UCI

Brain-Inspired Learning Machines

Pattern Recognition II: Deep Artificial Neural Networks

Emre Neftci

Department of Cognitive Sciences, UC Irvine,

October 14, 2016

Gradient-Descent Learning in Neural Networks

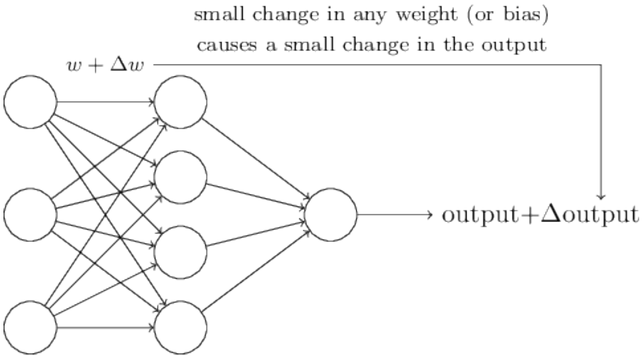
Reminder:The Perceptron Learning Rule

Error = Number of Misclassified Samples

To minimize error, repeat for every data sample:

$$\text{new } w_i = w_i + \eta(\text{target} - \text{output})x_i \quad \text{for every i,}$$
$$\text{new } b = b + \eta(\text{target} - \text{output}),$$

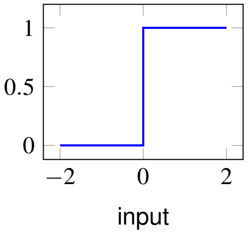
where η is a “learning rate”.



Problem with threshold units: A tiny Δw can induce a flip (large Δ output)

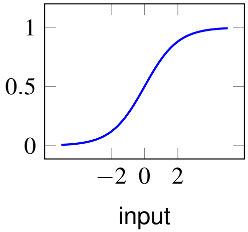
Threshold unit

$$\text{output} = \begin{cases} 0 & \text{if } input \le 0 \\ 1 & \text{if } input > 0 \end{cases}$$



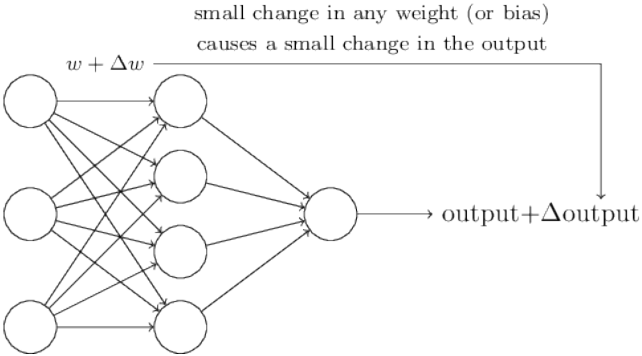
Sigmoid unit

$$\text{output} = \sigma(\text{input}) = \frac{1}{1 + e^{-input}}.$$

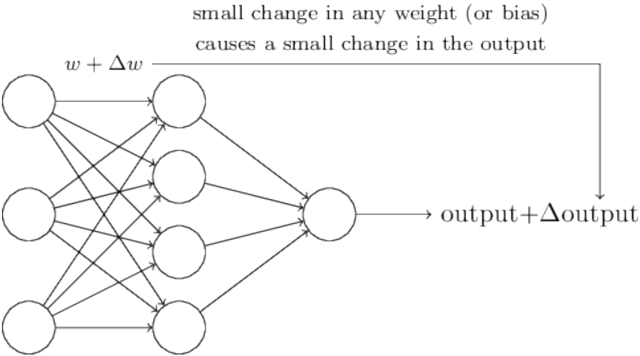


$$input = \sum_j w_j x_j + b$$

The sigmoid unit is smoother (its derivative is continuous)



$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j$$



$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j$$

Derivative of Sigmoid:

$$\frac{\partial\, \text{output}}{\partial w_j}$$

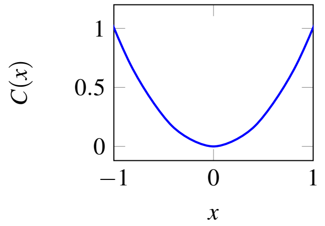
Cost (Error) function: a number representing how the Neural Network performed.

