Predictive Coding

Note on continuous-time optimization

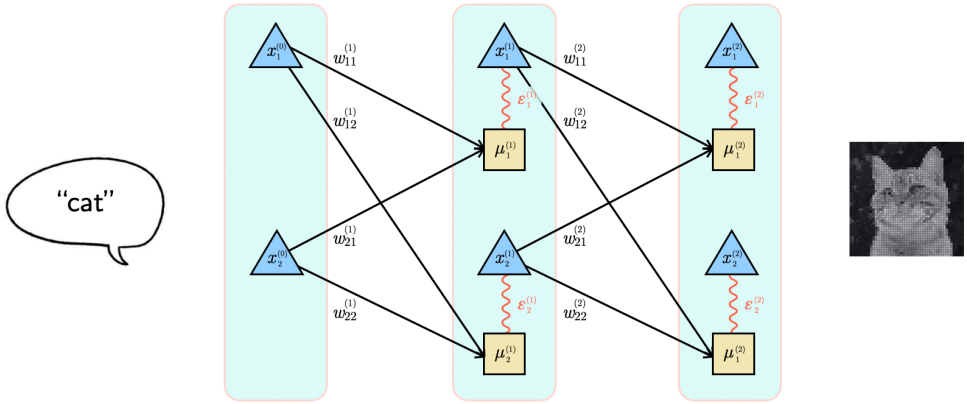
Instead of discrete update steps, we define the time-derivative of a parameter, which allows for continuous change. The common update rule:

$$w \leftarrow w - \alpha \cdot \frac{\partial J(w)}{\partial w}$$

corresponds to the following time-derivative:

$$\frac{dw(t)}{dt} = -\alpha \cdot \frac{\partial J(w)}{\partial w}$$

1. Generative predictive coding



1.1 Linear

Activity, prediction and error

Each neuron has some *activity*, x, which—together with all the other neurons in its layer—determines a *prediction* of each neuron's activity, μ, in the next layer, weighted by *synaptic weights*, w.

$$\mu_i^{(l)} = \sum_k w_{ik}^{(l)} \cdot x_k^{(l-1)}$$

$$\boxed{\boldsymbol{\mu}^{(l)} = \boldsymbol{W}^{(l)}\, \boldsymbol{x}^{(l-1)}}$$

This is a simplification, as we would usually also apply an activation function to the neurons' activities: $f(x_k^{(l-1)})$. We will look at this later in the section on *nonlinear predictive coding*.

The *error* (prediction error) for a particular activity is simply the difference between the activity and the prediction:

$$\begin{aligned}\varepsilon_i^{(l)} &= x_i^{(l)} - \sum_k w_{ik}^{(l)} \cdot x_k^{(l-1)} \\ &= x_i^{(l)} - \mu_i^{(l)}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\varepsilon}^{(l)} &= \boldsymbol{x}^{(l)} - \boldsymbol{W}^{(l)}\, \boldsymbol{x}^{(l-1)} \\ &= \boldsymbol{x}^{(l)} - \boldsymbol{\mu}^{(l)}\end{aligned}$$

$$\boxed{\boldsymbol{\varepsilon}^{(l)} = \boldsymbol{x}^{(l)} - \boldsymbol{W}^{(l)}\, \boldsymbol{x}^{(l-1)}}$$

Energy

One possible way of defining the *energy* of the network is as the sum of the square of all prediction errors (times 1/2 for later convenience). Which energy function to use depends on the use case.

$$\begin{aligned}E &= \frac{1}{2} \sum_\ell^L \sum_i^{N_\ell} (\varepsilon_i^{(\ell)})^2 \\ &= \frac{1}{2} \sum_\ell^L \|\boldsymbol{\varepsilon}^{(\ell)}\|^2\end{aligned}$$

This is the quantity we wish to minimize.

Inference: updating the activity

The goal is to minimize the network's energy by adjusting the neuronal activities, x, and the synaptic weights, w. How do all the x and w have to change to minimize the energy? We move in the negative direction of the energy gradient with respect to x or w, respectively (using appropriate learning rates α, β).

$$\begin{aligned}\frac{dx_i^{(l)}}{dt} &= -\alpha \cdot \frac{\partial E}{\partial x_i^{(l)}} \\ &= -\alpha \cdot \frac{\partial}{\partial x_i^{(l)}} \left[\sum_\ell^L \sum_k^{N_\ell} \frac{1}{2} (\varepsilon_k^{(\ell)})^2 \right]\end{aligned}$$

