Transformed data

Lecture 20

Consider a transform with a single parameter: $\vec{s}(\vec{x}, \xi)$ $[\ \vec{s}(\vec{x}, 0) = \vec{x}\ ]$

In the infinite data limit, the sum-of-squares error function is given by:

$$E = \frac{1}{2} \int\int d\vec{x}\, dt\, (y(\vec{x}) - t)^2\, p(t|\vec{x})\, p(\vec{x})$$

$y(\vec{x})$: 1D output (single output node) for simplicity

Imagine that each $\vec{x}$ is perturbed many times: $\vec{x} \to \vec{s}(\vec{x}, \xi)$, where ξ is drawn from $p(\xi)$.

Then

$$\tilde{E} = \frac{1}{2} \int\int\int d\vec{x}\, dt\, d\xi\, (y(\vec{s}(\vec{x}, \xi)) - t)^2\, p(t|\vec{x})\, p(\vec{x})\, p(\xi)$$

$\tilde{E}$: over expanded data set

Now, assume that $\int d\xi\, \xi\, p(\xi) = E(\xi) = 0$,

$E(\xi^2) = \int d\xi\, \xi^2 p(\xi) = \lambda \Leftarrow$ small, variance

s.t. we only consider "small" transformations of $\vec{x}$.

Then

$$\vec{S}(\vec{x},\xi) = \vec{S}(\vec{x},0) + \xi \underbrace{\frac{\partial}{\partial \xi} S(\vec{x},\xi)\Big|_{\xi=0}}_{\vec{\tau}} +$$

$$+ \frac{\xi^2}{2} \underbrace{\frac{\partial^2}{\partial \xi^2} \vec{S}(\vec{x},\xi)\Big|_{\xi=0}}_{\equiv \vec{\tau}'} + \Theta(\xi^3)$$

Next,

$$\vec{y}(\vec{S}(\vec{x},\xi)) \approx \vec{y}\left(\vec{x} + \xi\vec{\tau} + \frac{\xi^2}{2}\vec{\tau}'\right) \approx$$

$$\approx \vec{y}(\vec{x}) + \xi \tau_i \frac{\partial \vec{y}(\vec{x})}{\partial x_i} + \frac{\xi^2}{2}\tau_i' \frac{\partial y(\vec{x})}{\partial x_i} +$$

$$+ \frac{\xi^2}{2}\tau_i\tau_j \frac{\partial^2 y(\vec{x})}{\partial x_i \partial x_j} \frac{\partial y(\vec{x})}{\partial x_i}$$

sums over i,j implied

Then

$$\bar{E} \approx \frac{1}{2}\iiint d\vec{x}\,dt\,d\xi\; p(t|\vec{x})\,p(\vec{x})\,p(\xi) \times$$

$$\times \left[y(\vec{x}) + \xi\tau_i \frac{\partial y}{\partial x_i} + \frac{\xi^2}{2}\tau_i' \frac{\partial y}{\partial x_i} + \frac{\xi^2}{2}\tau_i\tau_j \frac{\partial^2 y}{\partial x_i \partial x_j} - t \right]^2 \approx$$

($\int d\xi\, p(\xi) = 1$)

$$\approx \underbrace{\frac{1}{2}\iint d\vec{x}\,dt\,[y(\vec{x})-t]^2\, p(t|\vec{x})\,p(\vec{x})}_{E} \quad (+)$$

$$\oplus \underbrace{E(\xi)}_{=0}\tfrac{1}{2}\int\!\!\int d\vec{x}\,dt \;\ldots\; + \underbrace{E(\xi^2)}_{\lambda}\tfrac{1}{2}\int d\vec{x}\,dt\; p(t|\vec{x})\,p(\vec{x}) \times$$

$$\times\Big[(y(\vec{x})-t)\Big[\tau_i'\frac{\partial y}{\partial x_i}+\tau_i\tau_j\frac{\partial^2 y}{\partial x_i\,\partial x_j}\Big]+$$

$$+\tau_i\frac{\partial y}{\partial x_i}\tau_j\frac{\partial y}{\partial x_j}\Big]\,.$$

So, $\tilde{E} = E + \lambda\Omega$, where

$$\Omega = \tfrac{1}{2}\int d\vec{x}\; p(\vec{x})\Big[\,(y(\vec{x}) - \underbrace{E[t|\vec{x}]}_{\int dt\, p(t|\vec{x})\,t})\,[\ldots] +$$

$$+\tau_i\frac{\partial y}{\partial x_i}\tau_j\frac{\partial y}{\partial x_j}\Big]\,.$$

