# Machine Learning Methods for Neural Data Analysis

## Spike Sorting by Deconvolution

Scott Linderman

*STATS 220/320 (NBIO220, CS339N). Winter 2023.*

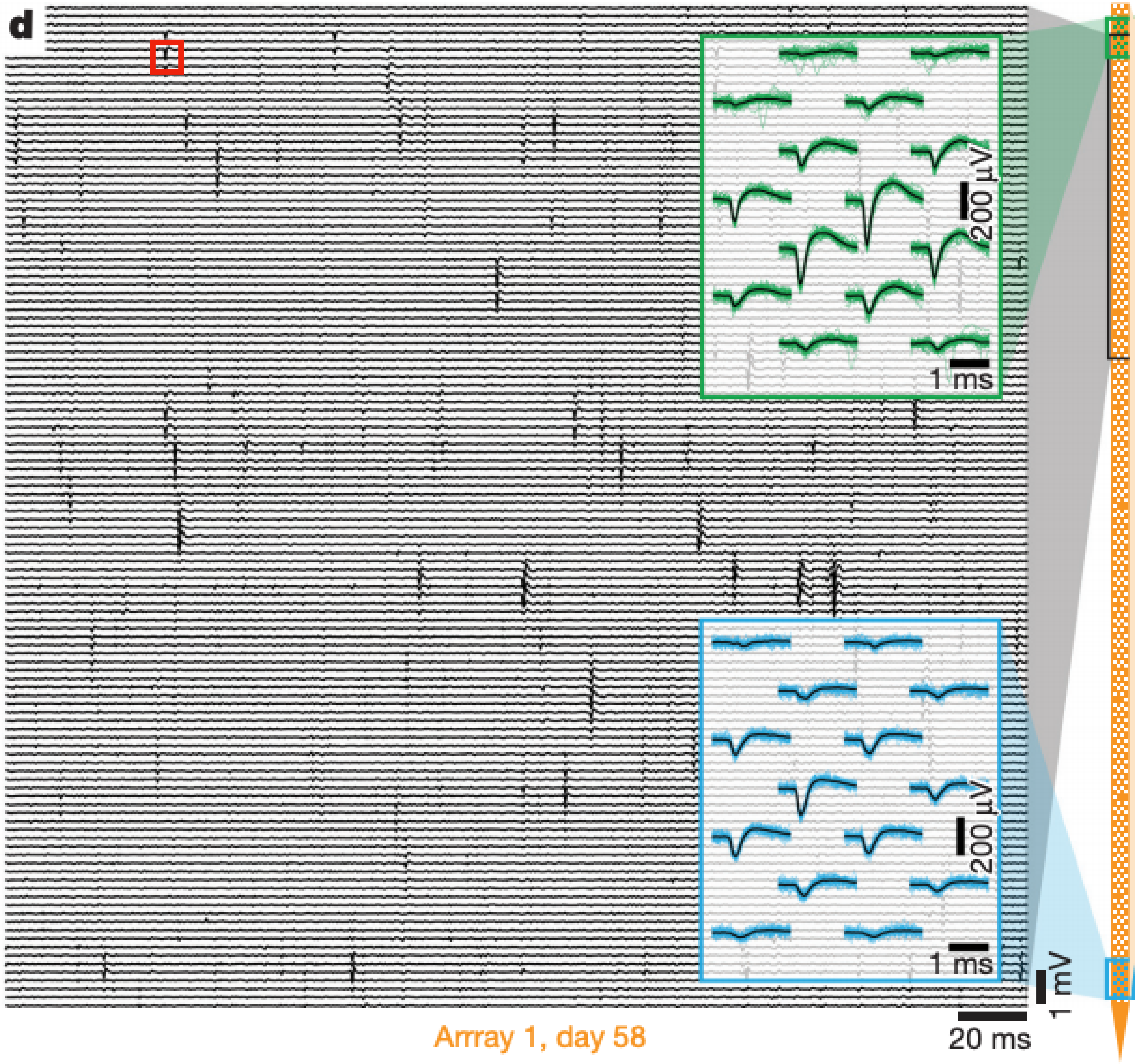
# Announcements

- Lab 1 due **Thursday, 11:59pm**.
- **Author contributions**: Please add a short paragraph at the end of your lab describing if/how you divided the work. E.g.
    - *Alice, Bob, and Chuck worked through Part 1 in class, and then met twice more to finish the remainder of the lab as a group.*
    - *Alice, Bob, and Chuck worked through Part 1 in class. Alice took the lead on Part 2, then Bob finished Parts 3 and 4. Chunk completed Part 5. All three met to discuss and check their solutions, and then combine them into a single report.*
- Please feel free to ask (and answer!) questions on **Ed**. Don't share code solutions directly, but hints and clarifications are fine.
- I will try to post future labs and team assignments further in advance.

# Spike Sorting by Deconvolution

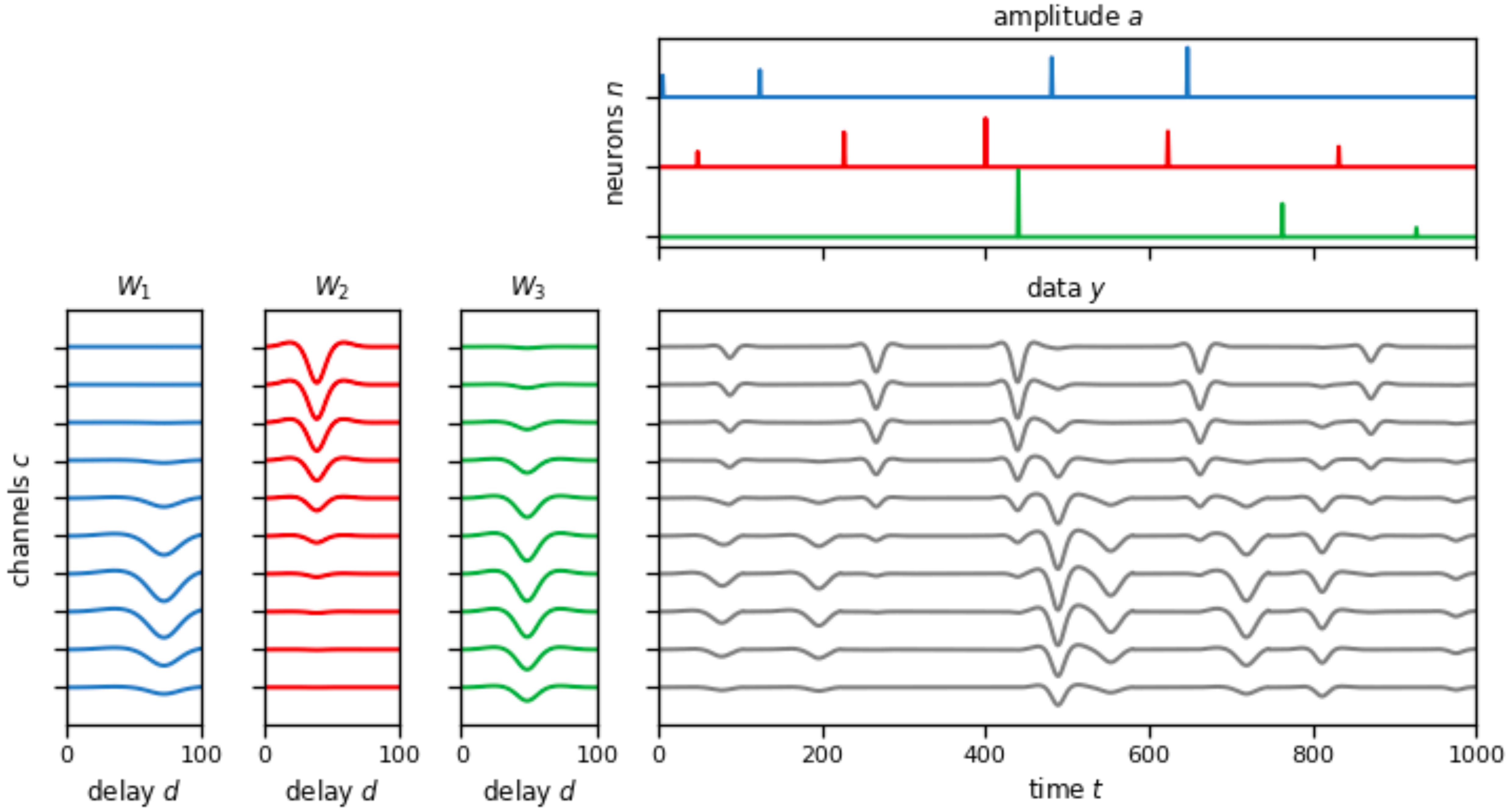
# Improving upon the simple model

- Our simple model was a good warm-up, but **downsampling** to 2ms bins **isn't very practical**.
- In reality, the average voltage over a spike can be $\approx 0$, so **you might miss spikes altogether**!
- Next, we'll extend the simple model with a more realistic one using **convolutions**.
- The resulting model will be very similar to **Kilosort** [Pachitariu et al., 2023]



