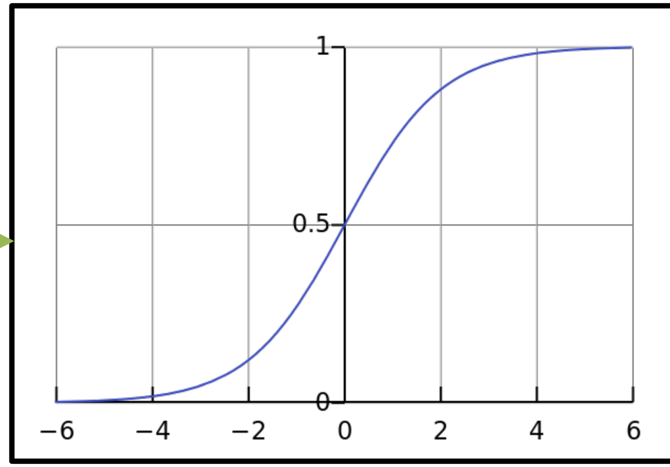
Intro

- In this video you will learn the tricks that help to train really deep networks!

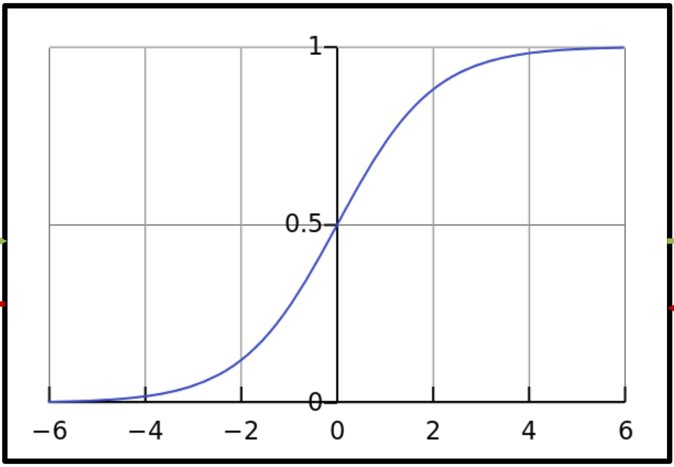
Sigmoid activation

x



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

Sigmoid activation



x

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

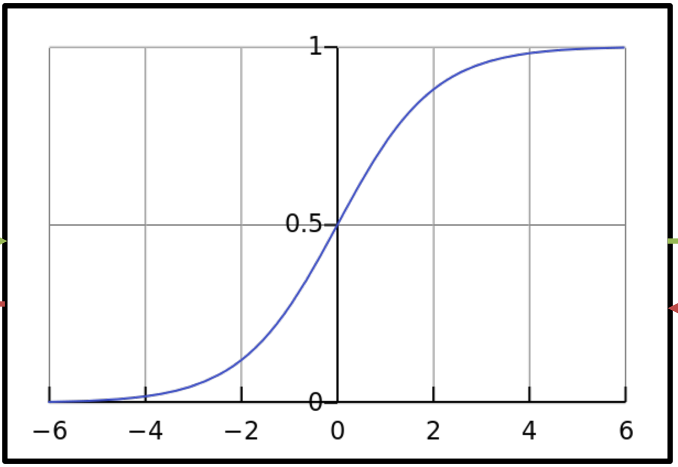
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial x}$$

$$\frac{\partial L}{\partial \sigma}$$

$$\frac{\partial \sigma}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Sigmoid activation