- Perceptrons: Cost function = Number of Misclassified Samples
- Sigmoid Units: Cost function = Mean Squared Error (MSE)

$$C_{\mathrm{MSE}} = \sum_{\text{training set}} \sum_{i} (output_i - target_i)^2.$$

Objective: Minimize the cost function.

Example: Find x that minimizes $C(x) = x^2$



Incremental change in Δx:

$$\Delta C \approx \underbrace{\frac{\partial C}{\partial x}}_{=\text{Slope of } C(x)} \Delta x \tag{1}$$

With $\Delta x = -\eta \frac{\partial C}{\partial x}$, $\Delta C \approx -\eta \left(\frac{\partial C}{\partial x}\right)^2$

Gradient Descent for finding the optimal x

$$\text{new } x = \text{old } x - \eta \frac{\partial C}{\partial x} \tag{2}$$

Gradient Descent

$$\Delta w_{ij} = -\eta \frac{\partial C_{\text{MSE}}}{\partial w_{ij}}$$
$$\Delta b_i = -\eta \frac{\partial C_{\text{MSE}}}{\partial b_i}$$

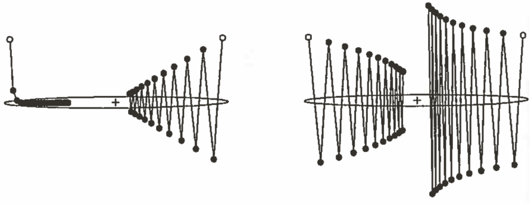
Cost function $C_{\text{MSE}}(w, b)$:

$$C_{\text{MSE}}(w,b) = \sum_{\text{training set}} \sum_i (output_i - target_i)^2.$$
$$\frac{\partial C_{\text{MSE}}}{\partial w_{ij}} = 2 \sum_{\text{training set}} \sum_i (output_i - target_i) \frac{\partial output_i}{\partial w_{ij}}.$$

For the Sigmoid neuron:

$$\frac{\partial output_i}{\partial w_{ij}} = output_i(1 - output_i) input_j$$

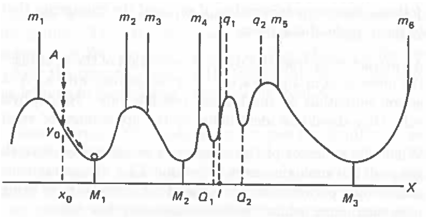
- Adjusting the Learning rate η:



η increasing from left to right

η too small = slow convergence, η too large = no convergence

- Gradient Descent can get stuck in local minima:

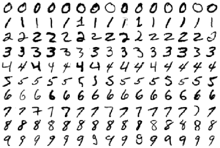


In practice, not a big problem, but it can slow down learning.

Stochastic can escape local minima

Multinomial Classification with “One-Hot” Representation

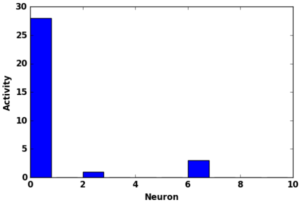
In many datasets, targets are discrete classes, but neural networks units output numbers in the range $[0, 1]$



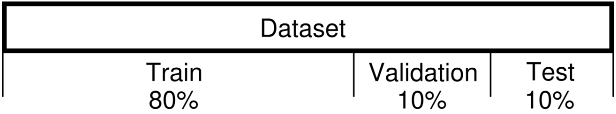
e.g. MNIST: 10 classes

Representing classes with an output layer:

- Output Layer: one unit per label
- Transform label=k to target vector=$(0, \ldots, \underbrace{1}_{\text{position k}}, \ldots 0)$
- Predicted class is defined as the position of output neuron with the highest activity