# 10,000ft view

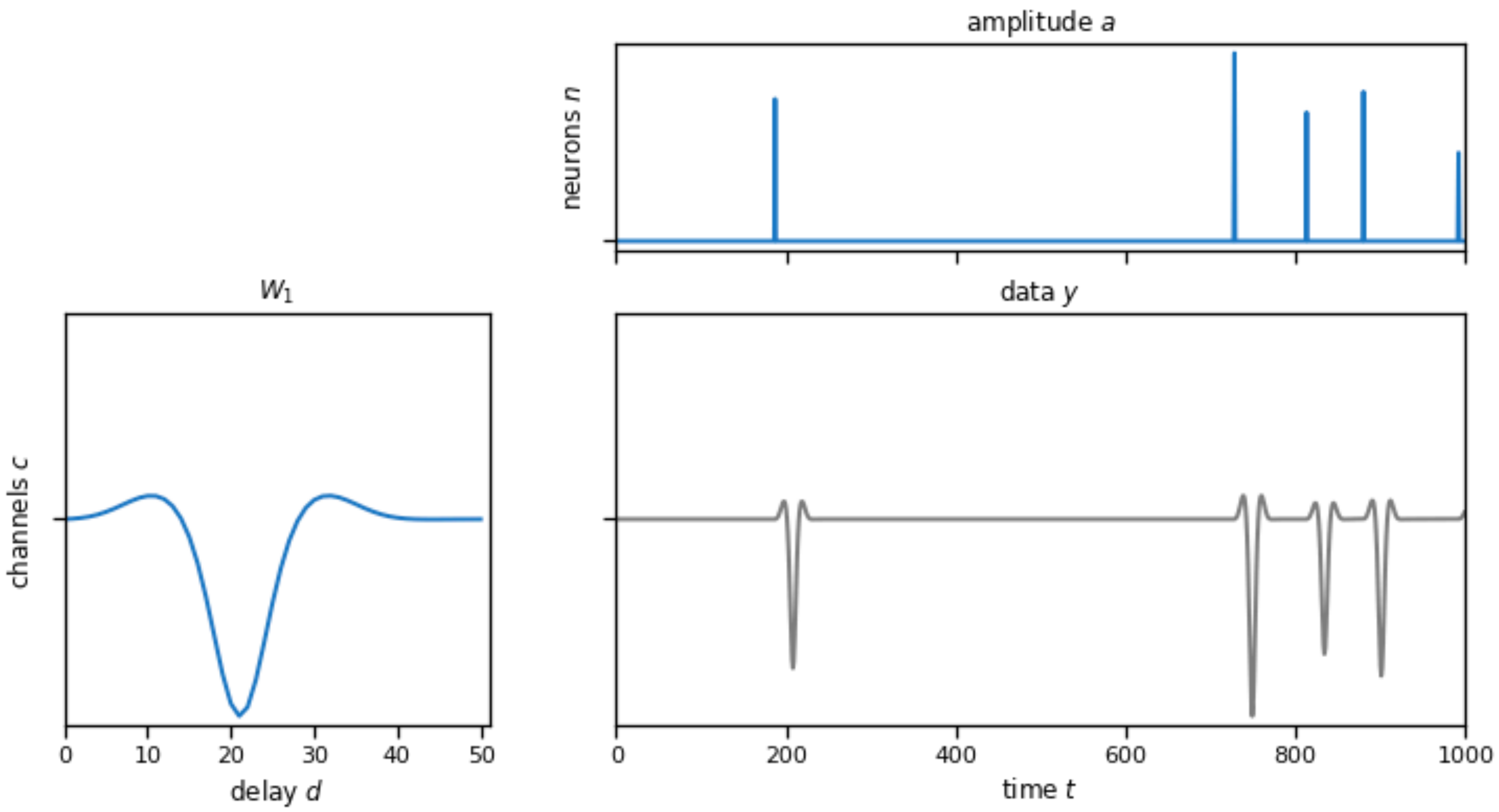
- **Idea:** each time a neuron spikes, it adds a scaled copy of its template to the measured voltage.
- Formally, we model the data as a **sum of convolutions** of templates and amplitudes for each neuron, plus noise.



# 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.



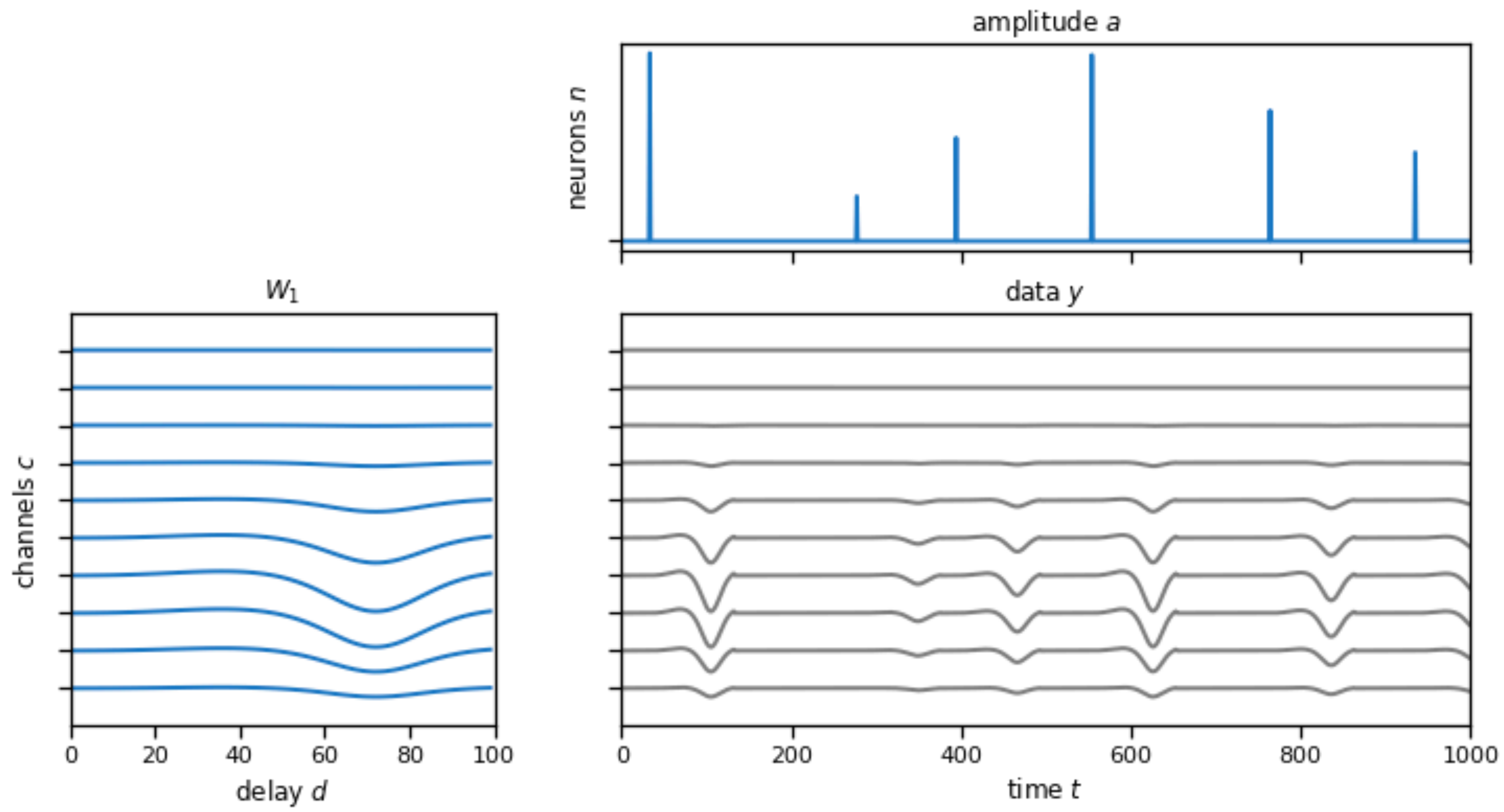
# Convolution

## With multiple output channels

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

$$\mathbf{y}_t = \begin{pmatrix} [a \circledast w_1]_t \\ \vdots \\ [a \circledast w_N]_t \end{pmatrix} = \begin{pmatrix} \sum_{d=1}^{D} a_{t-d} w_{1,d} \\ \vdots \\ \sum_{d=1}^{D} a_{t-d} w_{N,d} \end{pmatrix}$$

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



# Convolution

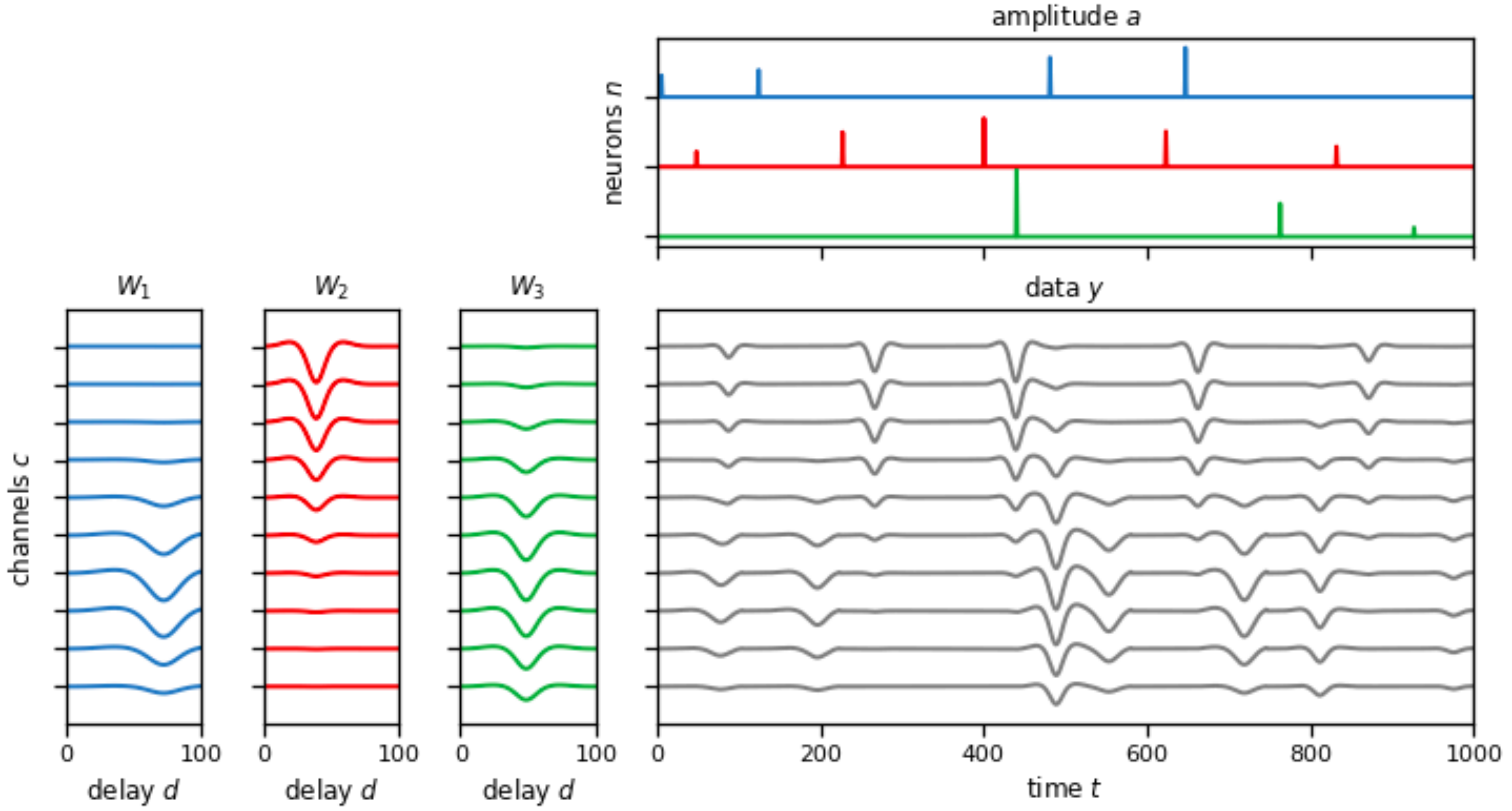
## With multiple input & output channels