x



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

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial x}$$

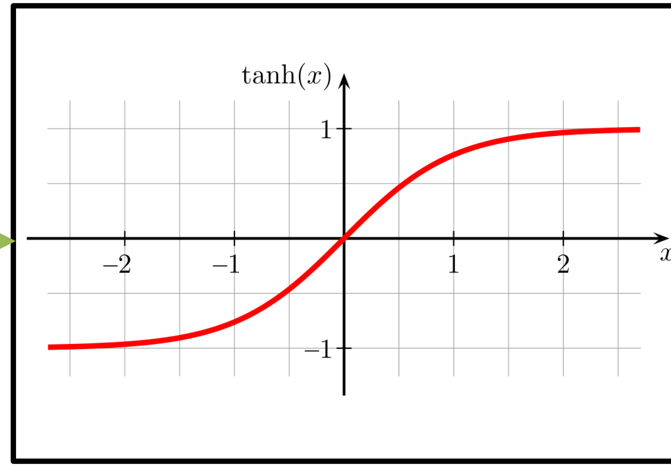
$$\frac{\partial L}{\partial \sigma}$$

$$\frac{\partial \sigma}{\partial x} = \sigma(x)(1 - \sigma(x))$$

- Sigmoid neurons can saturate and lead to **vanishing gradients**.
- Not zero-centered.
- e^x is computationally expensive.

Tanh activation

x

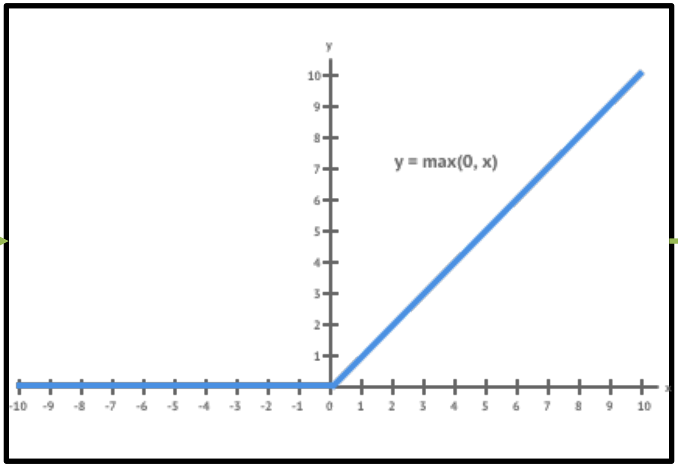


$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

- Zero-centered.
- But still pretty much like sigmoid.