We can also think of $x_i^{(l)}$ as being a function of time, $x_i^{(l)}(t)$, but since we don't really care about the value of x at any particular point in time, I will leave it out for simplicity. Notice that $x_i^{(l)}$ only affects (I) its own error (layer l), $\varepsilon_i^{(l)}$, and (II) the next layer's $(l+1)$ predictions, $\mu_j^{(l+1)}$ (all j). All other terms of the energy function are constant w.r.t. $x_i^{(l)}$ and therefore 0 in the derivative. Let's tackle these two effects separately and add them together afterwards.

$$
\begin{aligned}
\text{(I)} &= -\frac{\partial}{\partial x_i^{(l)}}\left[\frac{1}{2}(x_i^{(l)} - \mu_i^{(l)})^2\right] \\
&= -(x_i^{(l)} - \mu_i^{(l)}) \\
&= -\varepsilon_i^{(l)}
\end{aligned}
$$

$$
\begin{aligned}
\text{(II)} &= -\frac{\partial}{\partial x_i^{(l)}}\left[\frac{1}{2}\sum_j (x_j^{(l+1)} - \mu_j^{(l+1)})^2\right] \\
&= -\frac{1}{2}\sum_j \frac{\partial}{\partial x_i^{(l)}}(x_j^{(l+1)} - \mu_j^{(l+1)})^2
\end{aligned}
$$

We can plug in the definition of the prediction μ:

$$
= -\frac{1}{2}\sum_j \frac{\partial}{\partial x_i^{(l)}}\left[x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot x_k^{(l)}\right]^2
$$

Notice that k is the index of neurons in layer l, and i is one of these k.

Then, we can use the chain rule and get an outer and inner derivative.

In the inner derivative, only the term that includes $w_{ji}^{(l+1)}$ is non-zero.

$$
= -\frac{1}{2}\sum_j 2 \cdot (x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot x_k^{(l)}) \cdot (-w_{ji}^{(l+1)})
$$

This nicely simplifies to:

$$
= \sum_j (x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot x_k^{(l)}) \cdot (w_{ji}^{(l+1)})
$$

$$
= \sum_j (x_j^{(l+1)} - \mu_j^{(l+1)}) \cdot (w_{ji}^{(l+1)})
$$

The first term in the sum is simply the error, so we can simplify again:

$$
= \sum_j \varepsilon_j^{(l+1)} \cdot w_{ji}^{(l+1)}
$$

Putting both of these together, we get:

$$\boxed{\frac{\partial x_i^{(l)}}{\partial t} = \alpha \cdot \left[-\varepsilon_i^{(l)} + \sum_j \varepsilon_j^{(l+1)} \cdot w_{ji}^{(l+1)} \right]}$$

And in vector notation for the neuron activities of the entire layer:

$$\boxed{\frac{\partial \boldsymbol{x}^{(l)}}{\partial t} = \alpha \cdot \left[-\boldsymbol{\varepsilon}^{(l)} + \boldsymbol{W}^{(l+1)} \boldsymbol{\varepsilon}^{(l+1)} \right]}$$

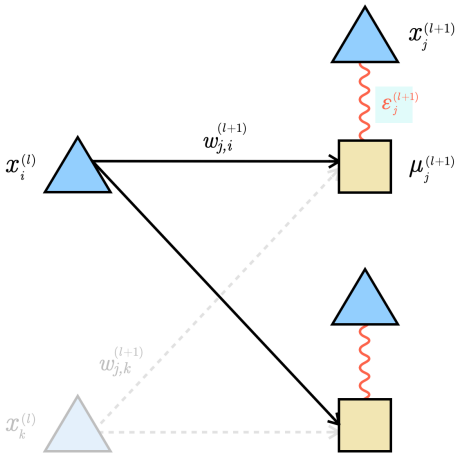


Figure 1: One layer of a predictive coding network, with one neuron (neuron i in layer l) highlighted. Triangles correspond to representation neurons, whereas rectangles correspond to predictions. Alternatively, instead of the predictions, this can be visualized using error neurons. On the left side of the network (bottom of the hierarchy, layer 0) would be the latent vector, and on the right side (top of the hierarchy, layer $L-1$) would be the (sensory) data that the network attempts to explain.

Learning: updating the weights

But not only x changes over time to minimize the network's energy (inference). The synaptic weights, w, also change (learning):

$$\frac{dw_{ji}^{(l)}}{dt} = -\beta \cdot \frac{\partial E}{\partial w_{ji}^{(l)}}$$

Indices k (and i) still refer to neurons in the *current layer*, but I will index this layer as $l-1$, while j still refers to neurons in the *next layer*, but I will index this layer as l.