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

$$\mathbf{X} = \mathbf{A} \circledast \mathbf{W}$$

by which we mean

$$x_{n,t} = \sum_{k=1}^{K} \sum_{d=1}^{D} a_{k,t-d} w_{k,n,d}.$$



# Cross-Correlation

## In one dimension

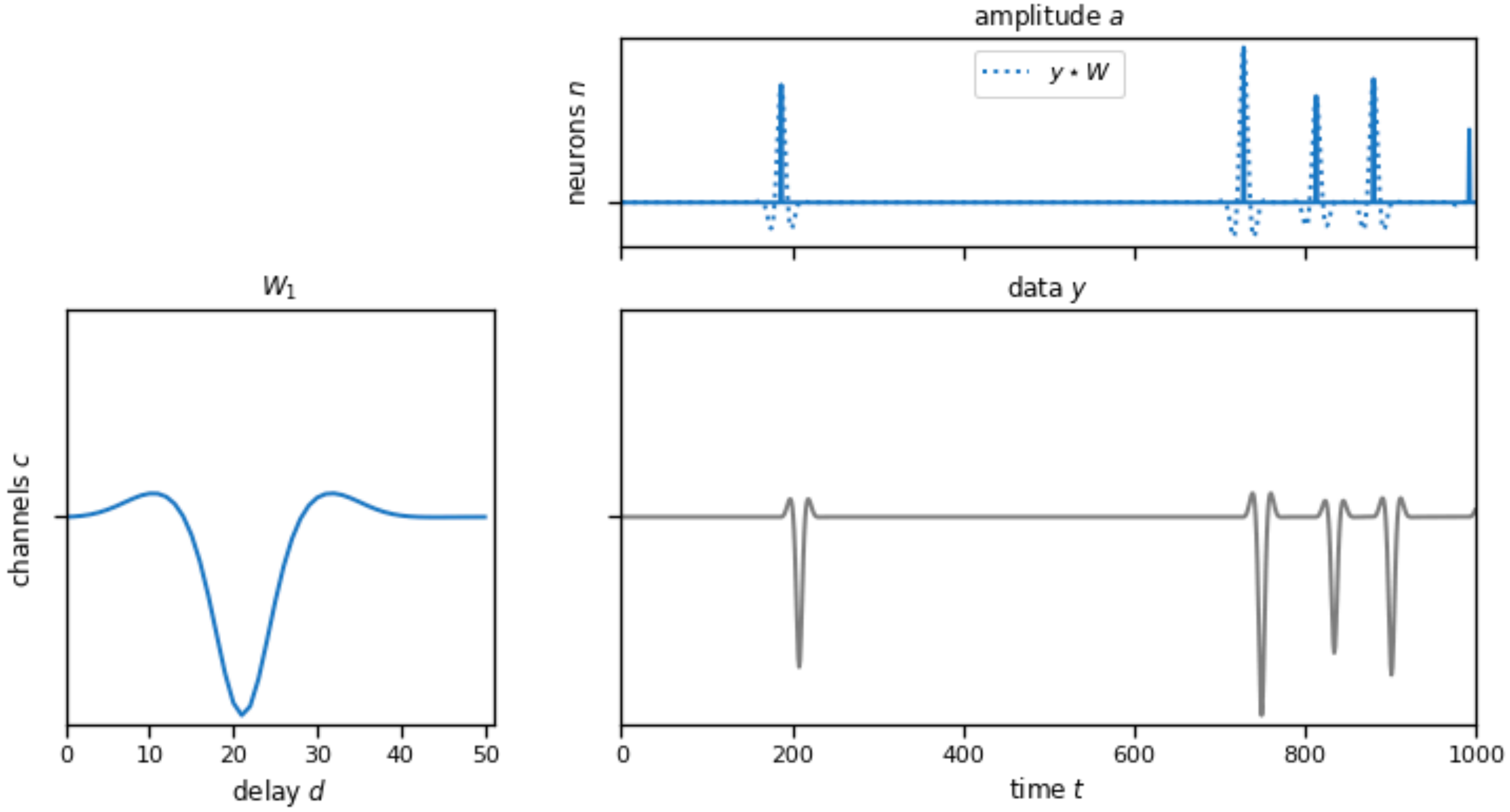
- Cross-correlation essentially goes in reverse.
- In signal processing, the cross-correlation is a sliding dot product of data $y(t)$ and template $w(t)$, which produces a new function $[y \star w](t)$.
- For discrete time, real-valued inputs,

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

- 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

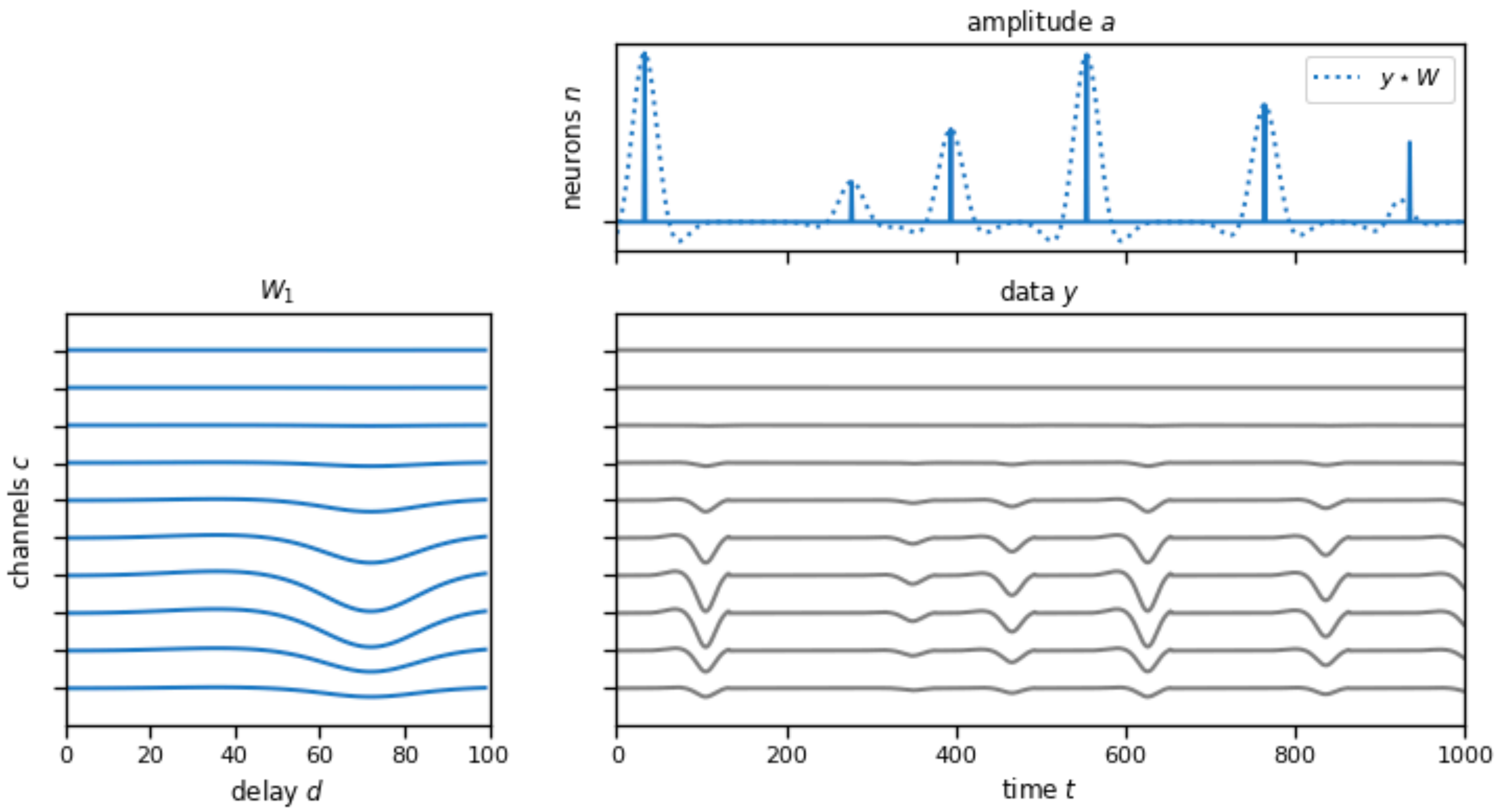
## With multiple channels

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

$$[\mathbf{Y} \star \mathbf{W}]_t = \sum_{n=1}^{N} \sum_{d=-\infty}^{\infty} y_{n,t+d} w_{n,d}.$$

- 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.



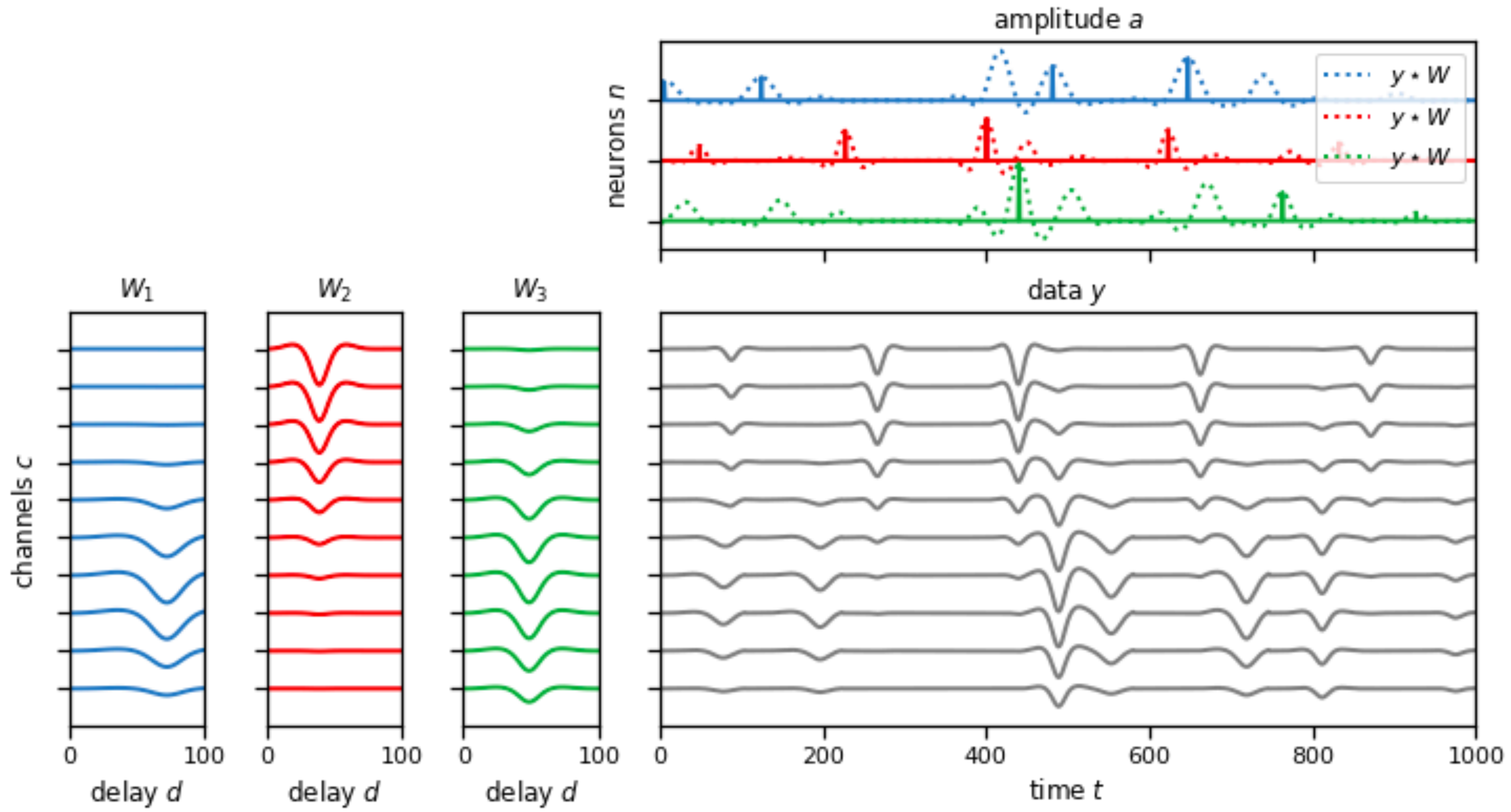
# Cross-Correlation

## With multiple input & output channels

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

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

$$= \begin{pmatrix} \sum_{n=1}^{N} \sum_{d=-\infty}^{\infty} y_{n,t+d} w_{1,n,d} \\ \vdots \\ \sum_{n=1}^{N} \sum_{d=-\infty}^{\infty} y_{n,t+d} w_{K,n,d} \end{pmatrix}$$