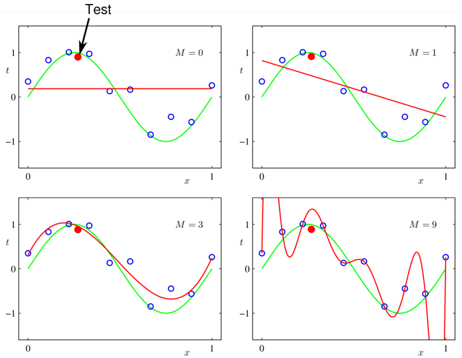


Training, Testing and Validation Datasets



- Training set: Apply learning rule to sampling in dataset
- Testing set: Set that is never used during training to test the classifier
- Validation set: Set for monitoring overfitting and testing algorithm with different learning parameters

M is the degree of a fitting polynomial: The higher M, the more parameters there are.



Too few parameters: Underfitting, Too many parameters: Overfitting

Regularization is a technique used to constrain the complexity of the neural network by introducing *a priori* knowledge

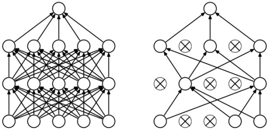
There are many regularization techniques. Most common:

- L^p **norm regularization**: punish large weights ($p \in \mathbb{N}$)

$$\text{New cost function } = C_{MSE} + \lambda \sum_{ij} w_{ij}^p$$

  λ is the regularization parameter.

- **DropOut**: During training, randomly drop 50% of the outputs

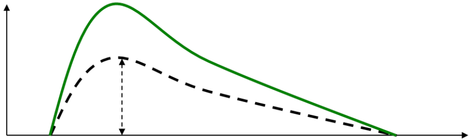


(a) Standard Neural Net (b) After applying dropout.

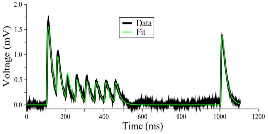
- **Data Augmentation**:

Synaptic Plasticity: Learning in Spiking Neural Networks

Long-Term Plasticity



Short-Term Plasticity



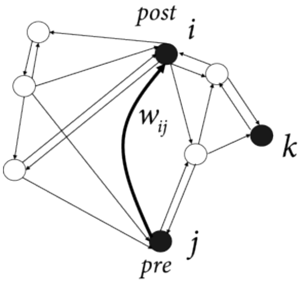
Tsodyks and Markram, *Proceedings of the National Academy of Sciences of the USA*, 1997

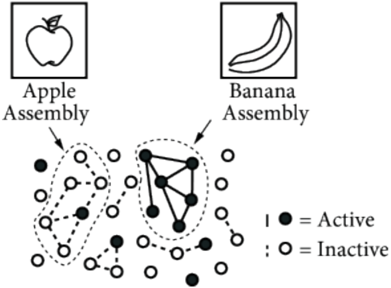
- Induced over seconds, persistance over >10 hours
- Many mechanisms: Change in number of Receptors, Release Probability, ...

- Induced over fractions of a second
- Recovery over seconds
- Change in probability of vesicle release, ...

More on synaptic plasticity Mechanisms: Feldman, *Annual review of neuroscience*, 2009

Slide modified from Gerstner *et al.* 2015





When an axon of cell j repeatedly or persistently takes part in activating cell i, then j's efficiency as one of the cells activating i is increased

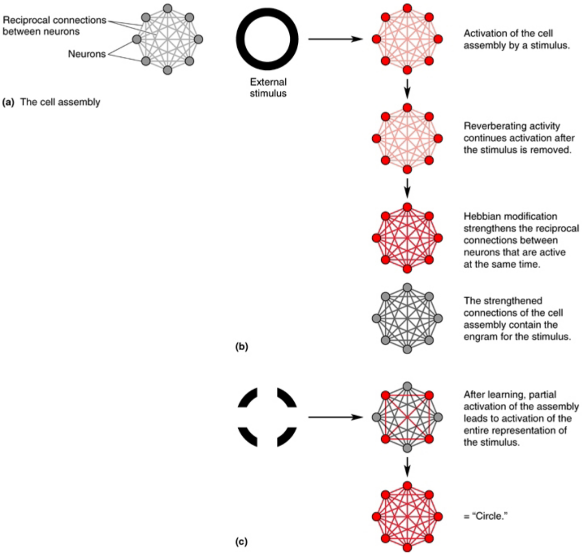
Hebb,, 1949

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = \eta \nu_i \nu_j$$

- Plasticity rule operating on local information
- Captures correlations in activity
- Unsupervised