ReLU activation

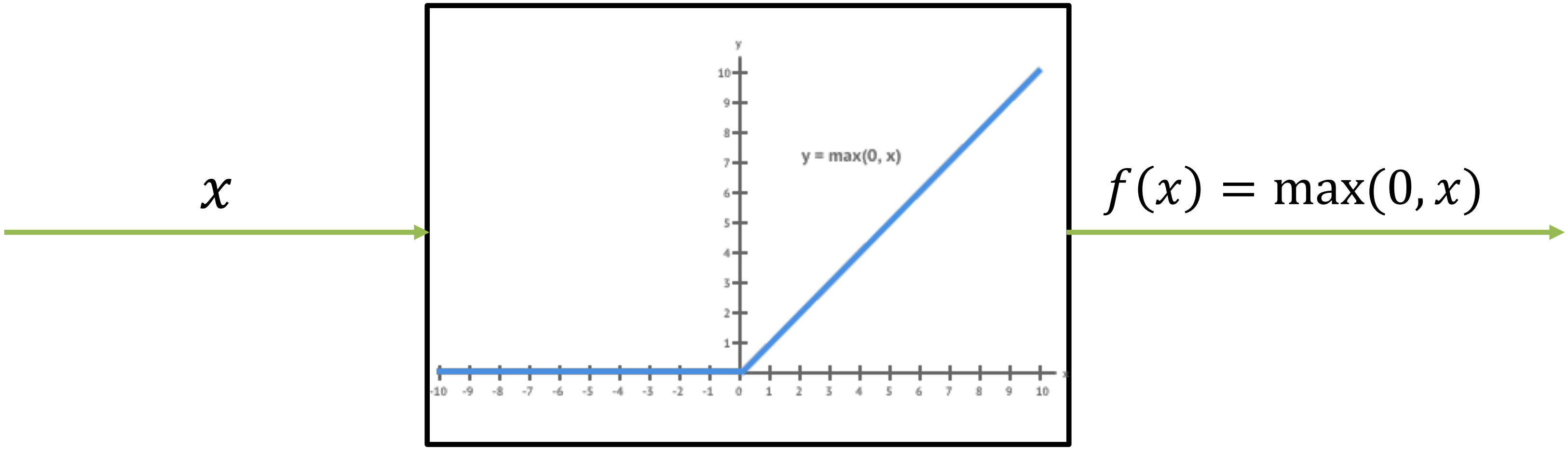
x



$f(x) = \max(0, x)$

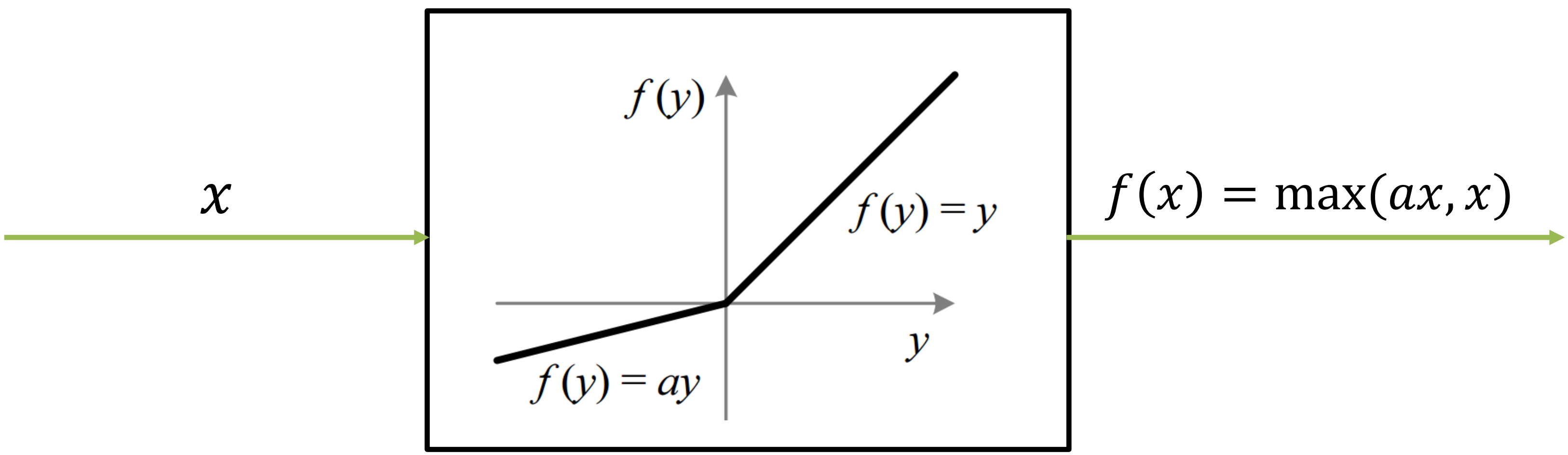
- Fast to compute.
- Gradients do not vanish for $x > 0$.
- Provides faster convergence in practice!