And the negative energy derivative looks like this:

$$-\frac{\partial E}{\partial w_{ji}^{(l)}} = -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\sum_{\ell}^{L}\sum_{k}^{N_\ell}\frac{1}{2}(\varepsilon_k^{(\ell)})^2\right]$$

Notice that ℓ is the running variable, and l is one of these layers.

In fact, layer l is the only layer ℓ that depends on $w_{ji}^{(l)}$:

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(\varepsilon_j^{(l)})^2\right]$$

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(x_j^{(l)} - \mu_j^{(l)})^2\right]$$

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(x_i^{(l)} - \sum_k w_{jk}^{(l)} \cdot x_k^{(l-1)})^2\right]$$

$$= -\frac{1}{2} \cdot 2 \cdot (x_j^{(l)} - \sum_k w_{jk}^{(l)} \cdot x_k^{(l-1)}) \cdot (-x_i^{(l-1)})$$

$$= (x_j^{(l)} - \sum_k w_{jk}^{(l)} \cdot x_k^{(l-1)}) \cdot (x_i^{(l-1)})$$

$$= (x_j^{(l)} - \mu_j^{(l)}) \cdot (x_i^{(l-1)})$$

$$= \varepsilon_j^{(l)} \cdot x_i^{(l-1)}$$

Therefore, we get:

$$\boxed{\frac{\partial w_{ji}^{(l)}}{\partial t} = \beta \cdot \varepsilon_j^{(l)} \cdot x_i^{(l-1)}}$$

And in vector notation:

$$\boxed{\frac{\partial \boldsymbol{W}^{(l)}}{\partial t} = \beta \cdot \boldsymbol{\varepsilon}^{(l)} \, (\boldsymbol{x}^{(l-1)})^{\top}}$$

