**4.4.1 Regularizace ztrátové funkce**

V tomto odstavci zmíníme často používaný způsob, jak dále zdokonalit výkon sítě, zejména v souvislosti s efektem přetrénování. Přetrénování vlastně znamená, že síť se během učení nastavuje na přesný vzorec trénovací množiny, tedy doslova si síť přehnaně věrně zapamatuje a reprodukuje trénovací příklady. Velmi dobře tedy zafunguje právě na těchto příkladech, ale na jakékoliv odchylce od nich je její odhad velmi špatný. Mluvíme pak o *generalizační chybě* (chybě zevšeobecňování příkladů), což je vlastně jeden z ústředních problémů neuronových sítí. Experimentálně bylo zjištěno, že přetrénovaná síť obsahuje část vazebných koeficientů w, které jsou v absolutní hodnotě výrazně větší než ty ostatní. Intuitivně se dá říci, že právě to je ten způsob, jak si síť některé motivy pamatuje přehnaně více na úkor zevšeobecnění příkladů. Způsob, jak toto potlačit, je přičíst ke ztrátové funkci člen úměrně závisející na velikosti všech synapsí (vah). Následná optimalizace ztrátové funkce (její minimalizace) bude tedy usilovat o rovnoměrné zmenšení všech vazeb, a tím také k vyrovnávání jejich vzájemné velikosti. Tento postup se nazývá regularizace. Její nejčastější forma je tzv. *L2 regularizace*, což znamená přičtení součtu kvadrátů všech vazeb w do ztrátové funkce:

Kde je vhodná regularizační konstanta (obvykle 0 – 0,1) a součet probíhá přes všechny vazby v síti.

Ztrátová funkce s L2 regularizací bude tedy vypadat takto:

neboli:

**+**

Při použití metody gradientního sestupu se při následném derivování ztrátové funkce součet kvadrátů změní na prostý lineární součet a zjednodušeně řečeno se tedy v algoritmu při optimalizaci bude jednat pouze o přičtení váženého součtu všech váhových koeficientů vrstvy ***l***. Použití L2 regularizace není zásadní pro všechny typy sítí, avšak výkon modelu se často může výrazně zlepšit (jak jsem již zmínil, L2 značně snižuje šanci na přetrénování sítě).

**4.5 Zpětné šíření chyby**

*Algoritmus zpětného šíření chyby* (*backpropagation algorithm*) je velmi mocný nástroj pro optimalizaci váhových koeficientů mnoha typů neuronových sítí. V podstatě stojí v pozadí většiny současných optimalizačních algoritmů. Odvození tohoto postupu není až tak technicky náročné, nicméně je poněkud zdlouhavé a málo přehledné, protože pracuje doslova v „džungli indexů“. Naštěstí mají výsledné vztahy docela jasnou interpretaci, a dávají tak vhled do toho, co se v síti při výpočtu děje.

O co se tedy v tomto algoritmu jedná? Jde o to, že při metodě gradientního sestupu je třeba znát gradient, tedy soubor parciálních derivací ztrátové funkce podle všech vah a biasů v síti (viz podkapitola o GD – při hledání minima v krajině vah a biasů se pohybujeme v opačném směru gradientu podle těchto parametrů). V případě varianty jednovrstvého perceptronu – ADALINE – to bylo ještě jednoduché, gradient ztrátové funkce – tedy příslušné derivace – bylo jednoduché spočítat, protože ztrátová funkce byla přímo funkcí příslušných vah a jejich derivace byla tedy lineární (viz příslušný výpočet v případě funkce SSE). Nicméně v případě skutečných vícevrstvých neuronových sítí narážíme na tento výpočetní problém: ztrátová funkce je funkcí aktivací výstupních neuronů, jejichž aktivace je funkcí aktivací neuronů předešlé vrstvy, jejich aktivace je funkcí aktivací neuronů předešlé vrstvy… atd. atd., přičemž máme najít derivace ztrátové funkce podle vah v každé libovolné vrstvě. Matematicky se jedná o problém derivování mnohanásobně složené funkce, což vede na součiny postupně se hromadících aktivací v pořadí od poslední – výstupní vrstvy. Přeskočíme tedy dlouhý odvozovací řetězec plný stromovitých indexovaných struktur a podíváme se na výsledek v případě naší modelové dvouvrstvé sítě.

Nazvěme výstupní chybou sítě prostý rozdíl výstupu sítě ***a(out)*** a správného cílového výsledku ***y*** daného příkladu, přičemž bereme v úvahu vektorový charakter celé výstupní vrstvy:

***δ(out) = a(out) – y***

Pak je chyba skryté vrstvy dána výrazem:

***δ(h) = δ(out)(W(out))T ʘ***kde ***δ(h)*** je opět vektor chyb celé skryté vrstvy a symbol ***ʘ*** označuje tzv. Hadamardův součin (*Hadamard product*), který není nic jiného, než součin obou vektorů po jednotlivých složkách (tedy výsledek je nový vektor, jehož i-tá složka je součinem i-tých složek jednotlivých vektorů v součinu). Faktor derivací v tomto součinu závisí na tvaru aktivační funkce a může být tímto tvarem zjednodušen, což je právě případ sigmoidy, nicméně pro další úvahu není tento faktor důležitý. Důležité je, že vektor chyb skryté vrstvy je získán maticovým násobením vektoru výstupní chyby převrácenou maticí vah (transpozice). Můžeme si tento součin ručně rozepsat, ale názorněji vidíme přímo na obrázku níže, že chybu skryté vrstvy získáme, jako bychom chybu výstupní vrstvy tlačili naopak (proti proudu dopředného výpočtu) i s násobením příslušnými váhovými koeficienty (tato symetrie je dána právě převrácením – transpozicí matice vah). Takto se výstupní chyby „protlačí“ přes všechny váhové vektory ***W*** zpět k neuronu. Jeho chybu počítáme všemi možnými cestami, které jsou v síti možné (stromovitá struktura synapsí), a nakonec se všechny možné příspěvky od těchto různých cest sečtou. Právě proto se metoda jmenuje „algoritmus zpětného šíření chyby“. Zahlédnout tento postup ve výše zmíněné formuli vyžaduje trochu práce a cviku, proto volím tento jazykový a obrázkový opis. Získané chyby pro každý neuron v síti potom tvoří základ pro dopočet skutečných gradientů, tedy parciálních derivací ztrátové funkce podle vah a biasů těchto vnitřních neuronů. Pro úplnost uvádím jejich formu:

***J(W) =***

***J(W) =***

kde ajsou aktivace j-tého neuronu skryté, respektive vstupní vrstvy. Tyto vztahy fungují přímo pro naši demonstrační dvouvrstvou síť, nicméně v případě vícevrstvých sítí to je naprosto obdobné. Vztahy mezi vstupní, skrytou a výstupní vrstvou jsou pak obecně vztahy mezi jakoukoliv vrstvou předcházející dané vrstvě, danou vrstvou a vrstvou následující.

Tako podkapitola była trochu náročnější na představivost, nicméně to asi dobře demonstruje, proč tento algoritmus „backpropagation”, ačkoliv je již skoro 40 let teoreticky známý, mohl dojít svého praktického uplatnění až s příchodem rychlých a velkokapacitních grafických karet a moderního optimalizačního software.



Obr. 4.9 Popsané schéma algoritmu zpětného šíření chyby