ReLU activation



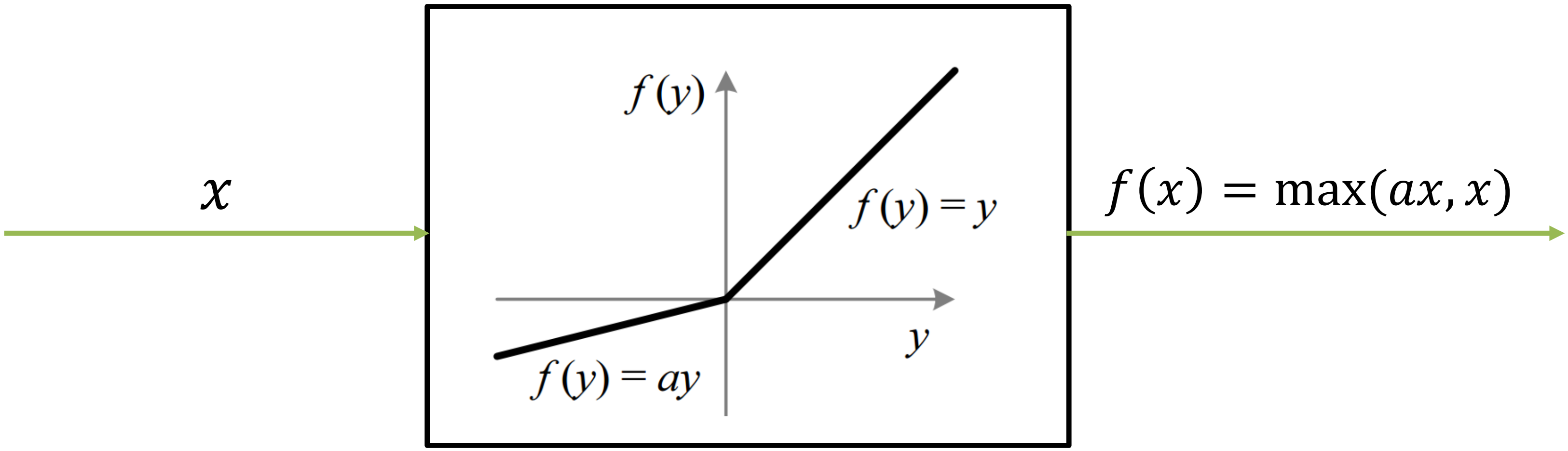
- Fast to compute.
- Gradients do not vanish for $x > 0$.
- Provides faster convergence in practice!
- Not zero-centered.
- Can die: if not activated, never updates!

Leaky ReLU activation



- Will not die!

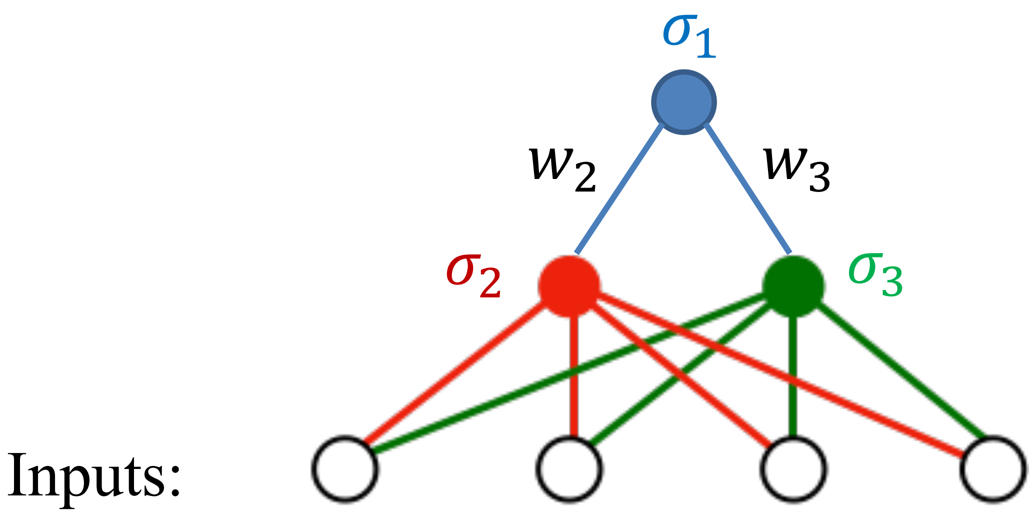
Leaky ReLU activation



- Will not die!
- $a \neq 1$

Weights initializations

Maybe start with all zeros?