"Neurons that fire together wire together"

Hebb's Cell Assembly



Generalized Hebbian learning: Introduce dependence on pre-synaptic and post-synaptic activities, and the weight itself:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = F(w_{ij}, \nu_i, \nu_j)$$
$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + a_2(w_{ij})\nu_i\nu_j + \dots \tag{3}$$

Pre	On	Off	On	Off
Post	On	On	Off	Off

Gerstner and Kistler,, 2002

Generalized Hebbian learning: Introduce dependence on pre-synaptic and post-synaptic activities, and the weight itself:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = F(w_{ij}, \nu_i, \nu_j)$$
$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + a_2(w_{ij})\nu_i\nu_j + \ldots \quad (3)$$

Pre	On	Off	On	Off
Post	On	On	Off	Off
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i\nu_j$	+	0	0	0

Gerstner and Kistler,, 2002

Generalized Hebbian learning: Introduce dependence on pre-synaptic and post-synaptic activities, and the weight itself:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) &= F(w_{ij}, \nu_i, \nu_j) \\ \frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) &= a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + a_2(w_{ij})\nu_i\nu_j + \dots \end{aligned} \tag{3}$$

Pre Post	On On	Off On	On Off	Off Off
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i\nu_j$	+	0	0	0
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i\nu_j - c$	+	-	-	-

Gerstner and Kistler,, 2002

Generalized Hebbian learning: Introduce dependence on pre-synaptic and post-synaptic activities, and the weight itself:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = F(w_{ij}, \nu_i, \nu_j)$$
$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + a_2(w_{ij})\nu_i\nu_j + \dots \tag{3}$$

Pre	On	Off	On	Off
Post	On	On	Off	Off
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i\nu_j$	+	0	0	0
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i\nu_j - c$	+	-	-	-
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto (\nu_i - c)\nu_j$	(+)	0	-	0

Gerstner and Kistler,, 2002

Generalized Hebbian learning: Introduce dependence on pre-synaptic and post-synaptic activities, and the weight itself:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) &= F(w_{ij}, \nu_i, \nu_j) \\ \frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) &= a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + a_2(w_{ij})\nu_i\nu_j + \dots \end{aligned} \tag{3}$$

Pre Post	On On	Off On	On Off	Off Off
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i \nu_j$	+	0	0	0
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto \nu_i \nu_j - c$	+	-	-	-
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto (\nu_i - c)\nu_j$	(+)	0	-	0
$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \propto (\nu_i - \langle \nu_i \rangle)(\nu_j - \langle \nu_j \rangle)$	+	-	-	+

Gerstner and Kistler,, 2002

Modulated Hebb rule: Neuromodulators + Hebbian Learning

$$\frac{\mathrm{d}}{\mathrm{d}t}w_{ij}(t) = F(w_{ij}, \nu_i, \nu_j, mod(t)) \tag{4}$$

Example modulators can be rewards, error, attention, novelty.

Examples:
Reinforcement learning:

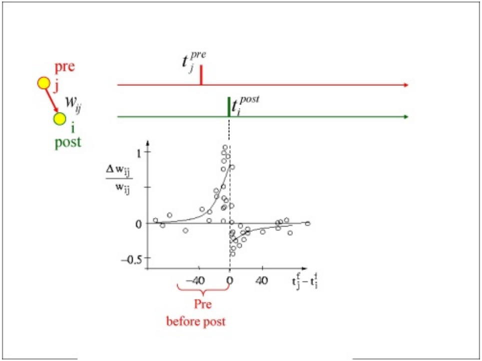
$$\frac{\mathrm{d}}{\mathrm{d}t}w_{ij}(t) \propto reward(t)\nu_i\nu_j \tag{5}$$