# 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 $(\text{out\_channels})$. 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, $\text{in\_channels}$ should be divisible by the number of groups. Default: 1

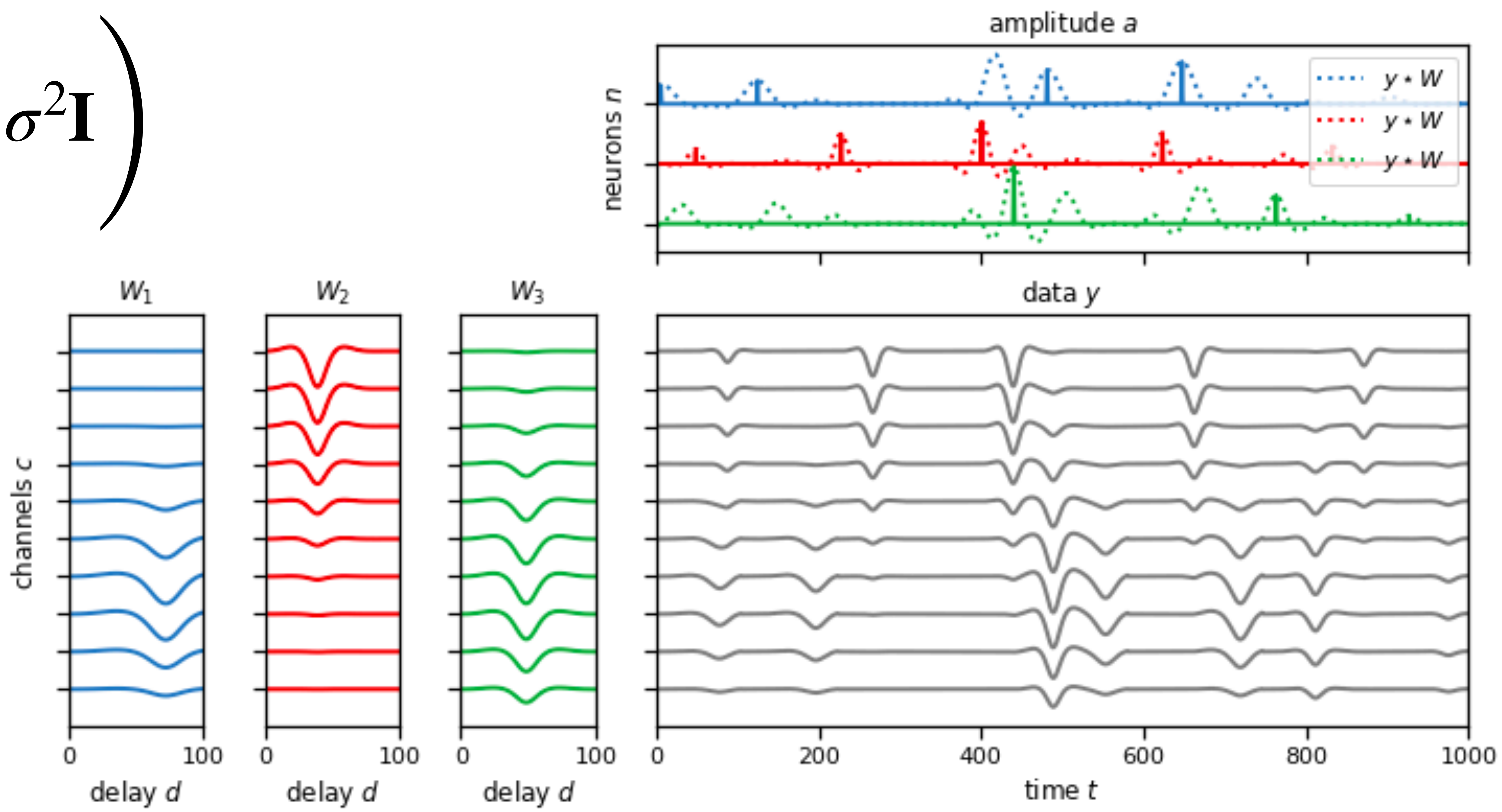
# Spike sorting by deconvolution

# Probabilistic Model

## Likelihood

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

$$p(\mathbf{X} \mid \mathbf{A}, \mathbf{W}) = \prod_{t=1}^{T} \mathcal{N}\left(\mathbf{x}_t \,\middle|\, \sum_{k=1}^{K} [\mathbf{a}_k \circledast \mathbf{W}_k]_t, \sigma^2 \mathbf{I}\right)$$



# Probabilistic Model

## Prior on spike amplitudes

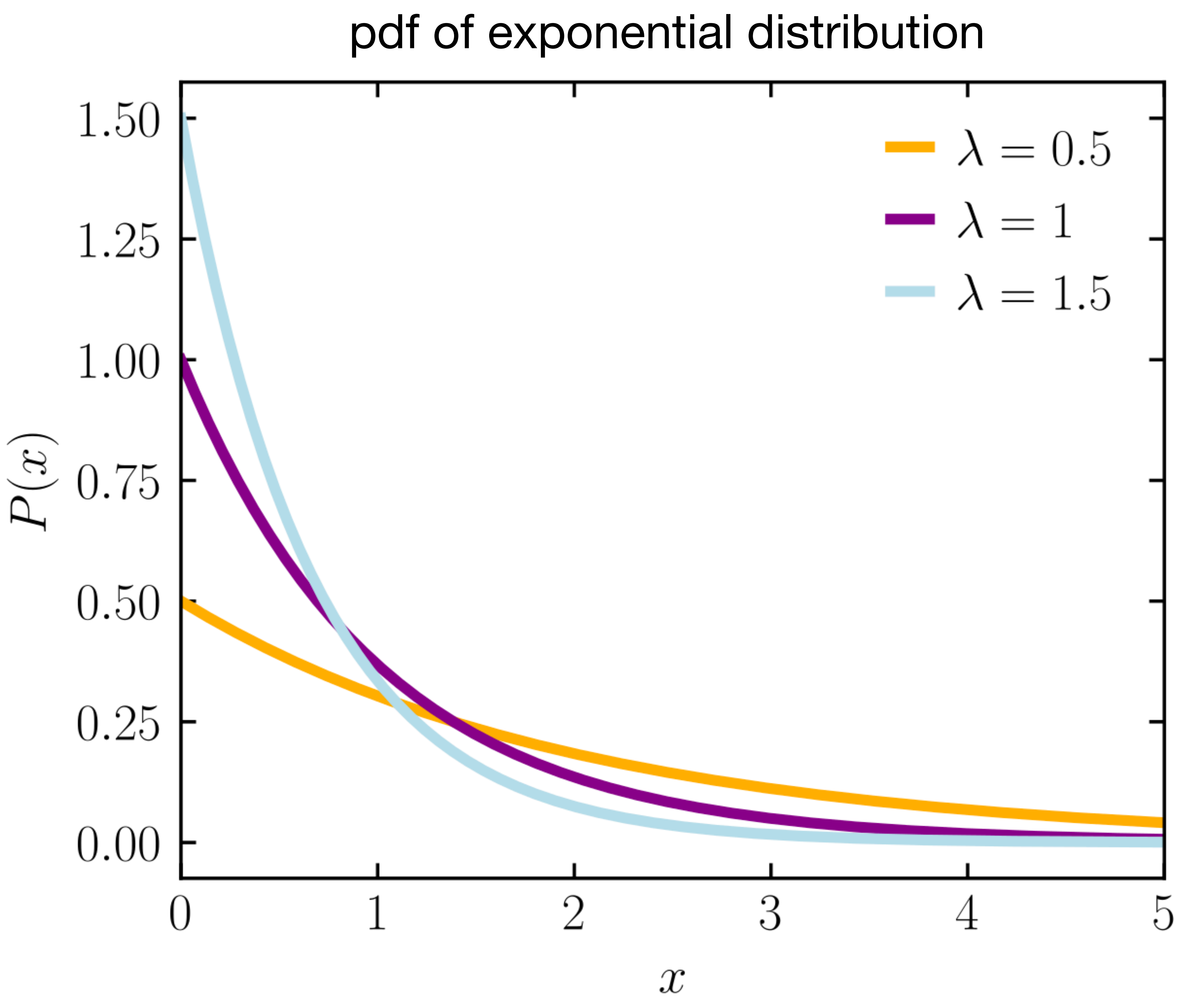
- Assume the spike amplitudes are drawn from an exponential distribution.

$$a_{k,t} \sim \mathrm{Exp}(\lambda)$$

- 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 use a heuristic solution in this week's lab.