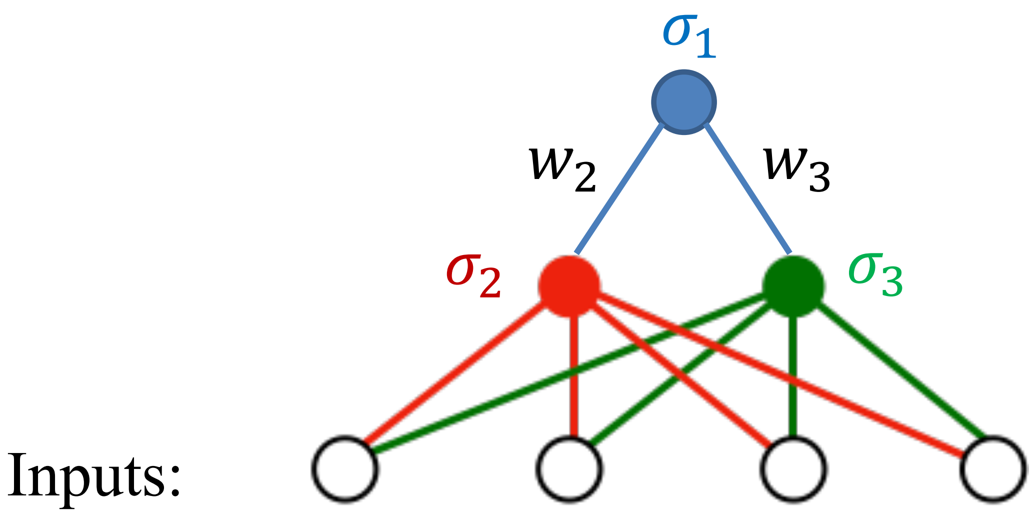


$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

Weights initializations

Maybe start with all zeros?



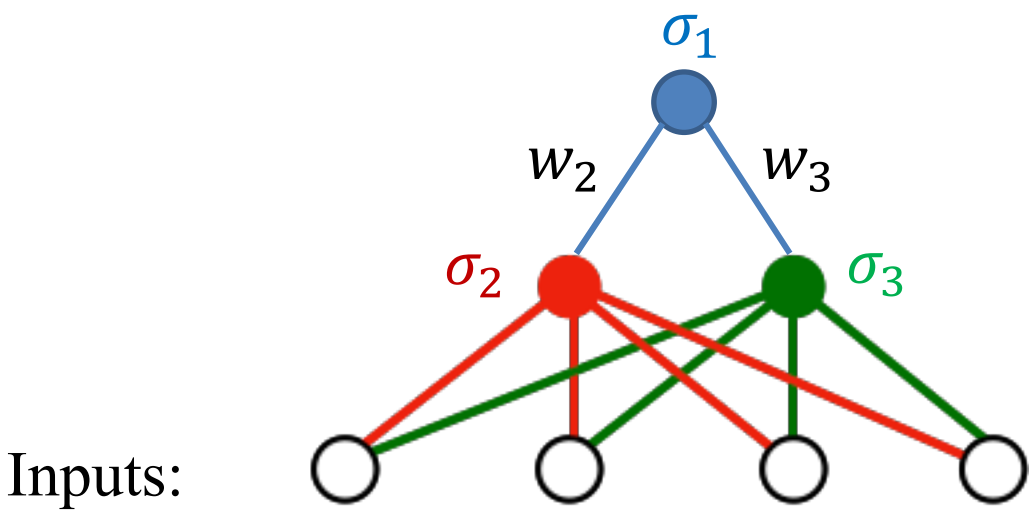
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

σ_2 and σ_3 will always get the same updates!

Weights initializations

Maybe start with all zeros?



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \sigma_1}\sigma_1(1-\sigma_1)\sigma_3$$

σ_2 and σ_3 will always get the same updates!

- Need to break symmetry!
- Maybe start with small random numbers then?
- But how small? $0.03 \cdot \mathcal{N}(0,1)$?

Weights initializations

- Linear models work best when inputs are normalized.
- Neuron is a linear combination of inputs + activation.
- Neuron output will be used by consecutive layers.

Weights initializations

Let’s look at the neuron output **before activation**: $\sum_{i=1}^{n} x_i w_i$.

If $E(x_i) = E(w_i) = 0$ and we generate weights independently from inputs, then $E\left(\sum_{i=1}^{n} x_i w_i\right) = 0$.

But variance can grow with consecutive layers.

Empirically this hurts convergence for deep networks!