Florian, *Neural Computation*, 2007

Supervised Learning:

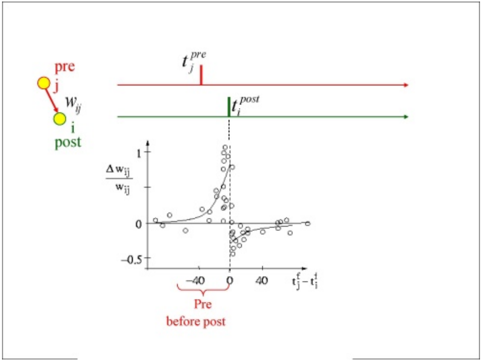
$$\frac{\mathrm{d}}{\mathrm{d}t}w_{ij}(t) = Error_i(t)a_1^{pre}\nu_j \tag{6}$$

Spike-Timing Dependent Plasticity



Bi and Poo, *J. Neurosci.*, 1998

Jesper Sjostrom and Wulfram Gerstner (2010), Scholarpedia, 5(2):1362.



Gerstner and Kistler,, 2002

Spike-Time Dependent Plasticity Rule:

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_i^n - t_j^f) \tag{7}$$

W: Learning Window
t_i^n: nth spike time of post-synaptic neuron i
t_j^f: fth spike time of pre-synaptic neuron i

Jesper Sjostrom and Wulfram Gerstner (2010), Scholarpedia, 5(2):1362.

On-line Implementation of the Spike-Time Dependent Plasticity Rule:

$$
\begin{aligned}
\tau_+ \frac{\mathrm{d}}{\mathrm{d}t} x_j &= -x_j + a_+ \sum_f \delta(t - t_j^f) \\
\tau_- \frac{\mathrm{d}}{\mathrm{d}t} y &= -y + a_- \sum_n \delta(t - t^n) \\
\frac{\mathrm{d}}{\mathrm{d}t} w_j &= x(t) \sum_n \delta(t - t^n) + y(t) \sum_f \delta(t - t_j^f)
\end{aligned} \tag{8}
$$

$\delta(t)$: Delta Dirac function (= spike at time t)
a_+: Amplitude of LTP
a_-: Amplitude of LTD
τ_+: Temporal window of LTP
τ_-: Temporal window of LTD

On-line Implementation of the Spike-Time Dependent Plasticity Rule:

$$
\begin{aligned}
\tau_+ \frac{\mathrm{d}}{\mathrm{d}t} x_j &= -x_j + a_+ \sum_f \delta(t - t_j^f) \\
\tau_- \frac{\mathrm{d}}{\mathrm{d}t} y &= -y + a_- \sum_n \delta(t - t^n) \\
\frac{\mathrm{d}}{\mathrm{d}t} w_j &= x(t) \sum_n \delta(t - t^n) + y(t) \sum_f \delta(t - t_j^f)
\end{aligned} \tag{8}
$$

$\delta(t)$: Delta Dirac function (= spike at time t)
a_+: Amplitude of LTP
a_-: Amplitude of LTD
τ_+: Temporal window of LTP
τ_-: Temporal window of LTD

More on white board

STDP implementation with Brian2

code/brian2_stdp.py

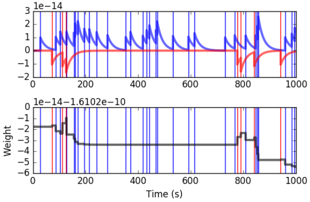
```
from brian2 import *
#Neuron parameters
Cm = 50*pF; gl = 1e-9*siemens; taus = 5*ms
sigma = 3/sqrt(ms)*mV; Vt = 10*mV; Vr = 0*mV;
#STDP Parameters
taupre = 20*ms; taupost = taupre
apre = .01e-12; apost = -apre * taupre / taupost * 1.05

eqs = '''
dv/dt = -gl*v/Cm + isyn/Cm + sigma*xi: volt (unless refractory)
disyn/dt = -isyn/taus : amp
'''

Pin = PoissonGroup(10, rates = 30*Hz)
P = NeuronGroup(1, eqs, threshold='v>Vt', reset='v = Vr',
                        method='euler', refractory=5*ms)
S = Synapses(Pin, P, '''w : 1
                        dx/dt = -x / taupre : 1
                        dy/dt = -y / taupost : 1''',
             on_pre='''isyn += w*amp
                        x += apre
                        w += y''',
             on_post='''y += apost
                        w += x''')

S.connect()
S.w = '(rand()-.5)*1e-9'
mon = StateMonitor(S, variables=['w','x','y'], record=range(5))
s_mon = SpikeMonitor(P)
p_mon = SpikeMonitor(Pin)

run(1*second, report='text')
```