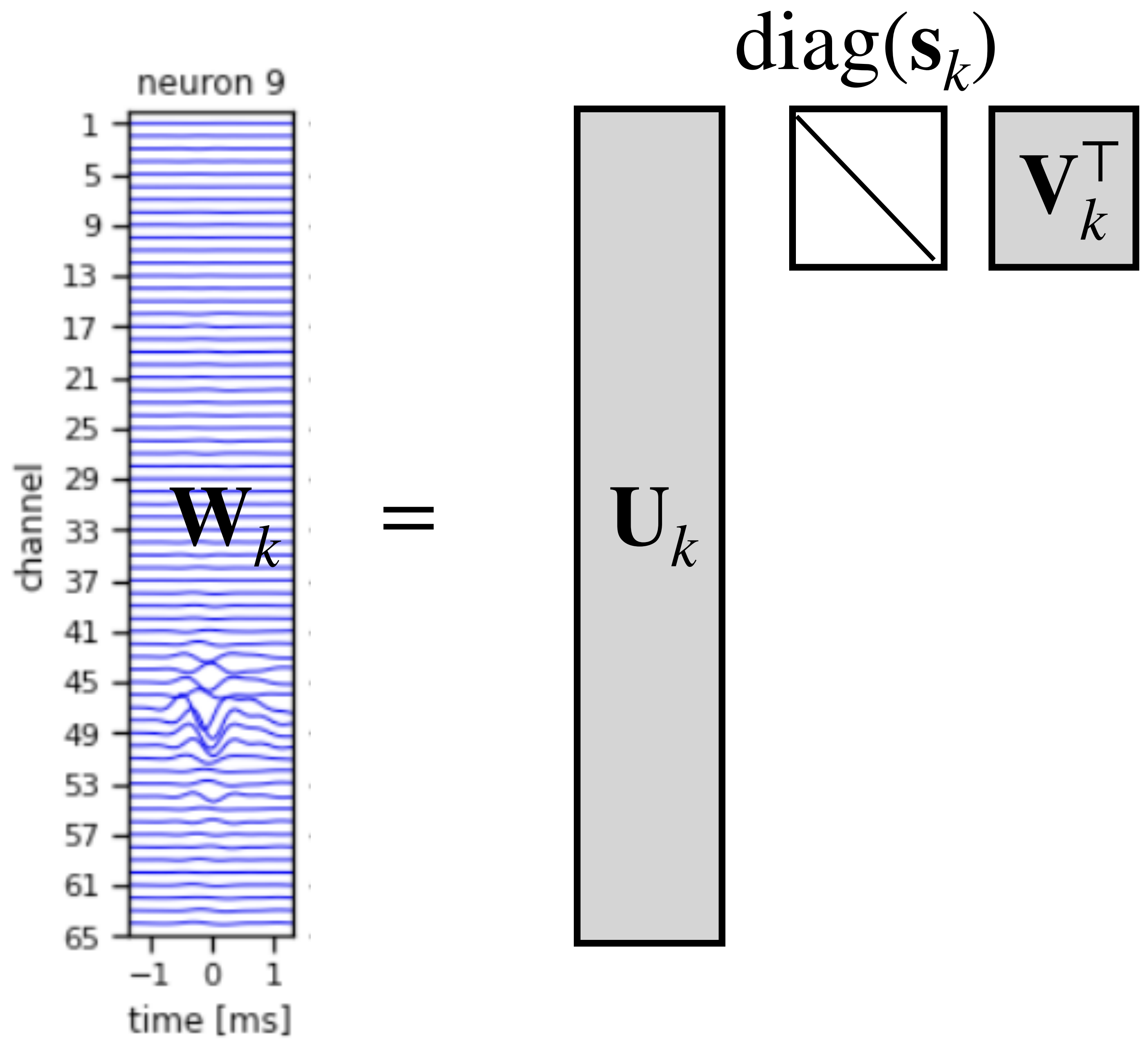



https://en.wikipedia.org/wiki/Exponential_distribution

# Probabilistic Model

## Scale invariance via Frobenius norm constraint

- What is the generalization of the unit-norm constraint $\mathbf{w}_k \in \mathbb{S}_{N-1}$ to matrices?
- Assume the matrix $\mathbf{W}_k \in \mathbb{R}^{N \times D}$ has unit **Frobenius norm** $||\mathbf{W}_k||_\mathrm{F} = 1$.



# Probabilistic Model

## Aside: The Frobenius norm and the SVD

- The Frobenius norm is the $\ell_2$ norm of the flattened matrix,

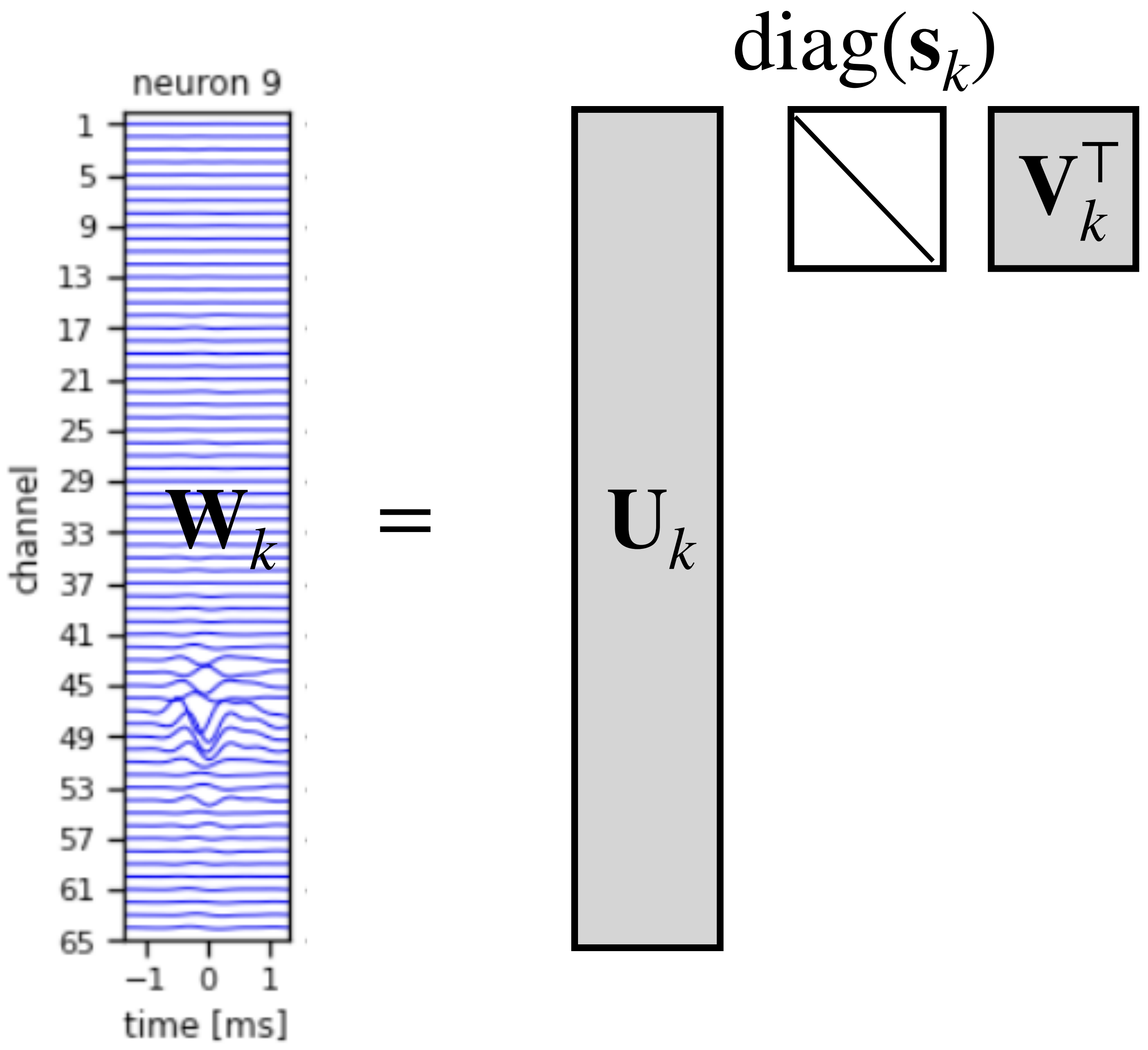
$$\|\mathbf{W}\|_F^2 = \sum_{n=1}^{N} \sum_{d=1}^{D} w_{n,d}^2 = \mathrm{vec}(\mathbf{W})^\top \mathrm{vec}(\mathbf{W}) = \|\mathrm{vec}(\mathbf{W})\|_2^2$$

- We can also write it as a trace,

$$\|\mathbf{W}\|_F = \sqrt{\mathrm{Tr}(\mathbf{W}^\top \mathbf{W})}$$

- Or in terms of the singular values,

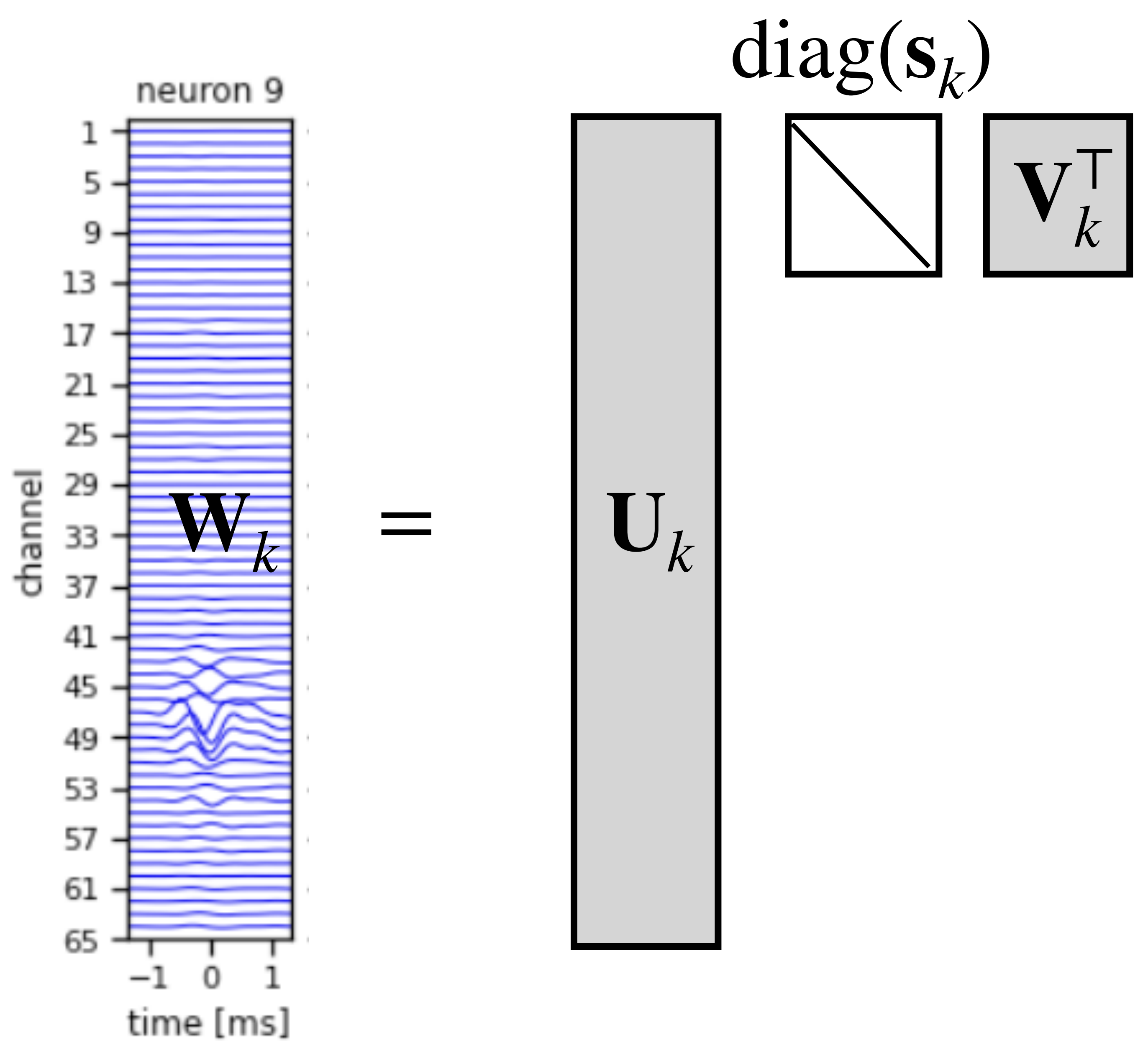
$$\|\mathbf{W}\|_\mathrm{F} = \|\mathbf{s}\|_2$$



# Probabilistic Model

## Scale invariance via Frobenius norm constraint

- What is the generalization of the unit-norm constraint $\mathbf{w}_k \in \mathbb{S}_{N-1}$ to matrices?
- Assume the matrix $\mathbf{W}_k \in \mathbb{R}^{N \times D}$ has unit **Frobenius norm** $\|\mathbf{W}_k\|_\mathrm{F} = 1$.
- This is equivalent to constraining the **singular values** to be normalized $\|\mathbf{s}_k\|_2 = 1$.



