{nt} denote number of spikes by neuron n in window $[t, t + \Delta t)$ and assume $s_{nt} \sim \text{Po}(\lambda \Delta t)$. Assume $N = 100$ independent neurons with rates of $\lambda = 10$ spikes/sec and windows of size $\Delta t = 2$ms.
- Let $s_t = \sum_{m \neq n} s_{mt}$ denote number of spikes on all but neuron n.

  Since the sum of independent Poisson r.v.'s is Poisson, the number of spikes on other neurons is distributed as,

  $$s_t \sim \text{Po}((N-1)\lambda \Delta t).$$

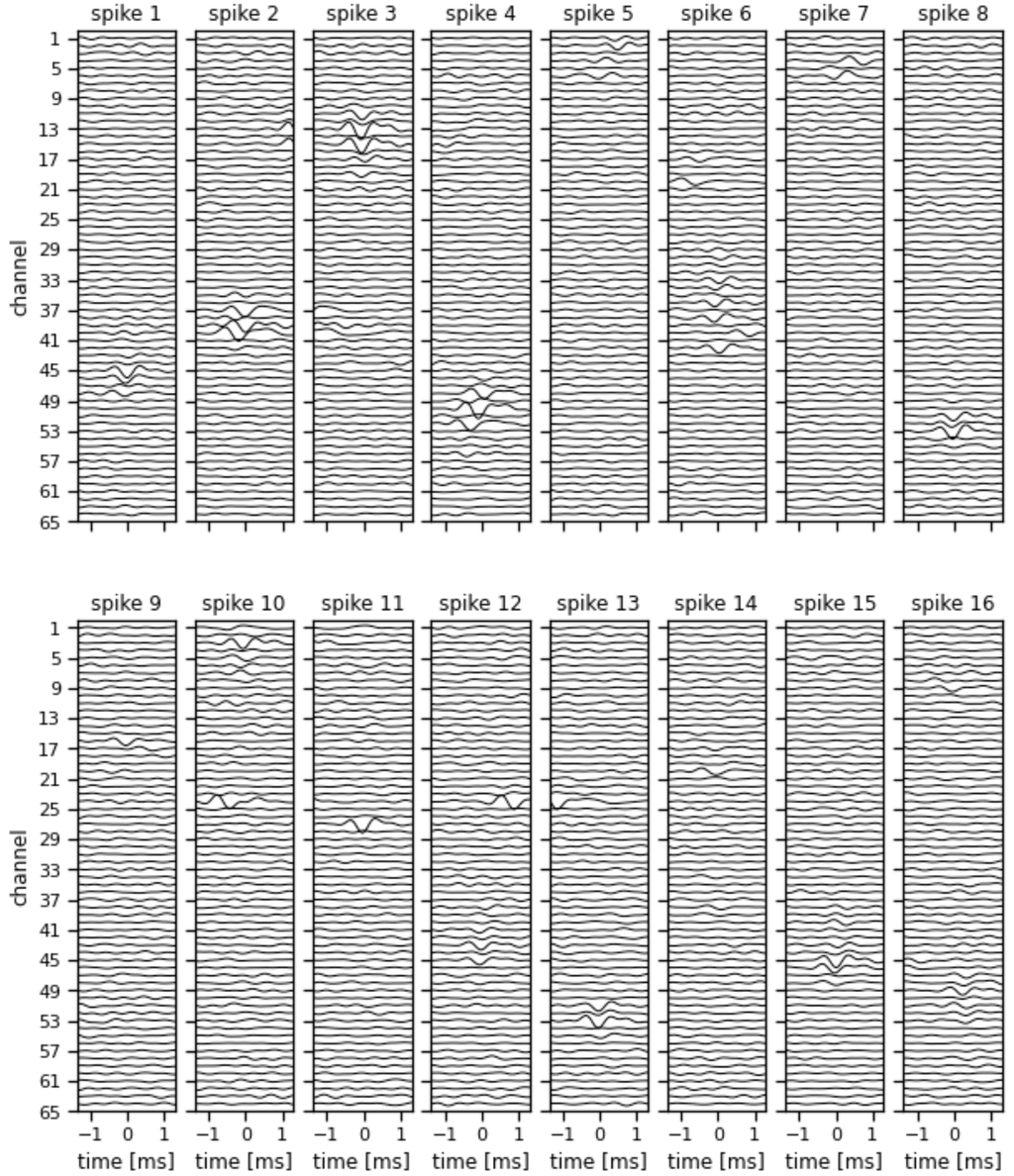
  The probability of seeing zero other spikes is $\Pr(s_t = 0) = e^{-(N-1)\lambda \Delta t}$, exponentially decreasing in the number of neurons. For the parameters above, that's $\Pr(s_t = 0) \approx e^{-2} = .13$, which means most windows should have another spike.



Recap

One failure mode of basic spike sorting

- Our mixture model assumed each spike waveform could be **attributed to exactly one neuron**.
- However, with high probability, the waveforms are the **superposition of many spikes**.
- The superposition will bias the waveform estimates since it introduces non-Gaussian “noise” to the windows.

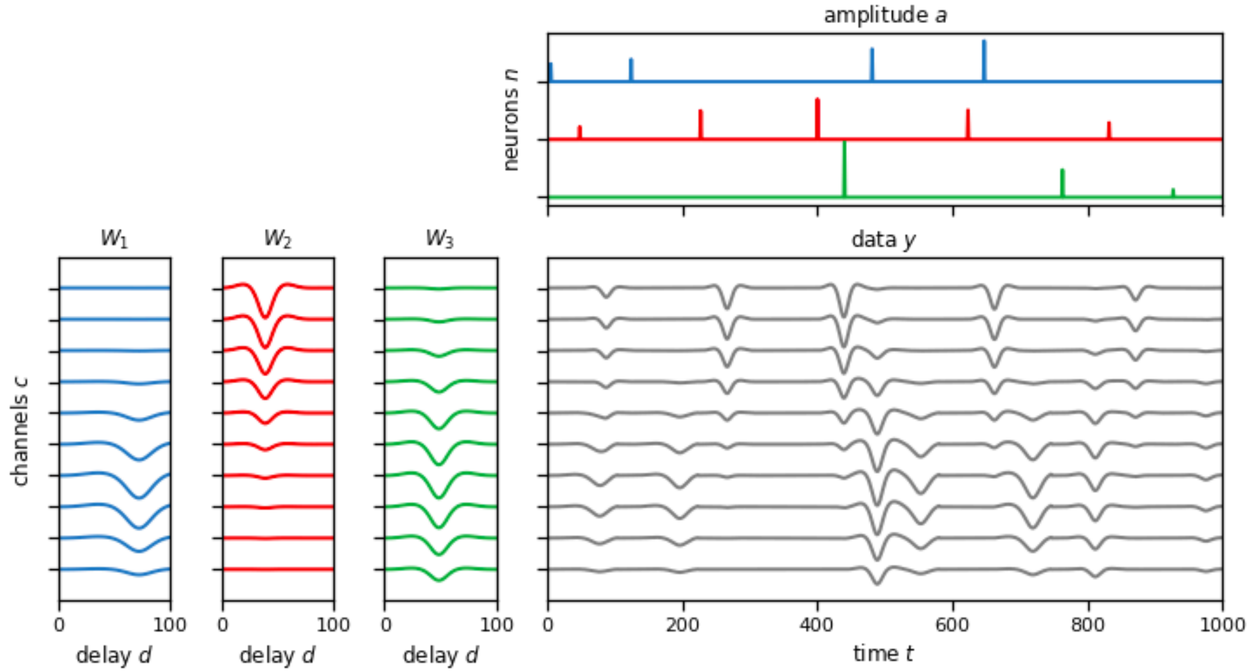


Convolution and cross-correlation

Where are we going with this?

10,000ft view

- **Idea:** each time a neuron spikes, it adds a scaled copy of its template to the measured voltage.
- Rather than extracting windows around spikes, let's infer the **spike amplitude** of each neuron in each time bin.
- **Most amplitudes will be zero** because neurons fire only a few times a second and we're sampling at 30kHz.
- Formally, we model the data as a **sum of convolutions** of templates and amplitudes for each neuron, plus noise.