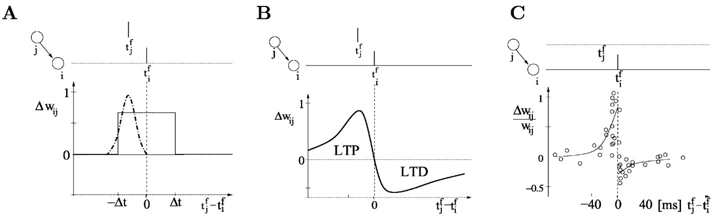


Fig. 3A–C. Learning window. The change Δw_{ij} of the synaptic efficacy depends on the timing of pre- and postsynaptic spikes. **A** The *solid line* indicates a rectangular time window as it is often used in standard Hebbian learning. The synapse is increased if the pre- and the postsynaptic neuron fire simultaneously with a temporal resolution Δt. The *dashed–dotted line* shows an asymmetric learning window useful for sequence learning (Herz et al. 1989; Gerstner and van Hemmen 1993). The synapse is strengthened if the presynaptic spike arrives slightly before the postsynaptic one, and is therefore partially 'causal' in firing it. **B** An asymmetric biphasic learning window as introduced in model studies of delay selection (Gerstner et al. 1996). A synapse is strengthened (long-term potentiation, *LTP*) if the presynaptic spike arrives slightly before the postsynaptic one, but is decreased (long-term depression, *LTD*) if the timing is reversed. The biphasic learning window is sensitive to the temporal contrast in the input. **C** Experimental results have confirmed the existence of biphasic learning windows. *Data points* redrawn after the experiments of Bi and Poo (1998)

If the pre- and post-synaptic neuron spike times are independent:

$$\left\langle \frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \right\rangle \cong \nu_i \nu_j \underbrace{\int W(s)\mathrm{d}s}_{\text{Area under learning window}} \qquad (9)$$

A More General Spike-Time Dependent Plasticity Rule

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} w_j = {} & a_0(w_{ij}) \\ & + a_1^{pre}(w_{ij}) \sum_f \delta(t - t_j^f) \\ & + a_1^{post}(w_{ij}) \sum_n \delta(t - t_i^n) \\ & + x(t) \sum_n \delta(t - t^n) + y(t) \sum_f \delta(t - t_j^f)\end{aligned} \tag{10}$$

Implements the generalized Hebb rule:

$$\langle \frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \rangle \cong a_0(w_{ij}) + a_1^{pre}(w_{ij})\nu_j + a_1^{post}(w_{ij})\nu_i + \nu_i \nu_j \int W(s)\mathrm{d}s \tag{11}$$

- Start with `code/brian2_activation_function.py`
- Find a parameter regime in which the activation function is continuous
- Find a function that fits the activation function (*e.g.* see Sigmoid, Softplus with ARP)
- Starting from least squares, compute the weight update dynamics $\frac{\mathrm{d}}{\mathrm{d}t}w$
- Write this rule in the form of generalized STDP
- Propose a spiking network diagram that would implement this rule (hand-in or upload a picture)
- Is your rule "local"? If not how many *different* non-local inputs to you need per neuron?

Optional (Hard):

- Start with `code/brian2_perceptron_learn.py`
- Create targets using one-hot representation
- Implement this rule in Brian2 using (generalized) STDP
- Train, Validate & Test