# Probabilistic Model

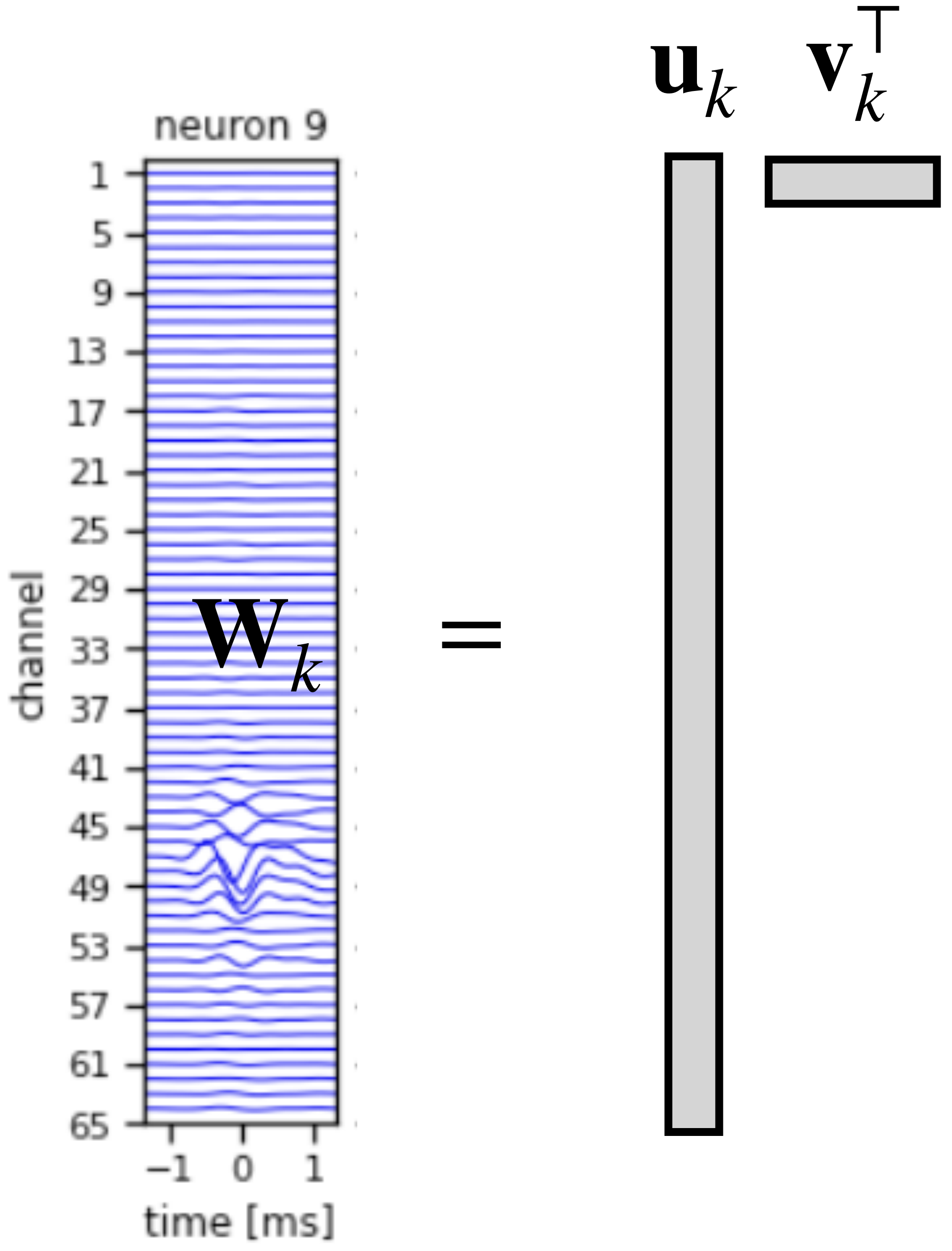
## Low-rank constraint

- This view suggests a further assumption: constraint the **rank** of the templates as well.

- If we constrain it to be rank 1 (i.e., only one nonzero singular value), then

$$\mathbf{W}_k = \mathbf{u}_k \mathbf{v}_k^\top$$

where $\mathbf{u}_k \in \mathbb{S}_{N-1}$ is the **spatial footprint** and $\mathbf{v}_k \in \mathbb{S}_{D-1}$ is the **temporal profile**.



# MAP estimation

# Maximum a posteriori estimation

## Coordinate ascent

- Initialize templates $\mathbf{W}$ and set $\mathbf{A} = 0$.
- Iterate until convergence:
    - For neuron $k = 1, \ldots, K$:

      a. Optimize **amplitudes** $a_k$ for neuron $k$.

      b. Optimize **templates** $\mathbf{W}_k$ for neuron $k$.

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

# Maximum a posteriori estimation

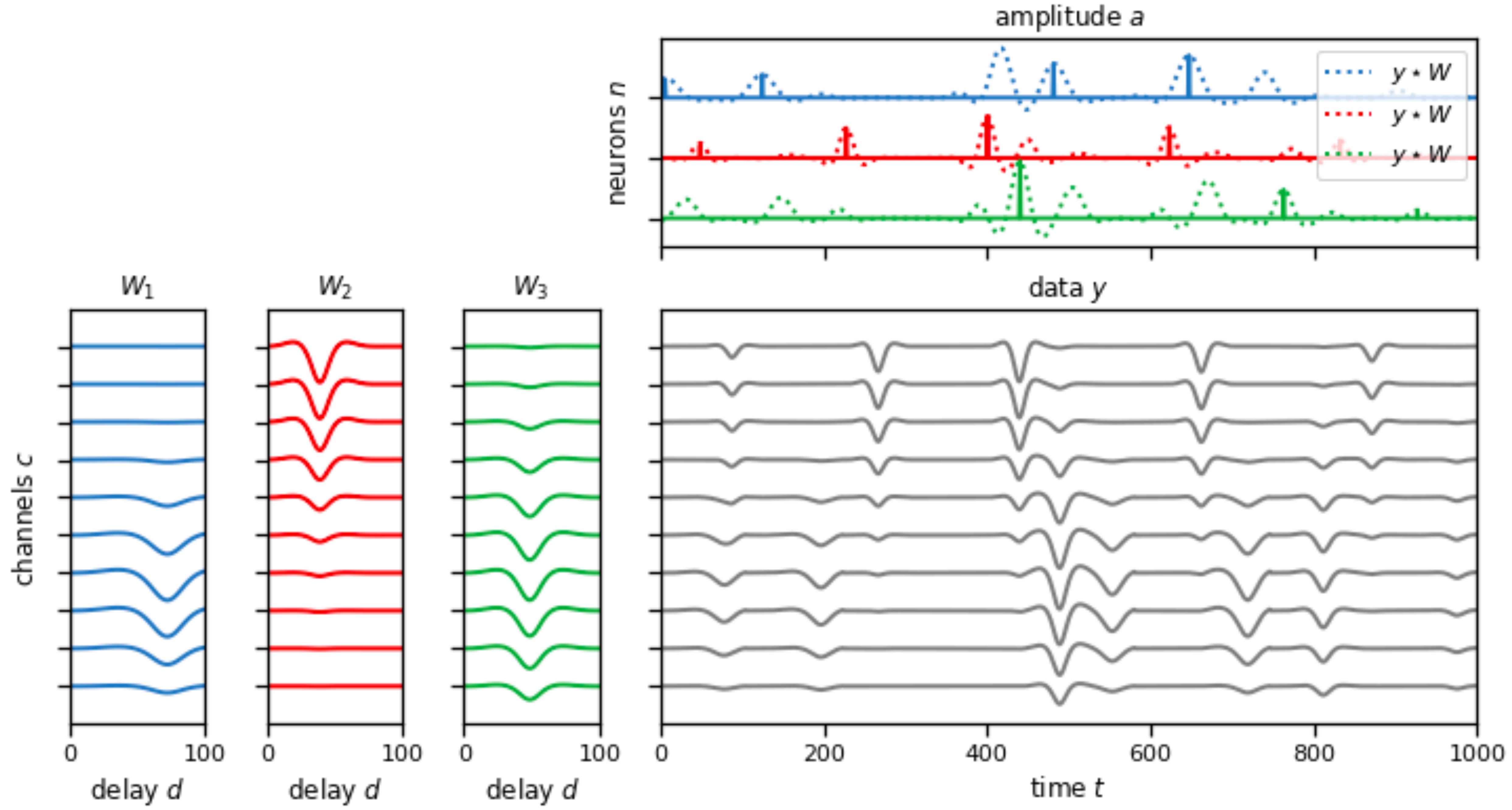
## Optimizing the amplitudes

As a function of $\mathbf{a}_k$,

$$\log p(\mathbf{X}, \mathbf{W}, \mathbf{A}) =$$
$$= \log p(\mathbf{X} \mid \mathbf{A}, \mathbf{W}) + \log p(\mathbf{a}_k; \lambda)$$
$$= -\frac{1}{2\sigma^2} \|\mathbf{R} - \mathbf{a}_k \circledast \mathbf{W}_k\|_{\mathrm{F}}^2 + \log p(\mathbf{a}_k) + \mathrm{c}$$

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

$$\mathbf{R} = \mathbf{X} - \sum_{j \neq k} [\mathbf{a}_j \circledast \mathbf{W}_j]$$



# Maximum a posteriori estimation

## Optimizing the amplitudes

Expanding the square

$$\begin{aligned}\log p(\mathbf{X}, \mathbf{W}, \mathbf{A}) &= -\frac{1}{2\sigma^2}\|\mathbf{R} - \mathbf{a}_k \circledast \mathbf{W}_k\|_\mathrm{F}^2 + \log p(\mathbf{a}_k) + \mathrm{c} \\ &= \underbrace{-\frac{1}{2\sigma^2}\|\mathbf{a}_k \circledast \mathbf{W}_k\|_\mathrm{F}^2}_{\mathscr{L}_2(\mathbf{a}_k)} + \underbrace{\frac{1}{\sigma^2}\langle \mathbf{R}, \mathbf{a}_K \circledast \mathbf{W}_k \rangle_\mathrm{F}}_{\mathscr{L}_1(\mathbf{a}_k)} + \log p(\mathbf{a}_k) + \mathrm{c}\,.\end{aligned}$$

where $\mathbf{r}_t \in \mathbb{R}^N$ is the $t$-th column of the residual $\mathbf{R}$.