Convolution

In one dimension

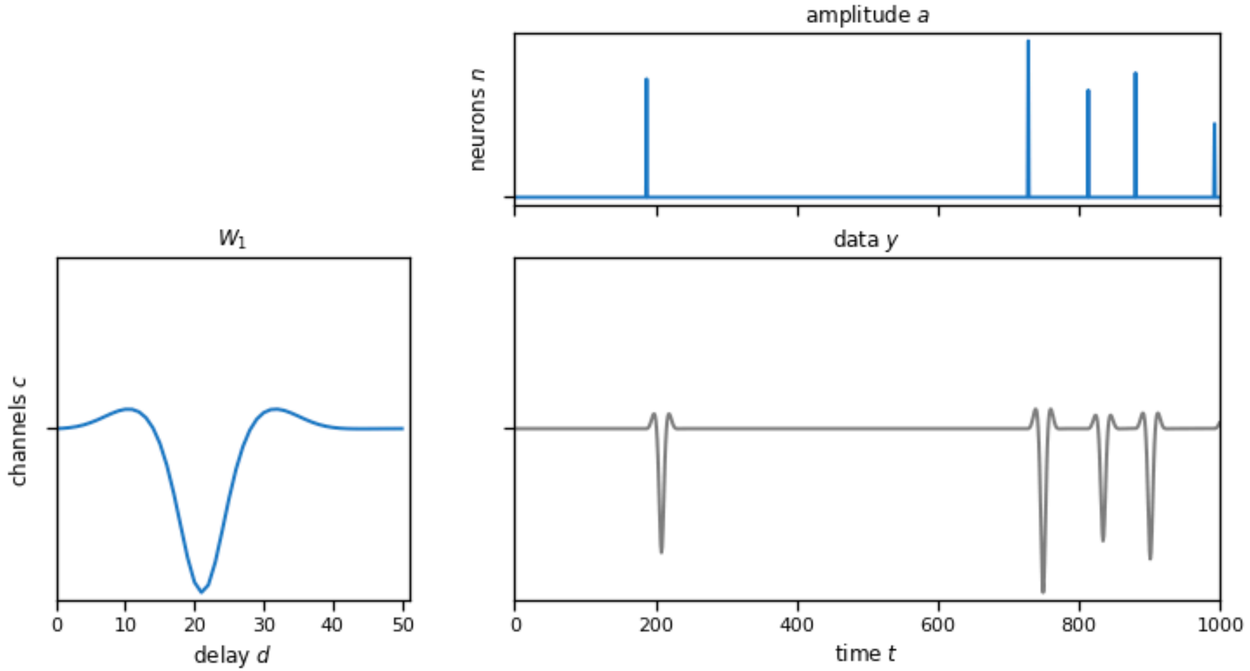
- **Convolution** is an operation that takes in a signal $a(t)$ and a filter $w(t)$ and outputs

$$y(t) = [a \circledast w](t) = \int a(t-\tau)w(\tau)\,\mathrm{d}\tau.$$

- In **discrete time** this becomes,

$$y_t = [a \circledast w]_t = \sum_{d=-\infty}^{\infty} a_{t-d} w_d$$

- **Causal** filters are constrained so that $w_d = 0$ for $d < 0$. Then y_t is only influenced by $a_{1:t}$.
- Our filters will also have **bounded support** so that $w_d = 0$ for $d \geq D$. Then y_t is only influenced by $a_{t-D+1:t}$.
- In our case, the signal is the time series of spike amplitudes, and the filter is the waveform template. Every time there's a spike, we plop down a scaled template.

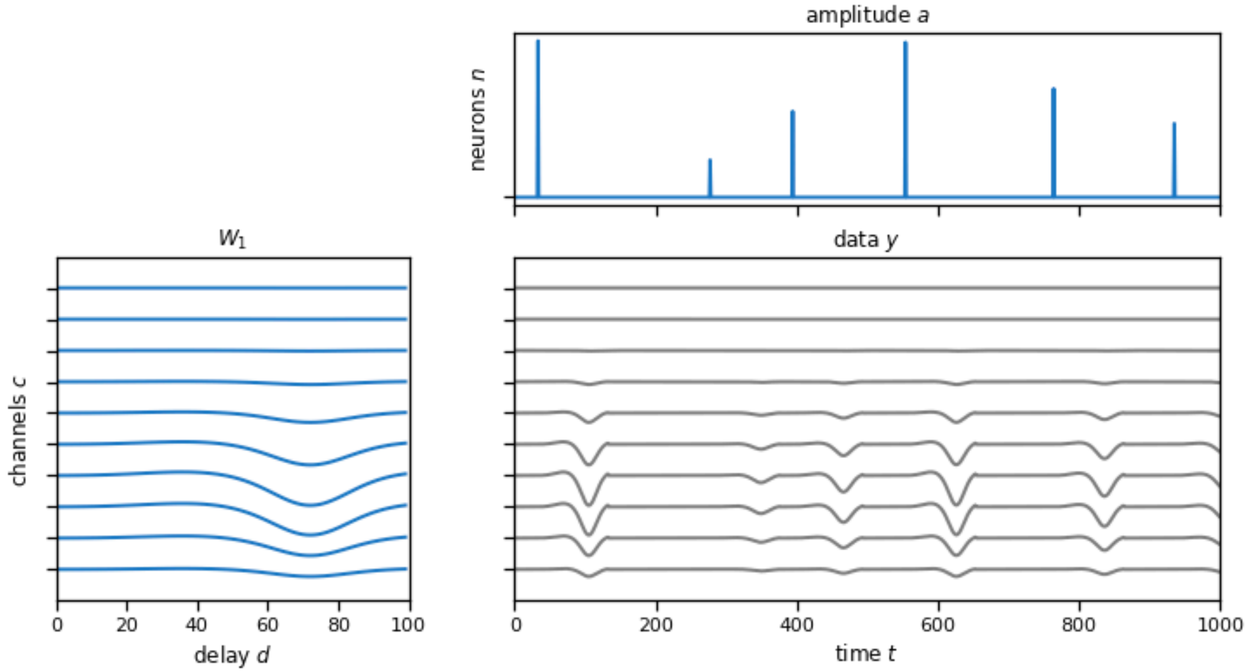


Convolution

With multiple output channels

- We need to convolve the amplitude signal with **multiple filters** in parallel, **one for each channel** of the voltage recording.

$$y_t = \begin{pmatrix} [a \circledast w_1]_t \\ \vdots \\ [a \circledast w_C]_t \end{pmatrix} = \begin{pmatrix} \sum_{d=1}^{D} a_{t-d} w_{1d} \\ \vdots \\ \sum_{d=1}^{D} a_{t-d} w_{Cd} \end{pmatrix}$$



- (I'm going to index d from $1, \ldots, D$ because the notation is a bit simpler.)

Convolution

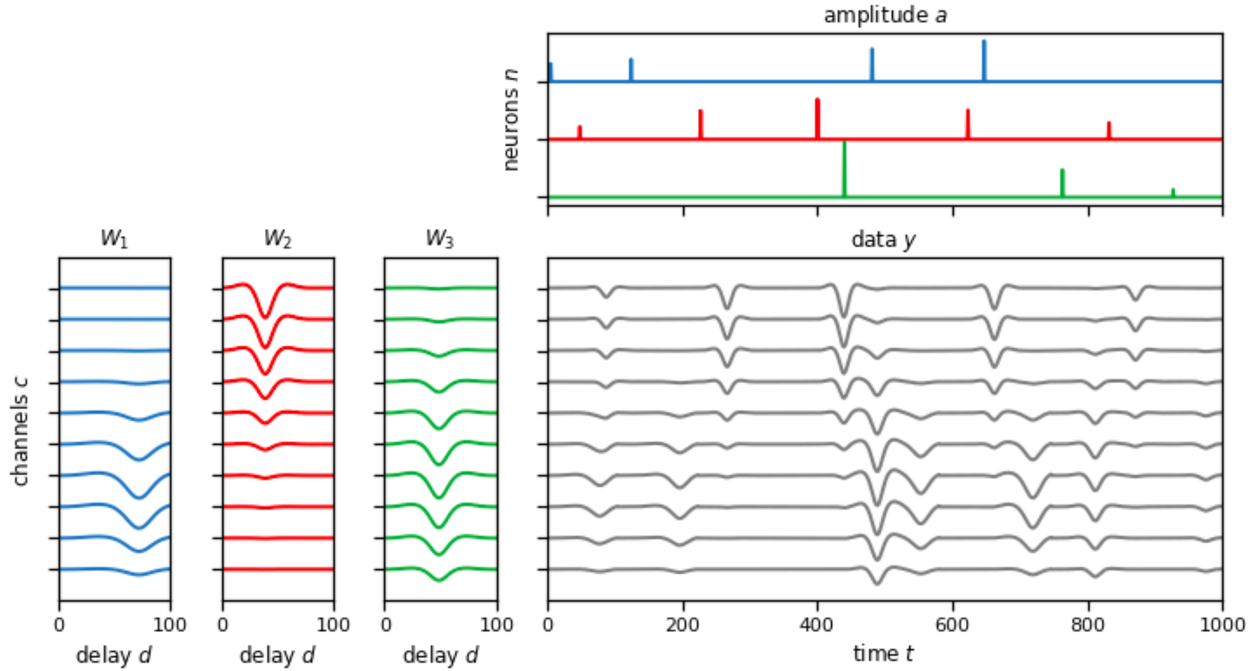
With multiple input & output channels

- Finally, we need to sum convolutions of multiple input signals, **one for each neuron** in the model.