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$

i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i) Var(w_i) \end{pmatrix} =$$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i) Var(w_i) \end{pmatrix} =$ w_i and x_i have 0 mean

$$= \sum_{i=1}^{n} Var(x_i) Var(w_i) = Var(x)[\boldsymbol{n\, Var(w)}]$$

Weights initializations

Let's look at the variance of $\sum_{i=1}^{n} x_i w_i$:

$Var(\sum_{i=1}^{n} x_i w_i) =$ i.i.d. w_i and mostly uncorrelated x_i

$= \sum_{i=1}^{n} Var(x_i w_i) =$ independent factors w_i and x_i

$$= \sum_{i=1}^{n} \begin{pmatrix} [E(x_i)]^2 Var(w_i) \\ +[E(w_i)]^2 Var(x_i) \\ +Var(x_i) Var(w_i) \end{pmatrix} =$$

w_i and x_i have 0 mean

$$= \sum_{i=1}^{n} Var(x_i) Var(w_i) = Var(x)[\boldsymbol{n\, Var(w)}]$$

We want this to be 1

Weights initializations

- Let's use the fact that $Var(aw) = a^2 Var(w)$.

- For $[\boldsymbol{n}\,\boldsymbol{Var}(\boldsymbol{aw})]$ to be 1
we need to multiply $\mathcal{N}(0,1)$ weights ($Var(w) = 1$)
by $a = 1/\sqrt{n}$.

- Xavier initialization (Glorot et al.)
multiplies weights by $\sqrt{2}/\sqrt{n_{in} + n_{out}}$.

- Initialization for ReLU neurons (He et al.)
uses multiplication by $\sqrt{2}/\sqrt{n_{in}}$.

Batch normalization

- We know how to initialize our network to constrain variance.
- But what if it grows during backpropagation?
- Batch normalization controls mean and variance of outputs **before activations**.

Batch normalization

- Let's normalize h_i − neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \boldsymbol{\beta_i}$$

0 mean, unit variance

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \boldsymbol{\beta_i}$$

0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \boldsymbol{\beta_i}$$

0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!
- During testing we will use an exponential moving average over train batches:

$0 < \alpha < 1$

$$\mu_i = \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 = \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2$$

Batch normalization

- Let's normalize h_i – neuron output before activation:

$$h_i = \boldsymbol{\gamma_i} \boxed{\frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}} + \boldsymbol{\beta_i}$$

0 mean, unit variance

- Where do μ_i and σ_i^2 come from? We can estimate them having a **current training batch**!
- During testing we will use an exponential moving average over train batches:

$0 < \alpha < 1$

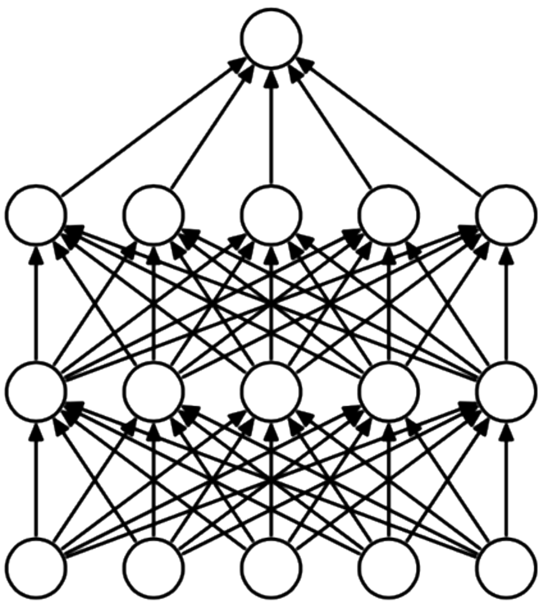
$$\mu_i = \alpha \cdot \mathbf{mean_{batch}} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 = \alpha \cdot \mathbf{variance_{batch}} + (1 - \alpha) \cdot \sigma_i^2$$