$\uparrow \int dt\, p(t|\vec{x}) = 1$

Now, note that $y(\vec{x}) = E[t|\vec{x}]$ minimizes E and "almost" minimizes $\tilde{E}$: $y(\vec{x}) = E[t|\vec{x}] + \Theta(\xi)$ minimizes $\tilde{E}$ in fact

Thus, the 1st term in Ω is $\Theta(\xi)$ while the 2nd is $\Theta(\xi^0)$, so that

$$\Omega \simeq \frac{1}{2}\int d\vec{x}\, p(\vec{x})\Big(\tau_i\underbrace{\frac{\partial y}{\partial x_i}}_{\text{1D Jacobian}}\Big)^2 \Leftarrow \text{same as before!}$$

Finally, in a special case

$\vec{x} \to \vec{x} + \vec{\xi}$ we obtain:

$\vec{\xi}$ ↑ random translations : $p(\vec{\xi}) = \prod_i p(\xi_i)$

$\vec{S}(\vec{x}, \vec{\xi}) = \vec{x} + \vec{\xi} \Rightarrow \underbrace{\frac{\partial S_k}{\partial \xi_i} = \delta_{ik}}$ not really needed

Then $y(\vec{S}) \simeq y(\vec{x}) + \xi_i \nabla_i y(\vec{x}) +$
$\vec{S} = \vec{x} + \vec{\xi}$

$+ \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(\vec{x})$.

$\nabla_i \nabla_j = \frac{\partial''}{\partial x_i}$ /

Then $\bar{E} = \frac{1}{2} \iiint d\vec{x}\, dt\, d\vec{\xi}\; p(t|\vec{x})\, p(\vec{x})\, p(\vec{\xi}) \times$

$\times \left[y(\vec{x}) + \xi_i \nabla_i y(\vec{x}) + \frac{1}{2} \xi_i \xi_j \nabla_i \nabla_j y(\vec{x}) - t \right]^2 =$

$= E + \underbrace{E(\xi_i)}_{\int d\xi_i\, \xi_i\, p(\xi_i)} \frac{1}{2} \iint d\vec{x}\, dt \ldots \oplus$
$\parallel$ 0

$\oplus \frac{1}{2} E(\xi_i \xi_j) \iint d\vec{x}\, dt\; p(t|\vec{x})\, p(\vec{x}) \times$

$\times \left[(y(\vec{x}) - t) \nabla_i \nabla_j y(\vec{x}) + \nabla_i y(\vec{x})\, \nabla_j y(\vec{x}) \right] =$

$= E + \lambda \Omega$.

$$E(\xi_i \xi_j) = \begin{cases} \int\int d\xi_i d\xi_j \, \xi_i \xi_j \, p(\xi_i) p(\xi_j) = 0 \;, & i \neq j \\ \int d\xi_i \, \xi_i^2 \, p(\xi_i) \equiv \lambda \;, & i = j \end{cases}$$

Here, $\Omega = \frac{1}{2} \int d\vec{x} \, p(\vec{x}) \Big[\underbrace{(y(\vec{x}) - E[t|\vec{x}])}_{O(|\xi|), \text{ discard}} \underbrace{\vec{\nabla}^2}_{\sum_i \nabla_i^2} y(\vec{x}) +$