$$Y = A \circledast W$$

which means

$$y_{ct} = \sum_{n=1}^{N} \sum_{d=1}^{D} a_{n,t-d} w_{ncd}.$$



Cross-Correlation

In one dimension

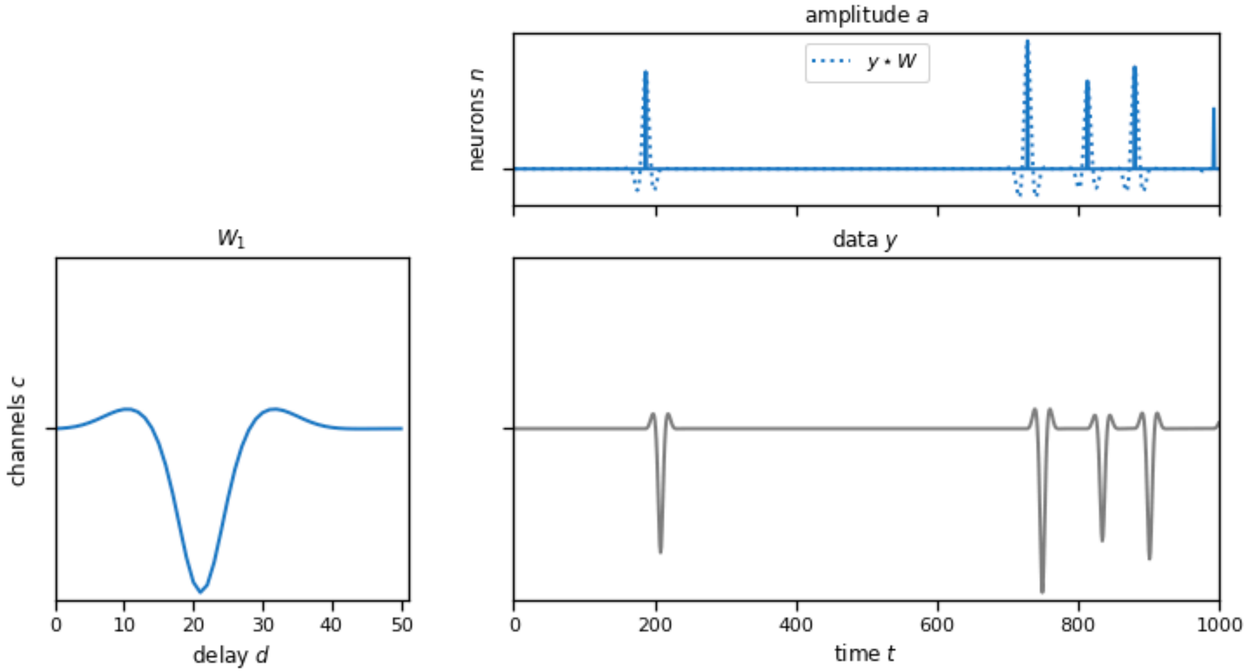
- Cross-correlation, in some sense, goes in reverse.
- In signal processing, the cross-correlation is a sliding dot product of data $y(t)$ and template $w(t)$, which produces a new function $[y \star w](t)$.
- For discrete time, real-valued inputs,

$$[y \star w]_t = \sum_{d=-\infty}^{\infty} y_{t+d} w_d.$$

- With a change of variables, we see that cross-correlation is equivalent to convolution with a time-reversed filter $\overleftarrow{w}$:

$$[y \star w]_t = \sum_{d=\infty}^{-\infty} y_{t-d} w_{-d} = [y \circledast \overleftarrow{w}]_t.$$

- (Note: the definition of cross-correlation is not unique. This definition is consistent with `np.correlate` but opposite of Wikipedia.)



Cross-Correlation

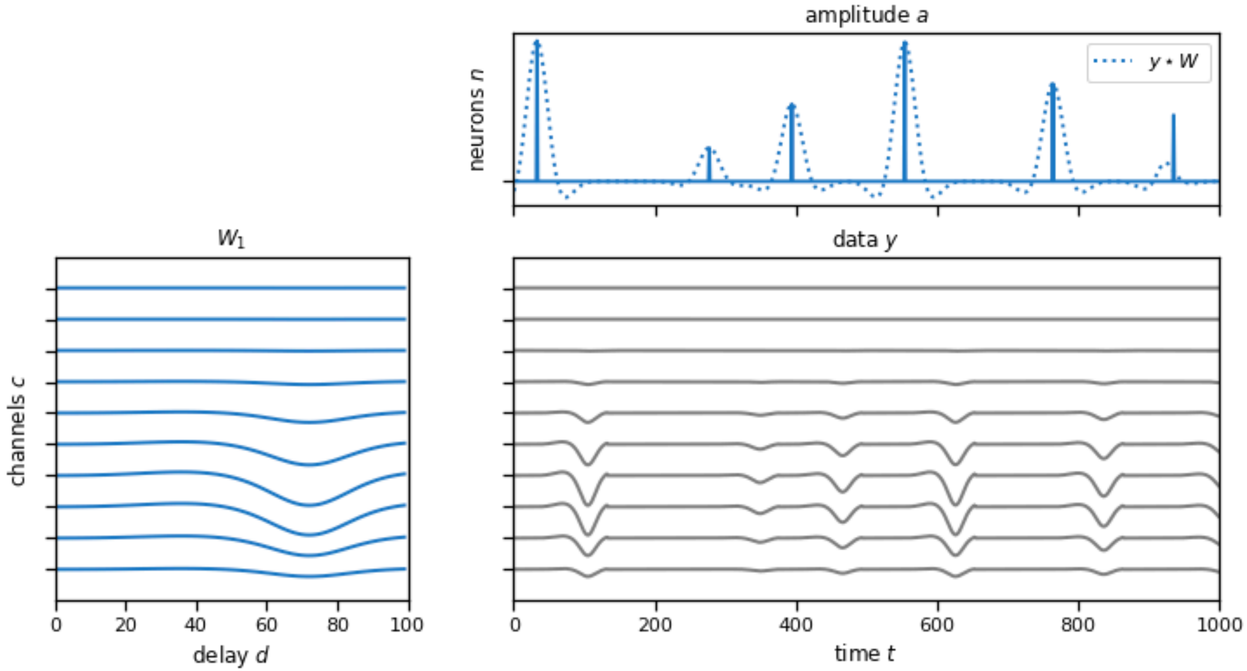
With multiple channels

- As before, we can extend this definition to handle multiple channels

$$[Y \star W]_t = \sum_{c=1}^{C} \sum_{d=-\infty}^{\infty} y_{c,t+d} w_{cd}.$$

- The cross-correlation measures the similarity of the data and the template at each point in time.

- The **auto-correlation** is the cross-correlation of a signal with itself.



