How to empirically estimate receptive field models by regression.

- We can estimate the receptive field properties of cells from electrical recordings of neurons by estimating the best model using *regression*. This assumes that the receptive field can be expressed by a parameterized function (e.g., a neuron model where the parameters are the weights).
- Recall that the receptive field properties of neurons are traditionally found by probing their response to different perceptual dimensions, such as orientations and frequency. This gives a classification of the type of the receptive field but does not specify its receptive field weights $\vec{w}$ unless strong assumptions are made (e.g., that the receptive field is a Gabor function).

Estimating receptive field models by regression.

- The regression method makes few assumptions about the functional form of the receptive field, but it does require more data. It needs a stimulus data set of $\mathcal{S} = \{(S^\mu, \vec{I}^\mu) : \mu = 1, ..., N\}$ of inputs $\vec{I}^\mu$ (i.e. the image patches shown to the neuron) and outputs S^μ (i.e., the firing rates). The receptive field must be expressed by a parameterized function, the simplest is $g(\vec{I} : \vec{w}) = \sigma(\vec{w} \cdot \vec{I})$, where $\sigma(.)$ is a sigmoid function.
- Regression specifies a cost function of the parameters $\vec{w}$ such as:

$$F(\vec{w}) = \frac{1}{|\mathcal{S}|} \sum_{\mu \in \mathcal{S}}^{N} E(S^\mu - g(I^\mu; \vec{w}))$$

  where $E(.)$ is a penalty function, e.g.,$(S^\mu - g(I^\mu; T))^2$.
- This minimization can be done by standard computer packages. It outputs an estimate of the model parameters $\vec{w}^*$ and an error measure $F(\vec{w}^*) = \frac{1}{|\mathcal{S}|} \sum_{\mu \in \mathcal{S}} E(S^\mu - g(I^\mu; \vec{w}^*))$.

Is the training set representative?

- In practice, it is unrealistic to show the neuron all possible stimuli because there are so many possible image stimuli. Hence researchers must choose a restricted set of stimuli. If neurons are linear, or a known nonlinear function of a linear filter, then this should not matter because the superposition principle enables us to estimate the receptive field from a limited number of stimuli. But linearity is, at best, an approximation.
- In practice, the choice of stimuli can matter considerably. If the stimulus set does not contain the types of stimuli that the neuron is most sensitive to, then regression can output unreliable estimates for novel data. This is a problem for any learning based method (including deep networks). The function that is estimated by regression may perform well for stimuli which are similar to those in the training set, but perform very badly for novel stimuli.

Example: Talebi and Baker

- Talebi & Baker (2012) used regression to estimate the receptive fields of neurons. They used three different stimulus training sets: (1) white noise (WN), (2) oriented bars (B), and (3) natural images (NI). This gives three estimates for the receptive fields $\vec{w}_{WN}, \vec{w}_B, \vec{w}_{NI}$ by using stimulus sets $\mathcal{S}_{WN}, \mathcal{S}_B, \mathcal{S}_{NI}$.
- For each training data set, they computed the prediction errors F_{WN}, F_B, F_{NI} which are the errors for that data set, e.g., $F_{WN}(\vec{w}^*_{WN}) = \frac{1}{|\mathcal{S}_{WN}|} \sum_{\mu \in \mathcal{S}_{WN}} E(S^\mu - g(I^\mu; \vec{w}^*_{WN}))$. These prediction errors were small and showed the models fit each stimulus training set well.
- But they also studied how well the estimated receptive field from one stimulus predicted the other data sets. They computed the prediction errors $F_{WN}(\vec{w}^*_B)$, $F_{WN}(\vec{w}^*_{NI})$, $F_B(\vec{w}^*_{WN})$, $F_{WN}(\vec{w}^*_{NI})$, $F_{NI}(\vec{w}^*_{WN})$, $F_{WN}(\vec{w}^*_B)$. These prediction errors were large and showed that models did not *generalize* between the data sets. The best predictions were made by the natural image stimulus set.

Regression and Non-Linear Models

- Regression is a very general technique which dates back to Legendre (1805) and Gauss (1809) (Gauss claimed he had used it since 1795 but did not publish because it was "trivial"). The classic version was to fit data points to straight lines.
- More generally, regression can be used to fit a conditional probabilistic model $P(S|I;\vec{w})$ to data $\{(S^n, I^n) : n = 1, ..., N\}$. $P(S|I;\vec{w})$ is the probability of the output S conditioned on the input I, and $\vec{w}$ denote parameters of the model.
- The parameters $\vec{w}$ are estimated by minimizing $-\sum_{n=1}^{N} \log P(S^n|I^n;\vec{w})$ with respect to $\vec{w}$. We can recover fitting data points to straight lines, $y = ax + b$, by setting $S = y, I = x$ and $P(S|I,\vec{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\{-(y - ax - b)^2/(2\sigma^2)\}$, where $\vec{w} = (a, b, \sigma)$.

Regression for Artificial Neurons

- We can specify an artificial neuron by $P(S|\vec{I};\vec{w},w_0)=\frac{\exp\{S(\vec{w}\cdot\vec{I}+w_0)\}}{1+\exp(\vec{w}\cdot\vec{I})+w_0}$. $S=1$ means the neuron fires and $S=0$ means it does not. $\vec{w},w_0$ are the synaptic strengths.
- The expected probability of firing $\sum_{S=0,1} SP(S|\vec{I};\vec{w},w_0)=\sigma(\vec{w}\cdot I+w_0)$. Hence the expected firing rate is the standard linear model with a sigmoid non-linearity.
- Given a training set $\{(S^n,I^n):n=1,...,N\}$ we can estimate $\vec{w},w_0$ by minimizing $-\sum_{n=1}^{N}\log P(S^n|\vec{I}^n;\vec{w},w_0)$. This is essentially what Talebi and Baker did (except they used a slightly different loss function $E(\vec{w},w_0)$.