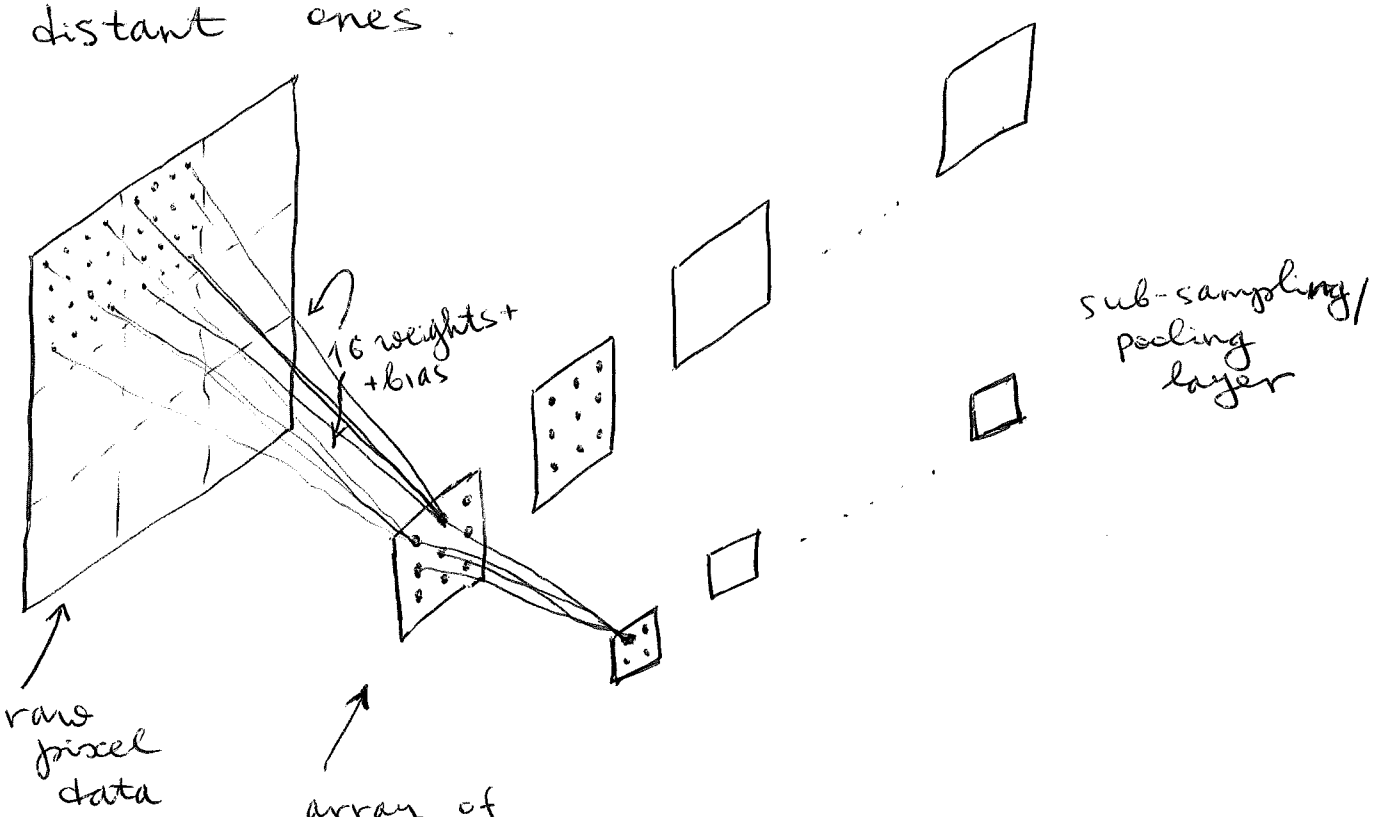
$$+ \underbrace{\nabla_i y(\vec{x}) \, \nabla_i y(\vec{x})}_{\|\vec{\nabla} y\|^2} \Big] =$$

$$= \frac{1}{2} \int d\vec{x} \, p(\vec{x}) \, \|\vec{\nabla} y(\vec{x})\|^2 \,.$$

Tikhonov regularization

Convolutional networks

Idea: avoid manual feature extraction by extracting higher-and-higher level invariant features from raw pixel data. Use the fact that nearby pixels are more strongly correlated than distant ones.



; each node in a given feature map has the same weights + bias $\Rightarrow$ acts as a kernel transform (or convolution): $h\left(\sum_j w_{kj} x_j\right)$

these weights are called a filter bank

Idea: a given feature map detects the presence of a given feature anywhere within a map $\Rightarrow$ $\Rightarrow$ weights must be shared since the feature to be detected is the same.

Each pooling node combines data from several (contiguous) nodes in a given feature map (or several feature maps), to reduce dimension of the representation and decrease sensitivity to small shifts & distortions.

Implementation: 1. $\sigma(\tilde{w}\underbrace{\langle z \rangle}_{\text{average of all inputs}} + \tilde{w}_0)$ [older]

2. $\max\{z_j\}$ max-pooling

Finally, 2-3 stages of feature extraction (convolution) & pooling are stacked, with the number of feature maps going up in a given layer as features become higher-level (but lower-dimensional).

The final two layers are a convolutional layer which is fully connected to an output layer, typically with soft-max activation functions for $K > 2$ classification.

Modern implementation:

1. Use ReLU(·) as $h(\cdot)$ in convolutional layers, where

   $\underbrace{\text{ReLU}(z) = \max(z, 0)}_{\text{rectified linear unit}}$, rather than $\sigma(z)$ or $\tanh(z)$.

   Typically learns much faster with ReLU(·).

2. Use stochastic gradient descent for training => steepest descent informed by a few datapoints at a time rather than all of them.