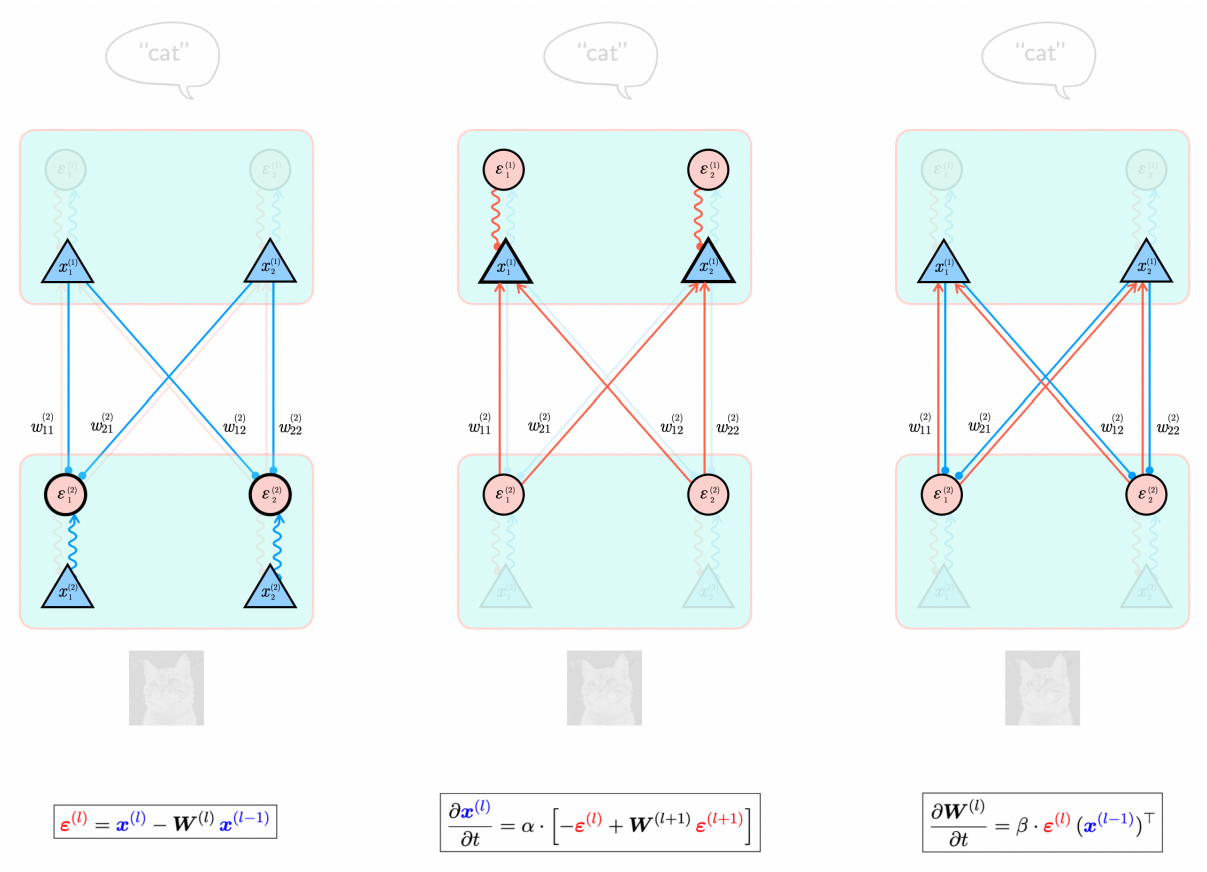


Figure 2: (Left) Computing the errors. (Middle) Updating the neuron activities. (Right) Updating the weights.

1.2 Nonlinear predictive coding

In nonlinear predictive coding, we simply apply an elementwise nonlinear activation function f to $\boldsymbol{x}$ when computing the prediction $\boldsymbol{\mu}$.

$$\boldsymbol{\mu}^{(l)} = \boldsymbol{W}^{(l)} f(\boldsymbol{x}^{(l-1)})$$

Inference: updating the activity

In order to derive the time-derivative of $\boldsymbol{x}$, we use the same approach as before, but use the new formula for the prediction.

$$\frac{dx_i^{(l)}}{dt} = -\alpha \cdot \frac{\partial E}{\partial x_i^{(l)}}$$

$$= -\alpha \cdot \frac{\partial}{\partial x_i^{(l)}} \left[\sum_{\ell}^{L} \sum_{k}^{N_\ell} \frac{1}{2} (\varepsilon_k^{(\ell)})^2 \right]$$

The case distinction between (I) and (II) is still valid, as the same neurons are affected as before; they are simply affected differently. In fact, (I) is exactly the same, as it doesn't depend on $\boldsymbol{\mu}$ (only as a constant w.r.t. $\boldsymbol{x}$).

$$\text{(I)} = -\frac{\partial}{\partial x_i^{(l)}} \left[\frac{1}{2} (x_i^{(l)} - \mu_i^{(l)})^2 \right]$$

$$= -(x_i^{(l)} - \mu_i^{(l)})$$

$$= -\varepsilon_i^{(l)}$$

$$
\begin{aligned}
\text{(II)} &= -\frac{\partial}{\partial x_i^{(l)}} \left[\frac{1}{2} \sum_j (x_j^{(l+1)} - \mu_j^{(l+1)})^2 \right] \\
&= -\frac{1}{2} \sum_j \frac{\partial}{\partial x_i^{(l)}} (x_j^{(l+1)} - \mu_j^{(l+1)})^2
\end{aligned}
$$

In the following we now see that μ is nonlinear in x:

$$
\begin{aligned}
&= -\frac{1}{2} \sum_j \frac{\partial}{\partial x_i^{(l)}} \left[x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot f(x_k^{(l)}) \right]^2 \\
&= -\frac{1}{2} \sum_j 2 \cdot (x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot f(x_k^{(l)})) \cdot (-w_{ji}^{(l+1)} \cdot f'(x_i^{(l)})) \\
&= \sum_j (x_j^{(l+1)} - \sum_k w_{jk}^{(l+1)} \cdot f(x_k^{(l)})) \cdot w_{ji}^{(l+1)} \cdot f'(x_i^{(l)}) \\
&= \sum_j (x_j^{(l+1)} - \mu_j^{(l+1)}) \cdot w_{ji}^{(l+1)} \cdot f'(x_i^{(l)}) \\
&= \sum_j \varepsilon_j^{(l+1)} \cdot w_{ji}^{(l+1)} \cdot f'(x_i^{(l)})
\end{aligned}
$$

Putting both of these together, we get:

$$
\boxed{\frac{\partial x_i^{(l)}}{\partial t} = \alpha \cdot \left[-\varepsilon_i^{(l)} + \sum_j \varepsilon_j^{(l+1)} \cdot w_{ji}^{(l+1)} \cdot f'(x_i^{(l)}) \right]}
$$

And in vector notation for the neuron activities of the entire layer:

$$
\boxed{\frac{\partial \boldsymbol{x}^{(l)}}{\partial t} = \alpha \cdot \left[-\boldsymbol{\varepsilon}^{(l)} + \boldsymbol{W}^{(l+1)} \left(\boldsymbol{\varepsilon}^{(l+1)} \odot f'(\boldsymbol{x}^{(l)}) \right) \right]}
$$