# Maximum a posteriori estimation
## Optimizing the amplitudes

Further expanding the quadratic term,

$$
\begin{aligned}
\mathscr{L}_2(\mathbf{a}_k) &= -\frac{1}{2\sigma^2}\|\mathbf{a}_k \circledast \mathbf{W}_k\|_F^2 \\
&= -\frac{1}{2\sigma^2}\sum_{t=1}^{T}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} a_{k,t-d}^2 w_{k,n,d}^2 + 2\sum_{d=1}^{D}\sum_{d'=1}^{d-1} a_{k,t-d}a_{k,t-d'}w_{k,n,d}w_{k,n,d'}\right) \\
&\approx -\frac{1}{2\sigma^2}\sum_{t=1}^{T} a_{k,t}^2 \|\mathbf{W}_k\|_F^2 \\
&= -\frac{1}{2\sigma^2}\sum_{t=1}^{T} a_{k,t}^2
\end{aligned}
$$

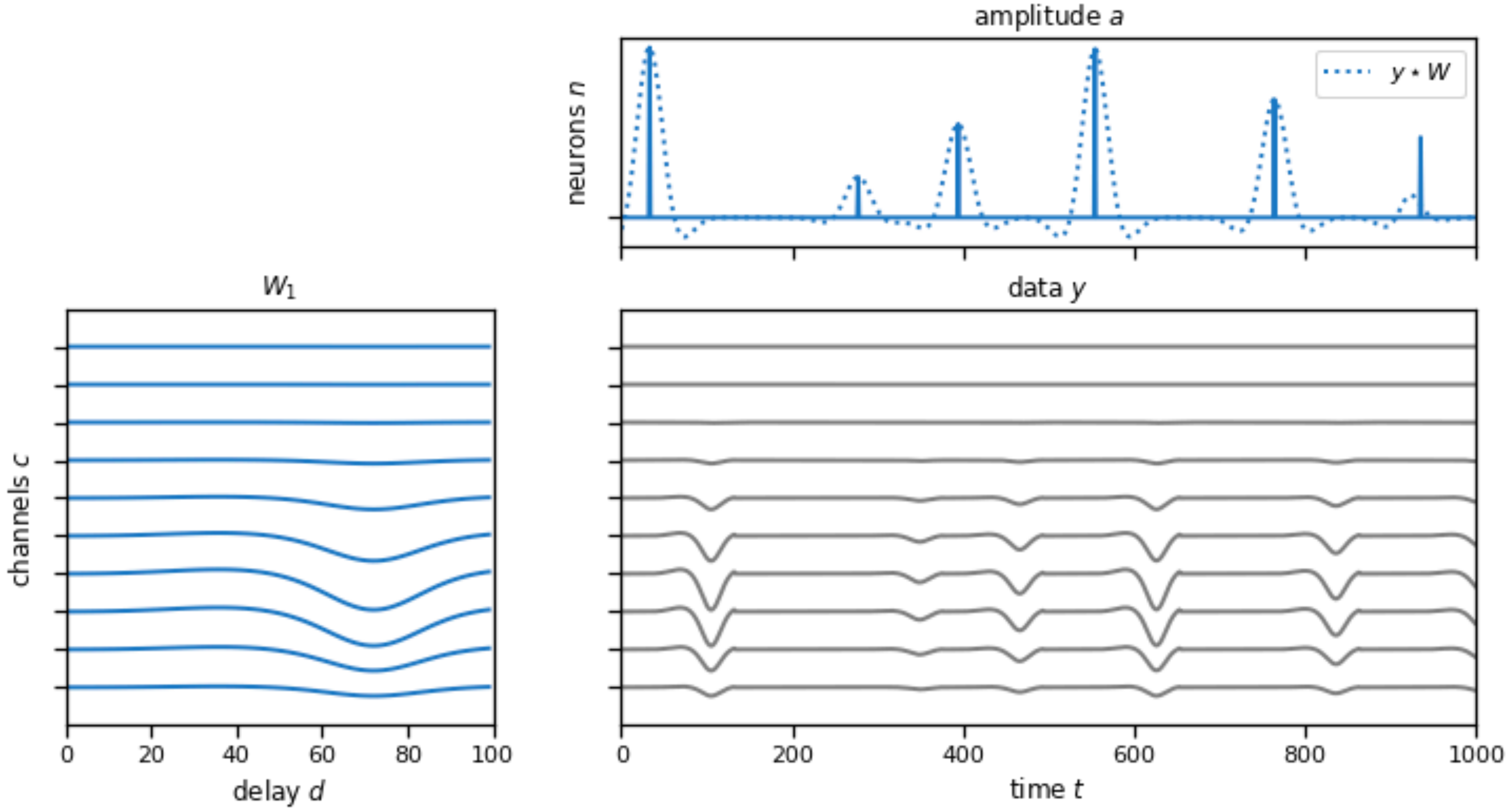
with equality when nonzero entries (i.e. “spikes”) of $\mathbf{a}_k$ are separated by at least $D$ samples.

# Maximum a posteriori estimation

## Optimizing the amplitudes

Now take the linear term…

$$
\begin{aligned}
\mathscr{L}_1(\mathbf{a}_k) &= \frac{1}{\sigma^2}\langle \mathbf{R}, \mathbf{a}_k \circledast \mathbf{W}_k \rangle \\
&= \frac{1}{\sigma^2}\sum_{t=1}^{T}\sum_{n=1}^{N} r_{n,t}[\mathbf{a}_k \circledast \mathbf{w}_{k,n}]_t \\
&= \frac{1}{\sigma^2}\sum_{t=1}^{T}\sum_{n=1}^{N}\sum_{d=1}^{D} a_{k,t-d} r_{n,t} w_{k,n,d} \\
&= \frac{1}{\sigma^2}\sum_{t=1}^{T} a_{k,t} \sum_{n=1}^{N}\sum_{d=1}^{D} r_{n,t+d} w_{k,n,d} \\
&= \frac{1}{\sigma^2}\sum_{t=1}^{T} a_{k,t} [\mathbf{R} \star \mathbf{W}_k]_t
\end{aligned}
$$




where $[\mathbf{R} \star \mathbf{W}_k]_t$ is the cross-correlation of the residual and the template for neuron $k$.

# Maximum a posteriori estimation

## Optimizing the amplitudes

Putting it all together

$$\mathscr{L}(\mathbf{a}_k) = \sum_{t=1}^{T} \left[ -\frac{1}{2\sigma^2} a_{k,t}^2 + \frac{1}{\sigma^2} \mu_{k,t} a_{k,t} - \lambda a_{k,t} \right] + \mathrm{c},$$

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

# Maximum a posteriori estimation

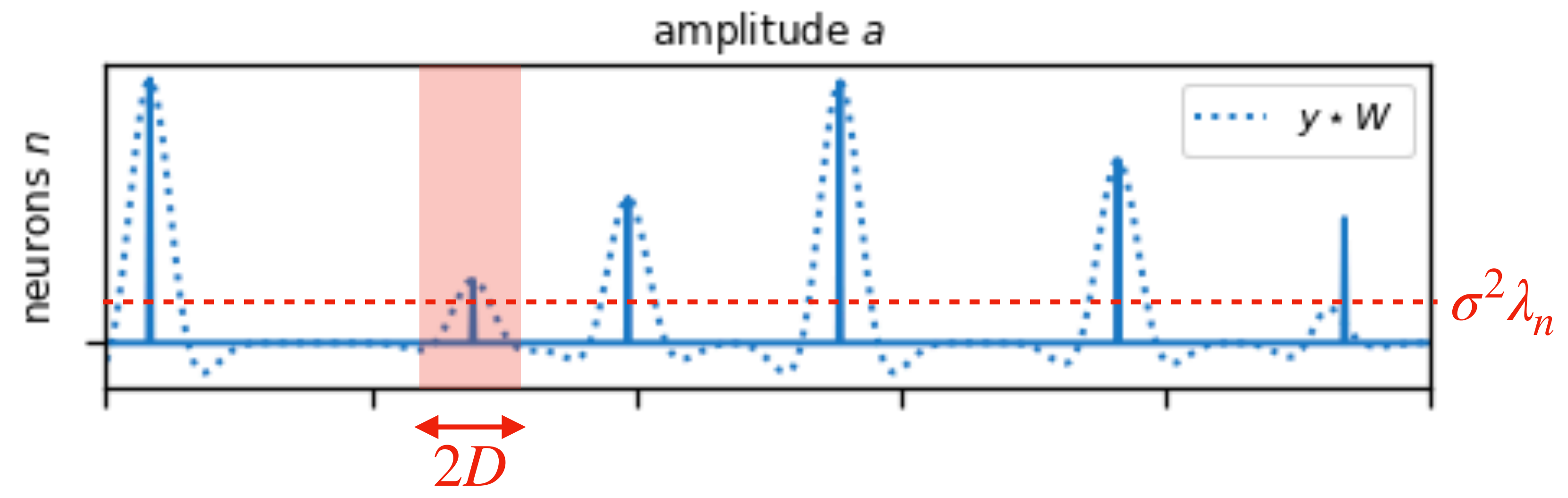
## Completing the square and solving for the optimal amplitudes

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

$$a_{k,t} = \max\left\{0, \mu_{k,t} - \sigma^2\lambda\right\}$$

- However, we also want spikes to be well-separated; i.e. $a_{k,t} > 0 \implies a_{k,t+d} = 0$ 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

As a function of $\mathbf{W}_k$

$$\log p(\mathbf{X}, \mathbf{A}, \mathbf{W}) = \frac{1}{2\sigma^2} \sum_{t=1}^{T} \langle a_{k,t} \mathbf{R}_t, \mathbf{W}_k \rangle + c'$$

$$= \frac{1}{2\sigma^2} \left\langle \sum_{t=1}^{T} a_{k,t} \mathbf{R}_t, \mathbf{W}_k \right\rangle + c'$$

where

$$\mathbf{R}_t = \begin{bmatrix} r_{1,t} & \cdots & r_{1,t+D} \\ \vdots & & \vdots \\ r_{n,t} & \cdots & r_{n,t+D} \end{bmatrix}$$

is a slice of the residual matrix (`R[:,t:t+D]` in code).