Regression and Non-Linear Models

- Regression can be used for any type of model $P(S|I;\vec{w})$. No need to use it for artificial neuron (perceptron). Particularly because they do not fit the data very well (e.g., predictions trained on some data sets do not generalize to other datasets). The artificial neuron model is known to be a poor approximation to neurons.
- A natural alternative is to use a multi-layer perceptron which consists of artificial neurons stacked on top of each other. For example $P(S|\vec{I};\vec{w},w_0,\{\vec{w}^m,w_0^m\}) = \frac{\exp\{S(\vec{w}\cdot\vec{z}+w_0)\}}{1+\exp(\vec{w}\cdot\vec{z})+w_0}$, where $\vec{z}=(z_1,...,z_M)$ with $z_m = \sigma(\vec{w}^m\cdot\vec{I}+w_0^m)$ for $m=1,...,M$.
- This multi-layer perceptron has M neurons in the first later with activities $z_1,...,z_m$ (and weights $(\vec{w}^m,w_0^m)$) and a final output neuron with weights $(\vec{w},w_0)$. The weights can be estimated (learned) by performing regression to minimize $-\sum_{n=1}^{N}\log P(S^n|\vec{I}^n;\vec{w},w_0,\{\vec{w}^m,w_0^m\})$. This minimization can be done by the backpropagation algorithm (see later lecture). Mel and his collaborators showed that this predicts neural responses better than artificial neurons *provided the dataset is big enough* (this is possible for *in vitro* neurons, removed from the brain).

Regression for Multi-Layer Perceptrons and CNNs

- We can consider other type of nonlinear models. For example, we can introduce other types of non-linearities such as max-pooling. $P(S|\vec{I};\{\vec{w}^m, w_0^m\}) = \frac{\exp\{S \max_{m=1}^{M} z_m)\}}{1+\exp\{\max_{m=1}^{M} z_m)\}}$, with $z_m = \sigma(\vec{w}^m \cdot \vec{I} + w_0^m)$ for $m = 1, ..., M$.
- We can do this for convolutional neural networks. These are a variant of multilayer perceptrons that have several convolutional layers followed by fully connected layers. The convolutional layers contain filterbanks (meany different neurons) which are copied so they are the neural weights are the same at each spatial position. They also have max-pooling in subregions of the image. They can also have other types of nonlinearities (e.g., weight normalization).
- The convolutional neural networks also have several fully connected layers following the convolutional layers (except ReNet). In these fully connected layers, spatial information is lost. They end in a "decision layer" which outputs probabilities. If the convolutional network is trained to perform object classification, then the output is a probability distribution over a binary-valued vector $\vec{S} + (S_1, ..., S_L)$, where $S_i = 1$ and $S_j = 0 \; \forall j \neq i$ means that the image is classified as object i (e,g., as a cat). Hence deep networks specify a probability distribution $P(\vec{S}|\vec{I};\{\vec{w}, w_0)\}$, where $\{\vec{w}, w_0)\}$ denote the weights of the neurons at all layers. They can be trained/letnt ny regression (using backpropagation) if there is a sufficiently large training dataset $\{(\vec{S}^n, \vec{I}^n) : n = 1, ..., N\}$.

Mathematics of Convolutional Neural Networks

- CNNs have a set of receptive fields at each position $z(y)^m = f(\sum_x w^m(y-x)I(x) + w_0^m)$ where $m = 1, ..., M$ specifies a filterbank (i.e., the same at each spatial position y, hence *convolutional*). $f(.)$ is a function (sigmoid $\sigma(.)$ or ReLu $\max(., 0)$). The filters $w^a(x-y)$ are only non-zero for small $|x-y|$ (i.e. they have finite receptive field size). They also have max-pooling, so the spatial input to the next level at position y is $\max z(y')^m : y' \in Nbh(y)$, where $Nbh(y)$ is a small spatial neighbourhood of y.
- Convolutional networks have several layers of these "convolutional forms" followed by a few "fully-connected" layers where the neurons z^m receive input for all "neurons" at the previous layer – e.g., $z^m = f(\sum_{a,y} w_a(y)z(y)^a)$. The final layer can be expressed as $P(S|z^m; \vec{w}) = \frac{\exp\{-\sum_i S_i \sum_m w_{im} z^m\}}{\sum_{i=1}^L \exp\{-\sum_m w_{im} z^m\}}$. Here $\{S_i : i = 1, ..., L\}$ denotes the identify of the object.

CNNs and Estimating Receptive Field of Neurons

- Nobody (yet) has enough training data to learn the receptive fields of real neurons from training data (where the training data specifies if the real neuron fires to an input image patch). The CNN has too many parameters. It is impractical even for simple multi-layer perceptrons unless the neuron is *in vitro* (meaning that you can stimulate the neuron at all synapses systematically by electric shocks).
- Instead you can train a deep network on a visual task light object recognition using a huge dataset like ImageMet (millions of images). This gives you a hierarchical set of receptive fields (for neurons at different layers of the network). The receptive fields at the lowest layer can be directly visualized and look like Gabor functions (as one would expect). The higher layer receptive fields are non-linear and much harder to interpret (see later in the course).
- Then we can compare the predictions of the neurons in CNN to the activity of the real neuron. Either we can try to find an artificial neurons whose prediction matches the activity of the real neuron. Or we can train a simple regression model where the input is a subset of artificial neurons from the CNN.
- This is used to model receptive fields of high-level neurons in IT (e.g., DiCarlo, Yasmins).When applied to V1 it shows that the real neurons are typically much better modeled by neurons which are *not* at the lowest levels and hance are nonlinear (TS Lee).