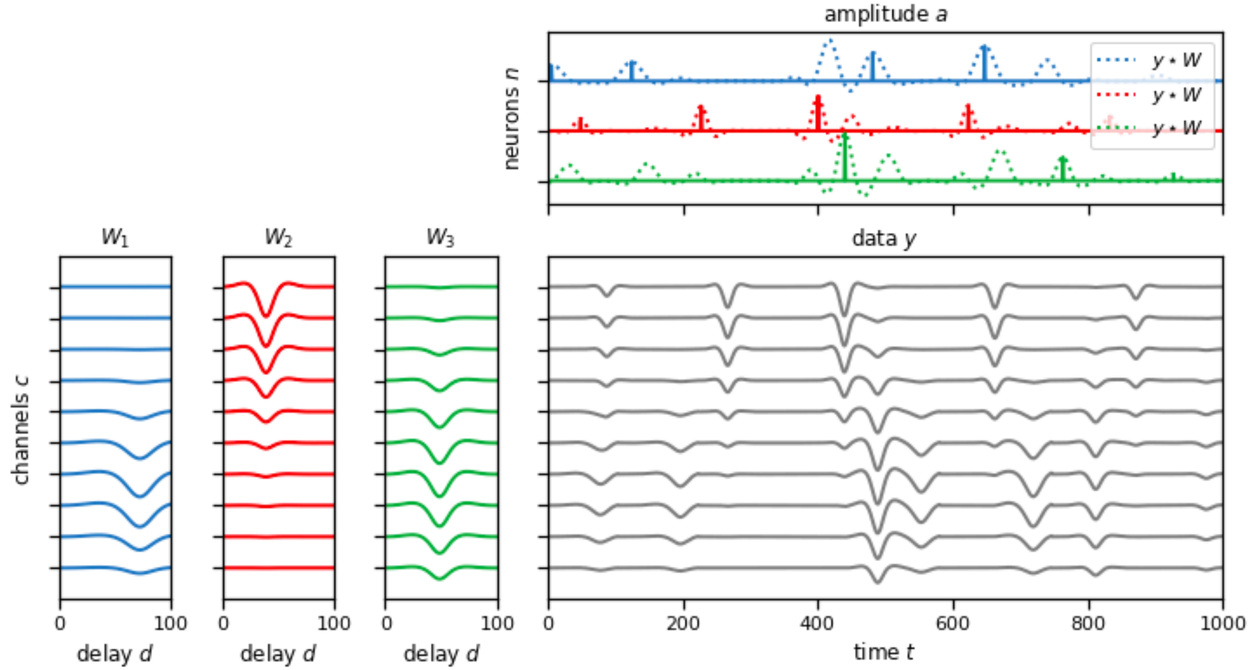
Cross-Correlation

With multiple input & output channels

- As before, we can extend this definition to handle multiple channels

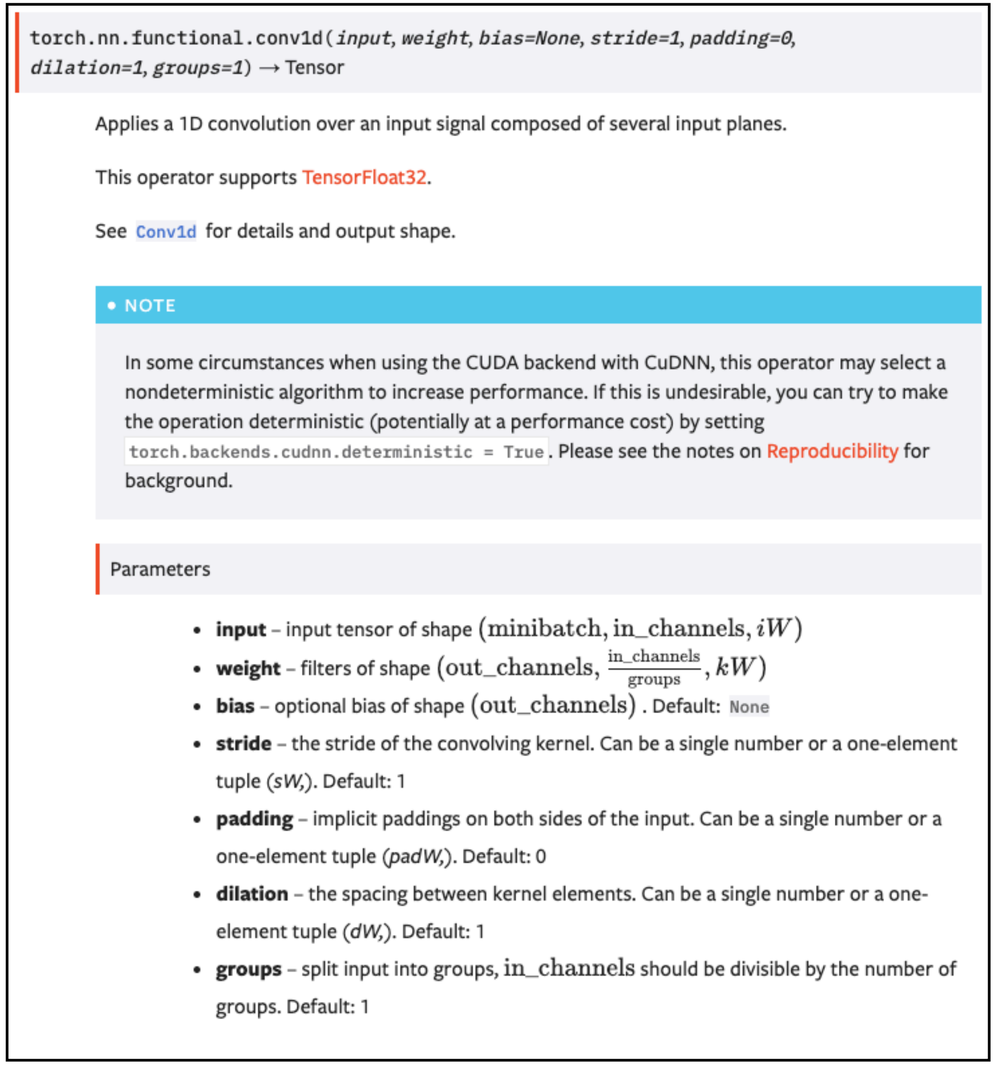
$$[Y \star W]_t = \begin{pmatrix} [Y \star W_1]_t \\ \vdots \\ [Y \star W_N]_t \end{pmatrix}$$

$$= \begin{pmatrix} \sum_{c=1}^{C} \sum_{d=-\infty}^{\infty} y_{c,t+d} w_{1cd} \\ \vdots \\ \sum_{c=1}^{C} \sum_{d=-\infty}^{\infty} y_{c,t+d} w_{Ncd} \end{pmatrix}$$



Convolution and Cross-Correlation in Pytorch

- Pytorch (and other deep learning libraries) have fast, **GPU-backed implementations** of convolutions.
- **What they call convolution is actually cross-correlation!**
- But remember, we can always get convolution by cross-correlating with the flipped filter.
- For discrete time signals, you have to play with **padding** to handle **edge effects**.
- By default, these functions operate on **mini-batches** of inputs, so you need to add an extra leading dimension to your signal.
- There are **lots of other options** to read about (strides, dilations, groups), but we won't use them this week.



`torch.nn.functional.conv1d(`*`input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1`*`)` → Tensor

Applies a 1D convolution over an input signal composed of several input planes.

This operator supports TensorFloat32.

See `Conv1d` for details and output shape.

• NOTE

In some circumstances when using the CUDA backend with CuDNN, this operator may select a nondeterministic algorithm to increase performance. If this is undesirable, you can try to make the operation deterministic (potentially at a performance cost) by setting `torch.backends.cudnn.deterministic = True`. Please see the notes on Reproducibility for background.

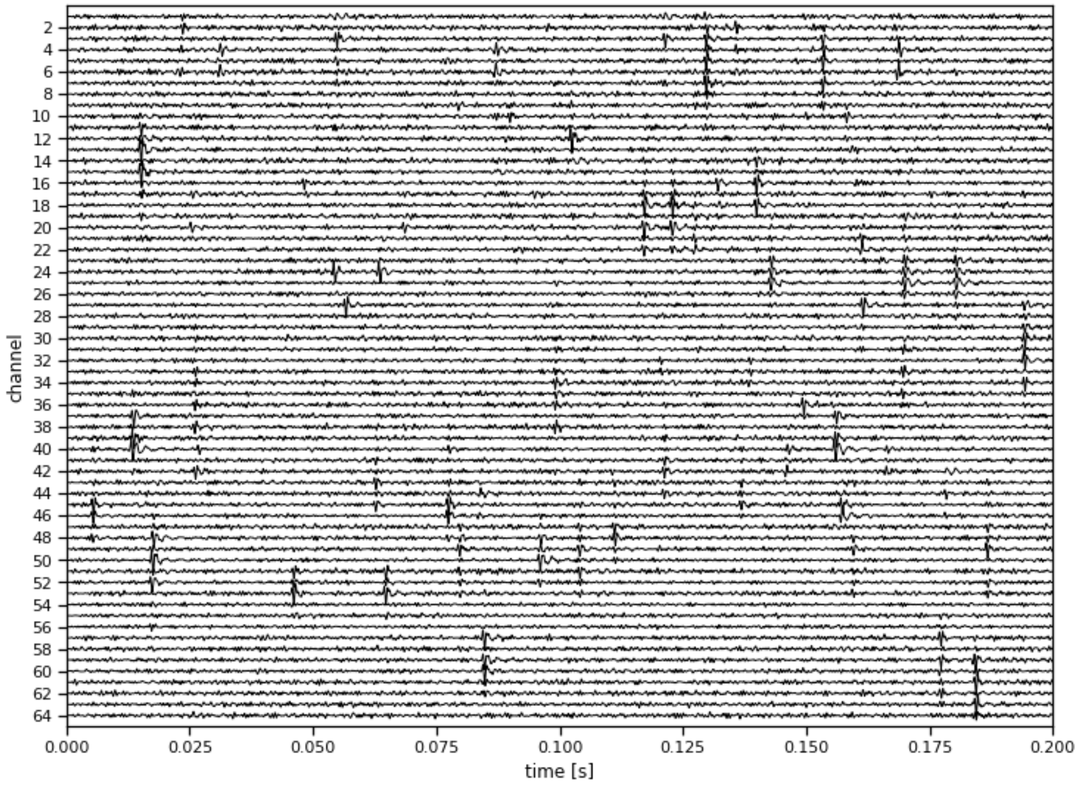
Parameters

- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iW)$
- **weight** – filters of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, kW)$
- **bias** – optional bias of shape (out_channels). Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a one-element tuple *(sW,)*. Default: 1
- **padding** – implicit paddings on both sides of the input. Can be a single number or a one-element tuple *(padW,)*. Default: 0
- **dilation** – the spacing between kernel elements. Can be a single number or a one-element tuple *(dW,)*. Default: 1
- **groups** – split input into groups, in_channels should be divisible by the number of groups. Default: 1