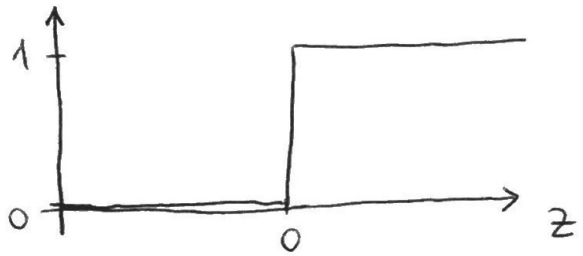
3. Use backpropagation to compute the gradients

4. In deep NNs, pretrain intermediate convolutional layers using Boltzmann machines (afterwards, refine pre-trained weights by backpropagation)

Non-linear activation functions

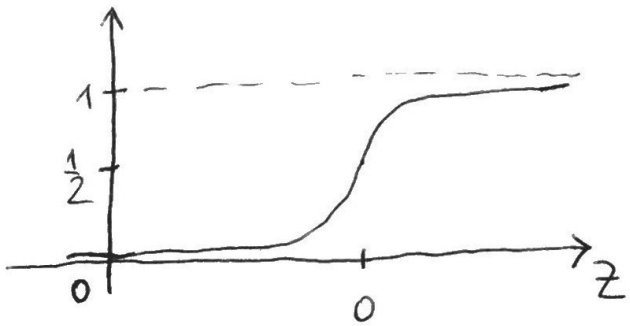
"Classical" activation functions are prone to saturation on the tails, where the gradients become small or vanish completely:

Perceptron



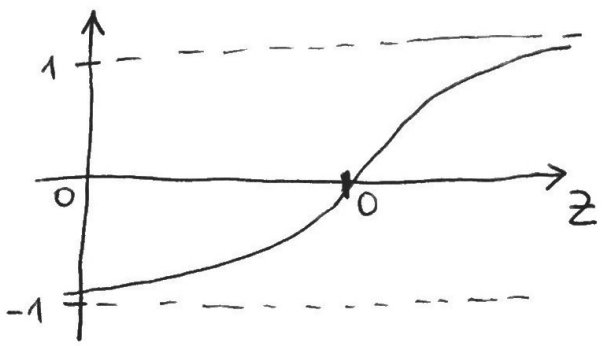
$\theta(z)$

Sigmoid



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

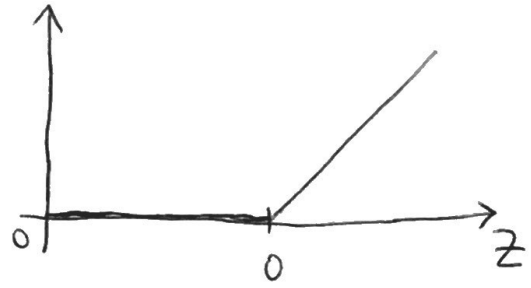
Tanh



$\tanh(z)$

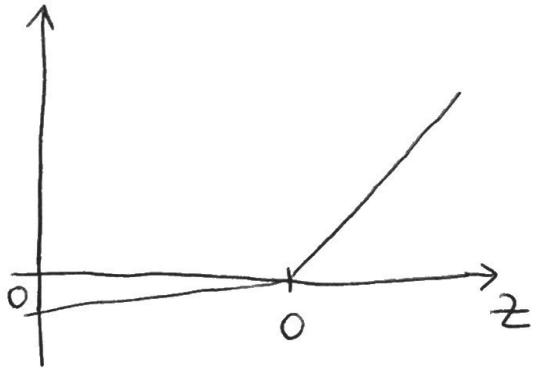
Gradient saturation is widely believed to reduce convergence and/or predictive power. Therefore, other activation functions have been proposed: (adopted)

ReLU



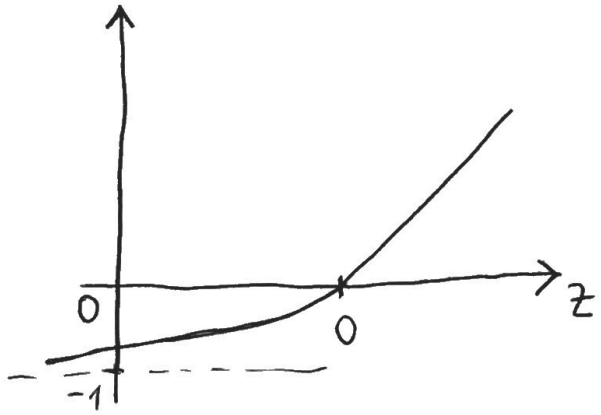
$\max(0, z)$

Leaky ReLU



$$\begin{cases} z, & z \geq 0 \\ 0.1z, & z < 0 \end{cases}$$

ELU



$$\begin{cases} z, & z \geq 0 \\ e^z - 1, & z < 0 \end{cases}$$