Learning: updating the weights

$$-\frac{\partial E}{\partial w_{ji}^{(l)}} = -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\sum_{\ell}^{L}\sum_{k}^{N_\ell}\frac{1}{2}(\varepsilon_k^{(\ell)})^2\right]$$

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(\varepsilon_j^{(l)})^2\right]$$

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(x_j^{(l)} - \mu_j^{(l)})^2\right]$$

$$= -\frac{\partial}{\partial w_{ji}^{(l)}}\left[\frac{1}{2}(x_i^{(l)} - \sum_k w_{jk}^{(l)} \cdot f(x_k^{(l-1)}))^2\right]$$

Notice that $f(x_i^{(l-1)})$ is constant w.r.t. $w_{ji}^{(l)}$.

So the inner derivative is simply $-f(x_i^{(l-1)})$:

$$= -\frac{1}{2} \cdot 2 \cdot (x_j^{(l)} - \sum_k w_{jk}^{(l)} \cdot f(x_k^{(l-1)})) \cdot (-f(x_i^{(l-1)}))$$

$$= (x_j^{(l)} - \sum_k w_{jk}^{(l)} \cdot f(x_k^{(l-1)})) \cdot f(x_i^{(l-1)})$$

$$= (x_j^{(l)} - \mu_j^{(l)}) \cdot f(x_i^{(l-1)})$$

$$= \varepsilon_j^{(l)} \cdot f(x_i^{(l-1)})$$

Therefore, we get:

$$\boxed{\frac{\partial w_{ji}^{(l)}}{\partial t} = \beta \cdot \varepsilon_j^{(l)} \cdot f(x_i^{(l-1)})}$$

And in vector notation:

$$\boxed{\frac{\partial \boldsymbol{W}^{(l)}}{\partial t} = \beta \cdot \boldsymbol{\varepsilon}^{(l)}\, f(\boldsymbol{x}^{(l-1)})^\top}$$

Algorithm 1 : Inference Learning in Predictive Coding Networks

Require: `model`, x, $y = \text{None}$, $\alpha = 0.1$, $\beta = 0.01$, $T = 20$

for step $t \leftarrow 1, \ldots, T$ **do**
  $\varepsilon \leftarrow$ `model.compute_errors`(h, x, y)
  for layer $\ell \leftarrow 1, \ldots, L-1$ **do**
    $h^{(\ell)} \leftarrow h^{(\ell)} + \alpha \cdot$ `model.compute_activity_update`(ℓ, h, ε)
for layer $\ell \leftarrow 1, \ldots, L-1$ **do**
  $W^{(\ell)} \leftarrow W^{(\ell)} + \beta \cdot$ `model.compute_weight_update`(ℓ, h, ε)
return model

Algorithm 2 : Inference Learning in Predictive Coding Networks (unsupervised)

Require: `model`, x, $\alpha = 0.1$, $\beta = 0.01$, $T = 20$

$\boldsymbol{h}^{(0)} \leftarrow \boldsymbol{x}$
$\boldsymbol{\varepsilon}^{(L)} \leftarrow \mathbf{0}$
for step $t \leftarrow 1, \ldots, T$ **do**
  for layer $\ell \leftarrow 1, \ldots, L-1$ **do**
    $\boxed{\boldsymbol{\varepsilon}^{(\ell)} \leftarrow \boldsymbol{h}^{(\ell)} - \boldsymbol{W}^{(\ell)}\, \boldsymbol{h}^{(\ell-1)}}$
  for layer $\ell \leftarrow 1, \ldots, L-1$ **do**
    $\boxed{\boldsymbol{h}^{(\ell)} \leftarrow \boldsymbol{h}^{(\ell)} + \alpha \cdot \left[-\boldsymbol{\varepsilon}^{(\ell)} + \boldsymbol{W}^{(\ell+1)}\, \boldsymbol{\varepsilon}^{(\ell+1)}\right]}$
for layer $\ell \leftarrow 1, \ldots, L-1$ **do**
  $\boxed{\boldsymbol{W}^{(\ell)} \leftarrow \boldsymbol{W}^{(\ell)} + \beta \cdot \left[\boldsymbol{\varepsilon}^{(\ell)}\, (\boldsymbol{x}^{(\ell-1)})^{\top}\right]}$
return model

Some notes on this: This is the pseudocode that can be used with or without labels y. If labels y are provided, then during optimization the outputs are clamped to those values, forcing the activities and weights to minimize energy under this constraint.