Spike sorting by deconvolution

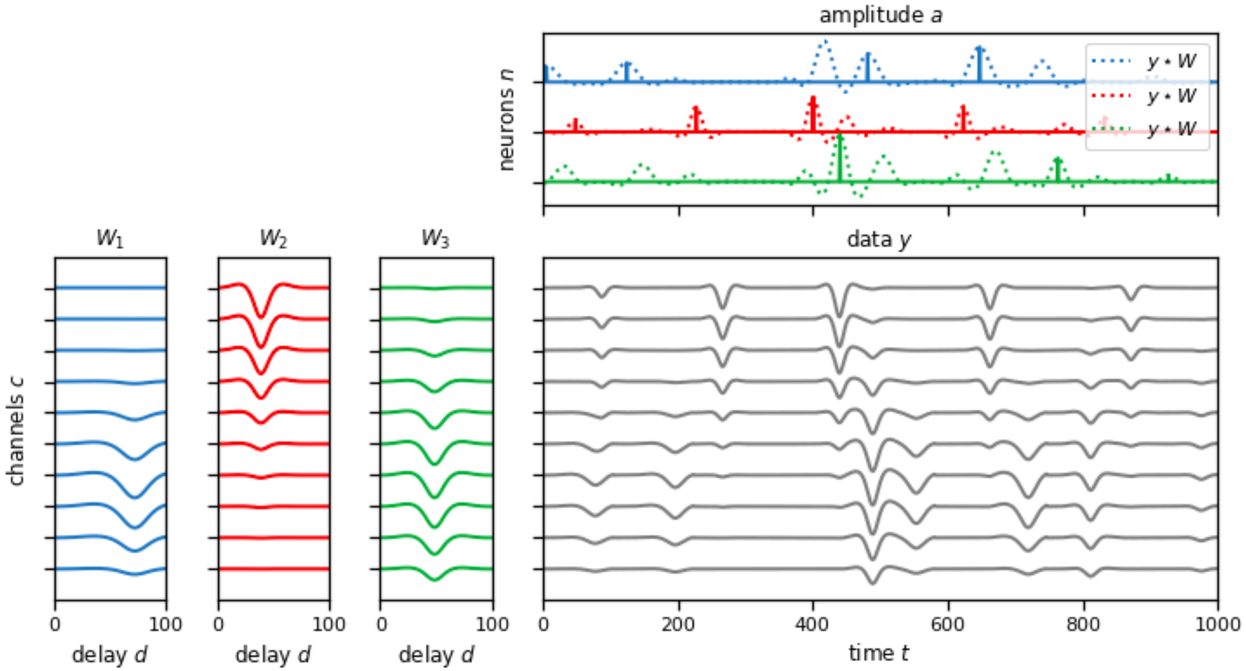
Constants

- Let T denote the number of **samples**.
- C denote the number of **channels**.
- D denote the **duration** (in samples) of a spike waveform. E.g. 81 samples $\approx$ 2.7ms at 30kHz.
- N denote the (unknown) number of **neurons** that generated the spikes.
- K denote the **rank** of the templates.



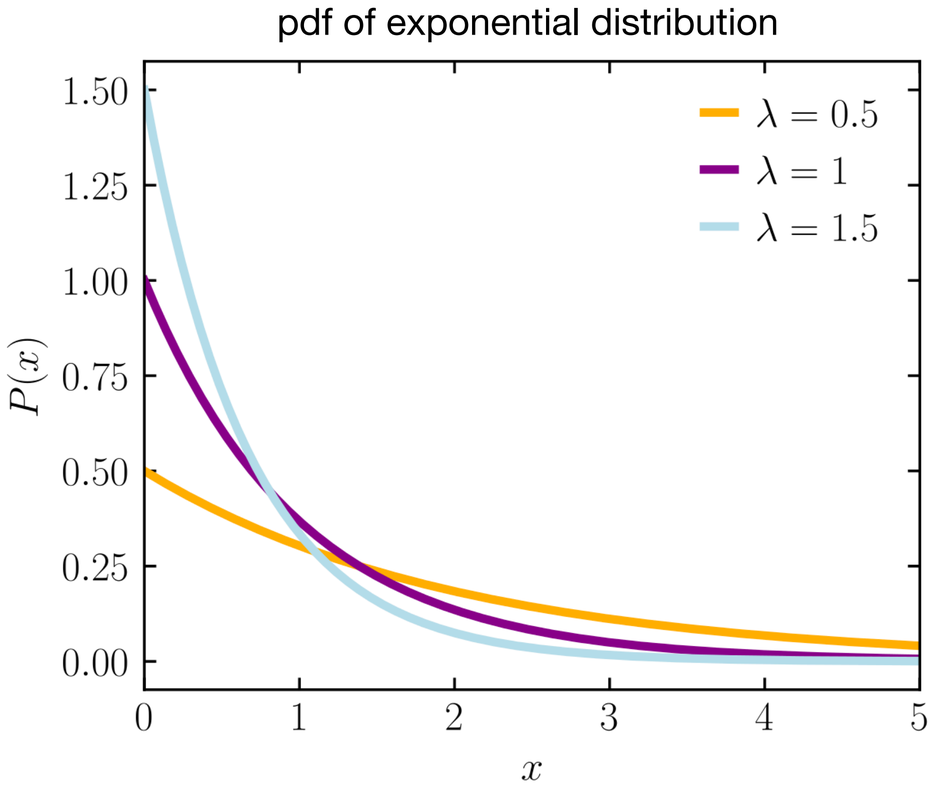
Data and Latent Variables

- **Data:**
  - Let $Y \in \mathbb{R}^{C \times T}$ denote the (filtered and standardized) voltage.
- **Latent Variables:**
  - Let $A \in \mathbb{R}_{+}^{N \times T}$ denote the time series of spike amplitudes for each neuron.
- **Parameters:**
  - Let $W \in \mathbb{R}^{N \times C \times D}$ denote the array of templates for each neuron.



Hyperparameters

- **Hyperparameters:**
  - Let $\lambda_n \in \mathbb{R}_+$ denote the inverse scale (i.e. rate) of the prior distribution on amplitudes for neuron n.
  - Let σ^2 denote the noise variance (shared by all channels).

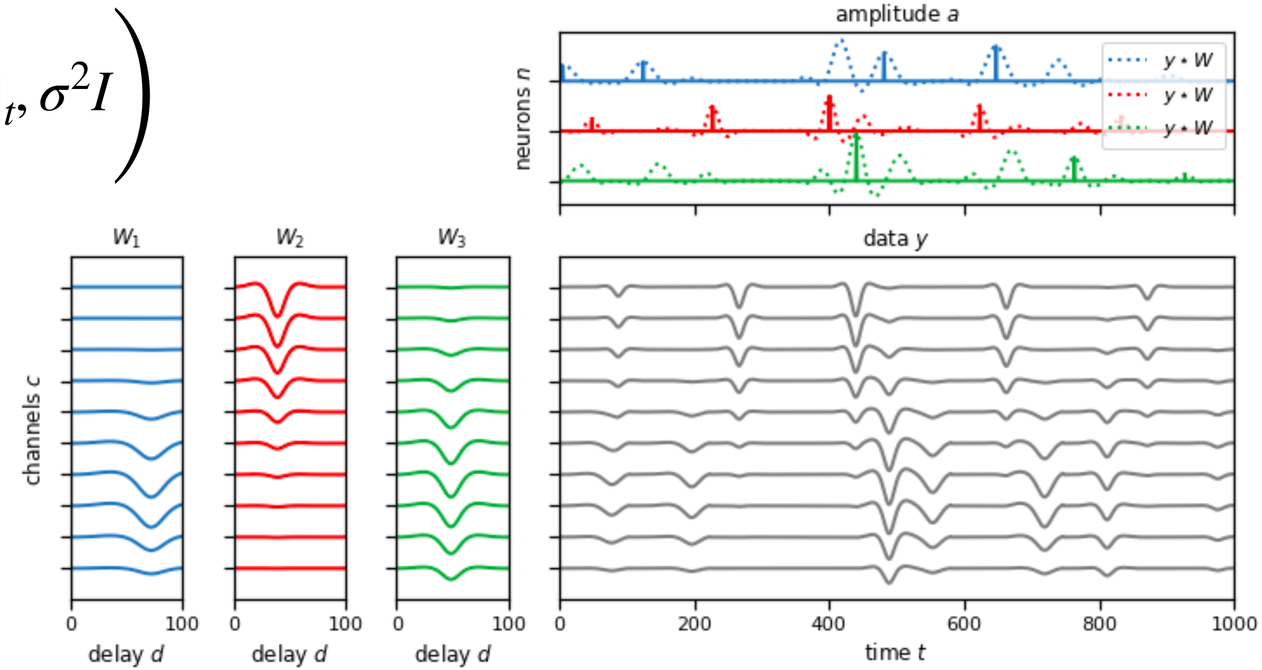


Probabilistic Model

Likelihood

- Assume each spike is a noisy, scaled version of the template of the neuron that generated it.