# Maximum a posteriori estimation

## Optimizing the templates

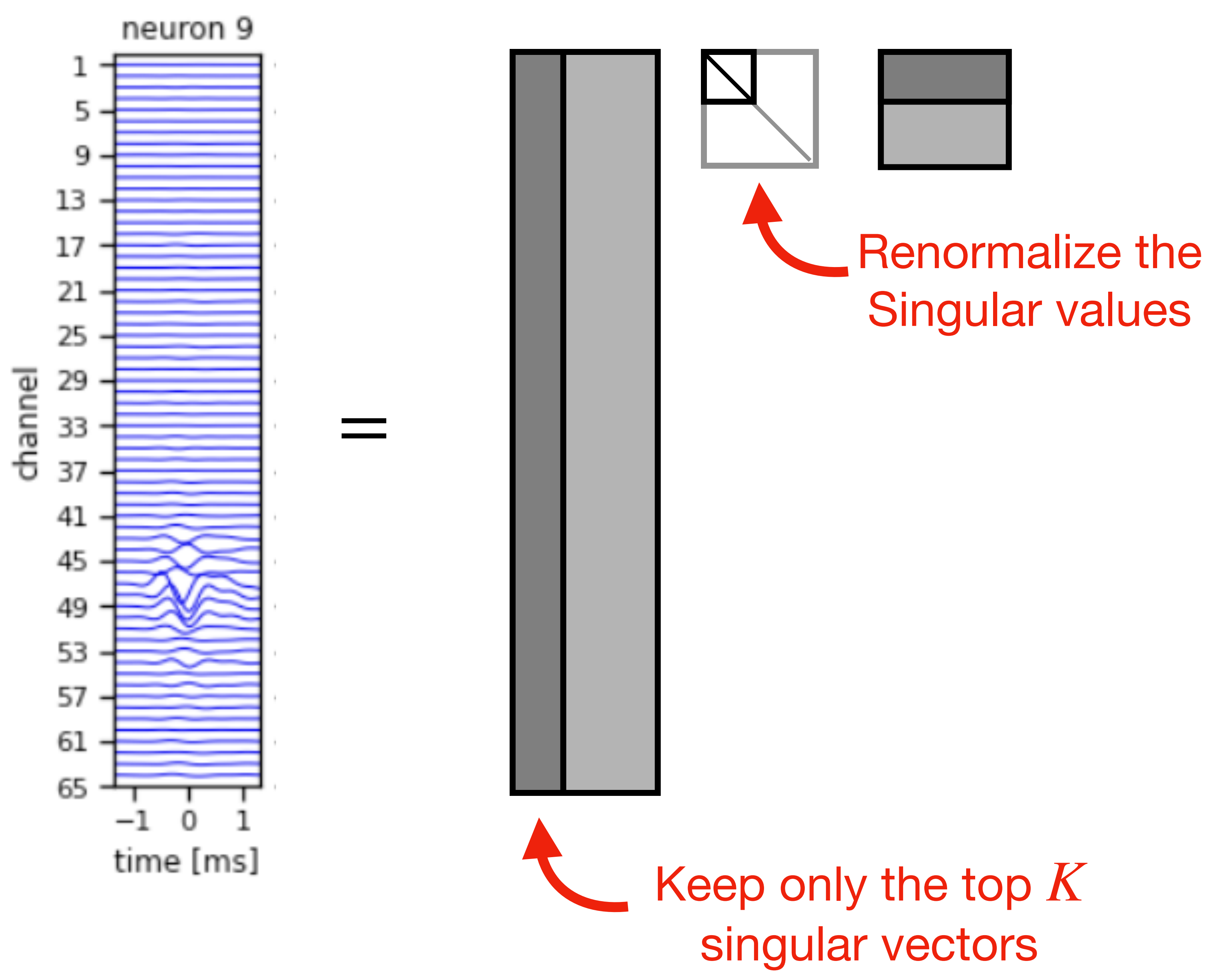
We want to maximize this log joint probability over the space of low-rank, unit-norm matrices,

$$\mathbf{W}_k^\star = \arg \max_{\mathbf{W}_k \in \mathbb{S}_R^{N,D}} \left\langle \sum_{t=1}^{T} a_{k,t} \mathbf{R}_t, \mathbf{W}_k \right\rangle$$

The solution is to set the waveform matrix "proportional to" the weighted sum of residual matrices by taking its SVD and renormaling the singular values.

$$\mathbf{W}_k^\star = \sum_{r=1}^{R} \bar{s}_r \mathbf{u}_r \mathbf{v}_r^\top \quad \text{where} \quad \bar{s}_r = \frac{s_r}{\sqrt{\sum_{r'=1}^{R} s_{r'}^2}}.$$

$$\sum_{t=1}^{T} a_{k,t} \mathbf{R}_t \qquad \mathbf{U}_k \quad \mathrm{diag}(\mathbf{s}_k) \quad \mathbf{V}_k^\top$$



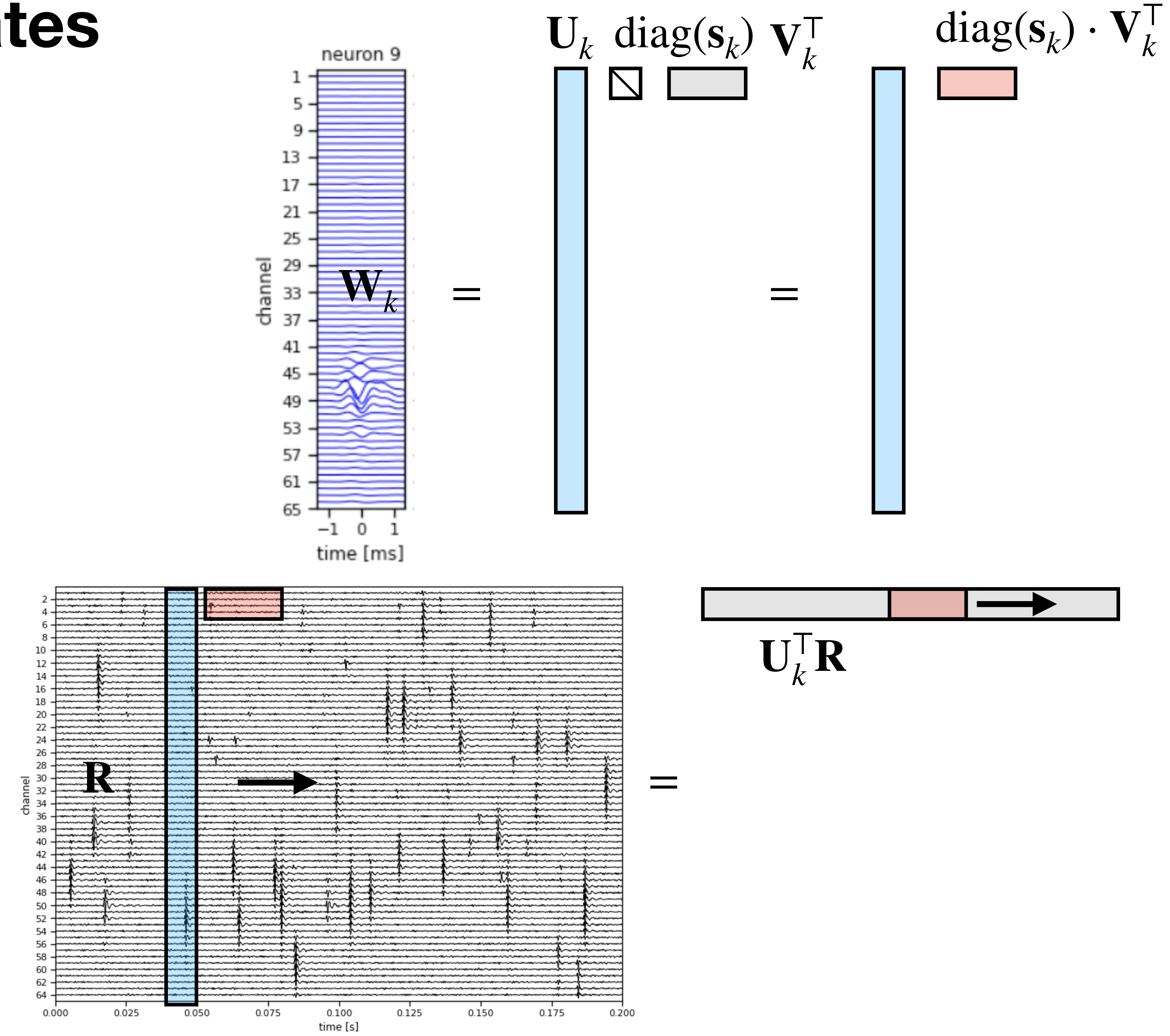
# More efficient computation

## Leveraging the low-rank templates

We can compute the "scores" for amplitude updates more efficiently by leveraging the low-rank templates,

$$
\begin{aligned}
[\mathbf{R} \star \mathbf{W}_k]_t &= \sum_{n=1}^{N} \sum_{d=1}^{D} r_{n,t+d} w_{k,n,d} \\
&= \sum_{d=1}^{D} \mathbf{r}_{t+d}^\top \mathbf{w}_{k,:,d} \\
&= \sum_{d=1}^{D} \mathbf{r}_{t+d}^\top \mathbf{U}_k \mathbf{S}_k \mathbf{v}_{k,:,d} \\
&= \sum_{d=1}^{D} (\mathbf{U}_k^\top \mathbf{r}_{t+d})^\top [\mathbf{S}_k \mathbf{V}_k^\top]_{:,d} \\
&= [(\mathbf{U}_k^\top \mathbf{R}) \star (\mathbf{S}_k \mathbf{V}_k^\top)]_t
\end{aligned}
$$

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



# Conclusion

- We developed a basic spike sorting model that was good for building intuition, but not very practical.
- We developed a new model for the voltage in terms of a superposition of templates convolved with spike amplitudes for each neuron.
    - Along the way, we learned about convolution and cross-correlation.
- 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

- **Simple Spike Sorting** and **Spike Sorting by Deconvolution** course notes.
- Convolution and cross-correlation:
    - Chapter 9 of *The Deep Learning Book* (deeplearningbook.org/contents/convnets.html)
    - Start reading up on PyTorch convolutions! https://pytorch.org/docs/stable/generated/torch.nn.functional.conv1d.html
- Spike sorting:
    - Pachitariu, Marius, Shashwat Sridhar, and Carsen Stringer. "Solving the spike sorting problem with Kilosort." bioRxiv (2023).
        - The model we presented is a slightly modified version of *Kilosort*