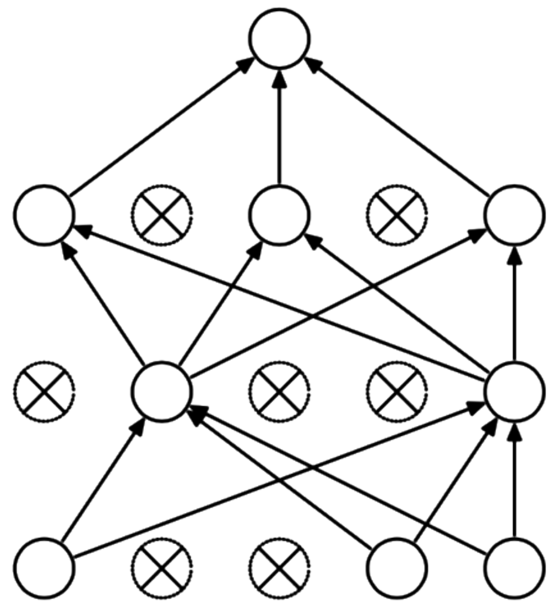
- What about γ_i and β_i? Normalization is a differentiable operation and we can apply backpropagation!

Dropout

- Regularization technique to reduce overfitting.
- We keep neurons active (non-zero) with probability p.
- This way we sample the network during training and change only a subset of its parameters on every iteration.



(a) Standard Neural Net



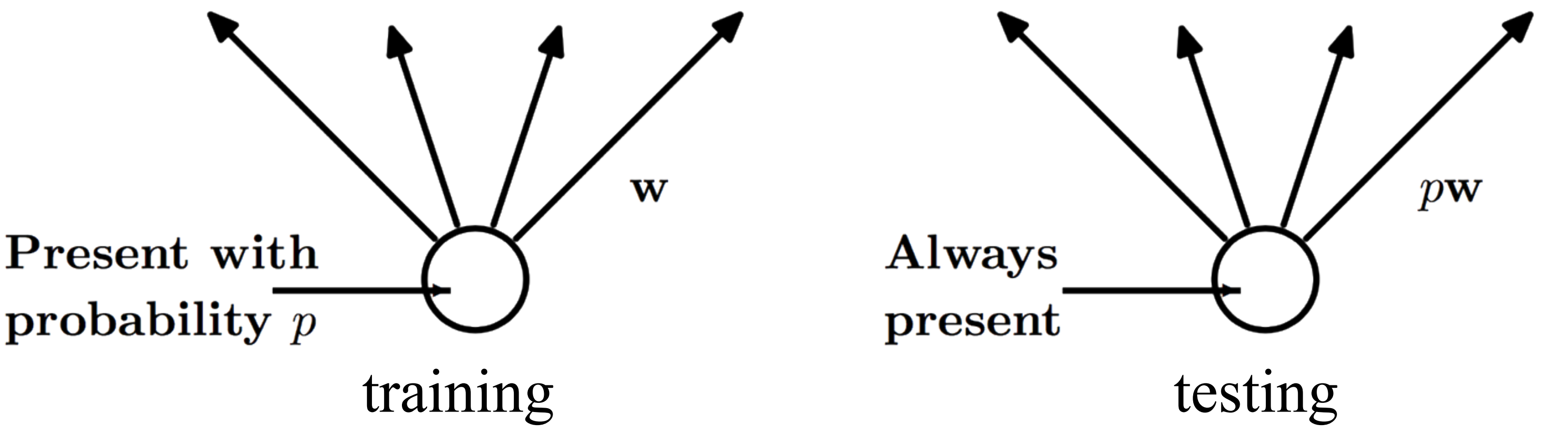
(b) After applying dropout.

Nitish Srivastava, http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Dropout

- During testing all neurons are present but their outputs are multiplied by p to maintain the scale of inputs:

$$\text{Expected input weight: } p \cdot w + (1 - p) \cdot 0$$



Nitish Srivastava, http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

- The authors of dropout say it's similar to having an ensemble of exponentially large number of smaller networks.

Data augmentation

- Modern CNN’s have millions of parameters!
- But datasets are not that huge!
- We can generate new examples applying distortions: flips, rotations, color shifts, scaling, etc.

Francois Chollet, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Data augmentation

- Modern CNN's have millions of parameters!
- But datasets are not that huge!
- We can generate new examples applying distortions: flips, rotations, color shifts, scaling, etc.
- Remember: CNN's are invariant to translation

Francois Chollet, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Takeaways

- Use ReLU activation
- Use He et al. initialization
- Try to add BN or dropout
- Try to augment your training data
- In the next video you will learn how modern convolutional networks look like