$$p(Y \mid A, W, \sigma^2) = \prod_{t=1}^{T} \mathcal{N}\left(y_t \mid \sum_{n=1}^{N} [a_n \circledast W_n]_t, \sigma^2 I\right)$$



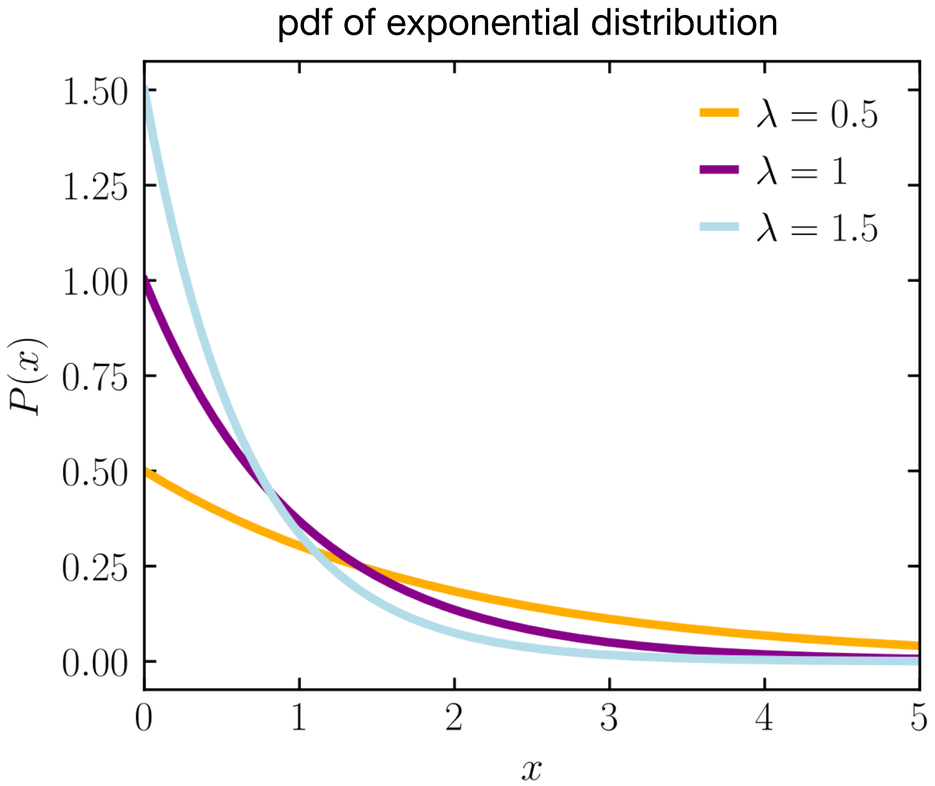
Probabilistic Model

Prior on spike amplitudes

- Assume the spike amplitudes are drawn from an exponential distribution.

$$a_{nt} \sim \mathrm{Exp}(\lambda_n)$$

- This simple prior will lead to sparse amplitudes, but it does not encode any dependencies between time steps.
- Ideally, we would also like to prohibit two spikes within D samples of each other.
- We'll discuss how to do this next week (hopefully).

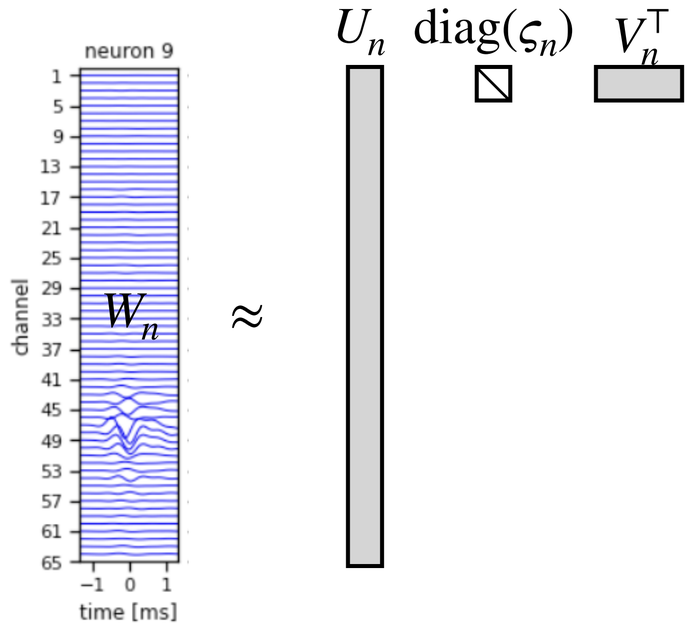


https://en.wikipedia.org/wiki/Exponential_distribution

Probabilistic Model

Prior on templates

- Like last week, we assume W_n is uniformly distributed on the set of rank-K, unit-norm matrices, which we call $\mathcal{S}_K$.
- Recall, this is equivalent to constraining the L_2-norm of the singular values to be one.



https://en.wikipedia.org/wiki/Exponential_distribution

Inference

Maximum a posteriori estimation

Coordinate ascent

- Initialize templates W and set $A = 0$.
- Iterate until convergence:
  - For neuron $n = 1, \ldots, N$:

    a. Optimize **amplitudes** a_n for neuron n.

    b. Optimize **templates** W_n for neuron n.

[In each case, maximize log joint probability wrt one variable, holding others fixed.]

Maximum a posteriori estimation

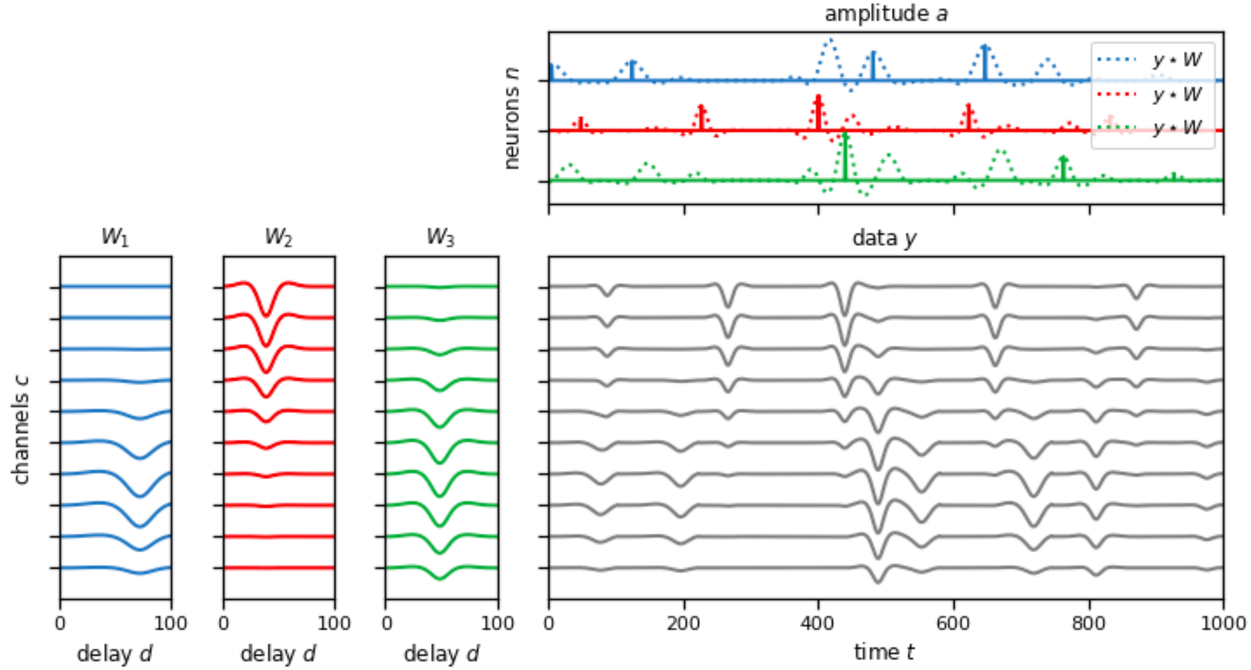
Optimizing the amplitudes

Let

$$\mathscr{L}(a_n) \triangleq \log p(Y \mid A, W, \sigma^2) + \log p(a_n \mid \lambda_n)$$
$$= -\frac{1}{2\sigma^2}\|R_n - a_n \circledast W_n\|_F^2 + \log p(a_n) + \mathrm{c}$$

where $R_n \in \mathbb{R}^{C \times T}$ is the residual for neuron n, defined as

$$R_n \triangleq Y - \sum_{m \neq n} [a_m \circledast W_m]$$



Maximum a posteriori estimation

Optimizing the amplitudes

Expanding the square

$$\mathscr{L}(a_n) = -\frac{1}{2\sigma^2}\|R_n - a_n \circledast W_n\|_F^2 + \log p(a_n) + \mathrm{c}$$

$$= \underbrace{-\frac{1}{2\sigma^2}\|a_n \circledast W_n\|_F^2}_{\mathscr{L}_1(a_n)} + \underbrace{\frac{1}{\sigma^2}\sum_{t=1}^{T} r_{n,:,t}^{\top}[a_n \circledast W_n]_t}_{\mathscr{L}_2(a_n)} + \log p(a_n) + \mathrm{c}\,.$$

where $r_{n,:,t} \in \mathbb{R}^C$ is the t-th column of the residual R_n.

Maximum a posteriori estimation

Optimizing the amplitudes

Further expanding the first term

$$
\begin{aligned}
\mathscr{L}_1(a_n) &= -\frac{1}{2\sigma^2}\|a_n \circledast W_n\|_F^2 \\
&= -\frac{1}{2\sigma^2}\sum_{t=1}^{T}\sum_{c=1}^{C}\left(\sum_{d=1}^{D} a_{n,t-d}^2 w_{ncd}^2 + 2\sum_{d=1}^{D}\sum_{d'=1}^{d-1} a_{n,t-d}a_{n,t-d'}w_{ncd}w_{ncd'}\right) \\
&\approx -\frac{1}{2\sigma^2}\sum_{t=1}^{T} a_{nt}^2\|W_n\|_F^2 \\
&= -\frac{1}{2\sigma^2}\sum_{t=1}^{T} a_{nt}^2
\end{aligned}
$$

with equality when nonzero entries (i.e. “spikes”) of a_n are separated by at least D samples.

Maximum a posteriori estimation

Optimizing the amplitudes

Now take the second term…

$$\mathscr{L}_2(a_n) = \frac{1}{\sigma^2} \sum_{t=1}^{T} r_{n,:,t}^{\top} \left(\sum_{d=1}^{D} a_{n,t-d} w_{nd} \right)$$

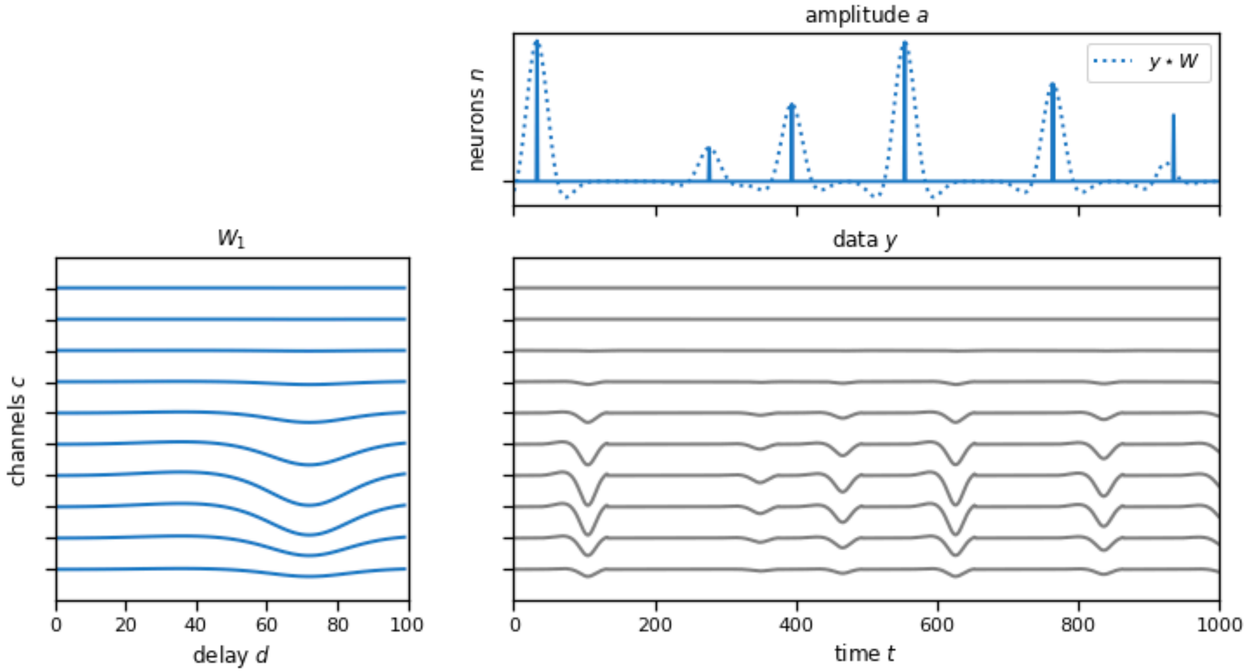
$$= \frac{1}{\sigma^2} \sum_{t=1}^{T} a_{nt} \sum_{d=1}^{D} r_{n,:,t+d}^{\top} w_{nd}$$

$$= \frac{1}{\sigma^2} \sum_{t=1}^{T} \mu_{nt} a_{nt}$$

where

$$\mu_{nt} \triangleq \sum_{d=1}^{D} r_{n,:,t+d}^{\top} w_{nd} = [R_n \star W_n]_t \,.$$

is the cross-correlation of the residual and the template for neuron n.



Maximum a posteriori estimation

Optimizing the amplitudes

Putting it all together

$$\mathscr{L}(a_n) = \sum_{t=1}^{T} \left[-\frac{1}{2\sigma^2} a_{nt}^2 + \frac{1}{\sigma^2} \mu_{nt} a_{nt} - \lambda_n a_{nt} \right] + \text{c},$$

which separates into a sum of quadratic objective functions for each time t.

Maximum a posteriori estimation

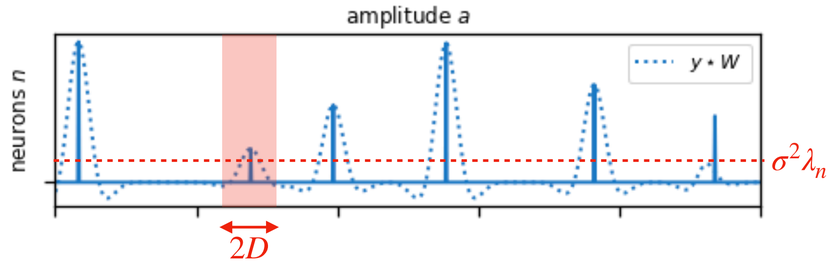
Completing the square and solving for the optimal amplitudes

- Like last week, the maximum, subject to non-negativity constraints, is obtained at

$$a_{nt} = \max\left\{0, \mu_{nt} - \sigma^2\lambda_n\right\}$$

- However, we also want spikes to be well-separated; i.e. $a_{nt} > 0 \implies a_{n,t+d} = 0$ for $d = 1,\ldots,D$.

- We'll enforce this with a **simple heuristic**: use `find_peaks` to select local maxima of this "score" signal.



Maximum a posteriori estimation

Optimizing the templates

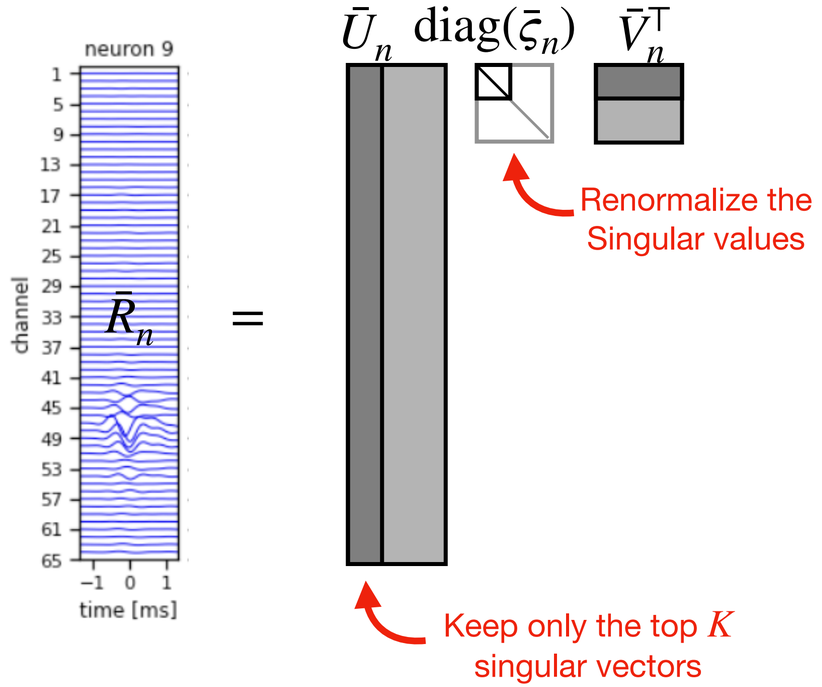
- Just like last week, best unit-norm low rank template is found by the truncated and normalized SVD of $\bar{R}_n \in \mathbb{R}^{C \times D}$, where

$$\bar{R}_n = \sum_{t:a_{nt}>0} a_{nt} R_{n,:,t:t+D}$$

- **Implementation trick:** we can write this as

$$\bar{R}_n = R_n \left[a_n \circledast I_D \right]^\top$$

- The convolution produces a $D \times T$ matrix of lagged spike amplitudes.



More efficient computation

Leveraging the low-rank templates

Recall that our "scores" μ_{nt} were defined as

$$\mu_{nt} = [R_n \star W_n]_t \triangleq \sum_{d=1}^{D} r_{n,:,t+d}^\top w_{n,:,d}$$

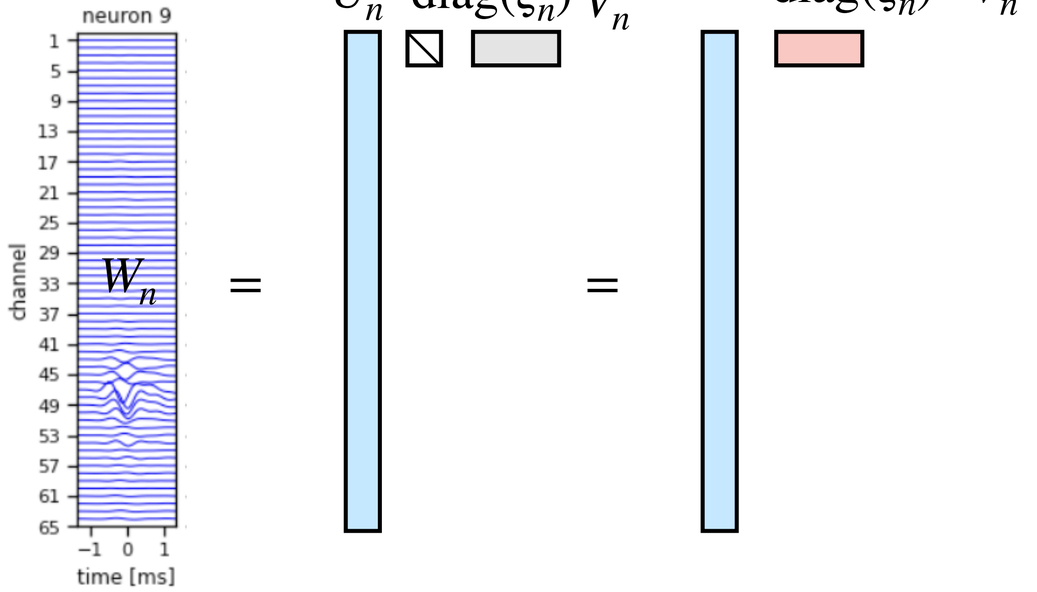
$$= \sum_{d=1}^{D} r_{n,:,t+d}^\top \left(\sum_{k=1}^{K} u_{n,:,k} \varsigma_{nk} v_{nkd} \right)$$

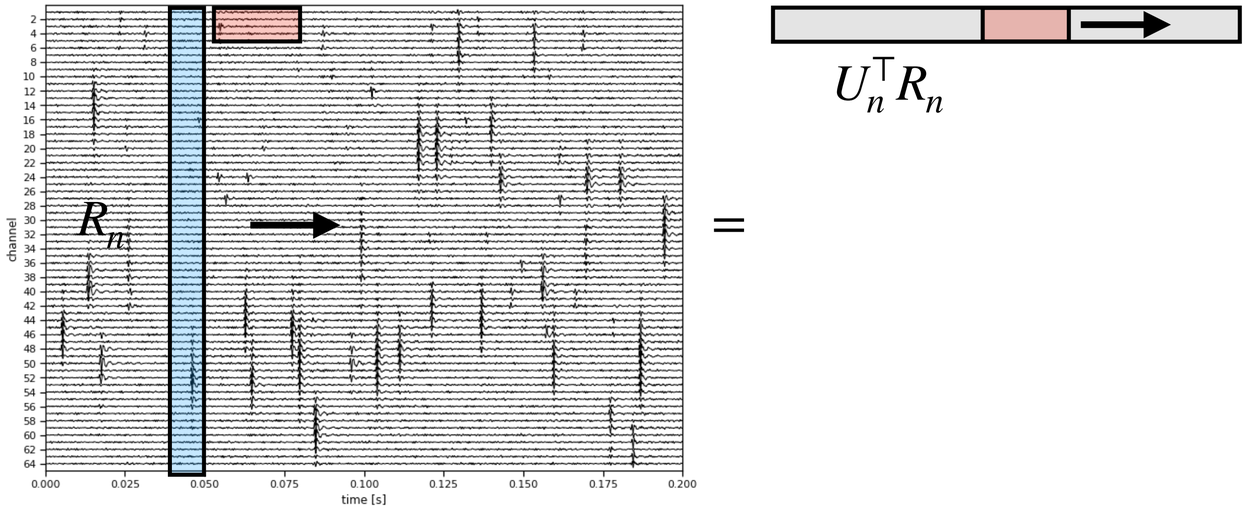
$$= \sum_{k=1}^{K} \sum_{d=1}^{D} (r_{n,:,t+d}^\top u_{n,:,k}) (\varsigma_{nk} v_{nkd})$$

$$= \sum_{k=1}^{K} [(u_{n,:,k}^\top R_n) \star (\varsigma_{nk} v_{n,k,:})]_t$$

$$= [(U_n^\top R_n) \star (\mathrm{diag}(\varsigma_n) V_n^\top)]_t$$

In other words, we cross-correlate the projected residual.





Conclusion

- Our basic model ignored **overlapping spikes**, which occur with high probability in modern recordings.
- We developed a new model for the voltage in terms of a superposition of templates convolved with spike amplitudes for each neuron.
- We derived a **coordinate ascent algorithm** for *maximum a posteriori* (MAP) inference.
- **Next time**: you'll implement the algorithm in lab! You'll learn a bit of PyTorch for implementing the convolutions and cross-correlations, then test it out on the GPU.

Further reading

- Ch 2.2 of the course notes.
- Convolution and cross-correlation:
    - Chapter 9 of *The Deep Learning Book* (deeplearningbook.org/contents/convnets.html)
- PyTorch:
    - We'll use it in lab, so take a look at some tutorials: https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- Spike sorting:
    - Pachitariu, Marius, Nicholas Steinmetz, Shabnam Kadir, Matteo Carandini, and Harris Kenneth D. 2016. "Kilosort: Realtime Spike-Sorting for Extracellular Electrophysiology with Hundreds of Channels." Cold Spring Harbor Laboratory. https://doi.org/10.1101/061481.
        - The model we presented is a slightly modified version of *Kilosort*
    - Lee, Jinhyung, Catalin Mitelut, Hooshmand Shokri, Ian Kinsella, Nishchal Dethe, Shenghao Wu, Kevin Li, et al. 2020. "YASS: Yet Another Spike Sorter Applied to Large-Scale Multi-Electrode Array Recordings in Primate Retina." Cold Spring Harbor Laboratory. https://doi.org/10.1101/2020.03.18.997924.
        - Another spike sorting algorithm similar to Kilosort but with spatial constraints on the templates.